WEB Services

Interoperabilidad entre Aplicaciones

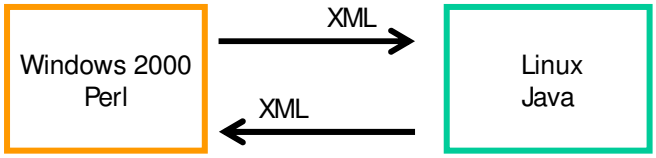
Modelo Programa-Programa …..P2P

Contenido

- Servicios en la Web
- El papel de XML y la plataforma Java
- Las APIs de Java para XML
  - JAXP
  - JAXB
  - JAX-RPC
  - JAXR
  - JAXM
- Ejemplo
  - Códigos
  - Instalación y ejecución

Servicios en la Web

- Es un servicio disponible en Internet que utiliza sistema de intercambio de información vía XML y no es dependiente de ningún sistema operativo ni de algún lenguaje de programación.



- Una aplicación envía una request a un servicio en una determinada URL usando el protocolo SOAP encima de HTTP. El servicio Web recibe la petición, la procesa y devuelve una respuesta.
- Ejemplo típico:
  - un servicio Web de valores de bolsa donde la request solicita el valor de una acción determinada y la respuesta ofrece ese valor. En este caso la request y la response son partes del mismo método de llamada.

2

Servicios en la Web

- Otro ejemplo es un servicio que ofrece una ruta eficiente para la entrega de productos. En este caso, una empresa envía una request con los destinos, el servicio Web los procesa y determina la ruta de mínimo coste. En esta caso, el algoritmo que resuelve la ruta óptima podría tardar y por tanto el response no debe formar parte de la request.
- En muchos casos, una empresa puede ser consumidora de servicios Web y al mismo tiempo ofrecer sus servicios Web a otras.
  - Típico ejemplo es un distribuidor al por mayor tiene un rol de consumidor cuando usa el servicio Web para comprobar si una empresa le ofrece un producto determinado y será un proveedor de servicio Web en el caso de que ofrezca a sus clientes información de sus productos.

3

El role de XML y la plataforma Java

- Los Web Services dependen de la habilidad de las diferentes compañías (partes) en comunicarse entre sí incluso si utilizan diferentes plataformas.
- XML (e**X**tensible **M**arkup **L**anguage), es un lenguaje de marcado que ofrece portabilidad de los datos. XML se usa ya en casi todos los entornos Web y cada vez más, las empresas apuestan por este lenguaje para intercambio de datos.
- También la plataforma Java ha sido clave para la evolución de esta tecnología. SUN tiene tiene una API Java para XML haciendo la integración en las aplicaciones muy interesante.

http://java.sun.com/webservices/

4

Roles en los Web Services

- Service Provider
  - Implementa el servicio y lo hace disponible en Internet.
- Service Requestor
  - Consumidor de servicios. Inicializa un nuevo servicio Web creando una nueva conexión con y realiza peticiones XML.
- Service Registry
  - Directorio de servicios centralizado. El registry ofrece un lugar centralizado donde los desarrolladores puedan publicar nuevos servicios o encontrar los ya publicados.

5

Los 4 Protocolos en Web Services

- Service Transport
  - HTTP, SMTP, FTP, …
- XML messaging
  - XML-RPC, SOAP XML.
- Service Description
  - WSDL
- Service Discovery
  - UDDI

6

XML-RPC

- Es un protocolo sencillo que utiliza mensajes XML para realizar RPC.
- Las Request son codificadas en XML y enviadas vía HTTP Post.
- Las XML Response son devueltas en el cuerpo de un mensaje de Response HTTP.
- XML-RPC es independiente de la plataforma.
- Permite la comunicación entre aplicaciones diversas.

7

Ejemplo de Request XML-RPC

```
<?xml version="1.0">
<methodCall>
   <methodName>weather.getWeather<methodName>
   <params>
      <param><value>10016</value></param>
   <params>
<methodCall>
```

8

Ejemplo de Response XML-RPC

```
<?xml version="1.0">
<methodResponse>
   <params>
      <param><value><int>32</int></param>
   <params>
<methodResponse>
```

9

SOAP (Simple Object Access Protocol)

- Es un protocolo basado en XML para el intercambio de información entre computadores.
- Es independiente de la plataforma.
- La forma normal de usarlo es como RPC sobre HTTP.
- SOAP es más complejo que XML-RPC. Hace uso de XML namespaces y de XML Schemas. Sin embargo, el cuerpo de la Request SOAP especifica tanto el método a ejecutar como los parámetros.

