

# Create ABS Project in Twenty minutes!

*Jérôme BENOIS*

*Open-Source : <jerome.benois AT gmail.com>  
Corporate : <benois AT argia-engineering.fr>*





# Presentation Goals

- Installing ABS
- Develop your first application





# Agenda

Show ...

- How to install ABS?
- How to use UML model?
- How to generate code?
- How to code, test, configure and run application?
- Dealing with Web Services!





# How to Install ABS?





# Install command line

- Create directory : `~/abs`
- Download installer :
  - <http://prdownloads.sourceforge.net/abs/workshop-repository-manager-0.6.jar?download>
- Run :
  - `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`





# Install command line

- Register environment variable : `ABS_HOME=~/.abs`
- Add in your execution path : `ABS_HOME/tc/ant/bin`
- Configure ABS (choose default value with ENTER) :
  - `cd ABS_HOME`
  - `abs init`





# Install Eclipse Plugin

- Download eclipse 3.3 : <http://download.forge.objectweb.org/acceleo/eclipse-europa-linux-modeling-topcased-acceleo-2.1.1.zip>
- Use eclipse update manager :
  - "Help" -> "Software Updates" -> "Find and Install ..."
  - Add new remote site : <http://sharengo.org/update/europa>
  - *And choose all features.*



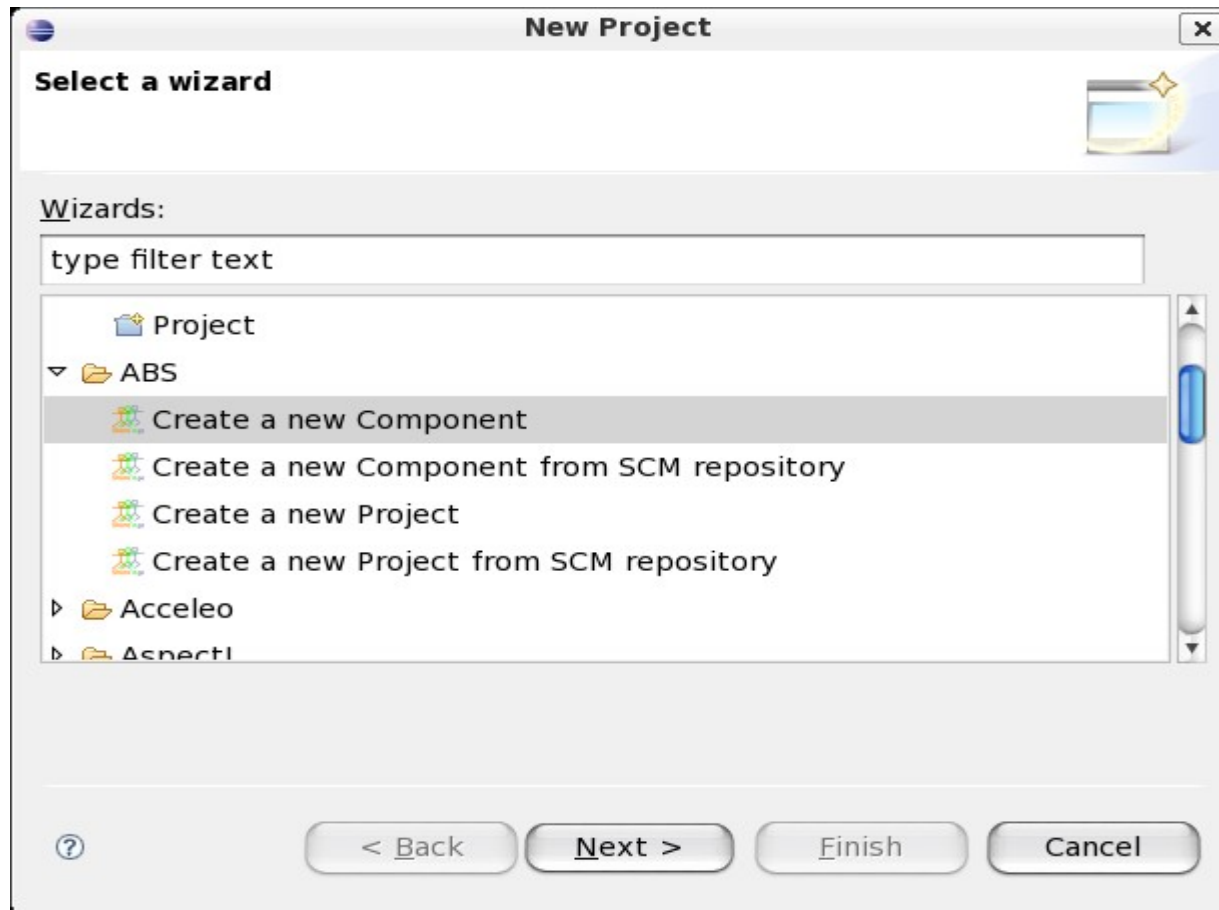


# How to use UML Model?



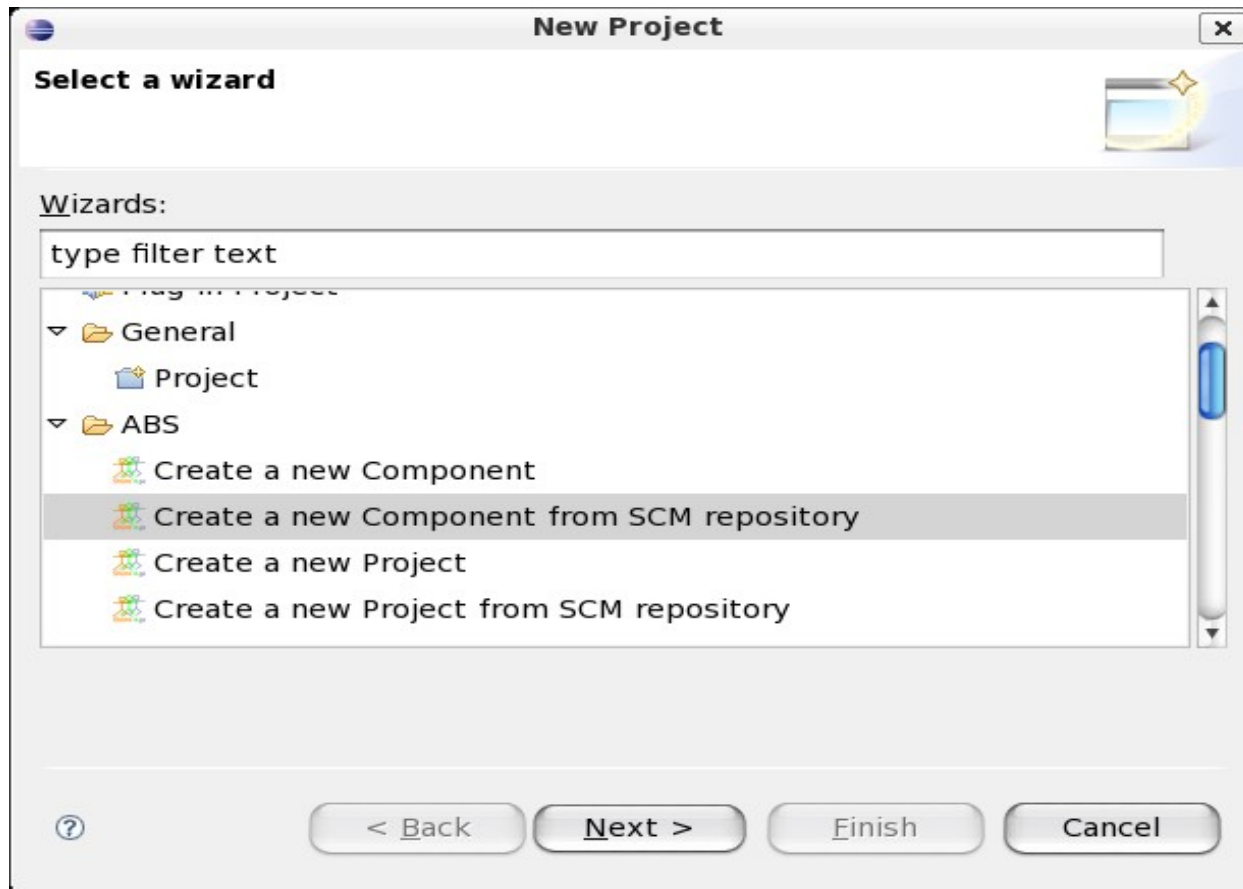


# Create "person" component





# Get technical component (I)





# Get technical component (1)

**New ABS Component from SCM repository**  
This wizard create a new ABS component from SCM repository.

**SCM Repository :**

- sharengo (http://svn.sharengo.org)
- argia (http://svn.argia.fr)
- bc (:pserver:benois@cvs.open-model.org:/bc)

**Name :**

middleware/gluon-core/trunk

**Tag (optional) :**

?

