Système Distribué avec RMI

Yassamine Seladji

yassamine.seladji@gmail.com

17 juin 2019

Introductions

- **RMI** est un protocole de communication utilisé dans les applications réparties.

Introductions

- **RMI** est un protocole de communication utilisé dans les applications réparties.
- RMI permet l'appel de méthodes distantes.

Introductions

- **RMI** est un protocole de communication utilisé dans les applications réparties.
- RMI permet l'appel de méthodes distantes.
- Ces méthodes doivent être écrites en **Java**.

Introductions

- **RMI** est un protocole de communication utilisé dans les applications réparties.
- RMI permet l'appel de méthodes distantes.
- Ces méthodes doivent être écrites en **Java**.
- En d'autre termes, **RMI** permet la communication entre **JVM** différentes.

Introductions

- **RMI** est un protocole de communication utilisé dans les applications réparties.
- RMI permet l'appel de méthodes distantes.
- Ces méthodes doivent être écrites en **Java**.
- En d'autre termes, **RMI** permet la communication entre **JVM** différentes.
- **RMI** assure des échanges entre **différents systèmes d'exploitation** possédant une JVM.

La mise en œuvre

- Mettre en place une communication **RMI** nécessite la maîtrise du code et la réécriture de certaines **interfaces**.

La mise en œuvre

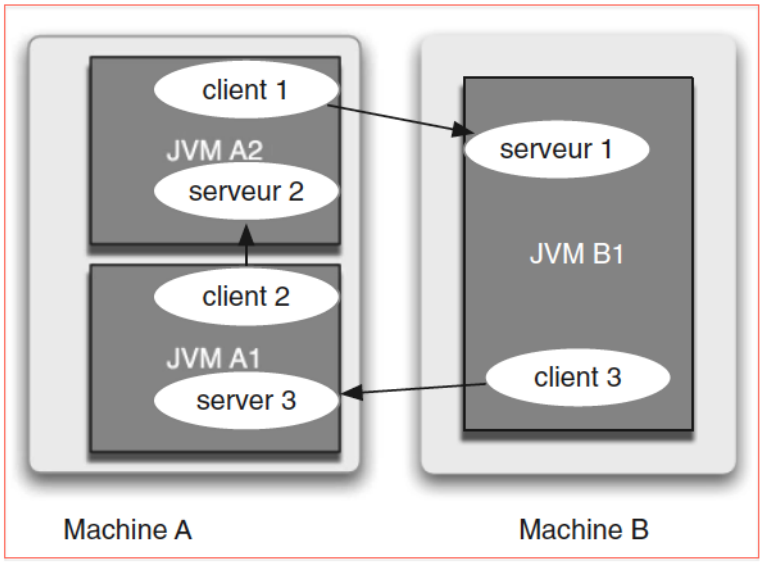
- Mettre en place une communication **RMI** nécessite la maîtrise du code et la réécriture de certaines **interfaces**.
- RMI définit un mode client/serveur.

