Remote Method Invocation en Java (RMI)

Modélisation et construction des applications réparties (Module M-4102C)

J. Christian Attiogbé

Fevrier 2015

RMI : extension objet de RPC

Le mécanisme RMI est l'extension aux concepts OBJET, du *Remote Procedure Call (RPC)*.

Principe général du RMI

Le RMI permet, de la même manière que le RPC, d'appeler une méthode sur une machine distante. La machine distante offre à travers une application, un objet O accessible à distance ; les méthodes de O sont ainsi appelées à distance par des applications dites clientes.

La machine distante doit être connue des applications clientes.
L'objet offert à distance et ses méthodes doivent être connues des machines clientes.

Principe du Remote Procedure Call

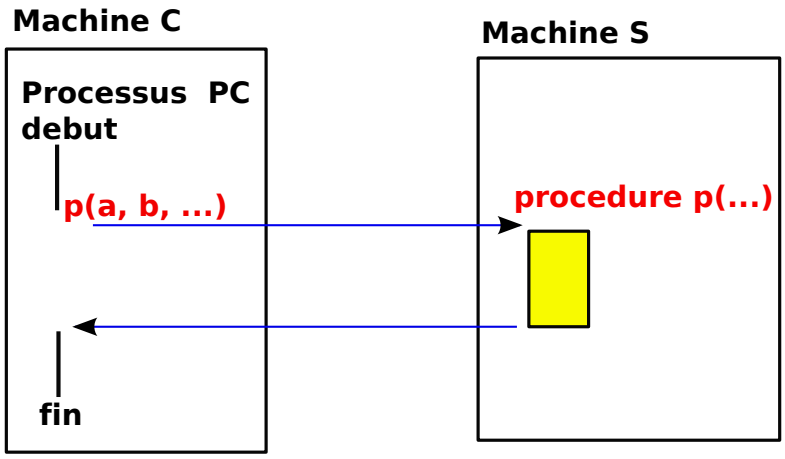

Figure : Principe du RPC

Intérêt et difficultés du RPC

- Structuration simple du code du client
- Abstraction du réseau
- Facilité des mises à jour
- Complexité de la gestion des paramètres
- Complexité de la sémantique
- Difficultés de gestion des pannes (clients ou serveur)
- Gestion des aspects hétérogénéité

Mise en œuvre du RPC

L'interaction RPC est mise en œuvre par une structuration en couche.

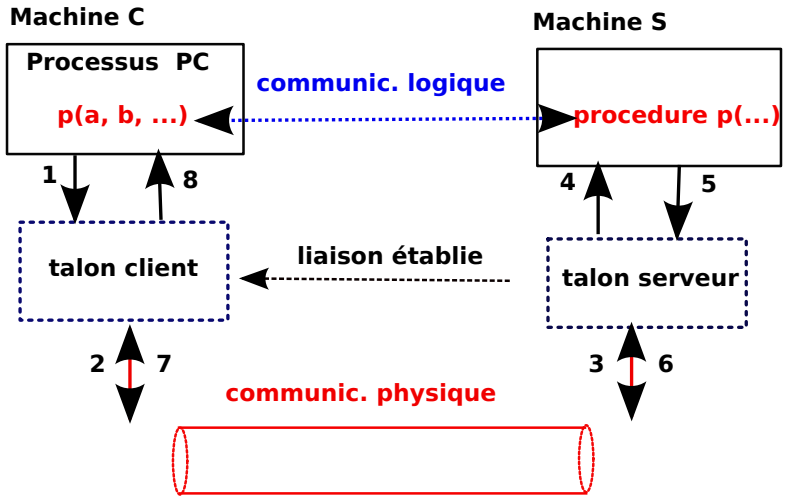

Figure : Interaction client/serveur avec RPC

Pour que cela marche, le client doit donc connaitre l'environnement du

Mise en place de l'environnement du serveur

La procédure appelée à distance est au préalable publiée via un annuaire (serveur de noms : nomProcédure adrServeur, numPort). L'annuaire sera interrogé par les clients avant de pouvoir communiquer avec les serveurs.

