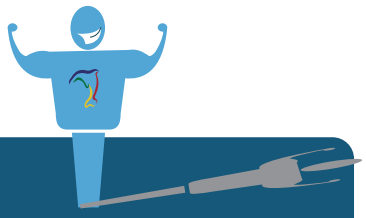
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

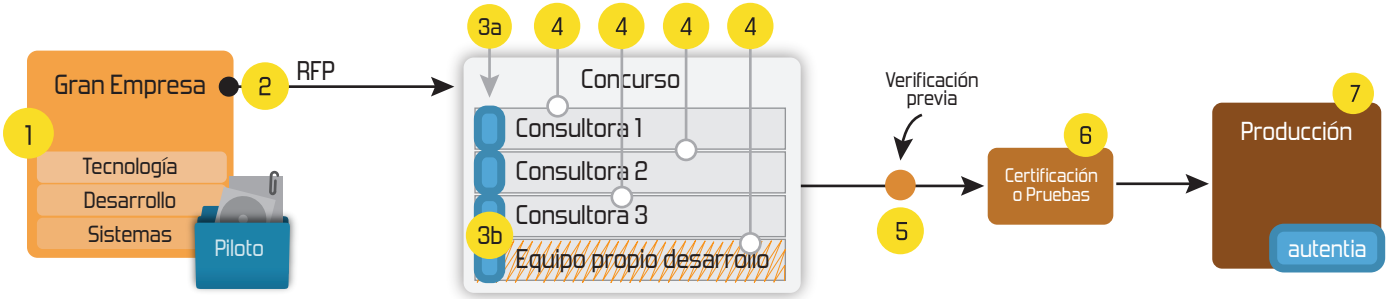
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Iván Zaera Avellón

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en info@autentia.com

Implementación de referencia de Portlets: Pluto

Introducción

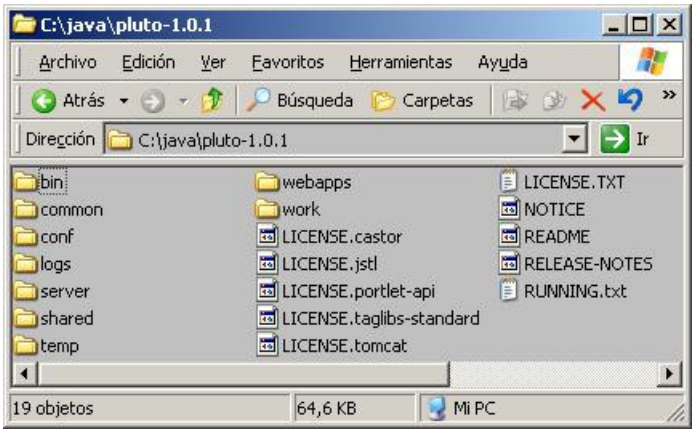
Apache Pluto es la implementación de referencia de la JSR 168. Las JSR (Java Specification Request) son estandarizaciones de tecnologías de forma que pasen a formar parte de la plataforma Java. En particular, la JSR 168 describe el API estándar de Java para desarrollo de Portlets. Un portlet es un componente Java que puede ser utilizado dentro de un portal web. Para entendernos, un portal es una aplicación web que distribuye sus componentes visuales de manera similar a como lo hacen los sistemas de ventanas de escritorio, es decir, se divide la pantalla en ventanas (no solapadas entre si) cada una de las cuales es responsable de producir su contenido.

A grandes rasgos, un portlet es algo similar a un servlet pero menos versátil, en el sentido de que no se puede producir la salida HTML que se desee y no se pueden hacer todas las cosas que se hacen en un servlet, como mandar redirecciones o cabeceras al cliente. Y si es mas limitado, ¿qué ventaja tiene entonces? Pues que los portlets pueden ser añadidos, quitados y recolocados de manera dinámica dentro de un portal, sin necesidad de reprogramar nada. Adicionalmente, una vez hemos desarrollado un portlet, lo podemos introducir en cualquier portal que siga la especificación JSR 168 y funcionará, aunque no sea áquel para el que lo diseñamos originalmente. Claramente, es un gran paso hacia la componentización de aplicaciones web.

En este tutorial vamos a desarrollar un portlet sencillo con Pluto 1.0.1 fijándonos en el manejo del portlet a bajo nivel. No nos va a interesar desarrollar portlets de manera eficiente, sino ver como son las tripas de un contenedor de portlets.

Instalación

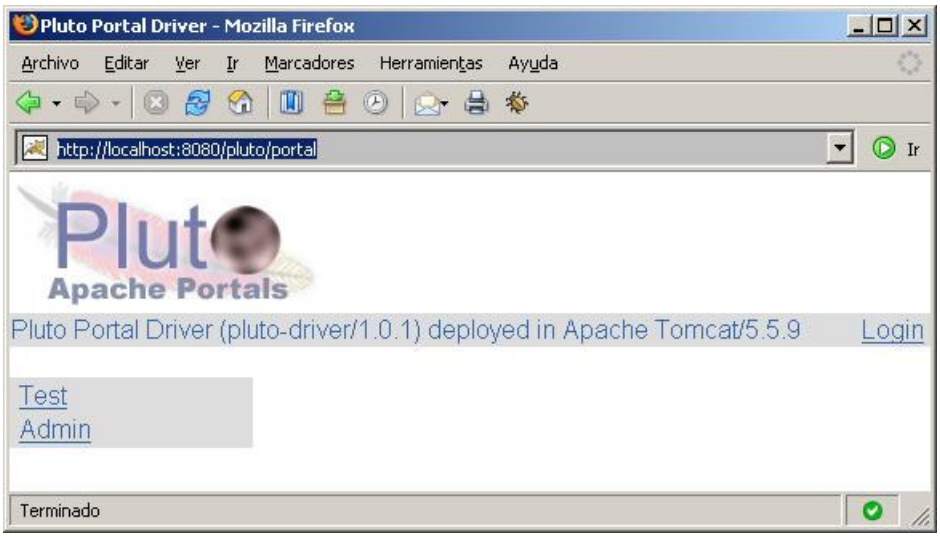
Nos bajamos Pluto de http://portals.apache.org/pluto/ y lo descomprimimos en el directorio que queramos. Al final de la descompresión, este directorio contendrá la aplicación Pluto junto con un Apache Tomcat que viene incluido. Se puede ejecutar Pluto en otro servidor de aplicaciones que no sea el que nos instala él por defecto, pero los desarrolladores han incluido un Tomcat para hacer mas fácil las pruebas y la ejecución.



