Programmation Réseau

RMI

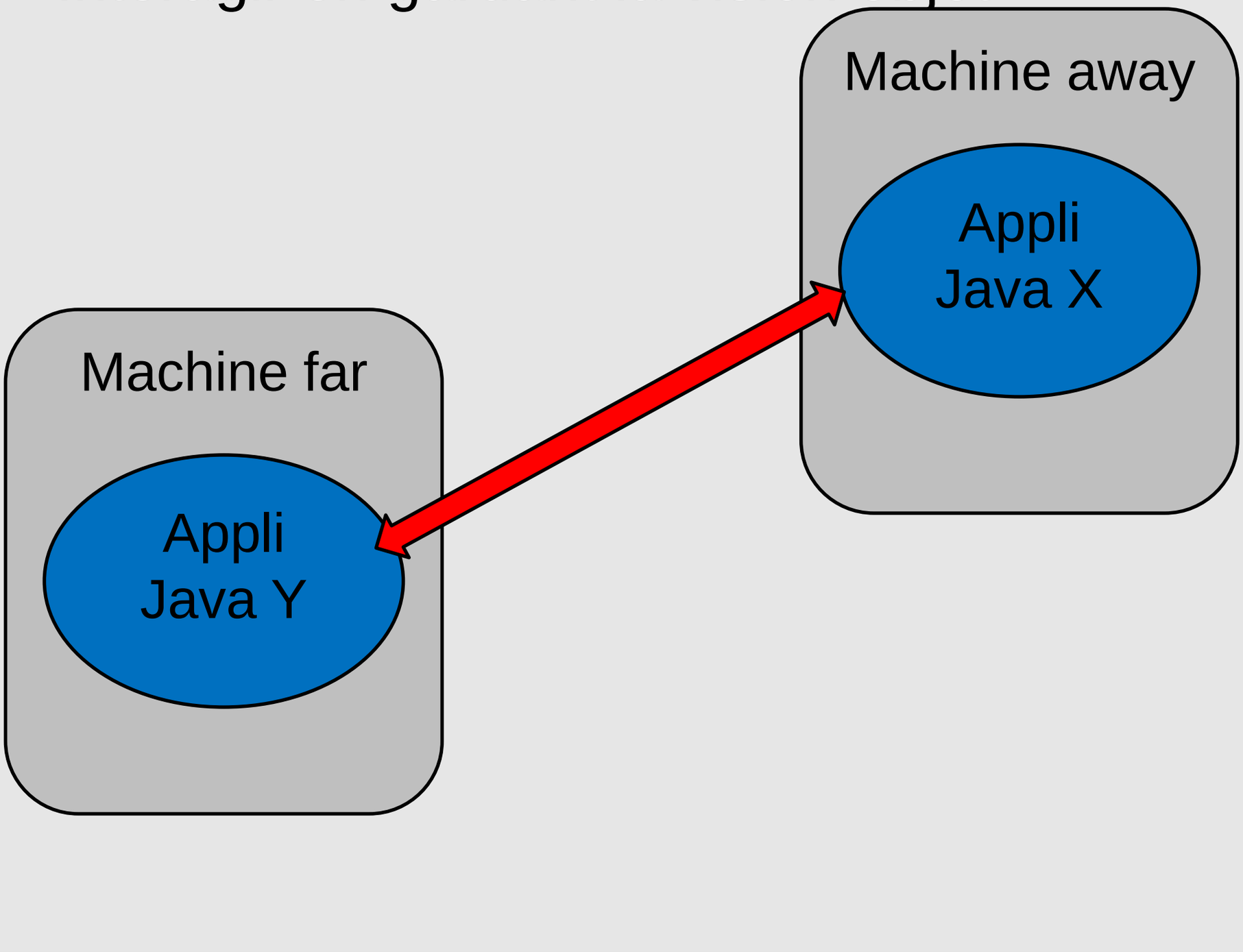
Jean-Baptiste.Yunes@liafa.jussieu.fr

Coloriages: François Armand
armand@informatique.univ-paris-diderot.fr

UFR Informatique

2011-2012

Interagir en gardant la vision objet



Les RMI de Java

Les applications RMI sont des applications bâties sur le modèle objet de Java et dans lesquelles les objets sont répartis dans différents processus (en

