Remote Method Invocation

Cnam Paris
jean-michel Douin, douin au cnam point fr
22 Janvier 2008

Notes de cours consacrées à RMI

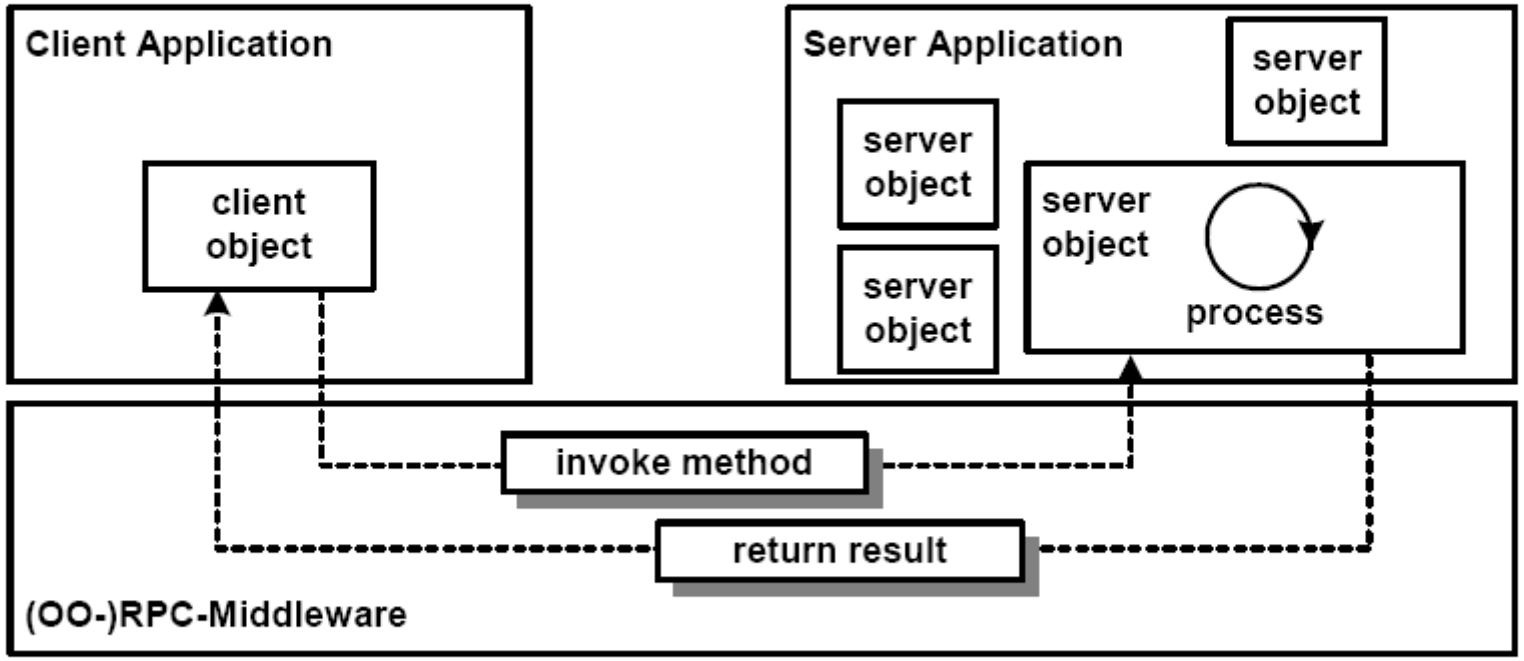
Principale **bibliographie**

- RMI
  http://today.java.net/pub/a/today/2004/06/01/RMI.html?page=1RMI
  http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html
  http://java.sun.com/docs/books/tutorial/rmi/index.html

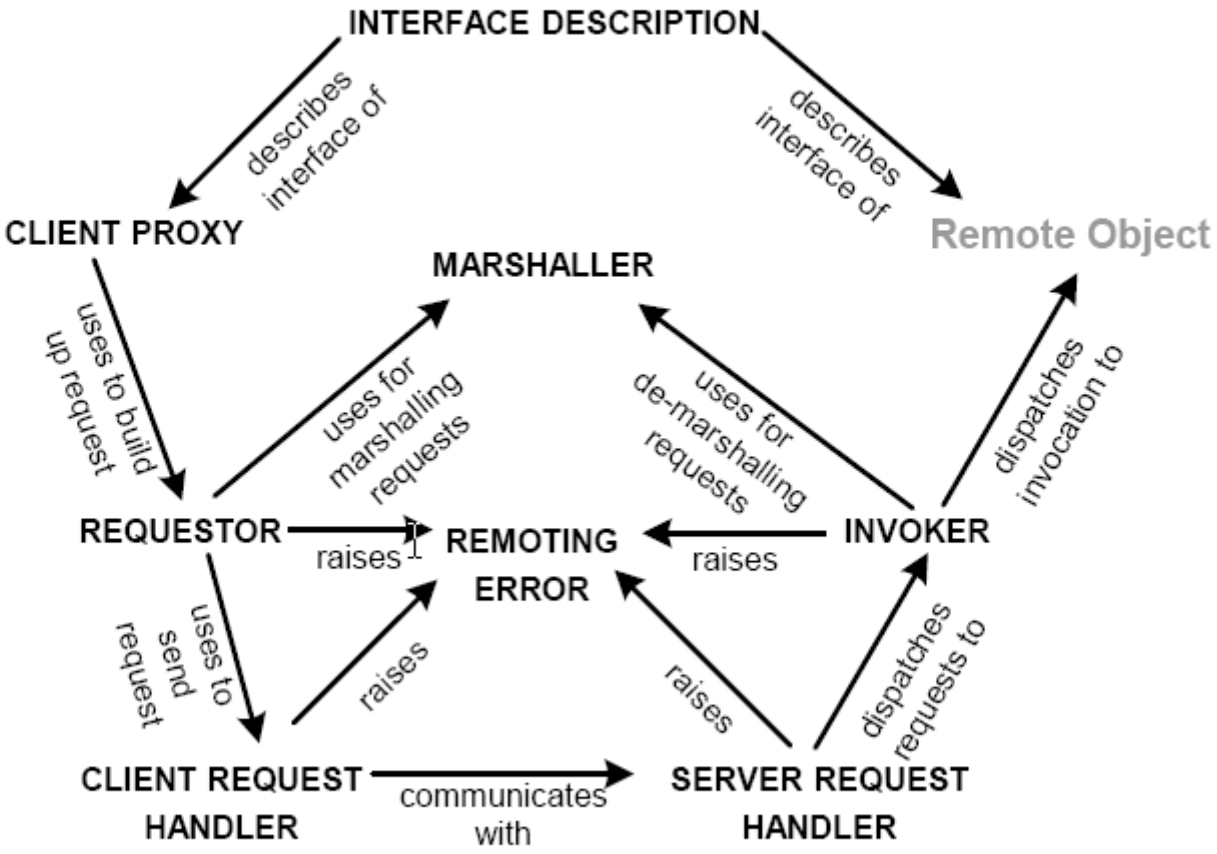
- http://www.infosys.tuwien.ac.at/Staff/**zdun**/teaching/evs/**evs.pdf**

- Pattern : Proxy, Adapter
  http://www.transvirtual.com/users/peter/patterns/overview.html
  http://www.eli.sdsu.edu/courses/spring98/cs635/notes/index.html

Remoting Style: RPC



Basic Remoting Patterns



Différentes étapes

- **Côté serveur**
  1. Création de l'annuaire, un service de nommage des services
  2. Création du service, enregistrement de celui-ci auprès de l'annuaire
  3. Un bail de ce service, convenu entre l'annuaire et le service effectif
  4. Le service reste actif, attend les requêtes des clients
     1. Une variante comme le chargement du service à la demande existe

- **Côté client**
  1. Interrogation de l'annuaire, à propos du service
  2. En retour, réception du mandataire chargé de la communication
  3. Exécution distante du service
     1. Emballage des paramètres, sélection de la méthode
     2. Déballage des résultats retournés

Java - RMI : les bases

- **TCP/IP**

- **Java uniquement : RMI/JRMP**
  - **Le protocole Java Remote Method Protocol JRMP**
  - **JVM (Java Virtual Machine) clients comme serveurs**

- **Ouverture aux autres**

avec RMI/IIOP-CORBA/IIOP

  - **Internet Inter-ORB Protocol/Common**
  - **Voir les outils du jdk : *idlj, tnameserv, orbd***

Sommaire : mise en oeuvre

- **Chapitre 1**
  - Par l'exemple : Un client et un serveur de méthodes
  - téléchargement : le rôle de **rmiregistry**
  - Le serveur et (**rmic** ou l'utilisation du Pattern Proxy, depuis 1.5 DynamicProxy)
- **Chapitre 2**
  - Passage de paramètres et retour de fonctions
  - Les exceptions
- **Chapitre 3**
  - Téléchargement du code
  - Un serveur http : pour le téléchargement de classes
- **Vers une méthode de développement**
  - Méthode : Usage du Pattern Adapter
  - Exemple : Un client et un serveur de tâches
- **Critiques**
  - Depuis le JDK1.2 **rmid** et la classe java.rmi.activation.Activatable, un autre support
- **Annexe : un « Tchache / logiciel de causerie »**

Objectifs en images : deux machines


```
public class Client{
  ServeurDeMethodes s= ...

  s.methodeLointaine();

}
```

RMI

```
public class ServeurDeMethodes{

  void methodeLointaine(){
    System.out.println("hellooo");
  }

}
```

==> <Quoi, .