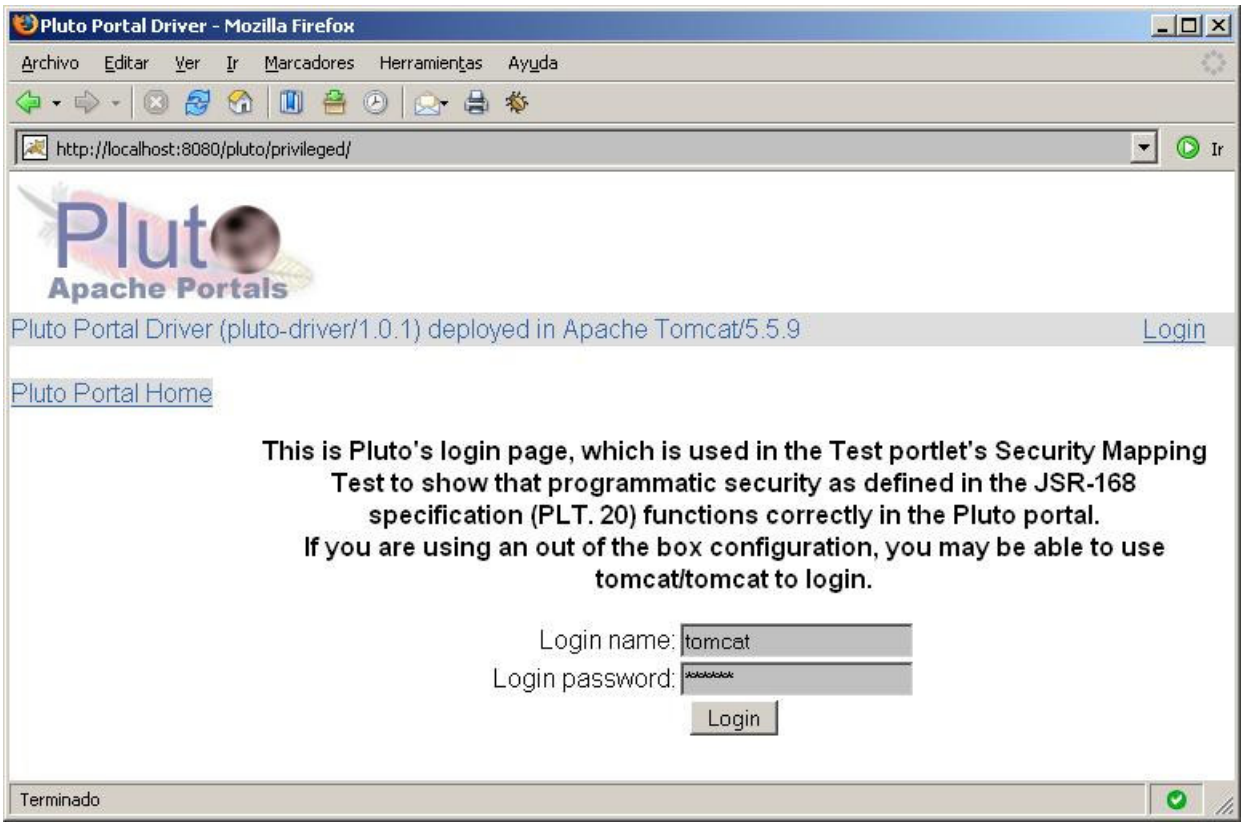
Entramos en el directorio "bin" y ejecutamos el archivo "startup.bat". Esto lanzará un Tomcat que arrancará el contenedor de portlets Pluto. Este Tomcat escucha en el puerto 8080, así que si tenemos otro programa escuchando en ese mismo puerto, hay que pararlo antes de arrancar Pluto.

Comprobación de la instalación

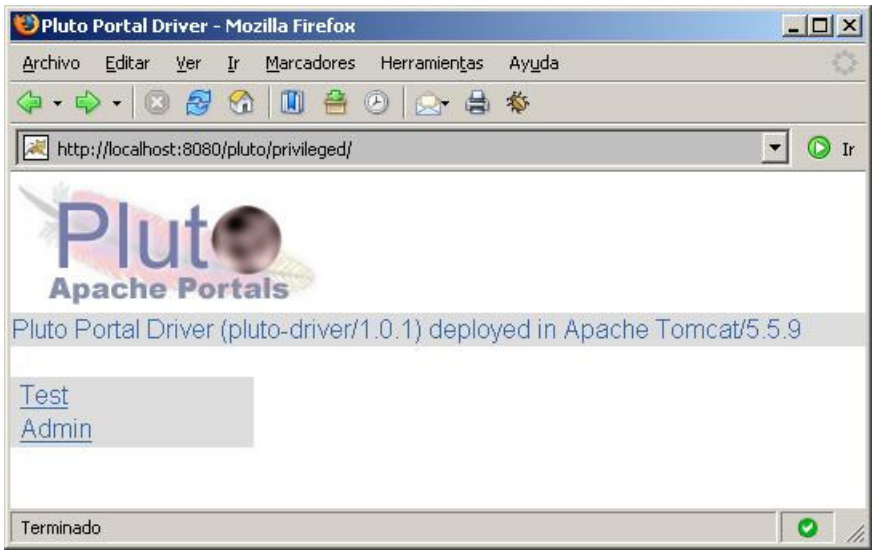
Una vez arrancado Pluto, arrancamos un navegador y nos conectamos a "http://localhost:8080/pluto/portal".