La mise en œuvre

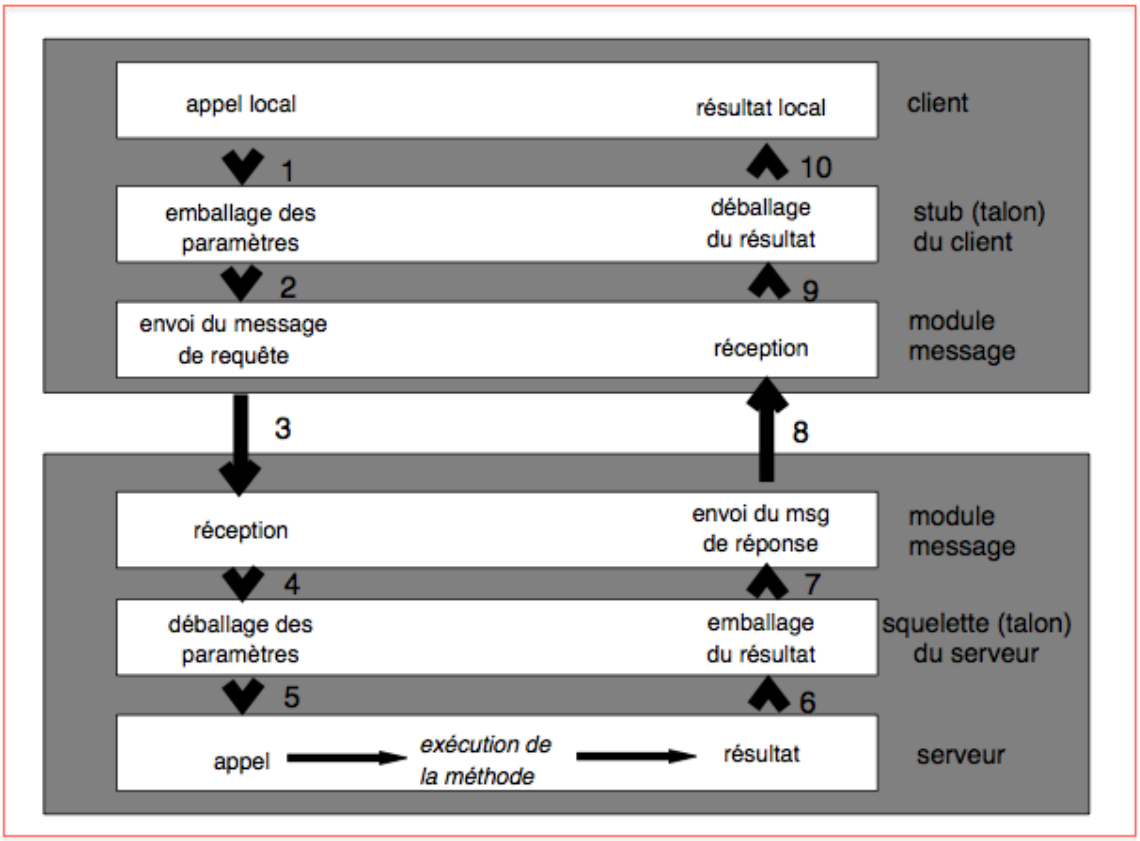
- Mettre en place une communication **RMI** nécessite la maîtrise du code et la réécriture de certaines **interfaces**.
- RMI définit un mode client/serveur.
- Le serveur et le client représentent des **objets Java**.

La mise en œuvre

- Mettre en place une communication **RMI** nécessite la maîtrise du code et la réécriture de certaines **interfaces**.
- RMI définit un mode client/serveur.
- Le serveur et le client représentent des **objets Java**.

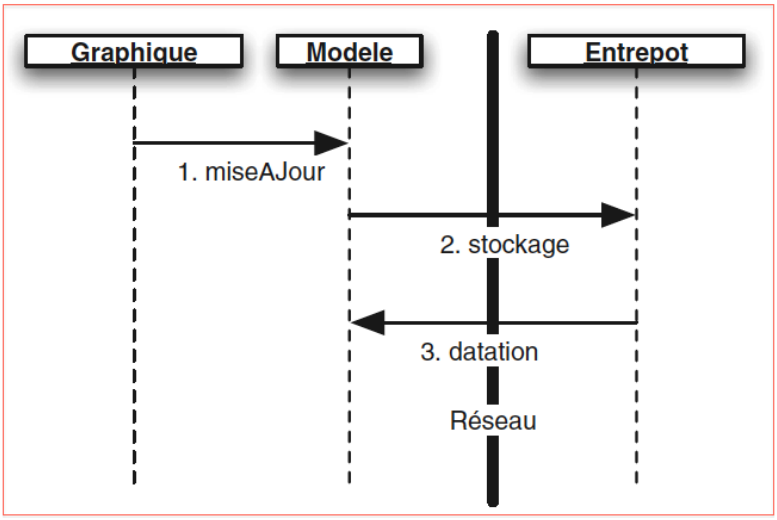


Les étapes d'un appel à distance



La mise en œuvre : Exemple

Exemple d'une application en réseau.



La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.
- Le lancement du serveur puis du client.

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.
- Le lancement du serveur puis du client.

La définition des interfaces

- L'interface est une projection des méthodes de l'objet accessibles via le réseau.