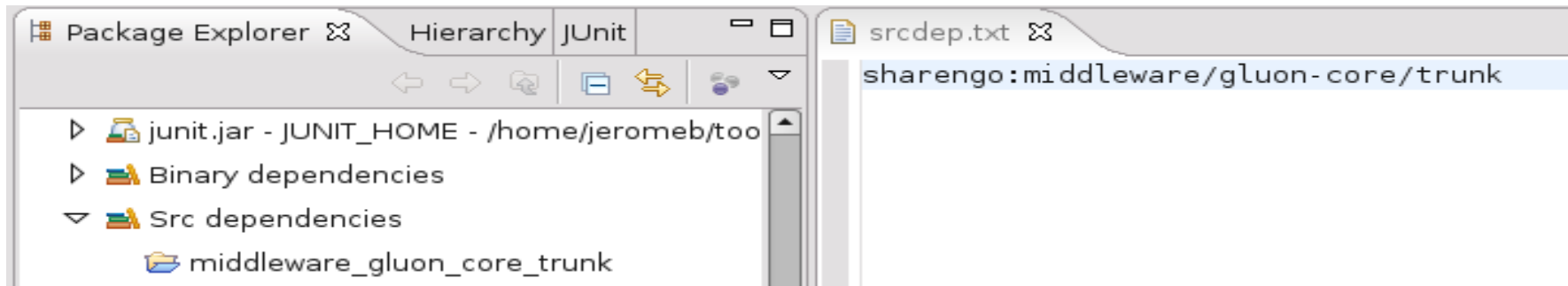
< Back   Next >   Finish   Cancel





# Add source dependency

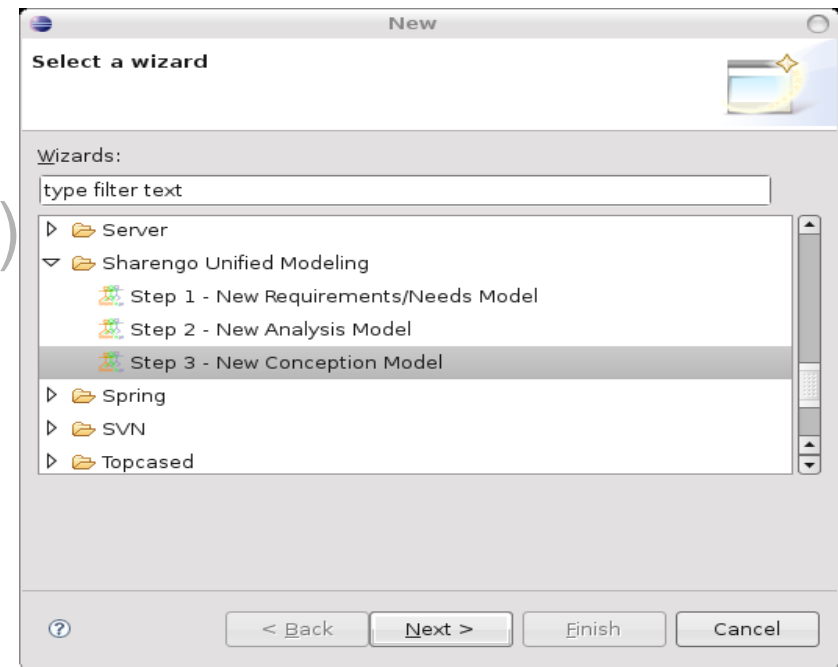
- Edit srcdep.txt file in “person” component and add one technical dependency :





# Create a Conception Model

- Create a new model in server/model/conception:
  - conception.uml (Model)
  - conception.uml.di (Diagrams)
  - conception.properties  
(used to configure generation)



# Define model name

- Define model name (used to define root package of all classes):

The screenshot shows the IDE interface for defining a UML Model. The 'Name' field is set to `org::sharengo::person` and the 'Visibility' is set to `public`. The 'Profiles' and 'Stereotypes' sections are also visible.

Property	Value
Name:	org::sharengo::person
Visibility:	public
Profiles	
Stereotypes	



# How to apply Stererotype?

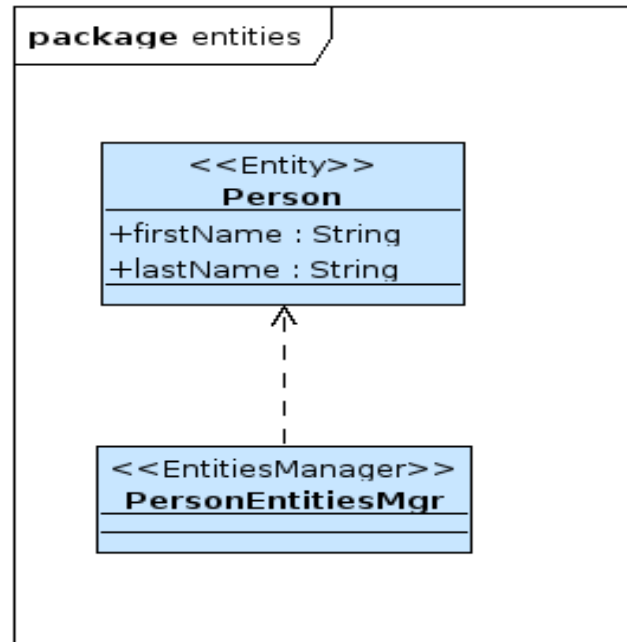
- You can apply stereotype in properties view:

The screenshot shows a UML modeling environment with a class diagram and the Properties view. The class diagram displays a package named 'entities' containing a class named 'Person' with the stereotype '<<Entity>>' and two properties: '+firstName : String' and '+lastName : String'. The Properties view at the bottom shows the 'Model' section with 'Stereotypes' expanded. The 'Available Stereotypes' list includes 'Sharengo.conception::Service', 'Sharengo.conception::Ui', 'Sharengo.conception::Process', 'Sharengo.conception::EntitiesManager', 'Sharengo.conception::View', and 'Sharengo.conception::Controller'. The 'Applied Stereotypes' list shows 'Sharengo.conception::Entity' has been applied to the selected class.