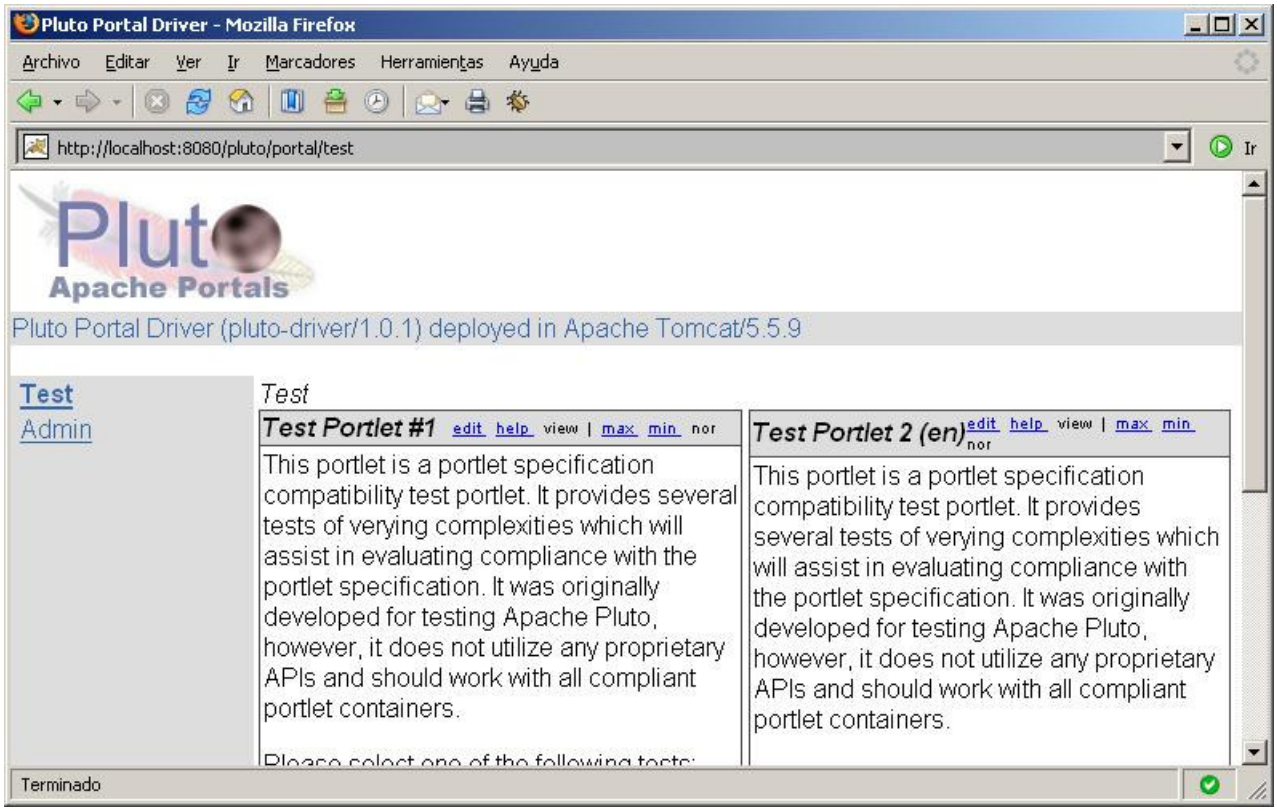
Lo primero que hacemos es pinchar en "Login" para pasar a la pantalla de autenticación. Esto se debe a que en la especificación JSR 168 se dice que el portal es el responsable de manejar la seguridad. Con esto se consigue que, al escribir un portlet no tengamos que preocuparnos de validar al usuario, facilitando la reubicación del portlet en distintos portales. Para entrar utilizaremos el usuario "pluto", con contraseña "pluto". Los usuarios se encuentran definidos en el fichero "tomcat-users.xml" del directorio "conf" del Tomcat que nos ha instalado Pluto.



Una vez autenticados, se nos muestra la pantalla de inicio:



Pinchamos en "Test" y pasaremos a ver los portlets:



Como se puede ver, lo único que tenemos configurado son dos portlets de prueba (cada una de las dos ventanitas con marco gris es un portlet). Si nos fijamos en la barra de titulo de cada portlet, vemos que hay seis enlaces. Como hemos dicho antes, el modelo de portlets simula el comportamiento de las aplicaciones de escritorio. así pues, estos enlaces son el botón de restaurar, el de maximizar y el de minimizar que nos encontramos en las ventanas de las aplicaciones de escritorio. Adicionalmente existen los enlaces de edición, ayuda y visualización, específicos de la especificación de portlets, que nos permiten cambiar el modo de funcionamiento del portlet.

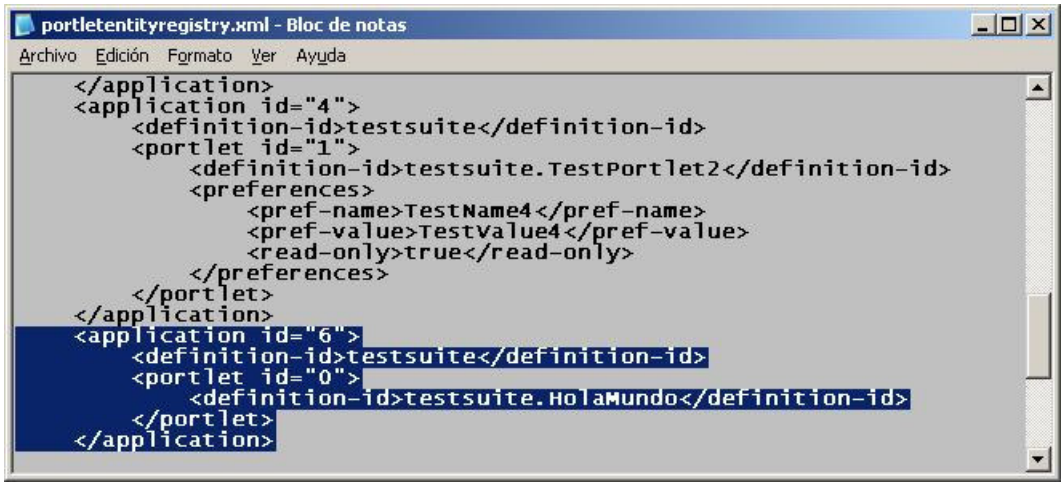
Creación de un portlet

Vamos a crear ahora nuestro primer portlet. Derrochando creatividad a mares, vamos a implementar un portlet que muestre la frase "Hola mundo". Lo colocaremos a la izquierda de los dos portlets de ejemplo.