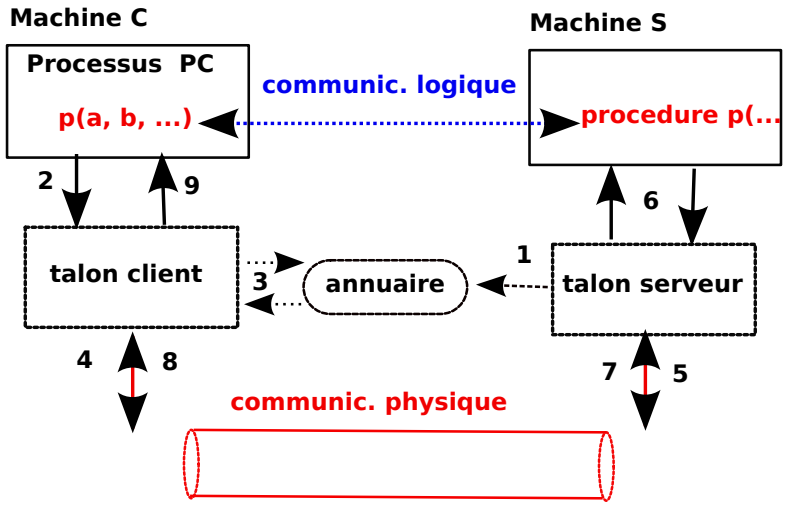

Figure : Principe du RPC

Remote Method Invocation en Java

En Java, une application ordinaire s'exécute dans la machine virtuelle locale.
Dans le cadre des applications réparties, le mécanisme RMI permet d'appeler et d'exécuter une méthode dans une machine virtuelle distante, différente de celle de l'application appelante.

Principe du mécanisme RMI en Java

- Une application objet Java offre sur une machine distante un objet accessible à distance et dont les méthodes peuvent être invoquées.
- L'application serveur, construit un objet accessible à distance, et l'enregistre auprès d'un serveur rmi, pour le rendre visible des applications clientes.
- Java offre pour cela le package java.rmi avec plusieurs sous-packages java.rmi.server , java.rmi.dgc, ...

Principe du mécanisme RMI en Java

Au cœur du principe Java RMI se trouvent,

- d'une part un objet accessible à distance OD (créé par le côté serveur),
- d'autre part
  - la classe stub (talon) qui représente l'objet OD côté client et
  - la classe skeleton (squelette) qui représente l'objet OD côté serveur.
- L'indépendance et la transparence des serveurs est réalisée par le mécanisme d'**annuaire rmi** qui contient les objets publiés et les serveurs qui les publient.
- L'exclusion mutuelle des accès aux objets est assurée par le mécanisme de synchronisation de Java
- Les applications clientes s'adressent d'abord à l'annuaire rmi pour pouvoir établir le lien avec les serveurs d'objets.

