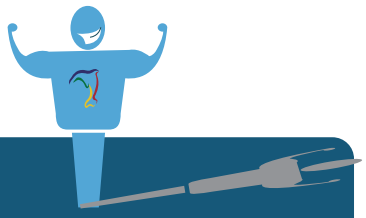
autentia

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...

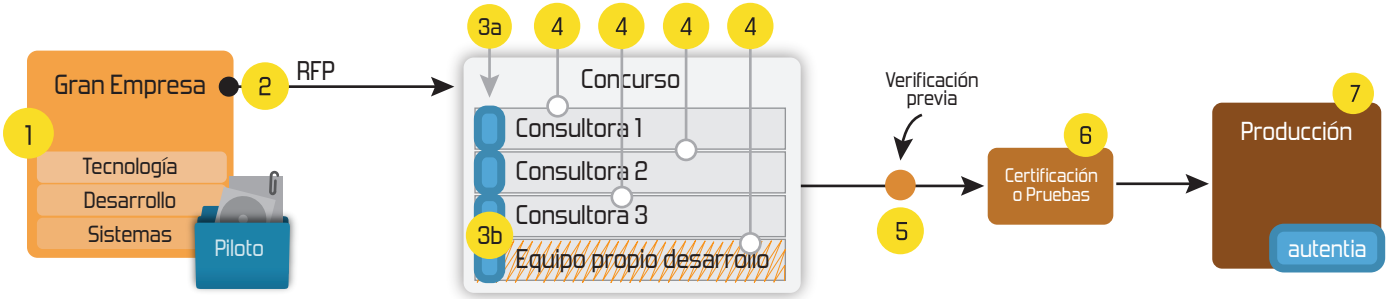
1. Desarrollo de componentes y proyectos a medida



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 conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por

Iván García Puebla

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Informática de Gestión por la Universidad de Valladolid.

Puedes encontrarme en Autentia

Somos expertos en Java/JEE

Descargar este documento en formato PDF: WebServicesConEstado.pdf

Fecha de creación del tutorial: 2008-10-27

Web Services con Estado

Introducción

Este tutorial está motivado por la pregunta de un alumno de un curso, acerca de mantener una sesión de diálogo entre un web service y su cliente. Veamos un ejemplo de lo que se viene al caso a denominar **web services con estado** (*stateful webservices*).

El código fuente del tutorial puede descargarse desde aquí.

Requisitos

Partimos del siguiente software

- Implementación de referencia del estándar JAX-WS 2.0/2.1 (https://jax-ws.dev.java.net/), versión de distribución 2.1.4, que podemos descargar de aquí: https://jax-ws.dev.java.net/2.1.4/. JAX-WS es el núcleo de METRO (conjunto de componentes que forman una WS stack, o pila de web services, de Sun).
- Por comodidad, usaré Eclipse como IDE.
- Servidor web Apache Tomcat 6.0.18, que podremos descargar desde http://tomcat.apache.org/download-60.cgi.

Si estás con Windows vista, la manera más adecuada de instalar la distribución JAX-WS es mediante el comando indicado en la web de descarga, es decir:

```
C:\TutorialWS>java -jar JAXWS2.1.4-20080502.jar
```

Creará la carpeta `C:\TutorialWS\jaxws-ri`. Crearemos una variable de entorno llamada `JAXWS_HOME` con el valor `C:\TutorialWS\jaxws-ri` y añadiremos `JAXWS_HOME\bin` al `PATH` del sistema.

Creación del web service con estado

Vamos a hacer un ejemplo relacionado con los tiempos que corren: un broker hace especulaciones en bolsa utilizando las operaciones expuestas por un web service con estado. El broker se validará, realizará operaciones y finalizará la 'sesión'. Un web service habitual (sin estado y habitualmente asíncrono) no sabría llevar la cuenta de lo ganado por el broker entre operación y operación, la información se pierde. Pero el que vamos a crear, sí mantiene memoria de lo acumulado, sin utilizar ningún tipo de persistencia en la lógica imllementada.

