RPC
Remote procedure call

L'appel de méthode à distance a pour objectif de fournir l'illusion d'un appel de méthode local.

Pour cela il faut définir un passage des arguments et du résultat.

De plus il faut avoir un programme en attente d'un appel

RPC

L'idée des RPC date de 1976 RFC 707,

La première utilisation Xerox 1981 « Courier »

La version la plus populaire est celle de SUN,

« ONC RPC » sur Unix

Microsoft a proposé un mécanisme similaire

MSRPC

RPC

Les RPC permettent une architecture distribuée sous forme client-server

Un client envoie un message à un serveur qui effectue l'opération et renvoie un résultat

Par rapport à un appel local, il y a des risques de pertes ou de pb de transmission

ONC RPC

Open Network Computing Remote Procedure Call

Les ONC RPC ou Sun RPC on été conçu par Sun pour « NFS-Network File System »

Il utilise les protocole UDP (en général) et TCP

Il adapté au langage C

Il est décrit dans la RFC 1831

ONC RPC

Les mécanismes d'authentification sont décrit dans les RFC 2695, RFC 2203, RFC 2623

En 2009, Sun a proposé une license BSD clause 3

Les données sont envoyées au format XDR

L'utilisation des serveurs est effectué via le port mapper

RPC-Port Mapper

Le port mapper ou rpc.portmap ou portmap

Est un service qui permet de faire le lien entre un service et un serveur en utilisant un mécanisme d'identification

L'objectif de ce service est de minimiser le nombre de port utilisé (111)

XDR
External Data Representation

Au départ format de Sun 1987

XDR est un format de l'IETF (Internet Engineering Task Force) de 1995 pour la sérialisation des données.

Les types sont : boolean, char (8 bit integer/character),int (32 bit integer), hyper (64 bit integer), float, double, quadruple (new in RFC1832), enumeration, structure, string, fixed length array, variable length array, union,opaque data

XDR
External Data Representation

XDR est décrit dans un ensemble de document :

RFC 1014 - 1987

RFC 1832 - 1995 – ajout quadruple précision float

RFC 4506 - 2006 – Augmentation de la sécurité

NFS
Network File System

NFS est un système de fichier de type client/serveur

Il a été développé par Sun en 1984

Il repose sur les RPC

La version 2 mars 1989

La version 3 1995

La version 4 2000

RMI
Remote Method Invocation

L'appel de méthode à distance a pour objectif de fournir l'illusion d'un appel de méthode local.

Pour cela l'objet distant doit être représenté localement par un proxy.

De plus l'objet distant doit avoir un mécanisme pour prendre en compte les demandes.

RMI

L'utilisation des RMI se fait souvent sous la forme d'Application Client Serveur.

Plus généralement on a une application distribuée.

Ce type d'application permet de répartir les calculs et les services.

RMI

Une application distribué de type RMI doit offrir les éléments suivants :

- localisé les objets distants,
- communiquer avec les objets distants,
- transmettre le bytecode correspondant à un objet.

La couche réseau devient transparente.

Exemple Client/Serveur

Une application va permettre l'exécution d'une tâche fournie par le client.

Et ensuite envoyer le résultat au client.

On va définir une interface pour les tâches `compute.Task` et une interface pour l'exécution `compute.Compute`.

java.io.Serializable

Serializable indique que l'on peut utiliser la technique de « sérialisation ».

Ce mécanisme permet de transformer un Objet en « stream ».

Puis de pouvoir reconstruire l'objet en utilisant son « stream ».

compute.Compute

```
package compute;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Compute extends
  Remote {
 Object executeTask(Task t) throws
  RemoteException;
}
```

`java.rmi.Remote`

L'interface `Remote` sert à identifier les méthodes qui pourront être invoquée à distance.

Tout objet que l'on souhaite utiliser à distance doit faire un implement de cette interface.

RMI propose la classe `java.rmi.server.UnicastRemoteObject` et `java.rmi.activation.Activable` pour en faciliter l'utilisation.

