Objets répartis : le protocole RMI

Olivier Caron

Polytech Lille
Avenue Paul Langevin Cité Scientifique Lille 1
59655 Villeneuve d'Ascq cedex

http://ocaron.plil.fr
Olivier.Caron@polytech-lille.fr

© Auteur inconnu

Vous ne pouvez pas comprendre la récursivité sans avoir d'abord compris la récursivité.

Plan

- Introduction aux objets répartis

Plan

- Introduction aux objets répartis
- Application au framework Java RMI

Les différentes approches existantes (1/2)

- Les points communs :

Les différentes approches existantes (1/2)

- Les points communs :
  - Construire de manière **modulaire** et *simplement* des applications réparties.

Les différentes approches existantes (1/2)

- Les points communs :
  - Construire de manière **modulaire** et *simplement* des applications réparties.
  - L'approche objet est un début de solution puisqu'elle apporte encapsulation et modularité.

Les différentes approches existantes (1/2)

- Les points communs :
  - Construire de manière **modulaire** et *simplement* des applications réparties.
  - L'approche objet est un début de solution puisqu'elle apporte encapsulation et modularité.
  - Il ne reste donc plus qu'à définir un protocole réseau pour faire communiquer des objets entre eux.

Les différentes approches existantes (2/2)

- L'approche DCOM de microsoft (ex. OLE) réservée au monde Microsoft !

Les différentes approches existantes (2/2)

- L'approche DCOM de microsoft (ex. OLE) réservée au monde Microsoft !
  - mono-plateforme, multi-langages

Les différentes approches existantes (2/2)

- L'approche DCOM de microsoft (ex. OLE) réservée au monde Microsoft !
  - mono-plateforme, multi-langages
- La spécification RMI de javasoft.

Les différentes approches existantes (2/2)

- L'approche DCOM de microsoft (ex. OLE) réservée au monde Microsoft !
  - mono-plateforme, multi-langages
- La spécification RMI de javasoft.
  - multi-plateformes, mono-langage (Java)

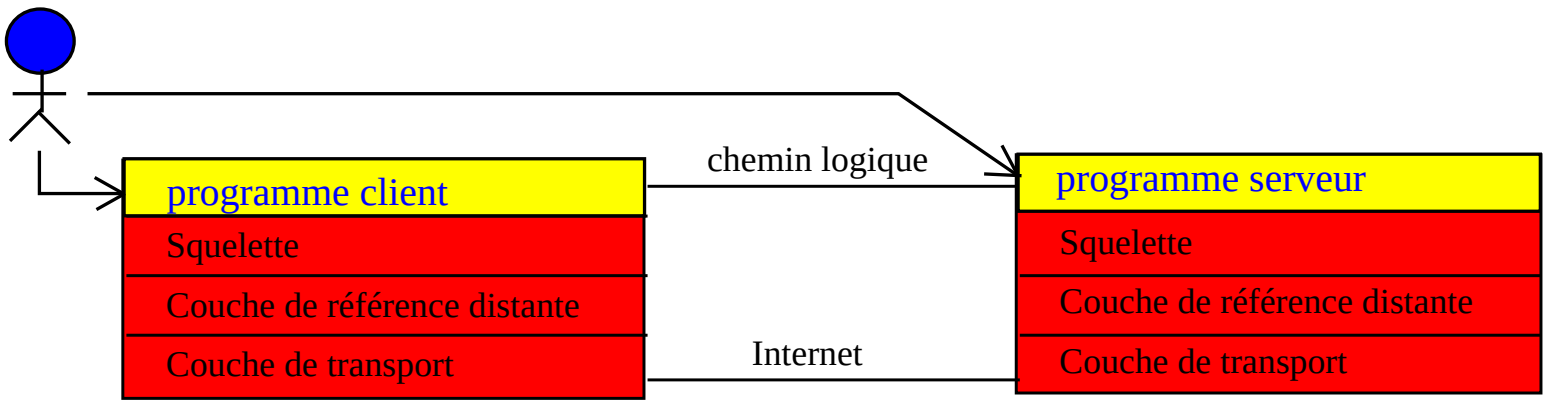
Les différentes approches existantes (2/2)

- L'approche DCOM de microsoft (ex. OLE) réservée au monde Microsoft !
  - mono-plateforme, multi-langages
- La spécification RMI de javasoft.
  - multi-plateformes, mono-langage (Java)
- La norme CORBA (IDL, . . .)