on comprend donc :

entre objets

appels de méthodes

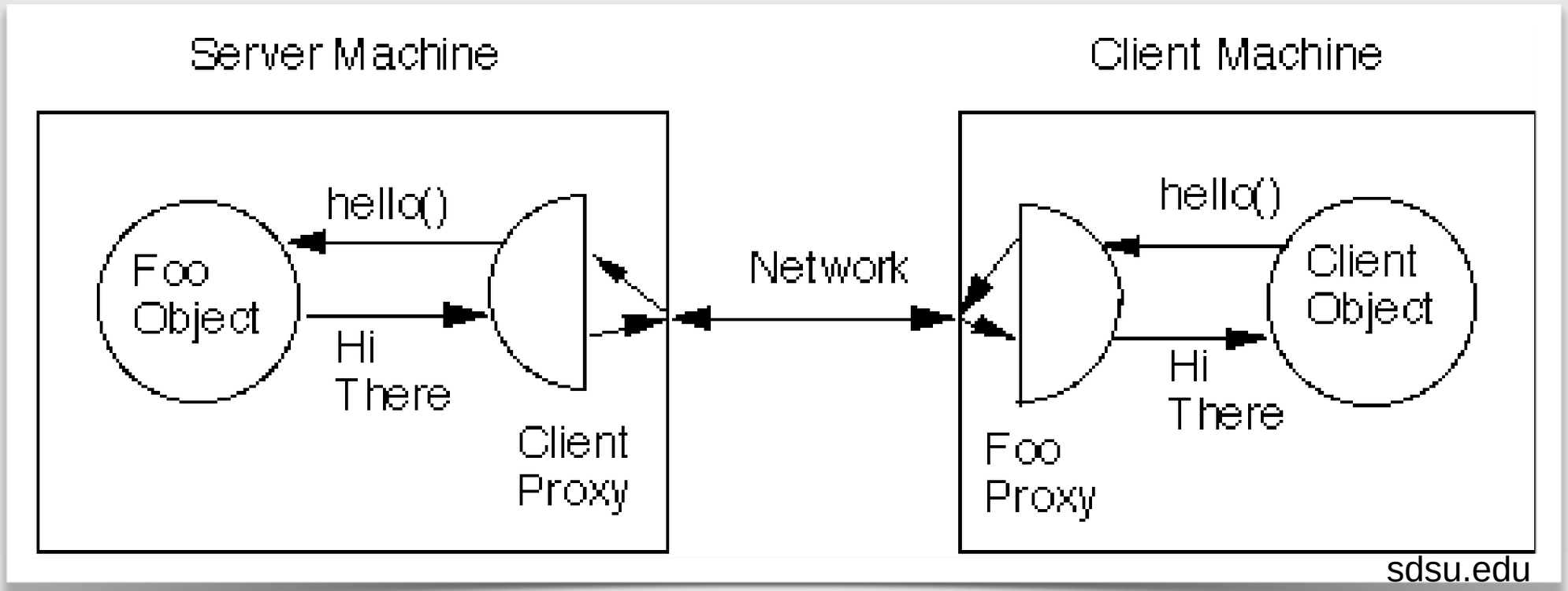
transparente la

Un appel de méthode sur un objet distant doit être **syntaxiquement**
méthode sur un objet local

communications nécessaires pour accéder à un objet, en fournissant un objet local de substitution:

qui délègue tous les appels, à travers le

Le schéma général est alors le suivant :



sdsu.edu

Ce mécanisme repose sur une technique bien connue de délégation :

Le design pattern **proxy**

Pour les RMI, la terminologie est différente

celle des RPC :

talon (stub)

le proxy côté serveur est un **squelette** (skeleton)

uquebec.ca

Bien entendu de nombreux problèmes sont à résoudre :

Comment créer les talons et squelettes ?

