La plate-forme Java RMI

Frank Singhoff

Bureau C-203

Université de Brest, France

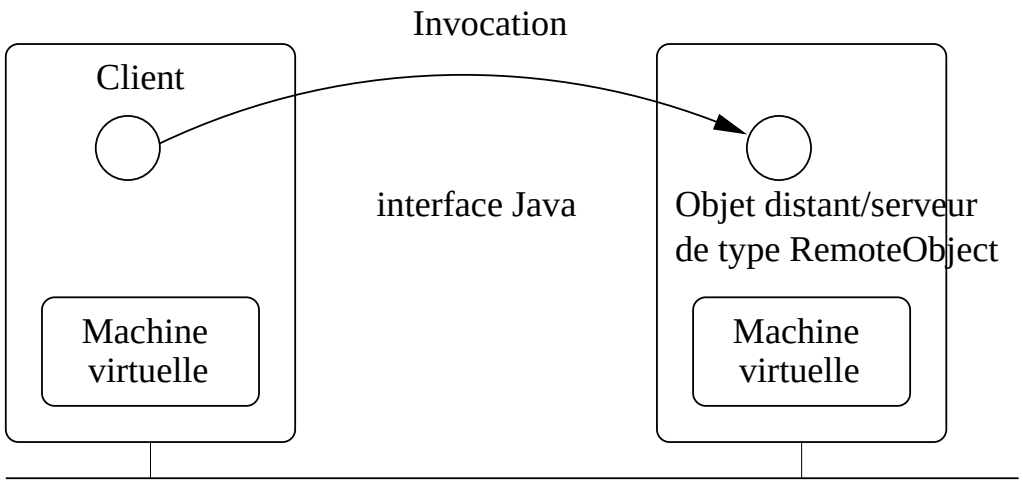
Laboratoire Lab-STICC UMR CNRS 6285

singhoff@univ-brest.fr

Sommaire

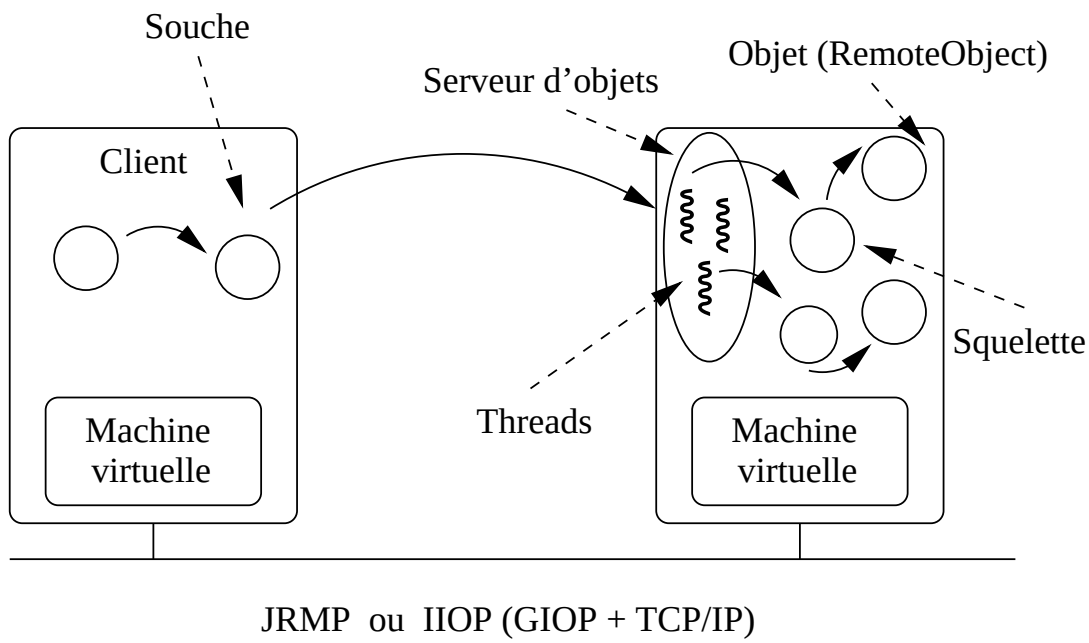
1. Le modèle objets et les services de base.
2. Service de code mobile.
3. Résumé.
4. Références.

Le modèle objet de Java /RMI



- RMI [MIC 98, DOW 98, E.D 00] (Remote Method Invocation) = pouvoir invoquer un objet d'une JVM autre que la JVM locale.
- Pas de prise en compte de l'interopérabilité mais simple d'utilisation.
- Pas de transparence à la localisation.
- Quasi transparence d'accès : exceptions différentes, passage des objets locaux par copie (et non par référence, lorsque l'invocation est locale), objets distants passées par référence (souches).

