RMI
Remote Method Invocation

Didier DONSEZ

Université Joseph Fourier (Grenoble 1) IMA – LSR/ADELE

Didier.Donsez@imag.fr, Didier.Donsez@ieee.org

Hafid Bourzoufi

Université de Valenciennes - ISTV

Sommaire

- Rappel et Limites des RPC (Remote Procedure Call)
- Principe des RMI
- Etapes de développement et d ’exécution
- Paramètres des méthodes
- Objet Activable
- Personnalisation de la couche de transport
- Ramasse-Miette distribuée
- Autour des RMI
  - CORBA, IIOP, EJB

Rappel des RPC

RPC (Remote Procedure Call)

- modèle client/serveur
- appel de procédures à distances entre un client et un serveur
  - le client appelle une procédure
  - le serveur exécute la procédure et renvoie le résultat
- Outil **`rpcgen`**
  - génère la souche d 'invocation et le squelette du serveur
    (en C, C++, Fortran, …)
  - la souche et le squelette ouvre un socket BSD et encode/décode les paramètres
- Couche de présentation **XDR** (eXchange Data Representation)

  format pivot de représentation des données de types primitifs et structurés (tableau de longueur variable, structures) quelque soit

  ↳ l 'architecture (Little Endian/Big Endian, IEEE, …)

  ↳ le langage (ordre ligne/colonne dans les tableaux C et les tableaux Fortran)

  ↳ ou le système (ASCII, IBM 'ECDCII, ...)