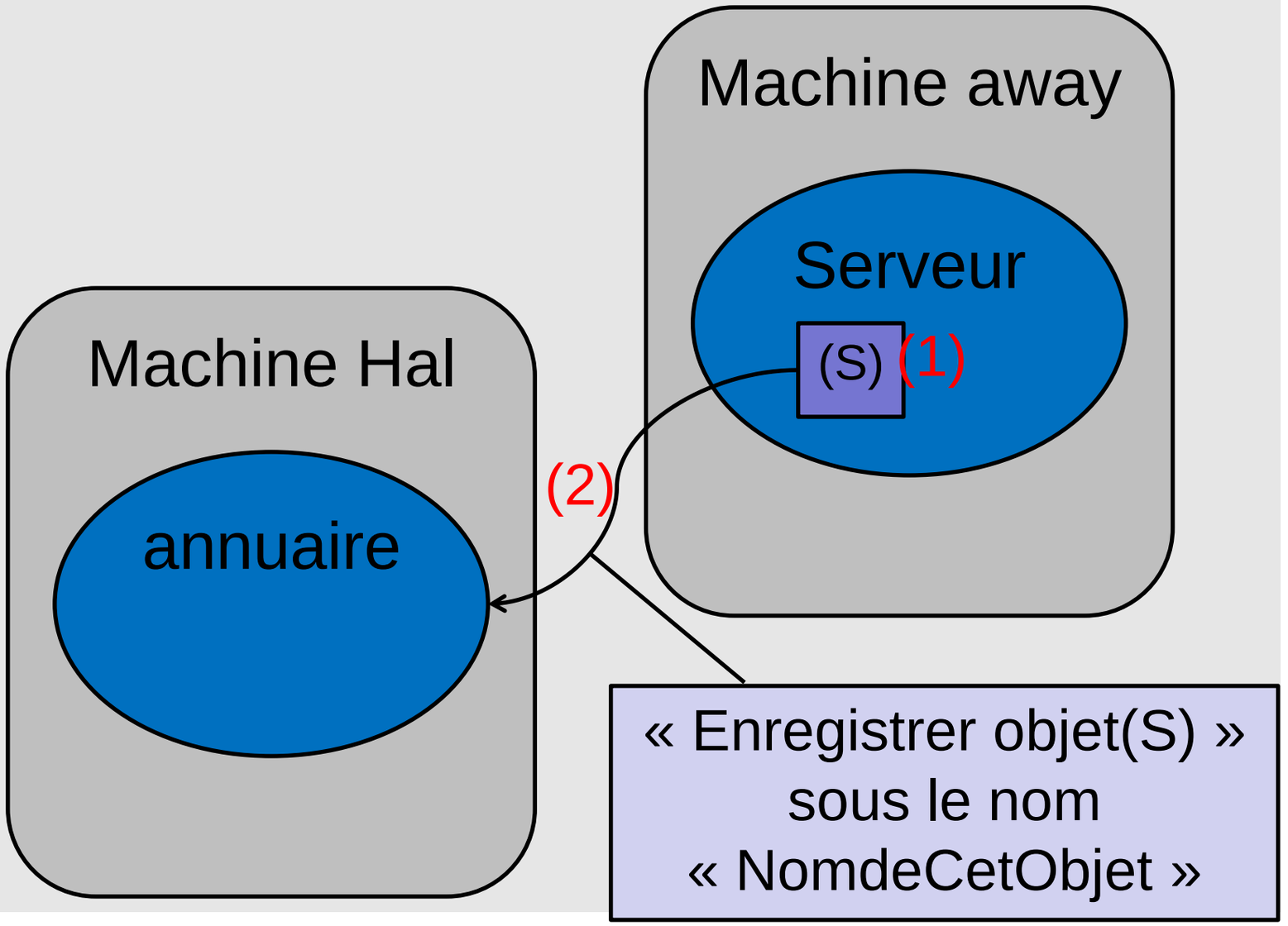
Comment localiser un objet distant ?