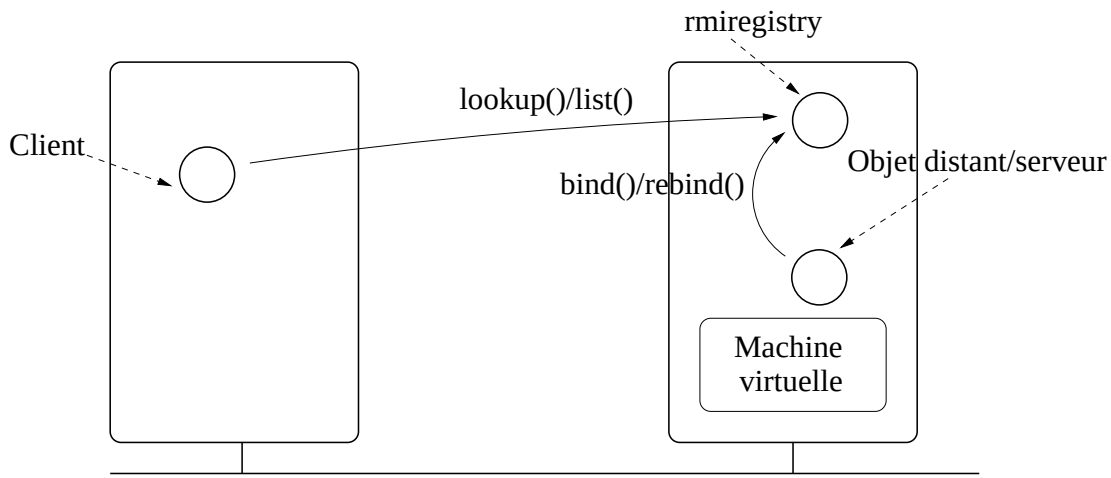
Fonctionnement de la plate-forme



- Protocole d'acheminement des invocations sur TCP : IIOP ou JRMP [a].
- Utilisation de souches et squelettes générés à partir de l'interface de l'objet.
- Serveur d'objets : exécute les invocations d'objets de façon multithread.
- Opération d'activation $\Longrightarrow$ création du serveur d'objets (thread d'écoute).

[a] Java Remote Methode Protocol.

Désignation et service de nommage (1)



- Désignation : $//machine : port/objet$ où $objet$ constitue le nom symbolique désignant un $RemoteObject$ et le $port$ constitue l'espace de désignation $\Longrightarrow$ pas de transparence à la localisation.
- Serveur de nom : commande $rmiregistry$.
- Phase de liaison : $bind$, $rebind$, $unbind$.
- Phase de résolution : $lookup$, $list$. La méthode $lookup$ permet au client d'obtenir une souche.

Désignation et service de nommage (2)

- La classe statique $Naming$:

```
public final class Naming {

  public static Remote lookup(String nom)
    throws ...
  public static void bind(String nom, Remote obj)
    throws ...
  public static void unbind(String nom)
    throws ...
  public static void rebind(String nom, Remote obj)
    throws ...
  public static String[] list(String nom)
    throws ...
}
```

Support de l’hétérogénéité

1. Système d’exploitation et langage de programmation : machine virtuelle Java .

2. Codage/décodage, sérialisation/désérialisation

3. Services pour le code mobile. Ne capture pas le contexte d’exécution d’un thread.

Autres services

1. Ramasse-miettes réparti pour les objets distants (de type $RemoteObject$).
2. Gestionnaire de sécurité ($RMISecurityManager$).
3. Archivage des invocations effectuées sur un objet (méthodes $getLog/setLog$).
4. Activation sur invocation d'objets ($rmid$) ; activations dites permanentes.
5. Création ou recherche de serveurs de noms (classe $LocateRegistry$).
6. ...

Exemple : client/serveur (1)

- L'interface (Hello.java) :

```
public interface Hello extends java.rmi.Remote
{
        String lireMessage()
           throws java.rmi.RemoteException;
}
```

- Interface Java normale sauf :
  - Héritage de $java.rmi.Remote \Longrightarrow$ objet distant.
  - Utilisation d'exceptions spécifiques ($RemoteException$).