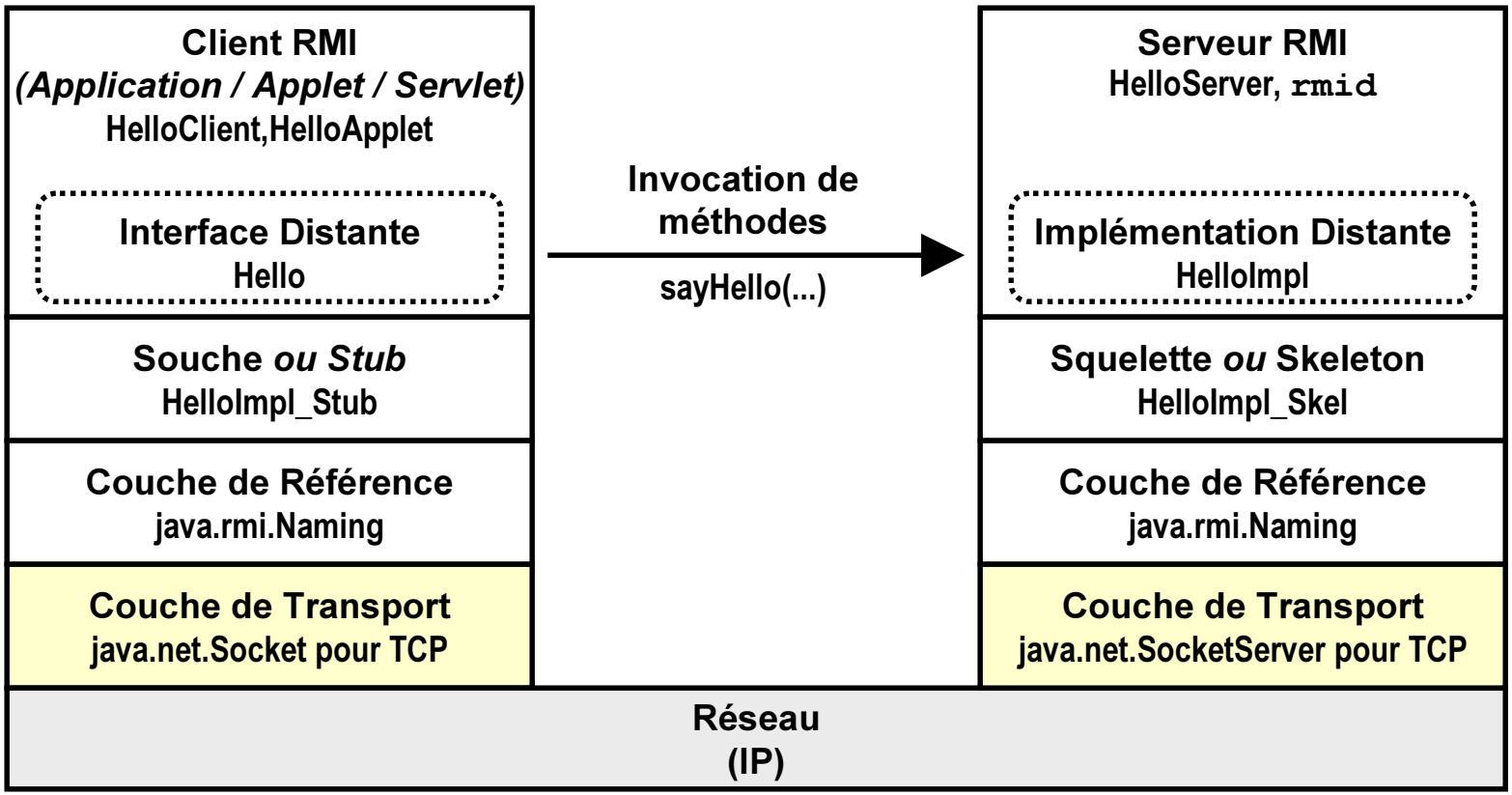
Limites des RPC

- Limitations
  - paramêtres et valeur de retour sont des types primitifs
  - programmation procédurale
  - dépendance à la localisation du serveur
  - pas d 'objet
  - pas de « référence distante »
- Evolutions
  - CORBA
    - Multilangage, multi-plateforme (architecture+OS), MuliVendeurs
  - Java RMI
    - mono-langage : Java, multiplateforme : de JVM à JVM
  - DCOM / Object RPC / .NET Remoting
    - multi-langages, plateforme Win32 principalement, il existe des implémentations (non MicroSoft) pour Unix, Propriétaire
  - .NET Remoting
    - multi-langages (CLR), plateforme Win32 principalement
    - Normalisé à l'ECMA et à l'ISO
  - SOAP (Simple Access Object Protocol)
    - multi-langages, multi-plateforme
    - Réponse et requête en XML (DTD SOAP), Transport sur HTTP, IETF

Principes des RMI

- RPC à la Java
  - invoquer de façon simple des méthodes sur des objets distribués.
- Outils
  - pour la génération des stub/skeleton, l 'enregistrement par le nom, l 'activation
    - Tous les détails (connexion, transfert de données ..)
      sont transparents pour le développeur grâce au stub/skeleton généré
- Mono-langage et Multiplateforme.
  - Java : de JVM à JVM *(les données et objets ont la même représentation qqs la JVM)*
- Orienté Objet
  - Les RMIs utilisent le mécanisme standard de sérialisation de JAVA
    pour l 'envoi d 'objets.
- Dynamique
  - Les classes des Stubs et des paramêtres
    peuvent être chargées dynamiquement via HTTP (http://) ou NFS (file:/)
- Sécurité
  - un SecurityManager vérifie si certaines opérations sont autorisés par le serveur

Structure des couches RMI (i)
l 'architecture logique



Structure des couches RMI (ii)

- Souche *ou Stub* (sur le client)
  - représentant local de l'objet distant qui implémente les méthodes "exportées" de l 'objet distant
  - "marshalise" les arguments de la méthode distante et les envoie en un flot de données au serveur
  - "démarshalise" la valeur ou l 'objet retournés par la méthode distante
  - la classe xx_Stub peut être chargée dynamiquement par le client (Applet)
- Squelette *ou Skeleton* (sur le serveur)
  - "démarshalise" les paramètres des méthodes
  - fait un appel à la méthode de l'objet local au serveur
  - "marshalise" la valeur ou l 'objet renvoyé par la méthode

Structure des couches RMI (ii)