10

Ejemplo de mensaje SOAP

Valor de bolsa:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type:text/xml; charset="utf-8"
Content-Length:xxxx
SOAPAction:"Some-URI"
<SOAP-ENV:Envelope
     Xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
 SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/>
<SOAP-ENV:Body>
   <m:GetLastTradePrice xmlns:m="some-URI">
     <symbol>IBM</symbol>
   </m:GetLastTradePrice>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

11

Ejemplo de mensaje SOAP

Response de :

```
HTTP/1.1 200 OK
Content-Type:text/xml; charset="utf-8"
Content-Length:xxxx

<SOAP-ENV:Envelope
     Xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
 SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/>

   <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="some-URI">

       < Price>100.21</Price>

     </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

12

WSDL (Web Services Description Language)

- WSDL es una gramática XML para especificar una interfaz pública de un servicio Web.
- Esta interfaz puede incluir información de:
  - métodos disponibles,
  - tipos de datos de los mensajes,
  - protocolos a utilizar,
  - dirección donde localizar el servicio.

13

Ejemplo de WSDL

```
<?xml version="1.0"?>
  <definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
        <all>
        <element name="tickerSymbol" type="string"/>
        </all>
        </complexType>
      </element>
      <element name="TradePrice">
        <complexType>
        <all>
        <element name="price" type="float"/>
        </all>
        </complexType>
      </element>
    </schema>
  </types>
```


Ejemplo de WSDL

```
<message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
</message>
<portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
      </operation>
</portType>
```

Ejemplo de WSDL

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
          <soap:operation
                soapAction="http://example.com/GetLastTradePrice"/>
          <input>
                <soap:body use="literal"/>
          </input>
          <output>
                <soap:body use="literal"/>
          </output>
    </operation>
</binding>
<service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
          <soap:address location="http://example.com/stockquote"/>
    </port>
</service>
</definitions>
```

16

UDDI (Universal Description, Discovery and Integration of Web Services)

- UDDI representa la capa de búsqueda de servicios dentro de los protocolos Web Services.
- Es una especificación acerca de la información de los registros distribuidos basados en la Web.
- http://www.uddi.org

17

UDDI (Universal Description, Discovery and Integration of Web Services)

- El componente principal de UDDI es el registro de negocios UDDI.
- Consiste en información de los servicios en tres formas diferentes:
  - Páginas blancas
    - Incluye dirección, contacto, …
  - Páginas amarillas
    - Incluye categoría de compañías según taxonomías estándar.
  - Páginas verdes
    - Información técnica acerca de los servicios del negocio.

18

Ejemplo de Servicios disponibles

- Como ejemplo de funcionamiento de la pila de protocolos, IBM ofrece un browser para:
  - Buscar por servicios Web.
  - Ver sus descripciones.
  - Invocar a sus servicios. ……

Emerging Technologies Toolkit:

http://www-106.ibm.com/developerworks/webservices/demos/ettk/

19

Las APIs de Java SUN para XML

- La API de Java para XML permite escribir aplicaciones Web en Java.
- La API consta de diferentes APIs:
  - Orientadas a los documentos
    - Java API for XML Processing (**JAXP**):
      - Permite el procesamiento de documentos XML con varios parsers.
    - Java Architecture for XML Binding (**JAXB**)
      - Permite el procesamiento de documentos XML usando JavaBeans.
  - Orientadas a los procedimientos
    - Java API for XML-based RPC (**JAX-RPC**):
      - Envía llamadas de métodos SOAP a terceras partes en Internet y recibe los resultados.
    - Java API for XML Messaging (**JAXM**):
      - Envía mensajes SOAP en Internet de forma estándar.
    - Java API for XML Registries (**JAXR**):
      - Ofrece una forma estándar para acceder a los registros de negocio y compartir información.