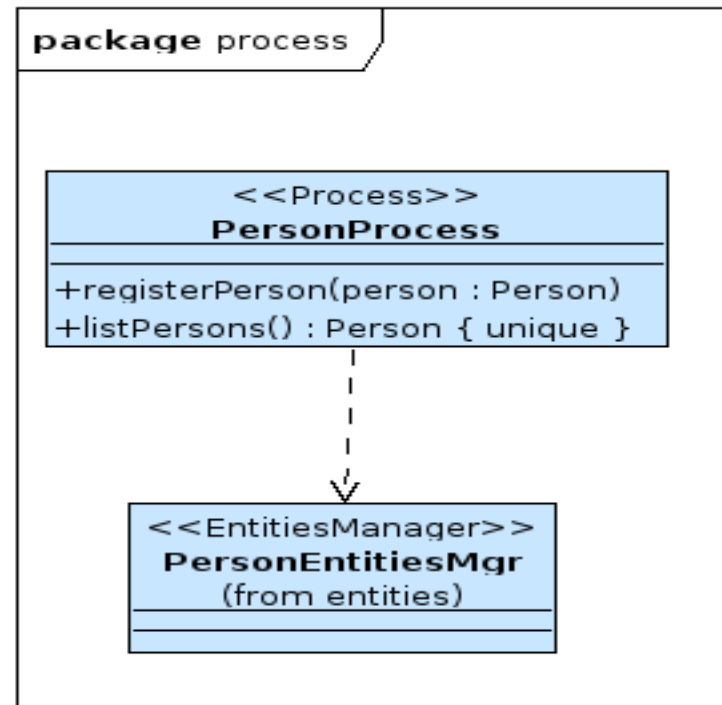


# Populate entities package

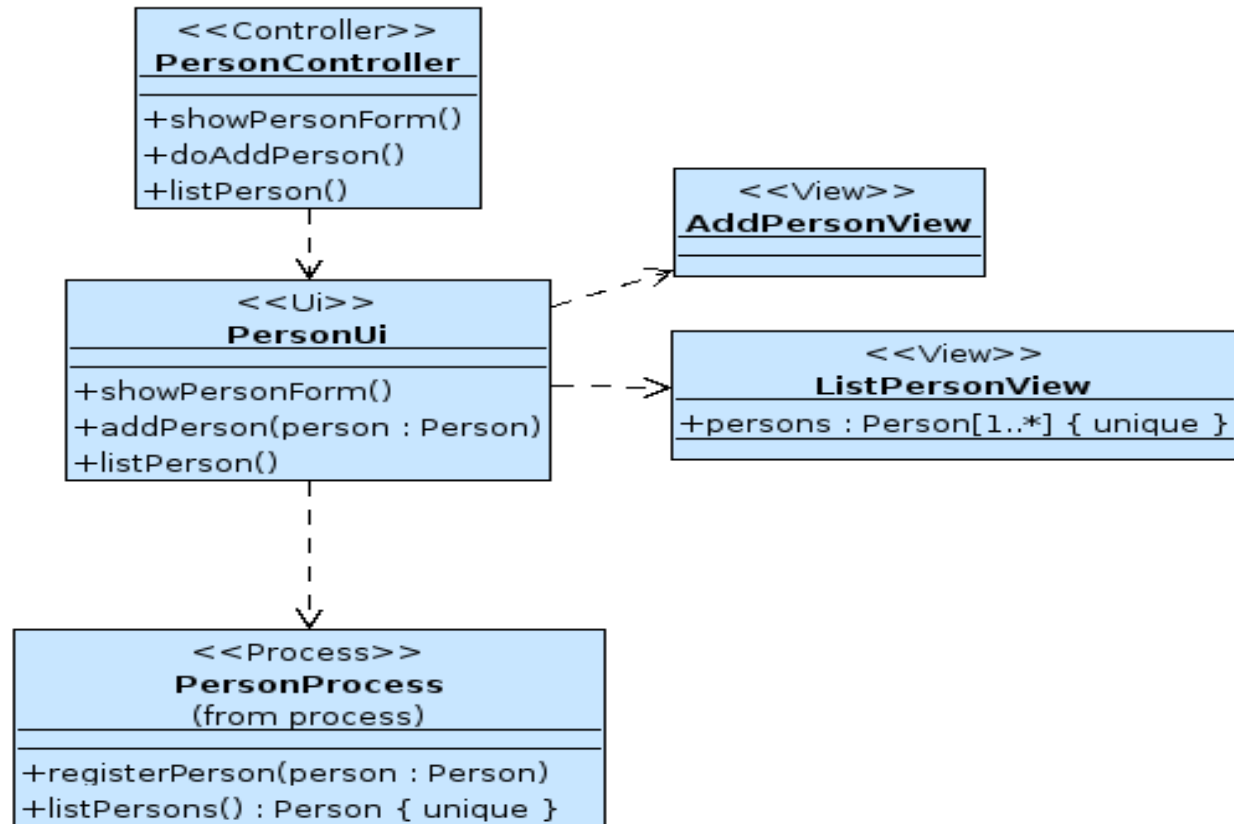




# Populate process package



# Populate ui package





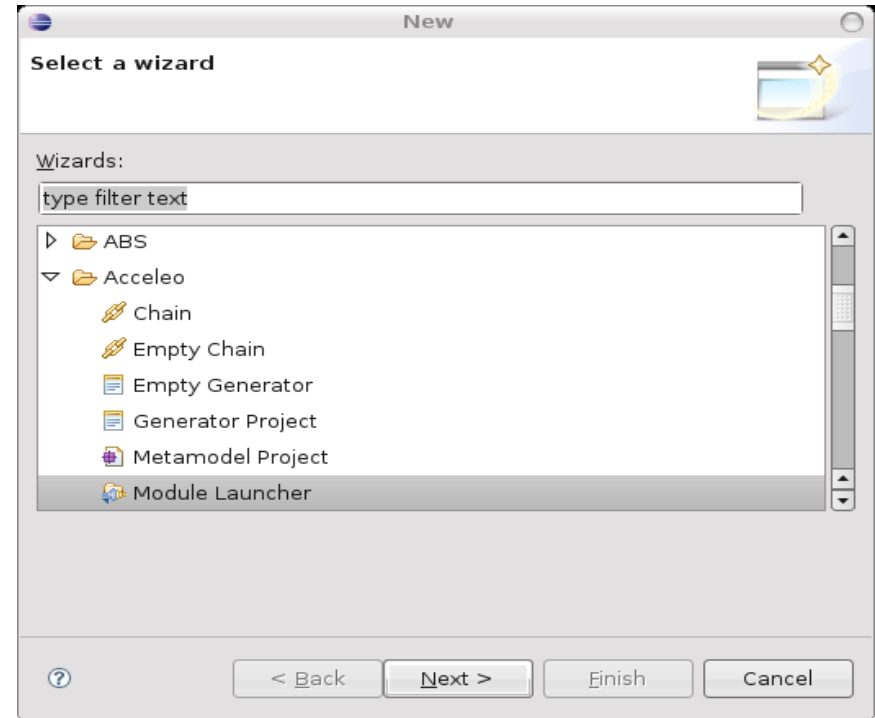
# How to generate code?





# Choose target Architecture (1)

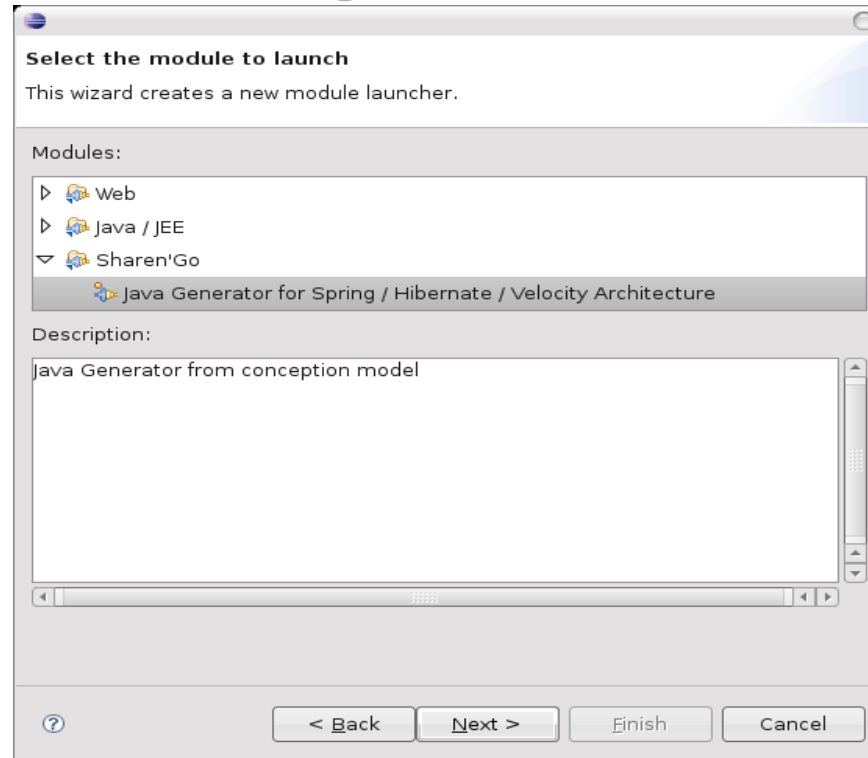
- Choose architecture / Platform Specific Model  
(launch wizard from server/model/conception)





# Choose target Architecture (II)

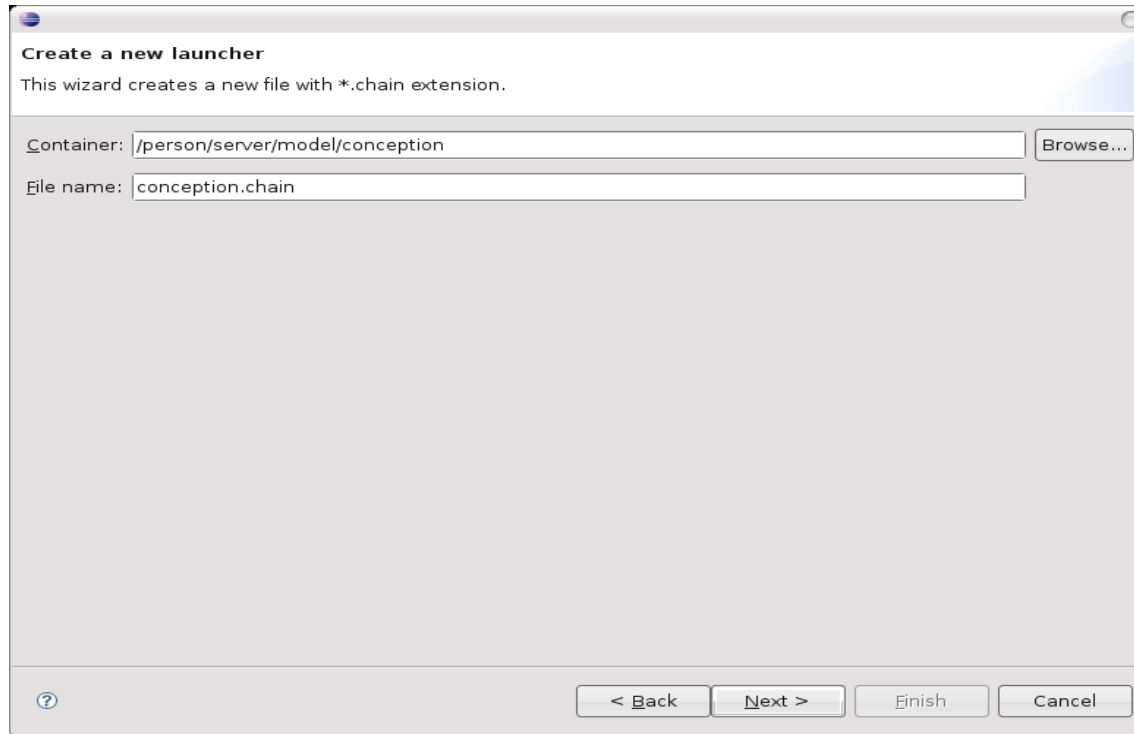
- In architecture catalog :





