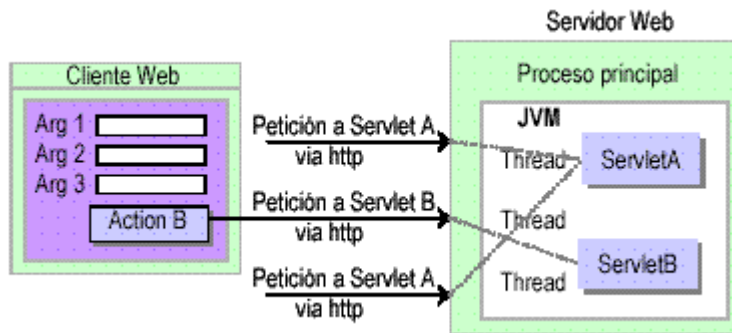


Interacció entre Aplicacions: objectes distribuïts i invocació remota

En l'anterior pràctica s'ha vist com ampliar la funcionalitat d'un servidor web incorporant servlets que atenen peticions web. Usen els mètodes de http GET, amb els arguments codificats en l'URL (urlencoded), i POST, on els arguments viatgen al cos de la petició. En general, els servlets serviren per ampliar la funcionalitat de qualsevol servidor afegint nous serveis o "comandes" que s'invoquen des d'un formulari html.



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

El model client web+formulari html / servidor web+extensions (cgi o servlet) és adequat per a aplicacions en les quals l'usuari utilitza un client web i omple un formulari. S'invoça una única operació al servidor amb un conjunt de paràmetres textuais i sense estructura.

Tanmateix, si es desitja comunicar processos o objectes arbitraris, intercanviar objectes o estructures de dades (cal "serialitzar-los" perquè puguin passar per la xarxa), invocacions bidireccionals, etc. el model anterior no ho permet. En tot cas, si es pretén interactuar amb un objecte remot de forma similar a com s'interactua amb un objecte local, l'anterior no serveix.

En aquesta pràctica s'experimentarà amb objectes distribuïts: un mecanisme d'invocació remota dels mètodes d'un objecte gairebé com si fos a la mateixa màquina. RMI (Invocació Remota de Mètodes) permet localitzar un objecte remot, enviar i rebre per un canal de comunicació els arguments i resultats d'una invocació, i tractar els errors com excepcions de Java. De tot això s'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 en l'ús d'un client web, ni en formularis html, ni en extensions d'un servidor web, ni en l'ús d'una connexió http.

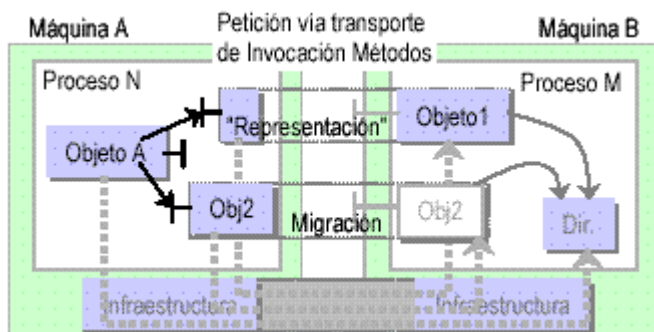


Figura 1: Mecanismes d'invocació remota entre objectes

En realitat, també poden trobar-se objectes que corren dins d'un procés client web (Applets, controls Active-X, o altres extensions), que es comuniquen amb un procés servidor en una altra màquina que a més o en lloc de comunicar-se amb una extensió del servidor web, pot fer-ho amb un altre objecte servidor remot.

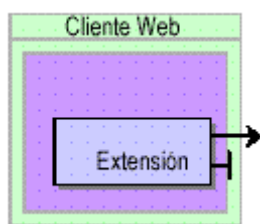


Figura 2: Un entorn client (rarament servidor) pot ser un objecte que amplia un client web

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 que porten com arguments d'entrada i/o sortida objectes, estructures de dades i referències a altres objectes.

Aquests mecanismes solen intentar ocultar la separació al programador (transparència), per a això les invocacions sempre es fan a un objecte local (pròxim al client del servei). Aquest objecte pot ser:

- un representant de l'objecte distant (un objecte representant o stub o referent, que només s'encarrega d'interactuar i intercanviar informació, amb seu representat)
- un objecte local que ha vingut de lluny per atendre 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 textuais dels mètodes GET i POST d'http. La representació d'informació i invocació pot ser específica d'un llenguatge com en Java-RMI (Invocació Remota de Mètodes), o independent de cada llenguatge com en CORBA-IIOP (Arquitectura comuna de tractament de peticions entre objectes) el que permet comunicar parts escrites en llenguatges diferents.