20

Ejemplo completo

- El dueño de una cadena de cafeterías (**CoffeeBreak**) quiere expandir su negocio vendiendo distintos tipos de cafés por Internet.
- Le indica al gerente de la compañía que busque nuevos distribuidores de cafés para conocer sus precios para poder decidir con que nuevas compañías va a realizar negocios.
- No desea tener en stock ningún producto.

21

COFFEE BREAK

- Tareas a realizar según el gerente:
  - **TAREA 1. Buscar nuevos distribuidores.**
  - **TAREA 2. Solicitarles la lista de precios de sus productos.**
  - **TAREA 3. Comparar precios y Realizar los pedidos al distribuidor.**
  - **TAREA 4. Venta de café por Internet.**

22

COFFEE BREAK

- El gerente indica al Ingeniero del Software (IS) que realice esta tarea buscando en las diferentes compañías.
- **TAREA 1. Buscar nuevos distribuidores**
  - El IS decide que la mejor forma es buscar un registro UDDI (Universal Description, Discovery, and Integration (UDDI) registry), donde la propia empresa se ha registrado.
  - El IS utiliza JAXR para enviar una consulta de búsqueda de distribuidores de café. La implementación con JAXR utiliza JAXM para enviar el query al registro, aunque esto es transparente para el IS.
  - El registro UDDI recibe la pregunta y aplica el criterio de búsqueda indicado en código JAXR para buscar las empresas que cumplen dicho criterio de búsqueda.
  - Cuando la búsqueda se ha completado, el registro devuelve la información de cómo contactar con dichas empresas. Aunque el registro utiliza JAXM detrás de todo este proceso, la respuesta al IS es código JAXR.

23

COFFEE BREAK

- **TAREA 2. Solicitar la lista de precios de sus productos**
  - El siguiente paso del IS es solicitar la lista de precios de cada uno de los distribuidores.
  - En la tarea anterior obtuvo una descripción WSDL de cada uno de los distribuidores, que indica el procedimiento para obtener los precios y además también la URI a donde se debe enviar la request.
  - La aplicación realiza las llamadas a los procedimientos remotos RPC a través de la API JAX-RPC y obtiene las respuestas de los distribuidores.
  - La empresa ya tenía implementado con el distribuidor anterior el intercambio de mensajes basado en XML-schemas para realizar las compras. Se utilizaba la API JAXM para solicitar el precio actual y el distribuidor devolvía la lista de precios en un mensaje JAXM.

24

COFFEE BREAK

- **TAREA 3. Comparar precios y Realizar los pedidos al distribuidor.**
  - Al recibir las respuestas con los precios, el IS procesará la lista usando SAX.
  - Una vez que la aplicación tiene todos los precios de los distintos distribuidores, los compara y presenta los resultados.
  - Cuando el dueño de la empresa y el gerente deciden que distribuidores escoger, se podrá comenzar a hacer negocios con ellos solicitando los pedidos
  - Los pedidos a los nuevos distribuidores se envían a través de JAX-RPC; las peticiones al distribuidor anterior se realizaba con JAXM. Cada distribuidor, que utilice JAX-RPC o JAXM, debe responder enviando un mensaje de confirmación con la orden del pedido y la fecha de envío.

25

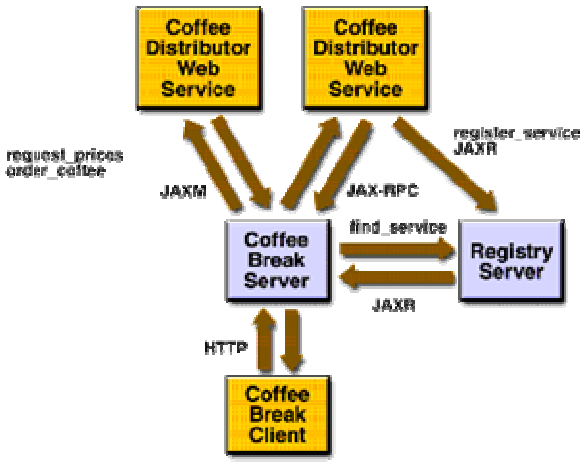
COFFEE BREAK

- **TAREA 4. Venta de café por Internet.**
  - Al mismo tiempo que las tareas anteriores, la empresa se ha estado preparando para la venta de café por Internet.
  - Publicará en Internet un formulario con los precios que además sirva como pedido.
  - Antes de que realice dicha publicación, la compañía debe determinar que precios pondrá a los productos.
  - El IS escribe una aplicación que cargará a cada producto un 135% al inicial.
  - El IS utiliza JSP para crear un pedido por Internet según las productos y las cantidades seleccionadas por el cliente.