# Choose target Architecture (III)

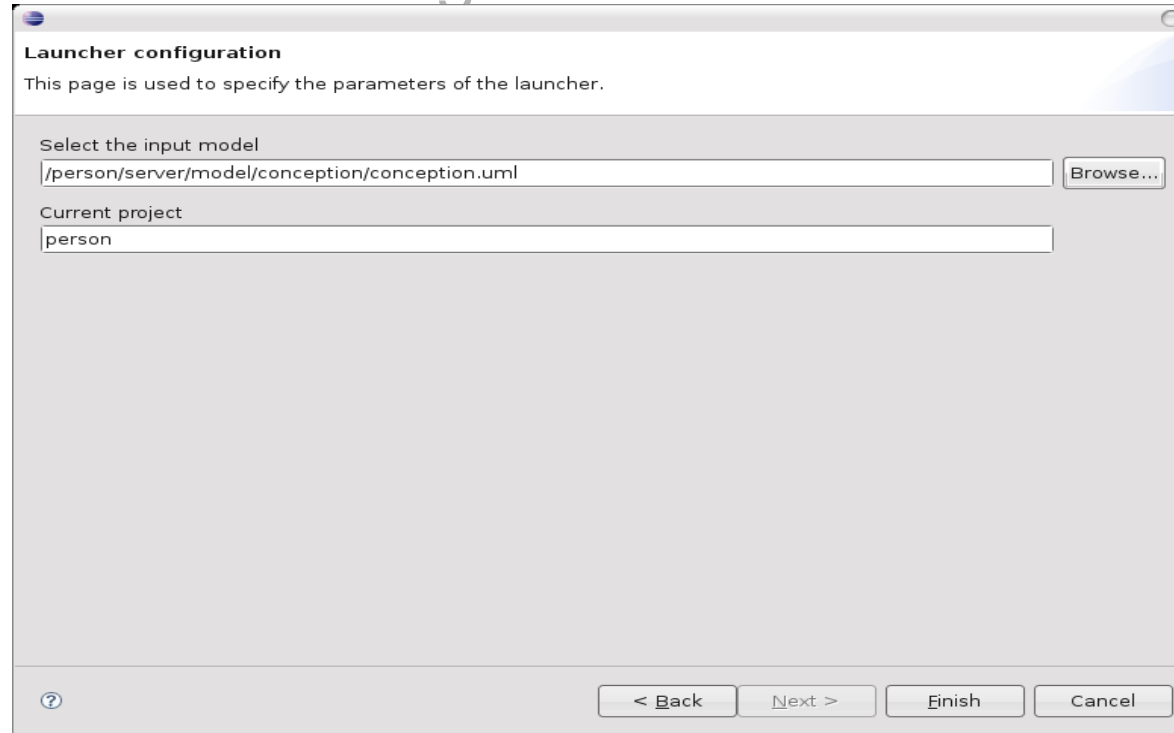
- In architecture catalog :





# Choose target Architecture (IV)

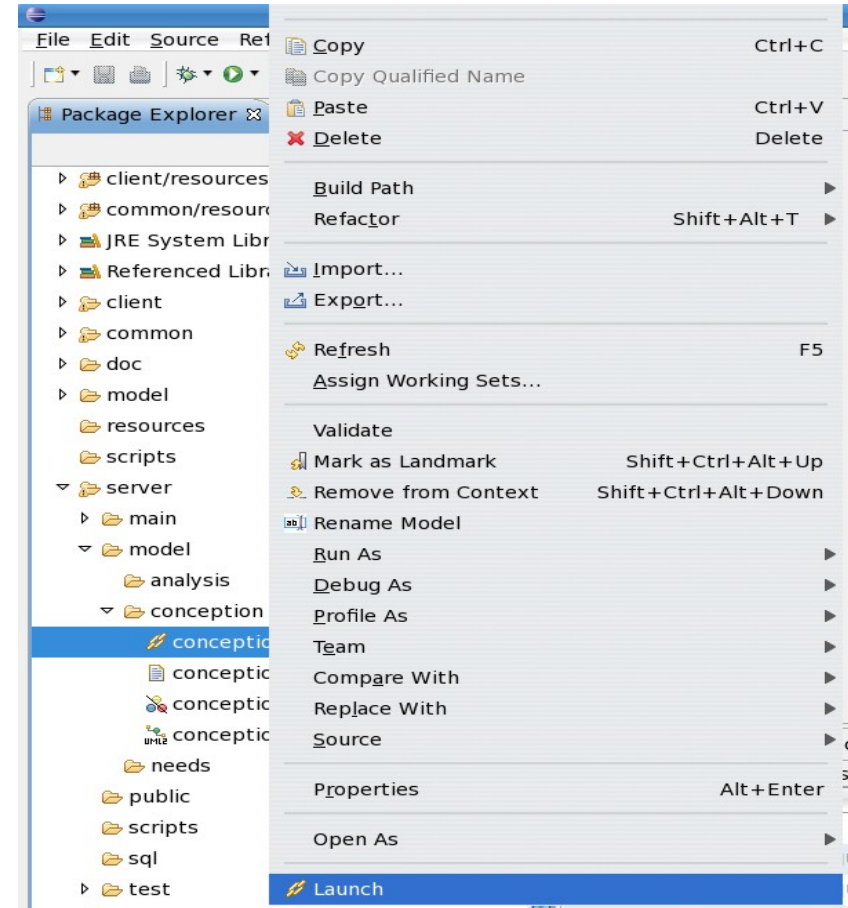
- In architecture catalog :





# Choose target Architecture (IV)

- Generate code :
  - use conception.launch :





How to code, test, configure  
and run application?





# Configure application

- Edit `srcdep.txt` file to add source dependency:  
`sharengo:middleware/gluon-core/trunk`





# Unit test on EntityManager

- Run generated unit tests to validate EntityManager :

The screenshot shows an IDE interface with two main panels. On the left, the 'JUnit' view displays the results of a test run: 'Finished after 1,628 seconds', 'Runs: 3/3', 'Errors: 0', and 'Failures: 0'. Below this, a tree view shows the package 'org.sharengo.person.entities.impl.PersonEntities' with three test methods: 'testCreatePersonEntitiesMgr', 'testSavePersonEntitiesMgr', and 'testDeletePersonEntitiesMgr'. On the right, the 'PersonEntitiesMgrTest.java' file is open, showing the following code:

```
/**
 * Test the creation of the entity PersonEntitiesMgr.<br/>
 * <ul><li>Step 1 : Create an entity</li>
 * <li>Step 2 : Search the entity and verify it exist</li></ul>
 */
public final void testCreatePersonEntitiesMgr() throws Exception {
    // fill attributes with exemple values
    Person person = getPerson();

    // Execute the tested code
    personEntitiesMgr.create(person);

    // Verify result
    boolean found = false;
    for(Person currentPerson : personEntitiesMgr.findAll()) {
        if (currentPerson.equals(person)) {
            found = true;
        }
    }
    assertTrue("Person not created", found);
}
```





# Coding process Unit test

- Coding Process unit test and run :

The screenshot displays an IDE interface with two main panes. The left pane shows the JUnit test runner results, and the right pane shows the source code of the test class.

**JUnit Test Runner Results:**

- Package Explorer: Hierarchy JUnit
- Finished after 2,539 seconds
- Runs: 2/2
- Errors: 0
- Failures: 2
- Test Class: org.sharengo.person.process.impl.PersonProcessTest [Runned]
- Test Methods: testRegisterPerson, testListPersons
- Failure Trace: junit.framework.AssertionFailedError

**Source Code (PersonProcessTest.java):**

```
public class PersonProcessTest extends AbstractBusinessLayerTests {

    /**
     * class dedicated logger
     */
    private static Log log = LogFactory.getLog(PersonProcessTest.class.getName());

    @Test
    public void testRegisterPerson() throws Exception {
        // Start of user code of testRegisterPerson()
        Person person = PersonHelper.createPerson();
        assertNull(person.getId());
        getPersonProcess().registerPerson(person);
        assertNotNull(person.getId());
        // End of user code of testRegisterPerson()
    }

    @Test
    public void testListPersons() throws Exception {
        // Start of user code of testListPersons()
        Collection<Person> persons = getPersonProcess().listPersons();
        assertNotNull(persons);
        int before = persons.size();
        getPersonProcess().registerPerson(PersonHelper.createPerson());
        getSession().flush();
        persons = getPersonProcess().listPersons();
        assertNotNull(persons);
        assertEquals(before+1, persons.size());
        // End of user code of testListPersons()
    }
}
```

# Coding Process

- Coding Process :

```

public class PersonProcessImpl implements IPersonProcess {
    /**
     * class dedicated logger
     */
    private static Log log = LoggerFactory.getLog(PersonProcessImpl.class
        .getName());

    private IPersonEntitiesMgr personEntitiesMgr = null;

    public void setPersonEntitiesMgr(IPersonEntitiesMgr personEntitiesMgr) {
        this.personEntitiesMgr = personEntitiesMgr;
    }

    public IPersonEntitiesMgr getPersonEntitiesMgr() {
        return this.personEntitiesMgr;
    }

    /**
     *
     * @param person
     */
    public void registerPerson(Person person) {
        // Start of user code of void registerPerson(Person person)
        personEntitiesMgr.create(person);
        // End of user code
    }

    /**
     *
     * @return
     */
    public Collection<Person> listPersons() {
        // Start of user code of Collection<Person> listPersons()
        Collection<Person> persons = personEntitiesMgr.findAll();
        return persons;
        // End of user code
    }
}

```



# Run process Unit test

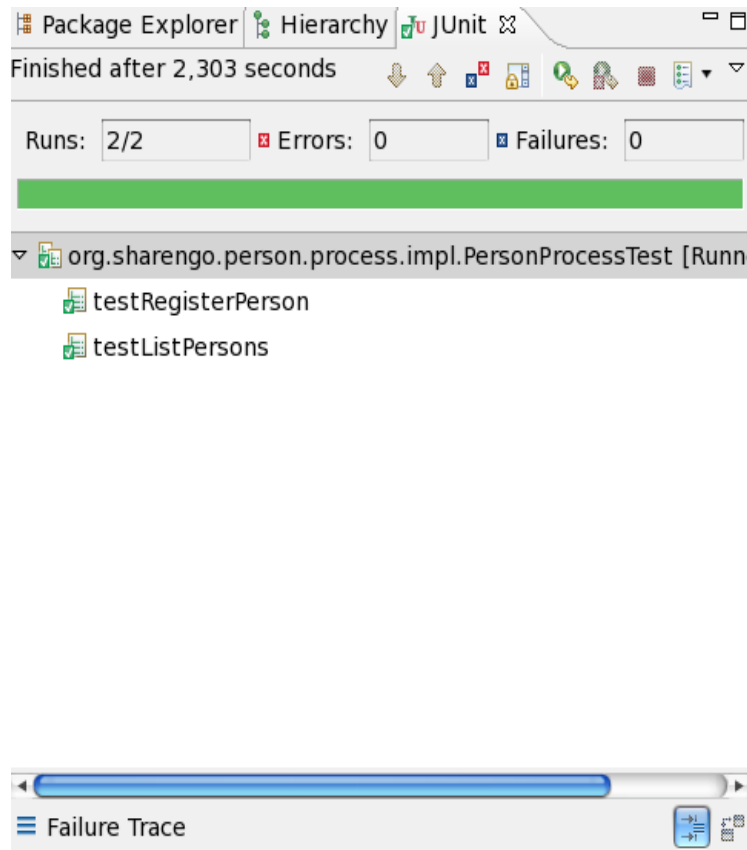
• Yeah !

