Code mobile : Un exemple

DEA - Modélisation et implémentation des systèmes complexes -

D. Olivier

Année 2002/2003

1 Introduction

Avec la machine virtuelle Java, la sérialisation et RMI, il est possible de réaliser du code mobile. C'est l'objectif de ce document de présenter un tel exemple.

C'est un exemple très simple, dans lequel on définit un code mobile, donc serialisable, un serveur qui met à disposition ce code mobile et un client éventuellement distant. Le serveur est un objet distribué RMI qui pour l'exemple développé va créer son propre registre RMI.

2 Mise en œuvre

On définit quatre classes Java :

- `Agent` qui correspond au code mobile, cette classe stocke une information (`donnee`) initiale qui est succeptible d'être modifié par les clients, lorsqu'ils éxécutent ce code mobile. On stocke également la machine d'origine.
- `AgentFactory`, c'est l'interface de l'objet serveur distribué. Cette interface est invoquée par le client et les appels de méthodes sont gérés par RMI.
- `AgentFactoryImpl`, c'est l'implementation de l'interface. Elle crée une instance de la classe Agent (objet mobile) et l'envoie au client sur demande `getAgent()`. Cette classe accepte également des instances de la classe Agent et récupère la modification de donnée.
- `AgentClient` qui comme son nom l'indique est le client du serveur. Il récupère tout d'abord la référence de l'objet serveur (`AgentFactory`) dans le registre RMI, puis fait migrer l'objet mobile `Agent` et exécute les méthodes de l'objet mobile.

2.1 La classe `Agent`

Définition du code mobile :

```
package mobile;
import java.net.*;                                  // Pour obtenir le nom des machines

public class Agent implements java.io.Serializable  // Indispensable pour la mobilité
{
    int donnee;                                     // L'info stockée
    String origine;                                 // La machine d'origine
    public Agent() throws UnknownHostException      // Exception pouvant etre levée
    {
        origine = InetAddress.getLocalHost().toString();
    }

    public void calcule()                           // La fonction du code mobile
    {
```

```
        donnee += 113;
    }

    public int getDonnee()
    {
        return donnee;
    }

    public void setDonnee(int donnee)
    {
        this.donnee = donnee;

    }

    public String douViensTu()
    {
        return origine;
    }

    public String ouEsTu() throws Exception              // Pour connaître la machine où se
                                                         // situe le code qui s'exécute
    {
        return InetAddress.getLocalHost().toString();
    }
}
```

2.2 La classe AgentFactory

Interface permettant la mobilité :

```
package mobile;
import java.rmi.*;

public interface AgentFactory extends Remote
{
                // Pour faire migrer le code (à la demande)
    public Object getAgent() throws Exception;
                // Pour récupérer le résultat du traitement du code mobile
    public void   setAgent(Object a) throws Exception;
}
```

On peut remarquer que l'on manipule que des `Object`, ceci permet de la généricité. D'une part, on peut de ce fait faire migrer tout objet serialisable, d'autre part cela ne nécessite pas du coté client de connaître la classe `Agent` au moment de la compilation.

2.3 La classe AgentFactoryImpl

Le serveur est donc la mise en œuvre de l'interface :

```
package mobile;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class AgentFactoryImpl extends UnicastRemoteObject implements AgentFactory
```

```
{
    int data = 0;                          // Sert à fixer la donnée initiale véhiculée
                                           // par le code mobile et à récupérer la
                                           // valeur lorsque le code revient en son sein.

    public AgentFactoryImpl() throws RemoteException
    {
        super();
    }

    public Object getAgent() throws Exception                    // Permet la migration
    {
        Agent a = new Agent();
        a.setDonnee(this.data);
        return a;
    }

    public void setAgent(Object a) throws Exception              // Récupération du résulat
    {
        this.data += ((Agent) a).getDonnee();
        System.out.println("Maintenant l'agent est en :"+
                           ((Agent) a).ouEsTu());
    }

    public static void main(String[] argv)                       //argv[0] == port
    {
        try
            {
                LocateRegistry.createRegistry(Integer.parseInt(argv[0])); // Création d'un registre RMI
                System.out.println("Registre RMI lancé");
                LocateRegistry.getRegistry(              // Enregistrement du serveur dans le registre
                           Integer.parseInt(argv[0])).bind("AgentFactory",new AgentFactoryImpl());
                System.out.println("Agent factory enregistré");
            }
        catch(Exception e)
            {
                System.out.println("Exception interceptée :" + e);
                e.printStackTrace();
            }
    }
}
```

