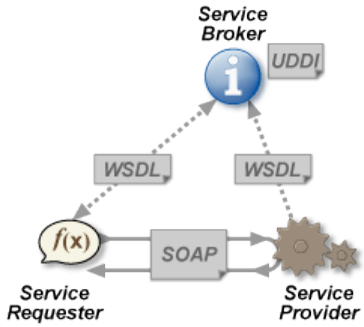
Pràctica 5 – WebServices

Pere Joan Martorell Calaf
Bernat Farrero Badal

Introducció

En aquesta pràctica utilitzarem un altre sistema de comunicació entre màquines, WebServices. WS és un sistema software que permet la interconnexió entre màquines independentment de la seva plataforma. El sistema consta d'una interfície descrita en llenguatge processable per la màquina, concretament WSDL (Web Services Description Language). Els sistemes que es connecten al Web Service usen missatges SOAP, generalment per mitjà de dades serialitzades en XML a través del protocol HTTP. SOAP (Simple Object Access Protocol) és l'estàndard que s'usa com a protocol d'intercanvi d'informació estructurada en un entorn distribuit. L'arquitectura SOAP consisteix en vàries capes de format, intercanvi, transport, missatge i extensió de protocol.



Arquitectura Web Services

Axis és un entorn per Java i C++ per crear processadors SOAP. En aquest pas ens permet convertir fàcilment a missatges SOAP objectes en llenguatge Java.

Instal·lació d'eines

Per dur-ho a terme primer de tot instal·larem el servidor web Tomcat al nostre directori de treball. Seguidament descarregarem l'Apache Axis, un framework per construir WebServices que inclou també el tcpmon, un monitor del tràfic TCP/IP.

```
wget http://studies.ac.upc.edu/FIB/PXC/lab/bin/axis-1_1.tar.gz
tar xzf axis-1_1.tar.gz
```

També necessitarem el Xerces, que ens servirà per processar i manipular XML i que haurem de col·locar juntament amb els altres *jars* de l'Axis dins el directori webapps del Tomcat perquè els pugui reconèixer com un servei més.

```
wget http://studies.ac.upc.edu/FIB/PXC/lab/bin/xerces.jar
mv xerces.jar axis-1_1/webapps/axis/WEB-INF/lib/
cp -R axis-1_1/webapps/axis/ ~/tomcat/webapps/
```

També ens descarregarem el JAF (Java Activation Framework) i el descomprimirem dintre la carpeta del Tomcat.

```
wget http://docencia.ac.upc.edu/FIB/PXC/lab/bin/jaf-1_1-fr.zip
unzip jaf-1_1-fr.zip
cp jaf-1.1/activation.jar ~/tomcat/webapps/axis/WEB-INF/lib/
```

Perquè l'Axis acabi de funcionar haurem de crear un fitxer axis.xml i introduir-hi la següent línia de codi:

```
<Context path="/axis" docBase="axis" debug="0" reloadable="true" />
```

Aquest fitxer el mourem dintre del directori del Tomcat.

```
cp axis.xml ~/tomcat/conf/Catalina/localhost/
```

Variables d'entorn

Per definir les variables d'entorn crearem un fitxer anomenat per exemple setCLASSPATH.sh que contindrà un script de bash amb el següent codi:

```
#!/bin/bash
export AXIS_HOME=$(pwd)/axis-1_1
export TOMCAT_HOME=$HOME/tomcat
export
CLASSPATH=.:$AXIS_HOME:$AXIS_HOME/lib/axis.jar:$AXIS_HOME/lib/j
axrpc.jar:$AXIS_HOME/lib/saaj.jar:$AXIS_HOME/lib/commons-
logging.jar:$AXIS_HOME/lib/commons-
discovery.jar:$AXIS_HOME/lib/wsdl4j.jar:$TOMCAT_HOME/webapps/ax
is/WEB-INF/lib/xerces.jar
```

- Aquestes rutes estan definides suposant que treballem des del nostre directori base d'usuari.

Per carregar les variables d'entorn en la sessió actual farem servir la comanda:

```
source setCLASSPATH.sh
```

Implementació del servei

Tenim tres fitxers: un per la interfície, que s'anomena *Lloguers.java;* un per la implementació de la classe del servei, que és diu *LloguersImpl;* i un altre per l'aplicació del client, que és diu *lloguerTester.java*.