Green Bar:

Business

Layer

is OK !



```
PersonProcessTest.java | PersonProcessImpl.java
public class PersonProcessTest extends AbstractBusinessLayerT

/**
 * class dedicated logger
 */
private static Log log = LogFactory.getLog(PersonProcessT

@Test
public void testRegisterPerson() throws Exception {
    // Start of user code of testRegisterPerson()
    Person person = PersonHelper.createPerson();
    assertNull(person.getId());
    getPersonProcess().registerPerson(person);
    assertNotNull(person.getId());
    // End of user code of testRegisterPerson()
}

@Test
public void testListPersons() throws Exception {
    // Start of user code of testListPersons()
    Collection<Person> persons = getPersonProcess().listP
    assertNotNull(persons);
    int before = persons.size();
    getPersonProcess().registerPerson(PersonHelper.create
    getSession().flush();
    persons = getPersonProcess().listPersons();
    assertNotNull(persons);
    assertEquals(before+1, persons.size());
    // End of user code of testListPersons()
}
```





# Coding PersonController

- Coding
- Web  
Controller:

```
PersonController.java
public void showPersonForm(HttpServletRequest request, HttpServletResponse response) throws TechnicalException {
    try{
        // Start of user code of void showPersonForm(HttpServletRequest request, HttpServletResponse response)
        response.setContentType("text/html; charset=UTF-8");
        OutputStreamWriter osw = new OutputStreamWriter(response.getOutputStream(), "UTF-8");
        personUi.showPersonForm(osw);
        osw.flush();
        // End of user code
    } catch (Exception e) {
        throw new TechnicalException(e);
    }
}
/**
 *
 */
public void doAddPerson(HttpServletRequest request, HttpServletResponse response) throws TechnicalException {
    try{
        // Start of user code of void doAddPerson(HttpServletRequest request, HttpServletResponse response)
        response.setContentType("text/html; charset=UTF-8");
        OutputStreamWriter osw = new OutputStreamWriter(response.getOutputStream(), "UTF-8");
        Person person = new Person();
        person.setLastName(request.getParameter("lastName"));
        person.setFirstName(request.getParameter("firstName"));
        personUi.addPerson(osw, person);
        osw.flush();
        // End of user code
    } catch (Exception e) {
        throw new TechnicalException(e);
    }
}
/**
 *
 */
public void listPerson(HttpServletRequest request, HttpServletResponse response) throws TechnicalException {
    try{
        // Start of user code of void listPerson(HttpServletRequest request, HttpServletResponse response)
        response.setContentType("text/html; charset=UTF-8");
        OutputStreamWriter osw = new OutputStreamWriter(response.getOutputStream(), "UTF-8");
        personUi.listPerson(osw);
        osw.flush();
        // End of user code
    } catch (Exception e) {
        throw new TechnicalException(e);
    }
}
```





# Coding PersonUImpl

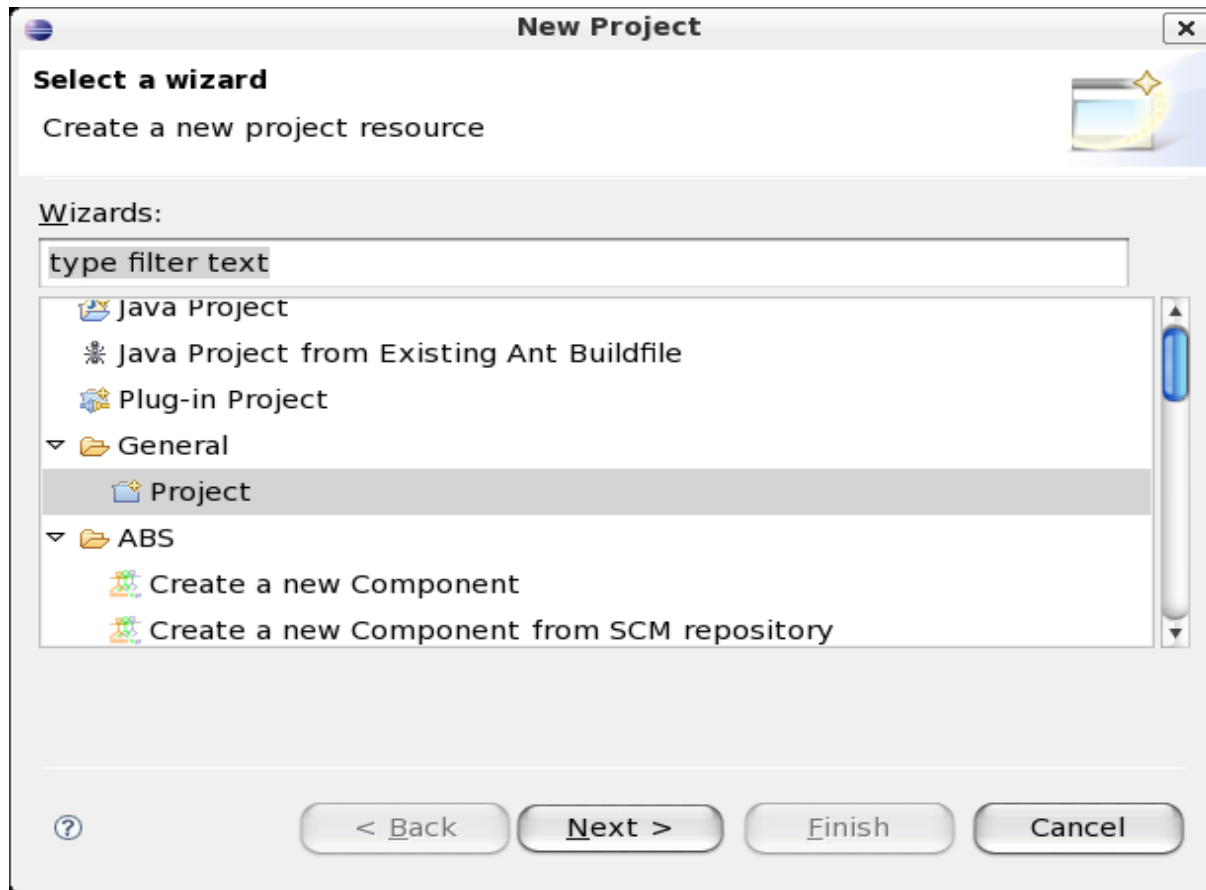
- Coding Ui :

```
PersonUImpl.java ✖  
  
public void showPersonForm(Writer writer) throws TechnicalException{  
    // Start of user code of void showPersonForm(Writer writer)  
    AddPersonView view = getAddPersonView();  
    view.show(writer);  
    // End of user code  
}  
  
public void addPerson(Writer writer, Person person) throws TechnicalException{  
    // Start of user code of void addPerson(Writer writer, Person person)  
    personProcess.registerPerson(person);  
    listPerson(writer);  
    // End of user code  
}  
  
public void listPerson(Writer writer) throws TechnicalException{  
    // Start of user code of void listPerson(Writer writer)  
    ListPersonView view = getListPersonView();  
    Collection<Person> persons = personProcess.listPersons();  
    view.setPersons(persons);  
    view.show(writer);  
    // End of user code  
}
```



