Interacció entre Aplicacions: objetes distribuits i invocació remota

A la pràctica anterior s'ha vist com extendre la funcionalitat d'un servidor web incorporant servlets que atenen les peticions web. La pràctica utilitzava els mètodes HTTP GET, amb els arguments codificats a la URL (urlencoded), i POST, on els arguments viatgen dins del cos de la petició. En general, els servlets serviran per extendre la funcionalitat de qualsevol servidor, afegint, d'aquesta manera, nous serveis o "comandes" que s'invoquen des d'un formulari HTML.

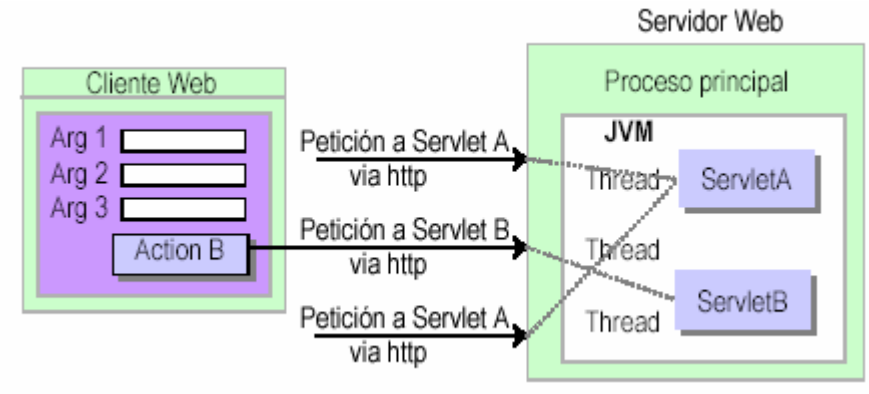

Figura 1.- Entorn típic d'invocació d'un servlet

Un avantatge important dels servlets respecte als CGI, està en que es pot seleccionar la política de servei (threads i instàncies): un cop instanciat un servlet a la màquina virtual Java (JVM) pot servir diverses peticions utilitzant un thread (processos "lleugers") per cada una, o pel contrari, un objecte (amb un sol thread) per cada petició. Un servlet pot també guardar informació que persisteix durant la vida de l'objecte, per exemple les connexions a les bases de dades. A més, la llibreria de servlets facilita i abstreu el pas de paràmetres i la seva codificació. El seguiment de visites successives d'un mateix client (sessions), la transformació de jocs de caràcters entre client i servidor...

El model de client web + formulari HTML / servidor web + extensions (CGI o servlet) és adient per aplicacions a les que l'usuari utilitza un client web i omple un formulari. S'invoca una única operació al servidor amb un conjunt de paràmetres textuals i sense estructura.

De totes maneres, si es desitja comunicar processos o objectes arbitraris, intercanviar objectes o estructures de dades (s'han de serialitzar per a que puguin passar per la xarxa), invocacions bidireccionals, també si es pretén interactuar amb un objecte remot de forma similar a com s'interactua amb un local etc... el model anterior no ho permet.

Aquesta pràctica experimentarà amb objectes distribuits: un mecanisme d'invocació remota dels mètodes d'un objecte, que donen una visió com si l'objecte estés a la mateixa màquina: RMI (Remote Method Invocation).

RMI permet localitzar un objecte remot, enviar i rebre per un canal de comunicació els arguments i resultats d'una invocació i tractar les excepcions de la mànera estàndard de Java. De tot això se n'encarrega la màquina virtual i el programador només ha d'introduir uns quants canvis al seu programa per distribuir-lo. Per tant, es tracta d'un mecanisme de comunicació entre aplicacions que no es basen amb l'ús d'un client web, ni en un formulari HTML, ni amb extensions d'un servidor web, ni amb la utilització d'una connexió HTTP.

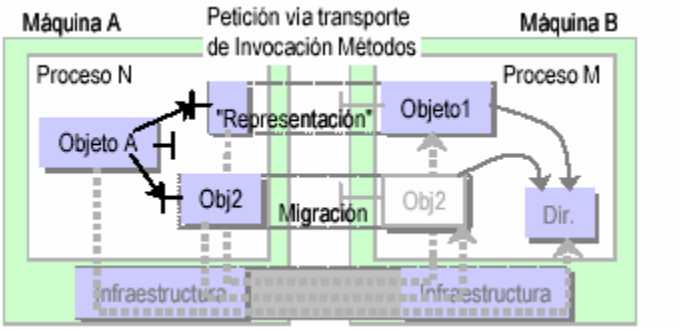

Figura 2.- Mecanismes d'invocació remota entre objectes

A la pràctica també poden trobar-se objectes que estan corrent dins d'un procés client web (Applets, controls Active-X o altes extensions). Aquests processos es comuniquen amb un procés del servidor en una altra màquina que a més en lloc de comunicar-se amb una extensió del servidor web, pot fer-ho amb un altre objecte d'un servidor remot.