Como Pluto es una implementación de referencia no provee muchas herramientas para facilitar la gestión de portales. Simplemente, si pinchamos en "Admin", podemos ver los portlets registrados, las páginas del portal, e incluso enviar un fichero war para desplegar una nueva aplicación de portlets. Nosotros, para hacer este tutorial más didáctico, vamos a añadir el portlet a mano dentro de la aplicación "testsuite", cambiando los ficheros de configuración de Pluto.

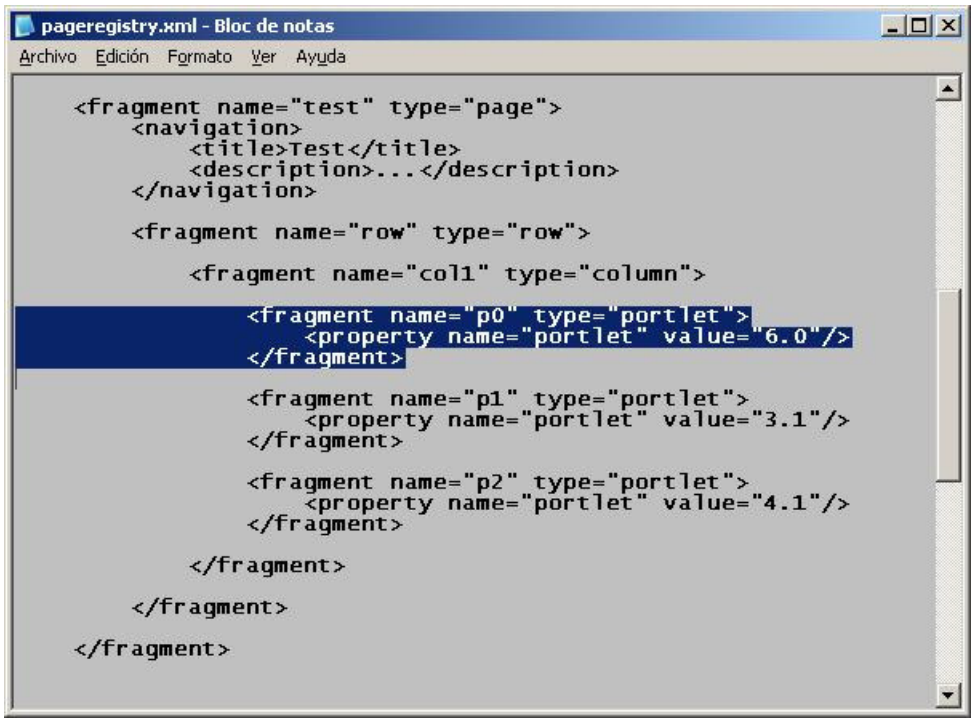
Registro del portlet

El primer paso es dar de alta el portlet. Para ello editamos el fichero "portletentityregistry.xml" del directorio "webapps/pluto/data" y registramos nuestro portlet "Hola mundo":



Asignación del portlet a la página Test

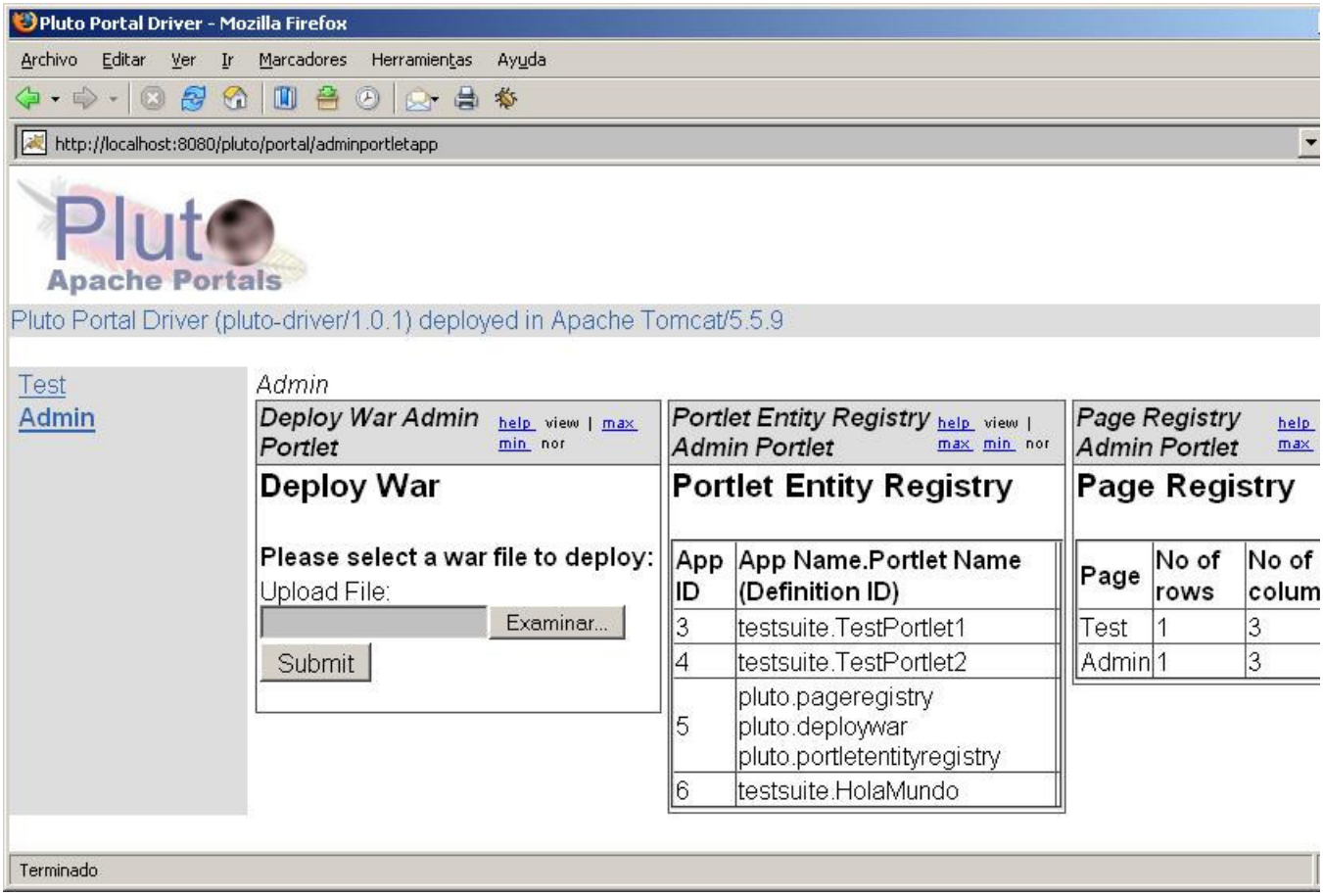
A continuación, asociaremos el portlet a la página "Test" del portal, para que se visualice. Para ello editamos el fichero "pageregistry.xml" en el mismo directorio que el "portletentityregistry.xml".



