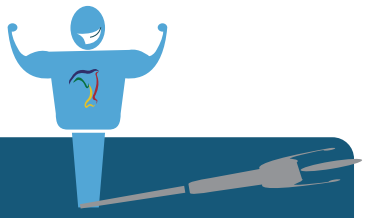
autentia

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...

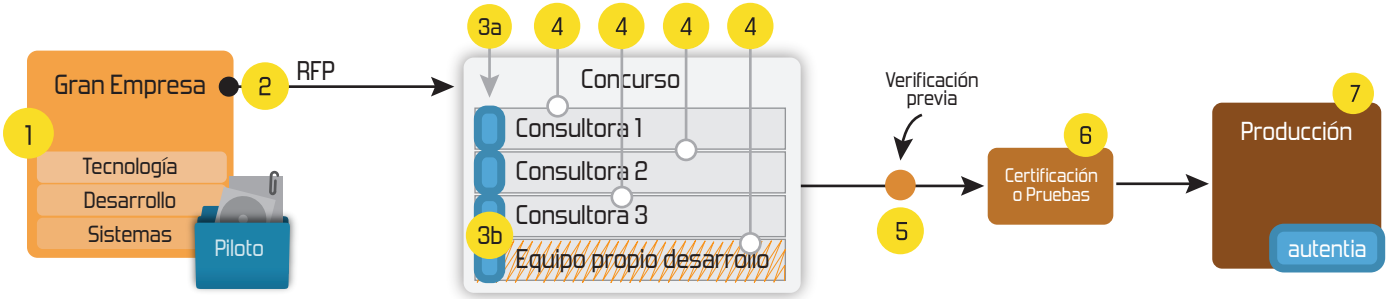
1. Desarrollo de componentes y proyectos a medida



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

Alberto Barranco Ramón

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2014-11-18

Tutorial visitado 19 veces Descargar en PDF

Analizando WSO2 Business Rules Server

0. Índice de contenidos.

1. Introducción

Mucho se ha hablado de la arquitectura **SOA**, de las ventajas y bondades de esta arquitectura. No solo se trata de **exponer servicios** y ya. Es también su gestión, el ciclo de vida de los mismos, su mantenimiento, evolución, documentación, o lo que se denomina **"gobierno SOA"**.

También podríamos incluir en esta arquitectura la medición de quien, cuando y cómo se consumen los servicios que exponemos en nuestra organización. La metainformación que saquemos de estos datos puede resultar en nuevos planes de negocio o la modificación de las estrategias actuales de nuestra empresa. Bajo mi punto personal de vista, la medición de estos datos es importantísima porque proporciona una información de como interactua el mercado/mundo con nosotros. Y no son datos imaginarios, o cómo creemos que se da esta relación, sino datos empíricos.

Hoy en día tenemos varias suites de herramientas que nos permiten construir y gestionar este tipo de arquitecturas, y encajar todas las piezas de forma que tengamos los servicios expuestos, documentados, con los procesos estandarizados y comunicados entre ellos.

Una de estas suites es **WSO2**, de la que ya hemos hablado en algún otro tutorial. A continuación vamos a ver que es **WSO2 Business Rules Server** y cómo se instala. Vamos a exponer un servicio con lógica de negocio en base a reglas y vamos a ver como podemos probarlo y consumirlo desde un cliente real.

2. Entorno

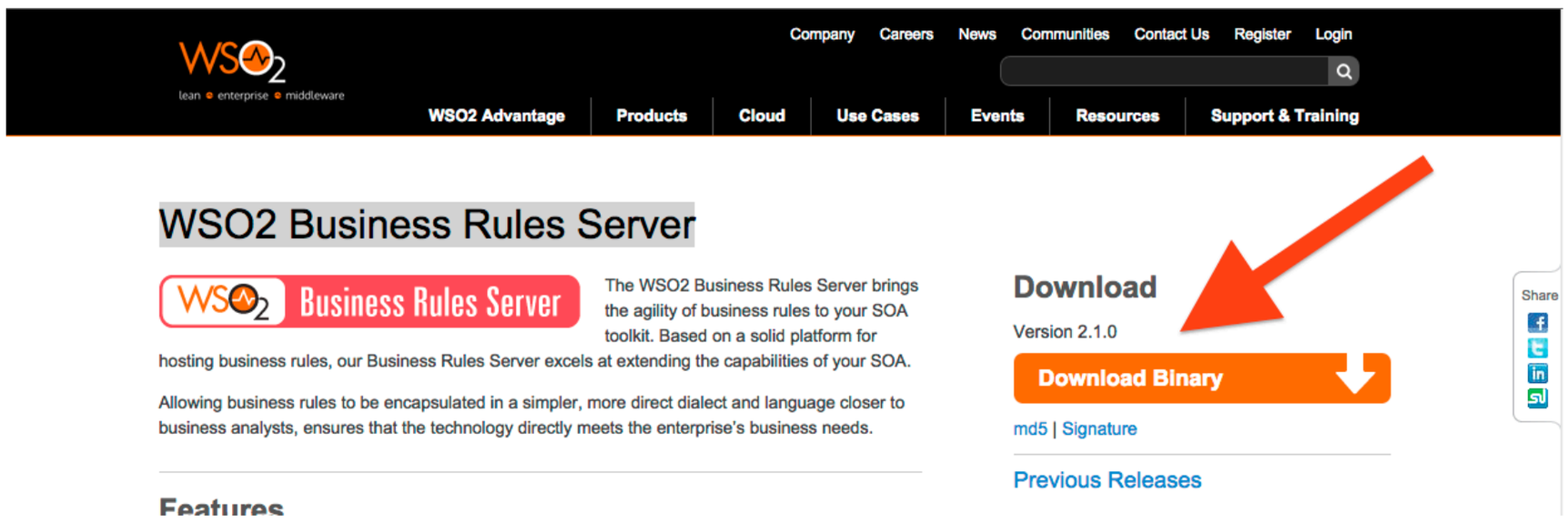
- Hardware: Portátil MacBook Pro 15' (2.3 GHz Intel i7, 16GB DDR3 SDRAM, 500GB HDD).
- Sistema Operativo: Mac OS Mavericks 10.9.5.
- Software relacionado con el tutorial: Eclipse Kepler, java 1.6.0_65, maven 3.1.1.

3. Instalando WSO2 Business Rules Server

Me gustaría empezar comentando que el producto es código libre, y que cualquiera puede usarlo y modificarlo, lo cual es

un punto muy importante a tener en cuenta. Con esta pieza de la suite de productos de **WSO2** lo que pretendemos es exponer los servicios, monitorizar los que están arrancados, los que no, ver el estado de la máquina, acceso a logs...

Antes que nada podemos entrar en el site oficial, leernos todo lo que ofrece y determinar si nos encaja o no. En caso afirmativo pinchamos en **Download Binary**.