Schéma général

(0) Définir une interface

(1) Créer un objet implémentant cette interface

(2) Lui associer un nom dans un annuaire



Java RMI

(3) R

(4)

(S)

Machine

(3)

annuaire

et »

Tout objet qui désire être exposé à travers les RMI doit :

implémenter une interface qui elle-même doit :

java.rmi.Remote

contenir des méthodes dont la signature contient une clause throws faisant apparaître

java.rmi.RemoteException

dont tous les paramètres ou valeurs de retour doivent être sérialisables, i.e. implémenter

java.io.Serializable

spécialiser la classe

java.rmi.server.UnicastRemoteObject

exposés

registre RMI

(RMI registry)

le service réseau correspondant est

rmiregistry

void bind(String nom,Remote o);

de la classe java.rmi.Naming

où :

nom est de la forme //machine:port/id
(machine et port sont optionnels)

o

statique (Object)lookup(String nom);
de la classe Naming

des méthodes

attention, il est plus prudent de ne

différent mécanismes de sécurité

politique de sécurité, deux façons de les paramétrer :

installer en interne un SecurityManager adéquat

surcharger en externe le paramétrage de la politique de sécurité, via la propriété java.security.policy

pour le paramétrage externe :

lancer la JVM en spécifiant quel fichier contient les paramètres de sécurité avec la commande

```
java -Djava.security.policy=fichier
```

créer un fichier spécifiant les valeurs des paramètres, comme (ici la politique la moins restrictive) :

```
grant {

}
```

On notera :

UnicastRemoteObject crée un Thread côté serveur; ce Thread est dédié à la gestion de la communication côté serveur

communications sont cachées et les erreurs doivent être traitées)

Limite de la transparence!

que des problèmes de concurrence peuvent apparaître

que le garbage collector est particulier (dgc + interface Unreferenced)

Exemple 1 : calculette

```
public interface Calculator extends
    java.rmi.Remote {

public long add(long a, long b) throws
    java.rmi.RemoteException;

public long sub(long a, long b) throws
    java.rmi.RemoteException;

}
```

Exemple 1 : calculette

Implémentation du service: CalculatorImpl.java

```
public class CalculatorImpl
    extends java.rmi.server.UnicastRemoteObject
    implements Calculator {

// Implementations must have an explicit constructor
// in order to declare the RemoteException exception

public CalculatorImpl()
    throws java.rmi.RemoteException {
    super();
}
```

Exemple 1 : calculette

Implémentation du service: CalculatorImpl.java (suite)

```
public long add(long a, long b)
     throws java.rmi.RemoteException {
     return a + b;
 }

public long sub(long a, long b)
      throws java.rmi.RemoteException {
      return a - b;
}
```

Exemple 1 : calculette

Implémentation du serveur: CalcSrv.java

```
import java.rmi.Naming;
public class CalcSrv {

    public CalcSrv() {
            try {
                Calculator c = new CalculatorImpl();

                Naming.rebind("rmi://localhost:1099/CalSrv", c);

            } catch (Exception e) {
                System.out.println("Trouble: " + e);
            }
    }
    public static void main(String args[]) {
            new CalcSrv();
    }
}
```

Exemple 1 : calculette

Implémentation du client: CalcClient.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

public class CalculatorClient {
    public static void main(String[] args) {
        try {
            Calculator c = (Calculator) Naming.lookup (
                                "rmi://localhost//CalSrv");
            System.out.println( c.sub(4, 3) );
        }
```

Compiler

Attention, il faut lancer la commande rmiregistry

Exécuter le serveur

Qui se met en attente

Exécuter un ou plusieurs clients

On peut réaliser les choses de manière plus « intégrées » pour se débarrasser du besoin de lancer rmiregistry.

Un autre exemple: (from wikipedia)

Interface:

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

public interface RmiServerIntf extends Remote {
    public String getMessage()
        throws RemoteException;
}
```

Un autre exemple:

Serveur

```
public class FaRmiServer extends UnicastRemoteObject
    implements RmiServerIntf {

    public static final String MESSAGE = "Hello world";

    // Constructeur à ne pas oublier
    public FaRmiServer() throws RemoteException {    }

    public String getMessage() { return MESSAGE; }
```

Un autre exemple:
Serveur

```
public static void main(String args[]) {

    // Create and install a security manager
    if (System.getSecurityManager() == null) {
            System.setSecurityManager(new
                                    RMISecurityManager());
    }
    try {
        //special exception handler for registry creation
        LocateRegistry.createRegistry(1099);
    } catch (RemoteException e) {
        //do nothing, error means registry already exists
    }
```

