5.1 Définition

Connu sous l'abréviation RMI pour Remote Method Invocation, est l'une des API (Application Programming Interface) Java qui permet de manipuler à distance des composants (objets) distribués d'une manière facile et transparente.

Grace à RMI, une machine A (Le client) peut invoquer une méthode fournit par un objet qui se trouve dans une autre machine B (Le serveur) sur un réseau comme si l'invocation se fait localement sur la même machine.

Cette technique manipule un demandeur de services (le programme qui fonctionne sur A et sollicite les services de B) et un fournisseur de services (le programme hébergé par B) si et seulement s'ils sont implémentés exclusivement en Java.

La transmission des invocations et des données (variables, paramètres de fonctions, valeur de retour, etc.) avec Java RMI se font sur TCP/IP grâce aux protocoles JRMP et IIOP par lequel il assure l'interopérabilité avec d'autres architectures comme CORBA.

5.2 Principe de Java RMI

Java RMI permet à un client de dialoguer avec un serveur à travers un ensemble d'étapes et par l'intermédiaires d'un ensemble d'éléments.

Commençant par les éléments, dont on peut citer les suivants (Fig.I) :

- **Le Client :** Le demandeur de service.
- **Le Serveur :** Le fournisseur de service.
- **Le Stub** (La souche) sur le client et le skeleton (le squelette) sur le serveur assurent la réception des invocations, la communication et l'échange de données entre le client et l'objet distant.
- **Le Registre :** c'est l'annuaire qui gère l'association nom-objet au niveau du serveur ; en d'autres termes, il référence les objets distants sur le serveur. La tache du registre est assurée par le package java.rmi.Naming.

- **Le transport :** c'est le moyen qui assure l'acheminement des invocations et le passage de données entre le client et le serveur à travers le fameux couple TCP/IP. Les packages java.net.Socket et java.net.SocketServer sont eux qui prennent cette tâche sur la couche transport par l'ouverture de sockets appropriés.

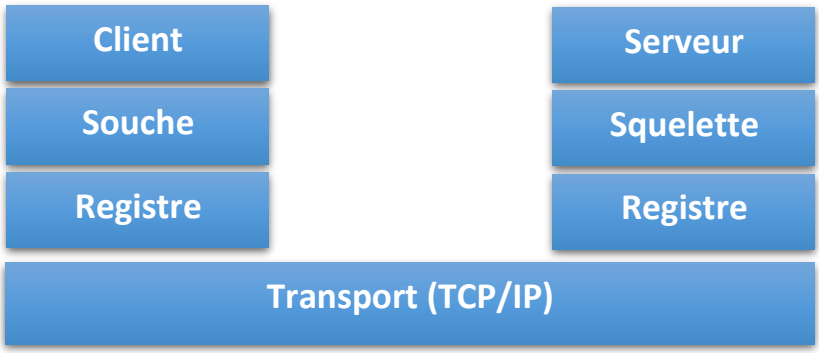

Fig.I : Structure des couches Java RMI

Les étapes sont ordonnées comme suit (Fig.2) :

- L'instanciation du serveur et son enregistrement auprès du l'annuaire (le Registry) avec un nom unique.
- La récupération des informations du serveur à travers son identifiant (le nom unique) avec lequel il s'est enregistré auprès du Registry.
- Le Registry envoie au client le stub afin qu'il puisse communiquer avec le serveur.
- Du coté client, la fonction principale du stub est le marshalling (la sérialisation) de données avant la transmission ; et du coté serveur, le skeleton assure la fonction inverse (désérialisation) après la réception de données.