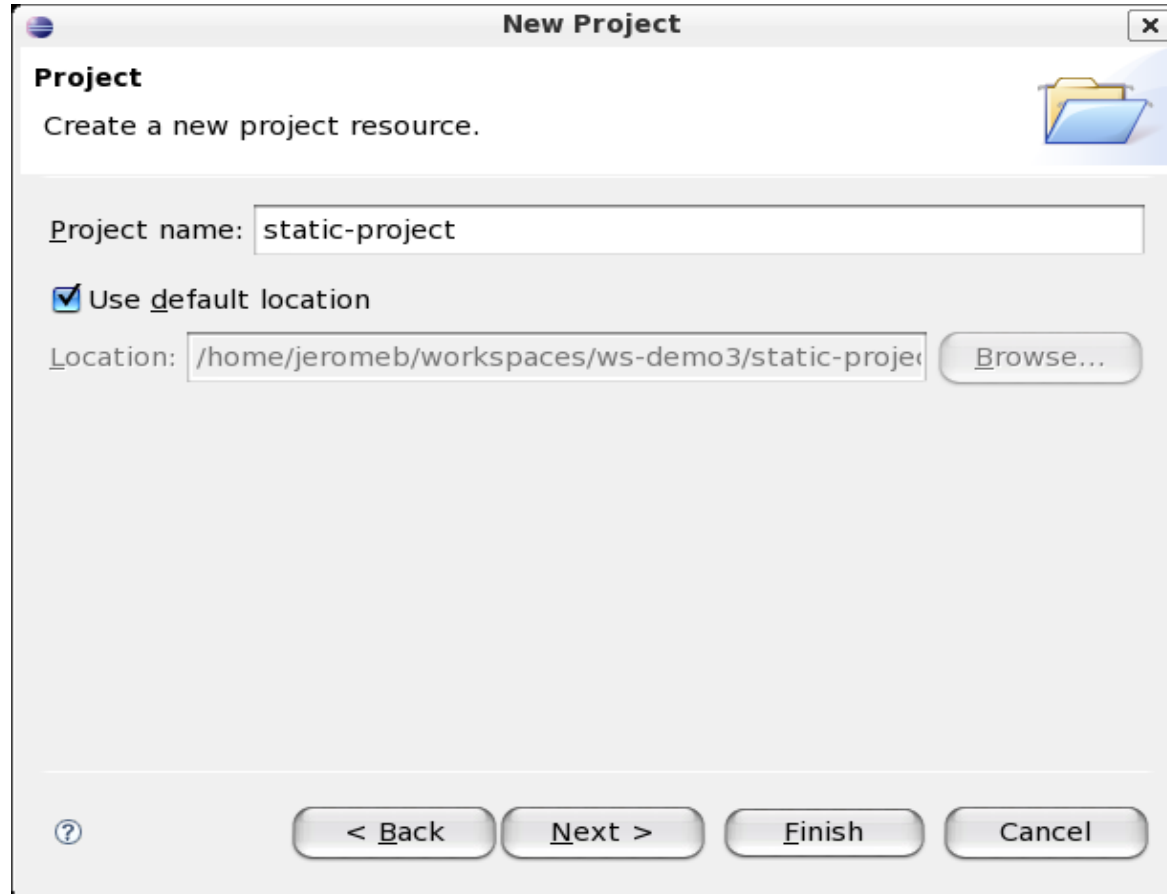


# Create static project (I)





# Create static project (II)





# Coding Velocity View (I)

- Coding AddPersonView.vsl in static-project:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<title>AddPersonView</title>
<meta http-equiv="Content-Type" content="application/xhtml+xml; charset=UTF-8" />
</head>
<body>
<h1>Add Person View</h1>
<br/>
<form method="post" action="actions/personController/doAddPerson">
Fisrt Name : <input type="text" name="firstName"/>
Last Name : <input type="text" name="lastName"/>
<input type="submit"/>
</form>
<br/>
<br/>
<a href="actions/personController/listPerson">List all persons</a>
</body>
</html>
```





# Coding Velocity View (II)

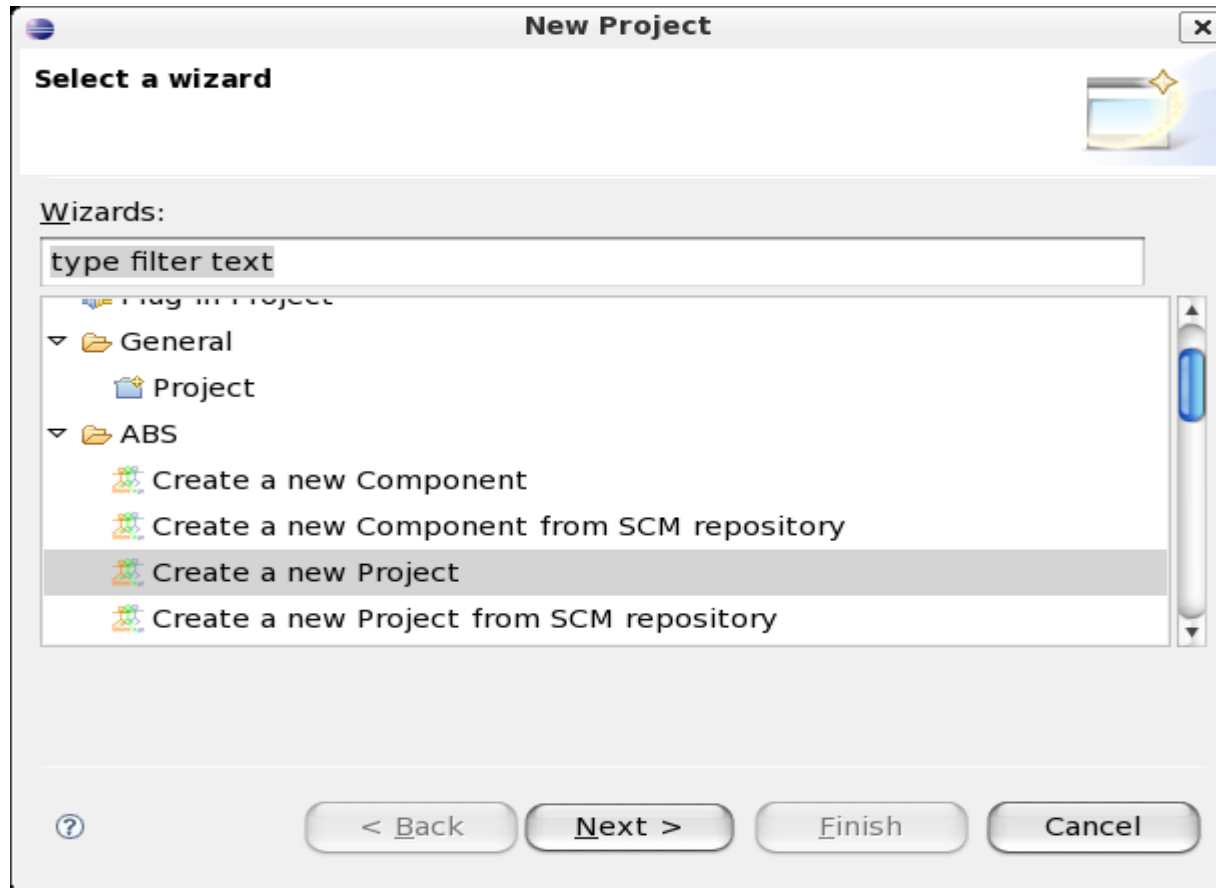
- Coding ListPersonView.vsl:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<title>ListPersonView</title>
<meta http-equiv="Content-Type" content="application/xhtml+xml; charset=UTF-8" />
</head>
<body>
<h1>List Person View</h1>
<br/>
<ul>
#foreach( $person in $persons )
<li>$person.firstName $person.lastName</li>
#end
</ul>
</body>
</html>
```





# Create project (I)




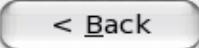
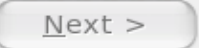
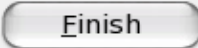
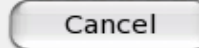


# Create project (II)

**New ABS Project**

This wizard create a new ABS project.

Name :

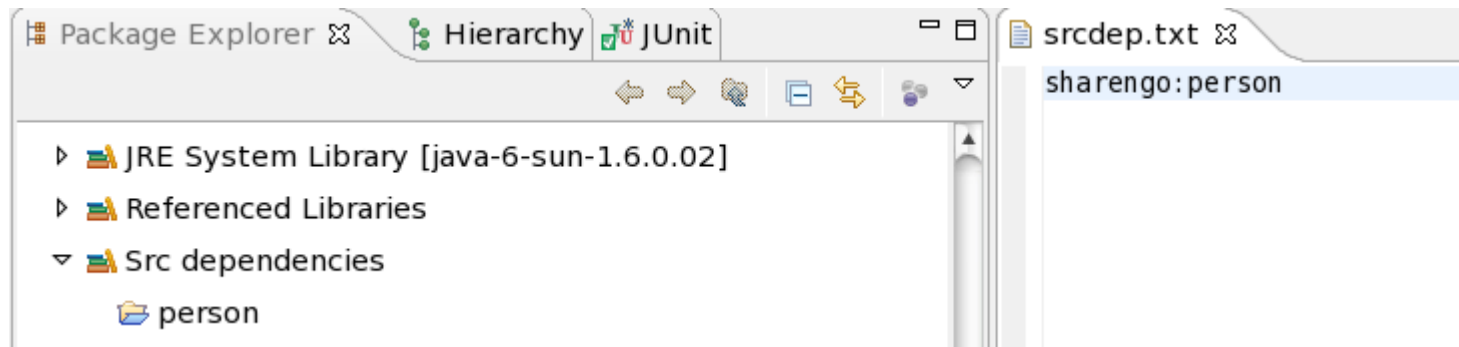
    

The image shows a screenshot of a software wizard window titled "New ABS Project". The window has a standard title bar with a close button (X) in the top right corner. Below the title bar, the text "New ABS Project" is displayed in bold. Underneath, a descriptive sentence reads "This wizard create a new ABS project." The main area of the window contains a label "Name :" followed by a text input field containing the text "person-management". At the bottom of the window, there is a horizontal bar containing four buttons: a help icon (question mark), a "< Back" button, a "Next >" button, a "Finish" button, and a "Cancel" button.