Invocació Remota de Mètodes (RMI)

RMI forma part de Java estàndard i ve inclòs en els JDK de Sun des de la versió 1.1. En aquesta pràctica usarem el suport RMI que inclou la màquina virtual Java2 del JDK 1.2, 1.3 o 1.4 de Sun

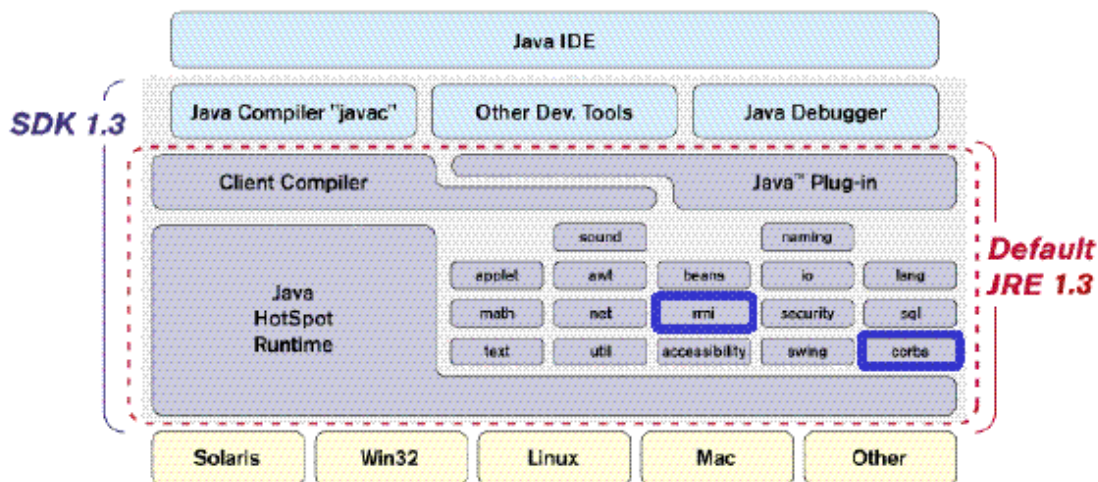


Figura 4: L'arquitectura de "Java 2 SDK, Standard Edition v. 1.3" que inclou rmi i corba.

Els avantatges principals 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 una transferència http per la forma en la qual es codifiquen les dades per a la serialització, per la utilització dels canals de comunicació (usualment connexions TCP/IP), per l'eficiència i senzillesa del stub servidor respecte a un servidor web.
- **Seguretat:** el gestor de seguretat "security manager" i altres components poden controlar quins objectes i quins mètodes pot invocar cada màquina clienta remota, així com restringir quines accions pot dur a terme un objecte sobre la màquina en la qual corre i evitar que els errors d'un procés pugui afectar a la seguretat i integritat de la màquina.
- **Funcionalitat:** s'obté gairebé la mateixa funcionalitat que ofereix el llenguatge a la comunicació entre objectes remots que entre objectes locals. A més, es pot escollir la forma en la qual un procés servidor atén les peticions, s'activa (s'instancia), s'elimina (a falta de referències, com un objecte local).

El model de programació distribuïda amb RMI

En primer lloc es presenta el cas més senzill de comunicació entre dos objectes que corren en diferents màquines o, com a mínim, que corren en diferents màquines virtuals Java (JVM) encara que sigui el mateix computador. És a dir, s'estudiarà el mecanisme d'invocació de mètodes entre objectes remots Java: Java-RMI.

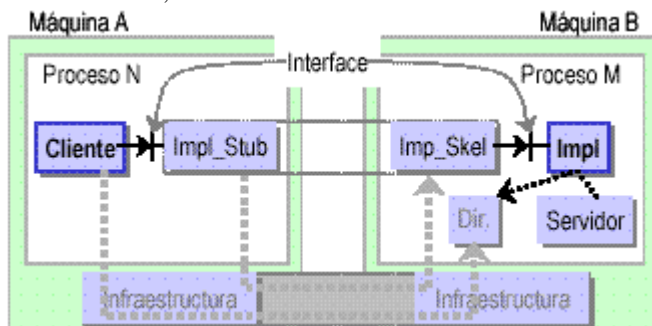


Figura 3: Un objecte client invoca mètodes d'un objecte remot.