La définition des interfaces

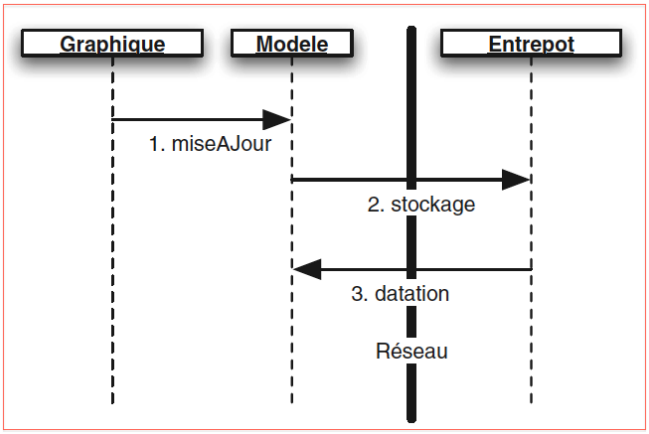
- L'interface est une projection des méthodes de l'objet accessibles via le réseau.
- **L'interface en RMI** est une interface Java simple qui hérite de l'interface **java.rmi.Remote**.

La définition des interfaces

- L'interface est une projection des méthodes de l'objet accessibles via le réseau.
- **L'interface en RMI** est une interface Java simple qui hérite de l'interface **java.rmi.Remote**.
- Les interfaces doivent obligatoirement lancer l'exception **java.rmi.RemoteException**.

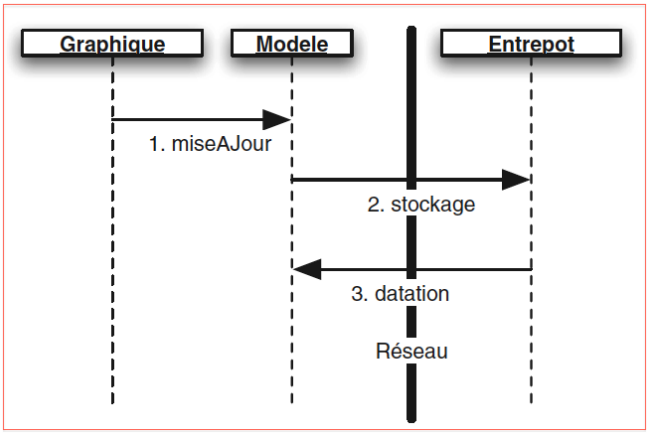
La définition des interfaces : Exemple

Exemple d'une application en réseau.



La définition des interfaces : Exemple

Exemple d'une application en réseau.




```
public interface ModelInterface extends java.rmi.Remote{
    public Date datation() throws java.rmi.RemoteException;
}
```

La définition des interfaces

En mettant en place une interface, il faudra s'assurer que :

- Les arguments sont **sérializables**.

La définition des interfaces

En mettant en place une interface, il faudra s'assurer que :

- Les arguments sont **sérializables**.
- Les méthodes accessibles doivent être **publiques**.

La définition des interfaces

En mettant en place une interface, il faudra s'assurer que :

- Les arguments sont **sérializables**.
- Les méthodes accessibles doivent être **publiques**.
-

```
public interface ModelInterface extends java.rmi.Remote{
    public Date datation() throws java.rmi.RemoteException;
}
```

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.
- Le lancement du serveur puis du client.

L'implémentation de l'objet distant.

En RMI, l'objet distant peut être implémenter de deux manières :

- Implémentation par héritage.

L'implémentation de l'objet distant.

En RMI, l'objet distant peut être implémenter de deux manières :

- Implémentation par héritage.
- Implémentation par délégation.

L'implémentation de l'objet distant.

Implémentation par héritage.

- L'objet doit hérité de **java.rmi.server.UnicastRemoteObject**.