Les différentes approches existantes (2/2)

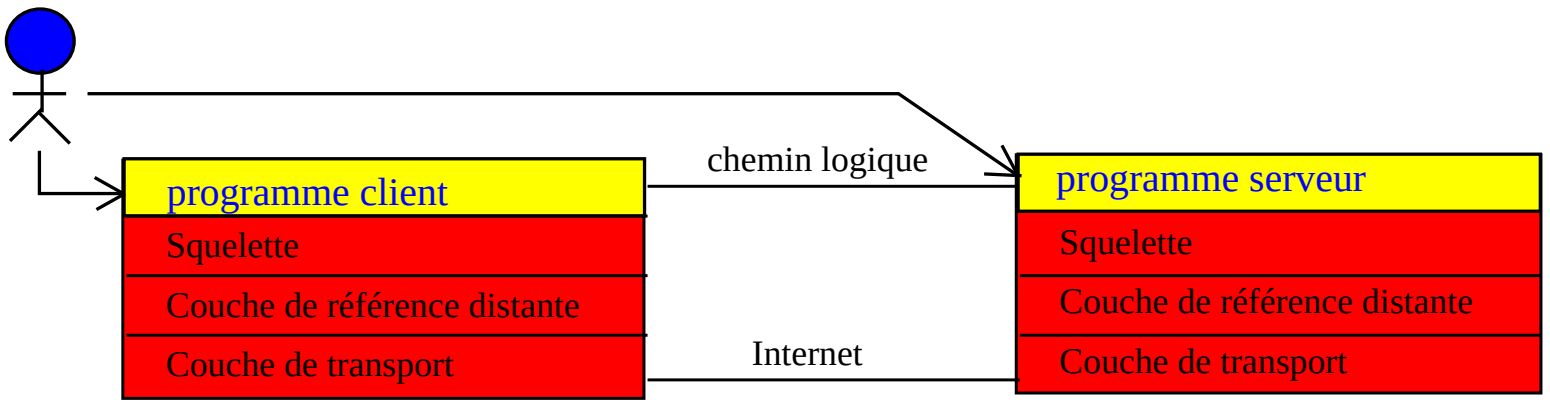
- L'approche DCOM de microsoft (ex. OLE) réservée au monde Microsoft !
  - mono-plateforme, multi-langages
- La spécification RMI de javasoft.
  - multi-plateformes, mono-langage (Java)
- La norme CORBA (IDL, . . .)
  - multi-plateformes, multi-langages

Principe de RMI



- mono-langage : Java

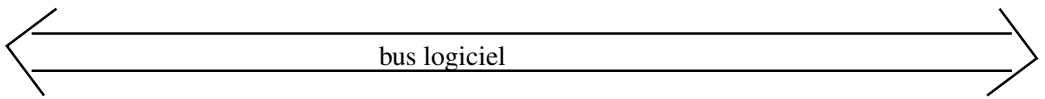
Principe de RMI



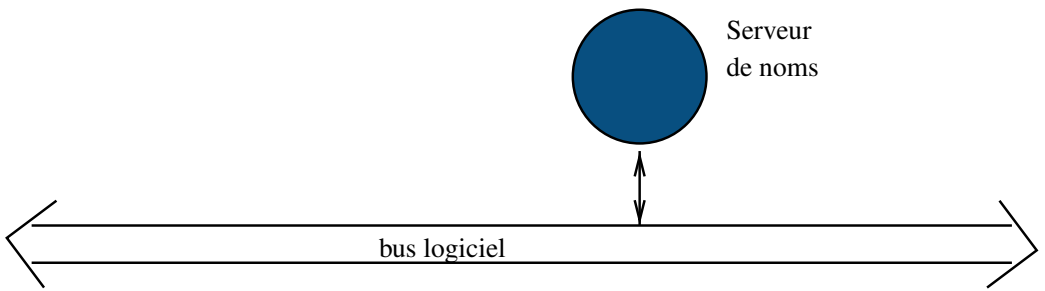
- mono-langage : Java
- multi-plateformes : Java :-)