26

Implementación de COFFEE BREAK

- Servidor
  - Servlets
  - JSP
  - Java Beans
- Cliente
  - HTML



27

JAX-RPC Distributor Service

- El servidor Coffee Break también actual de cliente al realizar llamadas remotas sobre el servicio distribuidor JAX-RPC.
- El código del servicio consiste en una interfaz del servicio, una clase de implementación del servicio y diferentes Beans que son utilizados como parámetros en algunos métodos y como valores de retorno.
- **Interfaz del servicio (Service Interface)**
  - **La interfaz del servicio (SupplierIF), define los métodos que pueden ser llamados por los clientes remotos. Los parámetros y los resultados retornados son componentes JavaBean.**
    - **AddressBean – información del cliente para la entrega.**
    - **ConfirmationBean – Identificación del pedido y fecha del mismo.**
    - **CustomerBean – información de contacto del cliente.**
    - **LineItemBean – item del pedido.**
    - **OrderBean – identificador de pedido, cliente, dirección, lista de items, precio total.**
    - **PriceItemBean – listado de precios (nombre del café y precio distribuidor).**
    - **PriceListBean – lista de precios.**

28

JAX-RPC Distributor Service

- Debido a que estos componentes se utilizan en otros ejemplos, se han puesto en el directorio:

  <*JWSDP_HOME*>/docs/tutorial/examples/cb/common/src

- El código fuente de la interfaz **SupplierIF** está en:

  <*JWSDP_HOME*>/docs/tutorial/examples/cb/jaxrpc/src.

```
package com.sun.cb;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface SupplierIF extends Remote {
   public ConfirmationBean placeOrder(OrderBean order) throws
                                                    RemoteException;
   public PriceListBean getPriceList() throws RemoteException;
}
```

29

JAX-RPC Distributor Service

- **Implementación del Servicio:**
  - La clase "**SupplierImpl**" implementa los métodos "**placeOrder**" y "**getPriceList**", definidos en la interfaz "**SupplierIF**".
  - Estos métodos son muy sencillos para dejar más visible el código relacionado con JAX-RPC.
- El método "**placeOrder**" acepta como entrada un pedido de café y devuelve la confirmación de tal pedido. Para simplificar, este método confirma toda petición y pone como día de envío el siguiente día de haber hecho el pedido. (La fecha es calculada por "**DateHelper**", una clase útil que reside en el directorio cb/common.

```
public ConfirmationBean placeOrder(OrderBean order) {
Date tomorrow = com.sun.cb.DateHelper.addDays(new Date(), 1);
ConfirmationBean confirmation = new ConfirmationBean(order.getId(),
                                                     tomorrow);
return confirmation;
}
```

30

JAX-RPC Distributor Service

- El método "**getPriceList**" devuelve un objeto "**PriceListBean**" con la lista de los nombres de café y su precio que pueden solicitarse con este servicio.
- El método "**getPriceList**" crea elobjeto PriceListBean invocando al método privado llamado loadPrices. En este caso los precios se toman del fichero "**SupplierPrices.properties**".

```
public PriceListBean getPriceList() {
   PriceListBean priceList = loadPrices();
   return priceList;
}

private PriceListBean loadPrices() {
   String propsName = "com.sun.cb.SupplierPrices";
   Date today = new Date();
   Date endDate = DateHelper.addDays(today, 30);
   PriceItemBean[] priceItems = PriceLoader.loadItems(propsName);
   PriceListBean priceList = new PriceListBean(today, endDate,
    priceItems);
   return priceList;
}
```

31

JAX-RPC Distributor Service

- **Publicación del Servicio en el Registro**
  - Como deseamos que los clientes encuentren nuestro servicio, debemos publicarlo en un registro.
  - OrgPublisher: programa que publica el servicio.
  - OrgRemover: programa que elimina el servicio del registro.
  - Ninguno es parte de la aplicación Web, son programas independientes que se ejecutan con el comando siguiente:
    ant set-up-service
  - Una vez que el servicio se instale, se publica en el registro. De la misma forma, justo antes de que el servicio sea eliminado, se elimina del registro.

