Remote **M**ethod **I**nvocation

1999, Cnam Paris
jean-michel Douin, douin@cnam.fr

Notes de cours consacrées à RMI
Version du 18 Mars 2002

http://lmi92.cnam.fr:8080/tp_cdi/douin/Java_RMI.pdf

---

http://lmi92.cnam.fr:8080/tp_cdi/TpRmi/TpRmi.html

Principale bibliographie

Java Distributed Computing. Jim Farley. The Java series, O'Reilly 1998
`http://www.eli.sdsu.edu/courses/spring99/cs696/notes/index.html`

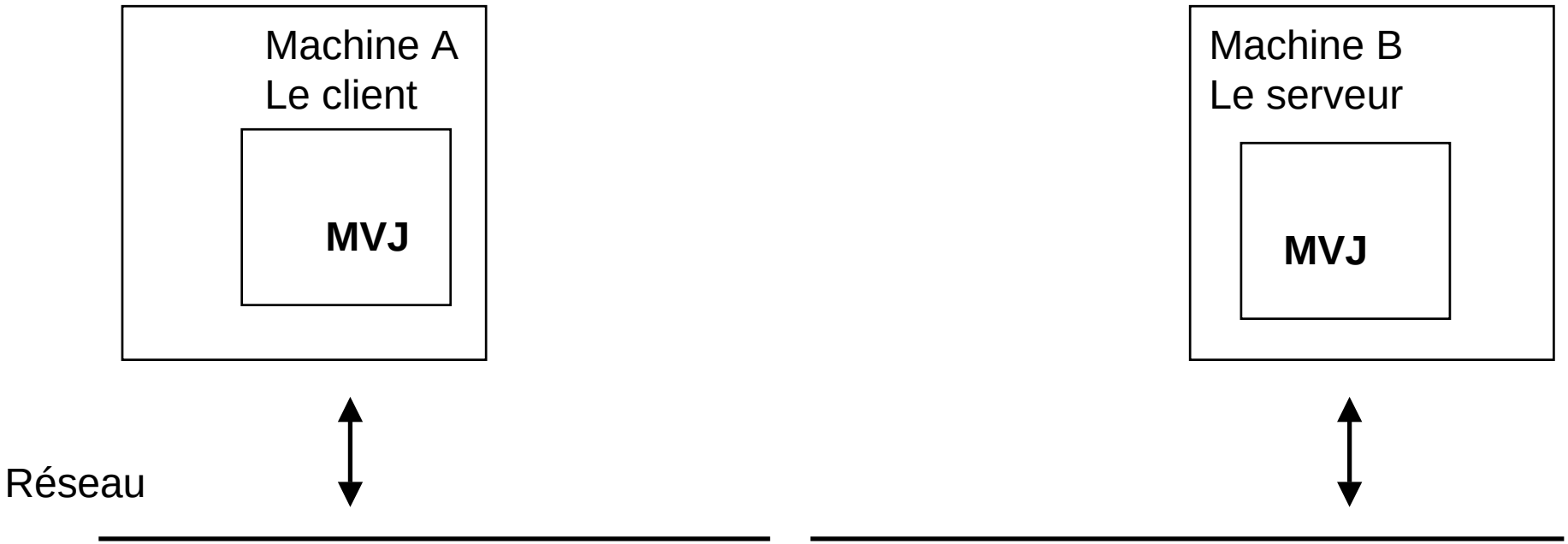
- RMI
  `http://java.sun.com/products/jdk/1.2/docs/guide/rmi/index.html`
  `http://java.sun.com/j2se/1.3/docs/guide/rmi/index.html`
  `http://java.sun.com/products/jdk/1.2/docs/guide/security/PolicyFiles.html`
  `http://java.sun.com/products/jdk/1.2/docs/guide/security/permissions.html`
  `http://java.sun.com/docs/books/tutorial/rmi/index.html`
- JNI
  `http://java.sun.com/docs/books/tutorial/native1.1/index.html`
  `http://lmi92.cnam.fr:8080/tp_cdi/douin/java_JNI.pdf`
- JDBC tutorial
  `http://java.sun.com/docs/books/tutorial/jdbc/index.html`

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

Sommaire

- **Episode 1**
  - Exemple : Un client et un serveur de méthodes
  - téléchargement : **rmiregistry**
  - Le serveur et **rmic** (le Pattern Proxy)
- **Episode 2**
  - Persistance
  - Passage de paramètres et retour de fonctions
  - Un serveur http : téléchargement de classes
  - Méthode : Usage du Pattern Adapter
  - Exemple : Un client et un serveur de tâches
- **Episode 3**
  - JNI + RMI : entrées-sorties distantes
  - JDBC + RMI : accès à une base de données
- **Episode 1.2**
  - JDK1.2 **rmid** et la classe java.rmi.activation.Activatable

Episode 1 : Objectifs




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

  s.methodeLointaine();