El único prerequisito obligatorio es tener instalada la máquina virtual de java (1.6.24 +). De todas formas aquí hay una lista más detallada.

Esto nos baja un archivo .zip. Lo podemos descomprimir en la ruta que queramos. Para ejecutarlo simplemente desde el directorio raíz de la instalación ejecutamos ./bin/wso2server.sh. Esto es para sistemas operativos basados en linux. Los usuarios de windows podéis usar el .bat equivalente.

La consola nos muestra si la aplicación arranca de forma correcta, y la url donde está la herramienta desplegada. Accedemos y nos logamos con el usuario: **admin** y password: **admin**.

```
[2014-11-07 15:46:50,939]  INFO {org.wso2.carbon.core.internal.StartupFinalizerServiceComponent} -  Server          :  WSO2 Business Rule Server-2.1.0
[2014-11-07 15:46:50,939]  INFO {org.wso2.carbon.core.internal.StartupFinalizerServiceComponent} -  WSO2 Carbon started in 10 sec
[2014-11-07 15:46:51,111]  INFO {org.wso2.carbon.ui.internal.CarbonUIServiceComponent} -  Mgt Console URL  : https://192.168.168.92:9443/carbon/
```

El hecho de bajarse un .zip, descomprimirlo, ejecutar un .sh y que funcione dice mucho. Después de pegarme con muchas herramientas, y pasar horas y horas por google y stackoverflow, bajar algo, ejecutarlo, y que funcione !! es un soplo de aire fresco.

4. Exponiendo reglas como servicios

Vamos a usar uno de los ejemplos de la web y nos vamos a crear una regla con las siguientes restricciones:

- Un pedido de la compañia A se acepta solo si el número de stock es mayor que 10.
- Un pedido de la compañía B se acepta cuando el precio es mayor que 100.
- Un pedido de la compañía C se acepta cuando el precio es mayor que 50, y el número de stock es menor que 200.

La regla sería la siguiente:

```
package OrderApproval;

import samples.userguide.OrderAccept;
import samples.userguide.OrderReject;
import samples.userguide.PlaceOrder;

rule "Order Approval Rule" dialect "mvel" no-loop true salience 4

when
$placeOrder : PlaceOrder( ( symbol == "Company A" && quantity > 10 ) || ( symbol == "
then

OrderAccept orderAccept = new OrderAccept();
orderAccept.setMessage("Accepted order for: "+ $placeOrder.quantity + " stocks of "+
$placeOrder.symbol +" at$ " + $placeOrder.price);
insertLogical(orderAccept);

end

rule "Company A Order Deny Rule" dialect "mvel" no-loop true salience 3

when
not ( OrderAccept())
$placeOrder : PlaceOrder( symbol == "Company A" )
then
retract($placeOrder);
OrderReject orderReject = new OrderReject();
orderReject.setReason("An Order for stocks of Company A is accepted only if the numbe
insertLogical(orderReject);
end

rule "Company B Order Deny Rule" dialect "mvel" no-loop true salience 2
when
not ( OrderAccept())
$placeOrder : PlaceOrder( symbol == "Company B" )
then
retract($placeOrder);
OrderReject orderReject = new OrderReject();
orderReject.setReason("An Order for stocks of Company B is accepted only if the stock
insertLogical(orderReject);
end

rule "Company C Order Deny Rule" dialect "mvel" no-loop true salience 1
when
not ( OrderAccept())
$placeOrder : PlaceOrder( symbol == "Company C" )
then
retract($placeOrder);
OrderReject orderReject = new OrderReject();
```


publish/subscribe con Spring Data Redis

» Creación paso a paso de un webscript Alfresco

» Integración de MonkeyTalk en iOS

» Soporte de Redis con Spring: RedisTemplate

Últimos Tutoriales del Autor

» Primeros pasos en Android (II)

» Eclipse Juno, la versión 4.2 de Eclipse

» Primeros pasos en Android.

» Configuración y tuning de servidores de producción

» Responsive CSS. Redimensionado de imagenes con Skeleton

```
orderReject.setReason("An Order for stocks of Company C is accepted only if the stock
insertLogical(orderReject);
end
```

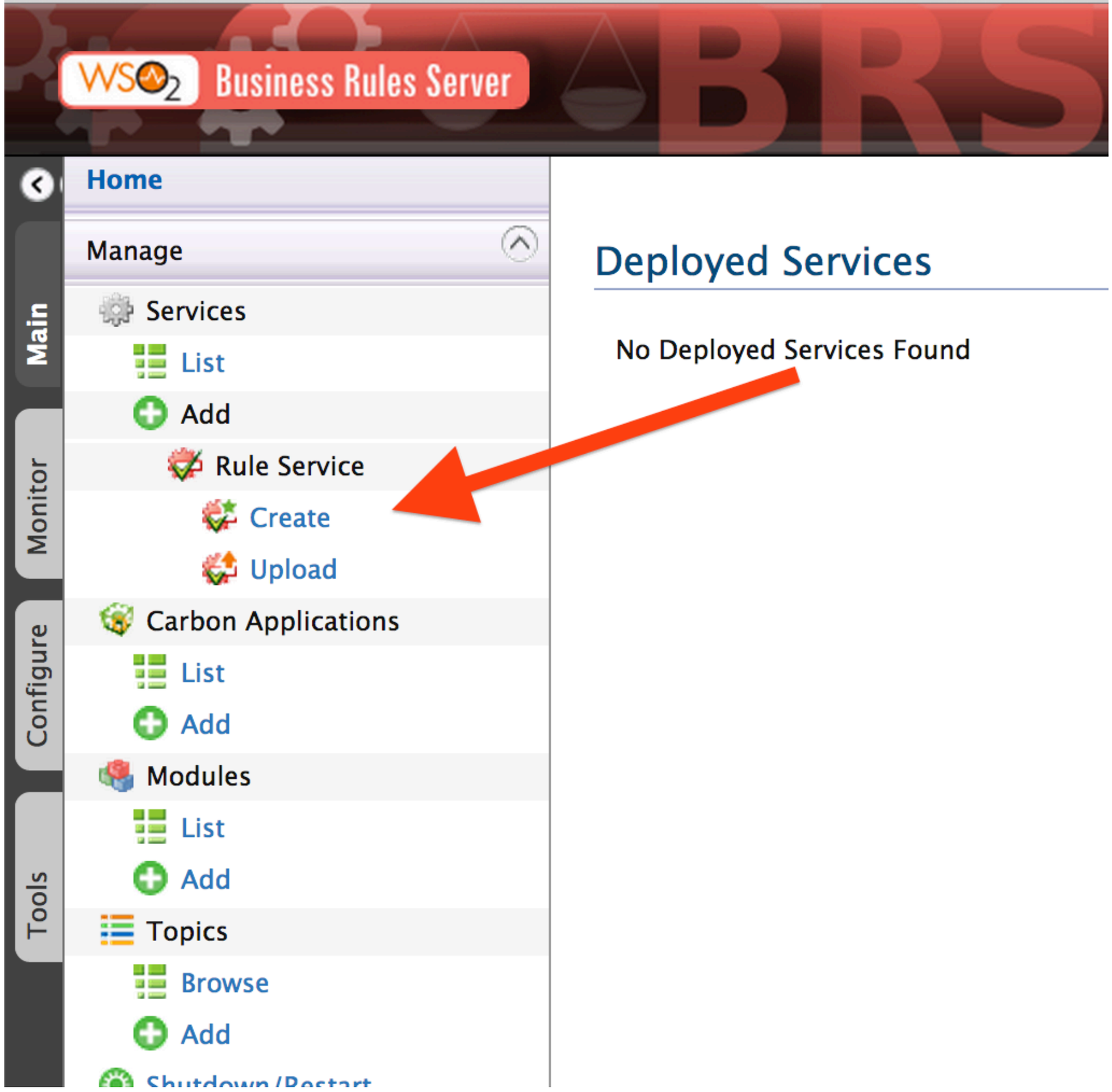
A estas reglas se les van a pasar hechos (facts) como entrada, y van a producir respuestas aplicando las reglas. A continuación vamos a poner el pedido y las posibles respuestas cuando aplicamos las reglas.

```
package samples.userguide;

/**
* Represents the fact of a customer places an order
*/
public class PlaceOrder {

    String symbol;
    int quantity;
    double price;

    public String getSymbol() {
        return symbol;
    }

    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
```