Últimos tutoriales

2008-10-27
Web Services con Estado

2008-10-24
Web Services con Axis2. Configuración y ejemplo

2008-10-22
Migración de JSP a Facelets

2008-10-22
Rock Band Wii en tu PC

2008-10-18
Cobertura: Como comprobar cuanto código prueban nuestros test

2008-10-17
maven-license-plugin: cómo gestionar la licencia de nuestros ficheros fuentes

2008-10-10
Cypal Studio: plugin de GWT para Eclipse

2008-10-10
Planificación de tareas con Spring

2008-10-09
Tutorial de Google Calendar Sync

2008-10-06
Instalación de GWT 1.5

Últimas ofertas de empleo

2008-10-03
Marketing - Experto en Marketing - MADRID.

2008-10-01
Atención a cliente - Call Center - MADRID.

2008-09-11
Otras Sin catalogar - BARCELONA.

En Eclipse creamos un nuevo proyecto web dinámico para facilitarnos posteriormente la generación del WAR, y agregamos las dependencias necesarias al Build Path, que serán los .jar existentes en `JAXWS_HOME\lib`.

Creamos un paquete llamado `com.autentia.ws.bolsa`, y ahí creamos las siguientes dos clases:

`CuentaInversion.java`:

```
package com.autentia.ws.bolsa;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;
import java.util.Random;
import java.util.Set;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;

import com.sun.xml.ws.developer.Stateful;
import com.sun.xml.ws.developer.StatefulWebServiceManager;

/**
 * <p>
 * Cuenta de inversion de un broker. Trabaja con un conjunto de titulos
 * conocidos
 * </p>
 * <p>
 * El broker puede manejar su cuenta de inversion desde cualquier parte ya que
 * se ofrece como un web service con estado
 * </p>
 *
 * @author Ivan Garcia Puebla - www.autentia.com
 * @version 1.0
 */
@Stateful
@WebService
@Addressing
public class CuentaInversion {

        // cartera de titulos de companias
        private Map<String, Activo> cartera;

        public final static String IENEGE = "IENEGE";
        public final static String BEBEUVA = "BEBEUVA";
        public final static String TECNOGUAY = "TECNOGUAY";
        public final static String INVESTIS = "INVESTIS";
        public final static String CASARIS = "CASARIS";

        /**
         * Alta de una cuenta de inversion
         */
        public CuentaInversion() {
                cartera = new Hashtable<String, Activo>(5);

                cartera.put(IENEGE, new Activo(IENEGE));
                cartera.put(BEBEUVA, new Activo(BEBEUVA));
                cartera.put(TECNOGUAY, new Activo(TECNOGUAY));
                cartera.put(INVESTIS, new Activo(INVESTIS));
                cartera.put(CASARIS, new Activo(CASARIS));
        }

        /**
         * Metodo que compra titulos sobre un precio minimo
         *
         * @param precioMinimo
         *            Precio minimo de compra
         * @param cantidad
         *            Cantidad de titulos
         */
        public void comprarPrecio(int precioMinimo, int cantidad) {

                if (precioMinimo < 0 || cantidad < 0)
                        return;

                // compramos titulos sobre el precio minimo
                Set<String> set = cartera.keySet();
                Iterator<String> it = set.iterator();
                while (it.hasNext()) {
                        Activo a = cartera.get(it.next());
                        if (a.getCotizacion() >= precioMinimo)
                                a.adquirir(cantidad);
                }
        }

        /**
         * Adquirir titulos de una compania
         *
         * @param ticker
         *            Ticker de la compania
         * @param cantidad
         *            Cantidad de titulos
         */
        public void comprarCompania(String ticker, int cantidad) {
                Activo a = cartera.get(ticker);
                if (a != null) {
                        a.getCotizacion();
                        a.adquirir(cantidad);
                }
```

```
}

/**
 * Obtiene los ingresos del broker
 *
 * @return Ingreso de la inversion actua
 */
public int getIngresos() {
        int total = 0;

        Set<String> set = cartera.keySet();
        Iterator<String> it = set.iterator();
        while (it.hasNext()) {
                total += ((Activo) cartera.get(it.next())).getMontante();
        }

        return total;
}

/**
 *
 * Este objeto es inyectado por la implementacion de referencia JAX-WS, y
 * ofrece varios metodos para manejar web services con estado.
 *
 */
public static StatefulWebServiceManager<CuentaInversion> manager;

/**
 * Fin de sesion explicito para liberar memoria
 */
public void finSesion() {
        manager.unexport(this);
}

/**
 * Entidad Activo Financiero
 *
 * @author Ivan Garcia Puebla - www.autentia.com
 * @version 1.0
 */
class Activo {

        private String ticker; // id de empresa que cotiza
        private int numero; // cantidad de titulos adquiridos
        private int montante; // capital de la inversion
        private int cotizacion; // precio de cotizacion

        /**
         * Constructor
         */
        public Activo(String ticker) {
                this.ticker = ticker;
                numero = 0;
                montante = 0;
                cotizacion = 0;
        }

        /**
         * @return el ticker del activo
         */
        public String getTicker() {
                return ticker;
        }

        /**
         * @param ticker
         *            establece el ticker del activo
         */
        public void setTicker(String ticker) {
                this.ticker = ticker;
        }

        /**
         * @return cantidad de titulos adquiridos
         */
        public int getNumero() {
                return numero;
        }

        /**
         * @return El montante
         */
        public int getMontante() {
                return montante;
        }

        /**
         * Obtiene el precio de cotizacion actual
         *
         * @return Precio actual
         */
        public int getCotizacion() {

                Random random = new Random();
                cotizacion = random.nextInt(10);

                return cotizacion;
        }
```

```
		/**
		 * Método que efectua la compra de acciones al precio actual de
		 * cotizacion
		 *
		 * @param numero
		 *            de acciones a comprar
		 */
		public void adquirir(int cantidad) {

			numero += cantidad;
			montante += (cantidad * cotizacion);

		}

	}

}
```