}
```

RMI

```
public class ServeurDeMethodes{

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

}
```

==> <*Quoi,* .

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

Comment>$_1$

- La "machine Serveur" est identifiée par une adresse IP ou un nom octroyé par l'administrateur du DNS
- La "classe Serveur" est associée à un nom répertorié(utilitaire jdk **rmiregistry**)
- La communication est prise en charge par une procuration fournie par le serveur au client et une interface sur le serveur(utilitaire jdk **rmic**)
- La classe `java.rmi.RMISecurityManager` autorise les accès distants et précise les contraintes d ’accès aux fichiers (en général un fichier ".**policy**")

- `import java.rmi.*;   /** en standard */`
- `import java.rmi.activation.*;`

Développement en Java, principes

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

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

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

Développement en Java : mode d'emploi

- **L ' interface est commune** aux clients et au serveur et recense les méthodes distantes, elle hérite de ***java.rmi.Remote*** et chaque méthode possède la clause ***throws java.rmi.RemoteException***

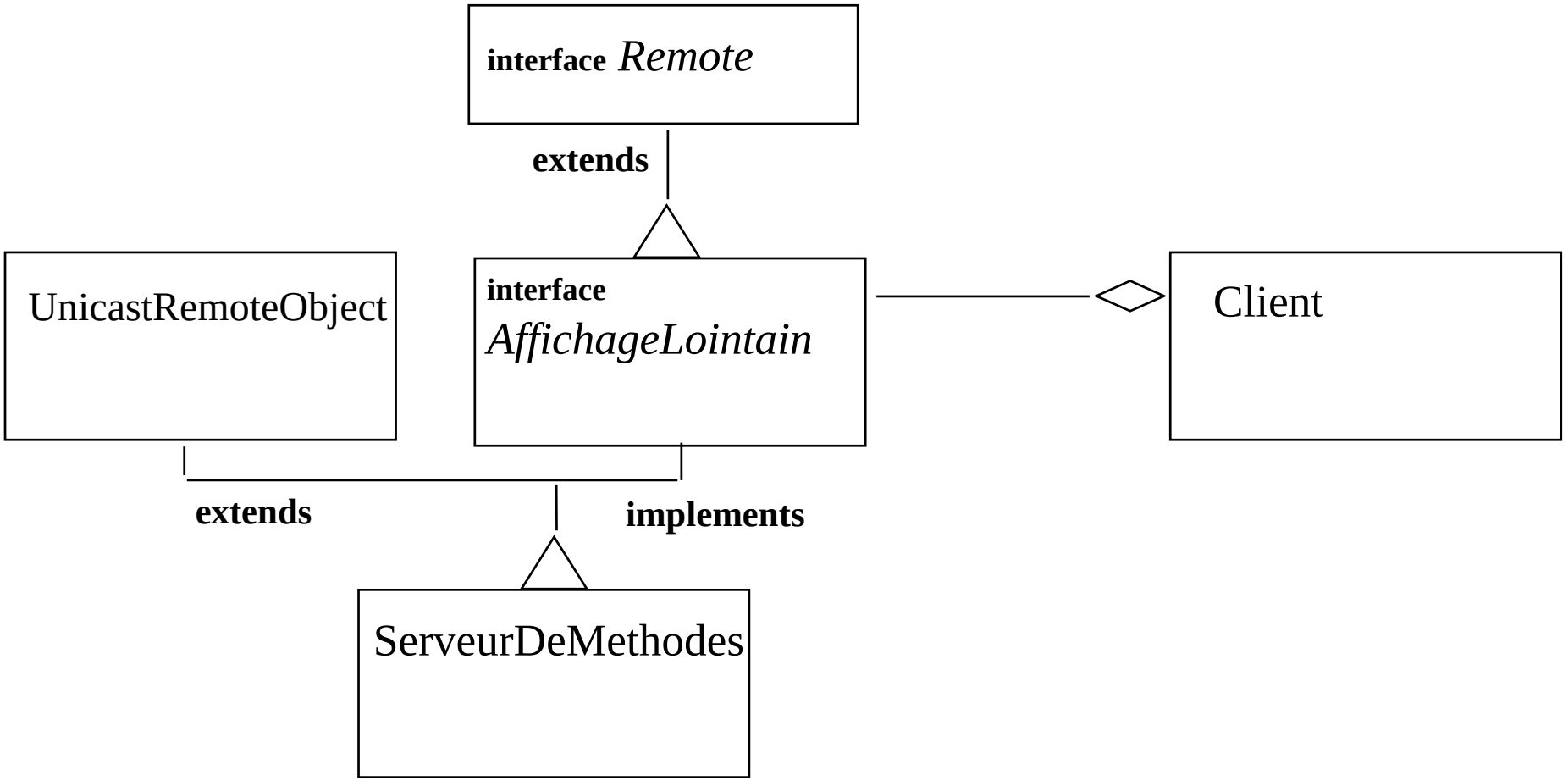
- **La classe Serveur** hérite de ***java.rmi.server.UnicastRemoteObject*** et implémente les méthodes de l 'interface commune, sans oublier le constructeur qui possède également la clause *throws*. Le serveur propose ses services à l 'aide de la méthode ***java.rmi.Naming.rebind***

- **Les classes Clientes** utilisent une instance de la classe Serveur obtenue par l 'appel de ***java.rmi.Naming.lookup.***

- *Et c'est tout !!! enfin presque*

- L'épisode 1.2 et le package java.rmi.activation proposeront quelques modifications à ce mode d'emploi

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 lmi93)

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

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

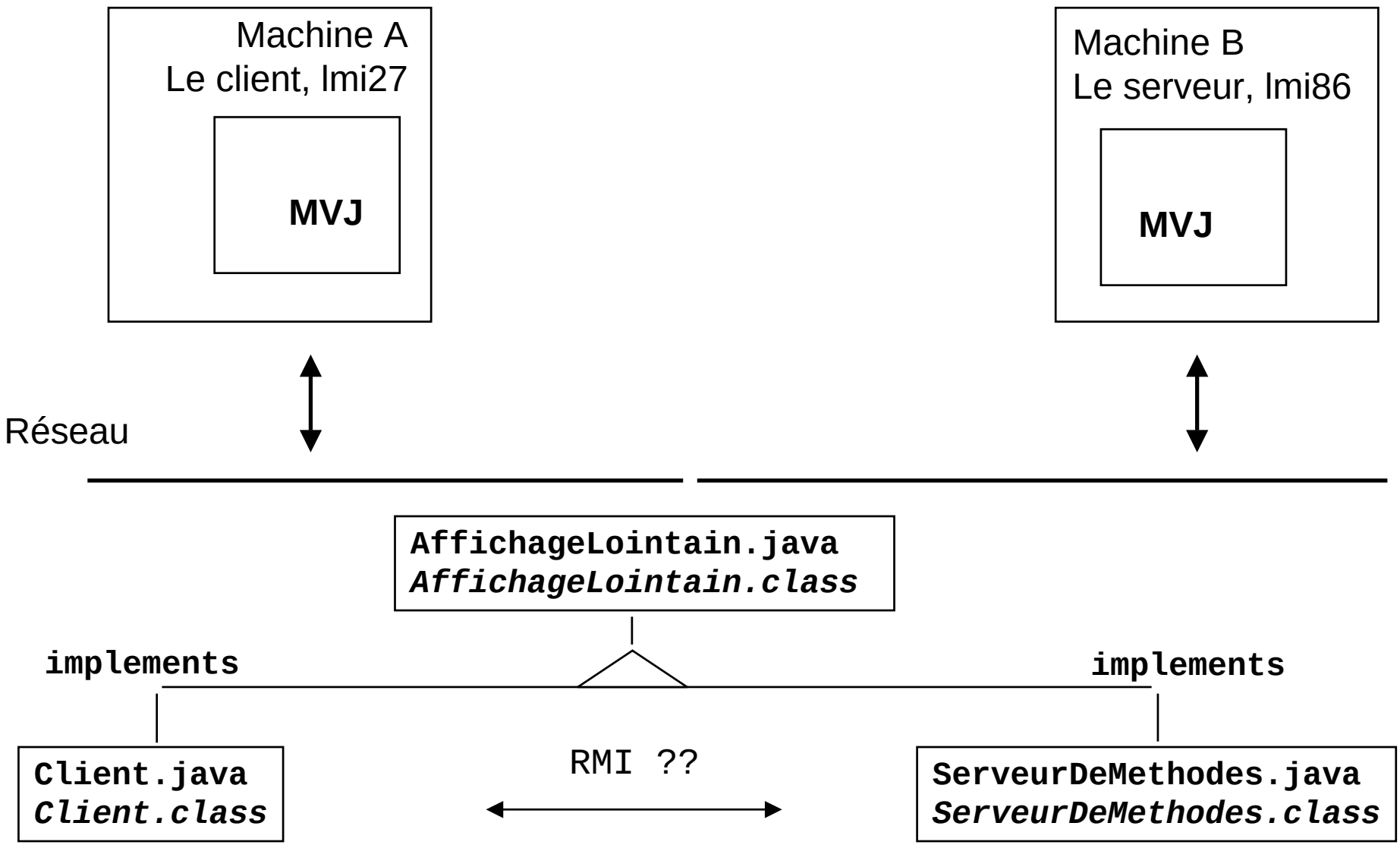
```
import java.rmi.*;
public class Client{