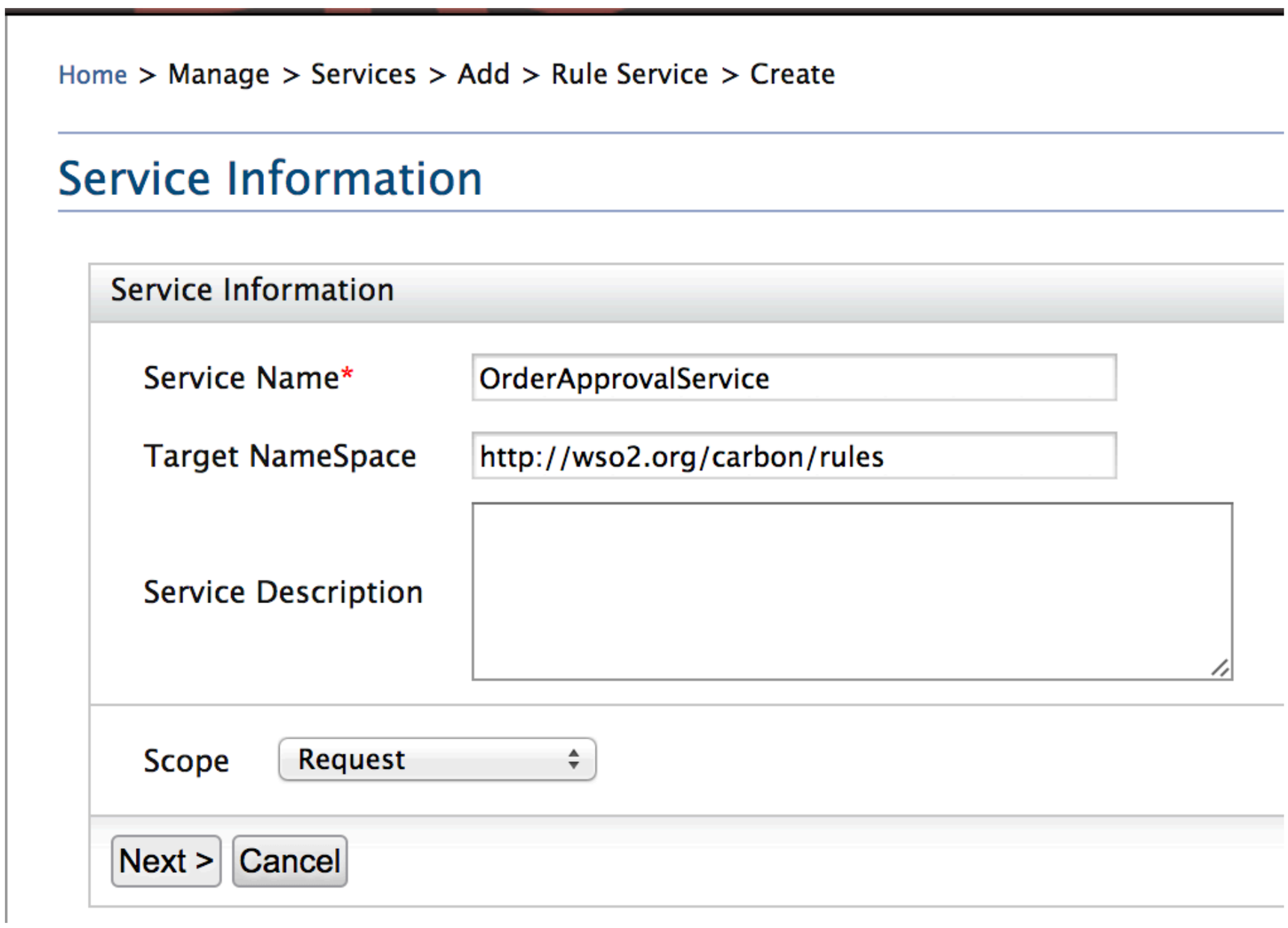
```
package samples.userguide;

/**
* Order accept notification
*/
public class OrderAccept {

    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

```
package samples.userguide;

/**
* Order Reject Notification
*/
public class OrderReject {

    private String reason;

    public String getReason() {
        return reason;
    }

    public void setReason(String reason) {
        this.reason = reason;
    }
}
```

Una vez tenemos esto ya podemos exponer la regla como servicio. Le pasaremos peticiones o facts, y nos responderá con los resultados que hemos definido. Para ello entramos en la url donde se nos ha desplegado la herramienta.



Hacemos click en create, y le damos un nombre al servicio y pulsamos en next



A continuación tenemos una serie de opciones para indicar nuestra regla. Mediante texto, mediante **WSO2 Governance Registry**, mediante url... La opción que me parece más interesante es mediante **Governance Registry**, pero por ahora simplemente vamos a copiar la regla en modo texto y pinchamos en siguiente.

Home > Manage > Services > Add > Rule Service > Create > Rule Set Information

Rule Set Information

Rule Set Information

Rule Script As* (•) Inlined ( ) Registry Key ( ) Rule File ( ) URL

Rule Script