El código es sencillo de entender. Disponemos de una clase `Activo.java`, que representa una entidad de tipo activo financiero. Este activo tiene unas variables de clase que almacenan su estado y un método para comprar determinadas cantidades de ellos.

La clase `CuentaInversion.java` es la que nos interesa, pues simplemente anotándola con `@WebService`, `@Stateful` y `@Addressing` la estamos convirtiendo en un web service con estado al que podemos invocar sea cual sea el protocolo de transporte. ¡Todo eso con 3 palabras! *Small is Beauty* ;-)

Destacamos dos métodos:

- `StatefulWebServiceManager`, método que JAX-WS implementará por nosotros, encargado de manejar el estado. Es recomendable consultar el javadoc: https://jax-ws-architecture-document.dev.java.net/nonav/doc/?jaxws/package-summary.html.
- `finSesion()`. Sino invocamos el unexport del manager, tendremos pérdida de memoria, al no librear recursos tras dejar de utilizar el web service. Otra solución es establecer un timeout, como indica la documentación del punto anterior.

La otra clase que compondrá la lógica de nuestro web service es `Cartera.java`:

```
package com.autentia.ws.bolsa;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.wsaddressing.W3CEndpointReference;

/**
 * Este es un servicio web sin estado endpoint del que tiene estado
 *
 * @author Ivan Garcia Puebla - www.autentia.com
 *
 */
@WebService
public class Cartera {

	/**
	 * Metodo que proporciona al cliente una referencia a una cuenta de
	 * inversion
	 *
	 * @param usuario
	 *            Usuario
	 * @param password
	 *            Clave
	 * @return Referencia a una instancia cuenta de inversion
	 */
	@WebMethod
	public synchronized W3CEndpointReference accesoCartera(String usuario,
	                String password) {

		CuentaInversion cuenta = null;

		if (!validar(usuario, password))
			return null;

		cuenta = new CuentaInversion();
		return CuentaInversion.manager.export(cuenta);

	}



	// TODO refactorizar este metodo
	/**
	 * Metodo de validacion
	 *
	 * @param usuario
	 *            Usuario
	 * @param password
	 *            Clave
	 * @return Acceso valido (true) o denegado (false)
	 *
	 */
	protected static boolean validar(String usuario, String password) {

		final String usuarioSecreto = "Alberto";
		final String passwordSecreta = "emc2";

		return (usuarioSecreto.equals(usuario) && passwordSecreta
				.equals(password));

	}

}
```