  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

d:/rmi/serveurDeMethodes/serveur/
et d:/rmi/serveurDeMethodes/client/

Comment ? : les commandes, le serveur

- **//lmi86/d:/rmi/serveurDeMethodes/serveur/**

  Compilation du serveur

  > **javac** ServeurDeMethodes.java

  > **rmic** ServeurDeMethodes

  Exécution

  > **start** rmiregistry

  (attention au CLASSPATH, le fichier _stub doit être inaccessible par rmiregistry)

  > **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 java.rmi.server.RMIClassLoader

rmic : **rmi c**ompiler, génération des fichiers _Skel.class et _Stub.class

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) **dos>start rmiregistry**
  2) **dos>start java SimpleHttpd 8080**

  Coté serveur :
  1) **dos> java -Djava.rmi.server.codebase=http://localhost:8080/ ServeurDeMethodes**

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

```
[LMI93/127.0.0.1] -- Request: GET /ServeurDeMethodes_Stub.class HTTP/1.1
[LMI93/127.0.0.1] -- 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 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) dos> java Client

  2) Les traces sur la console du serveur Web

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

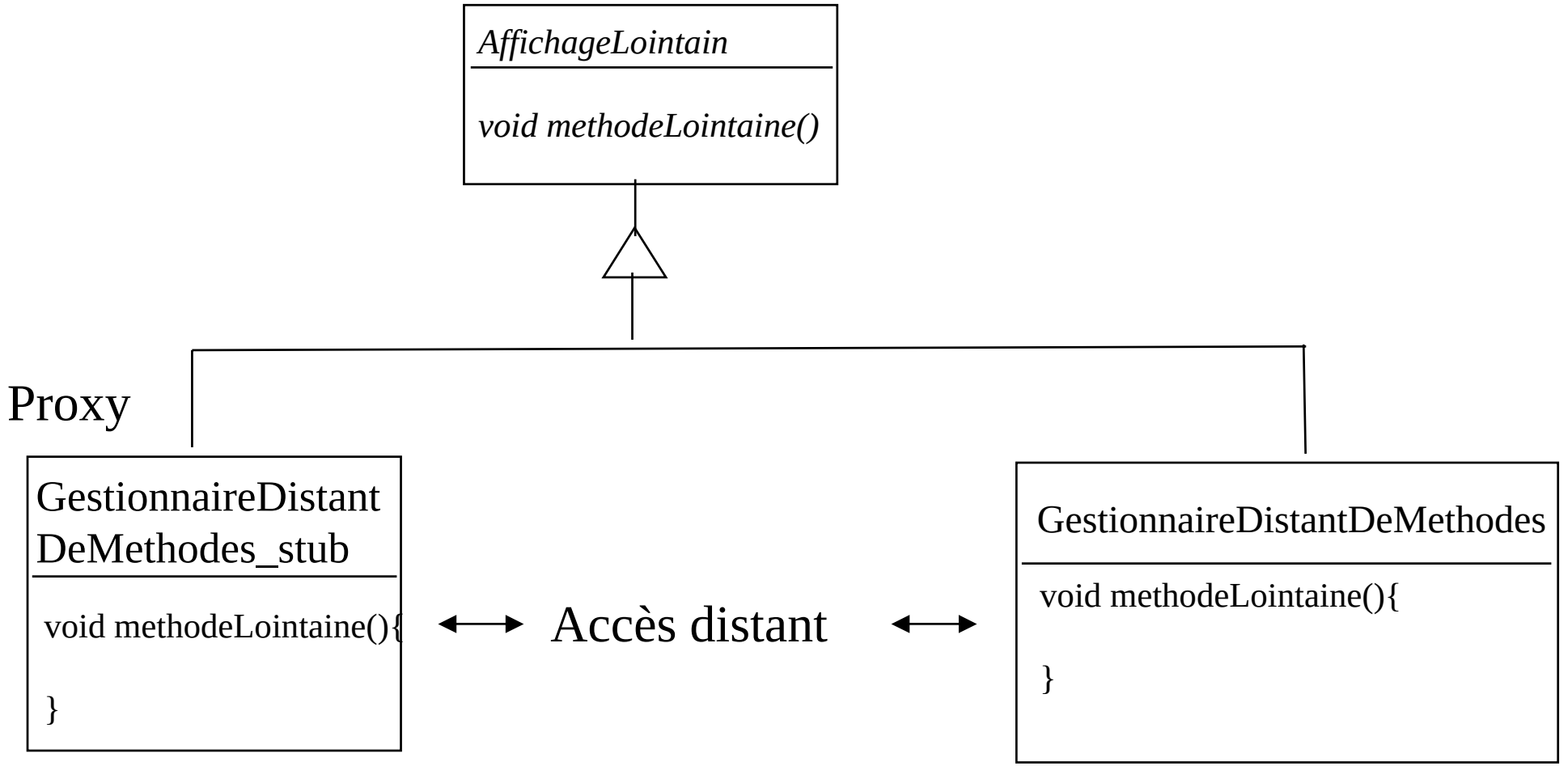
rmic, _skel.class, _stub.class

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)

Le pattern Proxy



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

Episode 2 : Objectifs


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

  s.executer(uneHorloge);

}
```