32

OrgPublisher {...}

```
package com.sun.cb;
import javax.xml.registry.*; import java.util.ResourceBundle;  import java.io.*;
public class OrgPublisher {
  public static void main(String[] args) {
    ResourceBundle registryBundle = ResourceBundle.getBundle
                                        ("com.sun.cb.CoffeeRegistry");
    String queryURL = registryBundle.getString("query.url");
    String publishURL = registryBundle.getString("publish.url");
    String username = registryBundle.getString("registry.username");
    String password = registryBundle.getString("registry.password");
    String endpoint = registryBundle.getString("endpoint");
    String keyFile = registryBundle.getString("key.file");
    JAXRPublisher publisher = new JAXRPublisher();
    publisher.makeConnection(queryURL, publishURL);
    String key = publisher.executePublish (username, password, endpoint);
    try {
      FileWriter out = new FileWriter(keyFile);
      out.write(key);    out.flush();    out.close();
    } catch (IOException ex) { System.out.println(ex.getMessage());
  }
 }
}
```

33

OrgPublisher

- El programa “OrgPublisher” comienza cargando los valores de las propiedades del fichero “CoffeeRegistry.properties”.
- A continuación el programa instancia una clase de utilidades “**JAXRPublisher**”.
- “OrgPublisher” se conecta al registro invocando el método “makeConnection” de “JAXRPublisher”.
- Para publicar el servicio, “OrgPublisher” llama al método “executePublish”, que acepta como entrada el nombre del usuario, la clave y un punto de fin.
  - El username y passwd son requeridos por el Servidor de Registro.
  - El punto de fin es la URL que los usuarios remotos usarán para contactar con el servicio JAX-RPC.
- El método “executePublish” de JAXRPublisherdevuelve una clave que identifica de manera única el servicio en el registro.
- OrgPublisher guarda esta clave en un fichero texto llamado “orgkey.txt”.

34

JAXRPublisher.java

- En primer lugar, el método “makeConnection” crea una conexión al Servidor de Registro. Debe especificar algunas propiedades de conexión usando la URL de query y de publicación desde “CoffeeRegistry.properties”.

```
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL", queryUrl);
props.setProperty("javax.xml.registry.lifeCycleManagerURL", publishUrl);
```

- A continuación, el método “makeConnection” crea la conexión usando las propiedades:

```
ConnectionFactory factory = ConnectionFactory.newInstance();
factory.setProperties(props);
connection = factory.createConnection();
```

- El método “executePublish” tiene tres argumentos: username, passwd y URI final. Comienza por obtener un objeto “RegistryService”, después un objeto “BusinessQueryManager” y finalmente un objeto “BusinessLifeCycleManager” que permite realizar queries y manejar datos del registro.

```
rs = connection.getRegistryService();
bqm = rs.getBusinessQueryManager();
blcm = rs.getBusinessLifeCycleManager();
```

35

JAXRPublisher.java

- Como es necesario identificarse por clave para publicar datos, utiliza el username y passwd para establecer las credenciales:

```
PasswordAuthentication passwdAuth = new PasswordAuthentication(
                                        username, password.toCharArray());
Set creds = new HashSet();
creds.add(passwdAuth);
connection.setCredentials(creds);
```

- Crea un objeto "**Organization**" con el nombre "**JAXRPCCoffeeDistributor**" y utiliza un objeto tipo User que servirá como primer contacto.

```
ResourceBundle bundle=ResourceBundle.getBundle("com.sun.cb.CoffeeRegistry");
org = blcm.createOrganization(bundle.getString("org.name"));
InternationalString s = blcm.createInternationalString
                                    (bundle.getString("org.description"));
 org.setDescription(s);
User primaryContact = blcm.createUser();
PersonName pName = blcm.createPersonName(bundle.getString("person.name"));
primaryContact.setPersonName(pName);
```