```
29 insertLogical(orderReject);
30 end
31
32 rule "Company B Order Deny Rule" dialect "mvel" no-loop true salience 2
33 when
34 not ( OrderAccept())
35 $placeOrder : PlaceOrder( symbol == "Company B" )
36 then
37 retract($placeOrder);
38 OrderReject orderReject = new OrderReject();
39 orderReject.setReason("An Order for stocks of Company B is accepted only if
40 insertLogical(orderReject);
41 end
42
43 rule "Company C Order Deny Rule" dialect "mvel" no-loop true salience 1
44 when
45 not ( OrderAccept())
46 $placeOrder : PlaceOrder( symbol == "Company C" )
47 then
48 retract($placeOrder);
49 OrderReject orderReject = new OrderReject();
50 orderReject.setReason("An Order for stocks of Company C is accepted only if
51 insertLogical(orderReject);
52 end
```

Position: Ln 52, Ch 4 Total: Ln 52, Ch 1735

[x] Toggle editor

Rule Type: Regular

< Back | Next > | Cancel

A continuación se nos pide que introduzcamos un jar con las clases java que va a usar la regla. No solo la entrada sino también cualquier tipo de salida. Son las clases que hemos visto antes. Para que vayamos rápido, yo me he generado un jar con maven con las tres clases. Os lo podéis bajar aquí mismo. Subimos ese fichero, nos confirman que ha ido todo bien y pinchamos en siguiente. Fijaros que abajo nos salen el número de facts que tenía nuestro jar, que son 3. Con esto podemos ver si hemos subido todo lo que queremos o no.

Home > Manage > Services > Add > Rule Service > Create > Rule Set Information > Facts Upload

Facts Upload

Facts Upload (.jar)

Path To Facts Archive (.jar) : Seleccionar archivo Ningún archivo seleccionado Upload

placeorder-1.0.jar Delete

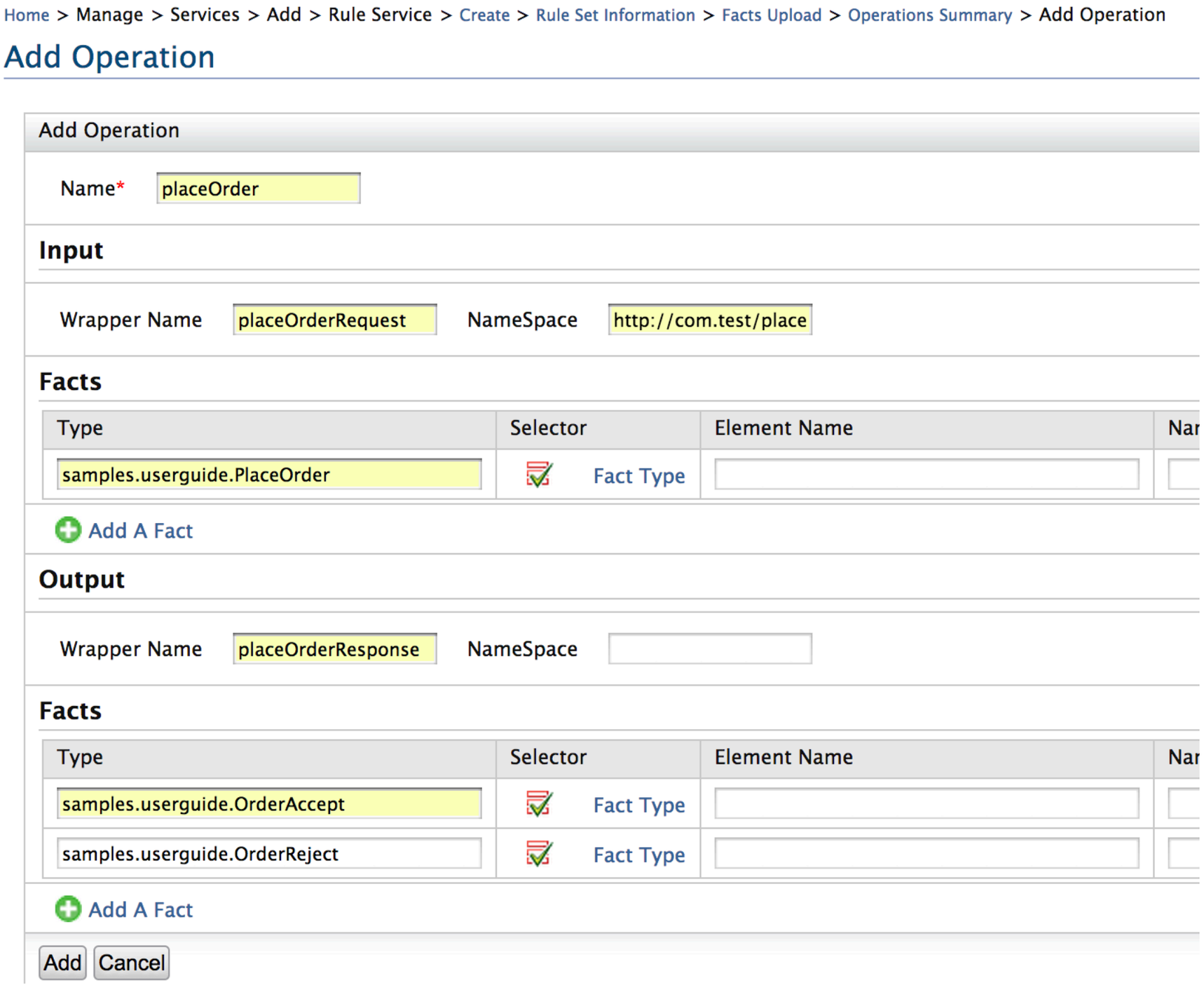
Number of Available Facts: WSO2 Carbon

< Back | Next > | Cancel

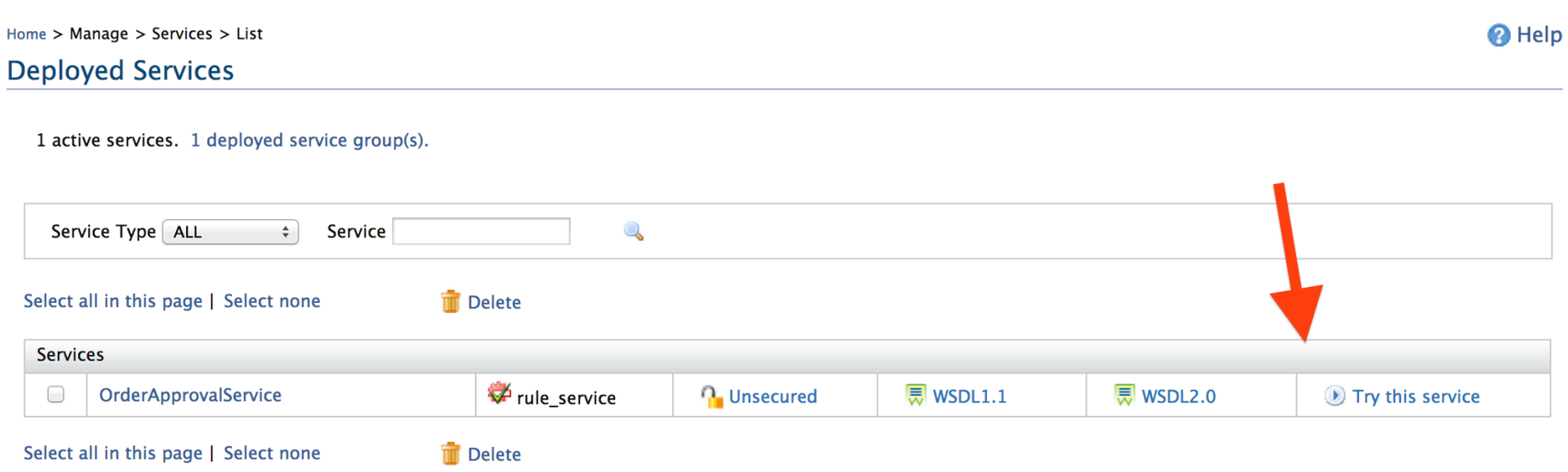
Facts file uploaded successfully.

OK

En el siguiente paso solo nos queda configurar que clases son facts de entrada y cuales son de salida. Lo hacemos como en la imagen. Cuando lo tengamos todo listo pulsamos en Add y Finish, y ya tenemos nuestro servicio publicado.



Para comprobar que lo hemos hecho bien, nos vamos en el panel de administración de la izquierda a Services >> List. Deberíamos ver nuestro servicio en la lista.



Si pulsamos en Try this service podemos generar un fact de entrada y comprobar que el servicio responde. Lo haríamos así.

OrderApprovalService

Using Endpoint - OrderApprovalServiceHttpSoap12Endpoint

Try an alternate **https** Hide

placeOrder

Send

Request