Couche des références distantes

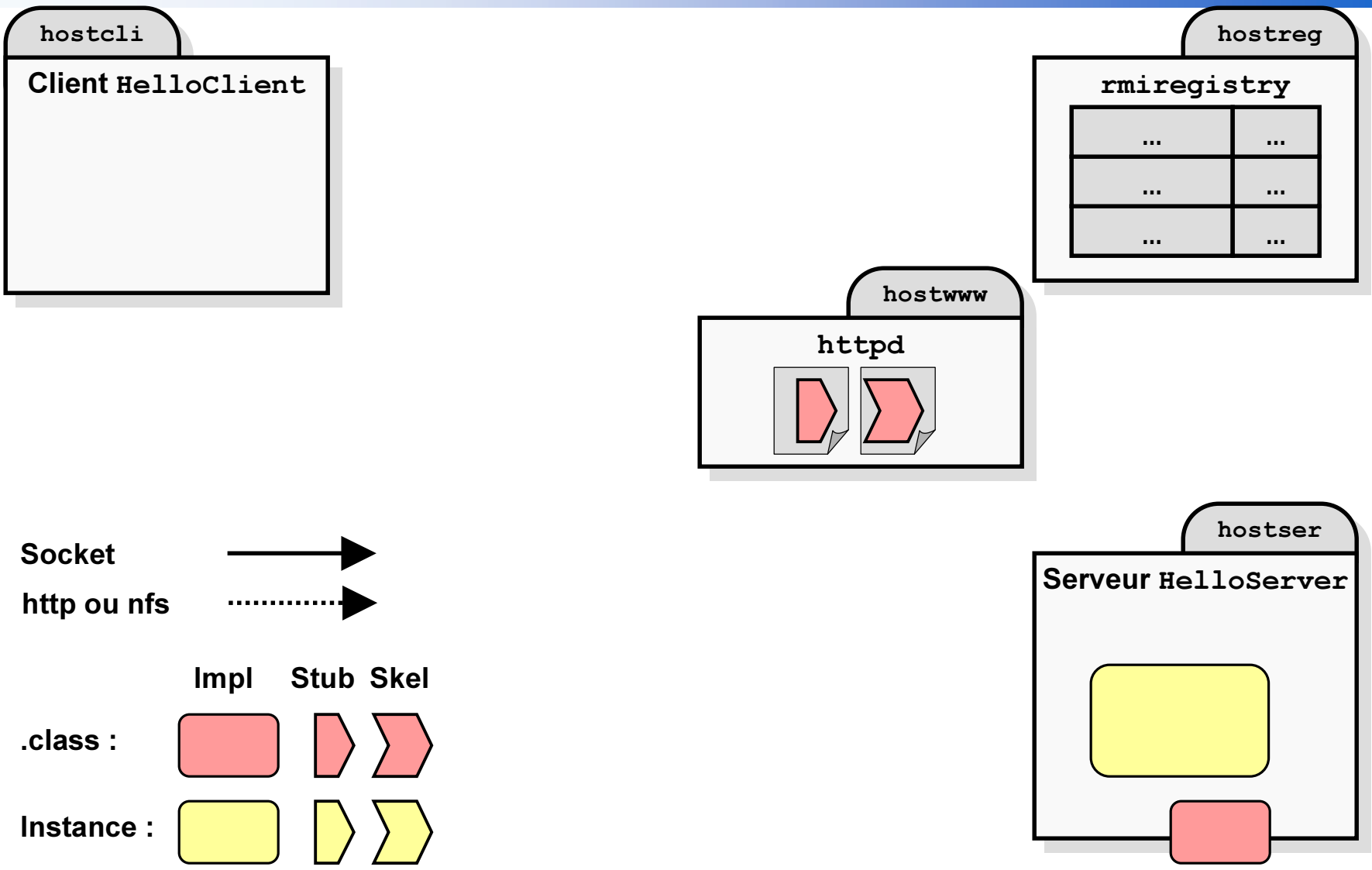
- traduit la référence locale au stub en une référence à l'objet distant
- elle est servie par un processus tier : rmiregistry