Interfície del Servei

La interfície del fitxer Lloguers.java únicament conté les capçaleres de les funcions del servei.

```
package lloguers;
public interface Lloguers
{
      // Mètode que ens retorna un valor que indica si el lloguer s'ha
registrat correctament
      public int lloga(String model, String submodel, int dies, int
numv, int desc, double total);
      // Mètode que ens retorna la llista de lloguers serialitzada
      public byte[] llistaLloguers(String user, String password);
}
```

Classe dadesLloguer

Dins del fitxer LloguersImpl.java, que és la implementació del servei, apart de la pròpia classe també tenim la classe *dadesLloguer*, que és la que guarda les dades referents a un lloguer:

```
//Classe que defineix un lloguer - Coneguda pel Client i Servidor
class dadesLloguer implements java.io.Serializable
{
	public String model_vehicle = null;
	public String sub_model_vehicle = null;
	public int dies_lloguer;
	public int num_vehicles;
	public int descompte;
	public double total;

	public dadesLloguer(String mv, String smv, int dl, int nv, int d,
double t)
	{
		model_vehicle = mv;
		sub_model_vehicle = smv;
		dies_lloguer = dl;
		num_vehicles = nv;
		descompte = d;
		total = t;
	}
}
```

- Aquesta classe haurà d'existir tant a la banda del servidor com a la del client, ja que ambdós tracten objectes de la classe dadesLloguer.

Funció lloga(..) del Servei

Entrant ja dintre la implementació de la classe LloguersImpl ens trobem la funció *lloga,* que registra un lloguer amb les dades introduïdes pel client i retorna un valor que indica si s'ha efectuat satisfactòriament o no.
El procediment és el següent:

1. Creem un lloguer nou amb les dades introduïdes pel client
2. Llegim una llista de lloguers del fitxer que usem com a base de dades o creem una llista nova en cas que no existeixi el fitxer. Afegim el nou lloguer a la llista de lloguers
3. Guardem la llista de lloguers actualitzada en un fitxer local
4. Retornem un 1 en cas que s'hagin produït errors en la lectura o escriptura del fitxer, altrament retornem un 0

El codi de la funció lloga(..) seria el següent:

```
public int lloga(String model, String submodel, int dies, int numv, int
desc, double total)
{
	int errors = 0;
	try
	{
		dadesLloguer lloguer = new dadesLloguer(model, submodel, dies,
numv, desc, total);
		//Llegim el fitxer de dades i n'extraiem la llista de lloguers
		errors = llegirFitxer();
		if (errors == 1) return 1;
		//Afegim el nou lloguer
		registreLloguers.add(lloguer);
		//Tornem a guardar la llista (var. global) al fitxer de dades
		errors = escriureFitxer();
	} catch(Exception ex)
	{
		ex.printStackTrace();
		return 1; // Si es produeixen errors informem al client
	}
	return errors;
}
```

En aquesta funció hem fet servir dos mètodes que cal mencionar: un per llegir el fitxer que conté la llista de lloguers, anomenat *llegirFitxer()*; i un per escriure la llista en un fitxer, anomenat *escriureFitxer()*.

Funció llegirFitxer()

Aquesta funció s'encarrega de llegir el fitxer de disc i fer la conversió al tipus d'objecte que ens interessa, en aquest cas, en una ArrayList. En cas que no trobem el fitxer on guardem la llista, creem una llista de lloguers buida. Aquest cas pot produir-se ja sigui perquè el fitxer ha estat esborrat o perquè estem executant el programa per primera vegada.
Per acabar, la funció també ens retorna un valor que indica si s'ha produït algun error durant la lectura.
A continuació en detallem el seu codi:

Funció escriureFitxer()

Aquesta funció s'encarrega d'escriure la llista de lloguers actualitzada amb el lloguer nou en un fitxer local anomenat *lloguers.data* que usem com a base de dades. En cas que es produeixi alguna excepció la funció retorna un 1, que indica que s'ha produït un error. El seu codi és el següent:

```
public int escriureFitxer()
{
	try
	{
		// Write to disk with FileOutputStream
		FileOutputStream f_out = new
FileOutputStream("lloguers.data");

		// Write object with ObjectOutputStream
		ObjectOutputStream obj_out = new ObjectOutputStream
(f_out);
		// Write object out to disk
		obj_out.writeObject (registreLloguers);
	} catch (Exception e) {return 1; }
	return 0;
}
```