Con esto hemos añadido el portlet en la página "Test", a la izquierda de los otros dos portlets de prueba existentes. El número mágico "6.0" del atributo "value" sale de que al registrar el portlet le hemos dado el id "0" dentro de la aplicación "6" (ver apartado anterior).

A ver si ha funcionado...

Rearrancamos Tomcat con los dos ficheros modificados y nos volvemos a conectar al portal desde el navegador. Nos autenticamos y pinchamos en "Admin" y lo que vemos es la siguiente pantalla:

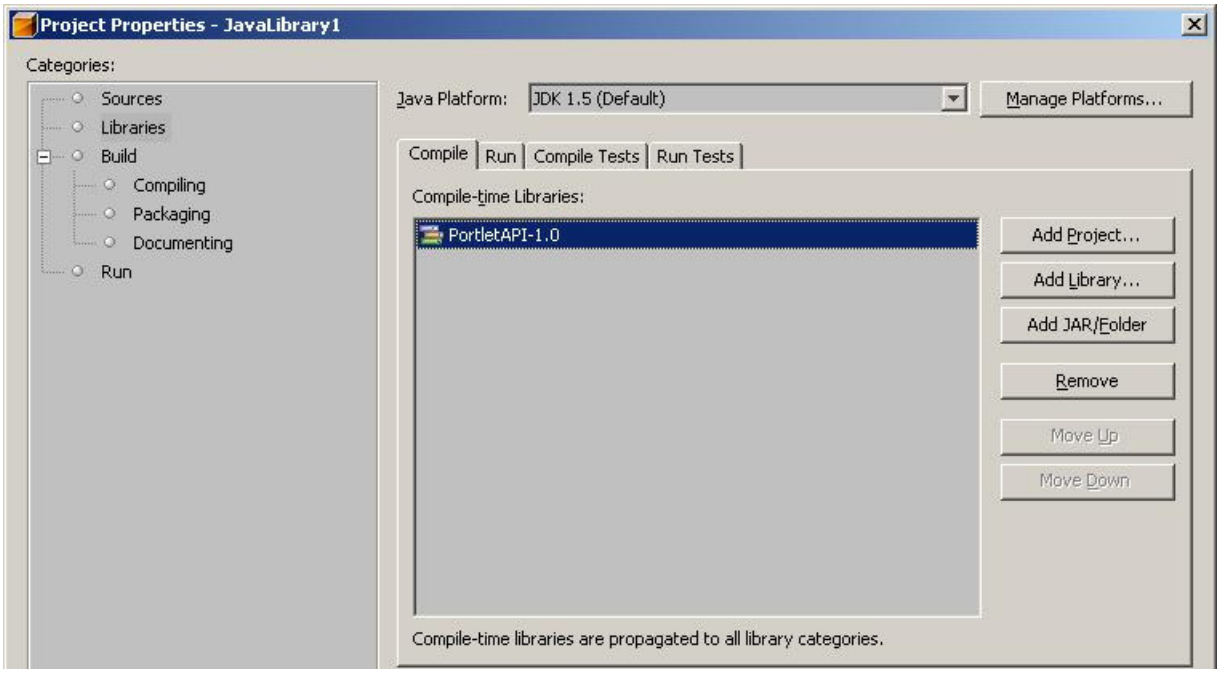


Como vemos en el "Portlet Entity Registry", nuestro nuevo portlet esta registrado. Además, en el "Page Registry", también aparece que la pagina "Test" tiene tres columnas, en lugar de las dos que tenía antes. Esto quiere decir que hemos dado de alta correctamente el portlet. Ahora sólo necesitamos implementarlo y desplegarlo en la aplicación web correspondiente (en nuestro caso "testsuite").

Implementar el portlet

Para implementar un portlet debemos crear un clase que herede de "GenericPortlet", al estilo de como implementamos los servlets (que heredan de "GenericServlet"). En general, la especificación de portlets utiliza, en donde es posible, terminología análoga a la de servlets, para hacer más fácil la comprensión. La clase "GenericPortlet" forma parte del Portlet API 1.0, cuyos ficheros JAR, javadoc y fuente podemos descargar de "http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html".

Una vez descargado el Portlet API 1.0, lo descomprimimos en un directorio y creamos, en nuestro IDE favorito, un proyecto Java al que añadiremos el fichero "portlet.jar" como librería.