```
1  <body>
2     <p:placeOrderRequest xmlns:p="http://com.test/placeOrder">
3        <!--0 or more occurrences-->
4        <ax2372:placeOrder xmlns:ax2372="http://com.test/placeOrder">
5           <!--0 to 1 occurrence-->
6           <xs:price xmlns:xs="http://userguide.samples">150</xs:price>
7           <!--0 to 1 occurrence-->
8           <xs:quantity xmlns:xs="http://userguide.samples">15</xs:quantity>
9           <!--0 to 1 occurrence-->
10          <xs:symbol xmlns:xs="http://userguide.samples">Company A</xs:symbol>
11       </ax2372:placeOrder>
12    </p:placeOrderRequest>
13 </body>
14
```

Position: Ln 6, Ch 59 | Total: Ln 13, Ch 545

Response

```
1 <axis2ns3:placeOrderResponse xmlns:axis2ns3="http://wso2.org/carbon/rules">
2    <orderAccept xmlns="http://userguide.samples">
3       <message>Accepted order for: 15 stocks of Company A at$ 150.0</message>
4    </orderAccept>
5 </axis2ns3:placeOrderResponse>
```

Como el stock es mayor que 10 y la Compañia es la A, nos devuelve un orderAccepted

Vamos a ver ahora como podríamos llamar a nuestro servicio desde fuera. Podeis usar cualquier cliente para web services.
Yo lo voy a hacer con un plugin de chrome(**Postman**)

http://192.168. :9 3/services/OrderApprovalService.OrderApprovalServiceHttpEndpoint/placeOrder POST

Content-Type text/xml Add preset Manage presets

Header Value

form-data x-www-form-urlencoded raw binary XML (text/xml)

```
1  <p:placeOrderRequest xmlns:p="http://com.test/placeOrder">
2        <!--0 or more occurrences-->
3        <ax2369:placeOrder xmlns:ax2369="http://com.test/placeOrder">
4           <!--0 to 1 occurrence-->
5           <xs:price xmlns:xs="http://userguide.samples">100</xs:price>
6           <!--0 to 1 occurrence-->
7           <xs:quantity xmlns:xs="http://userguide.samples">15</xs:quantity>
8           <!--0 to 1 occurrence-->
9           <xs:symbol xmlns:xs="http://userguide.samples">Company A</xs:symbol>
10       </ax2369:placeOrder>
11    </p:placeOrderRequest>
```

Send Preview Pre-request script Tests Add to collection

Body Cookies (8) Headers (4) Tests STATUS 200 OK TIME 124 ms

Pretty Raw Preview XML