Couche de transport

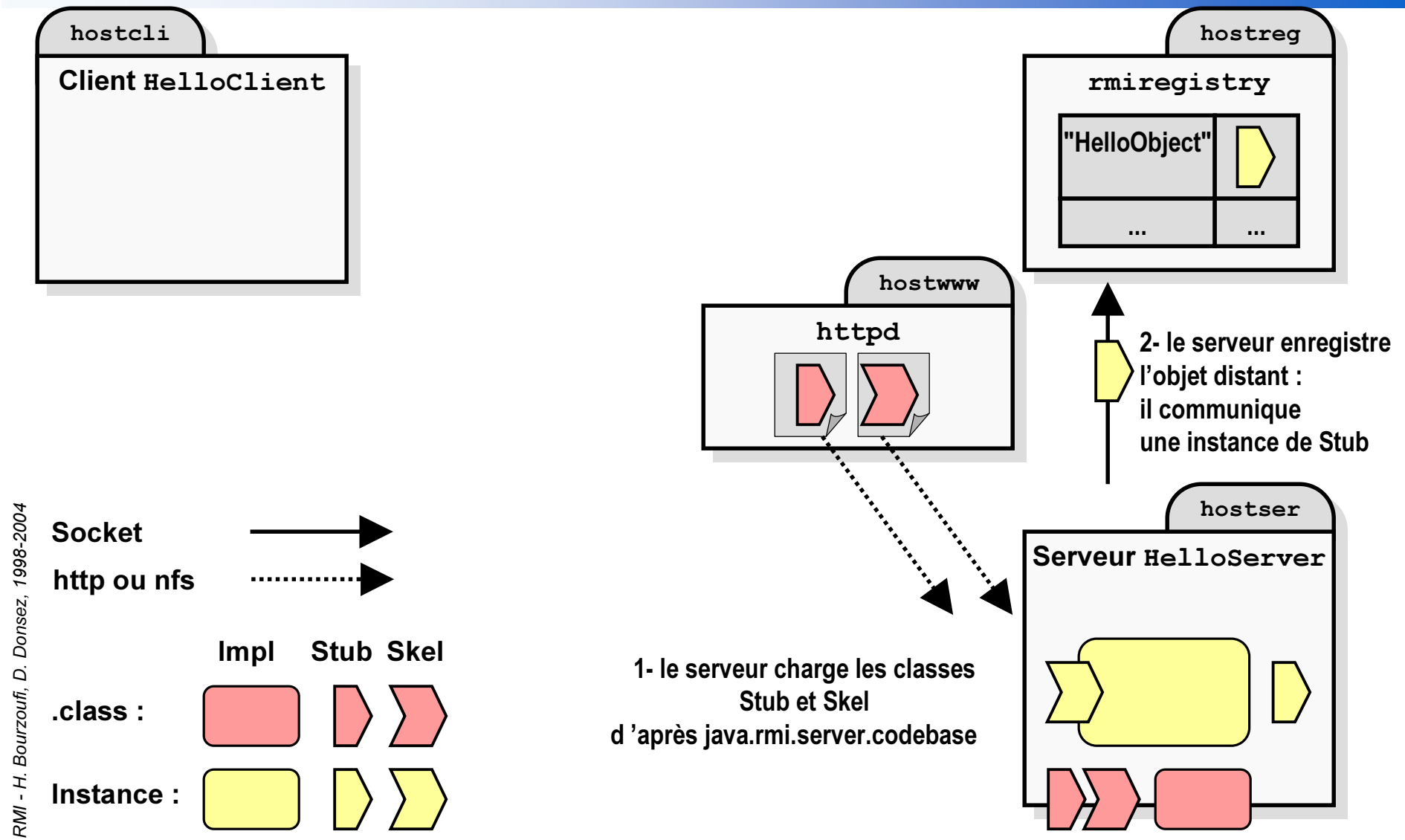
- écoute les appels entrants
- établit et gère les connexions avec les sites distants

- java.rmi.UnicastRemoteObject utilise les classes Socket et SocketServer (TCP)
- cependant d 'autres classes peuvent être utilisées par la couche transport (Compression sur TCP, SSL sur TCP, UDP)