Mécanisme Java RMI : Serveur

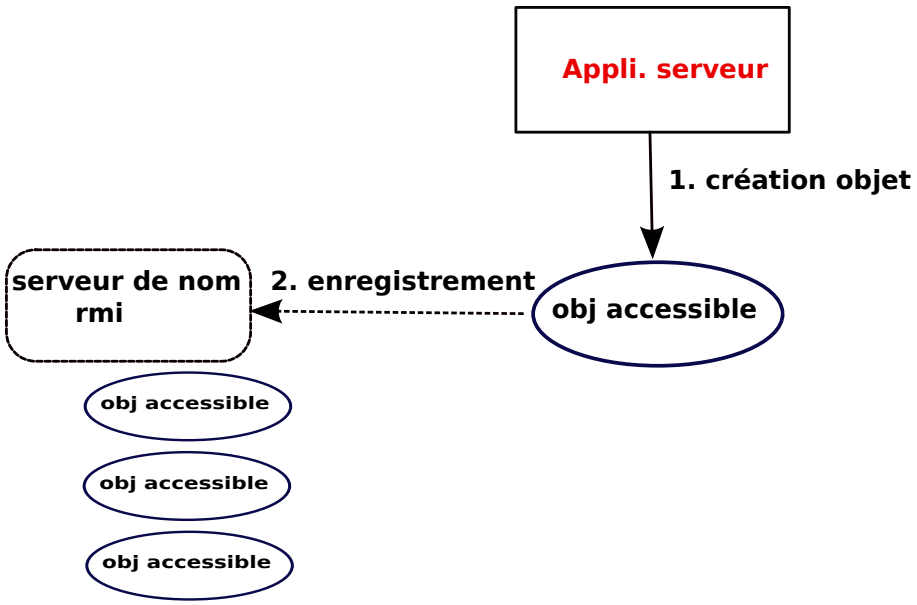

Figure : Création et publication d'un objet accessible à distance

Mécanisme Java RMI : Client

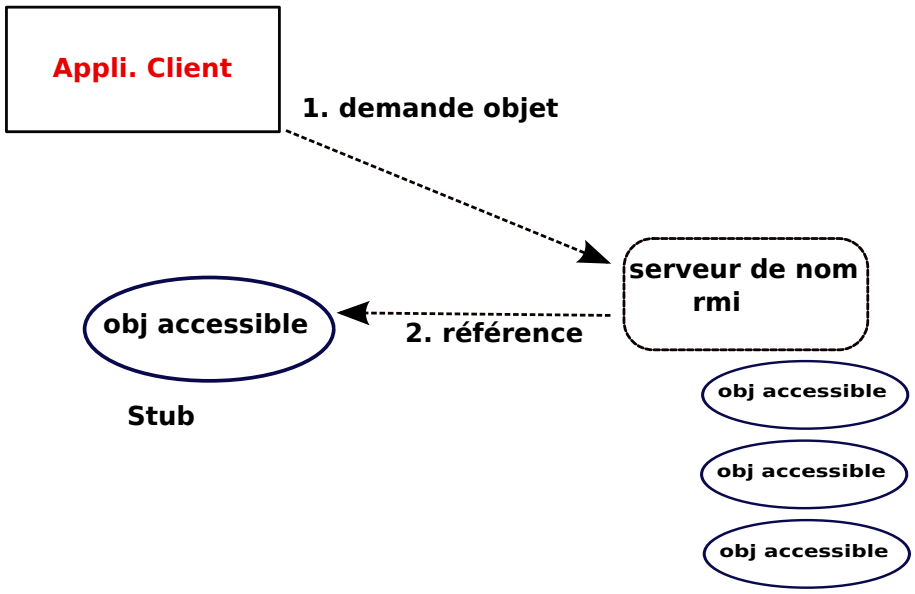

Figure : Récupération d'une référence à l'objet distant

Méthode pour le client/serveur RMI en Java

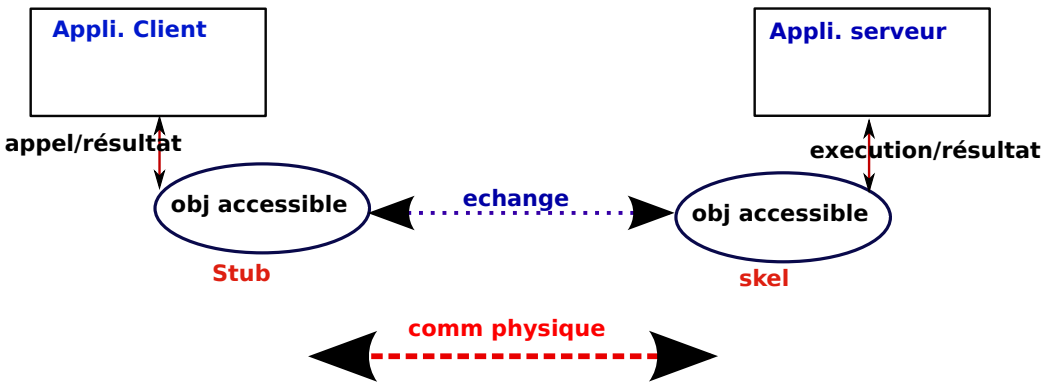

Figure : Interaction RMI via stub et skel

Démarche : côté d'une application serveur RMI

1. Créer un objet qu'on souhaite rendre accessible à distance (`nomObjetAD`)
2. Enregistrer l'objet créé dans le registre de noms d'un serveur, en attribuant à l'objet une URL spécifique : `rmi://nomOuAdresseServeur/nomObjetAD`

Pour que l'application serveur puissent enregistrer des objets et que les clients puissent accéder aux objets à distance, le registre de noms RMI (annuaire rmi) doit d'abord être exécuté sur la machine serveur (il utilise le port 1099).

```
rmiregistry &
```


Création d'un objet accessible à distance

Tout objet accessible à distance hérite de la classe Java.rmi.Remote.
Il s'agit d'une interface qui ne contient aucune méthode.
On spécifie l'interface de l'objet accessible à distance en héritant de la classe Java.rmi.Remote puis en définissant les méthodes qu'on souhaite rendre accessibles aux clients.
Les communications réseaux pouvant échouer, on doit gérer les exceptions sur toutes les méthodes.

```
/* fichier .java de la classe monInterfaceDistant : Serveur */
import java.rmi.*;
public interface monInterfaceDistant extends Remote {
    /* une méthode de test */
    public String getCode() throws RemoteException;
}
```

Implantation de l'interface côté serveur

Le serveur implémente son interface, avec le code java nécessaire.
Le mécanisme Java RMI offre la classe `UnicastRemoteObject` qui contient les primitives permettant l'appel de l'objet distant ;
par exemple, l'appel par le stub du client qui est unique (le stub n'a qu'une seule référence sur l'objet).
La classe qui implémente l'interface doit hériter de la classe `UnicastRemoteObject`.
Le mécanisme Java RMI offre l'utilitaire rmic pour générer le talon (stub) et le squelette (skeleton) à partir du bytecode (issu de la compilation java de la classe objet distant)

```
javac monInterfaceDistant.java // compilation interface
javac monObjetAccessibleDistance.java // compilation implémentation
rmic monObjetAccessibleDistance // génération stub et skel (...stub.class
...skel.class)
```