RMI

```
public class ServeurDeTaches{

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

  }

}
```

==> <*Quoi,* .

- *Comment transmettre les paramètres ?*
- *Comment les résultats de fonction sont-ils retournés aux clients ?*
- *Et si les paramètres utilisent des classes inconnues du serveur ?*

Comment>$_2$

- Aucune incidence sur le source Java excepté que les paramètres transmis doivent être des instances de **java.io.Serializable** (toutes les classes)
- Une copie des paramètres en profondeur est effectuée, pour cela la notion de persistance en Java est utilisée
- En retour de fonction, une copie est également effectuée
- Si les paramètres sont des instances de classes inconnues du serveur, celui-ci les « demande » au client

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

Passage de paramètres : exemple

- La signature de la méthode de l'interface est :

  `void executer(Runnable r){ }`

  ==> *Le serveur implémente cette méthode*

- En Java, toute instance d'une classe implémentant cette interface est compatible.

  Exemple : la classe Horloge développée par le client

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

avec `public class java.lang.Thread implements java.lang.Runnable{ ...}`

Le client :
```
ServeurDeTaches s = Naming.lookup(..)
Horloge uneHorloge = new Horloge();
s.executer( uneHorloge);
```

Sérialisation : principes

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

```
public class Horloge  extends Thread
                      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(uneHorloge);
```

*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);
Horloge uneHorloge = (Horloge)o.readObject();
```

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){}}}}
```

Mais cette classe est inconnue du serveur … Alors ...

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**

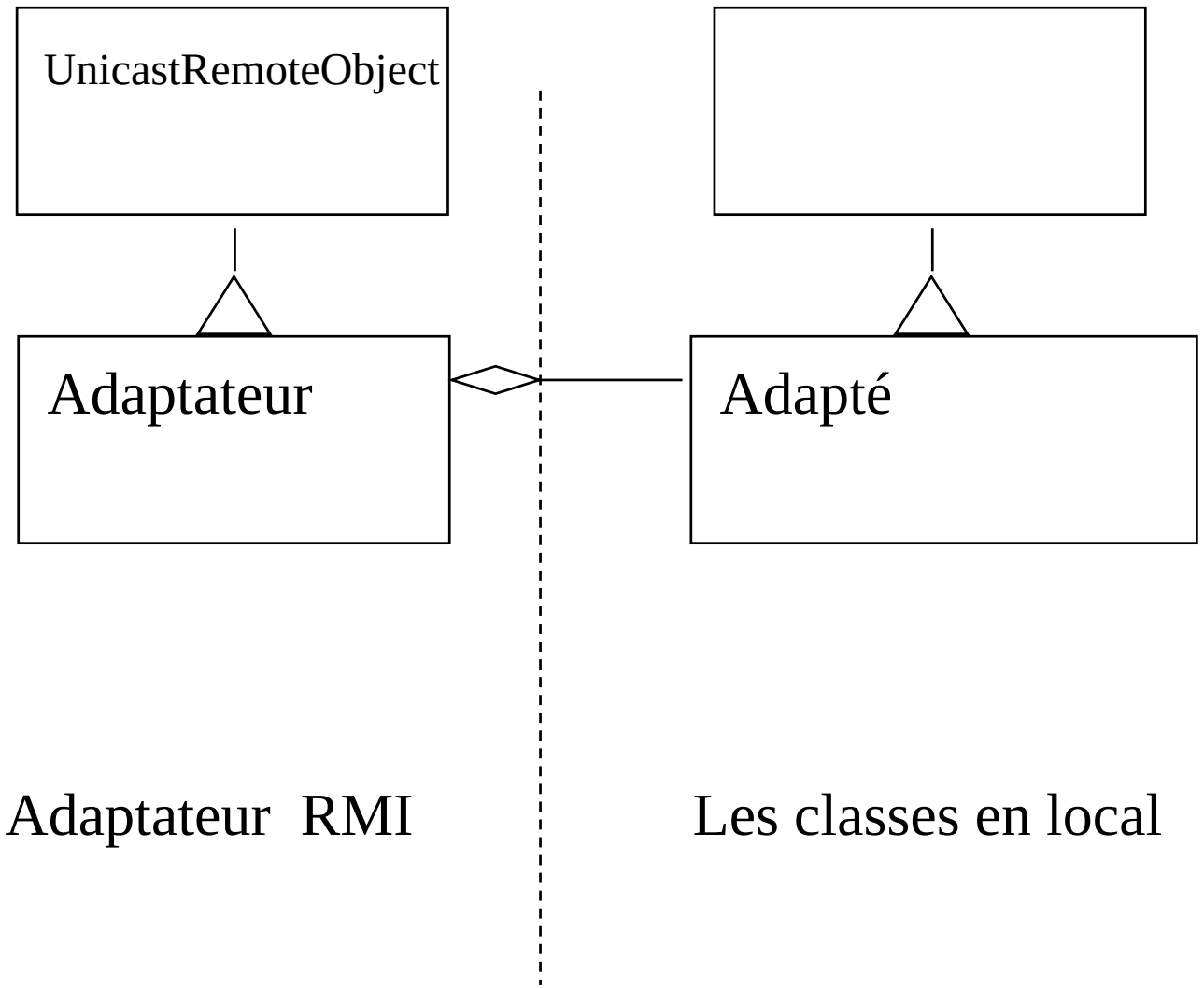
  > java **-Djava.rmi.server.codebase=http://lmi27:8088/D:/rmi/ServeurDeMethodes/Client/**
  -Djava.security.policy=java.policy Client