Appel à distance

Il faut mettre en place un serveur.

Mettre en place un « security manager ».

Il faut créer des instances des objets.

Enregistrer les objets pour pouvoir les invoquer.

UnicastRemoteObject

`java.rmi.server.UnicastRemoteObject`

La classe `UnicastRemoteObject` est une classe permettant de créer des objets pour une utilisation RMI.

Elle fournit un ensemble de méthodes comme `equals, hashCode, toString, ....`

Elle fournit aussi des mécanismes pour recevoir des appels de clients.

Security Manager

Le Security Manager a pour objectif de protéger l'utilisation des ressources du système.

Le Security Manager détermine ce que le code chargé peut faire.

Tous les programmes utilisant RMI et chargeant des classes doivent mettre en place un Security Manager.

Security Manager

```
if (System.getSecurityManager() ==
  null)
{
 System.setSecurityManager(new
  RMISecurityManager());
}
```

java.rmi.RMISecurityManager

- La classe RMISecurityManager fournit un SecurityManager pour des applications RMI chargeant du code.
- La classe RMISecurityManager ne peut pas être utilisé avec un Applet qui utilise sont propre SecurityManager.
- Pour le mettre en oeuvre on doit le fixer dans le main de l'application.

java.rmi.Naming

Pour pouvoir utiliser un objet il faut mettre en place un mécanisme de nommage.

La classe Naming permet de stocker et manipuler des référence sur des Objets.

Le format est de la forme suivante :

//host:port/name

(la valeur par défaut de port est 1099)

java.rmi.Naming

On peut ainsi enregister engine.

```
String name = "//host/Compute";
try {
 Compute engine = new ComputeEngine();
 Naming.rebind(name, engine);
 System.out.println("ComputeEngine bound");
 }
catch (Exception e) {
    System.err.println("ComputeEngine exception:
  " + e.getMessage());
}
```

ComputeEngine

La création de engine par :

```
Compute engine = new
  ComputeEngine();
```

permet d'obtenir un objet que l'on pourra utiliser à distance.

```
public class ComputeMul {
 public static void main(String args[]){
 if (System.getSecurityManager() == null) {
   System.setSecurityManager(new RMISecurityManager());}
 try {
  String name = "//" + args[0] + "/Compute";
  Compute comp = (Compute) Naming.lookup(name);
  Multiply task = new Multiply(new BigInteger(args[1]),
   new BigInteger(args[2]));
  BigInteger mul = (BigInteger)(comp.executeTask(task));
  System.out.println(mul);
  } catch (Exception e) {
  System.err.println("ComputeMul exception: " +
  e.getMessage());
  } } }
```

Application

Comme application on va faire le produit de 2 nombres de type BigInteger.

Le client va demander au serveur de faire le calcul.

On va donc passer en paramètre deux nombres.

Multiply

```
public class Multiply implements Task {BigInteger
  val1,val2,result;
public Multiply(BigInteger pVal1, BigInteger pVal2)
  {
 this.val1 = pVal1;
 this.val2 = pVal2;
 }
public Object execute() {
 return computeMul();
 }
public BigInteger computeMul() {
 return val1.multiply(val2);
 }}
```

Sécurité

On a vu qu'on utilise un `SecurityManager.`

Pour pouvoir exécuter du code mobile on va devoir modifier la politique de sécurité.

Pour cela on va devoir créer un fichier java.policy décrivant la politique de sécurité.

java.policy

```
grant {
permission
  java.net.SocketPermission"*:1024-
  65535","connect,accept";
permission java.io.FilePermission
"\\home\\abouju\\Enseignement\\Inf
  ormatique...\\tp1\\-",
  "read";
};
```

java.policy

`java.net.SocketPermission` permet de définir les droits sur les sockets. Permission pour les ports de 1024 à 65535.

`java.io.FilePermission` permet de définir les droits pour le système de fichier.