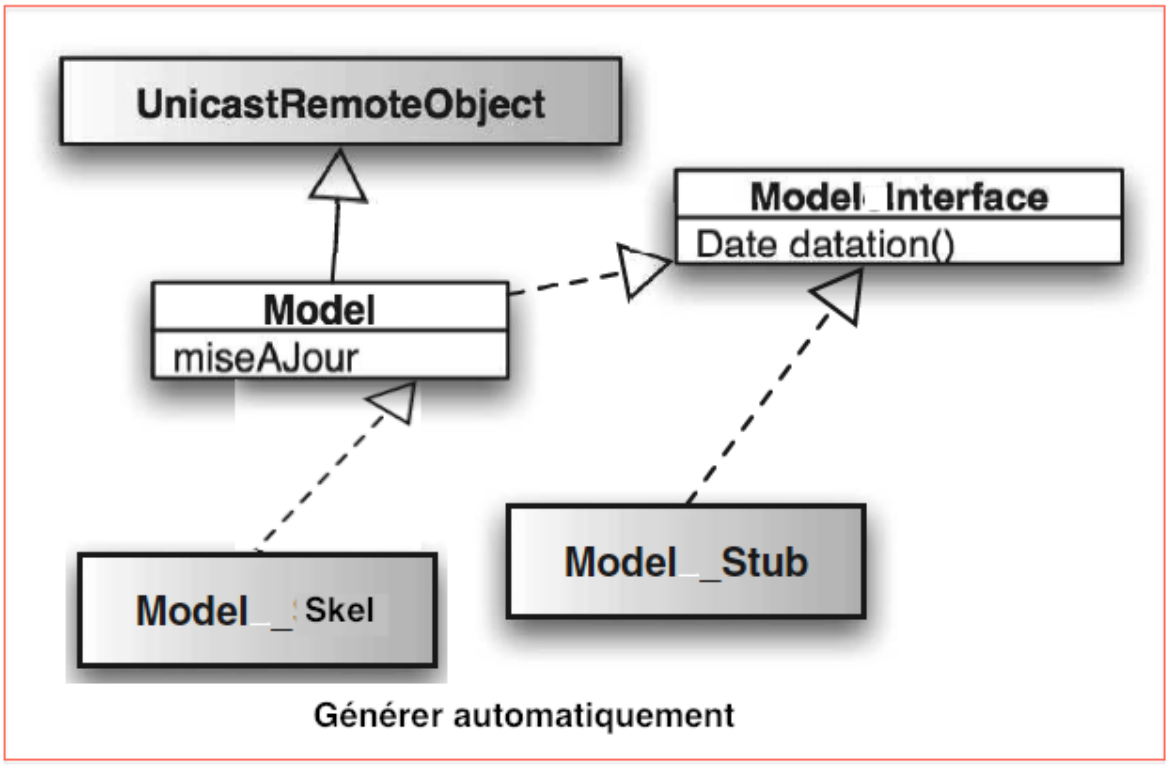
L'implémentation de l'objet distant.

Implémentation par héritage.

- L'objet doit hérité de **java.rmi.server.UnicastRemoteObject**.
- Le constructeur de l'objet doit lancer l'exception **java.rmi.RemoteException**.

L'implémentation de l'objet distant.

Implémentation par héritage.



L'implémentation de l'objet distant.

Implémentation par héritage.

```
public class Model extends UnicastRemoteObject implements ModelInterface{

  protected Model() throws RemoteException {
      super();
  }
  @Override
  public Date datation() throws RemoteException {
      return new GregorianCalendar().getTime();
  }
  public void miseAjour(){}
```

L'implémentation de l'objet distant.

Implémentation par délégation.

- Dans ce cas on utilise le mécanisme de mise en service.

L'implémentation de l'objet distant.

Implémentation par délégation.

- Dans ce cas on utilise le mécanisme de mise en service.
- La classe de l'objet distant n'hérite pas de **java.rmi.server.UnicastRemoteObject**.

L'implémentation de l'objet distant.

Implémentation par délégation.

- Dans ce cas on utilise le mécanisme de mise en service.
- La classe de l'objet distant n'hérite pas de **java.rmi.server.UnicastRemoteObject**.
- La mise à disposition de l'objet distant se fait comme suite :

L'implémentation de l'objet distant.

Implémentation par délégation.

- Dans ce cas on utilise le mécanisme de mise en service.
- La classe de l'objet distant n'hérite pas de **java.rmi.server.UnicastRemoteObject**.
- La mise à disposition de l'objet distant se fait comme suite :

```
Model model = new Model();
ModelInterface stub = (ModelInterface)
        UnicastRemoteObject.exportObject(model,0);
```

L'implémentation de l'objet distant.

Implémentation par délégation.