Cycle de vie d'une application répartie

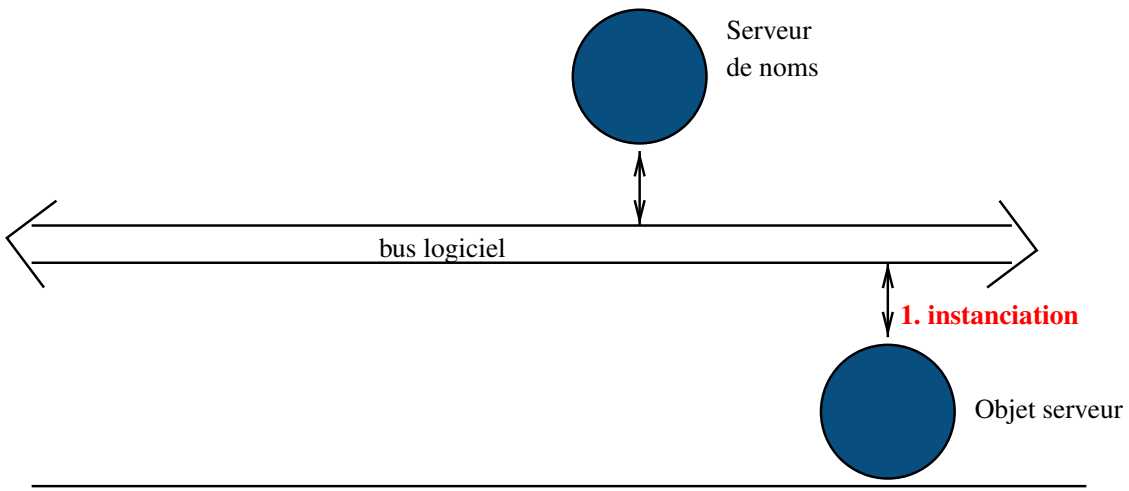
Cycle de vie d'une application répartie



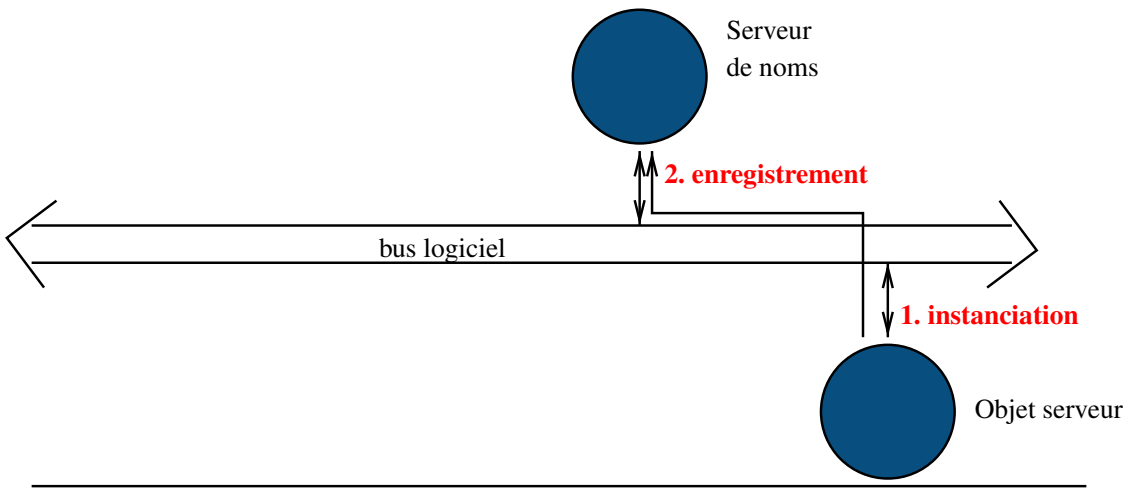
Cycle de vie d'une application répartie



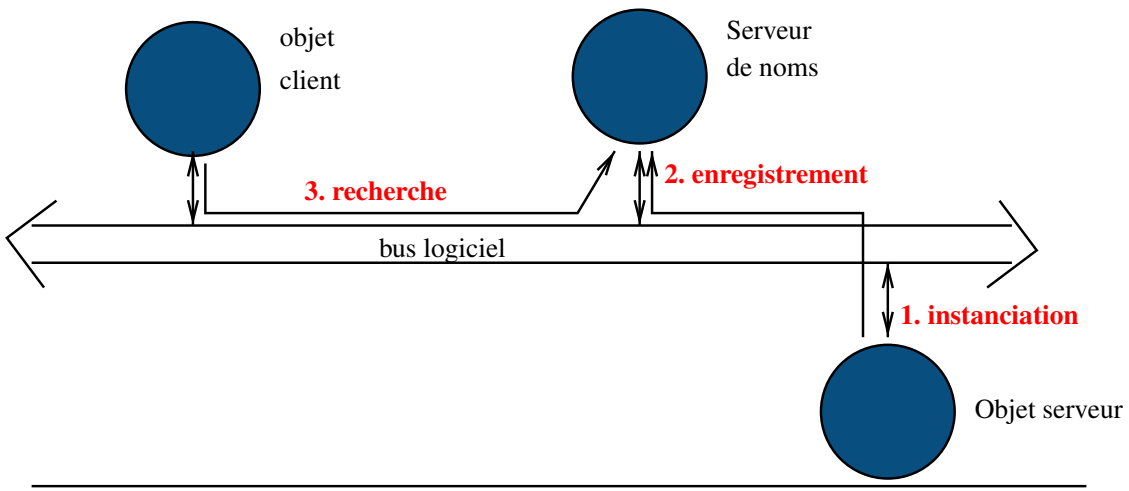
Cycle de vie d'une application répartie



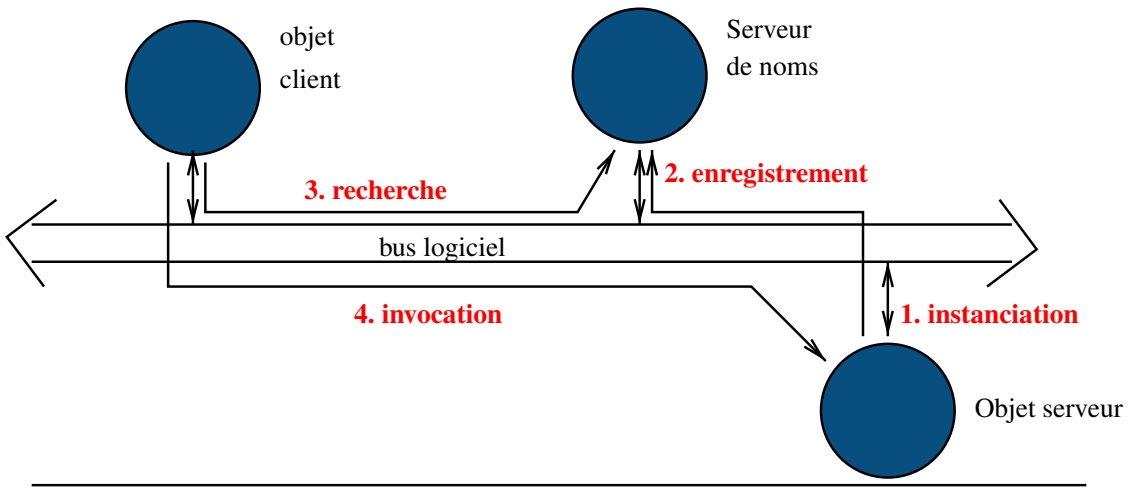
Cycle de vie d'une application répartie



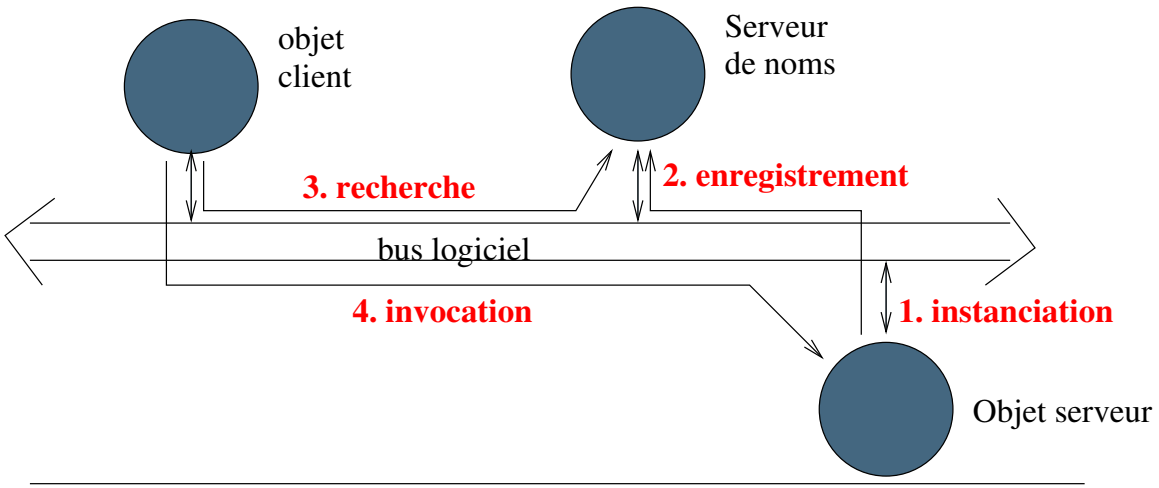
Cycle de vie d'une application répartie



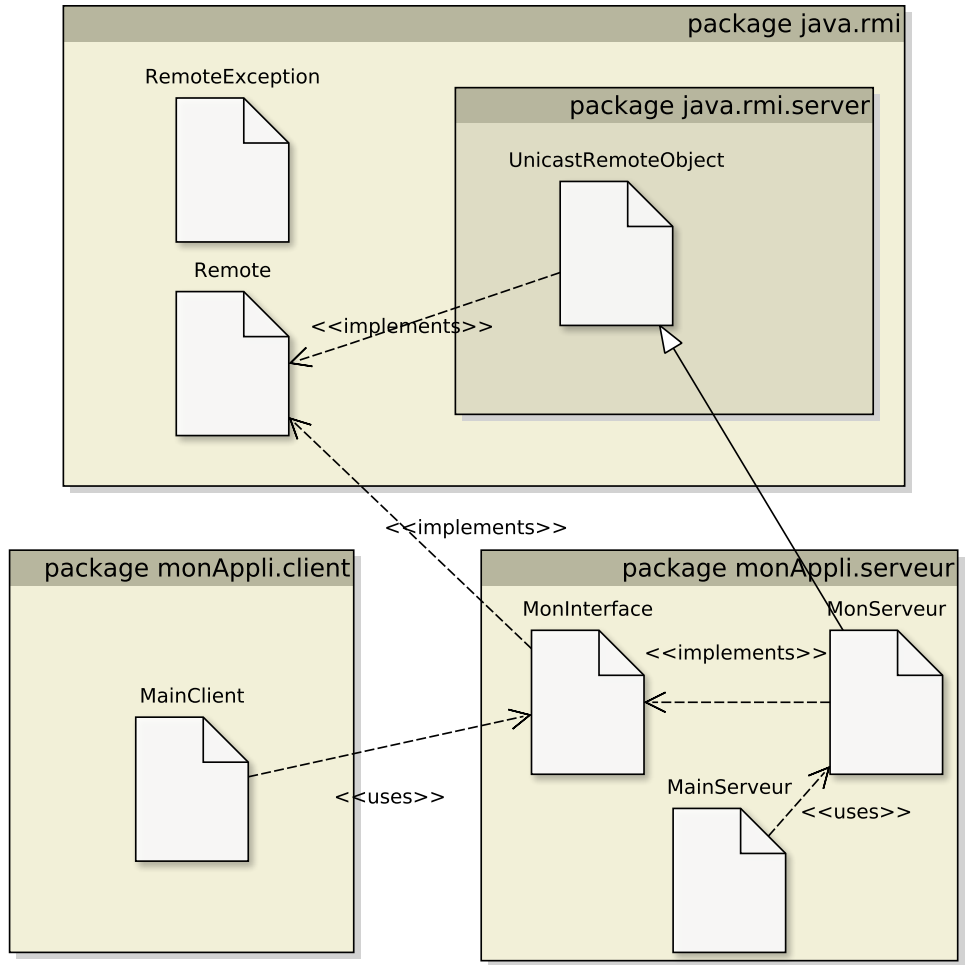
Cycle de vie d'une application répartie



Cycle de vie d'une application répartie



Le framework Java RMI



L'application Hello répartie

- **Définition de l'interface :**

L'application Hello répartie

- **Définition de l'interface :**
  - Le client ne connaît l'objet distant QUE par son interface.