Permission pour un chemin en lecture.

rmiregistry

la commande :

Microsoft Windows

```
start rmiregistry
```

UNIX-LINUX

```
rmiregistry &
```

Permet de mettre en place un système de nommage.

Par défaut on utilise le port 1099, on peut choisir un autre port en le passant en paramètre.

rmic (deprecated)

la commande :

`rmic`

Permet de mettre en place un mécanisme pour gérer les communications.

Pour cela on passe en paramètre le nom de la classe.

La commande génére 2 fichiers classes : `NomClasse_Stub.class` et `NomClasse_Skel.class.`

Lancement serveur

la commande :

```
java
-Djava.rmi.server.codebase= file:/...-
-Djava.rmi.server.hostname=serveur.univ-lr.fr
-Djava.security.policy=java.policy
engine.ComputeEngine
```

Permet de lancer le serveur et de préciser les chemins, le serveur et la politique de sécurité.

Lancement client

Permet de lancer le client, de préciser les chemins, le serveur et la politique de sécurité.

On suppose dans l'exemple qu'on peut obtenir le code par un serveur web.

Cela permet d'avoir des clients sur le web.

Lancement client

la commande :

```
Java
-Djava.rmi.server.codebase=
http://serveur.univ-lr.fr/tp/tp1
-Djava.security.policy=
java.policy
client.ComputeMul
serveur.univ-lr.fr 1000 1000
```

CORBA

Common Object Requesting Broker Architecture

Corba est un standard définit par l'OMG (Object Management Group) en 1991

Son objectif est de permettre à des composants écrit en plusieurs langages et sur plusieurs machine de fonctionner ensemble

OMG

Object Management Group

L'OMG est un consortium avec pour objectif le développement de systèmes objets distribués

Fondé en 1989 par 11 entreprises dont Hewlett-Packard, IBM, Sun Microsystems, Apple Computer, American Airlines, ...

OMG

Propositions de L'OMG :

CORBA,

DDS Data Distribution Service,

MDA Model Driven Architecture,

- UML Unified Modeling Language
- XML Metadata Interchange (XMI)
- MOF Meta-Object Facility

IDL
Interface description language

Est un langage qui permet de définir une interface utilisable indépendant des langages :

IDL existe en standard pour Ada, C, C++, Lisp, Ruby, Smalltalk, Java, Cobol, PL/I et Python

CORBA-Orbit

http://orbit-resource.sourceforge.net/

Orbit2 est un Object Request Broker (ORB)

Compatible Corba 2.4 avec les langages

C et Perl

Il a été conçu pour le GNU Object Model Environment (GNOME)

Son coeur est écrit en C avec un objectif de performance et de sécurité

CORBA-Orbit

http://orbit-resource.sourceforge.net/

Orbit2 est un Object Request Broker (ORB)

Compatible Corba 2.4 avec les langages

C et Perl

Il a été conçu pour le GNU Object Model Environment (GNOME)

Son coeur est écrit en C avec un objectif de performance et de sécurité

Bonobo

Bonobo est un cadre applicatif au dessus de corba qui permet la création de composants utilisable dans des applications

Bonobo est utilisé par exemple par l'application le GroupWare de GNOME

http://library.gnome.org/devel/platform-overview/s

D-BUS

http://www.freedesktop.org/wiki/Software/dbus

D-BUS offre un système pour permettre l 'échange de message entre application

D-BUS a été développé par Red Hat dans le cadre du projet « freedesktop.org »

Il est utilisé par KDE4 et Gnome

freedesktop.org

http://www.freedesktop.org

freedesktop.org

Freedesktop.org est projet créé par Havoc Pennington de Red Hat en 2000

Ce projet a pour objectif la mise en place d'outils pour la mise en place d'un environnement de travail

freedesktop.org

Quelques projets :

- X.Org
- HAL, (hardware abstraction layer)
- fontconfig
- Mesa 3D
- Xft,
- ...

Fin