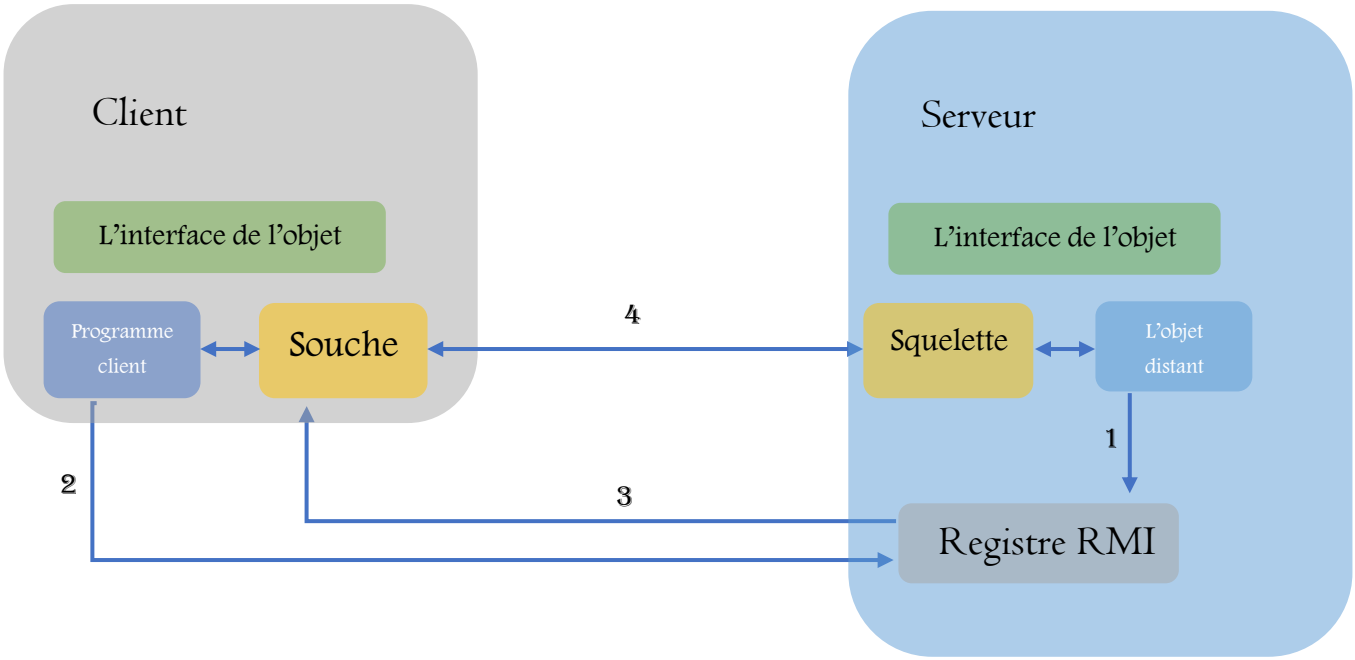

Fig.2 : Fonctionnement de Java RMI.

5.3 Création d'un objet et son invocation avec RMI

5.3.1 Coté serveur

- On crée une interface dans laquelle on déclare les prototypes de toutes les méthodes qu'un client peut invoquer à distance. Cette interface qui décrit l'objet distant hérite obligatoirement de la classe java.rmi.Remote pour montrer l'accessibilité à distance de l'objet.
- On crée une classe qui implémente l'interface.
- On crée un objet (distant) à partir cette classe et l'enregistre avec un nom dans le Registre RMI.
- On lance la compilation et l'exécution du registre RMI puis le serveur (Fig.7).

5.3.2 Coté client

- On demande la référence de l'objet distant à travers son nom.

- Une fois obtenue, on utilise la référence pour invoquer la méthode souhaitée avec les paramètres requis.
- On lance la compilation et l'exécution du client (Fig.8).

5.4 Hello world avec RMI

En suivant les étapes précédentes, on crée une application (Fig.4) qui permet à un client d'invoquer une méthode d'un objet distant qui affiche un simple message « Hello world ».

1. Définition de l'interface (Fig.3) :

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4  /**
5   *
6   * @author Loucif Hemza
7   */
8  public interface SayHello extends Remote{
9     public String say() throws RemoteException;
10 }
```

Fig.3 : La définition de l'interface.

La méthode « SayHello » lève l'exception **java.rmi.RemoteException** car elle est susceptible d'être invoquée à distance.

2. Définition de la classe qui correspond à l'objet distant (Fig.5) :
    - Cette classe doit dériver de la classe **UnicastRemoteObject** qui fournit des mécanismes pour exporter un objet distant et mettre le serveur à l'écoute des appels distant de clients éventuels.
    - Cette classe implémente l'interface **SayHello** en attribuant un code à la méthode invocables à distance **say ()**.

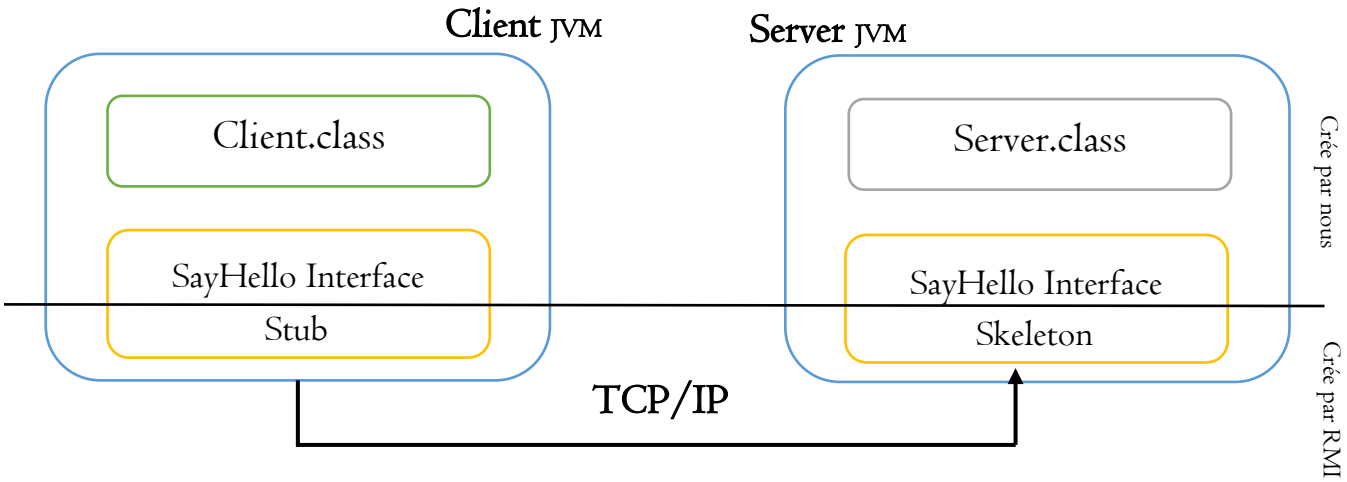

Fig.4 : Schématisation du Hello-world en Java RMI

- Après l'instanciation d'un objet distant et afin qu'il soit disponible, on l'enregistre dans le registre de noms RMI **reg** sous le nom « Hi-server » par la méthode **rebind** ().