Développement : une ligne de conduite

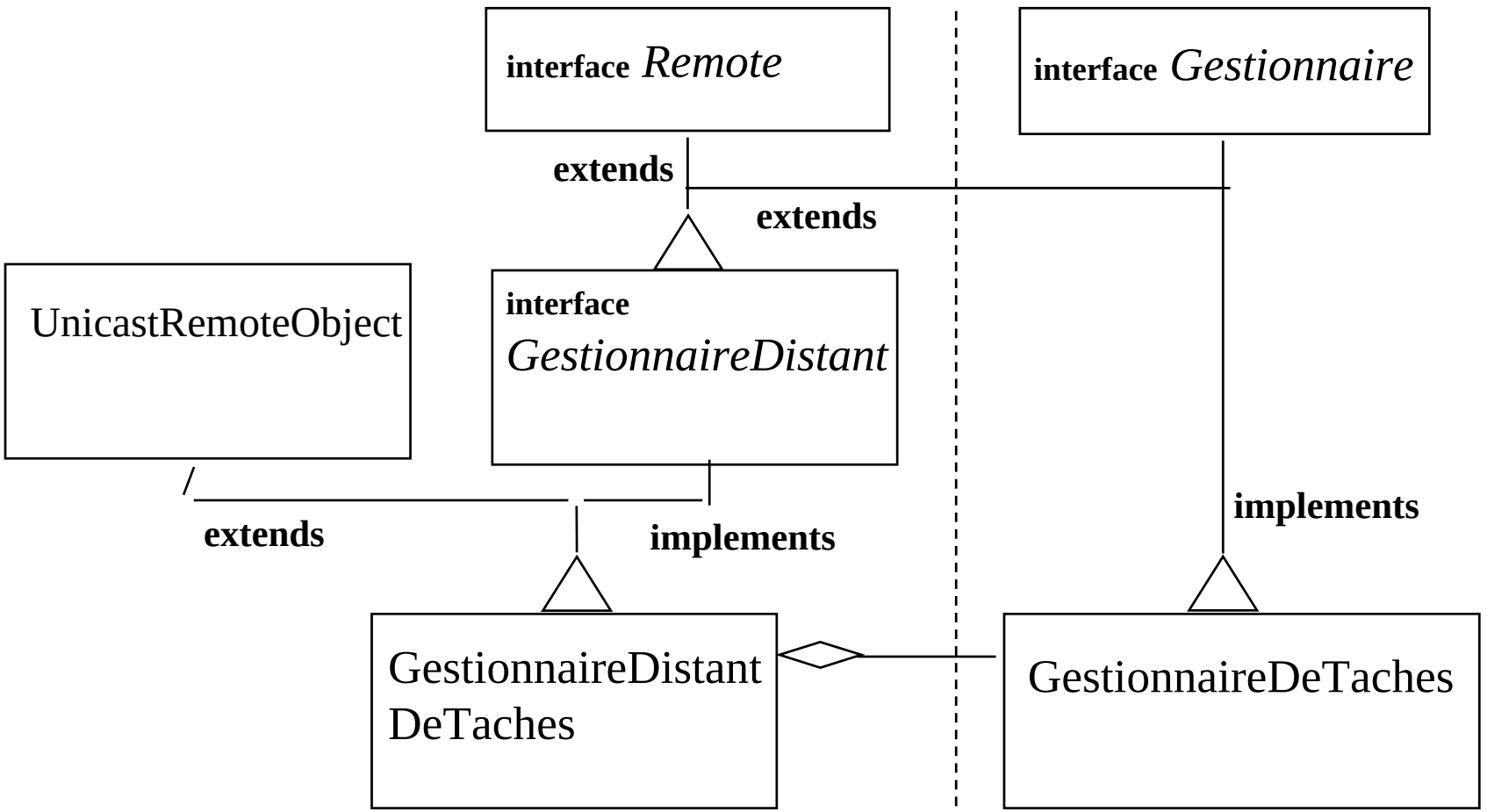
- **Peut-on effectuer un développement en local et mettre en œuvre simplement le mécanisme RMI ?**
- **Comment assurer un faible couplage des classes assurant le RMI et les autres classes ?**
- *Peut-on oublier RMI ?*
- **Une solution : usage du Pattern Adapter**

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

Exemple : Le serveur de tâches



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;
}
```

Contrainte:

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());

  }
}
```

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;

}
```

Cohérence des 2 interfaces 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();
  }
 }
```

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/ }**

Exercice : modification de TacheCliente.java

• Modifier TacheCliente.java en ajoutant de nouvelles taches exécutées à distance, ces taches doivent produire un résultat lu par le client, proposer une solution synchrone et asynchrone. Vérifier les performances par rapport à une version locale.