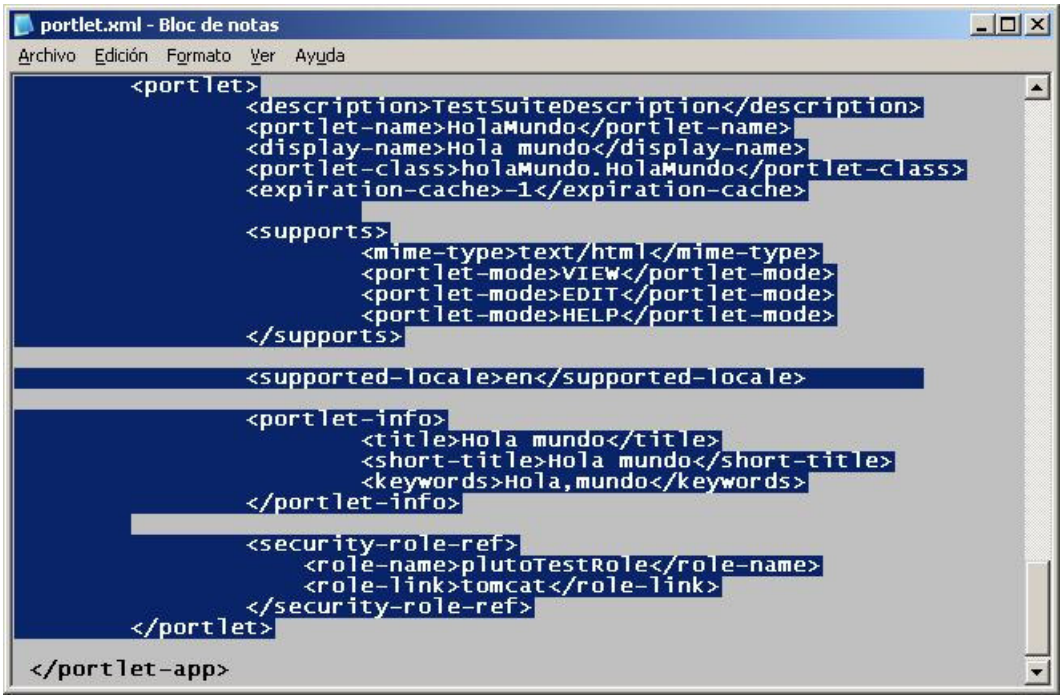


A continuación crearemos una clase llamada "HolaMundo" en el paquete "holaMundo" con el siguiente código fuente:

```
package holaMundo;
import java.io.IOException;
import java.io.PrintWriter;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
/**
 *
 * @author ivan
 */
public class HolaMundo extends GenericPortlet
{
 private String mensaje = "Hola mundo";

 public void doView(RenderRequest request, RenderResponse response)
 throws PortletException, IOException
 {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println(mensaje);
 }

 protected void doEdit(RenderRequest request, RenderResponse response)
 throws PortletException, IOException
 {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Edicion del mensaje:");
 out.println("<form action='"+response.createActionURL()+"'>");
 out.println("<input type='text' size='20' name='mensaje' value='"+mensaje+"'>");
 out.println("<input type='submit'>");
 out.println("</form>");
 }

 protected void doHelp(RenderRequest request, RenderResponse response)
 throws PortletException, IOException
 {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Esta es la pagina de ayuda del portlet 'Hola mundo'");
 }

 public void processAction(ActionRequest request, ActionResponse response) throws PortletException, IOException
 {
 mensaje = request.getParameter("mensaje");
 }
}
```

Compilamos la clase y ponemos el fichero .class generado en el directorio "webapps/testsuite/WEB-INF/classes/holaMundo" de Pluto. Esto es un poco chapucero, ya que la forma correcta de desplegar portlets es generando un archivo WAR con los recursos necesarios. Sin embargo, como hemos dicho antes, lo que vamos a hacer es añadir a mano un portlet a la aplicación "Test" que viene con Pluto y para ello es necesario que dicha aplicación pueda acceder al fichero .class de nuestro portlet. Es como si estuviésemos añadiendo un servlet a una aplicación web ya existente una vez esta ha sido desplegada.

Editamos el fichero "portlet.xml" del directorio "webapps/testsuite/WEB-INF" de Pluto para añadir nuestro portlet al final.



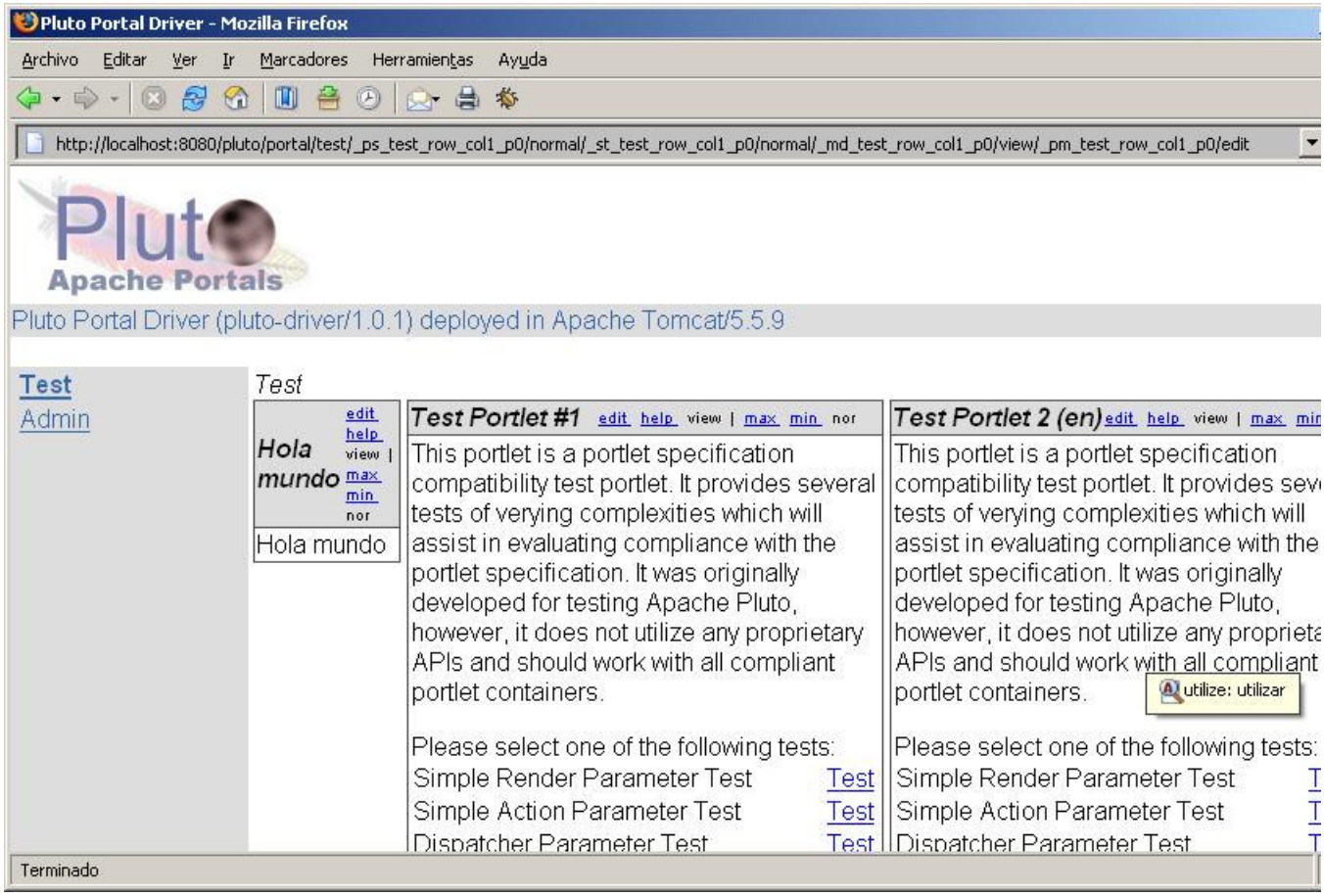
Como ultimo paso, editamos el fichero "web.xml" en el mismo directorio que el "portlet.xml". En este fichero vamos a añadir un servlet que invocara al portlet "Hola mundo". Dicho servlet lo provee Pluto y es el encargado de buscar el portlet en la configuración, instanciarlo, y llamarlo. Añadimos las lineas siguientes a continuación del último servlet:

```
<servlet>
<servlet-name>HolaMundo</servlet-name>
<display-name>HolaMundoWrapper (Pluto Invoker)</display-name>
<description>Auto Generated Portlet Invoker Servlet</description>

<servlet-class>org.apache.pluto.core.PortletServlet</servlet-class>
<init-param>
<param-name>portlet-class</param-name>
<param-value>holaMundo.HolaMundo</param-value>
</init-param>
<init-param>
<param-name>portlet-guid</param-name>
<param-value>testsuite.HolaMundo</param-value>
</init-param>
<security-role-ref>
<role-name>plutoTestRole</role-name>
<role-link>tomcat</role-link>
</security-role-ref>
</servlet>
<servlet-name>HolaMundo</servlet-name>
<url-pattern>/HolaMundo/*</url-pattern>
</servlet-mapping>
```