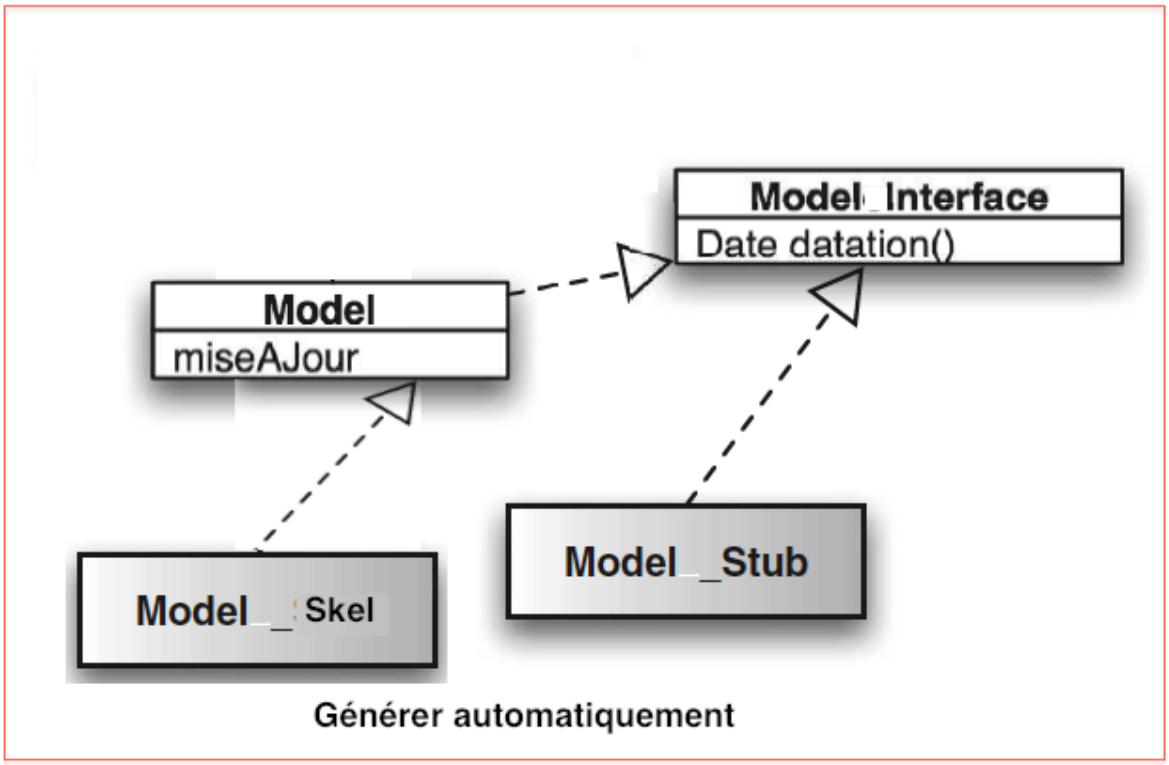
- Dans ce cas on utilise le mécanisme de mise en service.
- La classe de l'objet distant n'hérite pas de **java.rmi.server.UnicastRemoteObject**.
- La mise à disposition de l'objet distant se fait comme suite :

```
Model model = new Model();
ModelInterface stub = (ModelInterface)
        UnicastRemoteObject.exportObject(model,0);
```

- Le **0** représente l'allocation dynamique du port du serveur.

L'implémentation de l'objet distant.

Implémentation par délégation.



La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.
- Le lancement du serveur puis du client.

La génération du Stub et du Squelette.

Le fournisseur de service exporte une référence.

- Côté client :

La génération du Stub et du Squelette.

Le fournisseur de service exporte une référence.

- Côté client :
  - Le client reçoit cette référence.

La génération du Stub et du Squelette.

Le fournisseur de service exporte une référence.

- Côté client :
  - Le client reçoit cette référence.
  - Le **Stub** utilise cette référence pour envoyer des appels de méthodes au fournisseur.

La génération du Stub et du Squelette.

Le fournisseur de service exporte une référence.

- Côté client :
  - Le client reçoit cette référence.
  - Le **Stub** utilise cette référence pour envoyer des appels de méthodes au fournisseur.
  - En utilisant le processus de pliage des arguments (*marshalling*).

La génération du Stub et du Squelette.

Le fournisseur de service exporte une référence.

- Côté client :
  - Le client reçoit cette référence.
  - Le **Stub** utilise cette référence pour envoyer des appels de méthodes au fournisseur.
  - En utilisant le processus de pliage des arguments (*marshalling*).
  - En recevant la requette du serveur, il utilise le processus de dépliage pour récupérer le résultat.

La génération du Stub et du Squelette.