•*(Envoyez-moi vos réponses ! douin@cnam.fr, elles ajouter…à la prochaine mouture)*

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

   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 Tache(param1));
      mon1.executer(new Tache(param2));
     }catch(Exception e){throw e;}}}}
```

Episode 3

- interface vers d ’autres langages

- **JNI**

  Java Native Interface

  Appel d ’un existant développé en C, C++, assembleur

  Java <--> C

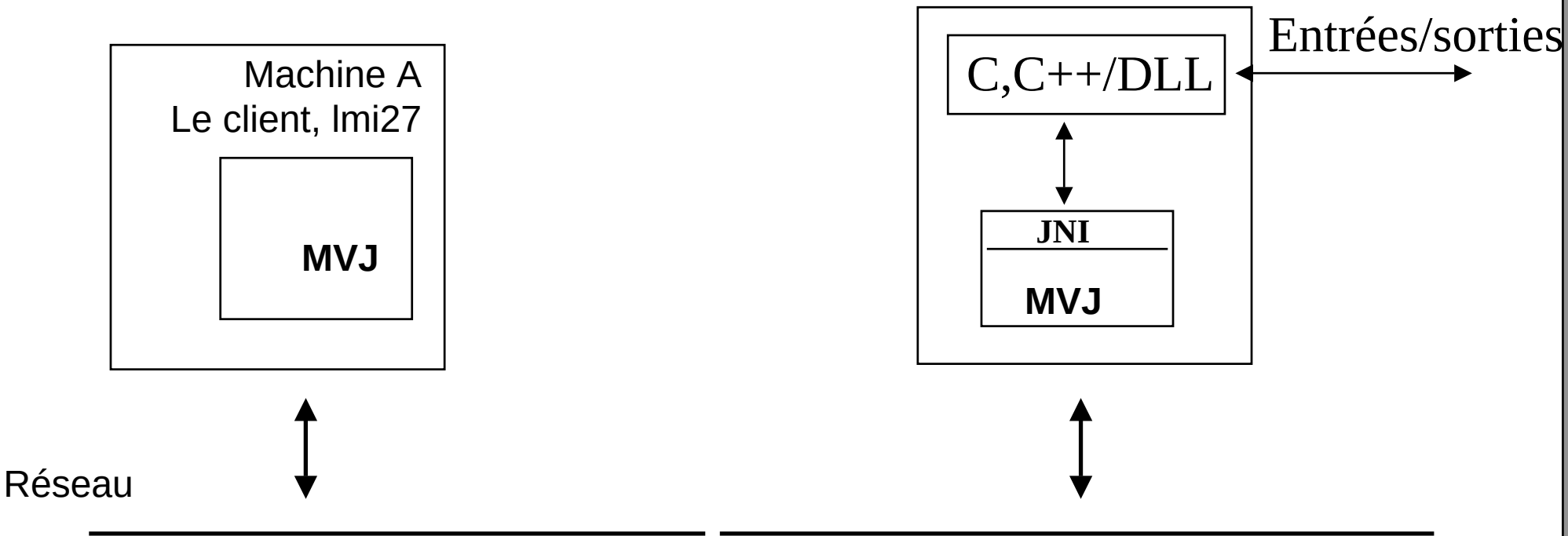
- **JDBC**

  Java Data Base Connection

  Exécution de requêtes SQL

  Java <--> SQL

JNI + RMI



- Applications existantes développées en C, C++, assembleur.
- Accès entrées-sorties distantes,
- ….

Le GestionnaireDeTaches en natif

- ***En exemple : le gestionnaire de taches est partiellement écrit en code natif, (pattern Adapter : l 'adaptateur reste le même seul l'adapté change)***

```
import java.util.Vector;
public class GestionnaireNatifDeTaches implements Gestionnaire{
  private ThreadGroup table;
  private Vector      liste;

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

  public native void executer( Runnable r) throws Exception;

  public native Vector liste() throws Exception;