Funció llistaLloguers(..) del Servei

A grans trets aquesta funció s'encarrega de comprovar si el nom d'usuari i contrasenya són correctes i retorna la llista de lloguers serialitzada en un array de bytes al client. Per fer el control d'errors posem un valor corresponent a cada tipus d'error a l'array que ens permetrà saber des del client quin error s'ha produït.
El procediment és el següent:

1. Llegim la llista de lloguers del fitxer local *lloguers.data*
2. Creem un array de bytes que ens servirà al mateix temps per emmagatzemar-hi la llista i per fer el control d'errors
3. Comprovem si s'ha produït algun error al llegir el fitxer
4. Comprovem si el nom d'usuari i la contrasenya són correctes
    a. Si són incorrectes posem un valor 0 a l'array i el retornem
5. En cas que siguin correctes, comprovem si té algun lloguer registrat
    a. Si no en té cap, ho indiquem posant un 1 a l'array i el retornem
    b. Si en té, serialitzem la llista de lloguers i la retornem al client

```
public byte[] llistaLloguers(String user, String pass)
{
	int errors = llegirFitxer();
	byte[] retorn = new byte[1];
	if(errors == 1)
	{
		retorn[0] = 3;
		return retorn;
	}
	Iterator iter = registreLloguers.iterator();

	if ((!usr.equals(user)) || (!pw.equals(pass)))
	{
		retorn[0] = 0;
		return retorn; //User-Password incorrecte
	}
	else
	{
		if(!iter.hasNext())
		{
			retorn[0] = 1;
			return retorn; //No hi ha cap
		}
		else retorn = serialize();
	}
	return retorn;
}
```

En aquesta funció hem fet servir el mètode *serialize(),* que serialitza l'objecte que conté la llista de lloguers i el converteix en un vector de bytes.
El concepte "serialitzar" el fem servir tant a la funció llegirFitxer, com escriureFitxer, com en la *serialize* mateixa.

Serialitzar consisteix en salvar l'estat actual d'un objecte en un stream, ja sigui per fer-lo persistent en un fitxer o per enviar-lo per la xarxa en forma de bytes. Aquest objecte serialitzat després pot ser recuperat per construir un objecte equivalent amb les mateixes dades.

Codi de la funció serialize():

```
public byte[] serialize()
{
	try
	{
		// Create an ObjectOutputStream
		ByteArrayOutputStream ba_out =
new ByteArrayOutputStream();
		ObjectOutputStream obj_out =
new ObjectOutputStream (ba_out);
		// Write object to Stream
		obj_out.writeObject (registreLloguers);
		byte[] outBytes = ba_out.toByteArray();
		return outBytes;
	} catch (Exception e) { return null; }
}
```

La limitació que ens imposa WebServices ens impedeix enviar tipus complexes a través de la xarxa, per això ens veiem obligats a fer la conversió de la llista de lloguers a un vector de bytes.

Classe lloguerTester

En la banda del client tenim el fitxer lloguerTester.java, que conté l'aplicació que crida els mètodes del servei explicats anteriorment.
Com hem comentat abans, el client també necessita tenir la classe dadesLloguer per poder interpretar la llista de lloguers que li arriben del servei remot, és per això que l'hem inclòs dins el codi d'aquest fitxer.

Com totes les pràctiques anteriors, la funció *main* de la classe ens ofereix un menú amb les opcions "Llogar vehicle", "Llistar lloguers" o "Sortir" que tenen un número associat. L'usuari introduirà el valor numèric per teclat i en cas que no estigui dintre dels intervals permesos es mostrarà un missatge d'error i el retornarà al menú principal.

La diferència principal es troba en la manera de fer les crides al servei remot.

Funció lloga(..) del Client

En primer lloc tenim la funció lloga(..), que recull les dades del lloguer de l'usuari i les envia fent la crida al mètode del servei remot.
El procediment a seguir és:

1. Creem un servei
2. Obtenim un *stub* del servei
3. Fem la crida al mètode remot lloga(..)
4. Comprovem que no s'hagin produït errors

El codi detallat de la funció és:

```
public static void lloga(String model, String submodel, int dies, int numv,
int desc, double total) throws Exception
{
	// crear un servei
lloguers.ws.LloguersService service = new
lloguers.ws.LloguersServiceLocator();
	// usar el servei per a obtenir un stub del servei
	lloguers.ws.Lloguers lloguers = service.getlloguers();
	// cridar el servei
	int errors = lloguers.lloga(model, submodel, dies, numv, desc,
total);
	if(errors == 1) System.out.println("S'han produit errors");
}
```