- *Où se trouve le serveur ?*
- *Comment déclenche t-on " void methodeLointaine()" ?*
- *Les accès sont-ils sécurisés ?*
- *Et les paramètres, les exceptions, les résultats ? ==> chapitre 2*

Comment>$_1$

- *Où se trouve le serveur ?*
- *Comment déclenche t-on " void methodeLointaine()" ?*
- *Les accès sont-ils sécurisés ?*

- La "machine Serveur" est identifiée par une adresse IP
  - un nom octroyé par l'administrateur du DNS
- La "classe ServeurDeMethodes" est associée à un nom répertorié
  - Un serveur de noms
- La communication est prise en charge par une procuration fournie par le serveur au client , patron Client Proxy
- La classe serveur autorise les accès distants
  - précise les contraintes d 'accès aux fichiers en général un fichier "**.policy**"

Interface commune

interface Commune « AffichageLointain »

```
void methodeLointaine();
```

C'est le Langage commun

Développement en Java, principes

- **Une interface** précise les méthodes distantes,
  - Elle est commune au serveur et aux clients,
  - Elle hérite (au sens Java entre interfaces) de java.rmi.Remote
    - (un marqueur, `public interface Remote{}`)

Le serveur et ses clients

- **Le serveur** hérite* d'une classe prédéfinie
  package java.rmi.server et
  s'inscrit auprès d'un gestionnaire de Noms/services

- **Ses clients**
  recherchent le service proposé,
  obtiennent une référence de l'objet distant,
  effectuent les appels de méthodes habituels,
    ces méthodes étant déclarées dans l'interface commune

** c'est une façon de faire, il y en a d'autres …*

Développement en Java : mode d'emploi

- **Un gestionnaire de noms/services (patron Registry)**
  - **rmiregistry est un des outils pré-installé**

- **Ce gestionnaire de noms/services est accessible**
  - Enregistrement du service

    java.rmi.Naming.bind java.rmi.Naming.rebind
  - Lecture du service

    java.rmi.Naming.lookup

Développement en Java : mode d'emploi

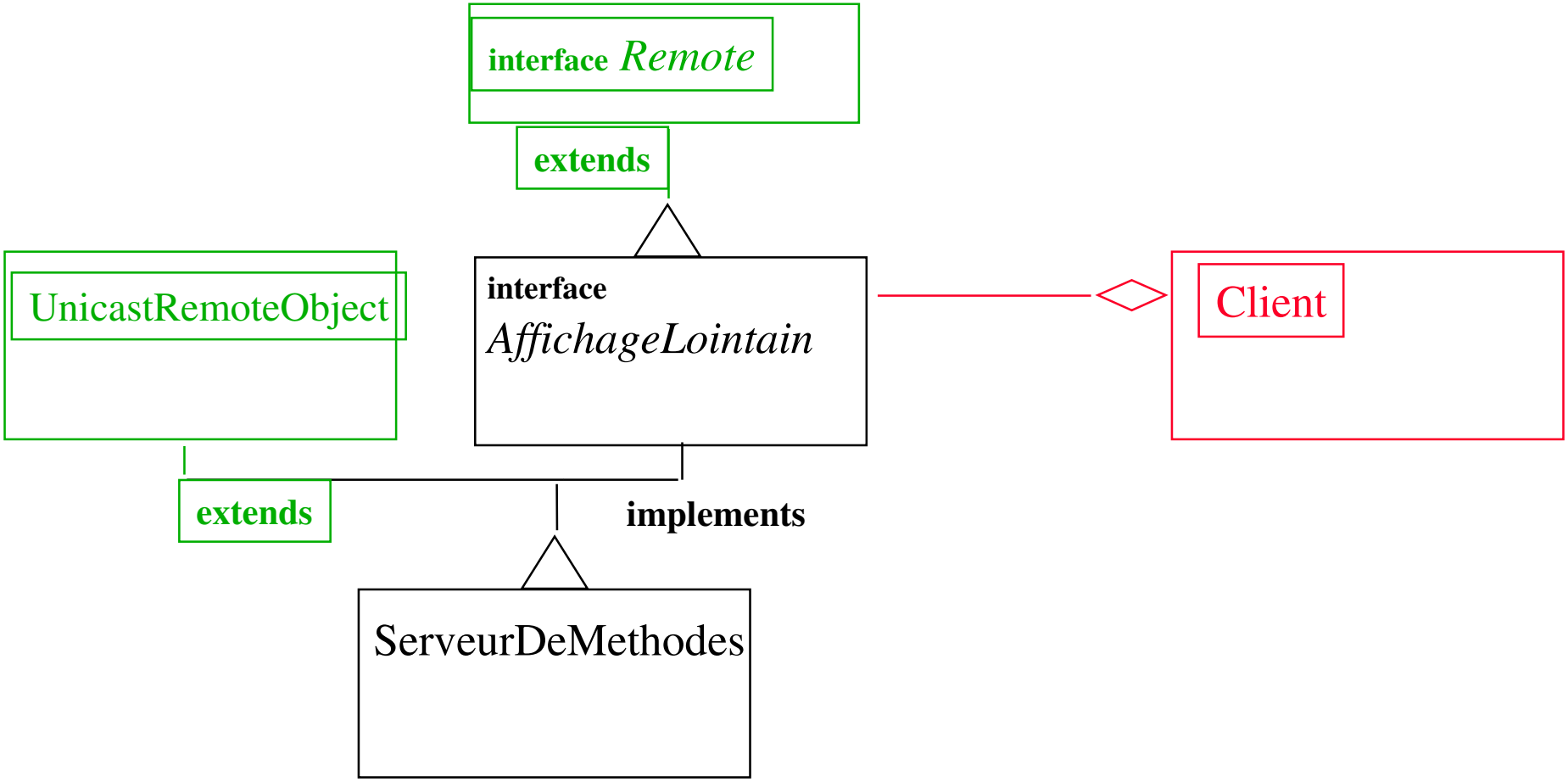
- **L ' interface est commune** aux clients et au serveur
  - Hérite de ***java.rmi.Remote***
  - Recense les méthodes distantes,
  - Chaque méthode possède la clause ***throws java.rmi.RemoteException***

- **La classe « Serveur »**
  - hérite de ***java.rmi.server.UnicastRemoteObject***
  - implémente les méthodes de l 'interface commune,
    - sans oublier le constructeur qui possède également la clause *throws*.
  - Le serveur propose ses services
    - Voir ***java.rmi.Naming.rebind***

- **Les classes Clientes**
  - Utilisent une instance/référence de la classe « Serveur »
    - ***Voir java.rmi.Naming.lookup.***

- *Et c'est tout !!!*

Exemple : un serveur de méthodes



Le serveur de méthodes

un client

Exemple : Interface commune aux clients et au serveur

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

public interface AffichageLointain extends Remote{

  public void methodeLointaine() throws RemoteException;

  public static final String nomDuService = "leServeurDeMethodes";
}
```

L'interface commune doit :

- hériter de l'interface **`java.rmi.Remote`**
- pour chaque méthode ajouter la clause **`throws java.rmi.RemoteException`**
- être publique

Exemple : Le serveur (machine lmi86)

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class ServeurDeMethodes    extends UnicastRemoteObject
                                  implements AffichageLointain{

  public void methodeLointaine() throws RemoteException{
    System.out.println("helloooo");
  }
  public ServeurDeMethodes () throws RemoteException{}

  public static void main(String[] args) throws Exception{
     try{
       AffichageLointain serveur = new ServeurDeMethodes();
       Naming.rebind(AffichageLointain.nomDuService, serveur);
       System.out.println("Le serveur lointain est pret");
    }catch(Exception e){throw e;}
  }}
```

Exemple : Le client (machine lmi27)

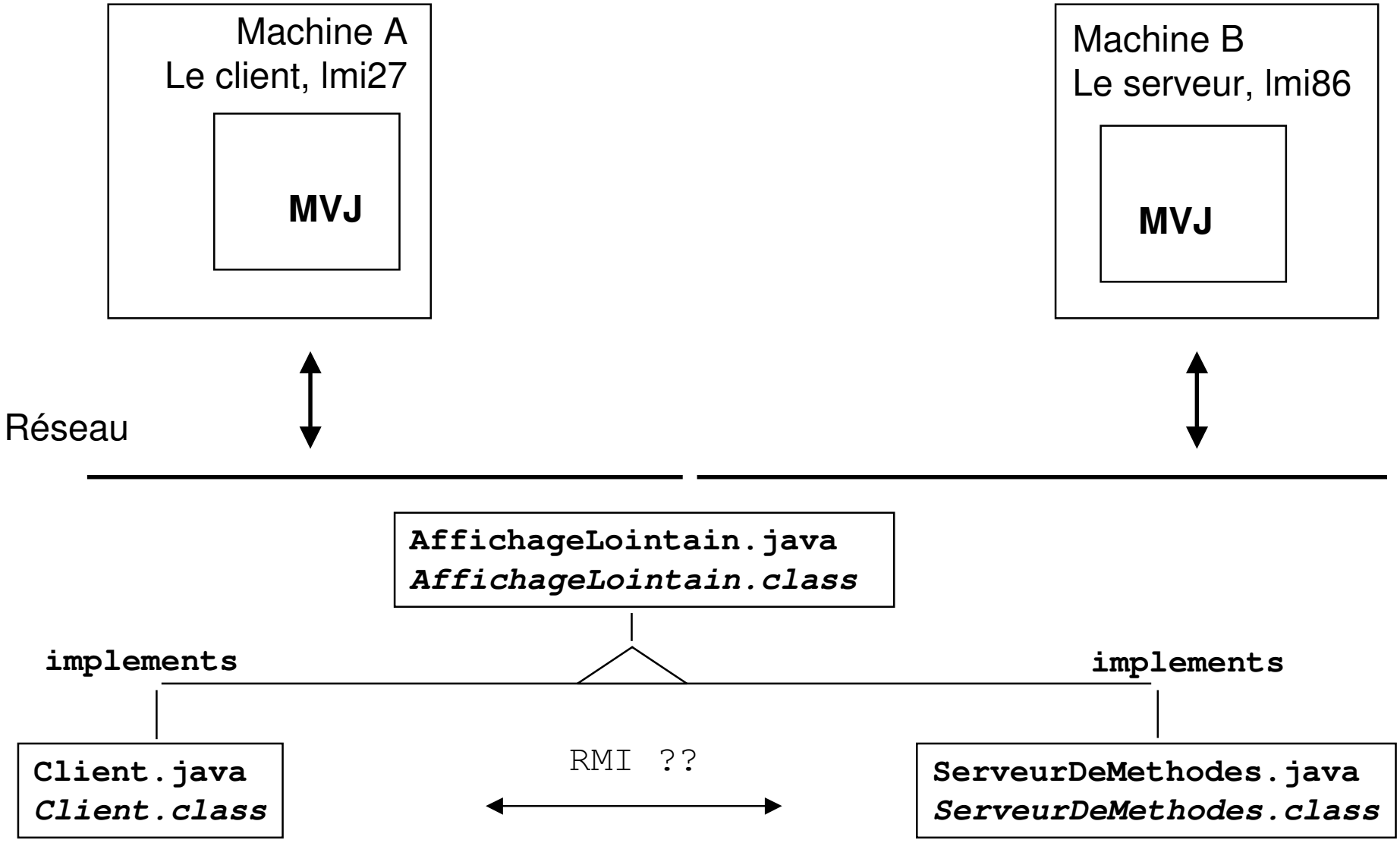
```
import java.rmi.*;
public class Client{

  public static void main(String[] args) throws Exception{
    System.setSecurityManager(new RMISecurityManager());
    AffichageLointain serveur = null;
    String nomComplet;

    nomComplet ="rmi://lmi86/" + AffichageLointain.nomDuService;
    try{
      serveur = (AffichageLointain) Naming.lookup(nomComplet);
      serveur.methodeLointaine();
    }catch(Exception e){ throw e;}
 }}
```

note : rmi://lmi86/ ou rmi://lmi86:1099/ ou //lmi86 (1099 est le port par défaut utilisé par rmiregistry)

Architecture



Comment ? : les commandes, l'interface

- L'interface est partagée ou recopiée
  - ***Compilation de l'interface commune aux clients et au serveur***
  - > **javac** AffichageLointain.java
  - *Le fichier AffichageLointain.class est partagé entre **lmi86** et **lmi27** ou*
  - *recopié sur ces deux machines dans les répertoires*
    - *Par exemple*

      d:/rmi/serveurDeMethodes/serveur/

      d:/rmi/serveurDeMethodes/client/

Comment ?