Quan un objecte vol comunicar-se amb un objecte remot:

- 1) ha de conèixer-lo o localitzar-lo prèviament. Algun procés a la màquina remota haurà instanciat o com a mínim registrat un objecte que ofereix cert servei. Per a això s'usa un servei de directori: el registre rmi (rmiregistry).

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

- 2) en realitat el que obté és una referència o 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 en 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 ocorregut entre stub client - stub servidor i objecte implementació és invisible i màgic per a ell.

L'únic diferent és la forma en què s'ha instanciat l'objecte (amb `Naming.lookup()` en lloc de `new()`) i que una invocació a aquest objecte pot generar alguna excepció nova pels errors que pugui introduir la xarxa (com `java.rmi.RemoteException`). RMI usa classes i interfícies del paquet: `java.rmi`. A continuació s'especifica en java l'interface d'un programa exemple que es mostrarà:

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

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

Una classe java ha d'implementar la interfície anterior. Per indicar que pot ser usada remotament, cal indicar que amplia `java.rmi.server.UnicastRemoteObject`. També s'ha de declarar un constructor perquè consti que pot generar l'excepció

`java.rmi.RemoteException`. Fora d'això, es tracta d'una classe normal:

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

// Constructor per 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 msg(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, almenys

```
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<A[fallar|resoldre]>
la ref
catch (NotBoundException nbe) { } // Pot fallar<A[fallar|resoldre]> la
cerca

```

A continuació pot veure's el codi d'un programa a la màquina servidor que instancia la classe XImpl i la publica en el servei de noms (Naming.rebind l'anuncia, i substitueix una anterior en cas que n'hi hagués alguna).

```

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 i) {
            System.out.println("Problema: " + e);
        }
    }
    public static void main(String args[]) {
        new XServidor();
    }
}

```

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

1. Definir les funcions de la classe remota com una interfície Java.
2. Escriure la classe implementació.
 - a. Declarar que implementa com a mínim un interfície remot.
 - b. Definir el constructor de l'objecte remot.
 - c. Proporcionar implementacions per als mètodes declarats en 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 fa en aquests exemples).
 - b. Crear una o més interfícies de l'objecte remot.
 - c. Registrar com a mínim un objecte remot en el registre RMI.
4. Escriure un programa client que usa el servei remot.

Una vegada comprovat que la JVM funciona (es pot executar javac i rmic) es pot copiar el programa anterior i provar que funciona correctament.

Passos a seguir:

- 1) Crear un directori de treball. Portar i descompactar el paquet zip amb el programa d'exemple.
- 2) Compilar el codi Java: `javac *.java`
Generar l'stub i skel de l'objecte implementació: `rmic XImpl`
- 3) Després d'això, s'ha de trobar el següent:

Origen	Resultado
X.java	X.class
XImpl.java	XImpl.class
Xservidor.java	XServidor.class
Xcliente.java	XCliente.class
<code>rmic XImpl</code>	XImpl_stub.class
	XImpl_skel.class

- 4) Engegar el registre d'rmi:
`rmiregistry & (unix)/start rmiregistry (windows).`
- 5) Executar el programa servidor, que instancia i anuncia un objecte de la classe XImpl:
`java XServidor & (unix)/start java XServidor (windows)`
- 6) Executar el programa client, que localitza una instància remota, i invoca els seus mètodes (en realitat els de l'objecte stub local):
`java XCliente (unix)/java XCliente (windows)`

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

Persistència i activació dels objectes

Un objecte accessible a distància l'instancia i anuncia en el servei de noms un programa que engega a la màquina servidor. També podria només anunciar-ho i activar-se automàticament si l'objecte Impl es declara com a:

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

L'objecte servidor respon a peticions i viu fins que ningú no el referencii per la qual cosa és "reciclat" pel garbage collector (distribuït). El cicle de vida és:

1. Crear i inicialitzar l'objecte implementació,
2. Anunciar l'objecte en el 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).

Segueix RMI un cicle de vida similar als servlet.