- Côte serveur :

La génération du Stub et du Squelette.

- Côte serveur :
  - Le **Squelette** effectue le processus du dépliage des arguments (*unmarshalling*).

La génération du Stub et du Squelette.

- Côte serveur :
  - Le **Squelette** effectue le processus du dépliage des arguments (*unmarshalling*).
  - Le **Squelette** invoque la méthode adéquate du serveur.

La génération du Stub et du Squelette.

- Côte serveur :
  - Le **Squelette** effectue le processus du dépliage des arguments (*unmarshalling*).
  - Le **Squelette** invoque la méthode adéquate du serveur.
  - Il envois le résultat en utilisant le processus de *marshalling*.

La génération du Stub et du Squelette.

- Le code du **Stub** et du **Squelette** sont générés **automatiquement** au moment de la compilation de la classe de l'objet distant et cela en utilisant le compilateur **javac**.

La génération du Stub et du Squelette.

Le rôle du **Stub** est de masquer au client :

- La localisation de la référence sur le serveur.

La génération du Stub et du Squelette.

Le rôle du **Stub** est de masquer au client :

- La localisation de la référence sur le serveur.
- L'appel des méthodes à distance.

La génération du Stub et du Squelette.

Le rôle du **Stub** est de masquer au client :

- La localisation de la référence sur le serveur.
- L'appel des méthodes à distance.
- Le pliage et le dépliage des arguments.

La génération du Stub et du Squelette.

Le rôle du **Stub** est de masquer au client :

- La localisation de la référence sur le serveur.
- L'appel des méthodes à distance.
- Le pliage et le dépliage des arguments.
- La cohérence entre les différents appels sur le serveur.

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.
- Le lancement du serveur puis du client.

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Le serveur crée un annuaire associé à un numéro de port, en utilisant la classe **java.rmi.registry.LocateRegistry**.

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Le serveur crée un annuaire associé à un numéro de port, en utilisant la classe **java.rmi.registry.LocateRegistry**.
  - Le serveur instancie l'objet et l'enregistre dans l'annuaire, en utilisant la méthode **bind(nomObjet,objet)** de la classe **Naming**.

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Le serveur crée un annuaire associé à un numéro de port, en utilisant la classe **java.rmi.registry.LocateRegistry**.
  - Le serveur instancie l'objet et l'enregistre dans l'annuaire, en utilisant la méthode **bind(nomObjet,objet)** de la classe **Naming**.
  - Le serveur lance l'annuaire afin que l'objet soit accessible via le réseau.

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Le serveur crée un annuaire associé à un numéro de port, en utilisant la classe **java.rmi.registry.LocateRegistry**.
  - Le serveur instancie l'objet et l'enregistre dans l'annuaire, en utilisant la méthode **bind(nomObjet,objet)** de la classe **Naming**.
  - Le serveur lance l'annuaire afin que l'objet soit accessible via le réseau.

```
Model modele = new Model();
LocateRegistry.createRegistry(1099);
Naming.bind("rmi://localhost:1099/topModel", modele);
```

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Dans le cas d'implémentation par héritage :

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Dans le cas d'implémentation par héritage :
    -

```
Model modele = new Model();
LocateRegistry.createRegistry(1099);
Naming.bind("rmi://localhost:1099/topModel", modele);
```

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Dans le cas d'implémentation par héritage :
    -

```
Model modele = new Model();
LocateRegistry.createRegistry(1099);
Naming.bind("rmi://localhost:1099/topModel", modele);
```

  - Dans le cas d'implémentation par délégation :

L'écriture du code de la mise à disposition

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté serveur :
  - Dans le cas d'implémentation par héritage :

```
Model modele = new Model();
LocateRegistry.createRegistry(1099);
Naming.bind("rmi://localhost:1099/topModel", modele);
```

  - Dans le cas d'implémentation par délégation :

```
Model modele = new Model();
ModelInterface stub = (ModelInterface)
        UnicastRemoteObject.exportObject(modele,0);
LocateRegistry.createRegistry(1099);
Naming.bind("rmi://localhost:1099/topModel", stub);
```

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.
- Le lancement du serveur puis du client.

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté client :
  - Le client demande une référence au serveur.

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté client :
  - Le client demande une référence au serveur.
  - Si le serveur se trouve dans une machine distante, le client devra connaitre son adresse **IP**.

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté client :
  - Le client demande une référence au serveur.
  - Si le serveur se trouve dans une machine distante, le client devra connaitre son adresse **IP**.
  - La référence est obtenue via l'annuaire en :

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté client :
  - Le client demande une référence au serveur.
  - Si le serveur se trouve dans une machine distante, le client devra connaitre son adresse **IP**.
  - La référence est obtenue via l'annuaire en :
    - appelant la méthode **lookup** de la classe **Naming**.

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté client :
  - Le client demande une référence au serveur.
  - Si le serveur se trouve dans une machine distante, le client devra connaitre son adresse **IP**.
  - La référence est obtenue via l'annuaire en :
    - appelant la méthode **lookup** de la classe **Naming**.
    - connaissant le nom logique de l'objet.

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté client :
  - Le client demande une référence au serveur.
  - Si le serveur se trouve dans une machine distante, le client devra connaitre son adresse **IP**.
  - La référence est obtenue via l'annuaire en :
    - appelant la méthode **lookup** de la classe **Naming**.
    - connaissant le nom logique de l'objet.
    - Dans notre exemple le nom logique est **topModel**.

L'écriture du code d'exploitation

Comment l'objet client va-t-il trouver la référence au serveur ?

- Coté client :
  - Le client demande une référence au serveur.
  - Si le serveur se trouve dans une machine distante, le client devra connaitre son adresse **IP**.
  - La référence est obtenue via l'annuaire en :
    - appelant la méthode **lookup** de la classe **Naming**.
    - connaissant le nom logique de l'objet.
    - Dans notre exemple le nom logique est **topModel**.

```
ModelInterface modele = (ModelInterface)
        Naming.lookup("rmi://localhost:1099/topModel");
Date uneDate = modele.datation();
System.out.println("la date "+uneDate.toString());
```

L'arrêt d'un objet serveur

- L'enregistrement d'un objet dans l'annuaire est bloquant :

L'arrêt d'un objet serveur

- L'enregistrement d'un objet dans l'annuaire est bloquant :
  - ```
    Naming.bind("rmi://localhost:1099/topModel", modele);
    ```
    ```

L'arrêt d'un objet serveur

- L'enregistrement d'un objet dans l'annuaire est bloquant :
  - ```
    Naming.bind("rmi://localhost:1099/topModel", modele);
    ```
  - Le code serveur reste en attente de requête sans terminer le programme.
    ```

L'arrêt d'un objet serveur

- L'enregistrement d'un objet dans l'annuaire est bloquant :
  - ```
    Naming.bind("rmi://localhost:1099/topModel", modele);
    ```
  - Le code serveur reste en attente de requête sans terminer le programme.
- Pour arrêter le programme, il faut désactiver l'objet serveur en le supprimant de l'annuaire, avec la méthode **unbind()**.
    ```

L'arrêt d'un objet serveur

- L'enregistrement d'un objet dans l'annuaire est bloquant :
  - ```
    Naming.bind("rmi://localhost:1099/topModel", modele);
    ```
  - Le code serveur reste en attente de requête sans terminer le programme.
- Pour arrêter le programme, il faut désactiver l'objet serveur en le supprimant de l'annuaire, avec la méthode **unbind()**.

```
LocateRegistry.getRegistry(1099).unbind("rmi://localhost:1099/topModel");
```
```

La mise en œuvre

La mise en œuvre d'un protocole **RMI** passe par les étapes suivantes :

- La définition des interfaces.
- L'implémentation de l'objet distant.
- La génération automatique du code du Stub et du squelette.
- L'écriture du code de la mise à disposition (serveur).
- L'écriture du code d'exploitation (client).
- Le lancement de l'annuaire.
- Le lancement du serveur puis du client.

Le lancement de l'annuaire

Le lancement de l'annuaire peut se faire de deux manières :

- Par code :

```
LocateRegistry.createRegistry(1099);
```

- Par ligne de commande : en lançant **rmiregistry**, suivie du numéro de port (par défaut 1099).

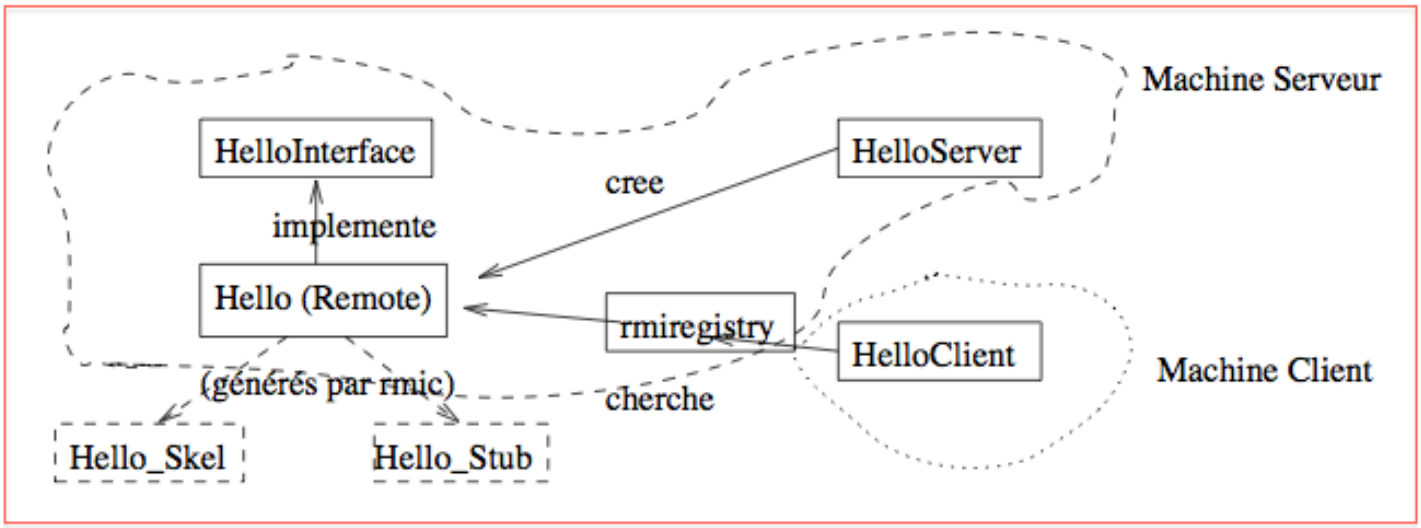
Le lancement du serveur puis du client

Le lancement de l'annuaire peut se faire de deux manières :

- Il faudra commencer par lancer le serveur afin qu'il soit à l'écoute du client.
- Le client se lancera par la suite.
- en utilisant la commande **java** sur un terminal.
- ou en utilisant un IDE.

Exercice d'application

Le but de l'exercice est de mettre en place une application "**Hello World**" répartie.



Exercice d'application

Les étapes à suivre :

- Créer l'interface **HelloInterface** qui hérite de la classe **Remote**.
- Proposer une implémentation de cette interface.
- Proposer une implémentation de la classe **HelloServer**.
- Proposer une implémentation de la classe **HelloClient**.

Exercice d'application : correction

La correction proposée :

```
public interface HelloInterface extends Remote {
    public String say() throws RemoteException;
}
```

```
public class Hello extends UnicastRemoteObject implements HelloInterface{
   private String message;

   protected Hello(String message) throws RemoteException {
       this.message = message;
   }

   public String say() throws RemoteException {
       return this.message;
   }
```

Exercice d'application : correction

```
public class HelloServer {
   public static void main(String[] argv) {
       try {
           LocateRegistry.createRegistry(1099);
           Naming.rebind("rmi://localhost:1099/Hello",new Hello("Hello, world!"));
           System.out.println("Hello Server is ready.");
       } catch(Exception e) {
           System.out.println("Hello Server failed: "+e);
       }
   }
```

```
public class HelloClient {
   public static void main(String[] argv) {
       try {
           HelloInterface hello = (HelloInterface) Naming.lookup
                   ("rmi://localhost:1099/Hello");
           System.out.println(hello.say());
       } catch(Exception e) {
           System.out.println("HelloClient exception: "+e);
       }
   }
```

Exercice d'application 2

Développer un couple de processus (P_A, P_B), tel que :

- P_A crée un objet A.
- P_B crée un objet B.
- Le processus P_A appel une méthode $B.m(..)$.
- La méthode $B.m(..)$ rappel une méthode de A.

Proposer une solution distribuée, tel que P_A et P_B tournent sur deux machines différentes.