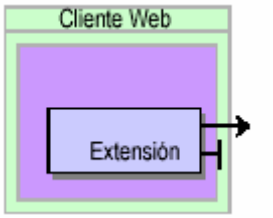

Figura 3.- Un entorn client (rarament servidor) pot ser un objecte que extengui un client web

Es tracta per tant, d'objectes o processos que es comuniquen entre dues màquines separades per una xarxa. Que intercanvien invocacions d'operacions o mètodes, i que porten com a arguments d'entrada i/o sortida objectes, estructures de dades i referències a altres objectes.

- Un representant de l'objecte remot (un objecte representant, stub o referent, que només s'encarrega d'interactuar i intercanviar informació amb el seu representat)
- Un objecte local que ha vingut d'una localització remota per atendre a una petició (ha migrat o s'ha transferit per valor).

L'intercanvi d'informació i la invocació té lloc sobre un transport de dades que pot ser TCP/IP i utilitza un format de representació de dades (serialització i deserialització) especial i diferent a les codificacions textuals dels mètodes GET i POST d'HTTP.

La representació d'informació i invocació pot ser específica d'un llenguatge com en Java-RMI, o independent del llenguatge com amb el cas de CORBA-IIOP (Arquitectura comú de tractament de peticions entre objectes), el que permet comunicar parts escrites en llenguatges diferents.

Invocació Remota de Mètodes (RMI)

RMI forma part del Java estàndard, i ve inclòs amb el JDK de Sun des de la versió 1.1. En aquesta pràctica s'utilitzarà el suport RMI que inclou la JVM2 del JDK 1.2, 1.3 o 1.4 de Sun

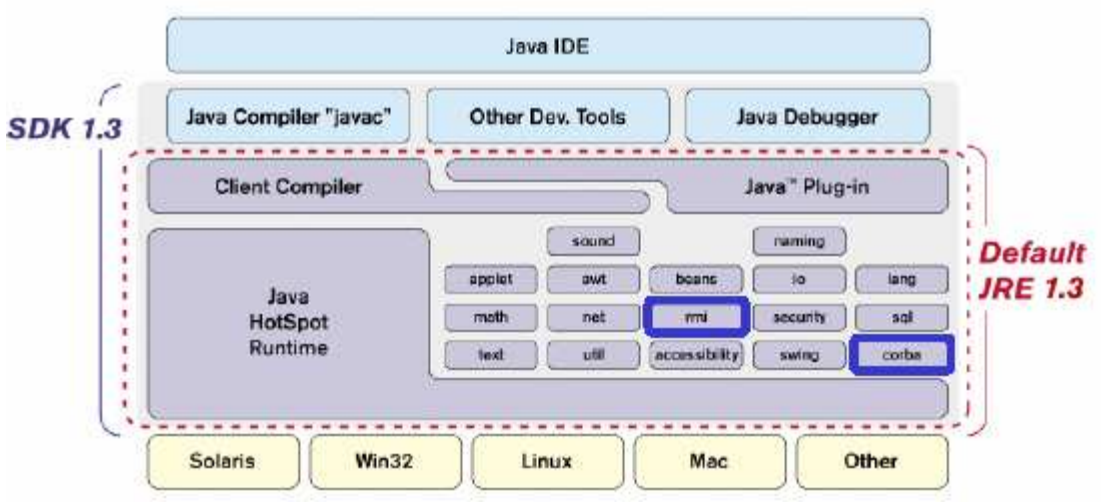

Figura 4.- L'arquitectura de Java 2 SDK, Standard Edition v. 1.3 que inclou RMI i Corba.

Els principals avantatges d'aquest model són els següents:

- *Transparència*: unes mínimes modificacions al codi font permeten distribuir objectes entre diverses màquines. L'entorn d'execució Java (JRE) oculta i facilita la invocació, localització, activació, intercanvi de dades i objectes per la xarxa (serialització).
- *Eficiència*: la invocació remota és més eficient que la transferència HTTP, això és per la forma en que es codifiquen les dades per serialitzar-les, i per la utilització dels canals de comunicació (normalment connexions TCP/IP), per la eficiència i senzillesa de l'stub servidor respecte a un servidor web.
- *Seguretat*: el gestor de seguretat "security manager" i altres components poden controlar quins objectes i quins mètodes poden invocar cada màquina client remota, aixi com restringir quines accions pot dur a terme un objecte sobre la màquina en la que està executant-se i evitar que les fallades d'un procés puguin afectar a la seguretat i integritat de la màquina.
- *Funcionalitat*: s'obté casi la mateixa funcionalitat que ofereix el llenguatge a la comunicació entre objectes remots que entre objectes locals. A més, es poden elegir la forma en que un procés servidor atengui a les peticions, s'activa (s'instancia), s'elimina (per falta de referències) com un objecte local.