36

JAXRPublisher.java

- Proporciona al objeto "JAXRPCCoffeeDistributor" una clasificación basada en North American Industry Classification System (NAICS). En concreto se utilizará la clasificación de "Other Grocery and Related Products Wholesalers".

```
Classification classification = (Classification)
                                    blcm.createClassification(cScheme,
                                    bundle.getString("classification.name"),
                                    bundle.getString("classification.value"));
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
```

- Seguidamente añade el servicio "JAX-RPC" denominado "JAXRPCCoffee Service" y le hace el bind. La URI del servicio para que accedan los cliente es:

http://localhost:8080/jaxrpc-coffee-supplier/jaxrpc/SupplierIF

37

JAXRPublisher.java

```
Collection services = new ArrayList();
Service service = blcm.createService(bundle.getString("service.name"));
InternationalString is = blcm.createInternationalString
    (bundle.getString("service.description"));

service.setDescription(is);
// Create service bindings Collection serviceBindings = new ArrayList();

ServiceBinding binding = blcm.createServiceBinding();
is = blcm.createInternationalString (bundle.getString("service.binding"));
binding.setDescription(is);

try { binding.setAccessURI(endpoint);
 } catch (JAXRException je) {
    throw new JAXRException("Error: Publishing this " + "service in the registry has
     failed because " + "the service has not been installed on Tomcat.");
 } serviceBindings.add(binding);

// Add service bindings to service
service.addServiceBindings(serviceBindings);

// Add service to services, then add services to organization
services.add(service);
org.addServices(services);
```

38

Servicio Distribuidor **JAXM**

- Realiza las gestiones para el intercambio de mensajes entre el distribuidor y el Coffee Break.
  - Que tipo de mensajes se mandarán.
  - El formato de esos mensajes.
  - Que tipo de envío de mensajes realizarán (full-half..).
- El servidor del **Coffee Break** envía dos tipos de mensajes:
  - Petición de precios actuales al mayorista
  - Peticiones de café de los clientes de Coffee Break
- El mayorista de cafés JAXM responde con dos tipos de mensajes:
  - Lista de precios actuales
  - Confirmación de pedidos.

39

Servicio Distribuidor **JAXM**

- Todos los mensajes intercambiados deben seguir una estructura XML especificada en un DTD para cada tipo de mensaje.
- Esto permite intercambiar mensajes incluso cuando cada uno utiliza formatos diferentes en sus documentos.
- Los cuatro tipos anteriores se especifican en los siguientes DTD:
  - request-prices.dtd
  - price-list.dtd
  - coffee-order.dtd
  - confirm.dtd
- Todos los mensajes intercambiados deben seguir una estructura XML especificada en un DTD para cada tipo de mensaje.

40

price-list.xml

```
<!DOCTYPE price-list
   PUBLIC "-//Sun Microsystems, Inc.//DTD priceList//EN"
   "http://java.sun.com/webservices/dtds/price-list.dtd">
<price-list
   <coffee>
         <coffee-name> Arabica </coffee-name>
         <price> 4.50 </price>
   </coffee>
   <coffee>
         <coffee-name> HouseBlend </coffee-name>
         <price> 5.00 </price>
   </coffee>
   <coffee>
         <coffee-name> Espresso </coffee-name>
         <price> 5.00 </price>
   </coffee>
   <coffee>
         <coffee-name> Dorada </coffee-name>
         <price> 6.00 </price>
   </coffee>
</price-list>
```

41

Cliente **JAXM**

- El servidor Coffee Break es un cliente JAXM envía una petición al distribuidor JAXM.
- Como se está usando el formato de request-response del sistema de mensajes JAXM, las aplicaciones cliente utilizan el método SOAPConnection para enviar mensajes.

```
SOAPMessage response = con.call(request, endpoint);
```

- El cliente tiene dos tareas importante s arealziar:
  - Crear y enviar la petición
  - Extraer el contenido de la respuesta.
- Estas tareas las realizan las clases PriceListRequest y OrderRequest.