Esta clase también es un web service pero sin estado. Mediante el método `accesoCartera()` estamos validando un usuario y devolviendo al otro extremo de la invocación (el cliente), una referencia al web service con estado que realmente nos interesa.

Teniendo estas clases, podemos generar el web service con los comandos:

```
wsgen -cp . -keep com.autentia.ws.bolsa.CuentaInversion
```

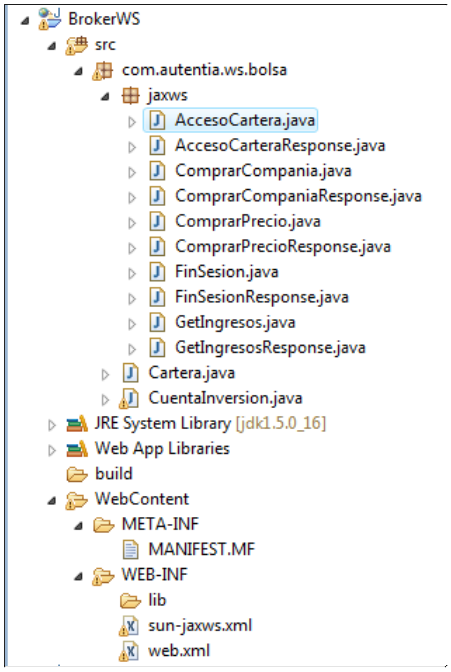
y

```
wsgen -cp -keep com.autentia.ws.bolsa.Cartera
```

Nos habrá generado el conjunto de clases necesarias para el funcionamiento del web service, en un nuevo paquete llamado `com.autentia.ws.bolsa.jaxws`.

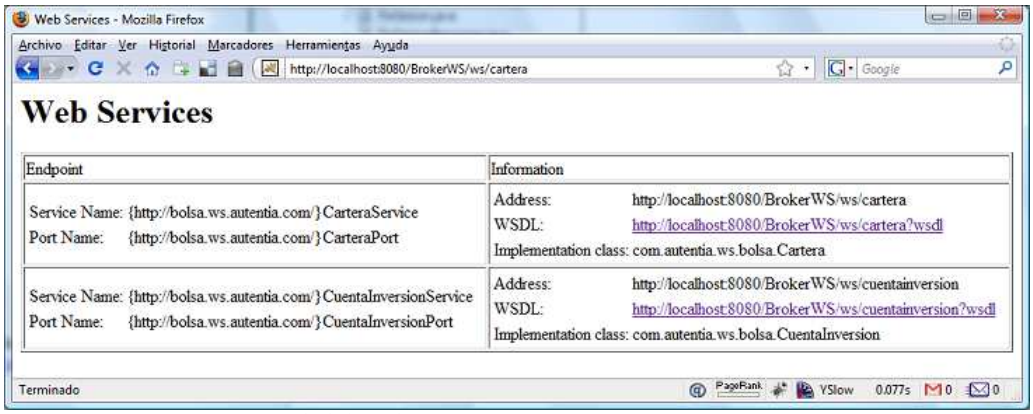
Asimismo implementamos los ficheros `web.xml` y `sun-jaxws.xml` (más información en: Metro: pila de webservices de Sun). Puedes ver su contenido en la descarga de los fuentes del tutorial.

En este punto, el proyecto tiene el siguiente aspecto:



Proyecto del web service en Eclipse

A continuación generamos el ensamblado war: `BrokerWS.war` y lo copiamos en el directorio webapps de Tomcat para su despliegue. Podremos acceder al estado del web service en la dirección http://localhost:8080/BrokerWS/ws/cartera:



Estado de los web services cartera y cuentainversion en Tomcat

Pasamos a generar el cliente.