  static{
    System.loadLibrary("GestionnaireNatifDeTaches");
  }
}
```

GestionnaireNatifDeTaches.h

Commande ***> javac GestionnaireNatifDeTaches.java***

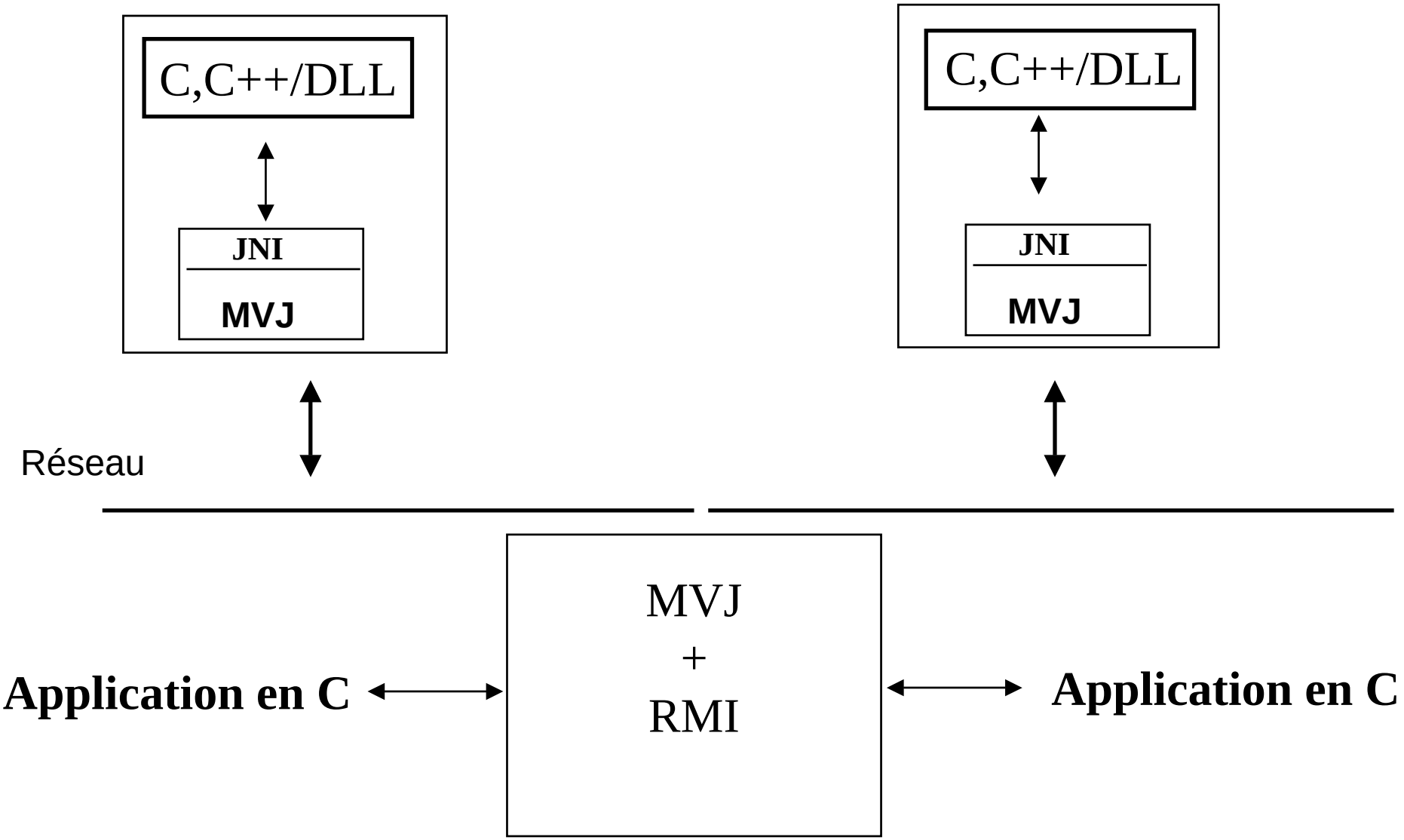
Commande ***> javah -jni GestionnaireNatifDeTaches***

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class GestionnaireNatifDeTaches */
#ifndef _Included_GestionnaireNatifDeTaches
#define _Included_GestionnaireNatifDeTaches

/* Class:     GestionnaireNatifDeTaches
 * Method:    executer
 * Signature: (Ljava/lang/Runnable;)V
 */
JNIEXPORT
void JNICALL Java_GestionnaireNatifDeTaches_executer(JNIEnv *, jobject, jobject);

/* Class:     GestionnaireNatif
 * Method:    liste
 * Signature: ()Ljava/util/Vector;
 */
JNIEXPORT
jobject JNICALL Java_GestionnaireNatifDeTaches_liste(JNIEnv *, jobject);
#endif
```

Java : assez !


C,C++/DLL
JNI
MVJ
C,C++/DLL
JNI
MVJ
Réseau
Application en C
MVJ
+
RMI
Application en C

JDBC + RMI

```
import java.sql.*;
public class GestionnaireDeTachesAvecDB implements Gestionnaire{
  ...
  private static   Connection dbConn;

  public void executer( Runnable r) throws Exception{
    …  tid =
    Statement s = dbConn.createStatement();
    int r = s.executeUpdate("INSERT INTO gestionnaireDB (tid, ……...  ");
  }
  public Vector liste() throws Exception{
    Statement s = dbConn.createStatement();
    ResultSet r = s.executeQuery("SELECT tid FROM gestionnaireDB ");
     … /* affectation du vecteur par le parcours de r */
  }

 static{ Class.forName("jdbc.hsqldb.JdbcDriver");
  dbConn = DiverManager.getConnection("jdbc:hsqldb:db1", "sa","");
  ...
  }}
```

Episode 1.2 : Activatable, Objectifs

- Activation des objets distants à la demande des clients sur le serveur
- L ’utilitaire jdk **rmid** engendre des instances à la demande
- en conséquence la classe ServeurDeTaches n’est plus résidente

java.rmi.Activatable

- **import** java.rmi.activation.Activatable;
- Il faut (entre autres)

  installer **rmid**, sur le serveur (par défaut le port 1098)

  > ***start rmid***

  en Java

  indiquer le répertoire dans lequel se trouve les classes du serveur

  soit ***file:/d:/jfd/rmi/ServeurDeTaches_Activatable/Serveur/***

  identifier le nom de la classe qui est chargée dynamiquement

  soit ***"GestionnaireDistantDeTaches"***

  ajouter un constructeur d'arité 2 en respectant cette signature

  soit *public* ***GestionnaireDistantDeTaches(ActivationID id, MarshalledObject data){***

  super(id,0); ...