Funció llistaLloguers(..) del Client

En segon lloc, tenim la funció llistaLloguers(..) que recull el nom d'usuari i contrasenya i els envia passant-los com a paràmetre del mètode remot. Aquesta crida retorna la llista de lloguers en forma de vector de bytes o retorna un vector d'un byte que conté un valor que es correspon amb un tipus concret d'error.
El procediment a seguir és:

1. Creem el servei
2. Obtenim un stub del servei
3. Fem la crida al mètode remot llistaLloguers(..) passant el nom d'usuari i el password com a paràmetres
4. Fem la comprovació d'errors. Si el vector conté un:
    a. '0' → el nom d'usuari o password són incorrectes
    b. '1' → no hi ha cap lloguer registrat
5. Si no conté errors, deserialitzem la llista de lloguers i la mostrem per pantalla

A continuació mostrem el codi de la funció:

```
      // crear un servei
      lloguers.ws.LloguersService service = new
lloguers.ws.LloguersServiceLocator();

      // usar el servei per a obtenir un stub del servei
      lloguers.ws.Lloguers lloguers = service.getlloguers();

      // cridar el servei
      byte[] bllista = lloguers.llistaLloguers(user, pw);
      try
      {
            //Recollim els errors, enmagatzemats en el primer i unic byte
de bllista
            if (bllista.length == 1)
            {
                  if(bllista[0] == 0) System.out.println("\nError: Nom
d'usuari o password incorrectes\n");
                  else if(bllista[0] == 1) System.out.println("\nNo hi ha
cap lloguer registrat");
                  else System.out.println("\n S'han produit errors\n");
            }
            else //Des-serialitzem la llista de dadesLloger i la mostrem
per pantalla
            {
                  ByteArrayInputStream ba_in = new
ByteArrayInputStream(bllista);

                  // Read object using ObjectInputStream
                  ObjectInputStream obj_in = new ObjectInputStream (ba_in);
                  Object obj = obj_in.readObject();
                  if (obj instanceof ArrayList)
                  {
                        ArrayList llista = (ArrayList) obj;
                        Iterator iter = llista.iterator();
                        text += "\nLlistat de lloguers:\n\n";
                        int num = 1;
                        do
                        {
                              dadesLloguer lloguer = (dadesLloguer)
iter.next();
                              text += "-------------------------\n";
                              text += "<< Lloguer " + num++ + " >>\n";
                              text += "Model: " + lloguer.model_vehicle +
"\n";
                              text += "Sub-model: " +
lloguer.sub_model_vehicle + "\n";
                              text += "Dies de lloguer: " +
lloguer.dies_lloguer + "\n";
                              text += "Nombre de vehicles: " +
lloguer.num_vehicles + "\n";
                              text += "Descompte: " + lloguer.descompte +
"%\n";
                              text += "Import: " + lloguer.total +" €\n";
                        } while (iter.hasNext());
                        System.out.println(text);
                  }
            }
```

Monitoratge amb Tcpmon

El paquet d'Axis ens proporciona una eina anomenada *Tcpmon* que ens permetrà interceptar els missatges de petició i resposta que s'envien client i servidor.
Per engegar-lo farem servir la següent comanda:

```
java org.apache.axis.utils.tcpmon &
```

Un cop obert, li haurem d'indicar el port on escoltarà la connexió, en el nostre cas el 8000; i el port on la dirigirà, el port 8080 que és el que ve per defecte amb Tomcat.

Compilació i desplegament

Un cop tenim tot el codi implementat passem a la fase de compilació i desplegament del nostre servei. Haurem d'empaquetar el nostre programa com un servei web i desplegar-lo en el sistema Axis d'Apache. Els passos a seguir seran:

1. *Java2WSDL:* generar el WSDL a partir de la interficie
2. *WSDL2Java:* generar l'empaquetament de la part de codi referent al servidor i els stubs perquè els clients puguin accedir al servei.
3. *Deploy:* Desplegar el nostre servei a l'Axis de l'Apache

El WSDL ens serveix per escriure serveis d'una manera estructurada.
Per començar creem un directori amb el nom de la nostra aplicació on col·locarem tots el fitxers que hem implementat:

```
mkdir lloguers
```

Compilem la interfície:

```
javac lloguers/Lloguers.java
```

Generem el WSDL a partir de la nostra interfície:

```
java org.apache.axis.wsdl.Java2WSDL -o lloguers.wsdl -
l"http://localhost:8080/axis/services/lloguers" -n urn:lloguers -
p"lloguers" urn:lloguers lloguers.Lloguers
```

Generem l'empaquetament de la part de codi referent al servidor i els stubs perquè els clients puguin accedir al servei:

```
java org.apache.axis.wsdl.WSDL2Java -o . -d Session -s -p lloguers.ws
lloguers.wsdl
```

Després d'haver executat aquestes comandes s'haurà generat un conjunt de fitxers dins del directori /ws que seran els descriptors de desplegament del servei juntament amb l'skeleton i l'stub.