# Add dependency to Person

- Edit srcdep.txt in your module' s project in order to add dependency:





# Configure Velocity

- Edit server/main/conf/velocity.properties in your module' s project:

```
#
# specify three resource loaders to use
#
#resource.loader = file, class, jar
resource.loader = file, class

#
# for the loader we call 'file', set the FileResourceLoader as the
# class to use, turn off caching, and use 3 directories for templates
#
file.resource.loader.description = Velocity File Resource Loader
file.resource.loader.class = org.apache.velocity.runtime.resource.loader.FileResourceLoader
file.resource.loader.path = /home/jeromeb/workspaces/ws-demo3/static-project
file.resource.loader.cache = false
file.resource.loader.modificationCheckInterval = 0

#
# for the loader we call 'class', use the ClasspathResourceLoader
#
class.resource.loader.description = Velocity Classpath Resource Loader
class.resource.loader.class =
org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader
```





# Configure DataBase Access (I)

- Edit server/main/conf/context.xml in your module' s project:

```
<Context
  path="/person-management"
  debug="1"
  reloadable="true">

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





# Configure DataBase Access (II)

- Create project database:

```
createdb -U postgres -E unicode person-management
```

- Edit `person-management/deployment/localhost/replace.server.properties`:  
`@@hibernate.datasource@@=person-management`





# Configure Application

Edit server/main/conf/applicationContext.xml in your module's project:

```
<?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/schema/beans/spring-beans-2.0.xsd
       http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-2.0.xsd">

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

  <bean id="velocityEngine" class="org.springframework.ui.velocity.VelocityEngineFactoryBean">
    <property name="configLocation" value="WEB-INF/classes/velocity.properties"/>
  </bean>

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

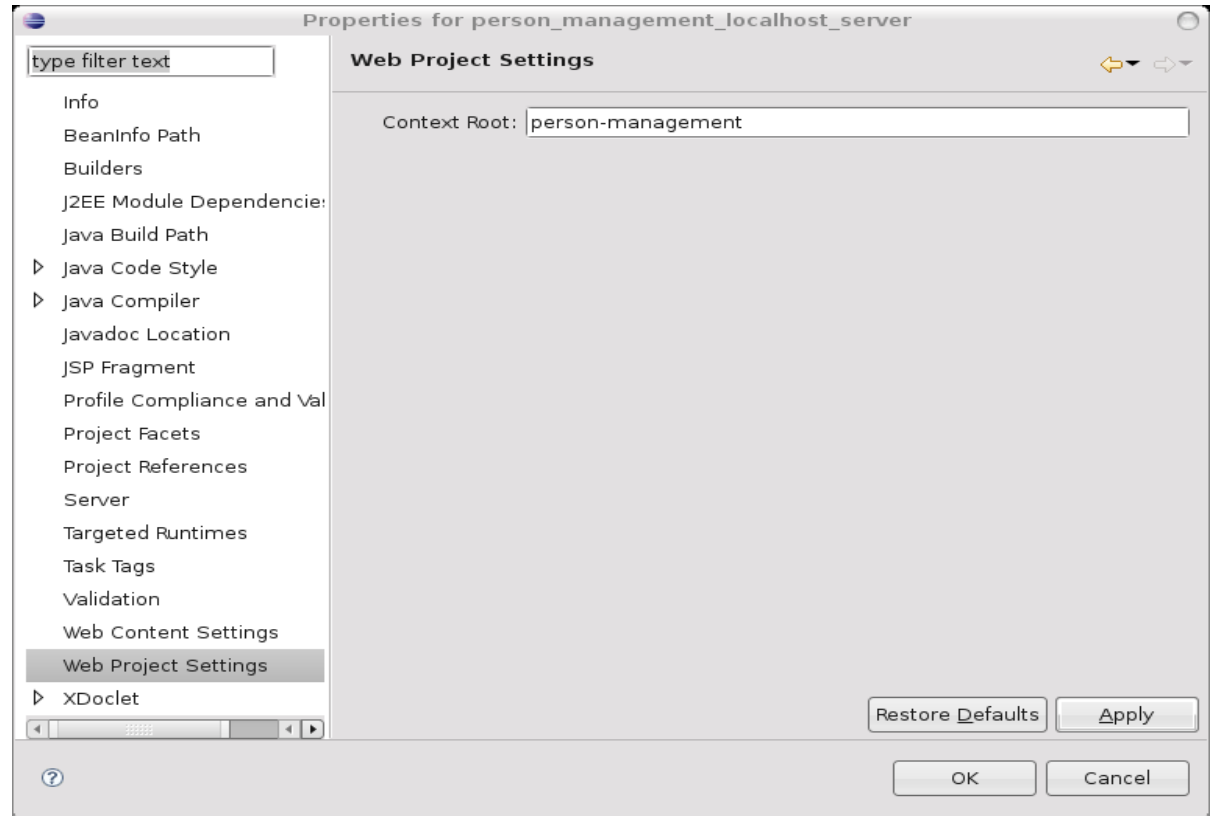
  <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingResources">
      <!-- Allow hibernate mapping file to use -->
      <list>
        <value>org/sharengo/person/entities/hibernate/Person.hbm.xml</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>
```





# Configure Tomcat

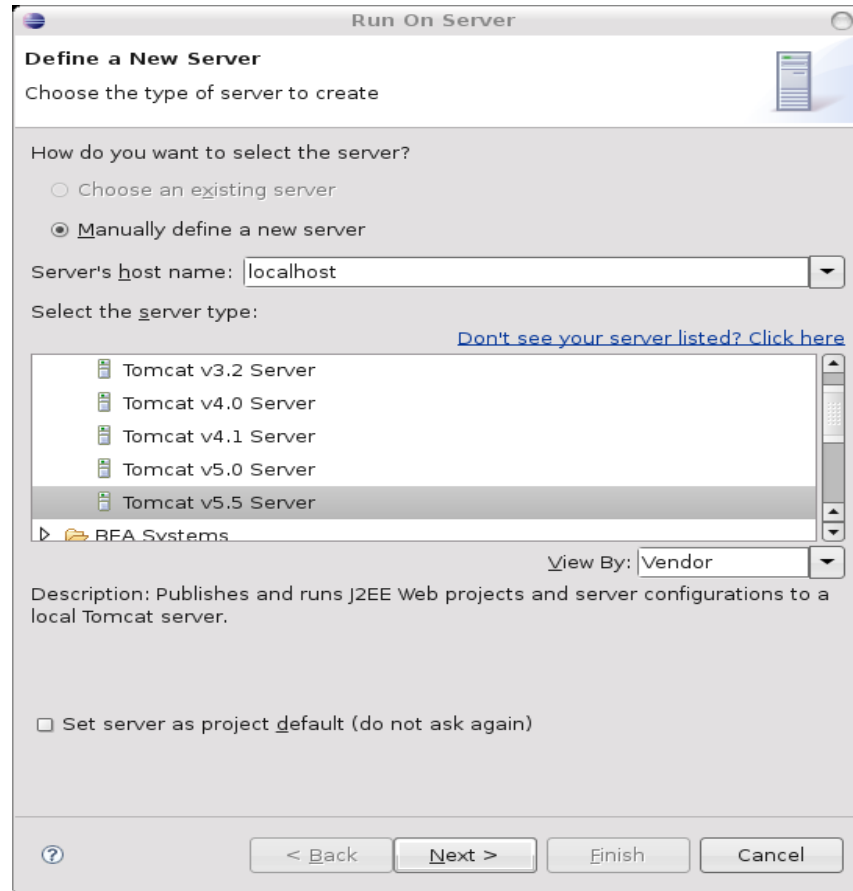
- Configure application context root for Tomcat:





# Run Tomcat (I)

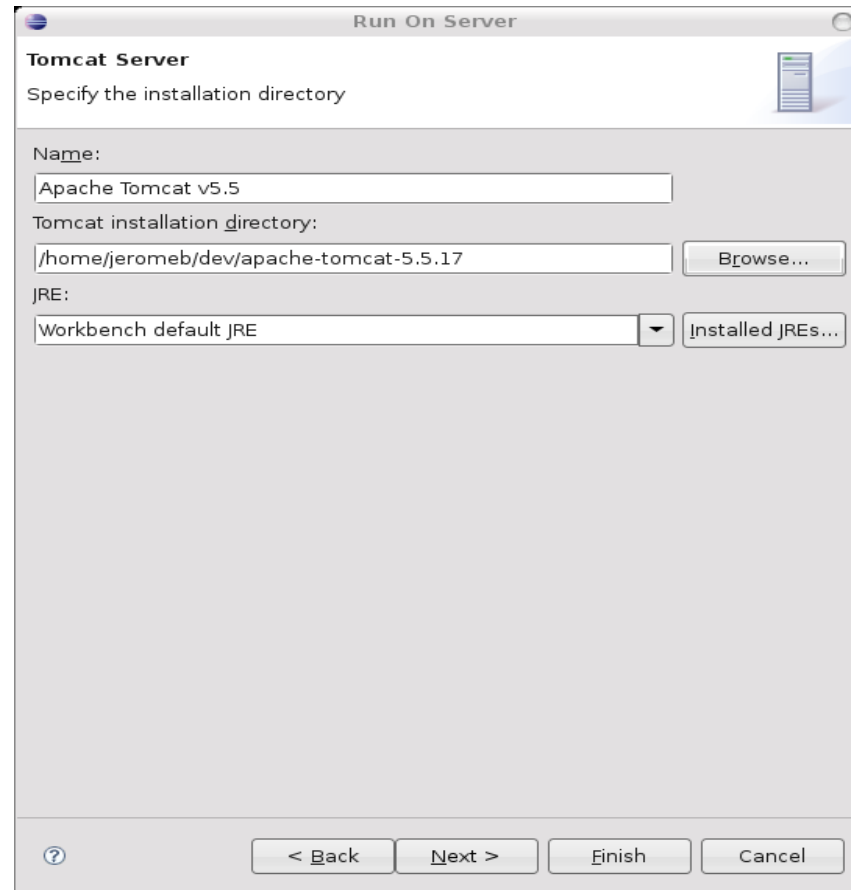
- Run Application on tomcat server:





# Run Tomcat (II)

- Run Application on tomcat server:

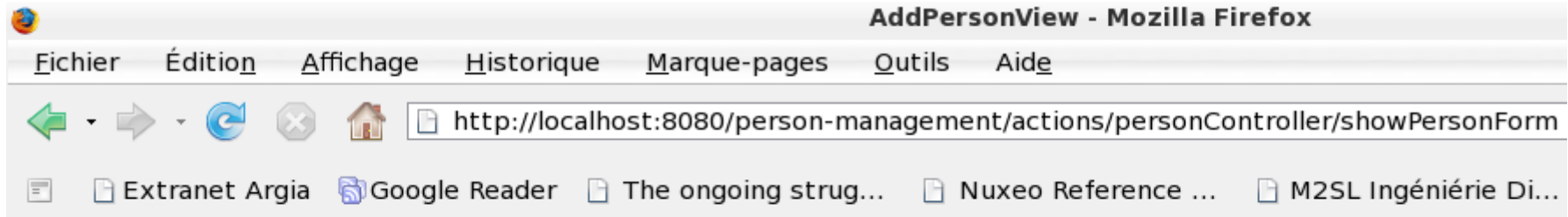




# Use application (1)

- You can use your application here:

<http://localhost:8080/person-management/actions/personController/showPersonForm>



## Add Person View

Fisrt Name :  Last Name :

[List all persons](#)

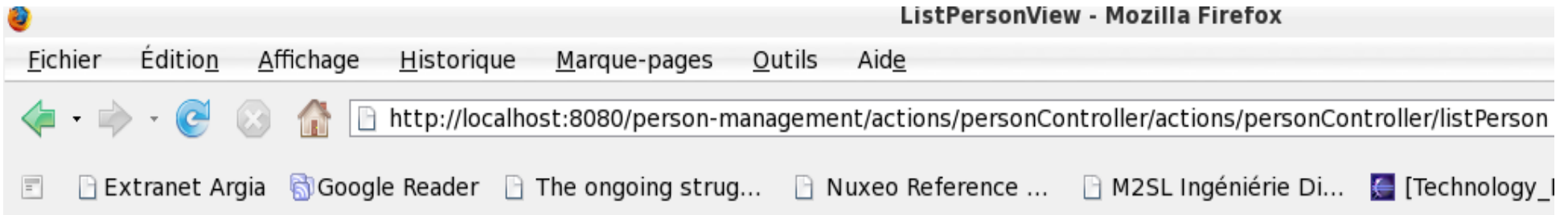




# Use application (1)

- You can use your application here:

<http://localhost:8080/person-management/actions/personController/actions/personController/listPerson>



## List Person View

- scott tiger





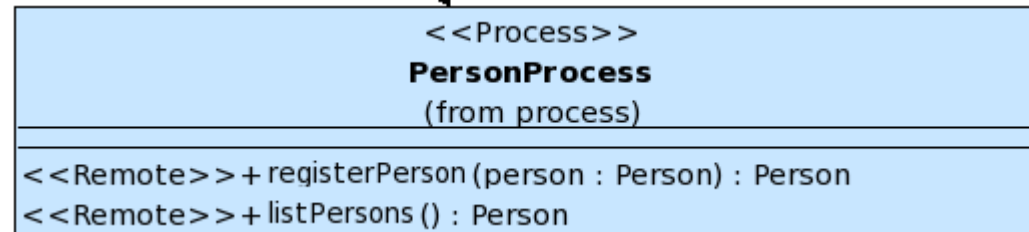
# Dealing with Web Services!





# Shared Business operation

- Use stereotype <<Remote>> on your <<process>> operations:



- Get component from repository and add source dependency (srcdep.txt on your component):

sharengo:middleware/gluon-xfire/trunk

- Add binary dependency: -spring-mock.jar





# Configure services startup

- Import Xfire config in `server/resources/META-INF/spring/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/schema/beans/spring-
beans-2.0.xsd">

<!-- Start of user code component definition -->
<import resource="classpath:META-INF/spring/org.sharengo.person/layer-controllers.xml"/>
<import resource="classpath:META-INF/spring/org.sharengo.person/layer-uis.xml"/>
<import resource="classpath:META-INF/spring/org.sharengo.person/layer-services.xml"/>
<import resource="classpath:META-INF/spring/org.sharengo.person/layer-daos-hibernate.xml"/>
<import resource="classpath:META-INF/spring/org.sharengo.person/layer-xfire-services.xml"/>

<!-- technical layers -->
<import resource="classpath:META-INF/spring/layer-hibernate.xml"/>
<import resource="classpath:META-INF/spring/layer-velocity.xml"/>
<import resource="classpath:META-INF/spring/layer-xfire.xml"/>
<!-- End of user code component definition -->

</beans>
```