Un autre exemple:
Serveur

```
try {
    //Instantiate RmiServer
    FaRmiServer obj = new FaRmiServer();

    // Bind this object instance to the name "RmiServer"
    Naming.rebind("//localhost/RmiServer", obj);

    System.out.println("PeerServer bound in registry");
    } catch (Exception e) {
        System.err.println("RMI server exception:" + e);
    }
}
}
```

Un autre exemple:
Serveur

```
try {
    //Instantiate RmiServer
    FaRmiServer obj = new FaRmiServer();

    // Bind this object instance to the name "RmiServer"
    Naming.rebind("//localhost/RmiServer", obj);

    System.out.println("PeerServer bound in registry");
    } catch (Exception e) {
        System.err.println("RMI server exception:" + e);
    }
}
}
```

Un autre exemple:
Client

```
public static void main(String args[]) {

    // Create and install a security manager
    if (System.getSecurityManager() == null) {
            System.setSecurityManager(new
                RMISecurityManager());
    }

    FaRmiClient cli = new FaRmiClient();
    System.out.println(cli.getMessage());
}
```

Un autre exemple:
Client

```
public class FaRmiClient {
    // "obj" is the reference of the remote object
    RmiServerIntf obj = null;
    public String getMessage() {
            try {
                obj = (RmiServerIntf) Naming.lookup
                                ("//localhost/RmiServer");
                return obj.getMessage();
            } catch (Exception e) {
                System.err.println(" exception: " + e);
                return e.getMessage();
            }
        }
```

On sait donc invoquer des méthodes sur un objet (serveur) depuis un client

Donc depuis plusieurs clients simultanément!

Les clients(resp. les serveurs) sont ils « fiables »?

Disparition brutale de serveurs!

Disparition brutale de clients!

Activation

Dans le modèle des objets distribués précédent les objets préexistent côté serveur

rapproche plus du modèle des services (à la mode du super-

il est important de savoir que

les objets activables sont regroupés par

lorsque nécessaire

Du côté serveur, un objet activable doit :

étendre la classe Remote

étendre la classe java.rmi.activation.Activatable

implémenter un constructeur à deux arguments de type

java.rmi.activationActivationID

java.rmi.MarshalledObject<T>

Ensuite il est nécessaire de créer un programme permettant la mise en place qui consiste à :

positionner une politique de sécurité adaptée pour la JVM qui sera lancée

déclaration après du système de nommage

La commande rmid correspond au démon qui prend en charge les objets activables

cette commande doit être utilisée en partenariat avec le registre RMI rmiregistry

```
import java.rmi.*;

public interface Hello extends Remote {
    String hello(String name) throws
 RemoteException;
}
```

Activation

```
import java.rmi.*;
import java.rmi.activation.*;
import java.io.*;

class ConcreteHello extends Activatable
                    implements Hello {
    private String name;
    public ConcreteHello(ActivationID id,
                                    MarshalledObject<String> o)
                throws RemoteException,IOException, ClassNotFoundException {
        super(id,0);
        name = o.get();
        System.out.println("ctor ConcreteHello "+name);
    }

    public String hello(String name)
                    throws RemoteException {
        return this.name+": Hello "+name;
    }
}
```

Activation

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

        // Politique de sécurité de la JVM activée
        Properties props = new Properties();
        props.put("java.security.policy", "my.policy");

        // paramètres de la JVM
        ActivationGroupDesc.CommandEnvironment ace = null;

       // Création d'un descripteur du groupe
        ActivationGroupDesc exampleGroup = ActivationGroupDesc(props, ace);

        // Enregistrement du groupe auprès du rmid
        ActivationGroupID agi =
                ActivationGroup.getSystem().registerGroup(exampleGroup);

      // objet qui sera passé en paramètre du constructeur
        MarshalledObject<String> data =
                new MarshalledObject<String>(new String("serveur"));
```

```
                                                    "file:/Users/yunes/RMI/activation/",
                                                    data);


    // Récupération de la souche
    Hello obj = (Hello)Activatable.register(desc);

    // Enregistrement de la souche auprès du rmiregistry
    Naming.rebind("///HelloServer", obj);
        System.exit(0);
    }
}
```