Cliente **JAXM**

- **Crear y Enviar la petición para obtener una lista actualizada.**
  - La lista es obtenida por el método getPriceList de PriceListRequest.
  - El método "**getPriceList**" comienza por crear la conexión para hacer la petición.
  - A continuación obtiene un objeto "**MessageFactory"** por defecto de forma que puede crear un mensaje tipo SOAPMessage.

```
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection con = scf.createConnection();
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
```

  - Luego accede al objeto "**SOAPEnvelope**", que será utilizado para crear un objeto "**Name"** para cada nuevo elemento que se crea y para acceder al objeto "**SOAPBody**", donde se pondrá en contenido del mensaje.

```
SOAPPart part = msg.getSOAPPart();
SOAPEnvelope envelope = part.getEnvelope();
SOAPBody body = envelope.getBody();
```

Cliente **JAXM**

- El fichero "**request-price.dtd"** especifica que el elemento más alto en la jerarquía es "**request-prices"** y que contiene un elemento "**request"**.
- El texto añadido a request es el texto a enviar con la request.
- Cada elemento nuevo que se añada a un mensaje deberá tener un objeto "**Name**" para identificarlo, el cual es creado por el método createName de "**Envelope"**.

```
Name bodyName = envelope.createName("request-prices",
                    "RequestPrices", "http://sonata.coffeebreak.com");
SOAPBodyElement requestPrices =
                    body.addBodyElement(bodyName);
```

44

Cliente **JAXM**

- En las siguientes líneas se añade al elemento "request-prices" (representado por SOAPBodyElement requestPrices. Entonces se añade un nuevo nodo de texto conteniendo el texto de la petición.
- Como no hay más elementos en la request, el código llama al método "**saveChanges"** sobre el mensaje a guardar.

```
Name requestName = envelope.createName("request");
SOAPElement request = requestPrices.addChildElement(requestName);
request.addTextNode("Send updated price list.");
msg.saveChanges();
```

- Una vez creado el mensaje, ese envía el mensaje al suministrador de café JAXM.
- El mensaje enviado vía el objeto "**SOAPConnection"** es el objeto tipo "**SOAPMessage" "msg"**, y el destino es la URI del suministrador JAXM.

```
URL endpoint = new URL(  "http://localhost:8080/jaxm-coffee-
                                supplier/getPriceList");
SOAPMessage response = con.call(msg, endpoint);
con.close();
```

- Cuando se ejecuta el método "**call**" , Tomcat ejecuta el servlet "**PriceListServlet**". Este servlet crea y devuelve un objeto "**SOAPMessage**" cuyo contenido es la lista precios de los distribuidores JAXM.

45

Cliente **JAXM**

- **Obtener la lista de precios**
  - En esta parte se realizan las siguientes tareas:
    - Recuperar la lista de precios de la respuesta, el objeto "SOAPMessage" devuelto por el método call.
    - Devolver la lista de precios como un bean tipo "**PriceListBean**".
- Se crea un objeto tipo Vector vacío que contendrá los elementos "coffee-name" y "price" que han sido extraidos de la respuesta.
- De la respuesta se accede al objeto "**SOAPBody"**, que contiene el contenido del mensaje.

```
Vector list = new Vector();
SOAPBody responseBody =
    response.getSOAPPart().getEnvelope().getBody();
```

- El siguiente paso consiste en recuperar el objeto "SOAPBodyElement".
- El método "**getChildElements"** devuelve un objeto Iterator que contiene todos los elementos hijos del elemento que fue llamado. **it1** contiene el objeto **SOAPBodyElement** "**bodyEl**", que representa el elemento "**price-list"**.

```
Iterator it1 = responseBody.getChildElements();
while (it1.hasNext()) {
   SOAPBodyElement bodyEl = (SOAPBodyElement)it1.next()
```

46

Cliente **JAXM**

- El objeto Iterator "**it2**" tiene los elementos hijo de bodyEl, que representa elementos "**coffee"**.
- Al llamar al método "**next"** con "**it2**" recuperamos el primer elemento de "**coffee"** en el elemento "**bodyEl"**. Si "**it2**" tiene otro elemento, el método "**next**" devolverá un nuevo elemento de "**coffee**"

```
Iterator it2 = bodyEl.getChildElements();
while (it2.hasNext()) {
   SOAPElement child2 = (SOAPElement)it2.next();
```