```
<axis2ns4:placeOrderResponse xmlns:axis2ns4="http://wso2.org/carbon/rules">
  <orderAccept xmlns="http://userguide.samples">
    <message> Accepted order for: 15 stocks of Company A at$ 100.0 </message>
  </orderAccept>
</axis2ns4:placeOrderResponse>
```

OUTLINE

▼ axis2ns4:

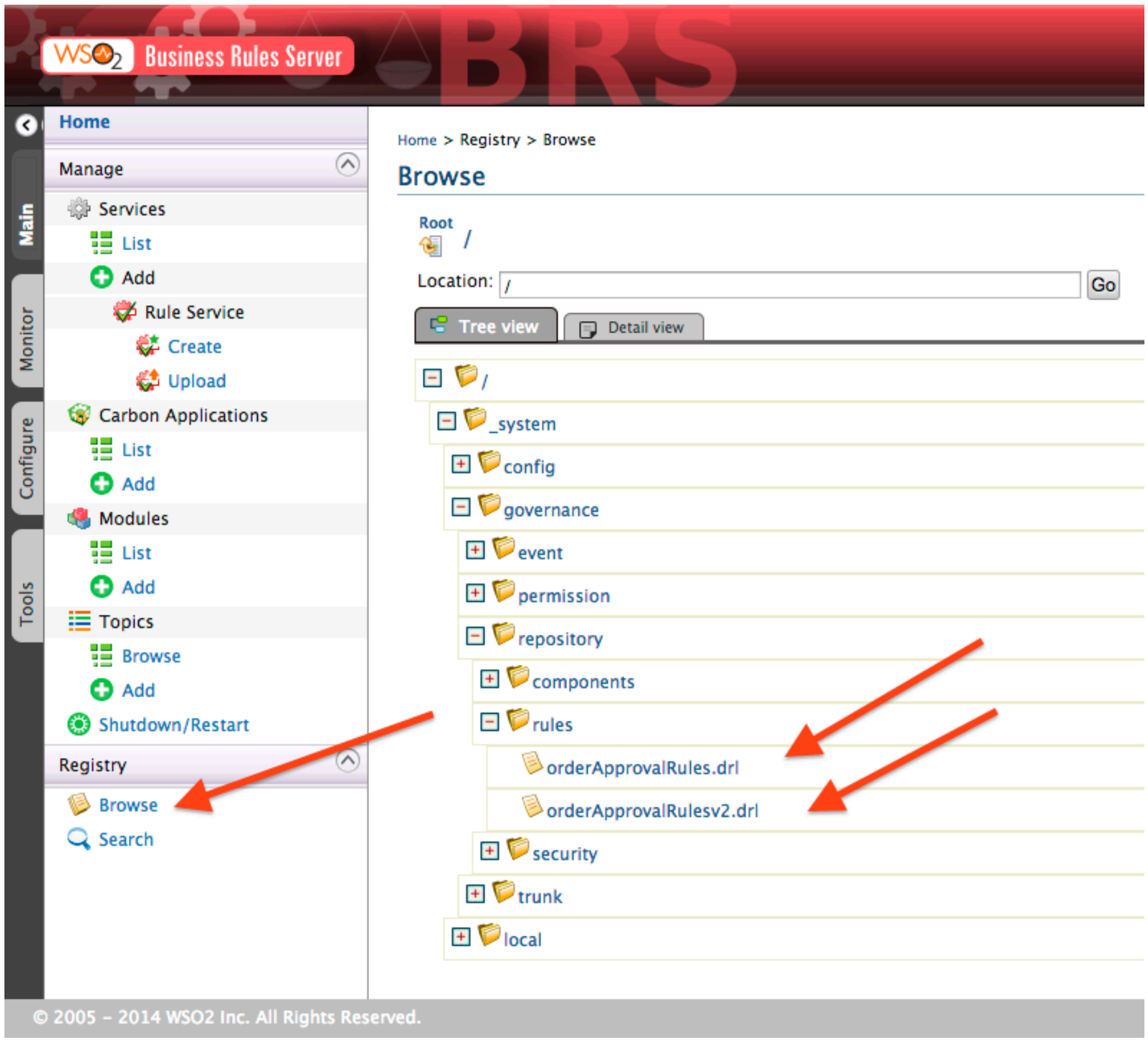
▼ orderA

Es tan sencillo como hacer un **post** al endpoint de nuestro servicio copiando la request que teníamos cuando hemos probado el servicio y añadiendo la cabecera de que vamos a enviar un xml. Como véis la respuesta es la misma que obteniamos desde wso2.

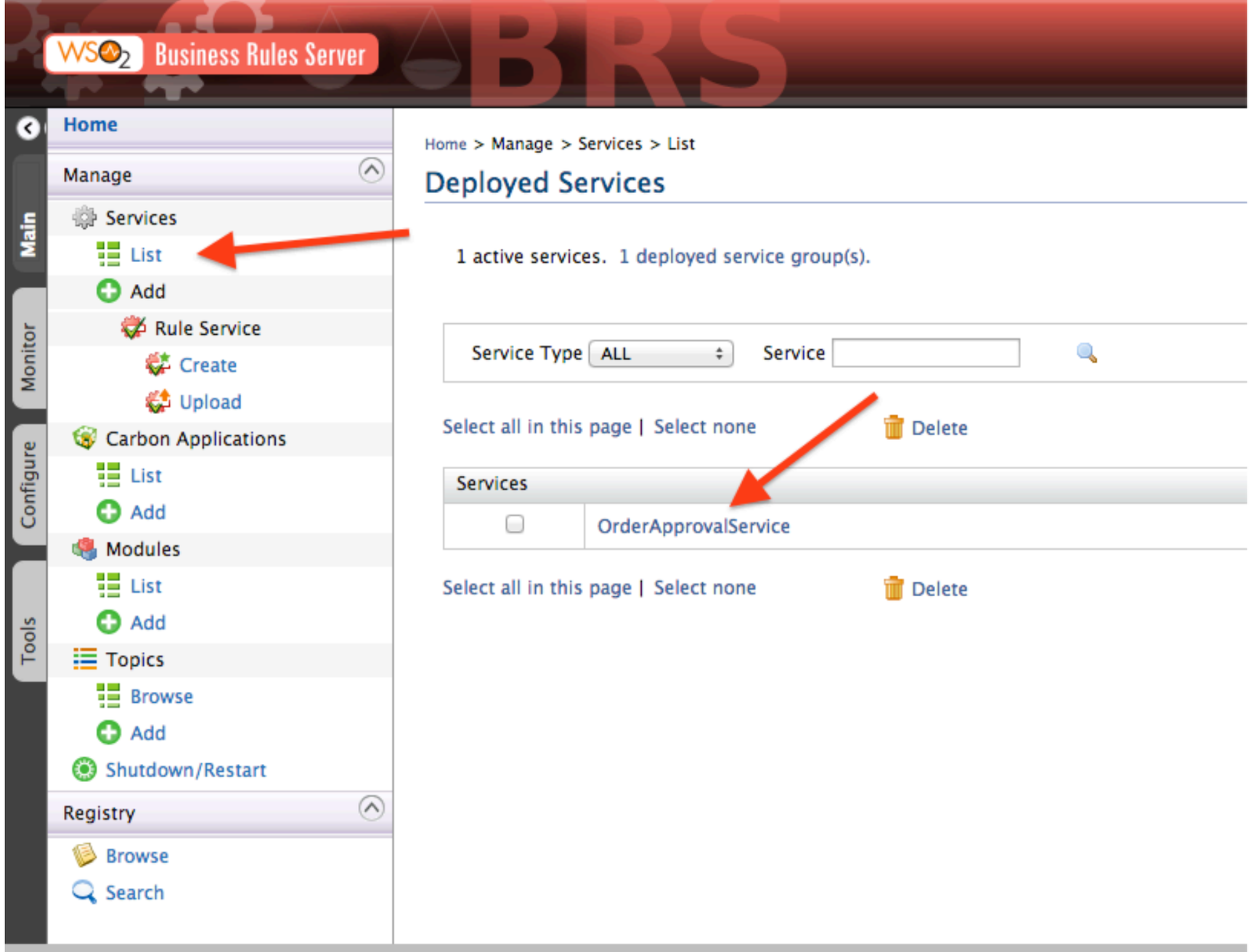
5. Repositorio de reglas integrado con WSO2 Governance Registry

Hay cierta parte de **WSO2 Governance Registry** (de ahora en adelante GR) que ya viene integrado por defecto en **WSO2 Business Rules Server** (de ahora en adelante BSR). Si nos descargamos GR vemos que en realidad tenemos muchas más opciones de las que vienen dentro de BRS. Por lo que he podido averiguar por internet, parece ser que tú te bajas un .zip con un producto de la suite de **WSO2**, pero si quieres integrar ciertos productos con otros lo tienes que hacer añadiendo features a la instalación que te bajas mediante el wizard de añadir features. He intentado sin éxito arrancar a la par estos dos productos y aun cambiando los puertos no he sido capaz. Lo que si que he podido comprobar es que cada uno tenía su propio repositorio, es decir, que no es compartido. Desde BRS no ves las reglas que tienes en el repositorio de GR, y esto me hace pensar que el filesystem virtual que se crea para almacenar los recursos es relativo a la instalación. No he visto forma de cambiar el fylesystem a una ruta alternativa, y no se si se puede.

