## Generación Esqueleto Servicio Web

En esta guía veremos cómo crear un las clases que implementan y generan el esqueleto de un servicio Web a partir de su WSDL (Web Service Definition Language).

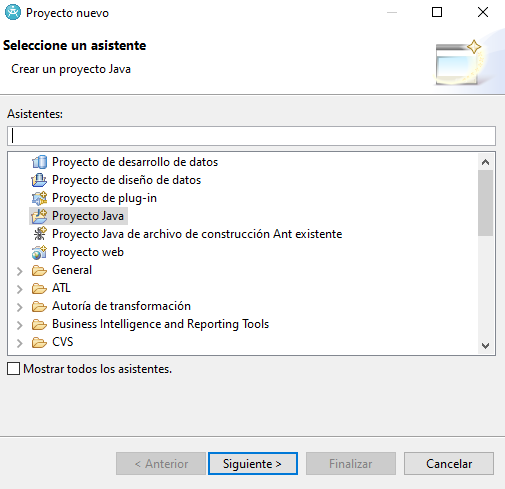
Para automatizar la creación de las clases del servicio, se utilizará Eclipse RSA, aunque también se podría realizar mediante el comando java ‘wsdl2java’.

## 0.-Incluir el fichero WSDL en un nuevo proyecto Java

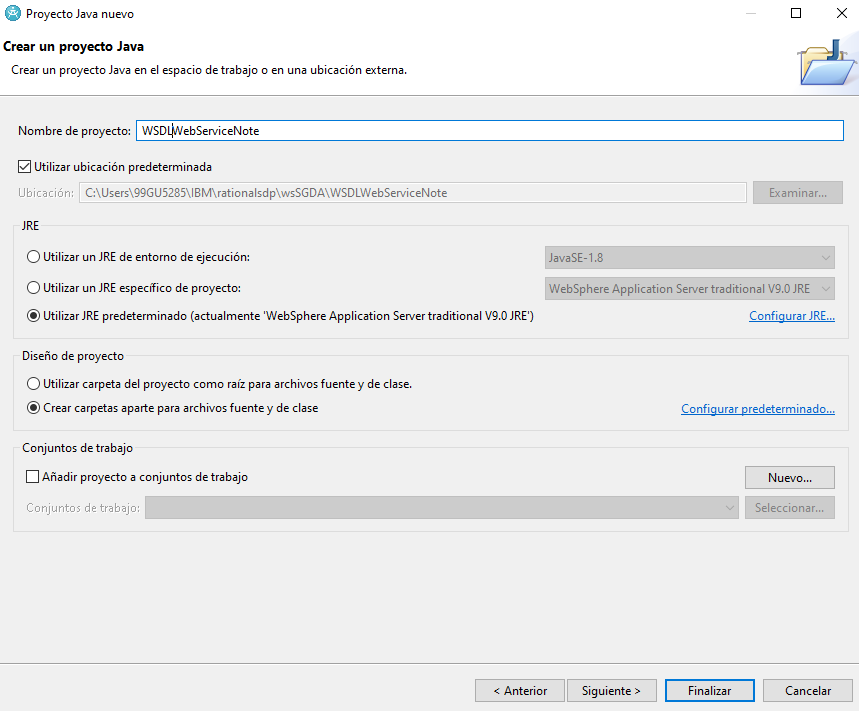
Para que RSA tenga visibilidad sobre el fichero WSDL que implementa el servicio, es necesario copiarlo en un nuevo proyecto de RSA. Esto será necesario para seleccionar el fichero en el paso 2. Es decir, este proyecto es únicamente para almacenar el WSDL del servicio, pero no tendrá ninguna utilizad una vez generado el esqueleto del mismo.

En RSA, Archivo > Nuevo > Proyecto

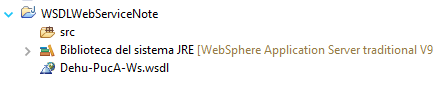
Y seleccionamos la opción ‘Proyecto Java’



Se dejan las opciones por defecto.



En este nuevo proyecto, se copia el WSDL del servicio web a implementar. Hay que cambiar la extensión a wsdl para que RSA localice el wsdl en el paso 2 de la guía.

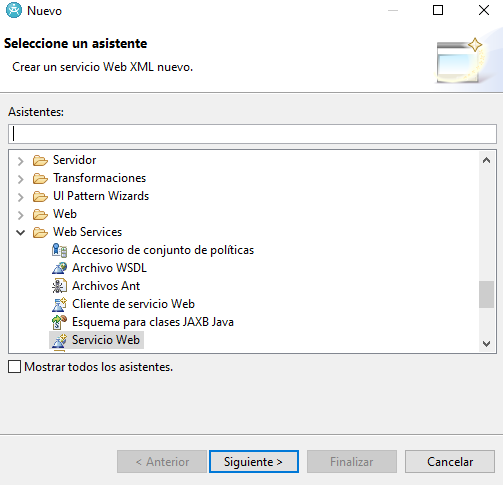


## 1.-Crear Nuevo Proyecto Servicio Web

Pasamos a crear el servicio web que implementará las operaciones del servicio definido en el WSDL.

En RSA*, ‘Archivo > Nuevo > Otras…*’

Y seleccionamos la opción Servicio Web bajo la carpeta Web Services.

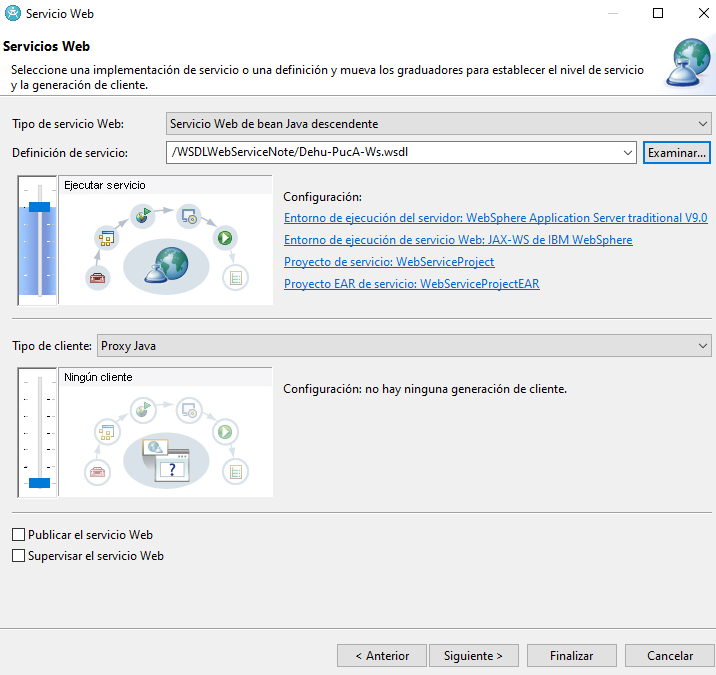
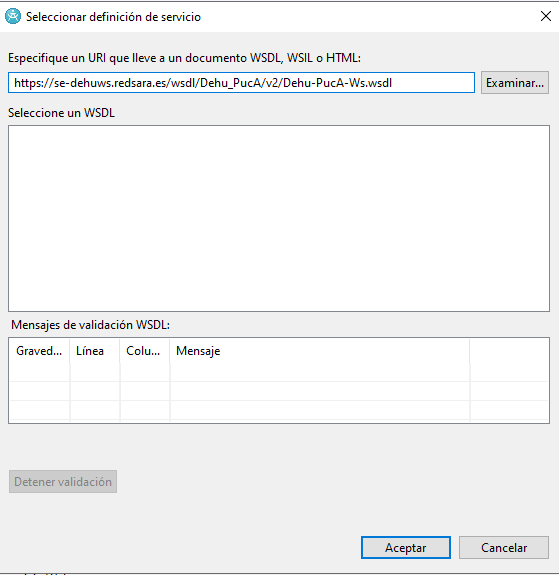


## 2.-Seleccionar WSDL del Servicio Web

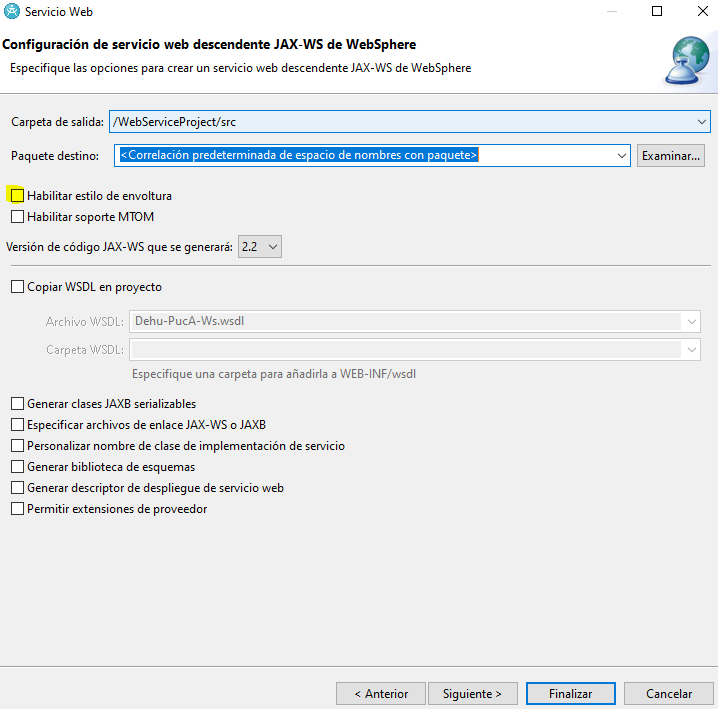
Aparecerá una nueva pantalla donde hay que seleccionar el WSDL con la definición del servicio a implementar. Para ello, se seleccionan las siguientes opciones:

**Tipo de Servicio Web:** Servicio Web de bean Java descendente

**Definición de servicio:** Se selecciona el fichero wsdl del servicio web a implementar. Es necesario crearse un proyecto previo donde incluir el WSDL para que RSA pueda ver en este paso el fichero WSDL (ver paso 0).

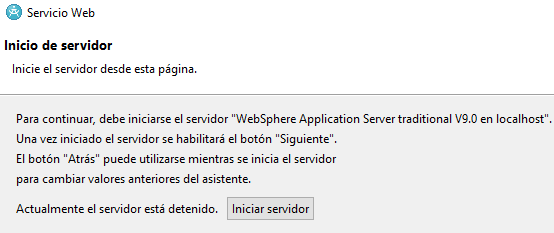


En la siguiente pantalla se dejan los valores por defecto excepto en la opción ‘Habilitar Estilo envoltura’ que se desactivará, aunque si el proyecto requiere opciones específicas, se podrían modificar las opciones que apliquen.



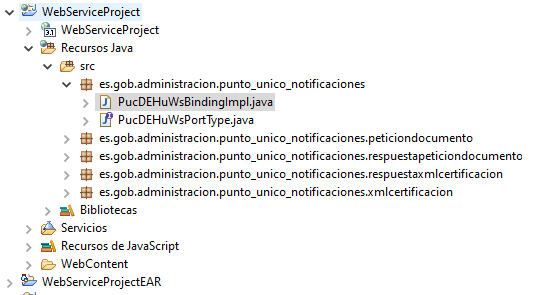
Pulsamos el botón ‘**Siguiente’**

RSA no pedirá iniciar el Servidor Websphere para completar la generación de las clases que implementarán el servicio web. Iniciamos el servidor y finalmente, cuando este acabe de arranca pulsamos el botón ‘Siguiente’ y ‘Finalizar’ para acabar el proceso de generación.



## 3.-Comprobación Clases Generadas

Una vez RSA finaliza el proceso de generación de las clases, es importante comprobar que se han creado en el proyecto ‘WebServiceProject’ las clases autogeneradas.



Además, RSA habrá creado un proyecto EAR WebServiceProjectEAR que será el que se utilice Websphere para desplegar el servicio Web.

Los nombres estos proyectos pueden cambiarse sin ningún problema si fuese necesario, así como el contexto donde escuchará el servicio web.

## 4.-Comprobar Endpoint y WSDL Servicio

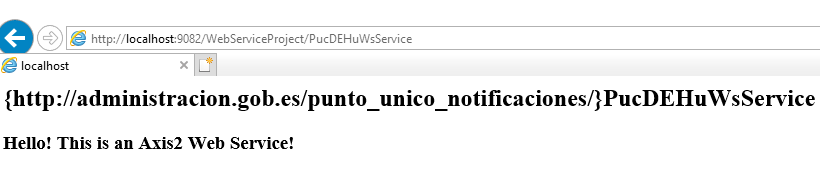
RSA, automáticamente, habrá desplegado el ear generado (WebServiceProjectEAR) en el servidor Websphere. Para comprobar que el despligue se ha realizado de forma correcta, se accederán a las siguientes URLS:

**-Endpoint del servicio:**

<http://localhost:9082/WebServiceProject/PucDEHuWsService>

Dependiendo de vuestro perfil de Websphere, el puerto puede variar (9081, 9082, 9083,9084…).

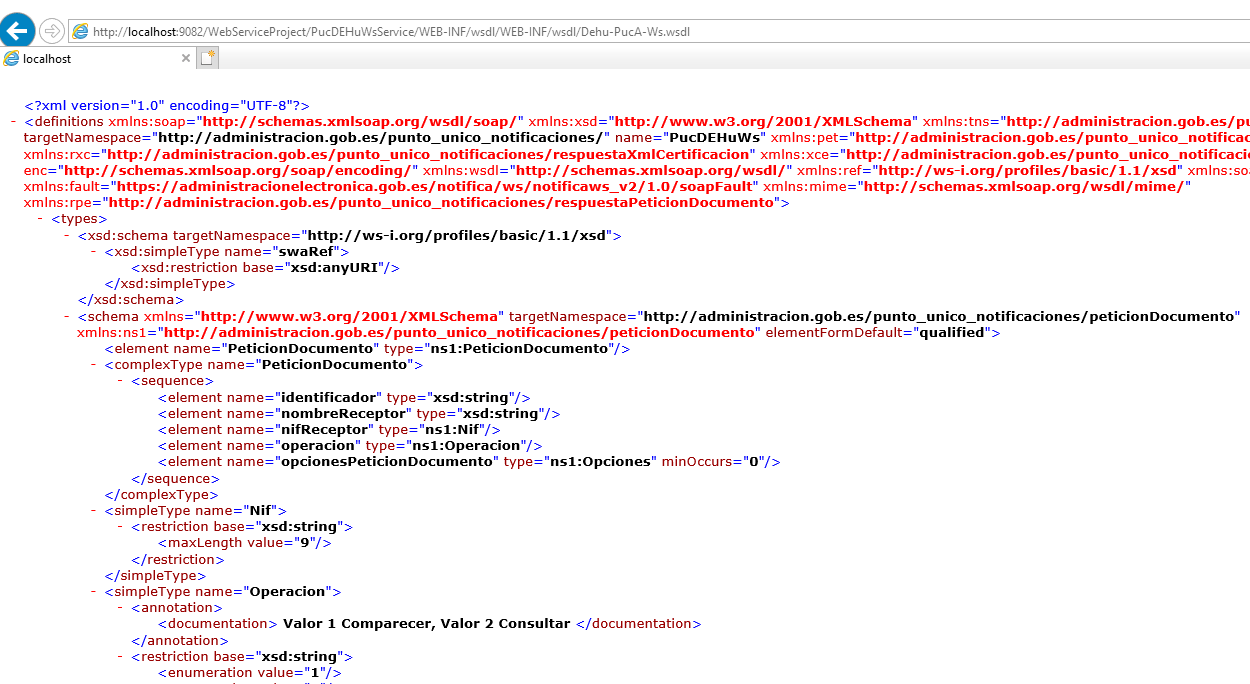
Accediendo a esta dirección en un navegador, se debería mostrar un mensaje indicando que el servicio web está escuchando.



**-WSDL del servicio:**

<http://localhost:9082/WebServiceProject/PucDEHuWsService?wsdl>

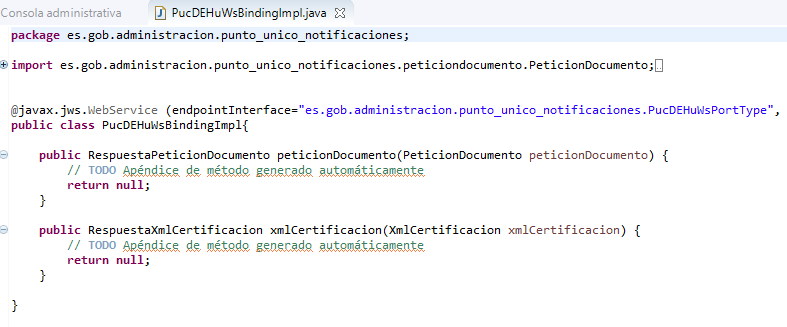
Al igual que en el caso del endpoint, dependiendo de vuestro perfil de Websphere, el puerto puede variar (9081, 9082, 9083,9084…).



## 5.-Métodos a implementar

La clase autogenerada que habrá que implementar con la lógica de negocio es la clase PucDEHuWsBindingImpl.

En ella están los 2 métodos que tiene el servicio web, con sus respectivos parámetros de entrada y salida.



## 6.-Probar el servicio y comprobar que llegan las peticiones

Para probar que el esqueleto de los métodos es correcto y que llegan los parámetros de entrada y se genera correctamente la respuesta, conviene antes de empezar con la lógica de negocio hacer un ‘dummy’ que simplemente imprima en consola los parámetros de entrada y genere una respuesta por defecto en la clase PucDEHuWsBindingImpl.

**public** RespuestaPeticionDocumento peticionDocumento(PeticionDocumento peticionDocumento) {

System.***out***.println("LLEGA PETICION");

System.***out***.println("Identificador " + peticionDocumento.getIdentificador());

System.***out***.println("NIf Receptor " + peticionDocumento.getNifReceptor());

System.***out***.println("Nombre Receptor " + peticionDocumento.getNombreReceptor());

System.***out***.println("Operacion " + peticionDocumento.getOperacion());

// Respuesta dummy

RespuestaPeticionDocumento resp =**new** RespuestaPeticionDocumento();

resp.setCodigoRespuesta("0");

resp.setDescripcionRespuesta("Operacion Dummy realizada correctamente");

resp.setIdentificador(peticionDocumento.getIdentificador());

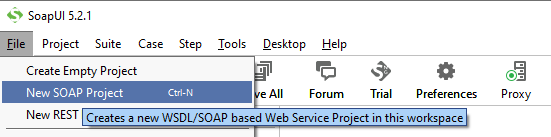
**return** resp;

}

E invocar al servicio desde SOAPUi, JUnit, JMeter o cualquier otro procesador de peticiones SOAP. Por ejemplo, con SOAPUi:

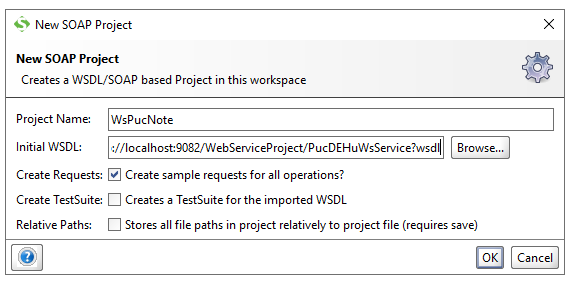
1. **Crear nuevo proyecto**

*‘File > New SOAP Project’*

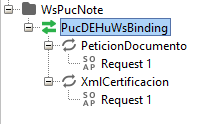


1. **Indicar el WSDL del servicio desplegado en Websphere**

Indicar la URL del WSDL desplegado en Websphere y pulsar el botón ‘OK’

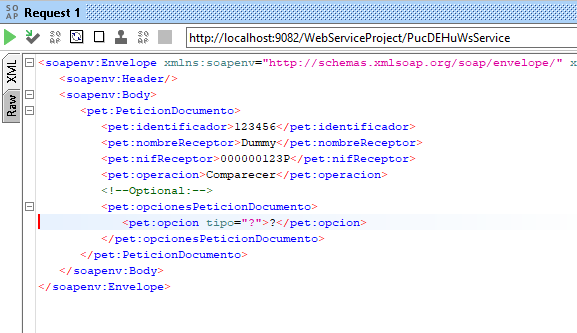


1. **Se generan las peticiones SOAP de las operaciones del servicio web**

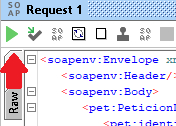


1. **Acceder a la petición de la operación PeticionDocumento**

Abrir la petición SOAP de la primera operación y rellenar los elementos de la petición, con valores de prueba.



Pulsar el botón verde de ‘Submit’ para enviar la petición.



1. **Comprobar la respuesta**

Comprobar que la respuesta coincide con los valores definidos en el dummy y que en la consola de Websphere se muestran los valores e la petición correctamente.