L'application Hello répartie

- **Définition de l'interface :**
  - Le client ne connaît l'objet distant QUE par son interface.
  - permet de cacher certaines méthodes.

L'application Hello répartie

- **Définition de l'interface :**
  - Le client ne connaît l'objet distant QUE par son interface.
  - permet de cacher certaines méthodes.
  - permet d'adapter les objets distants selon les clients (héritage multiple d'interfaces Java)

L'application Hello répartie

- **Définition de l'interface :**
  - Le client ne connaît l'objet distant QUE par son interface.
  - permet de cacher certaines méthodes.
  - permet d'adapter les objets distants selon les clients (héritage multiple d'interfaces Java)
- Diagramme de séquences UML favorise la définition des interfaces.

Définition de l'interface (1/2)

- Il est nécessaire de :

Définition de l'interface (1/2)

- Il est nécessaire de :
  - de faire hériter l'interface par l'interface `java.rmi.Remote`

Définition de l'interface (1/2)

- Il est nécessaire de :
  - de faire hériter l'interface par l'interface `java.rmi.Remote`
  - de préciser l'exception `java.rmi.RemoteException` pour toutes les méthodes de l'interface.

Définition de l'interface (2/2)

```
package gis4.al;

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

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

Construction de l'objet serveur distant (1/2)

- Il est nécessaire de (framework RMI) :

Construction de l'objet serveur distant (1/2)

- Il est nécessaire de (framework RMI) :
  - de faire hériter la classe de `java.rmi.server.UnicastRemoteObject` (gestion réseau)

Construction de l'objet serveur distant (1/2)

- Il est nécessaire de (framework RMI) :
  - de faire hériter la classe de `java.rmi.server.UnicastRemoteObject` (gestion réseau)
  - d'implémenter les méthodes de (ou des) l'interface

Construction de l'objet serveur distant (1/2)

- Il est nécessaire de (framework RMI) :
  - de faire hériter la classe de `java.rmi.server.UnicastRemoteObject` (gestion réseau)
  - d'implémenter les méthodes de (ou des) l'interface
  - de traiter `java.rmi.RemoteException` pour toutes les fonctions (fonction constructeur comprise).

Construction de l'objet serveur distant (2/2)

```
package gis4.al ;

import java.rmi.server.UnicastRemoteObject ;
import java.rmi.RemoteException ;

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

  public Hello(String message) throws RemoteException {
    super() ; this.message= "Bonjour de "+message ;
  }
  public String getHello() throws RemoteException {
    return this.message ;
  }
}
```

Génération des fichiers

- Exploitation du design pattern Proxy:

Génération des fichiers

- Exploitation du design pattern Proxy:
- Version Java $<$ 1.5 : souche réseau (stub) géneré par le compilateur `rmic`

Génération des fichiers

- Exploitation du design pattern Proxy:
- Version Java $<$ 1.5 : souche réseau (stub) géneré par le compilateur `rmic`
- Version Java $>=$ 1.5 : stub généré dynamiquement (dynamic class loader + reflection Java)

Génération des fichiers

- Exploitation du design pattern Proxy:
- Version Java $<$ 1.5 : souche réseau (stub) géneré par le compilateur `rmic`
- Version Java $>=$ 1.5 : stub généré dynamiquement (dynamic class loader + reflection Java)
- Compilation:

`javac *.java -d .`	$\rightarrow$	génère `gis4/al/*.class`
`rmic gis4.al.Hello`	$\rightarrow$	lecture de `gis4.al.Hello.class`
(rmic nécessaire Java $<$1.5)	$\rightarrow$	génère `gis4.al.Hello_Stub.class`

Lancement du serveur

Il faut,au préalable, lancer le registre de noms (rmiregistry) qui se comporte comme un DNS, il associe un nom à un objet.
Le programme va servir à :

- Créer l'objet

Lancement du serveur

Il faut,au préalable, lancer le registre de noms (rmiregistry) qui se comporte comme un DNS, il associe un nom à un objet.
Le programme va servir à :

- Créer l'objet
- Référencer l'objet créé au registre de noms en lui attribuant un nom.

Lancement du serveur

Il faut,au préalable, lancer le registre de noms (rmiregistry) qui se comporte comme un DNS, il associe un nom à un objet.
Le programme va servir à :

- Créer l'objet
- Référencer l'objet créé au registre de noms en lui attribuant un nom.
- Le nom est une url :
  `http://nomMachineRegistre/chaine`

L'API java.rmi.Naming

La classe `java.rmi.Naming` dispose des méthodes suivantes :

méthode	**exceptions**
static void rebind(String nom,Remote obj)	MalformedURLException,RemoteException
static void bind(String nom,Remote obj)	MalformedURLException,RemoteException ,AlreadyBoundException
static void unbind(String nom)	MalformedURLException,RemoteException ,NotBoundException
static Remote lookup(String nom)	MalformedURLException,RemoteException ,NotBoundException
static String [] list(String nom)	MalformedURLException,RemoteException

Le paramètre `nom` désigne l'URL.

Le programme Serveur

```
package gis4.al ;
import java.rmi.RemoteException ;
import java.net.MalformedURLException ;
import java.rmi.Naming ;
public class Serveur {
  public static void main(String args[]) {
    try {
      Hello h=new Hello(args[0]) ;  Naming.rebind(args[0],h) ;
      System.out.println("Serveur "+args[0]+" pret") ;
    } catch(RemoteException e1) {
      System.err.println("Erreur reseau dans hello") ;
    } catch(MalformedURLException e2) {
      System.err.println("Erreur adresse reseau dans Hello") ;
    } catch(ArrayIndexOutOfBoundsException e3) {
      System.err.println("Erreur: java gis4.al.Serveur nom") ;
    }
  }
}
```

Execution (1/2)

```
%> hostname
 phinaert02
%> rmiregistry &
%> java gis4.al.Serveur rmi://phinaert02/hello &
 Serveur rmi://phinaert02/hello pret
%> java gis4.al.Serveur rmi://phinaert02/coucou &
 Serveur rmi://phinaert02/coucou pret
%>
```

- Objets serveurs et registre de noms sont toujours co-localisés
- Lors du lancement du registre de noms, le `CLASSPATH` doit contenir les répertoires/jars des classes d'objets à enregistrer.

Execution (2/2)

- Le registre de noms est consultable à l'adresse `rmi://phinaert02`
- Les objets distants (instances de `Hello`) sont référencés dans le serveur de noms par les adresses URL : `rmi://phinaert02/hello` et `rmi://phinaert02/coucou`
- On peut accéder au serveur de noms à distance !
- Pas de hiérarchie dans les noms (à la différence de CORBA, JNDI)

Consultation des objets enregistrés

```
package gis4.al ;
import java.rmi.* ;   import java.net.* ;

public class Consultation {
  public static void main(String args[]) {
    try {
      String noms[] = Naming.list("rmi://"+args[0]) ;
      for (String nom : noms) System.out.println(nom) ;
    } catch(ArrayIndexOutOfBoundsException e1) {
      System.err.println("erreur Usage: "+
          "java gis4.al.Consultation nomServeur") ;
    } catch(MalformedURLException e3) {
      System.err.println("erreur syntaxe URL") ;
    } catch (RemoteException e4) {
      System.err.println("Erreur accès réseau") ;
} } }
```

Execution programme Consultation

```
%> java gis4.al.Consultation phinaert02
 //phinaert02:1099/coucou
 //phinaert02:1099/hello
```

Le programme client (1/3)

- Consultation du serveur de noms (lookup)
- Récupération d'une référence distante
- Utilisation comme un objet local (sauf `RemoteException`)

Le programme client (2/3)

```
package gis4.al ;

import java.rmi.* ;
import java.net.* ;

public class HelloClient {
  public static void main(String args[]) {
    try {
      HelloInterface h=
      (HelloInterface) Naming.lookup("rmi://"+args[0]+"/"+args[1]) ;
      String message = h.getHello() ;
      System.out.println(message) ;
```

Le programme client (3/3)

```
        } catch(ArrayIndexOutOfBoundsException e1) {
          System.err.println("erreur Usage: "+
              "java gis4.al.HelloClient nomServeur nomObjet") ;
        } catch(NotBoundException e2) {
          System.err.println(args[1]+" non référencé")   ;
        } catch(MalformedURLException e3) {
          System.err.println("erreur syntaxe URL") ;
        } catch (RemoteException e4) {
          System.err.println("Erreur accès réseau") ;
        }
    }
}
```

Execution Client

```
%> java gis4.al.HelloClient phinaert02 coucou
 Bonjour de rmi://phinaert02/coucou
%> java gis4.al.HelloClient phinaert02 hello
 Bonjour de rmi://phinaert02/hello
%> java gis4.al.HelloClient phinaert02 ola
 ola non référencé
```

Quelques distinctions (1/3)

```
HelloInterface h1 = (HelloInterface)
Naming.lookup("rmi://"+args[0]+"/hello") ;
HelloInterface h2 = (HelloInterface)
  Naming.lookup("rmi://"+args[0]+"/hello") ;
  if (h1==h2) System.out.println("meme objet") ;
```

Quelques distinctions (2/3)

```
import java.rmi.server.* ;
import java.rmi.* ;
public class Hello extends UnicastRemoteObject
  implements HelloInterface {
  public Hello() throws RemoteException { }
  public String getHello(Bonjour b)
    throws RemoteException {
    return b.getBonjour() ;
  }
}
```

```
public class Bonjour implements java.io.Serializable {
   public String getBonjour() { return "bonjour" ; }
}
```

Quelques distinctions (3/3)

- Activation à distance impossible sauf :
  - Système d'activation JDK 1.2 (`java.rmi.activation`)
  - Le **design pattern** fabrique

La fabrique générique (version base de données)

```
import java.rmi.* ;
public interface TypeObjectFactoryInterface extends Remote {
  public TypeObject create(TypeObjectKey id)
    throws RemoteException, FoundException ;
  public TypeObject findByPrimaryKey(TypeObjectKey id)
    throws RemoteException, NotFoundException ;
  public void destroy(TypeObjectKey id)
    throws RemoteException, NotFoundException ;
}
```

La fabrique `Hello`

```
import java.rmi.* ;

public interface HelloFabriqueInterface extends Remote {
  public HelloInterface create(String msg) throws RemoteException ;
}
```

```
import java.rmi.server.* ;
import java.rmi.* ;
public class HelloFabrique extends UnicastRemoteObject
  implements HelloFabriqueInterface {
  public HelloFabrique () throws RemoteException {}
  public HelloInterface create(String msg) throws RemoteException {
    return new Hello(msg) ;
  }
}
```

Conclusion

- Simple mais efficace.
- Trop simple ? (la classe `Naming`)
- Couplage possible avec CORBA.

Le mot de la fin

© Mitch Ratcliffe

Un ordinateur vous permet de faire plus de bêtises, beaucoup plus rapidement que n'importe quelle autre invention dans l'histoire de l'humanité. A l'exception notable des armes à feu et de la tequila.