- Où se trouve le code nécessaire ?
  - Le classpath pour rmi & rmiregistry, voir RMIClassLoader
  - -Djava.rmi.server.**codebase=une URL file:// http:// ftp://**
  - *format du paramètre codebase : protocol://host[:port]/file (séparés par un blanc)*
  - *-Djava.rmi.server.useCodebaseOnly=true protocol ::= ftp | http |file*

- Quelles stratégie de sécurité ?
  - Djava.security.policy=java.policy
  - Un exemple de stratégie des plus laxiste

```
grant{
   permission java.security.AllPermission;
};
```

Comment ? : les commandes, le serveur

- **//lmi86/d:/rmi/serveurDeMethodes/serveur/**
  - ***Compilation du serveur***
  - > **javac** ServeurDeMethodes.java
  - > **rmic** ServeurDeMethodes (inutile si >=1.5, un proxy est généré dynamiquement)
  - ***Exécution***
  - C:\> **start** rmiregistry
  - *(attention au CLASSPATH, les fichiers .class ne doivent pas être accessibles par rmiregistry)*
  - C:\serveur_rmi> **java** -Djava.rmi.server.**codebase**=file:/D:/rmi/ServeurDeMethodes/Serveur/ -Djava.security.policy=java.policy ServeurDeMethodes
  - *Pour plus d'informations sur la console associée à rmiregistry :*
  - > **java** -Djava.rmi.server.**codebase**=file:/D:/rmi/ServeurDeMethodes/Serveur/
  - -Djava.rmi.server.logCalls=true
  - -Djava.security.policy=java.policy ServeurDeMethodes
  - Voir si besoin java.rmi.server.RMIClassLoader

(rmic inutile si >=1.5)

- **rmic** : **rmi c**ompiler, génération des fichiers _Skel.class et _Stub.class (inutile si jdk > 1.5)
- **rmiregistry** : association nom et référence Java, port 1099 envoi de « Stub » aux clients
- **java** -Djava.rmi.server.codebase=file:/D:/rmi/ServeurDeMethodes/Serveur/ *: accès au _stub.class*
- -Djava.security.policy=java.policy *: en 2.0, contraintes d 'accès*

Traces du Comment

- **Traces avec un serveur http:**
  1) **>start rmiregistry**
  2) **>start java SimpleHttpd 8080**

Coté serveur :

- > java -Djava.rmi.server.**codebase**=**http://localhost:8080/ ServeurDeMethodes/**
  -Djava……..

- **Les traces sur la console du serveur Web**

```
[LMI86/127.0.0.1] -- Request: GET /ServeurDeMethodes_Stub.class HTTP/1.1
[LMI86/127.0.0.1] -- Request: GET /AffichageLointain.class HTTP/1.1
```

Comment ? : les commandes, le Client

- **//lmi27/d:/rmi/ServeurDeMethodes/Client/**
  - **> javac Client.java**
  - **> java -Djava.security.policy=java.policy Client**
  - *Le serveur de noms lmi86 est référencé dans le source du Client par :*
  - **...**

```
nomComplet ="rmi://lmi86/" + AffichageLointain.nomDuService;
try{
  serveur = (AffichageLointain) Naming.lookup(nomComplet);
```

Traces du Comment

- **Traces avec un serveur http:**
  - **Coté client :**

1) **> java Client**

2) **Les traces sur la console du serveur Web**

```
[LMI93/127.0.0.1] -- Request: GET /ServeurDeMethodes_Stub.class HTTP/1.1
```

rmic, _skel.class, _stub.class

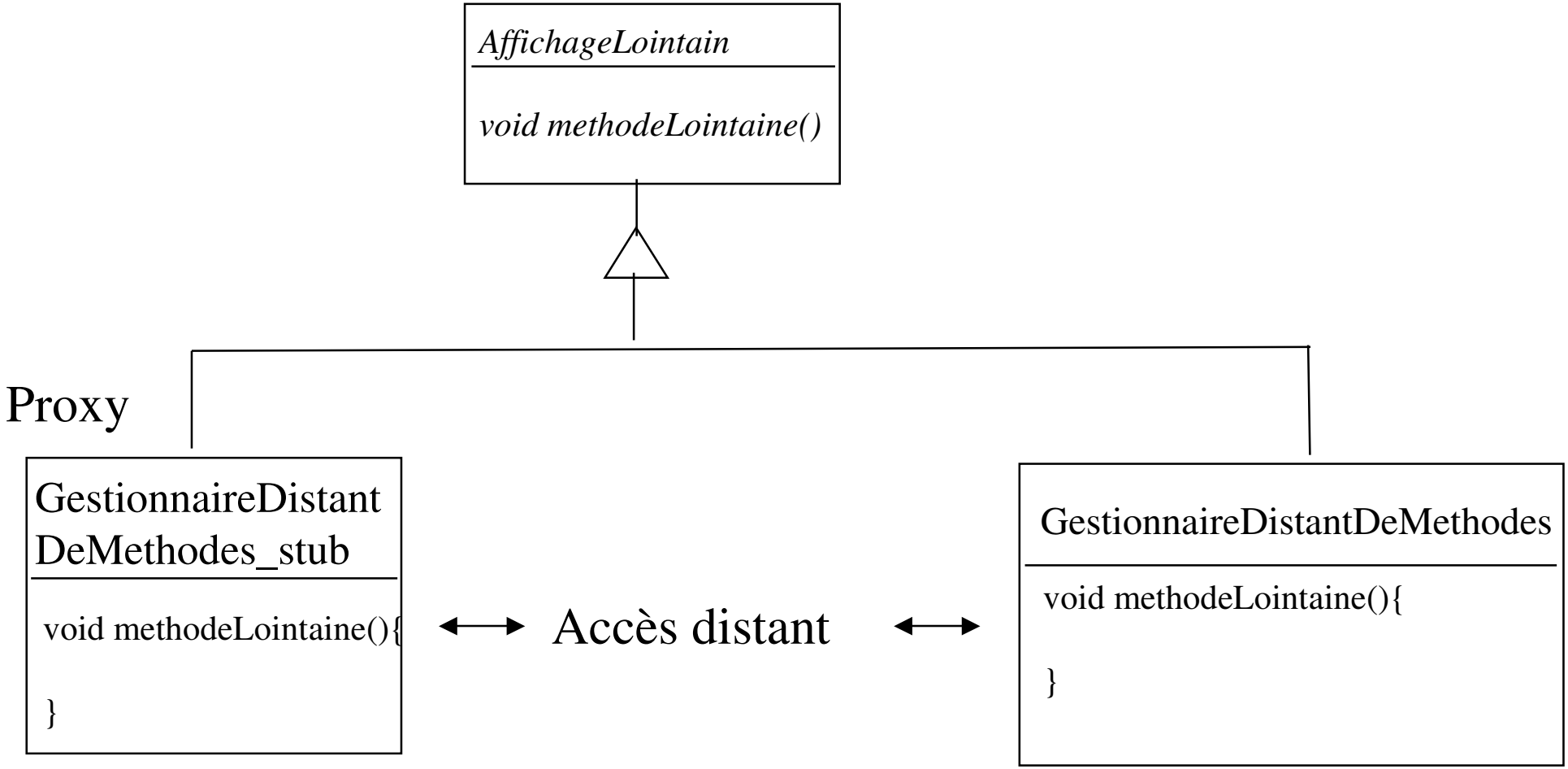
Communication et transmission des paramètres sur le réseau totalement transparents

_stub : interface réseau fournie au client

_skel : mis en forme des paramètres et résultats (inutile en 1.2)

_stub.class ou proxy si 1.5

Le pattern Client Proxy



En savoir un peu plus rmic -keep

- **rmic –keep ServeurDeMethodes**
  - **ServeurDeMethodes_stub.java**
  - **et**
  - **ServeurDeMethodes_skel.java**
- **rmic –keep –v1.2 ServeurDeMethodes**
  - **ServeurDeMethodes_stub.java**

rmiregistry

- **port 1099 par défaut**
  - **>start rmiregistry 1999**
  - *alors* `nomComplet ="rmi://lmi86:1999/" + ...`
- transfert du fichier _stub aux clients
- **Naming.rebind()** pour le serveur
- **Naming.lookup()** pour les clients

Note :

Si cet utilitaire en fonction de la variable CLASSPATH a accès aux fichiers _stub, la commande -Djava.rmi.server.codebase= xxxxx est ignorée

Voir java.rmi.registry.LocateRegistry

java.security.policy=java.policy

le fichier java.policy en exemples

```
grant{
  permission java.security.AllPermission;
};
```

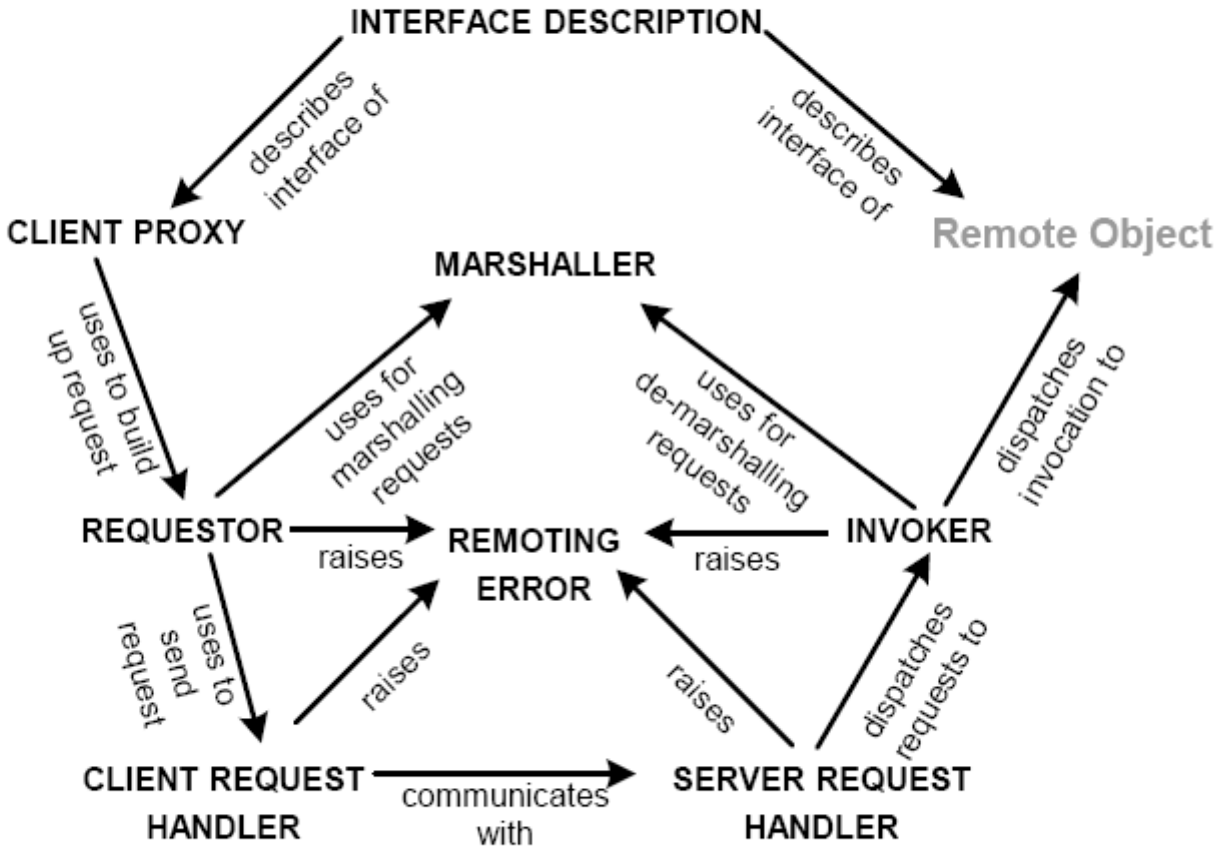
ou bien *(mieux)*

```
grant{
  permission java.net.SocketPermission "localhost",
   "connect,accept,listen";
  permission java.net.SocketPermission "lmi86.cnam.fr:8080-8089",
   "connect,accept";
};
```