El model de programació distribuida amb RMI

En primer lloc es presenta el cas més senzill de comunicació entre dos objectes remots, o al menys que estan corrent en JVM diferents a la mateixa màquina (que per l'efecte és el mateix). És a dir, s'estudiarà le mecanisme d'invocació de mètodes entre objectes remots Java-RMI:

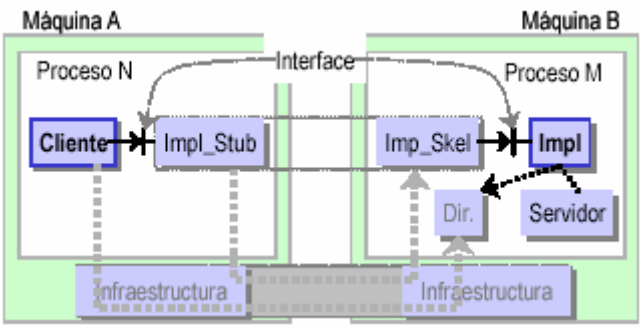

Figura 5.- Un objecte client invoca mètodes d'un objecte remot.

1. Se l'ha de conèixer o localitzar-lo prèviament. Algún procés d'alguna màquina remota haurà instanciat, o al menys, registrat un objecte que ofereix cert servei. Per això s'utilitza un servei de directori: el registre rmi (*rmiregistry*).

   ```
   X c = (X)Naming.lookup("rmi://servidor/ServeiX");
   ```

2. En realitat el que s'obté és una referència al representant local (stub) de l'objecte remot. Quan s'invoca un mètode de l'stub, aquest passa la petició a un representant remot (skeleton) que invoca localment a l'objecte que implementa el mètode. Aquest objecte (Impl) o bé estarà esperant peticions o serà activat automàticament a l'arribar una invocació (similar a un servlet). El resultat de la invocació segueix el mateix camí de tornada.

De cara a l'objecte client, tot és gairebé invisible: ha invocat un mètode d'un objecte local (l'stub), i ha obtingut una resposta local. El que ha passat entre l'stub client, l'stub servidor i l'objecte implementació és invisible i màgic per a ell.

La única cosa diferent és la forma amb la que s'ha instanciat l'objecte (amb `Naming.lookup()` en lloc de `new()`) i que una invocació amb aquest entorn pot generar una excepció nova per les possibles fallades que pot introduir la xarxa (com `java.rmi.RemoteException`).

RMI utilitza classes i interfícies del paquet `java.rmi`. A continuació s'especifica en java la interfície d'un programa exemple:

```
public interface X extends java.rmi.Remote {
  public long incr(long a) throws java.rmi.RemoteException;
  public String mesg(String a) throws java.rmi.RemoteException;
}
```

Cada operació pot fallar i generar una excepció `java.rmi.RemoteException`.

Una classe java ha d'implementar la interfície anterior per a indicar que podrà ser utilitzada remotament. S'ha d'indicar que extén de `java.rmi.server.UnicastRemoteObject`. També s'ha de declarar un constructor per a que consti que es pot generar l'excepció `java.rmi.RemoteException`. Per la resta, es tracta d'una classe normal:

```
public class Ximpl
extends java.rmi.server.UnicastRemoteObject implements X {
  // Constructor per a declarar l'excepció "RemoteException"
  public XImpl() throws java.rmi.RemoteException { super(); }

  public long incr(long a) throws java.rmi.RemoteException {
    return a+1;
  }
  public String mesg(String a) throws java.rmi.RemoteException {
    return "Hola Amic " + a + "!";
```

```
  }
}
```

L'objecte client ha d'instanciar la classe remota utilitzant el servei de noms (`Naming.lookup`) i és convenient tractar les excepcions que puguin produir-se, al menys `RemoteException`:

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
[...]
try {
  X c = (X)Naming.lookup("rmi://servidor/ServeiX");
}
catch (RemoteException re) { }            // El servidor pot fallar
catch (MalformedURLException nbe) { }     // Pot fallar la ref
catch (NotBoundException nbe) { }         // Pot fallar la búsqueda
```

A continuació pot trobar-se el codi d'un programa a la màquina servidora que instancia la classe `XImpl` i la publica al servei de noms (`Naming.rebind` la anuncia i substitueix una anterior si hi existís).