2.4 La classe AgentClient

Dans cette classe, on utilise la réflexivité de Java pour connaître la classe et les méthodes de l'objet qui a migré. Une fois les méthodes connues ont peu les invoquer.

```
package mobile;

import java.rmi.*;
import java.rmi.server.*;
import java.net.*;

import java.lang.reflect.*;              // Pour la reflexivité des classes
```

```
public class AgentClient
{
 public static void main(String[] argv) // argv[0]==machine serveur
                                        // argv[1]==port
 {
  try
    {
     // Phase d'initialisation :
     //      1 Liaison à l'objet distribué serveur de code mobile
     //      2 Migration du code sous la forme d'un objet

        String url = "rmi://"+argv[0]+":"+argv[1];
        System.out.println("-->␣Lancement␣du␣client");
                        // Pour controler les chargements dynamiques (fichier agents.policy)
        System.setSecurityManager(new RMISecurityManager());
                        // Récupération de la référence du serveur
        AgentFactory af = (AgentFactory) Naming.lookup(url+"/AgentFactory");
        System.out.println("-->␣On␣a␣obtenu␣la␣référence␣du␣Serveur");
        Object a = af.getAgent();                       // Migration du code, a est l'instance


     // On utilise le code mobile obtenu

        Class cl = a.getClass();                        // On récupère la classe de l'objet qui
                                                        // a migré.
        System.out.println("Classe␣de␣l'objet␣obtenu␣:␣"+ cl.getName());
        System.out.println("-->␣L'agent␣mobile␣est␣obtenu");
        Method ou = cl.getMethod("douViensTu", null);   // On cherche la méthode douViensTu()
                                                        // puis on invoque la méthode
        System.out.println("L'agent␣viens␣de␣␣␣␣␣␣␣␣:␣"+ ou.invoke(a, null));
        ou = cl.getMethod("ouEsTu", null);
        System.out.println("Maintenant␣l'agent␣est␣en:␣"+ ou.invoke(a, null));
        Method getDonnee =  cl.getMethod("getDonnee", null);
        System.out.println("Donnee␣avant␣calcul␣␣␣␣␣␣:␣" + ((Integer)getDonnee.invoke(a, null)).intValue());
        cl.getMethod("calcule", null).invoke(a, null);
        System.out.println("-->␣L'agent␣mobile␣calcule");
        getDonnee =  cl.getMethod("getDonnee", null);
        System.out.println("Donnee␣après␣calcul␣␣␣␣␣␣:␣" + ((Integer)getDonnee.invoke(a, null)).intValue());


     // On renvoie le code au serveur

        af.setAgent(a);
        System.out.println("-->␣L'agent␣est␣reparti");
    }
  catch (Exception e)
    {
        System.out.println("Exception␣interceptée␣" + e);
        e.printStackTrace();
    }
 }
}
```

2.5 Compilation et exécution

– Coté serveur :

```
[serveur]$ ls
mobile/
[serveur]
[serveur]$ ls mobile/
Agent.java  AgentFactory.java  AgentFactoryImpl.java
[serveur]$ javac mobile/*.java
[serveur]$ rmic -v1.2 -d . mobile.AgentFactoryImpl
[serveur]$ ls mobile/
Agent.class             Agent.java
AgentFactory.class      AgentFactory.java
AgentFactoryImpl.class  AgentFactoryImpl.java
AgentFactoryImpl_Stub.class
```

Au niveau compilation, il n'y a plus rien à faire, cependant il reste un problème au niveau du stub. En effet, le client doit posséder le code de la classe de stub, de même notre objet distribué serveur renvoie des objets que le client n'est pas supposé connaître initialement puisque le code doit migrer. Une manière simple de procéder est de copier les `.class` correspondant sur le système de fichier local du client, un comble pour du code mobile ! Nous n'allons pas procéder de cette façon, mais rendre ces classes disponibles coté serveur. Cela nécessite que du coté serveur, il y ait un serveur `http`. Dans ce cas, on place les fichiers `Agent.class` et `AgentFactoryImpl_Stub.class`, à l'aide d'un lien symbolique par exemple, dans un répertoire accessible par `http`.

ex : `~/public_html/rmi/download/mobile`

Il ne nous reste plus qu'à lancer le serveur.

```
[serveur]$ java -Djava.rmi.server.codebase=http://unemachine/~untel/rmi/download/\
> mobile.AgentFactoryImpl 1099 &
[1] 6674
[serveur]$ Registre RMI lancé
Agent factory enregistré
```

Le serveur tourne donc maintenant, il reste à s'occuper du client.

– Coté client :

```
[client]$ ls
agents.policy  mobile/
[client]$ ls mobile
AgentClient.java  AgentFactory.java
javac mobile/AgentClient.java
[client]$ ls mobile/
AgentClient.class   AgentClient.java
AgentFactory.class  AgentFactory.java
[client]$
```

On remarquera qu'il faut au client et au serveur l'interface `AgentFactory.java`. Ensuite on peut lancer l'exécution, mais il faut au préalable définir un fichier `*.policy` pour le gestionnaire de sécurité.

```
grant
{
 permission java.net.SocketPermission
        "*:1024-65535", "connect";
 permission java.net.SocketPermission
        "*:80", "connect";
};
```

On autorise les ports supérieurs à 1024 et le port `http`.

```
[client] java -Djava.security.policy=agents.policy mobile.AgentClient\
```

```
> unemachine 1099
--> Lancement du client
--> On a obtenu la référence du Serveur
Classe de l'objet obtenu  : mobile.Agent
--> L'agent mobile est obtenu
L'agent viens de          : unemachine/193.48.XYZ.ABC
Maintenant l'agent est en : autremachine/193.48.XYZ.EFG
Donnee avant calcul       : 113
--> L'agent mobile calcule
Donnee après calcul       : 226
--> L'agent est reparti
```

Du coté serveur, on obtient alors :

```
[serveur]
Maintenant l'agent est en : unemachine/193.48.XYZ.ABC
```