# Run Unit Tests

- Adapt PersonProcessTest and run unit test in remote mode with: « PersonProcessWebServiceTest.java »

The screenshot displays an IDE interface with a JUnit test runner window on the left and a code editor on the right. The test runner window shows a successful execution of 2/2 tests with 0 errors and 0 failures. The code editor shows the implementation of the `PersonProcessWebServiceTest` class, which extends `PersonProcessTest` and uses XFire for remote testing.

```
Package Explorer | Hierarchy | JUnit
Finished after 4,054 seconds
Runs: 2/2 | Errors: 0 | Failures: 0
org.sharengo.person.process.impl.PersonProcessWe
  testRegisterPerson
  testListPersons

PersonProcessWebServiceTest.java | PersonProcessTest.java | PersonProcessImpl.java
package org.sharengo.person.process.impl;

import org.codehaus.xfire.XFire;
import org.codehaus.xfire.XFireFactory;
import org.codehaus.xfire.server.http.XFireHttpServer;
import org.codehaus.xfire.service.ServiceRegistry;
import org.junit.Test;

import org.sharengo.person.process.IPersonProcess;
import org.sharengo.person.process.IPersonProcessWebService;

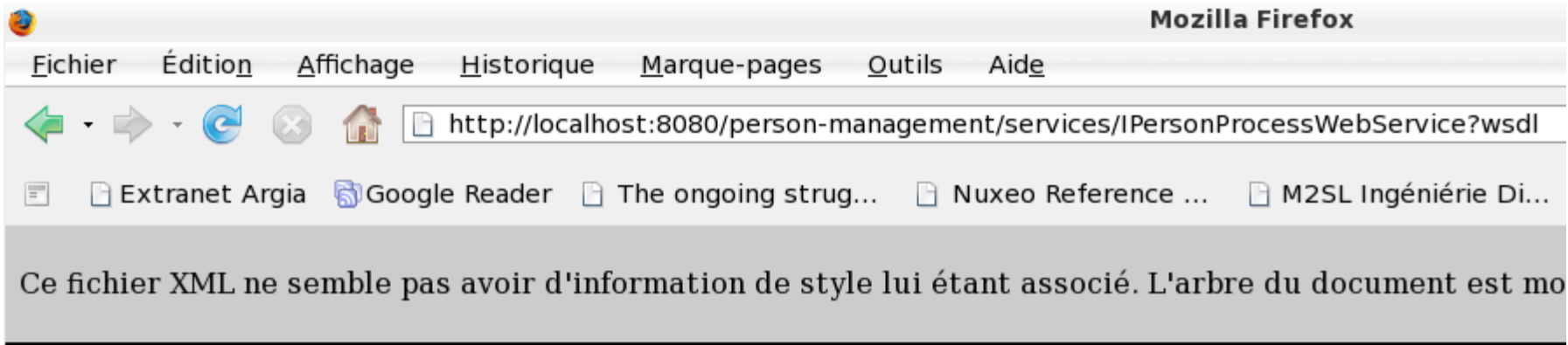
// Start of user code for import
// End of user code for import

public class PersonProcessWebServiceTest extends PersonProcessTest{
```





# Shared your Web Services



```

- <wsdl:definitions targetNamespace="http://process.person.sharengo.org">
- <wsdl:types>
  - <xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNa
    - <xsd:complexType name="Person">
      - <xsd:sequence>
        <xsd:element minOccurs="0" name="firstName" nillable="true" type="xsd:string"/>
        <xsd:element minOccurs="0" name="id" nillable="true" type="xsd:string"/>
        <xsd:element minOccurs="0" name="lastName" nillable="true" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  ..
  ..
  ..

```





Merci!





# Sources

- ABS reference doc : <http://sharengo.org/abs/>