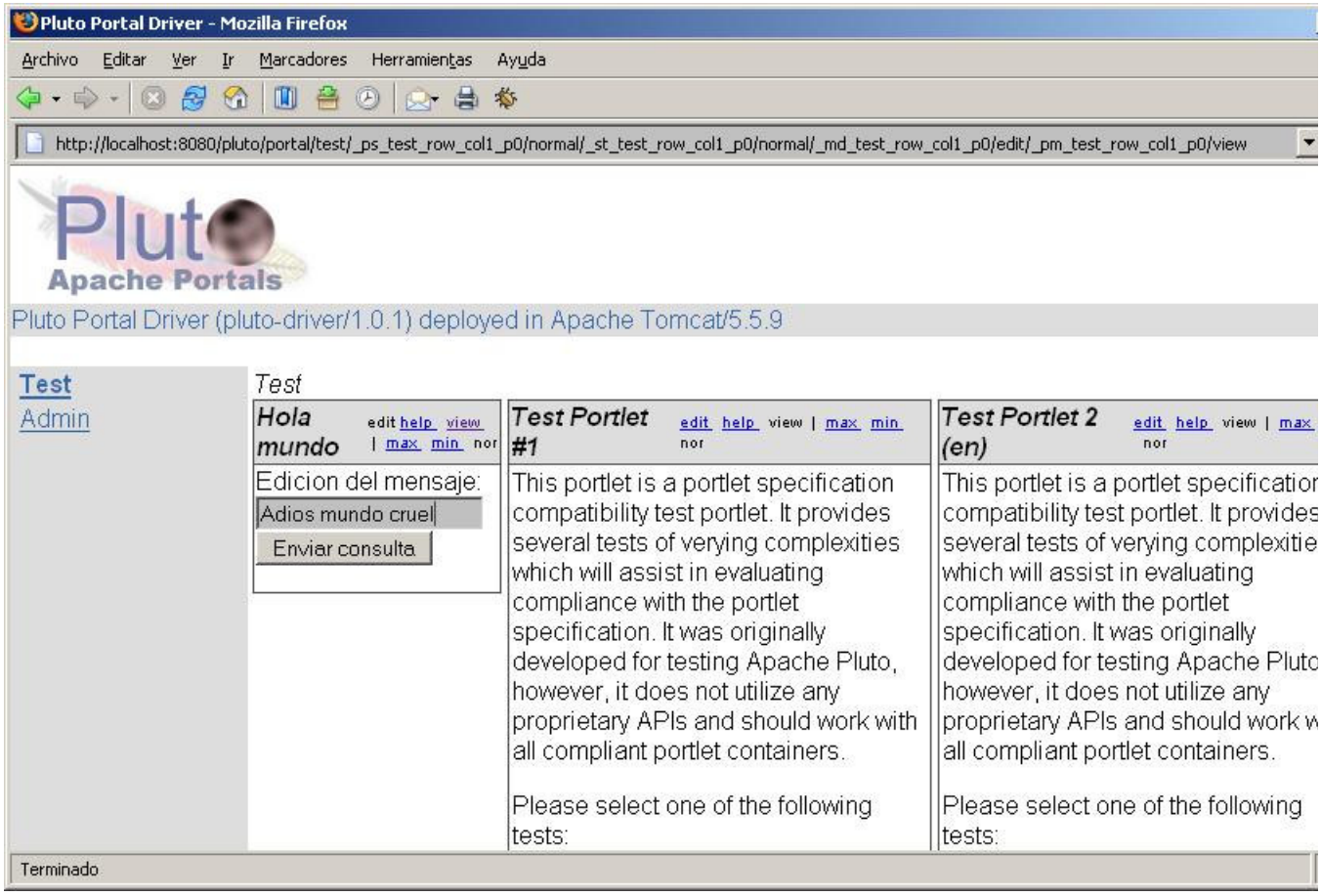
Una vez realizado el paso anterior, rearrancamos Pluto.