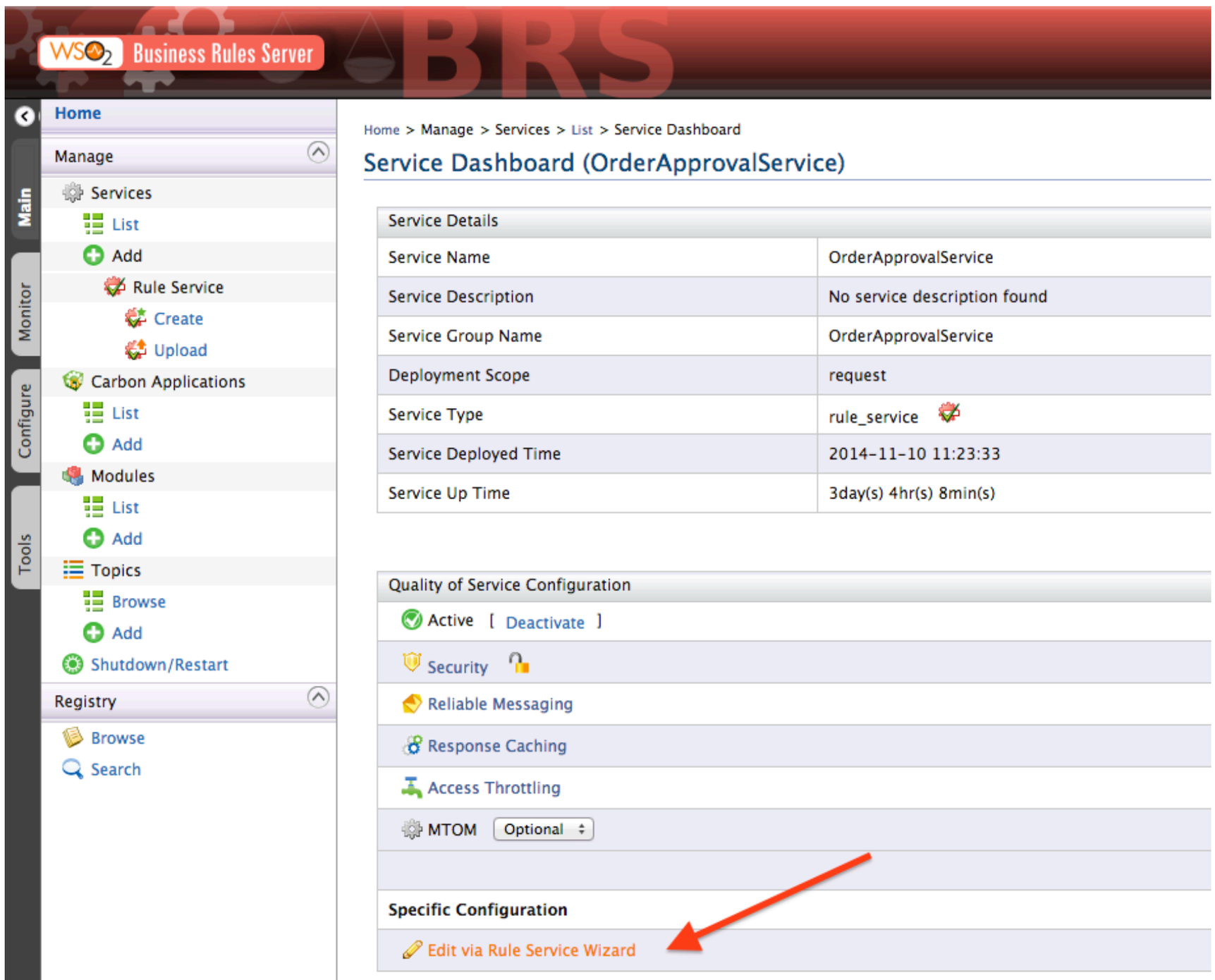
Para almacenar las reglas en el repositorio de BRS, nos podemos crear una carpeta rules y metemos dos reglas. La primera es la que tenemos en el ejemplo de arriba, y la segunda es una simple modificación donde modificamos un poco la regla para que en la Company A, el quantity en vez de ser mayor que 10, ponemos mayor que 15. Así vamos a tener dos reglas y veremos como podemos borrar una, meter la otra y el servicio responde en consonancia.



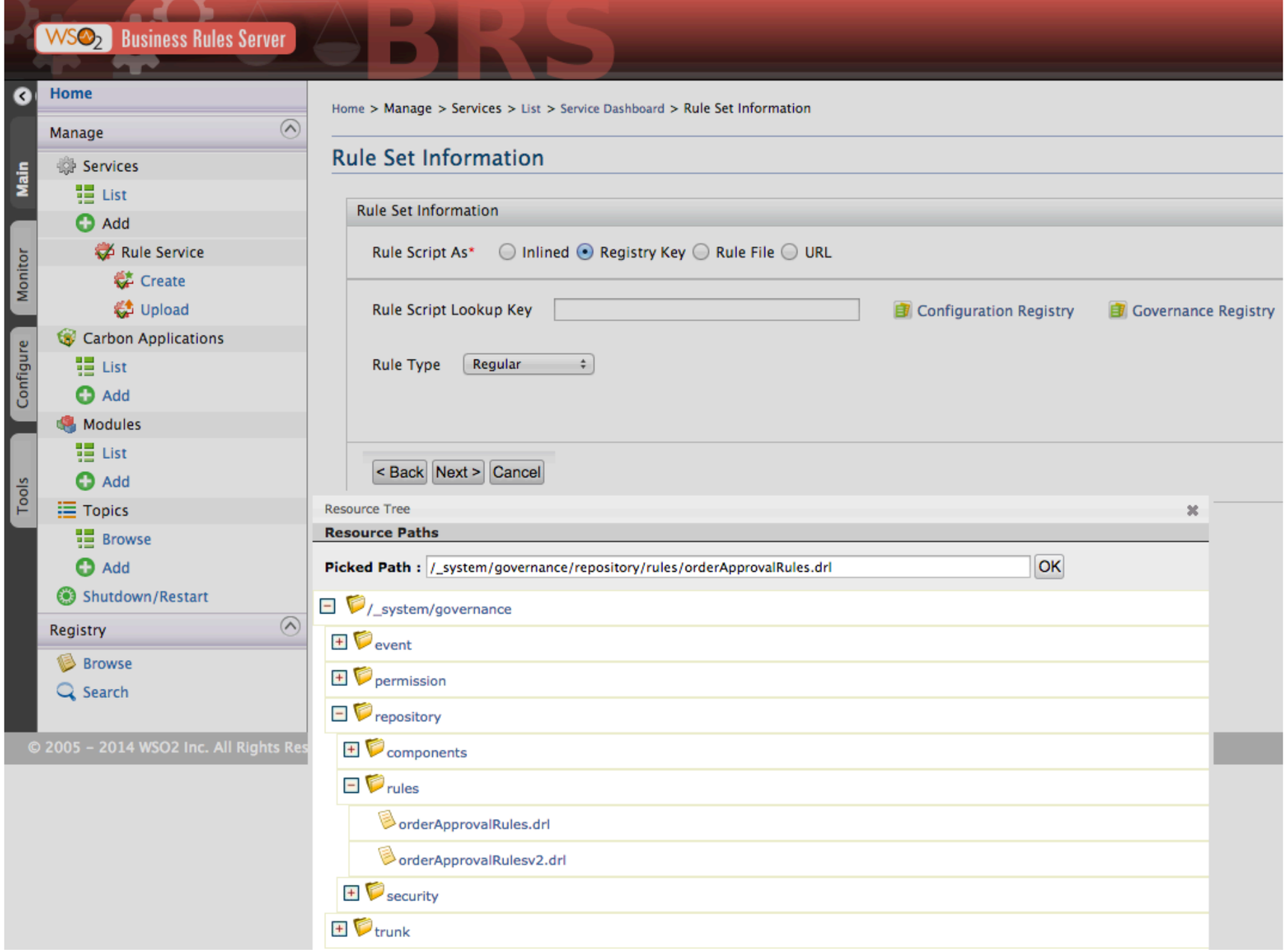
Ahora para aplicar la regla a nuestro servicio solo tenemos que ir al listado de servicios y pinchar en el nuestro, como se ve en la imagen siguiente.



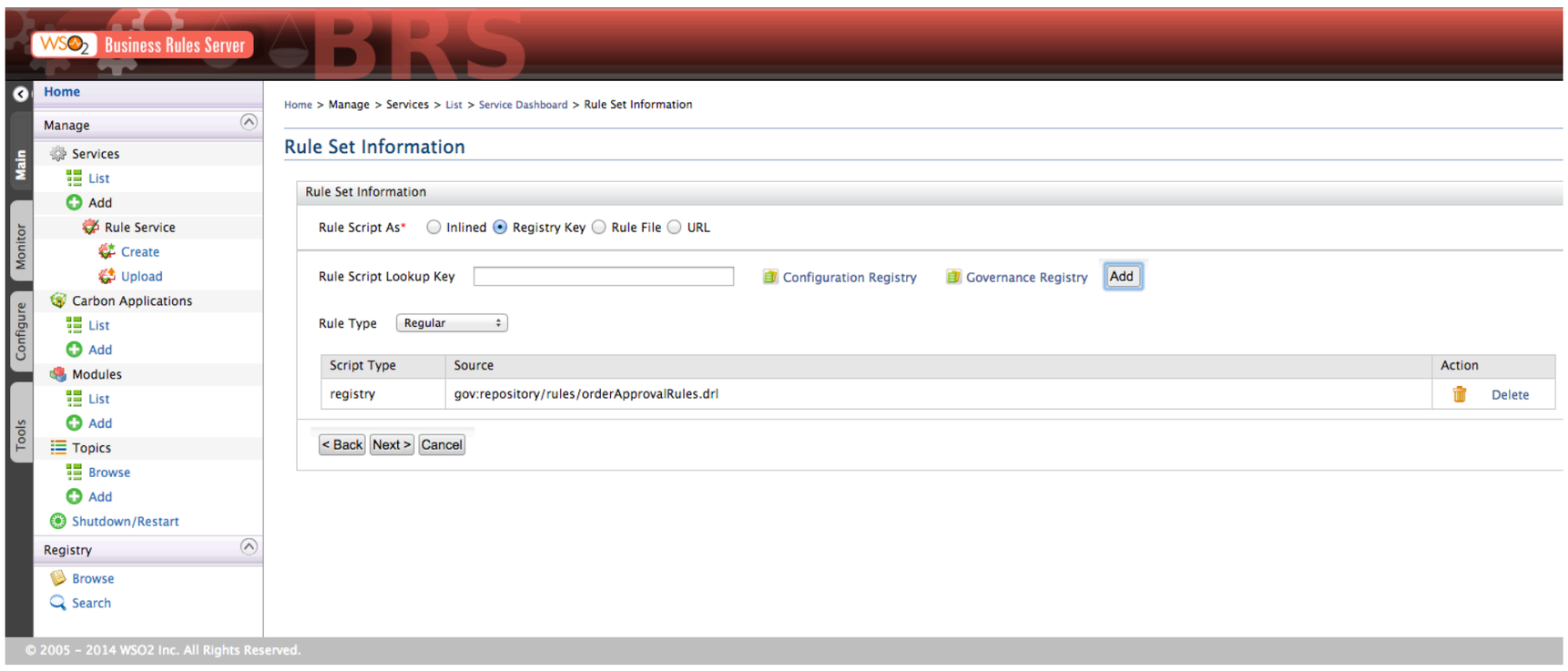
Pinchamos en el wizard de edición de servicio.



Pinchamos en siguiente, hasta que llegamos a la parte de definición de reglas, y pulsamos en Governance Registry. En la ventana emergente buscamos nuestra regla, pulsamos en OK, y después en Add.



Si todo va bien tendremos la regla seleccionada del repositorio, pulsamos en Next y finish. Podemos volver a probar el servicio con Try-it o con Postman y comprobar como sigue respondiendo.