```java
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

/**
 *
 * @author Loucif Hemza
 */
public class Server extends UnicastRemoteObject implements SayHello{
    public Server () throws RemoteException{
        super();
    }

    @Override
    public String say() throws RemoteException{
         return "Assalamo Alykom Server ^_^";
    }
    public static void main (String args[]) throws RemoteException{
        try{
        Registry reg=LocateRegistry.createRegistry(4445);
        reg.rebind("Hi-server", new Server());
        System.out.println(" The server is ready ...");
    }catch (RemoteException e){
        System.out.println("Exception "+e);
    }

}
}
```

Fig.5 : La définition du Serveur.

3. Définition du client (Fig.6) :
    - Après l'inscription de l'objet distant au service de nommage, le client utilise son nom pour obtenir une référence sur lui.
    - Le client recherche l'objet distant par lookup (nom de l'objet) qui retourne une référence sur cet objet.
    - Comme le montre l'instruction n°20 dans la fig.6, l'objet retourné par **lookup** () est de type **Remote** ; et dans ce cas nous devons appliquer un **casting** vers l'interface **SayHello** qui définit la méthode nous voulons invoquer.
    - Juste après le casting, on fait appel à la méthode sollicitée **say** ().

```java

import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

/**
 *
 * @author Loucif Hemza
 */
public class Client {
    public static void main(String args[]) throws RemoteException, NotBoundException{
        Client c= new Client ();
        c.connect();
    }

    private void connect() throws RemoteException{
        try{
         Registry reg = LocateRegistry.getRegistry("localhost", 4445);
         SayHello hello=(SayHello) reg.lookup("Hi-server");
         System.out.println("-->"+hello.say());

    }catch(NotBoundException|RemoteException e){
        System.out.println("Exception "+e);
    }
    }
}
```

Fig.6 : La définition du client.

Remarques

- L'invocation d'un objet distant peut se faire normalement dans une applet.
- Le registre RMI doit s'exécuter avant l'enregistrement de l'objet distant (par le serveur) ou l'obtention de sa référence (par le client).

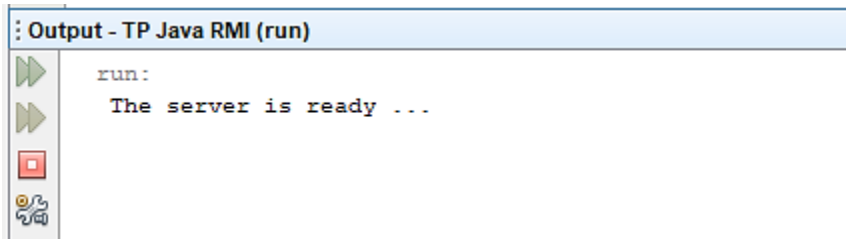

Fig.7 : le serveur à l'état prêt.

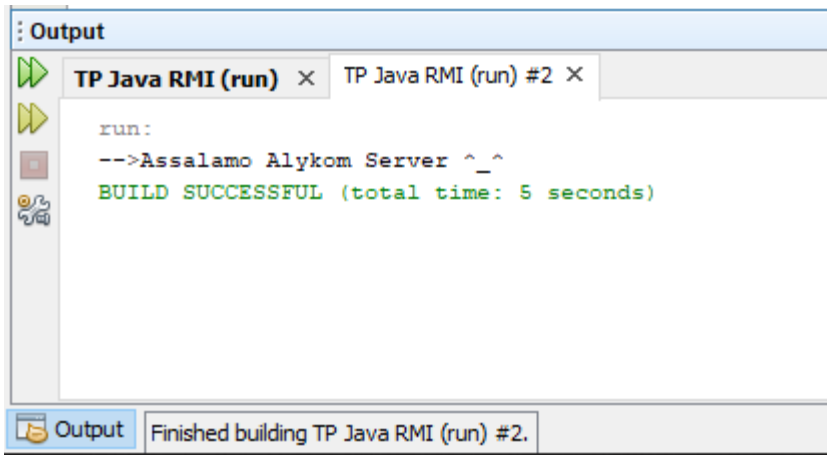

Fig.8 : l'invocation de la méthode distante a réussi.