Exercici 1:

Es tracta d'implementar amb RMI un servidor de publicació / subscripció molt senzill. El servidor tindrà els següents mètodes:

```
public void publish(String channel, Hashtable msg) throws
java.rmi.RemoteException;
public Vector pullMessages(String channel) throws
java.rmi.RemoteException;
```

El mètode publish enviarà un missatge al canal especificat. En el missatge podem incloure com camps de la Hashtable el nom de l'usuari i el contingut del missatge. Per a recuperar els missatges d'un canal s'invocarà al mètode pullMessages. És opcional l'usar una classe Missatge en lloc d'una Hashtable.

Crea una classe client Test.java que comprovi el correcte funcionament del servidor de publicació/subscripció.

En definitiva, els fitxers a lliurar i la seva nomenclatura serien els següents:

- PubSub.java: interface java que especifiqui els mètodes.
- PubServer.java: implementació de la interfície anterior.
- PServer.java: programa servidor que faci una instància i publiqui un objecte de la classe PubSub.
- PClient.java: programa client que comprovi el funcionament del servidor.
- Opcionalment, Missatge.java: classe missatge, si es desitja usar.

A més, podeu crear més classes i objectes auxiliars, queda al vostre criteri la ubicació o la creació del package adequat

Exercici 2:

Es tracta de modificar el servidor de l'exercici 1 per a afegir el següent mètode:

```
public void subscribe(String channel, Subscriber callback) throws  
java.rmi.RemoteException;
```

Aquest mètode registrarà un subscriptor interessat en missatges d'aquest canal. Cada vegada que s'envii un missatge a

aquest canal (publish) el servidor haurà de notificar a tots els subscriptors registrats en el mateix.

Recordeu que la classe Subscriber, per a poder ser notificada (pas per referència) ha de ser també un objecte RMI (extends java.rmi.Remote).

Crea una classe client PClient2.java que comprovi el correcte funcionament del servidor de publicació/subscripció.

Els fitxers a lliurar i la seva nomenclatura serien els següents:

- *PubSub.java*: interface java que especifiqui els mètodes.
- *PubServer.java*: implementació de la interfície anterior.
- *PServer.java*: programa servidor que faci una instància i publiqui un objecte de la classe PubSub.
- *Subscriber.java*: interface java que especifiqui els mètodes del subscriptor.
- *Callback.java*: implementació de la interfície anterior.
- *PClient.java*: programa client que comprovi el funcionament del servidor.
- *PClient2.java*: programa client que comprovi el funcionament del servidor amb dos o més canals.
- Opcionalment, *Missatge.java*: classe missatge, si es desitja usar.

A més, podeu crear més classes i objectes auxiliars, queda al vostre criteri la ubicació o la creació del package adequat.

NOTA: No s'han d'usar fitxers ni bases de dades.

Metodologia de Lliurament

S'ha de lliurar com a mínim:

- El codi font de les solucions.
- Els arxius binari corresponents (.class)
- Captura de les imatges amb el resultat de l'execució dels programes.
- Informe detallat contestant als punts del fitxer "informe.txt"

Format del lliurament

S'ha de lliurar tot junt (organitzat el contingut en carpetes com correspongui) en un sol arxiu comprimit (en format zip, tar+gzip, o tar+bzip2). L'arxiu s'ha de anomenar:

Cognom1_Cognom2_Nom-ASD-P1.ext

(on ext pot ser: zip, tgz, tar.gz, tbz2, o tar.bz2)

Estructura de fitxers del lliurament

L'estructura dels fitxers del lliurament ha de ser la següent:

/RMI

LLEGUEIXME.TXT	→comentari sobre el lliurament de RMI
/doc	→documentació
/src	→fonts
COMPILAR.BAT	→fitxer bat que compili la pràctica.
/bin	→executables.
EXECUTAR.BAT	→fitxer bat que executi la pràctica: el servidor i el cliente. I deixi en pantalla els resultats de l'execució
EXECUTAR.TXT	→explicació de l'execució
/img	→ captures que il·lustrin el funcionament
IMATGES.TXT	→descripció de les captures que s'adjunten

Referències

1. Java Remote Method Invocation (RMI); <http://java.sun.com/products/jdk/rmi/>
2. jGuru: Remote Method Invocation: Introduction;
3. <http://developer.java.sun.com/developer/onlineTraining/rmi/>
4. RMI i CORBA (RMI sobre IIOP); <http://developer.java.sun.com/developer/earlyAccess/idlc/>