La siguiente prueba que podemos hacer es cambiar la regla a la v2. Vemos como es prácticamente instantáneo el cambio y al llamar a nuestro servicio, solo nos responderá correctamente si la cantidad es mayor que 15 (recordad que cambiamos justamente esto). Esta parte me ha gustado bastante, porque no hace falta tirar el servidor, ni reiniciar nada. Simplemente cambiando la regla cambia el servicio.

6. Conclusiones

La documentación de la web está muy bien explicada, aunque es cierto que hay bastantes cosas que no se detallan. Me ha parecido una documentación un tanto escasa. Esto es debido al modelo de negocio de **WSO2**, en donde lo que te venden es el soporte. Por ejemplo yo desconozco donde se guarda el fichero de reglas dentro del repositorio, y muchos detalles que no se explican en la documentación. Esto me crea cierta incertidumbre. Como bien dicen ellos en su propia página o tienes tiempo para bucear en el código fuente y ver el detalle de las cosas, o tienes dinero y contratas soporte para que te lo den masticado. Con respecto a la comunidad, te referencian a stackoverflow. Una vez vas a stackoverflow ves que muchas preguntas están sin respuesta, por lo que la comunidad por ahora no está tan desarrollada como la de otros productos. Un saludo y espero que os haya resultado interesante.

7. Información sobre el autor

Alberto Barranco Ramón es Ingeniero Técnico en Informática de Gestión y Graduado en Ingeniería del Software por la Universidad Politécnica de Madrid

Mail: abarranco@autentia.com.

Twitter: @barrancoalberto

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo".

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» **Registrate** y accede a esta y otras ventajas «