- Para coger los valores de los elementos:

```
Iterator it3 = child2.getChildElements();
while (it3.hasNext()) {
   SOAPElement child3 = (SOAPElement)it3.next();
   String value = child3.getValue();
   list.addElement(value);
 }
}
}
```

47

Cliente **JAXM**

- El siguiente paso es poner en un **ArrayList priceItems** los nombres de los cafés y sus precios. Devolverá un bean tipo "**PriceListBean"**.

```
ArrayList priceItems = new ArrayList();

for (int i = 0; i < list.size(); i = i + 2) {
   priceItems.add(new PriceItemBean(list.elementAt(i).toString(), new
                  BigDecimal(list.elementAt(i + 1).toString())));
   System.out.print(list.elementAt(i) + " ");
   System.out.println(list.elementAt(i + 1));
 }

Date today = new Date();
Date endDate = DateHelper.addDays(today, 30);
PriceListBean plb = new PriceListBean(today, endDate, priceItems);
```

48

Pedido de Café

- El otro tipo de mensaje que el servidor Coffee Break puede enviar al distribuidor JAXM es la petición de café.
- Esto se lleva a cabo con el método "**placeOrder"** de "**OrderRequest**" según el DTD "**coffee-order.dtd**"
- ***Crear el pedido***
  - Como en el caso del cliente que requiere una lista de precios, el método "**placeOrder"** comienza creando un objeto "**SOAPConnection"**, un objeto "**SOAPMessage**", y accediendo a los objetos "**SOAPEnvelope"** y "**SOAPBody**".

```
SOAPConnectionFactory scf=
                SOAPConnectionFactory.newInstance();
SOAPConnection con = scf.createConnection();
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
SOAPPart part = msg.getSOAPPart();
SOAPEnvelope envelope = part.getEnvelope();
SOAPBody body = envelope.getBody();
```

49

Pedido de Café

- A continuación, se crean y añaden elementos XML a la petición.
- Como es requerido, el primer elemento es un "**SOAPBodyElement"**, en este caso un "**coffee-order**".

```
Name bodyName = envelope.createName("coffee-order",
                                    "PO", "http://sonata.coffeebreak.com");
SOAPBodyElement order = body.addBodyElement(bodyName);
```

- La aplicación añade el siguiente nivel de los elementos, poniendo primero "**orderID**".
- El valor dado a "**orderID**" se obtiene del objeto "**OrderBean"** pasado al método "**OrderRequest.placeOrded"**.

```
Name orderIDName = envelope.createName("orderID");
SOAPElement orderID = order.addChildElement(orderIDName);
orderID.addTextNode(orderBean.getId());
```

50

Pedido de Café

- El siguiente elemento "**customer**" tiene varios hijos con información del usuario-cliente. Esta información está en el bean "**OrderBean**".

```
Name childName = envelope.createName("customer");
SOAPElement customer = order.addChildElement(childName);
childName = envelope.createName("last-name");
SOAPElement lastName = customer.addChildElement(childName);
lastName.addTextNode(orderBean.getCustomer(). getLastName());
childName = envelope.createName("first-name");
SOAPElement firstName = customer.addChildElement(childName);
firstName.addTextNode(orderBean.getCustomer().getFirstName());
childName = envelope.createName("phone-number");
SOAPElement phoneNumber = customer.addChildElement(childName);
phoneNumber.addTextNode(orderBean.getCustomer().getPhoneNumber());
childName = envelope.createName("email-address");
SOAPElement emailAddress = customer.addChildElement(childName);
emailAddress.addTextNode(orderBean.getCustomer().getEmailAddress());
.....address, ....., café, cantidad y precio.....
```

51

Pedido de Café

- Una vez completado el mensaje, la aplicación lo envía y cierra la conexión.

```
URL endpoint = new URL(   "http://localhost:8080/jaxm-
  coffee-supplier/orderCoffee");
SOAPMessage reply = con.call(msg, endpoint);
con.close();
```

52

Cliente Web.....

53