Creación del cliente del web service con estado

Creamos un nuevo proyecto de tipo Java en Eclipse, y agregamos las librerias de JAX-WS, como en el caso del proyecto del servidor.

Para generar las clases necesarias para la comunicación con el web service ya publicado, ejecutamos los comandos:

```
wsimport -s src -p com.autentia.broker -Xnocompile http://localhost:8080/BrokerWS/ws/cartera?wsdl
```

y

```
wsimport -s src -p com.autentia.broker -Xnocompile http://localhost:8080/BrokerWS/ws/cuentainversion?wsdl
```

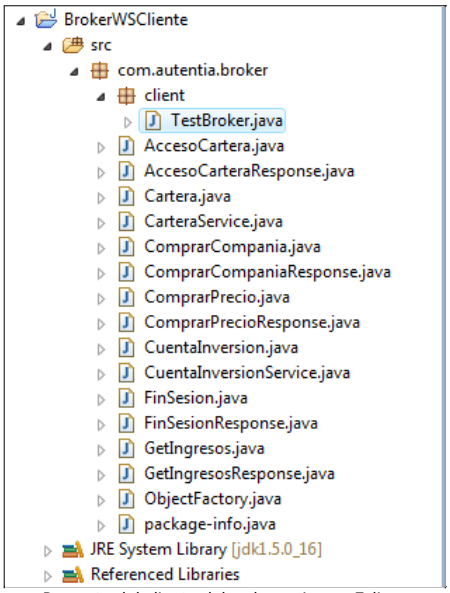
y se habrán creado en el paquete que hemos especificado, `com.autentia.broker`. En un paquete `com.autentia.broker.client` creamos una clase llamada `TestBroker.java` donde implementaremos la lógica del cliente:

```
package com.autentia.broker.client;

import com.autentia.broker.Cartera;
import com.autentia.broker.CarteraService;
import com.autentia.broker.CuentaInversion;
import com.autentia.broker.CuentaInversionService;

/**
 *
 * <p>
 * Cliente del web service de sesion de bolsa de un broker
 * </p>
 * <p>
 * Ejemplo de uso de un web service con estado
 * </p>
 *
```

```
 *
 * @author Ivan Garcia Puebla - www.autentia.com
 * @version 1.0
 *
 */
public class TestBroker {

        /**
         * @param args
         */
        public static void main(String[] args) {

                // creamos los objetos necesarios para el dialogo
                Cartera cartera = new CarteraService().getCarteraPort();
                CuentaInversionService cuentaService = new CuentaInversionService();

                try {

                        // nos validamos ante el servicio:
                        CuentaInversion cuenta = cuentaService.getPort(cartera
                                        .accesoCartera("Alberto", "emc2"), CuentaInversion.class);

                        // realizamos operaciones sobre nuestra cuenta de inversion

                        // compramos 25 titulos a mayor precio que 1$
                        cuenta.comprarPrecio(3, 25);
                        cuenta.getIngresos();
                        System.out.println("Montante acumulado: " + cuenta.getIngresos());

                        // compramos 100 titulos de una compania
                        cuenta.comprarCompania("BEBEUVA", 100);
                        System.out.println("Montante acumulado: " + cuenta.getIngresos());

                        // compramos 33 titulos de una compania que no es del broker
                        cuenta.comprarCompania("ssd", 33);
                        System.out.println("Montante acumulado: " + cuenta.getIngresos());

                        // compramos 1850 titulos de una compania
                        cuenta.comprarCompania("CASARIS", 1850);
                        System.out.println("Montante acumulado: " + cuenta.getIngresos());

                        // finalizamos la sesion
                        cuenta.finSesion();

                } catch (NullPointerException excepcionUsuarioNoAutorizado) {
                        System.out.println("broker ID o password incorrectos");
                }

        }

}
```