Probar el portlet

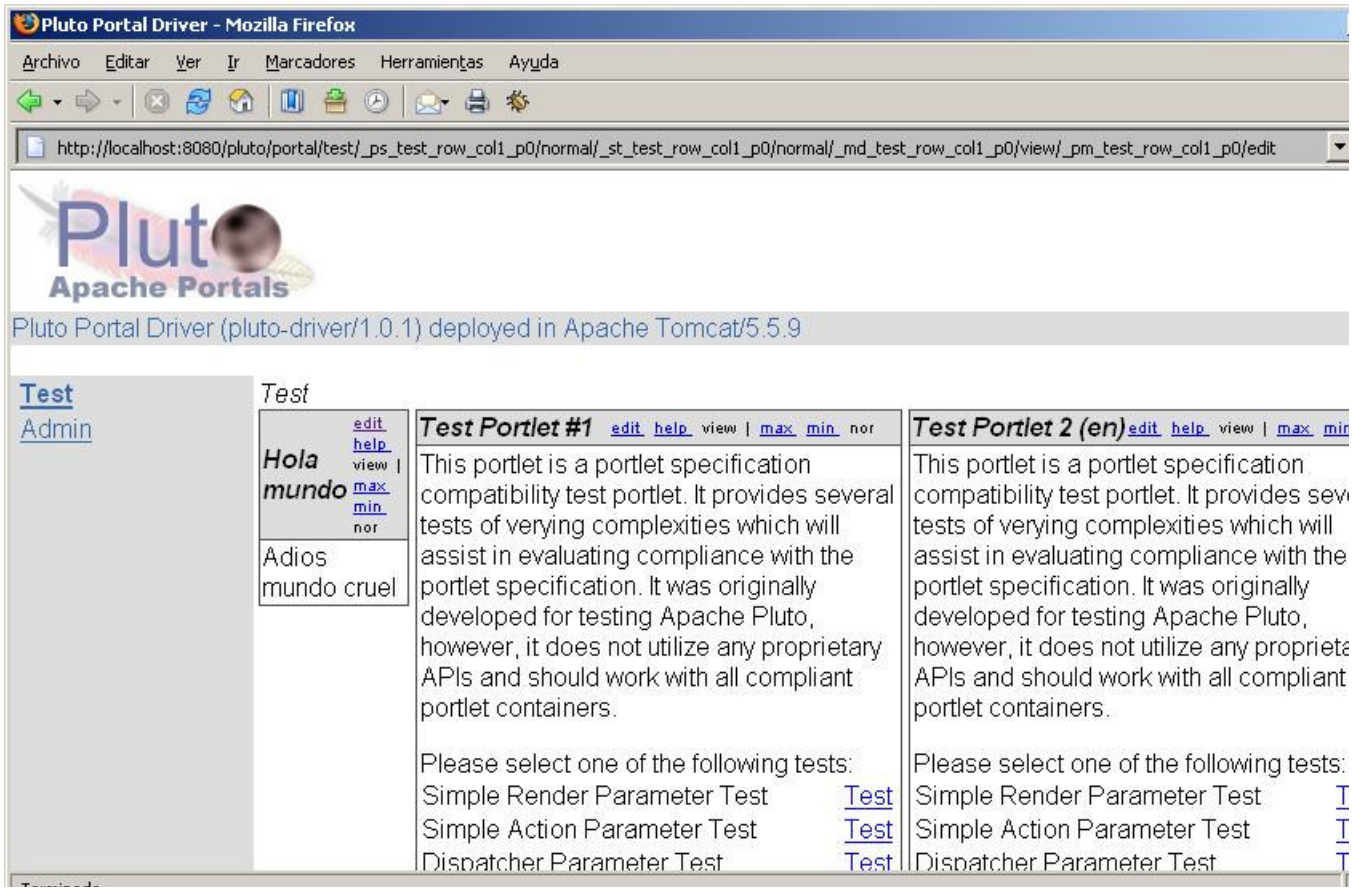
Ahora nos conectamos de nuevo a pluto con el navegador, hacemos el login y pinchamos en Test, con lo que veremos nuestro nuevo portlet:



Si pinchamos en "edit" se invocara al método "doEdit", que pinta un formulario para poder cambiar el mensaje:



Cambiamos el mensaje y pulsamos el botón de envío. Esto invocará al método "processAction" de nuestro portlet, que cambiara el valor del atributo "mensaje" al nuevo mensaje. La respuesta sera de nuevo el formulario. Si ahora pinchamos en "view" volveremos a la vista normal pero veremos que el mensaje ha cambiado:



Eso es casi todo, amigos

En este tutorial hemos visto como maneja internamente Pluto los portlets y como se implementa un portlet. No hay que ser muy avispado para ver que lo que hemos hecho es muy parecido al antiguo desarrollo de servlets pero teniendo cuidado de no manejar las URLs, peticiones, y respuestas a mano, sino usando el API de portlets para que nos las reescriba y así pueda mantener varios portlets en la misma pagina.

Otra peculiaridad que podemos apreciar es que esto nos retrotrae a los tiempos en que no teníamos JSPs ni paradigma MVC, con todo lo que ello conlleva. Es por ello que, como prueba de concepto este tutorial está bien, pero como aplicación del mundo real deja mucho que desear. Igual que en las aplicaciones web tratamos de emplear el paradigma MVC y la arquitectura multi-capa, es necesario usarlo también al desarrollar portlets.

No debemos olvidar que Pluto es una implementación de referencia y no una herramienta para el mundo real. Para el mundo real existen múltiples frameworks (como JetSpeed de Apache) que cumplen la JSR 168 pero facilitándonos la vida.

Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Gestión de contenidos y errores comunes	Os explicamos en que consiste la gestión de contenidos y cuales son los errores cometidos por multitud de empresas a la hora de abordar su implantación
Cachear porciones de JSPs	En este tutorial os ensañamos como incrementar increiblemente el rendimiento de vuestro Web basado en tecnología JSP con el FrameWork de cache OSCACHE
Instalar OpenCms	Open CMS es uno de los principales gestores de contenidos gratuitos basados en Java. Os ensañamos a instalarlo sobre MySQL
PHP Nuke en Windows 2000	Os mostramos como instalar paso a paso Php-Nuke en vuestro entorno Windows con MySQL
WebDav con IIS	En este tutorial podemos aprender como gestionar los contenidos de nuestro servidor Web sin necesidad de utilizar FTP.

Creando Servicios Web con Bea Workshop 8.1	En este artículo os mostramos como funciona la nueva herramienta de Bea, Workshop y como crear servicios web con ella