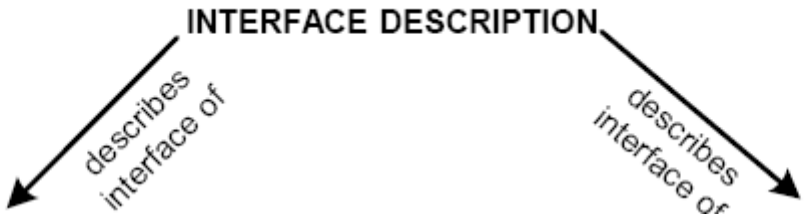
- http://java.sun.com/products/jdk/1.2/docs/guide/security/PolicyFiles.html
- http://java.sun.com/products/jdk/1.2/docs/guide/security/permissions.html

Récapitulatif

Basic Remoting Patterns

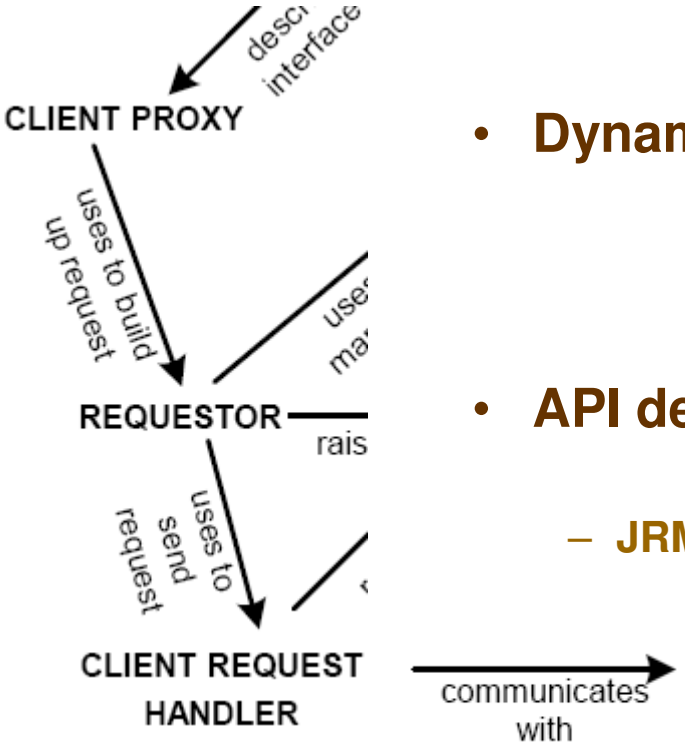


Interface commune en java



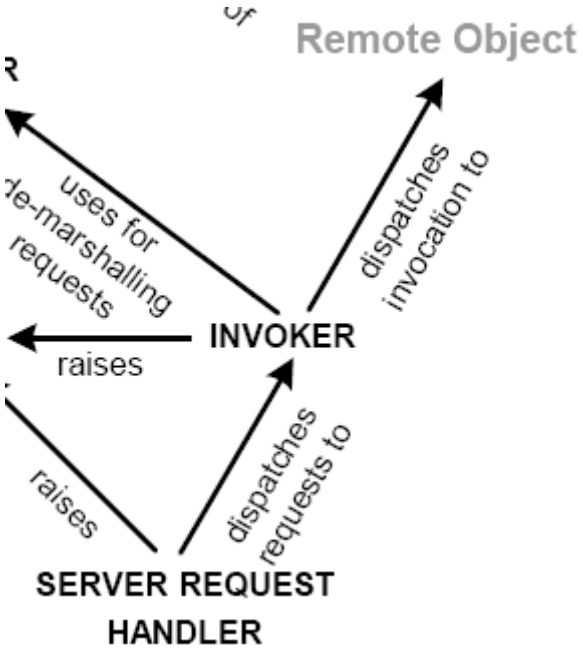
- **Interface commune ou langage commun**
  - **Entre les clients et le serveur**

Client Proxy



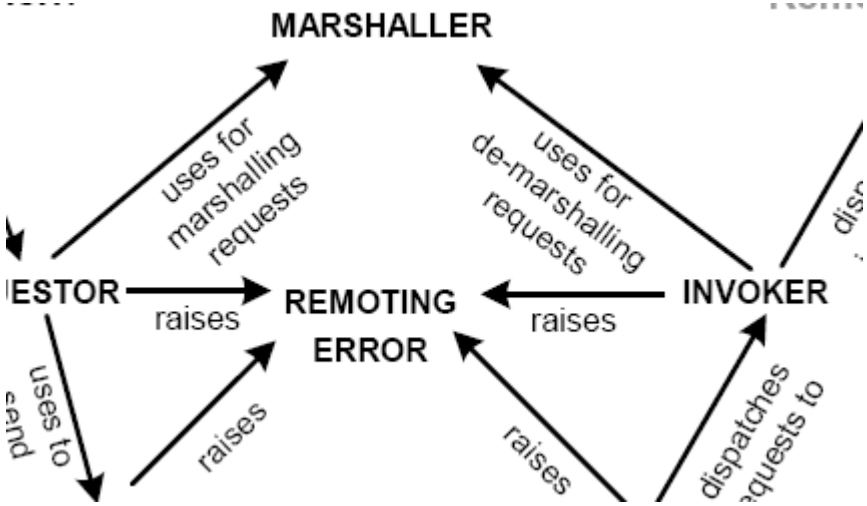
- **DynamicProxy**
- **API de java.net**
  - **JRMP & Socket**

Côté serveur



- **La classe « serveur »**
  - **extends UnicastRemoteObject**
- **API de java.net**
  - **JRMP & Socket**

Quid ?



- **Transmission des paramètres**
- **Levée d'exceptions**
  - **Marshaller comme Sérialisation …**

Chapitre 2 : Objectifs


```
public class Client{
  Calcul c = …

  res = c.moyenne(uneListe);

}
```

RMI

```
public class Calcul{

  Integer moyenne(List l){
    …

  }

}
```

==> <Quoi, .

- *Comment transmettre les paramètres ?*
- *Comment les résultats de fonction sont-ils retournés aux clients ?*
- *Une exception est levée côté serveur, comment informé le client ?*

Comment>$_2$

- *Comment transmettre les paramètres ?*
- *Comment les résultats de fonction sont-ils retournés aux clients ?*
- *Une exception est levée côté serveur, comment informé le client ?*

- Aucune incidence sur le source Java
  - excepté que les paramètres transmis doivent être des instances de **java.io.Serializable**
- Une copie des paramètres en profondeur est effectuée,
  - Serializable à tous les niveaux
- En retour de fonction, une copie est également effectuée
- Les Exceptions sont des objets comme les autres

```
interface marqueur
  public interface java.io.Serializable {}
```

Sérialisation : principes (rappels ?)

- Le paramètre est une instance de **java.io.Serializable**

```
public class XXXX implements java.io.Serializable{...}
```

Opérations internes : **- écriture** par copie de l'instance en profondeur
- lecture de l'instance

- ***Ecriture*** *de l'instance : ce qui est transmis sur le réseau*

```
OutputStream out = ...
ObjectOutputStream o = new ObjectOutputStream( out);
o.writeObject(obj);
```

*Les données d'instance sont copiées sauf les champs "**static**" et "**transient**"*

- ***Lecture*** *de l'instance : ce qui est lu depuis le réseau*

```
InputStream out = ...
ObjectInputStream o = new ObjectInputStream( out);
o.readObject();
```

Chapitre 3 : Objectifs


```
public class Client{
  ServeurDeTaches s; ...
  Horloge uneHorloge = ...

  s.executer(uneHorloge);

}
```

RMI

```
public class ServeurDeTaches{

  void executer(Runnable r){
    new Thread(r).start();

  }

}
```

==> <Quoi, .

- *Et si les paramètres utilisent des classes inconnues du serveur ?*

Comment>$_2$

Et si les paramètres utilisent des classes inconnues du serveur ?

- Le serveur les « demande » au client
  - -Djava.rmi.server.**codebase=une URL file:// http:// ftp://**
  - Côté client

Chargement du .class : Horloge.java

```
public class Horloge extends Thread
                     implements java.io.Serializable{

  public void run(){
    int sec=0,mn=0,h=0;

    while (true){
       try{
         Thread.sleep(1000);
         sec++;
         if(sec==59){
           mn++;sec =0;
           if(mn==59) { h++; mn=0; if (h==24) h=0;}
         }
        System.out.println(h + ":" + mn + ":" + sec);
       }catch(Exception e){}}}}
```

Le serveur demande "Horloge.class"

Horloge.class

```
public class ServeurDeTaches{

  void executer(Runnable r){
    new Thread(r).start();
  }

}
```

Comment est-il satisfait ?

- //lmi27/d:/rmi/ServeurDeMethodes/Client/
  - > java **-Djava.rmi.server.codebase=file:/D:/rmi/ServeurDeMethodes/Client/**
  - -Djava.security.policy=java.policy Client
  - ***ou avec l'aide d'un serveur http***
  - > start java **SimpleHttpd 8088**
  - > java **-Djava.rmi.server.codebase="http://lmi27:8088/D:/rmi/ServeurDeMethodes/Client/ http://lmi86:8088/D:/rmi/ServeurDeMethodes/Client/"**
  - -Djava.security.policy=java.policy Client

format du paramètre codebase : protocol://host[:port]/file (séparés par un blanc)
-Djava.rmi.server.useCodebaseOnly=true protocol ::= ftp | http |file

Conclusion intermédiaire

- **1.5, mise en œuvre facilitée,** par un proxy
  - **JavaRmiRuntime au lieu de l 'étape rmic**
  - le proxy est généré soit **en héritant** de la classe **UnicastRemoteObject**
  -
```
public class GroupeDeDsicussion
  extends UnicastRemoteObject implements Remote{ ...
```
  - soit en « exportant dynamiquement » le service

```
IndividuImpl lambda = new IndividuImpl(args[0]);
Remote stub = UnicastRemoteObject.exportObject(lambda, 0);
```

- Recherce du service, soit en utilisant la classe java.rmi.Naming
  - soit directement

```
Registry registry = LocateRegistry.getRegistry();
registry.rebind(args[0], stub);
```

Conclusion

- **RMI: Remote Method Innovation ?**
  - **passage de paramètres**
  - **transmission de code**
  - **Bail et ramasse miettes distribué**
- **RMI 1.2, Activatable, autre support**
- **J2SE >=1.5 : encore plus de transparence**
- **Suite logique ?**
  - **JINI**
    - **« Plug and work »**

Annexe1: interrogation de « rmiregistry »

```
import java.rmi.Naming;
import java.rmi.Remote;
import java.rmi.RMISecurityManager;

public class ListRegistry{

  public static void main(String[] args) throws Exception{
    System.setSecurityManager(new RMISecurityManager());

    try{
      String[] list = Naming.list(args[0]);
      for(int i =0; i<list.length;i++){

        Remote remote = (Remote) Naming.lookup(list[i]);
        System.out.println(list[i] + ", remote: " + remote);
      }
    }catch(Exception e){e.printStackTrace(); throw e;
    }
  }
}
```

Annexe2: « bootstrap » du client

http://www.aw.com/cseng/titles/0-20170043-3

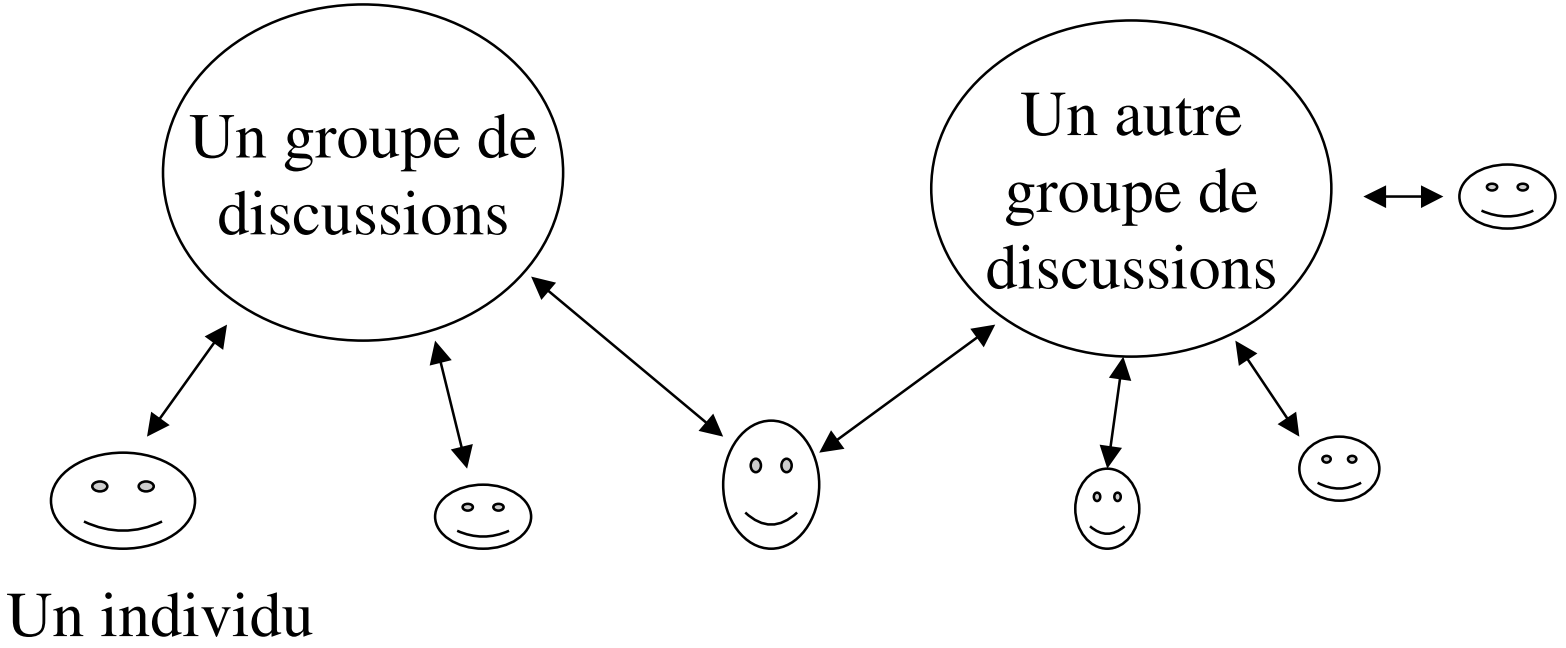
```
public interface Bootstrap extends Remote{
  Runnable getClient() throws RemoteException;
}


public class RMIClientBootstrap{
  private final static String server="rmi://localhost/boot";

  public static void main(String[] args)throws Exception{
    System.setSecurityManager(new RMISecurityManager());
    Bootstrap bootstrp = (Bootstrap)Naming.lookup(server); //
   (args[0])
    Runnable client = bootstrap.getClient();
    Thread t = new Thread(client); // ou client.run();
    t.start();
   }
}
```

téléchargement et exécution … (mobilité du code)

Annexe3: un Chat

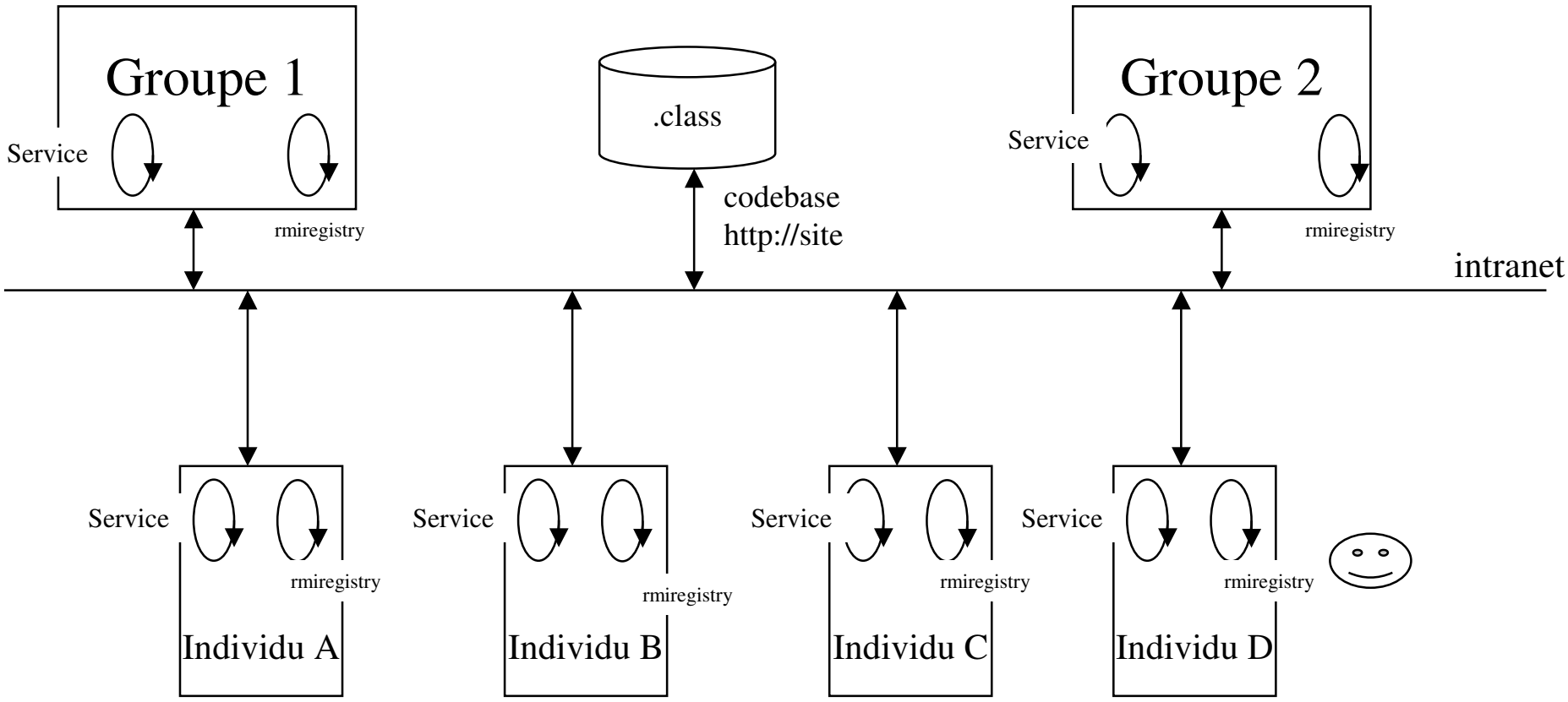


- **Mise en œuvre d'un « Chat »**
  - **Des groupes**
  - **Des individus**

Architecture retenue

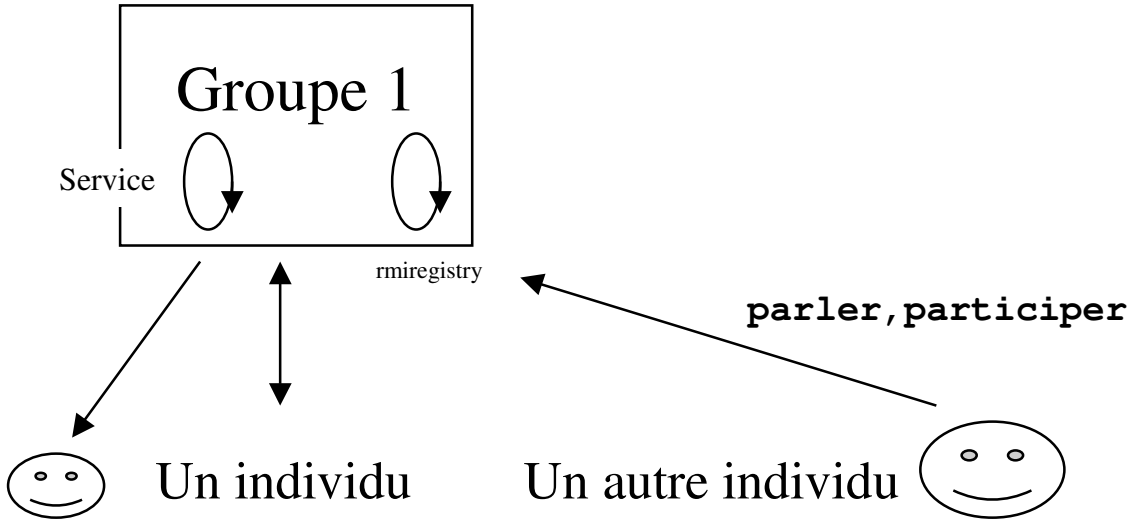
- **Serveur Web pour le codebase**
- **Un serveur rmiregistry par machine**
  - contrainte de sécurité
- **Un serveur par groupe,**
  - Enregistrements des individus comme participant
  - Diffusion des messages aux participants
- **Un serveur par individu**
  - Enregistrement auprès d'un ou de plusieurs groupes
  - Envoi d'un message au groupe

Architecture RMI



- rmiregistry, sur chaque machine; un serveur rmi par groupe, et par individu

interface GroupeDeDiscussion

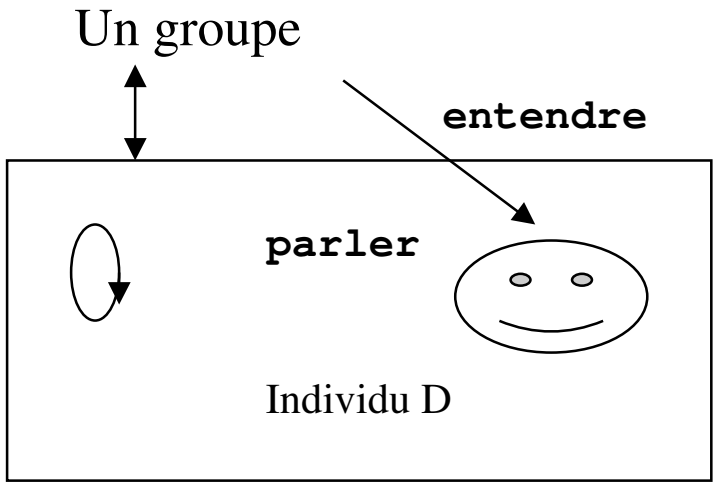



```
public interface GroupeDeDiscussion extends Remote{

 public void participer(Individu individu) throws RemoteException;
 public void sortir(Individu individu) throws RemoteException;

 public void saluer(Individu individu) throws RemoteException;
 public void parler(Individu individu, String phrase) throws RemoteException;
 public void chuchoter(Individu source,
                       String   phrase,
                       Individu destinataire) throws RemoteException;
 public Set<Individu> listeDesParticipants() throws RemoteException;
}
```

interface Individu



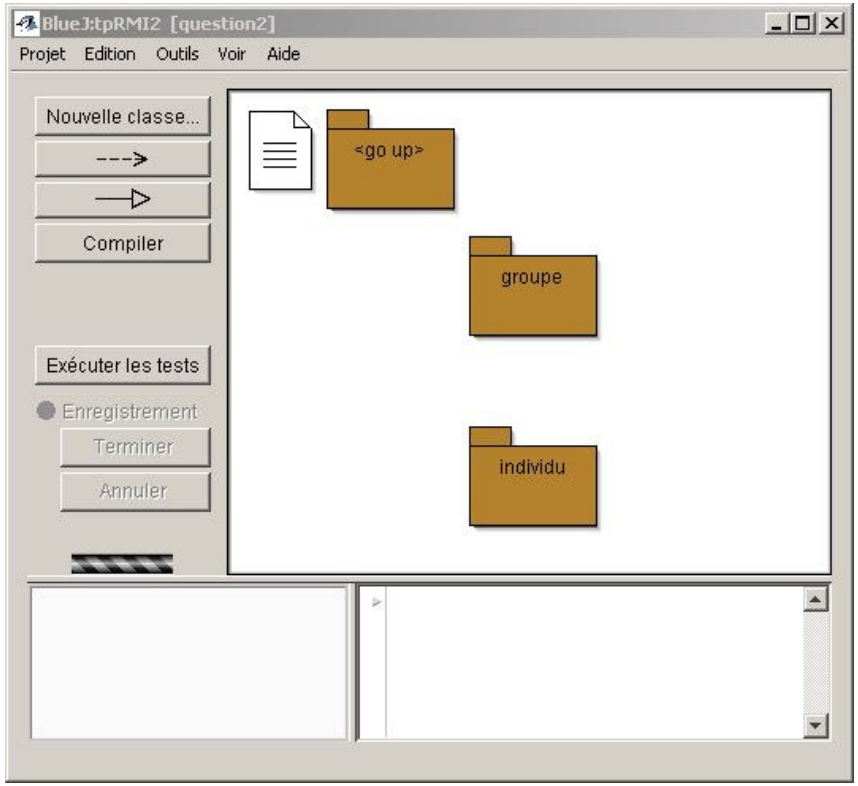

```
public interface Individu extends Remote, Serializable{

  public String nom() throws RemoteException;


  public void entendre(String phrase) throws RemoteException;

}
```

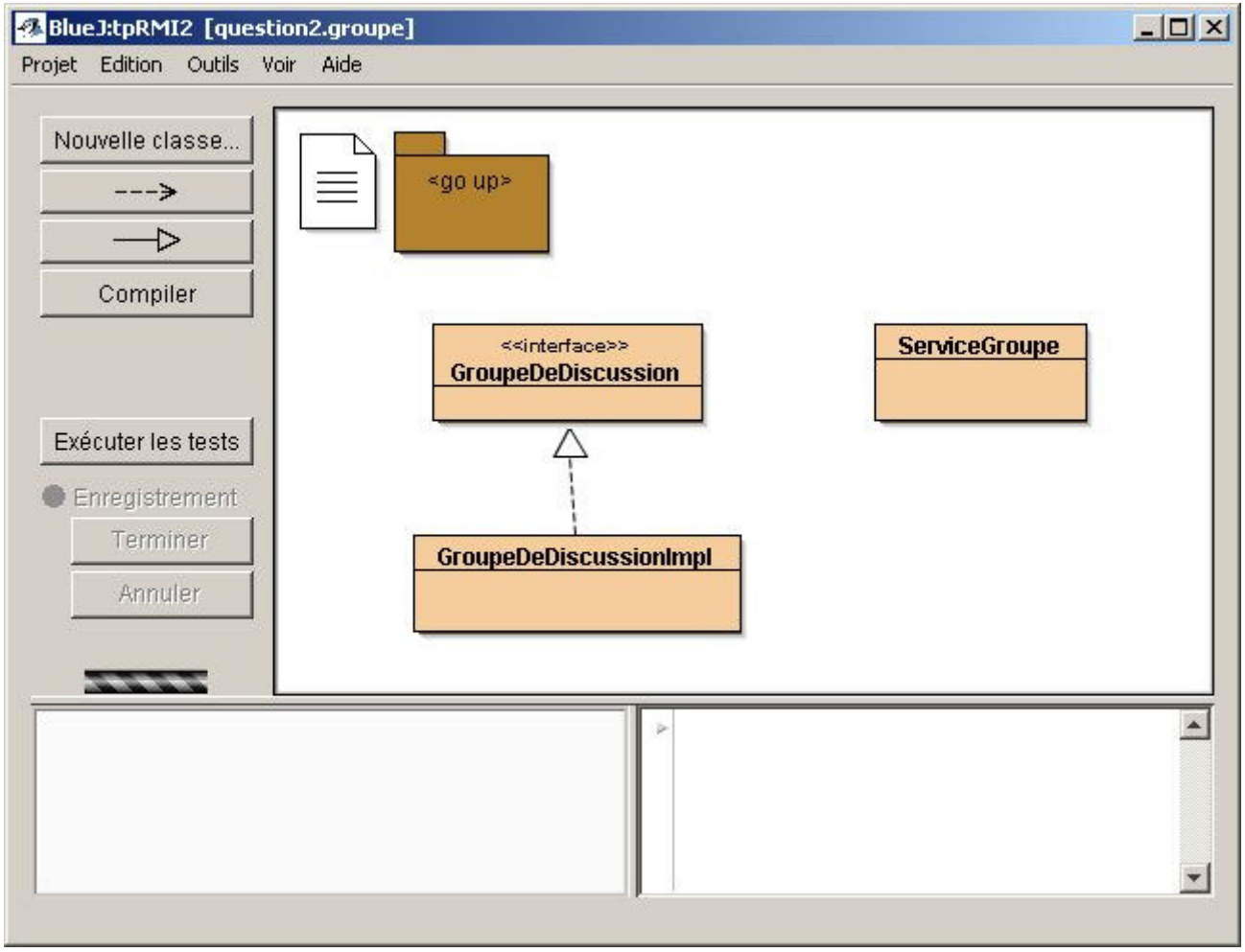
En Java : 2 paquetages



- groupe
  - interface GroupeDeDiscussion
  - classe GroupeDeDiscussionImpl
  - classe ServiceGroupe

individu
- interface Individu
- classe IndividuImpl
- classe ServiceIndividu

le paquetage groupe



Classe GroupeDeDiscussionImpl, un extrait

```
public class GroupeDeDiscussionImpl
  extends UnicastRemoteObject
       implements GroupeDeDiscussion{
         private Set<Individu> participants;
         private String        nomDuGroupe;
```

*Un individu **participe** à ce groupe de discussion*

```
public void participer(Individu individu) throws RemoteException{
     participants.add(individu);
   }
```

*Un individu **parle** au groupe*

```
public void parler(Individu individu, String phrase) throws RemoteException{
    for( Individu i : participants){
     try{
        if(!i.equals(individu)) // il demande aux autres de l'entendre
          i.entendre(nomDuGroupe + "_" + individu.nom() + " << " + phrase + " >>");
      }catch(RemoteException e){
        sortir(i); // parti ?
      }
    }
  }
```

Classe ServiceGroupe

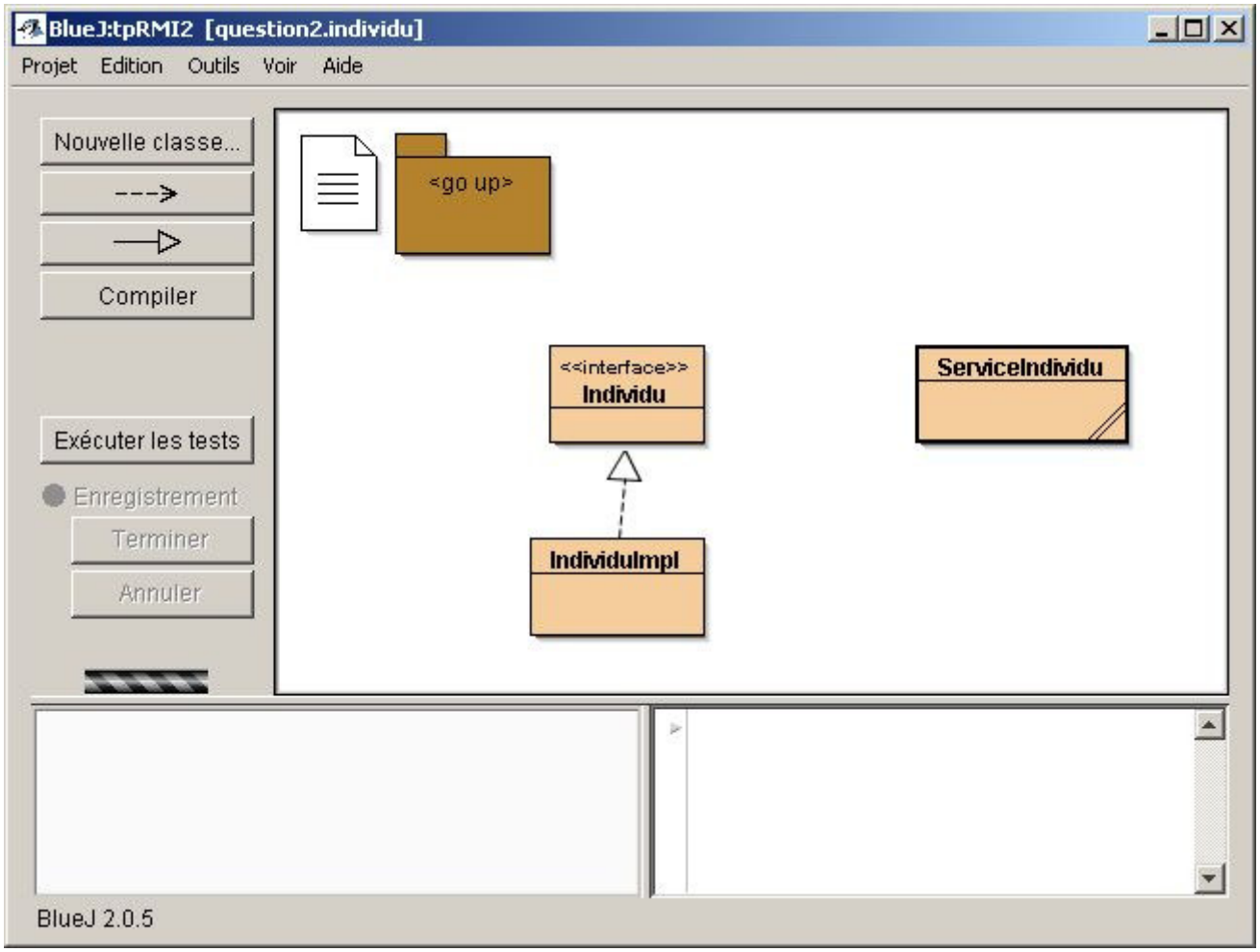
```
public class ServiceGroupe{

  public static void main(String[] args) throws Exception{
    if(args.length>0)
    try{
        Naming.rebind(args[0], new GroupeDeDiscussionImpl(args[0]));
        System.out.println("Le groupe installé en " +
                           InetAddress.getLocalHost().getHostAddress() +
                           "/" + args[0] + " est créé ...");


      }catch(Exception e){
        e.printStackTrace();
      }
    else
      System.out.println("usage >java -Djava.security.policy=... "  +
                                    "-Djava.rmi.server.codebase=...  " +
                                      "ServiceGroupe un_nom");


  }
}
```

le paquetage Individu



Classe IndividuImpl, un extrait

```
public class IndividuImpl implements Individu, Runnable{
```

*Un individu **rejoint** un groupe*

```
  public void rejoindre(GroupeDeDiscussion groupe){
    try{
        groupe.participer((Individu)Naming.lookup("rmi://" +
                InetAddress.getLocalHost().getHostAddress() + "/" + nom));
        this.groupe = groupe;
        this.groupe.saluer(this);
        ...
      }catch(Exception e){
        e.printStackTrace();
      }
    }
```

*Un individu **entend** ...*

```
    public void entendre(String phrase)  throws RemoteException{
      System.out.println(phrase);
    }
```

Classe IndividuImpl, suite de l 'extrait

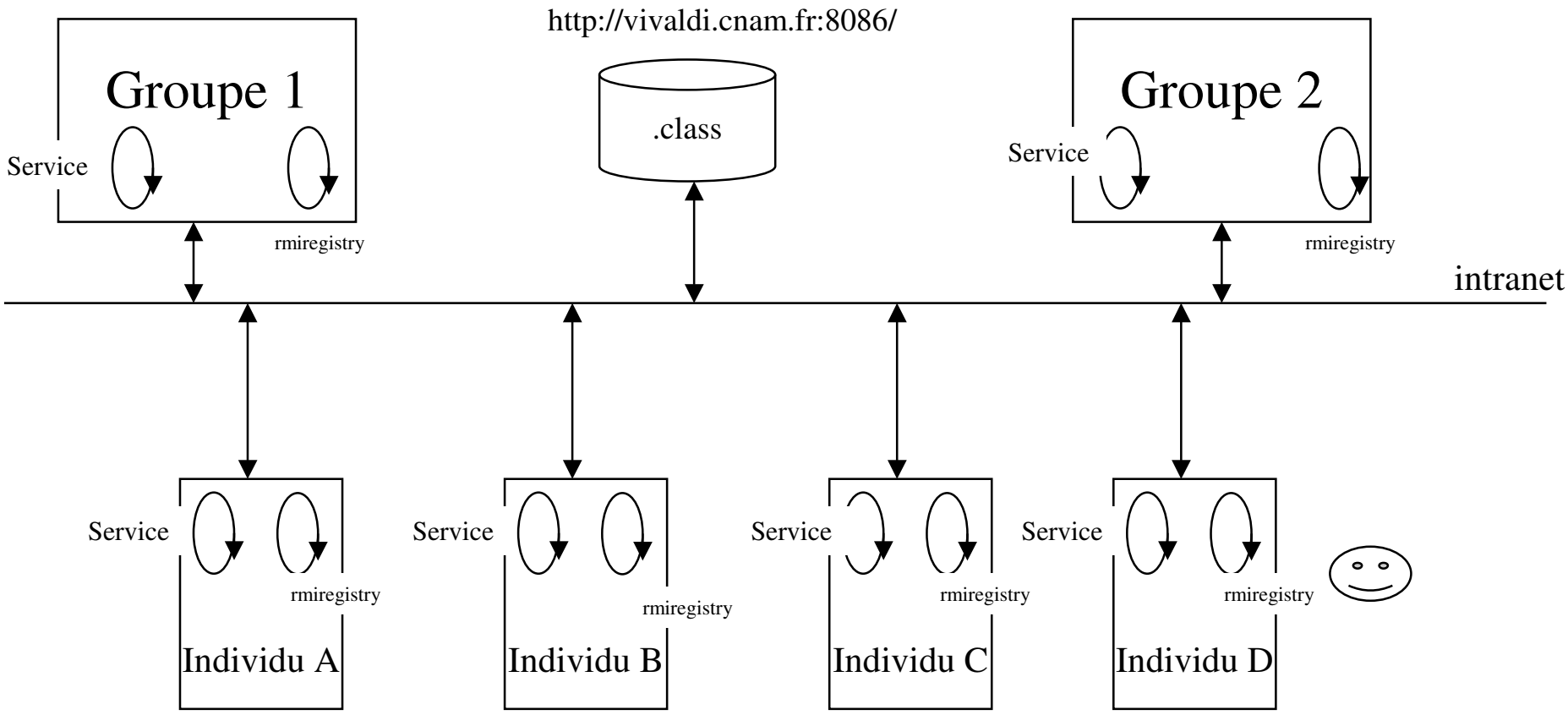
*Un individu **parle** ...*

```
public void parler(String phrase)  throws RemoteException{
   this.groupe.parler(this, phrase);
 }
```

*Un individu **chuchote***

```
public void chuchoter(String destinataire, String phrase)  throws
RemoteException{
  Set<Individu> liste = this.groupe.listeDesParticipants();
  for(Individu i : liste)
    if(i.nom().equals(destinataire)){
      this.groupe.chuchoter(this, phrase, i);
      break;
    }
}
```

Architecture RMI Chat au Cnam



- Un essai en intranet, reproductible chez vous ….

les commandes

- **le répertoire**
  - http://jfod.cnam.fr/rmi/
  - start rmiregistry (sur chaque machine groupe comme individu)
  - un groupe
  - **java -cp chat.jar -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://jfod.cnam.fr/rmi/chat.jar question2.groupe.ServiceGroupe nfp120**
    - *Le groupe installé en 163.173.228.59/nfp120 est créé …*
  - deux individus
  - **java -cp chat.jar -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://jfod.cnam.fr/rmi/chat.jar question2.individu.ServiceIndividu paul 163.173.228.59/nfp120**
  - **java -cp chat.jar -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://jfod.cnam.fr/rmi/chat.jar question2.individu.ServiceIndividu jean 163.173.228.59/nfp120**

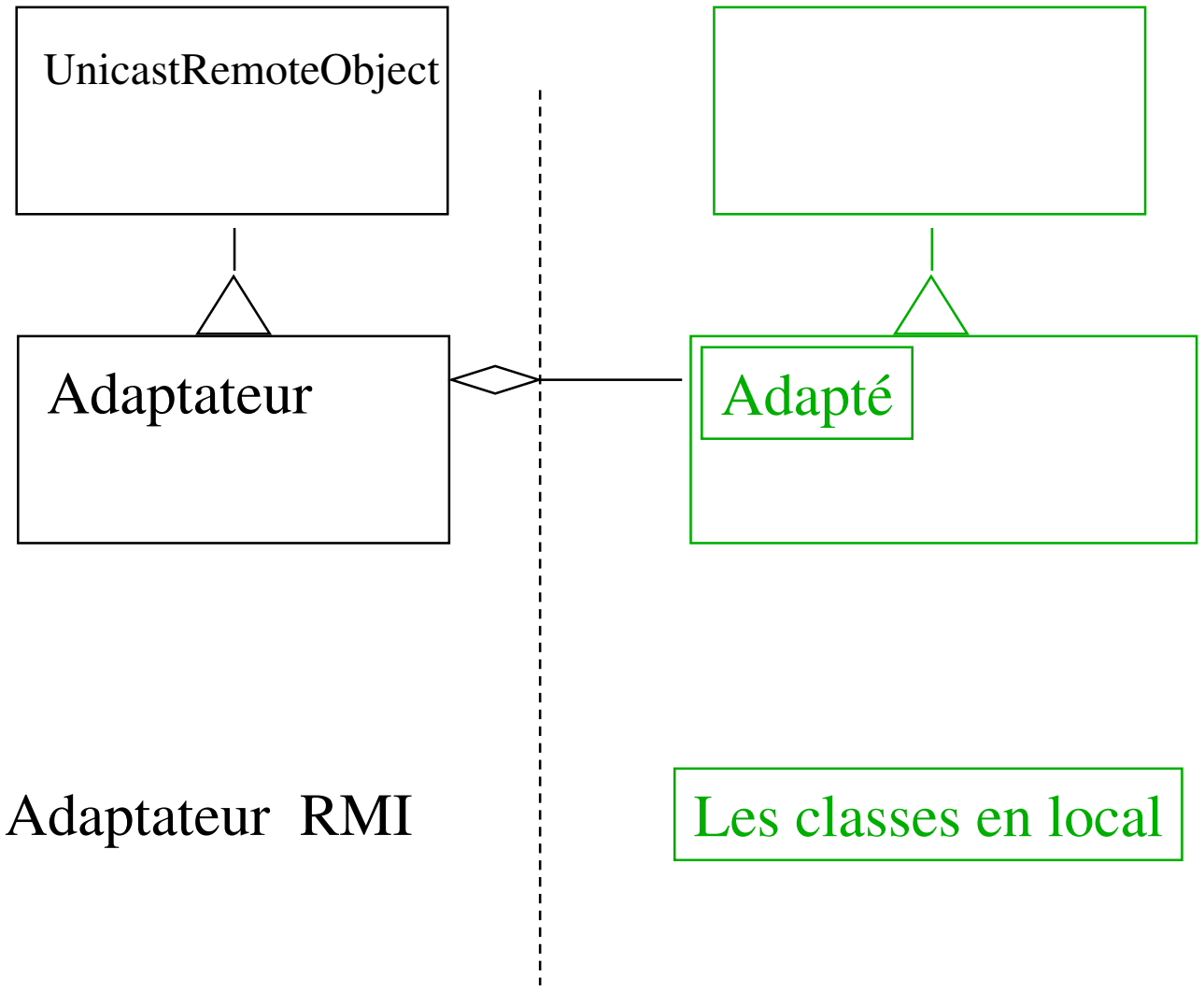
Annexe 4 : le patron Adapatateur à la rescousse

Développement : une ligne de conduite

- **Peut-on effectuer un développement en local et mettre en œuvre simplement le mécanisme RMI ?**
- **Comment assurer un faible couplage des classes assurant le RMI et les autres classes ?**
- *Peut-on oublier RMI ?*
- **Une solution : usage du Pattern Adapter**
  - **Un adaptateur RMI/version locale**

`http://www.transvirtual.com/users/peter/patterns/overview.html`
`http://www.eli.sdsu.edu/courses/spring98/cs635/notes/index.html`

En 2 étapes


UnicastRemoteObject
Adaptateur
Adapté
Adaptateur RMI
Les classes en local

Exemple le serveur de tâches


```
public class Client{
  ServeurDeTaches s; ...
  Horloge uneHorloge = ...

  s.executer(uneHorloge);

}
```

RMI

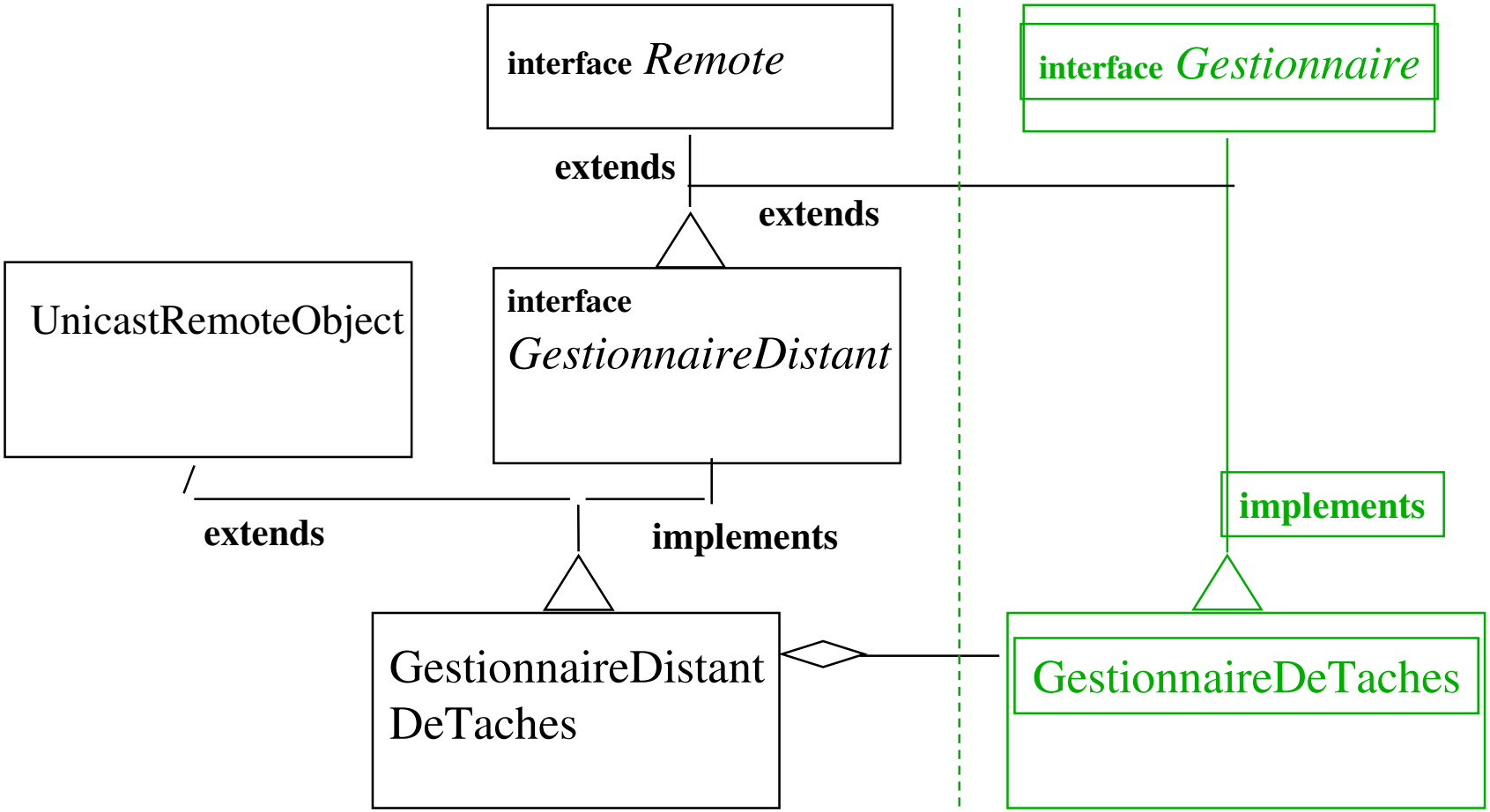
```
public class ServeurDeTaches{

  void executer(Runnable r){
    new Thread(r).start();

  }

}
```

Le serveur de tâches : diagrammes UML



Adaptateur RMI

Les classes en local : l'adapté

local : GestionnaireDeTaches (Adapté)

```
public interface Gestionnaire{

  public void executer( Runnable r) throws Exception;

  public java.util.Vector liste() throws Exception;
}
```

Légère contrainte syntaxique :

Chaque méthode possède la clause throws Exception

local : GestionnaireDeTaches (Adapté)

```
public class GestionnaireDeTaches implements Gestionnaire{
   private ThreadGroup table;
   private Vector      liste;

   public GestionnaireDeTaches(String nom){
     table = new ThreadGroup(nom);
     liste = new Vector();
   }

   public void executer( Runnable r) throws Exception{
     Thread t = new Thread(table,r); t.start();
     liste.addElement(r);
   }
   public java.util.Vector liste() throws Exception {
     return liste;
   }
 }
```

local : Test de l 'adapté

```
public class TestGestionnaireDeTaches{

  public static void main(String [] args) throws Exception{

    Gestionnaire mon1 = new GestionnaireDeTaches("mon1");

    mon1.executer(new Horloge());
    mon1.executer(new Horloge());

    System.out.println("mon1 : " + mon1.liste());

    mon1.executer(new Horloge());
    System.out.println("mon1 : " + mon1.liste());

  }
}
```

Comme toujours :

Effectuer avec soins « tous » les tests du futur adapté

GestionnaireDistant.java (Adaptateur)

- **Développement de l 'adaptateur**

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface GestionnaireDistant extends Remote,Gestionnaire{

  public static final String nomDuService = "leServeurDeTaches";

  public void executer( Runnable r) throws RemoteException,Exception;

  public java.util.Vector liste() throws RemoteException,Exception;

}
```

La Cohérence des 2 interfaces est assurée par :

GestionnaireDistant extends **Remote,Gestionnaire**

GestionnaireDistantDeTaches.java (Adaptateur)

```
public class GestionnaireDistantDeTaches extends UnicastRemoteObject
                                         implements GestionnaireDistant{

  private GestionnaireDeTaches gestionnaire;  // l'adapté


  public GestionnaireDistantDeTaches(GestionnaireDeTaches gestionnaire)
                                                 throws RemoteException{
     this.gestionnaire = gestionnaire;
  }

  public void executer( Runnable r) throws RemoteException,Exception{
     gestionnaire.executer(r);
  }

  public Vector liste()  throws RemoteException,Exception{
     return gestionnaire.liste();
  }
}
```

Systématique :
Chaque méthode se contente d'exécuter l'adapté

ServeurDeTaches.java

```
import java.rmi.*;
public class ServeurDeTaches{

  public static void main(String [] args) throws Exception{
    System.setSecurityManager( new RMISecurityManager());


    try{
      GestionnaireDistant mon1 = new GestionnaireDistantDeTaches(
                                        new GestionnaireDeTaches("mon1"));



      Naming.rebind( GestionnaireDistant.nomDuService, mon1);
      System.out.println("le serveur: " +
                         GestionnaireDistant.nomDuService +
                         " a demarre ");

     }catch(Exception e){throw e;}
   }
}
```

Le Client : TacheCliente.java

```
import java.rmi.*;
public class TacheCliente{

   public static void main(String [] args) throws Exception{
     String machine = "lmi27";
     if (args.length == 1) machine = args[0];
     System.setSecurityManager( new RMISecurityManager());
     GestionnaireDistant mon1 = null;

     String nom = "rmi://"+ machine +"/"+ GestionnaireDistant.nomDuService;

    try{
       mon1 = (GestionnaireDistant)Naming.lookup(nom);
    }catch(Exception e){throw e;}


   try{
     mon1.executer(new Horloge());
     mon1.executer(new Horloge()); System.out.println(mon1.liste());

     mon1.executer(new Horloge()); System.out.println(mon1.liste());
    }catch(Exception e){throw e;}}}
```

Comment ? : les commandes, le serveur

- **//lmi86/d:/rmi/ServeurDeTaches/Serveur/**

  - > set CLASSPATH= ... (le fichier _stub doit être inaccessible par rmiregsitry)
  - > start **rmiregistry**

  - > **java** -Djava.rmi.server.codebase=file:/D:/rmi/ServeurDeTaches/Serveur/
  - -Djava.security.policy=java.policy ServeurDeTaches

Comment ? : les commandes, le client

- **//lmi27/d:/rmi/ServeurDeTaches/Client/**
  - **> java -Djava.rmi.server.codebase=file:/D:/rmi/ServeurDeTaches/Client/**
  - **-Djava.security.policy=java.policy TacheCliente**

OU à l'aide d'un serveur http

- **//lmi27/d:/rmi/ServeurDeTaches/Client/**
  - **> start java SimpleHttpd 8088**
  - **> java -Djava.rmi.server.codebase=http://lmi27:8088/D:/rmi/ServeurDeTaches/Client/**
  - **-Djava.security.policy=java.policy TacheCliente**

- **codebase = {http://machine/repertoire/ | ftp:/repertoire/ | file:/}**