  }

  transmettre éventuellement des paramètres d'initialisation, à l 'enregistrement du service

  soit *m = new* ***MarshalledObject(parametre)***

Exemple : le gestionnaire de taches

- La classe GestionnaireDistantDeTaches

  Elle doit hériter de la classe **Activatable** et ajouter le constructeur d'arité 2

  public ***GestionnaireDistantDeTaches(ActivationID id, MarshalledObject data){..}***

  Cette classe sera chargée dynamiquement

  Usage en interne de *getClass* et *newInstance*

  Le constructeur d'arité 2 sera exécuté par **rmid**

  Des paramètres peuvent être transmis par l 'intermédiaire d'une instance de la classe java.rmi.MarshalledObject

  MarshalledObject(Object parametre) extends Object

  Le paramètre est sérialisé *(soit une instance de java.io.Serializable)*

  La méthode ***get*** effectue une copie du paramètre et celui-ci est ensuite dé-sérialisé

  les autres méthodes sont : ***equals*** et ***hashcode***

extends **java.rmi.activation.Activatable**

```
import java.rmi.*;
import java.rmi.activation.*;
public class GestionnaireDistantDeTaches extends Activatable
                                          implements GestionnaireDistant{

  private GestionnaireDeTaches gestionnaire;

  public GestionnaireDistantDeTaches(ActivationID id, MarshalledObject m)
     throws RemoteException, java.io.IOException, ClassNotFoundException{

     super(id,0);
     this.gestionnaire = (GestionnaireDeTaches) m.get();
  }

  public void executer( Runnable r) throws RemoteException,Exception{
    gestionnaire.executer(r);
  }
  public java.util.Vector liste()  throws RemoteException,Exception{
    return gestionnaire.liste();
  }}
```

La classe GestionnaireDeTaches

- **L ’adapté s ’est adapté**

- *this.gestionnaire = (GestionnaireDeTaches) m.get();*

```
import java.io.Serializable;
public class GestionnaireDeTaches  implements Gestionnaire,
                                                Serializable{

  private transient ThreadGroup table;
  private Vector      liste;

....
```

ServeurDeTaches$_1$

```
import java.rmi.*;
import java.rmi.activation.*;
import java.util.Properties;

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

   // référence sur "java.policy" , les contraintes de sécurité doivent
   // doit être fournies à l 'ActivationGroup, ici par l 'intermédiaire
   // d'une instance de Properties (qui hérite de HashTable)

    Properties props = new Properties();
    props.put("java.security.policy", "d:/jfd/rmi/java.policy");

     ActivationGroupDesc.CommandEnvironment ace = null;
     ActivationGroupDesc exampleGroup = newActivationGroupDesc(props,ace);
```

ServeurDeTaches$_2$

```
    // Obtention de son ID, (port)
    ActivationGroupID agi;
    agi = ActivationGroup.getSystem().registerGroup(exampleGroup);
    // Création
    ActivationGroup.createGroup(agi, exampleGroup, 0);


    // Création d 'une instance de ActivationDesc, cette instance
    // fournit toutes les informations à rmid

String location;
// le répertoire dans lequel se trouve la classe gérant les objets distants
location  = "file:/d:/jfd/rmi/ServeurDeTaches_Activatable/Serveur/";
// le paramètre d 'initialisation ici l'adapté adapté
MarshalledObject m = new MarshalledObject(new GestionnaireDeTaches("mon1"));

// Création
ActivationDesc desc;
desc = new ActivationDesc ("GestionnaireDistantDeTaches",location,m);
```

ServeurDeTaches$_3$

```
// enregistrement du service
try{

  GestionnaireDistant mon1 =(GestionnaireDistant)Activatable.register(desc);

// au lieu de
// GestionnaireDistant mon1 = new GestionnaireDistantDeTaches(
//                                        new GestionnaireDeTaches("mon1"));


  Naming.rebind( GestionnaireDistant.nomDuService, mon1);
  System.out.println("le serveur : " + GestionnaireDistant.nomDuService +
                     "  est enregistre ");
  }catch(Exception e){throw e;}}
  System.exit(0);
}

// le programme se termine
   //( une fenêtre liée à rmid mentionne les appels distants)
```

Les commandes, client et serveur

- ***Compilation*** : javac et rmic GestionnaireDistantDeTaches

- ***Exécution du serveur***

```
set CLASSPATH=
start rmiregistry
start rmid
set CLASSPATH=d:/jfd/rmi/ServeurDeTaches_Activatable/Serveur/
java
  -Djava.rmi.server.codebase=file:/d:/jfd/rmi/ServeurDeTaches_Activatable/
 Serveur/ -Djava.security.policy=java.policy ServeurDeTaches
```

- ***Exécution du client***

```
start java SimpleHttpd 8088
java
  -Djava.rmi.server.codebase=lmi27:8088/d:/jfd/rmi/ServeurDeTaches_
Activatable/Client/ -Djava.security.policy=java.policy TacheCliente
```

Activatable et plus

- **La classe peut ne pas hériter de Activatable**

  `Activatable.exportObject(this, id, 0);`

  `sera présente dans le corps du constructeur d'arité 2 de la classe distante`

  public ***ClasseDistante(ActivationID id, MarshalledObject data){..}***

- **On peut se servir de l ’instance de MarshalledObject afin de rendre certaines données du serveur persistantes**

  auprès de rmid

  m = new MarshalledObject(new File("d:/rmi/persistent.ser"));

  et au sein du constructeur

  f =(File) m.get();

  `voir http://java.sun.com/products/jdk/1.2/docs/guide/rmi/activation.html`

Conclusion

- **RMI: Remote Method Innovation ?**
  - **passage de paramètres**
  - **transmission de code**
- **RMI 1.2, Activatable**
- **JINI**
  - **« Plug and play »**
- **A suivre, à compléter**
  - Synchronisation entre plusieurs clients ? (synchronized…)
  - Passage de messages, RemoteEventObject,...
  - MulticastRemoteObject,
  - ....
  - JNDI, JRMS,