Exemple : client/serveur (2)

- Le client (HelloClient.java) :

```
...
String nom="//machine:port/HelloServeur";

Hello obj=(Hello) Naming.lookup(nom)

message=obj.lireMessage();
...
```

- Etapes effectuées par le client :

1. Consultation du service de nommage et obtention d'une souche.
2. Invocation de l'objet distant.

Exemple : client/serveur (3)

- Le serveur (HelloServeur.java) :

```
public class HelloServeur
     extends UnicastRemoteObject
    implements Hello
{
  public String lireMessage()  ...

  public static void main ...
  {
    HelloServeur MonServeur=new HelloServeur();
    String nom="//machine:port/HelloServeur";
    Naming.rebind(nom, MonServeur);
    ...
  }
}
```

Exemple : client/serveur (4)

- Etapes effectuées par le serveur :

1. Déclaration d'un objet distant $HelloServeur$ qui implante l'interface $Hello$. Objet de type $UnicastRemoteObject$ ou $RemoteObject$.

2. Fournir une implantation des méthodes invocables à distance.

3. Création de l'objet.

4. Activation de l'objet (création du serveur d'objets) : activation effectuée par le constructeur de $UnicastRemotreObject$. Si l'on utilise un $RemoteObject$, il faut invoquer $exportObject()$.

5. Publication vers le service de nom.

Exemple : client/serveur (5)

- **Compilation :**
  - Générer les souches et squelettes avec la commande :

    ```
    rmic HelloServeur
    ```

  - Compiler les sources Java .
- **Exécution ; dans le même répertoire, lancer :**
  - Le serveur de nom par la commande :

    ```
    rmiregistry num
    ```

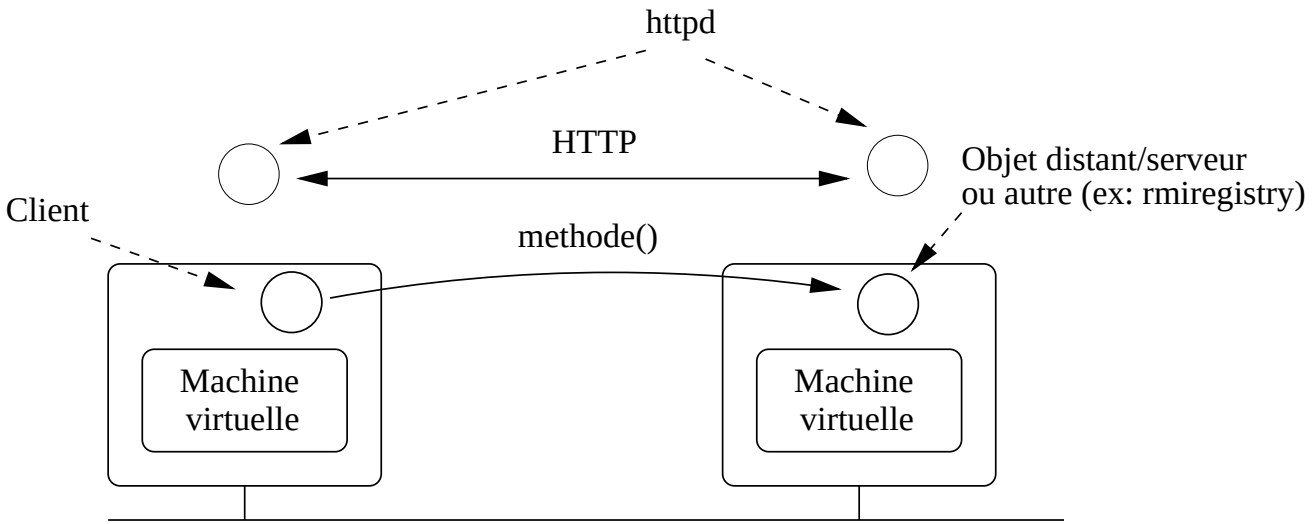
    où num est le numéro de port d'écoute du serveur.
  - Le serveur, puis le client par :

    ```
    java HelloServeur&
    java HelloClient
    ```

Sommaire

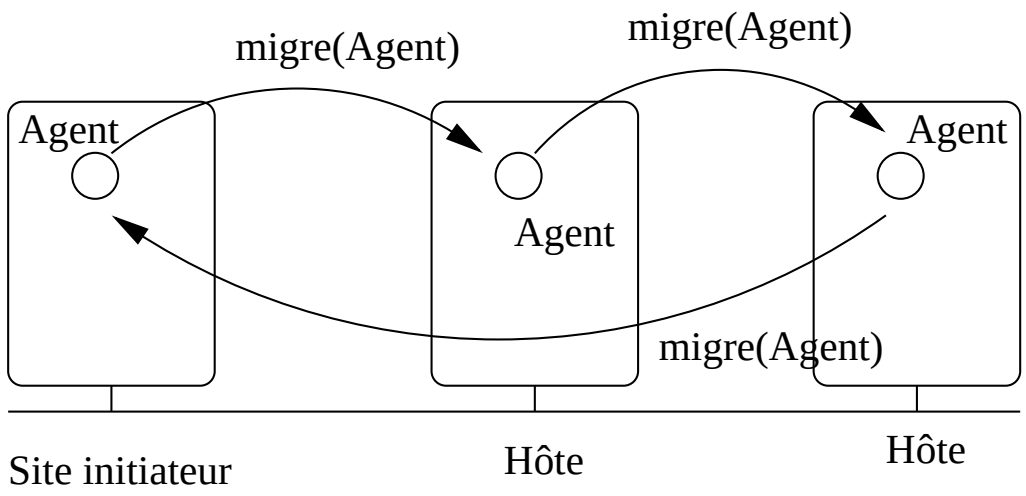
1. Le modèle objets et les services de base.
2. Service de code mobile.
3. Résumé.
4. Références.