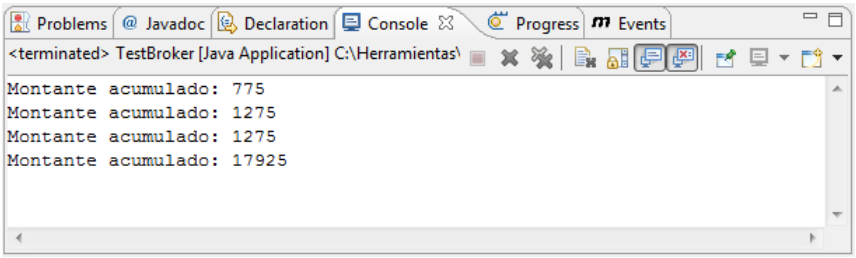
El código se explica por sí solo.

En este punto el estado del proyecto en Eclipse es:



Proyecto del cliente del web service en Eclipse

Ejecutamos la clase `Test.java` y obtenemos lo siguiente:



Resultado de la inversión de nuestro broker

Todo esto puedes ejecutarlo tú mismo con el código fuente del tutorial, preparado para ser importado como '*proyecto existente*' en Eclipse (JEE).

Sólo tendrás que añadir las librerías de `JAXWS_HOME\lib` a ambos proyectos.

Conclusión

Este tutorial muestra otra de las capacidades de los servicios web. Resulta cuando menos interesante seguir aprendiendo sobre ellos, ¿verdad David? :-)

- Puedes opinar sobre este tutorial haciendo clic aquí.
- Puedes firmar en nuestro libro de visitas haciendo clic aquí.
- Puedes asociarte al grupo AdictosAlTrabajo en XING haciendo clic aquí.
- Añadir a favoritos Technorati. ADD THIS BLOG TO MY Technorati FAVORITES

Servicio de notificaciones:

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales.

Formulario de subcripción a novedades:

E-mail [ ] Aceptar

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	pdf
Axis2. Ejemplo de creación de un servicio Web	En este tutorial veremos como crear un servicio web a partir de un interface Java así como otros aspectos de está tecnología.	2008-04-04	2702	pdf
Uso básico de AmberPoint express con Axis	En este tutorial se muestra cómo utilizar AmberPoint Express para monitorizar un servicio web que hayamos hecho nosotros, y se cuentan algunas cosas inesperadas que he encontrado.	2006-11-02	4087	pdf
Servicios Web RESTful en Axis 2	En este tutorial vamos realizar una descripción de REST y vamos a ver un ejemplo práctico de un Servicio Web RESTful en AXIS 2	2008-04-03	2951	pdf
Monitorización de Web Services con Glassfish Wsmonitor	En este tutorial vamos a realizar una introducción a una herramienta de monitorización de mensajes SOAP o Servicios Web en general.	2008-04-04	943	pdf
Web Services en tu IPAQ	Cesar Crespo nos enseña como programar accesos Web Services desde tu IPAQ en Visual C++ con PocketSOAP, Apache SOAP y Axis	2004-08-02	34678	pdf
Jersey: la implemetación de RESTFull de Sun	En este tutorial Germán nos enseña cómo usar RESTFull con la tecnología de Sun.	2008-04-05	1080	pdf
Metro: pila de webservices de Sun. Integración con Maven 2	En este tutorial Germán nos enseñara a integrar la generación de webservices con Metro y Maven2.	2008-04-05	1388	pdf
Trabajando con Axis	Utilizando Apache Axis, os mostramos otro interesante tutorial que ilustra su utilización para implementar web services	2006-04-07	34633	pdf
Web Services con Axis2. Configuración y ejemplo	Este es un tutorial básico que introduce a los servicios web en Java, y muestra cómo configurar el equipo con Eclipse, Apache Tomcat y Axis2 para poder crear luego un web service de ejemplo	2008-10-24	105	pdf
Metro: pila de webservices de Sun.	NE este tutorial Germán nos enseñara qué es y cómo usar Metro: pila de webservices de Sun en nuestras aplicaciones	2008-04-05	2203	pdf

Nota: