SERVICIOS SOFTWARE

Servicios Web con Apache Axis 2

Universidad de Cantabria – Facultad de Ciencias

Patricia López Martínez

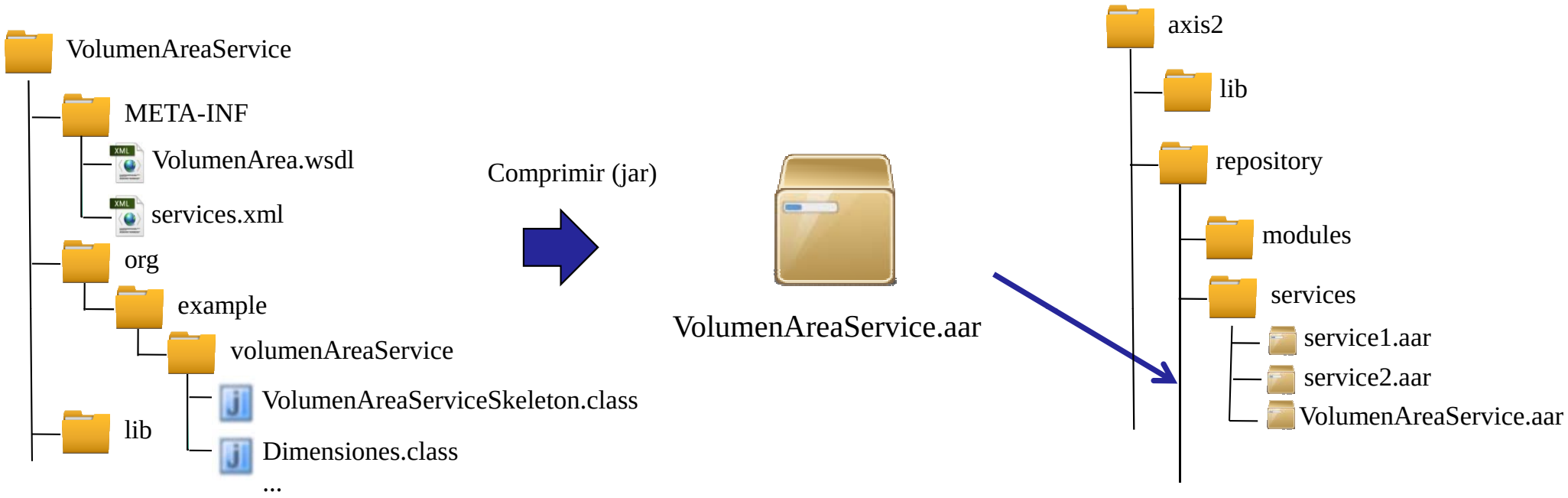
Apache Axis2

- Axis2 es un motor para desarrollo de Servicios Web y Servicios REST
  - Desarrollado por Apache Software Foundation
  - Versiones para Java y C => Versión Java: http://axis.apache.org/axis2/java/core
  - Arquitectura extensible gracias a la inclusión de Modules
- Axis2 soporta WSDL 1.1, WSDL 2.0, SOAP 1.1, SOAP 1.2 y REST
  - Soporta también algunas de las especificaciones WS-*
- Dos posibilidades para el despliegue de servicios:
  - Servidor Standalone => Proporciona su propio contenedor para testing
  - Como aplicación web desplegada en un servidor web externo, p.e. Tomcat
- Define un modelo propio de empaquetamiento para los servicios
  - Un servicio web Axis2 se distribuye como un fichero comprimido con extensión .aar
- Ofrece capacidad de “hot deployment” y “hot update”

Infraestructura de despliegue Axis2

Despliegue basado en archivo

- Proceso similar a Java EE
- Modelo de empaquetamiento propio => Archivos .aar
  - services.xml => Descriptor de despliegue
- Para desplegar se copia el archivo .aar en el directorio correspondiente
  - Version standalone => Directorio `repository\services` de la instalación de Axis2
  - Sobre Tomcat => Directorio `webapps\axis2\WEB-INF\services`



Descriptor services.xml

- Define el servicio y lo enlaza con la clase Java que lo implementa

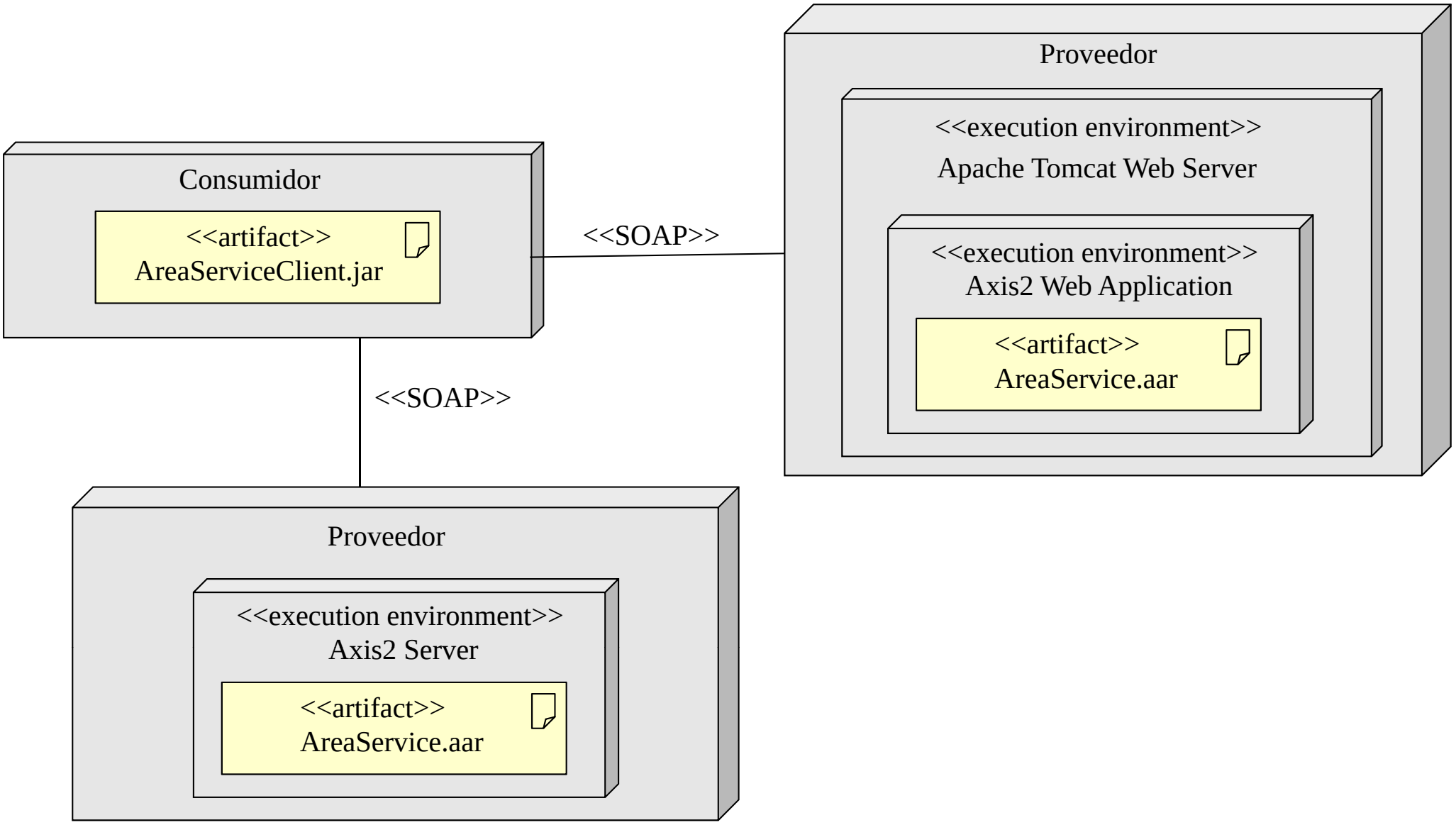
services.xml *

```
<serviceGroup>
    <service name="VolumenArea">
        <messageReceivers>
            <messageReceiver mep=http://www.w3.org/ns/wsdl/in-out
                             class="es.unican.is2.volumenArea.VolumenAreaMessageReceiverInOut"/>
        </messageReceivers>
        <parameter name="ServiceClass">es.unican.is2.volumenArea.VolumenAreaSkeleton</parameter>
        <parameter name="useOriginalwsdl">true</parameter>
        <parameter name="modifyUserWSDLPortAddress">true</parameter>

        <operation name="area" mep="http://www.w3.org/ns/wsdl/in-out" namespace="http://www.unican.es/ss/VolumenArea/">
            <actionMapping>http://www.unican.es/ss/example.org/VolumenArea/area</actionMapping>
            <outputActionMapping>http://www.unican.es/ss/example.org/VolumenArea/VolumenAreaInterface/areaResponse
             </outputActionMapping>
            <faultActionMapping faultName="DatosNoValidosException">http://www.unican.es/ss/example.org/VolumenArea
             </faultActionMapping>
            <faultActionMapping faultName="DatosNoValidosExceptionMsg">
                        http://www.unican.es/ss/example.org/VolumenArea/VolumenAreaInterface/area/Fault/DatosNoValidosException
             </faultActionMapping>
            <faultActionMapping faultName="DatosNoValidosExceptionMsg_Exception">
                        http://www.unican.es/ss/example.org/VolumenArea/VolumenAreaInterface/area/Fault/DatosNoValidosException
             </faultActionMapping>
        </operation>
     </service>
</serviceGroup>
```

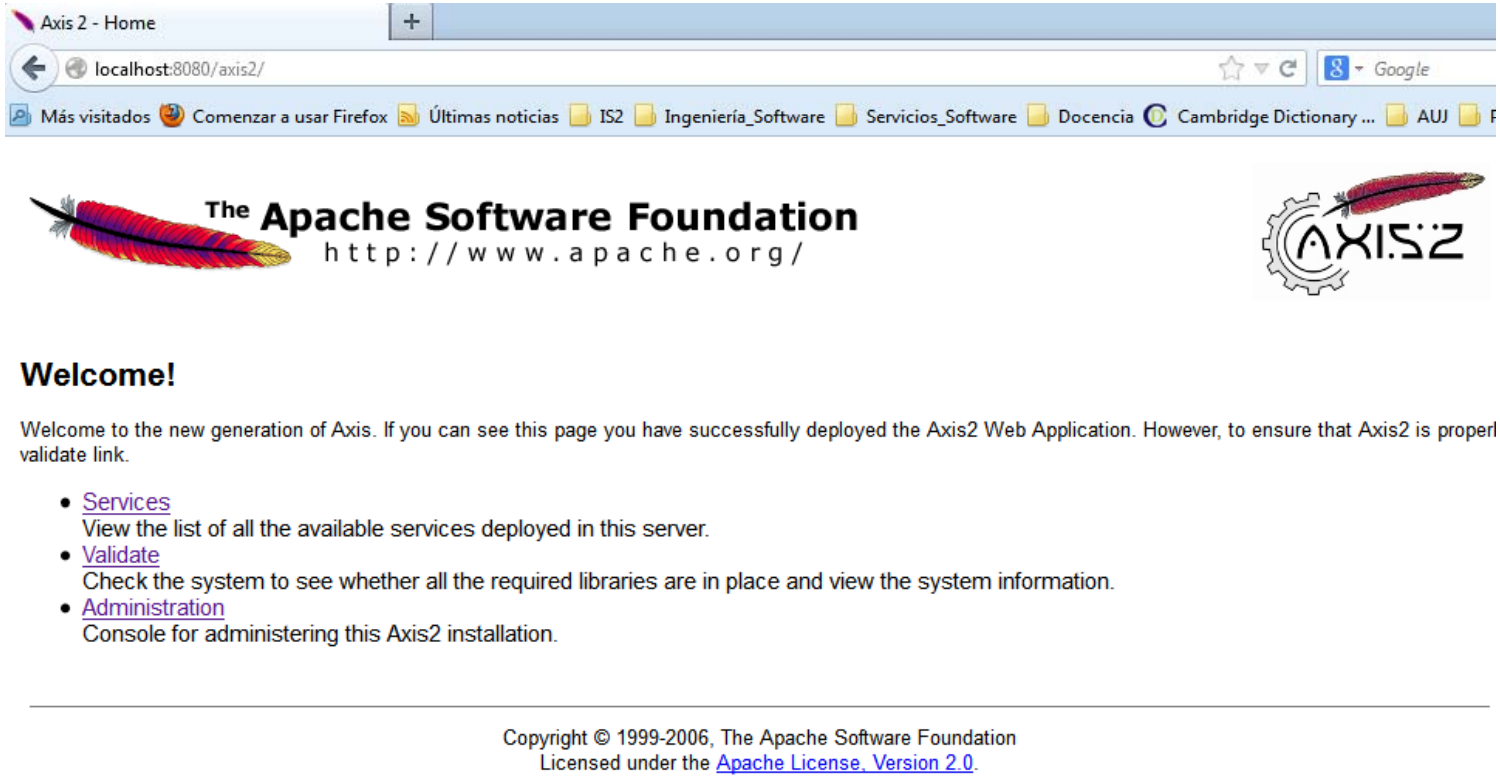
* Correspondiente al ejemplo de la transparencia 8

Infraestructura de despliegue Axis2 (cont.)



AXIS2 sobre Tomcat

- Copiar `axis2.war` en `%CATALINA_HOME%\webapps`
- Arrancar Tomcat (`%CATALINA_HOME%\bin\startup.bat`)
- Acceder a http://localhost:8080/axis2
- Para añadir servicios:
  - Desde la propia interfaz web
  - Copiando el fichero .aar en `%CATALINA_HOME%\webapps\axis2\web-inf\services`



Comandos básicos de Axis2

- **axis2server** => Arranca el servidor Axis2 en modo standalone
  - Una vez arrancado, disponible en `http://localhost:8080/axis2/services`
  - Por defecto usa el puerto 8080 pero se puede modificar en axis2.ini

- **wsdl2java** => Generación de código Java desde la descripción WSDL
  - Invocación:
    - `%AXIS2_HOME%\bin\wsdl2java.bat [options] –uri <Fichero_wsdl>`
  - Por defecto (sin opciones) genera sólo los stubs del cliente
  - Opciones principales:
    - – ss: genera los skeletons del servidor
    - – d: se define la opción de mapeado de XML a java: adb, jaxbri, XMLBeans, JiBX (usaremos jaxbri)
    - - sd: genera el descriptor de despliegue (services.xml)
    - - p <nombre_paquete>: indica el nombre del paquete Java donde se generan las clases (si no se pone usa las reglas por defecto para definir el nombre a partir del namespace, pero no lo hace bien, así que mejor asignarlo)
    - -wv <version>: Version de WSDL (1.1 / 2.0)

- **java2wsdl** => Genera la descripción WSDL de una clase(s) Java ya existente

Ejemplo Axis2: Creación de un proveedor de Servicio Web

1. Crear un directorio VolumenAreaProveedor y copiar VolumenArea.wsdl en él
   - Si se va a usar Eclipse como IDE, usar directamente el directorio raíz del proyecto Eclipse

2. Crear skeletons
   - `%AXIS2_HOME%\bin\wsdl2java.bat -d jaxbri -wv 1.1 –ss –sd –p es.unican.is2.volumenArea –uri VolumenArea.wsdl`
   - Se genera:
     - build.xml: Fichero de construcción ant
     - .\src : Directorio con el código del skeleton y clases auxiliares
     - .\resources: Contiene el fichero de despliegue, services.xml, y una copia del WSDL

3. Implementar el código de negocio del servicio
   - Directamente en el skeleton o haciendo uso de una clase auxiliar (ver siguiente transparencia)

4. Compilar el código: `ant compile.src`

5. Generar el archivo de despliegue (.aar): `ant jar.server`
   - Se crea el servicio desplegable (.aar) en el directorio .\build\lib

6. Desplegar el servicio en Axis2 sobre Tomcat y acceder a él
   - Arrancar el servidor Axis2 y comprobar que está disponible en http://localhost:8080/axis2/services/VolumenArea

Ejemplo Axis2: Código de implementación del servicio

Suponiendo que se cuenta con la siguiente clase auxiliar VolumenArea (ya implementada)

```
package org.example.businessClasses;

public class VolumenArea {

    public static double area(double alto, double ancho, double largo) {
        return ancho*largo;
    }

    public static double volumen(double alto, double ancho, double largo) {
        return ancho*largo*alto;
    }
}
```

El Skeleton se podría implementar así:

```
package org.example.volumenarea;

public class VolumenAreaSkeleton{

      public double area(org.example.volumenarea.types.Dimensiones area)throws DatosNoValidosException {
            if (area.getAncho() <0|| area.getLargo()<0) {
                    DatosNoValidosException e = new DatosNoValidosException();
                    DatosNoValidos d = new DatosNoValidos();
                    d.setError("El ancho o el largo no pueden ser negativos");
                    e.setFaultMessage(d);
                    throw e;
             }
             return VolumenArea.area(area.getAlto(), area.getAncho(), area.getLargo());
        }}
```

Ejemplo Axis2: Creación de un cliente (enlace estático)

1. Crear un directorio VolumenAreaCliente
   - Si se va a usar Eclipse como IDE, usar directamente el directorio raiz del proyecto Eclipse

2. Generar los stubs para el cliente
   - `%AXIS2_HOME%\bin\wsdl2java.bat -d jaxbri –s –uri http://localhost:8080/axis2/services/VolumenArea?wsdl`
   - Se genera:
     - build.xml: Instrucciones para ant
     - .\src : Directorio con el código del stub

3. Crear un cliente de prueba en el directorio src
   - Clase Java que usa el stub para acceder al servicio (Código en la siguiente trasparencia)

4. Compilar: `ant compile.src`

5. Ejecutar el cliente
   - Asegurarse que el servicio está disponible
   - Desde Eclipse => Ejecutar el cliente (añadiendo las librerías axis2 al BuildPath)
   - Desde Ant=> `ant jar.client` crea un jar (para que sea ejecutable hay que modificar el build.xml)

Código del cliente del servicio AreaService

Cliente estático

```
public class Client{
   public static void main(java.lang.String args[]) {
        try {
             VolumenAreaStub stub = new VolumenAreaStub();

             Dimensiones dim = new Dimensiones();
             dim.setAlto(10.0);
             dim.setAncho(2.0);
             dim.setLargo(3.0);

             try {
                 // Invocación válida
                 System.out.println(stub.area(dim));
                 // Invocación no válida
                 dim.setAncho(-2);
                 System.out.println(stub.area(dim));

             } catch (DatosNoValidosException e) {
                 System.out.println(e.getFaultMessage());
             }
        } catch (Exception e1) {
             e1.printStackTrace();
     }
   }
}
```

Stub

```
package  org.example.volumenarea;

public class VolumenAreaStub extends org.apache.axis2.client.Stub

public double area (org.example.volumenarea.types.Dimensiones dimensiones)
        throws java.rmi.RemoteException,
               org.example.volumenarea.DatosNoValidosException {
            ...
}
```

Soporte para invocación síncrona/asíncrona en AXIS2

- La invocación de un servicio se puede realizar en forma síncrona o asíncrona
  - La implementación del servicio es la misma
  - Es el cliente quién decide el tipo de invocación
  - Independiente del patrón de intercambio de mensajes definido para la operación

- Axis2 usa un mecanismo de callbacks para implementar el caso asíncrono

- El comando `wsdl2java` se configura para la generación de stubs síncronos y/o asíncronos
  - Por defecto wsdl2java genera tanto soporte síncrono como asíncrono
  - Con la opción –s genera sólo soporte síncrono

Soporte síncrono y asíncrono en AXIS2

Stubs síncronos

```
%AXIS2_HOME%\bin\wsdl2java.bat
-d jaxbri –s
–uri VolumenArea.wsdl
```

```
package  org.example.volumenarea;

public class VolumenAreaStub extends org.apache.axis2.client.Stub {

     public double area (Dimensiones dimensiones) throws DatosNoValidosException { ... }

     ...
```

VolumenAreaService.wsdl

```
%AXIS2_HOME%\bin\wsdl2java.bat
-d jaxbri
–uri VolumenArea.wsdl
```

Stubs síncronos y asíncronos

```
package org.example.volumenarea;

public class VolumenAreaStub extends org.apache.axis2.client.Stub {

      public double area (Dimensiones dimensiones) throws DatosNoValidosException { ... }

      public double startArea (Dimensiones dimensiones,
                    VolumenAreaCallbackHandler callback) throws DatosNoValidosException { ...
}
```

```
package org.example.volumenarea;

public abstract class VolumenAreaCallbackHandler {

       public void receiveResultArea(double result) { ... }

       public void receiveErrorArea(double result) { ... }

       ...
}
```

Desarrollo de un cliente asíncrono

1. Generar los stubs con la opción asíncrona
2. Implementar una clase que extienda a la clase `CallbackHandler` e implemente los métodos de recogida de datos y de errores
3. Invocar el servicio a través de los métodos `start<method>`

- Para poder ver mejor el comportamiento asíncrono de la invocación se puede modificar la implementación del servicio VolumenArea:

```
package org.example.volumenarea;

public class VolumenAreaSkeleton{

     public double area (org.example.areaservice.types.Dimensiones dimensiones) throws DatosNoValidosException {
             // Introducimos un retraso en la respuesta
             try {
                  Thread.sleep(10000);
              } catch (InterruptedException e) {}

             if (area.getAncho() <0|| area.getLargo()<0) {
                                 DatosNoValidosException e = new DatosNoValidosException();
                    e.setFaultMessage("El ancho o el largo no pueden ser negativos");
                    throw e;
             }
             return VolumenArea.area(area.getAlto(), area.getAncho(), area.getLargo());        }
    }
```

Rampart: WS-Security en servicios Axis2

- Apache Rampart es un modulo de extensión de Axis2 que implementa WS-Security, WS-SecurityPolicy, WS-SecureConversation and WS-Trust

- Instalación (Consultar “Guía de instalación de software de prácticas”)

- Para incluir información sobre seguridad en mensajes SOAP a través de Rampart, el módulo debe estar vinculado (engaged) tanto en el lado cliente como en el lado servidor

  - En la lado servidor, se vincula añadiendo la siguiente línea al descriptor de despliegue del servicio

    ```
    <module ref="rampart"/>
    ```

  - En lado cliente se hace programáticamente

```
public static void main(java.lang.String args[]) {
      try {
         VolumenAreaStub stub = new VolumenAreaStub();

         // configure and engage Rampart
         ServiceClient client = stub._getServiceClient();
         client.engageModule("rampart");

         ...
```

Autenticación con Rampart: Lado servidor (Paso 1)

1. Definición de la política de seguridad del servicio usando WS-SecurityPolicy
   - Se incluye en el WSDL del servicio (Para cada binding)

```
<!- Binding SOAP -->
<wsdl:binding name="VolumenAreaServiceSOAP" type="tns:VolumenArea">

  <wsp:Policy wsu:Id="UsernameToken" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <wsp:ExactlyOne>
      <wsp:All>
        <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
          <wsp:Policy>
            <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                                               securitypolicy/200702/IncludeToken/AlwaysToRecipient"/>
          </wsp:Policy>
        </sp:SupportingTokens>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>

  <soap:binding ...>

</wsdl:binding>
```

Esta es la política que se añade para soportar autenticación con usuario y contraseña en texto plano

Autenticación con Rampart: Lado servidor (Paso 1)

```
<!- Binding SOAP -->
<wsdl:binding name="VolumenAreaServiceSOAP" type="tns:VolumenArea">

  <wsp:Policy wsu:Id="UsernameToken" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <wsp:ExactlyOne>
      <wsp:All>
        <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
          <wsp:Policy>
            <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                                               securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                <wsp:Policy>
                    <sp:HashPassword/>
                <wsp:Policy>
            </wsp:Policy>
        </sp:SupportingTokens>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>

  <soap:binding ...>

</wsdl:binding>
```

Esta es la política que se añade para soportar autenticación con usuario y contraseña con digest

Autenticación con Rampart: Lado servidor (Paso 2)

2. Desarrollo del Callback que Rampart invoca para comprobar las credenciales
   - Se encarga de autenticar los UsernameToken que le llegan al servicio
   - Extiende a `javax.security.auth.callback.CallbackHandler`
   - Para entender mejor su funcionamiento: http://wso2.com/library/3733/

```
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.ws.security.WSPasswordCallback;

public class PWCBHandler implements CallbackHandler {

    @Override
    public void handle(Callback[] arg0) throws IOException, UnsupportedCallbackException {
        for (int i = 0; i < arg0.length; i++) {
            WSPasswordCallback pwcb = (WSPasswordCallback)arg0[i];
            String id = pwcb.getIdentifier();
            if (id.equals("patri")) {
             pwcb.setPassword("patriPW");
            } else {
                throw new UnsupportedCallbackException(arg0[i], "check failed");
            }
        }
    }

}
```

Este tipo de verificación funciona para el caso de autenticación digest (para texto plano ver ejemplo en Moodle)

Autenticación con Rampart: Lado servidor (Paso 3)

3. Configuración de Rampart y de la política en el descriptor del servicio (service.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
    <service name="VolumenArea">
        ...
        <!--  Añadimos el módulo Rampart -->
        <module ref="rampart"/>

        <!--  Mismo bloque que en el WSDL -->
        <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
                    xmlns:wsu="http://docs.oasis-open-org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                    wsu:Id="UsernameToken">
           <wsp:ExactlyOne>
              <wsp:All>
                 <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                       <sp:UsernameToken
                          sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                       </sp:UsernameToken>
                    </wsp:Policy>
                 </sp:SupportingTokens>


                 <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
                     <ramp:passwordCallbackClass>org.example.volumenarea.PWCBHandler</ramp:passwordCallbackClass>
                 </ramp:RampartConfig>

              </wsp:All>
           </wsp:ExactlyOne>
        </wsp:Policy>
    </service>
</serviceGroup>
```

Clase Password Callback

Autenticación con Rampart: Lado cliente

El proceso de desarrollo del cliente es el mismo que en el caso habitual, salvo que en el código del cliente hay que vincular Rampart e incluir las credenciales

```
public class ClientSimple {
   public static void main(java.lang.String args[]) {
      try {
         VolumenAreaStub stub = new VolumenAreaStub();

         // configure and engage Rampart
         ServiceClient client = stub._getServiceClient();
         client.engageModule("rampart");

         Options options = client.getOptions();
         options.setProperty(RampartMessageData.KEY_RAMPART_POLICY,
             loadPolicy("policy.xml"));

         options.setUserName("patri");
         options.setPassword("patriPW");

         Dimensiones dim = new Dimensiones();
         dim.setAlto(10.0);
         dim.setAncho(2.0);
         dim.setLargo(3.0);
         System.out.println("El area es :"+ stub.calculaArea(dim));

      } catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

policy.xml => Fichero con la política a aplicar (bloque <ws:policy> mostrado anteriormente)

Impte: Para que se pueda ejecutar desde eclipse, el archivo rampart-1.6.2.mar debe estar en el buildPath

Manejo de sesiones en Axis2

- En teoría, los servicios web son “stateless”
  - En la práctica es difícil desarrollar aplicaciones mínimamente complejas sin manejo de sesiones
- El concepto de sesión influye en el número de instancias del servicio que se crean para atender peticiones
- Axis2 ofrece cuatro opciones para el alcance de una sesión (session scope):
  - Request: Una instancia por petición
  - Transport: Una instancia por cada sesión de transporte (Cookies HTTP)
  - SOAP: Una instancia por cada sesión SOAP (definida por cabeceras especiales definidas en WS-Addressing)
  - Application: Una única instancia del servicio (ciclo de vida del servidor web)
- Cada servicio tiene su scope, definido en el services.xml del fichero .aar
  - Por defecto es request

```
<service name="ImpuestoCirculacion" scope="transport">
```

Notificación de inicio y fin de sesión

- Los métodos init() y destroy() de la interfaz Lifecycle son invocados por Axis2 en el servicio al inicio y fin de cada sesión
  - Se pueden usar para realizar acciones (o simplemente conocer) cuando se crea y se destruye cada instancia de servicio
  - Se sobrescriben en la clase que implementa el servicio

```
package org.example.volumenarea;

import org.apache.axis2.context.ServiceContext;
import org.apache.axis2.service.Lifecycle;

public class VolumenAreaSkeleton implements Lifecycle {

     public double area (org.example.volumenarea.types.Dimensiones dimensiones) throws DatosNoValidosException {
             ...
     }

    @Override
    public void init(ServiceContext arg0) throws AxisFault {
        System.out.println("Creada la instancia del servicio");
    }

    @Override
    public void destroy(ServiceContext arg0) {
        System.out.println("Destruida la instancia del servicio");
    }
}
```

Uso de variables en sesiones Axis2

- El mejor modo de almacenar variables “globales” en servicios Axis2 de manera que se mantengan entre sesiones es mediante el ServiceContext

Módulos de extension

- Se puede extender al funcionalidad de Axis2 a través de módulos
- Un módulo es un paquete