Implantation de l'interface côté serveur

```
/* .java de la classe monObjetAccessibleDistance : Serveur */
import java.rmi.*;
import java.rmi.server.*;
public class monObjetAccessibleDistance
          extends java.rmi.server.UnicastRemoteObject
          implements monInterfaceDistant {
  /* constructeur */
  public monObjetAccessibleDistance () throws RemoteException
      { super ();}
  public String getCode() throws RemoteException
      { return "camarche"; }
}
```

Création et publication d'un objet par le serveur

Le mécanisme Java RMI offre la méthode `rebind` via la classe Naming pour enregistrer un objet dans le registre de nom du serveur.

```
import ...
public class testServeurRmi {
   ... //constructeur de la classe + Exception
/* Création et enregistrement d'un objet par l'appli serveur */
   public static void main (String[] args) throws RemoteException,
      MalformedURLException, UnknownHostException {
      monObjetAccessibleDistance testSRVTPrmi =
            new monObjetAccessibleDistance(); // création de l'obj
     try{Naming.rebind("rmi://"+java.net.InetAddress.getLocalHost()+
            "1099"+"/testSRVTPrmi",testSRVTPrmi);
     } catch (MalformedURLException ee) {
           throw new MalformedURLException("Pbm de url");
     } // idem pour les autres catch
   } }
```


Mise au point Serveur : les import

```
/**
* TP Java RMI - Appli Serveur 2014/2015
*/

import java.rmi.*;
import java.net.MalformedURLException;
import java.net.UnknownHostException;
public class testSRVTPrmi {
    static public void testSRVTPrmi() throws RemoteException {
       ...
}
```

Méthode d'accès à l'objet, par le client RMI

- L'application client, s'adresse au serveur de noms auprès duquel l'objet est enregistré ; elle utilise le nom complet `rmi://adrServeur:numPort/nomObjet/`
- Il récupère un stub (talon) de l'objet, à l'aide de la méthode `lookup` de la classe `Naming`.
- L'objet récupéré est de la classe Remote.
- L'objet doit être casté en la classe Interface qui définit les méthodes utilisables de l'objet distant ; ensuite on appelle les méthodes.

Méthode d'accès à l'objet, par le client RMI

```
import java.rmi.*; /* clientRmi.java : coté client */
...
public class clientRmi {
    static public void clientRmi() throws RemoteException {
    }
    public static void main (String[] args)
    throws RemoteException,MalformedURLException, NotBoundException{
    try {
    Remote monStub = Naming.lookup("rmi://"+
       java.net.InetAddress.getLocalHost()+"1099"+"/testSRVTPrmi");
    if (monSTub instanceof monInterfaceDistant) {
      String lecode = ((monInterfaceDistant)monStub).getCode();
      System.out.println("code recupere = "+lecode);
    }
    catch (NotBoundException ee) {
        throw new NotBoundException("Pbm de liaison");}
    } // idem pour les autres catch exceptions
}
```

Conclusion

Java RMI

La technologie RMI (ici Java RMI) est un intergiciel (*middleware* basé sur les objets et qui permet de réaliser des applications distribuées en Java.
Elle nécessite de bien penser son application et le partage de données (les objets acessibles à distance) à l'avance.

- Technologie mature
- Bien intégrée dans les environnements Java (donc NON interopérable)
- Autre middleware non basé sur les objets : Message-Oriented Middleware (MOM)

Ouvertures

RMI dans d'autres environnements

- RPyC (Rempote Python call) `http://rpyc.readthedocs.org/`,
- Scala + Java ? (puisque Scala mange le Java, on doit pouvoir lui faire gober le Java RMI)
- ...

RMI et interopérabilité

Les Interopérables :

- Advanced Message Queuing Protocol `http://www.amqp.org/`
- CORBA
- Babel Middleware (pour le calcul scientifique) `https://computation.llnl.gov/casc/components/#page=home`
- SWIG `http://www.swig.org/`,
- ...

Exercices : à volonté

Modélisation à l'aide de Réseaux de Petri +

- Client/Serveur de ressources (via le nom de la ressources)...
- Clients/Serveur de ressources avec des paramètres
- Clients/Serveur de classement, score, ... de matchs (clients = équipes)
- ...

Références

- Sacha Krakowiak, Systèmes et Applications Réparties, Université Joseph Fourier
- A.S. Tannenbaum, *Distributed Operating Systems* Prentice Hall.
- Martin Quinson, Cours de Programmation d'Applications Réparties
- A.D. Birell and B.J. Nelson, *Implementing remote procedure calls* ACM Trans. on Comp. Syst., vol. 2(1), 1984.
- Satyanarayanan, H. Siegel *Parallel communication in a large distributed environment*, ACM Trans. On Comp, vol 39(3), 1990
- `http://docs.oracle.com/javase/tutorial/rmi/overview.html`
- `http://stackoverflow.com/questions/28892535/java-rmi-client-server-on-different-machines`