```
import java.rmi.Naming;

public class XServidor {
  public XServidor() { // El mètode constructor
    try {
      X c = new XImpl();
      Naming.rebind("rmi://localhost:1099/ServeiX", c);
    } catch (Exception e) {
      System.out.println("Problema: " + e);
    }
  }
  public static void main(String args[]) {
    new XServidor();
  }
}
```

En resum, els passos a seguir per dissenyar una aplicació distribuida amb RMI són:

1. Definir les funcions de la classe remota com una interfície Java.
2. Escriure una classe d'implementació.
   a. Declarar que implementa al menys una interfície remota.
   b. Definir el constructor de l'objecte remot.
   c. Proporcionar implementacions per als mètodes declarats a la interfície.
3. Escriure la classe servidor que instancia i anuncia l'objecte que implementa la interfície..
   a. Crear i instal·lar un "security manager" (no es farà amb l'exemple).
   b. Crear una o més instàncies de l'objecte remot.
   c. Registrar al menys un objecte remot al registre RMI.
4. Escriure un programa client que utilitzi el servei remot.

Un cop comprovat que la JVM funciona (es pot executar `javac` i `rmic`) es pot copiar el programa anterior i provar que funciona correctament. El codi es pot trobar a la web del laboratori de PXC (http://www.ac.upc.es/docencia/FIB/PXC/lab)

Passos a seguir:

1. Crear un directori de treball. Baixar i descomprimir el paquet .zip amb el programa d'exemple.
2. Compilar el codi Java: `javac *.java`
3. Generar l'stub de l'objecte implementació: `rmic Ximpl`
4. Un cop fet tot això hi haurà els següents fitxers:

Origen	*Resultat**
`X.java`	`X.class`
`XImpl.java`	`XImpl.class`
`Xservidor.java`	`Xservidor.class`
`Xcliente.java`	`Xcliente.class`
`rmic XImpl`	`XImpl_Stub.class`

* Versions anteriors del JDK també generava el fitxer Skel a la banda del servidor, ara ja no es genera.

5. Arrencar el registre d'RMI:

   `rmiregistry`& (Per UNIX) / `start rmiregistry` (Per Windows).

6. Executar el programa servidor, que instancia i anuncia un objecte de la clase `XImpl`:

   `java XServidor`& (Per UNIX) / `start java XServidor` (Per Windows)

7. Executar el programa client, que localitza una instància remota, i invoca els seus mètodes (en realitat els de l'objecte Stub local):

   `java XCliente` (Per UNIX) / `java XCliente` (Per Windows)

Hauria de respondre amb el missatge que s'hagi programat a la classe `XImpl`.

Persistència i activació dels objectes

Un objecte accessible a remotament s'instancia i s'anuncia pel servei de noms un programa que arrenca a la màquina servidora. També podria ser només anunciat i activar-se automàticament si l'objecte Impl es declara com:

```
public class Ximpl
extends java.rmi.server.Activatable implements X {
```

L'objecte servidor respon a peticions i viu fins que ningú el referencii, quan és reciclat pel garbage collector (distribuït). El cicle de vida és:

1. Crear i inicialitzar l'objecte implementació,
2. Anunciar l'objecte al registre (*rmiregistry*),
3. Respondre a invocacions de mètodes d'altres objectes locals o remots,
4. Treure l'anunci i morir, o esperar a ser reciclat (garbage collection).

RMI té un cicle de vida similar als servlet.

Exercicis

Exercici 1:

Modificar `Ximpl.java` de l'exemple anterior per a provar l'efecte d'invocar diverses vegades el mètode des del programa client. S'introduirà un comptador de visites i cada cop que s'invoqui qualsevol mètode d'un objecte de la classe `Ximpl`, s'incrementarà un comptador. El mètode `mesg` ha d'informar a més del número de visites:

```
>java Xclient Luis
5
!Hola Amic Lluís! (visites 2)
```

Exercici 2:

Escriure un programa distribuït pel lloguer de cotxes vist a la pràctica de Servlet i d'Apache CGI. Es tracta de modificar l'exemple anterior per construir un programa client i un programa servidor que puguin executar-se en màquines físiques diferents, o com s'ha comentat abans amb màquines virtuals diferents.

Passos a seguir:

1. Definir la interfície java que implementi la classe `lloguerDeCotxes`.
2. Escriure la implementació de la classe `lloguerDeCotxes` (es pot partir del codi de la pràctica anterior).
3. Modificar el programa servidor de l'exemple per a que instancii i anuncii un objecte de la classe `lloguerDeCotxes`.
4. Modificar el programa client de l'exemple per a que localitzi i invoqui mètodes d'un objecte de la classe `lloguerDeCotxes`. (les ordres poden passar-se com prefereixis: línia de comandes o menú texte).

Referències

1. Java Remote Method Invocation (RMI); http://java.sun.com/products/jdk/rmi/

2. jGuru: Remote Method Invocation: Introduction; http://developer.java.sun.com/developer/onlineTraining/rmi/

3. RMI y CORBA (RMI sobre IIOP); http://developer.java.sun.com/developer/earlyAccess/idlc/