Service de code mobile



- $.class$ chargées, soit par le chargeur normal de la JVM (CLASSPATH), soit par un démon $httpd \Longrightarrow$ les fichiers doivent être accessibles au démon.
- Tout objet sérialisable est téléchargeable : souche (ex : $rmiregistry$) ou autre.
- Téléchargement à partir de (applet) ou vers le serveur.

Exemple : agent mobile (1)



- **Fonctionnement :**
  - Une liste de sites, ou **Hôtes**, permettent l'accueil d'**agents**.
  - Un agent part d'un site **Initiateur**, puis, visite un ensemble d'hôtes pour y effectuer un traitement donné.
  - Finalement, l'agent revient sur le site de départ pour rapporter ses résultats.

Exemple : agent mobile (2)

- L'interface de l'agent : code + données mobiles

```
import java.io.Serializable;

public interface Agent extends Serializable
{
void traitement(...);

void afficheResultat();

String hoteSuivant();
}
```

- *Serializable* indique que l'objet, s'il est utilisé en paramètre d'une méthode, doit être sérialisé et encodé (marshalling).

Exemple : agent mobile (3)

- L'implantation de l'agent :

```
public class agentIngredient implements Agent
{
   private Float monPrix = new ...
   private String monIngredient="";
   ...

   public String hoteSuivant()
   ...

   public void afficheResultat()
   ...

   public void traitement(...)
   ...
}
```

Exemple : agent mobile (4)

- Le service de migration offert par les hôtes :

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hote extends Remote
{
   void migre(Agent a)
        throws RemoteException;
}
```

- Chaque hôte offre une interface permettant de recevoir un agent et de le lancer par un thread.

Exemple : agent mobile (5)

- La mise en œuvre de $migre$:

```
public void migre(Agent a) {
  threadAgent monThread = new threadAgent(...
  monThread.start();
  }
```

- Le thread, support de l'agent :

```
class threadAgent extends Thread  {
  public void run() {
    monAgent.traitement(...);
    String leSuivant = monAgent.hoteSuivant();
    Hote hote  = (Hote) Naming.lookup(leSuivant);
    hote.migre(monAgent);
  }
}
```

Exemple : agent mobile (6)

- Compilation : $rmic$ et $javac$.
- Lancement de l'hôte :

```
java -Djava.rmi.server.codebase=http://...
  -Djava.rmi.server.hostname=beru.univ-brest.fr
  -Djava.security.policy=java.policy
     Hote_implem
```

  - $server.codebase$ = URL du serveur Web où les $.class$ peuvent être chargées.
  - $server.hostname$ = localisation du serveur d'objets.
  - $java.policy$ = fichier pour RMISecurityManager.

- Lancement du $rmiregistry$: doit forcer le serveur de nom à obtenir la souche par le serveur web $\Longrightarrow$ un objet sérialisé contient l'URL permettant d'obtenir le code, rien sinon (CLASSPATH).

Sommaire

1. Le modèle objets et les services de base.
2. Service de code mobile.
3. Résumé.
4. Références.

Conclusion et résumé

- Java RMI : permettre l'invocation d'objets d'une JVM autre que la JVM locale.

- Plate-forme simple à utiliser. Quasi transparence d'accès mais pas de transparence à la localisation.

- Pas d'interopérabilité : de Java vers Java .

- Service de code mobile $\Longrightarrow$ nouvelles applications telles que agents mobiles, web.

Sommaire

1. Le modèle objets et les services de base.
2. Service de code mobile.
3. Résumé.
4. Références.

Références

[DOW 98] G. Downing. *Developing Java Distributed Applications with RMI*. IDG Books, February 1998.

[E.D 00] G. Roussel et E. Duris. *JAVA et Internet : concepts et programmation*. Vuibert Informatique, 2000.

[MIC 98] Sun Microsystems. « Java Remote Invocation Specification ». pages 1–124, 1998.