Compilem el codi del servei:

```
javac -source 1.4 lloguers/ws/*.java
```

Empaquetem les classes i les posem dintre el directori de l'Axis:

```
jar cvf lloguers.jar lloguers/*.class lloguers/ws/*.class
mv lloguers.jar ~/tomcat/webapps/axis/WEB-INF/lib/
```

Per acabar, despleguem el servei:

```
java org.apache.axis.client.AdminClient -p8080 lloguers/ws/deploy.wsdd
```

Test

Per provar el servei farem servir el nostre programa client. Primer de tot, el compilarem:

```
javac lloguers/lloguerTester.java
```

Definirem les variables d'entorn executant el nostre script setCLASSPATH.sh
Engegarem el Tomcat:

```
tomcat start
```

Executem el nostre programa client:

```
java lloguers.lloguerTester
```

Després de fer les proves pertinents aquests són els resultats del protocol SOAP que obtenim a través del Tcpmon.

Petició Tcpmon:

```
POST /axis/services/lloguers HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related,
text/*
User-Agent: Axis/1.1
Host: 127.0.0.1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 486

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:llistaLloguers
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:lloguers">
   <in0 xsi:type="xsd:string">pxc</in0>
   <in1 xsi:type="xsd:string">1234</in1>
  </ns1:llistaLloguers>
 </soapenv:Body>
</soapenv:Envelope>
```

Resposta Tcpmon:

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=FA231C195697B2DA5B2CBCDA97BDA127; Path=/axis
Content-Type: text/xml;charset=utf-8
Date: Thu, 29 Oct 2009 23:18:36 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:llistaLloguersResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:lloguers">
   <llistaLloguersReturn xsi:type="soapenc:Array"
soapenc:arrayType="xsd:byte[340]"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <item>-84</item>
    <item>-19</item>
    <item>0</item>
    <item>5</item>
    <item>115</item>
    <item>3</item>
(...)
    <item>108</item>
    <item>120</item>
   </llistaLloguersReturn>
  </ns1:llistaLloguersResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Exemples de interaccions amb l'aplicació i els seus resultats.

```
======================
+ Lloguer de Vehicles +
======================

Escull una opcio:
[1] Llogar vehicle
[2] Llistar lloguers
[0] Sortir

2
Introdueix el nom d'usuari:
pxc
Introdueix la contrasenya:
1234

Llistat de lloguers:

------------------------
<< Lloguer 1 >>
Model: Economic
Sub-model: Diesel
Dies de lloguer: 1
Nombre de vehicles: 1
Descompte: 1%
Import: 54.45 €
------------------------
<< Lloguer 2 >>
Model: Economic
Sub-model: Diesel
Dies de lloguer: 1
Nombre de vehicles: 1
Descompte: 1%
Import: 54.45 €
------------------------
<< Lloguer 3 >>
Model: Economic
Sub-model: Diesel
Dies de lloguer: 1
Nombre de vehicles: 1
Descompte: 1%
Import: 54.45 €
```

```
======================
+ Lloguer de Vehicles +
======================

Escull una opcio:
[1] Llogar vehicle
[2] Llistar lloguers
[0] Sortir

1
Pas 1: Introdueix el numero del model de vehicle
[1] Economic
[2] Semi-Luxe
[3] Luxe
[4] Limusina
2
Pas 2: Introdueix el numero de sub-model
[1] Diesel
[2] Gasolina
2
Pas 3: Introdueix el nombre de dies de lloguer
12
Pas 4: Introdueix la quantitat de vehicles
31
Pas 5: Introdueix el descompte aplicable al lloguer (entre 0 i 100)
50

------------------------------------------
<< Lloguer realitzat satisfactoriament >>
Model: Semi-Luxe
Sub-model: Gasolina
Dies de lloguer: 12
Nombre de vehicles: 31
Descompte: 50%
Import: 13950.0 €
------------------------------------------
```