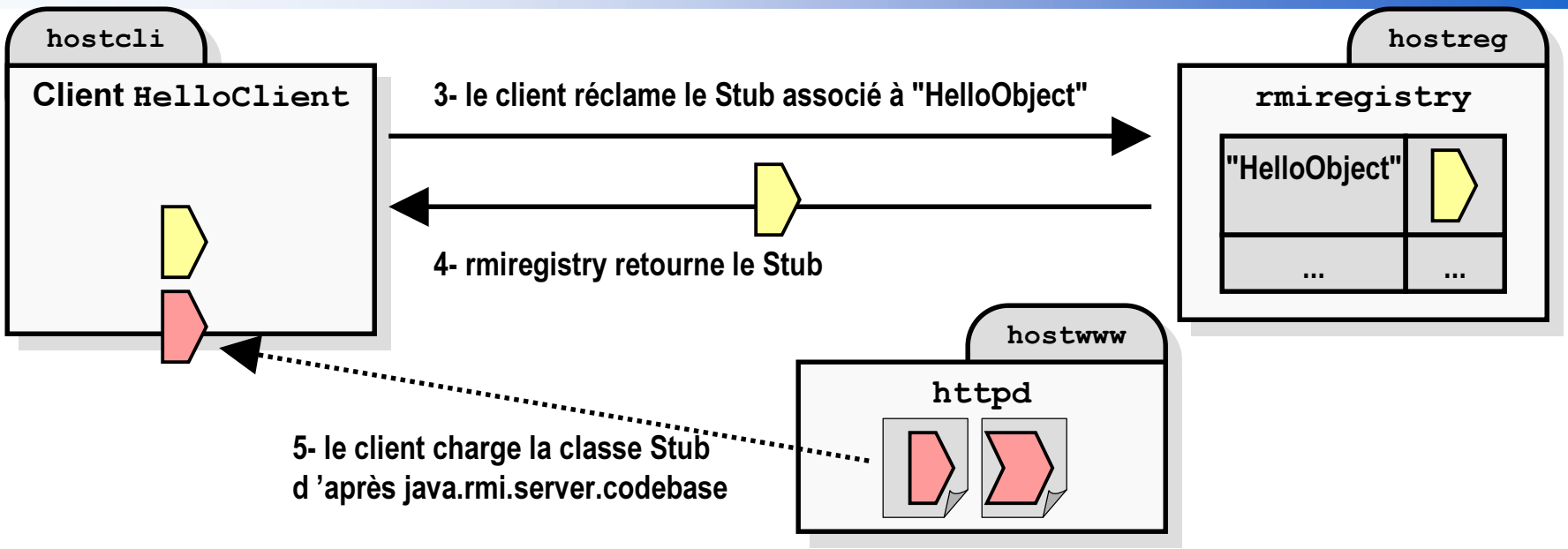
La configuration

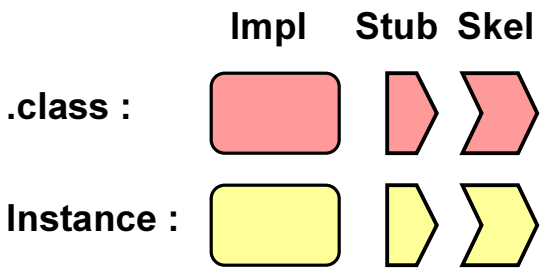


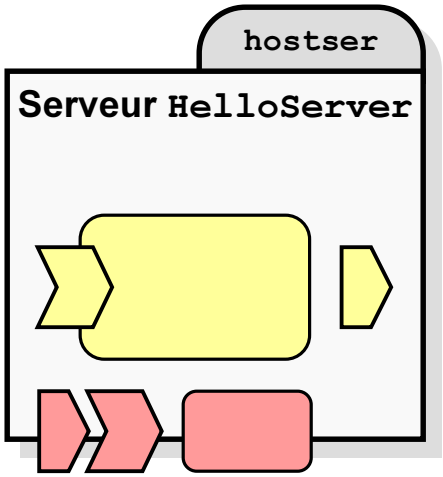
L ’enregistrement de l ’objet



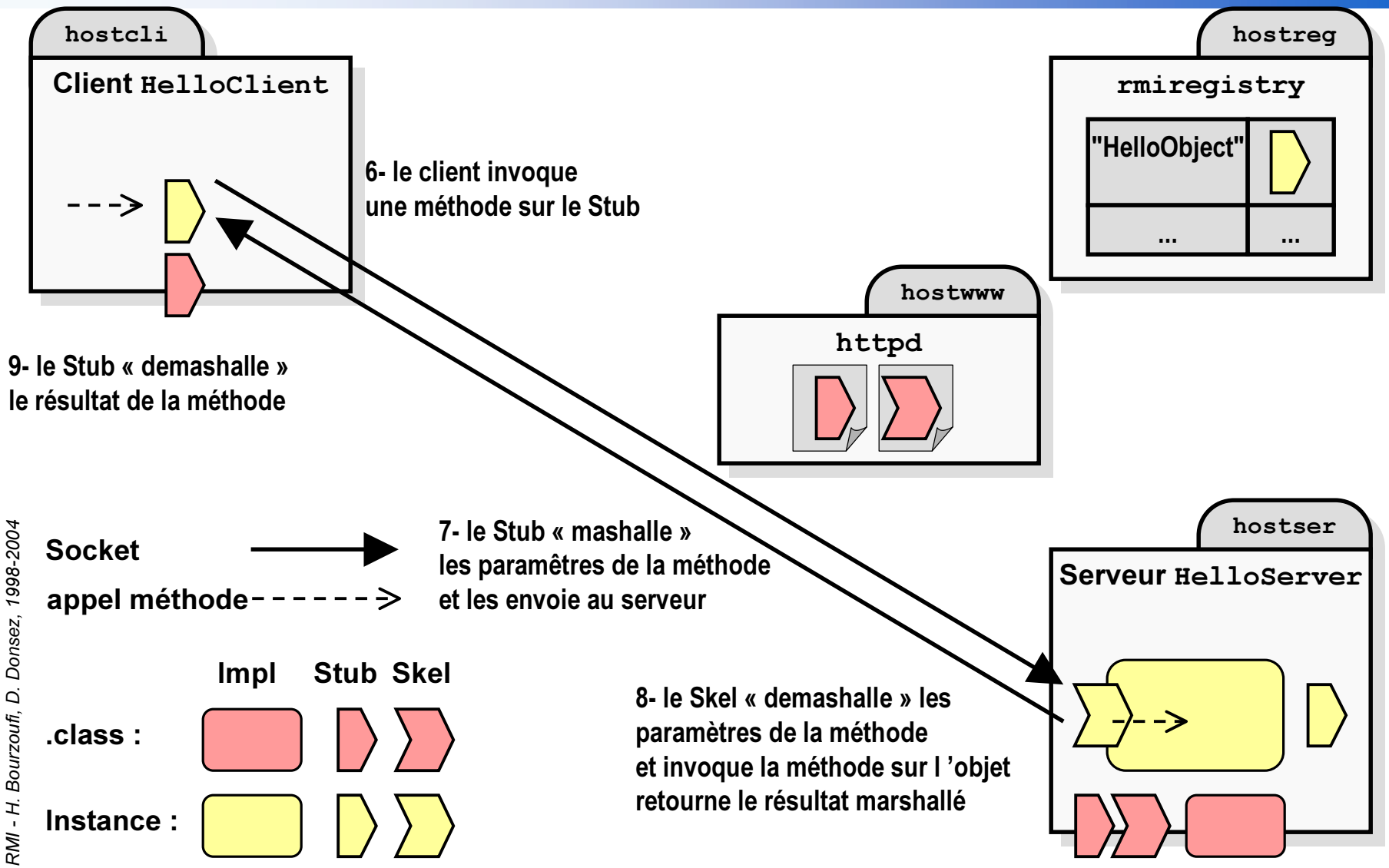
La récupération du Stub







Invocation d 'une méthode



Création et manipulation d'objets distants

- 5 Packages
  - **java.rmi** : pour accéder à des objets distants
  - **java.rmi.server** : pour créer des objets distants
  - **java.rmi.registry** : lié à la localisation et au nommage d'objets distants
  - **java.rmi.dgc** : ramasse-miettes pour les objets distants
  - **java.rmi.activation :** support pour l 'activation d 'objets distants.
- Etapes du développement

  1- Spécifier et écrire l'interface de l'objet distant.

  2- Ecrire l'implémentation de cette interface.

  3- Générer les Stub/Skeleton correspondants. (outil rmic)
- Etapes de l 'exécution

  4- Ecrire le serveur qui instancie l'objet implémentant l'interface, exporte son Stub puis attend les requêtes via le Skeleton.

  5- Ecrire le client qui réclame l 'objet distant, importe le Stub et invoque une méthode de l'objet distant via le Stub.

1- Spécifier l 'interface d'un objet distant

- Format
  - l 'interface étend **java.rmi.Remote**
  - les méthodes doivent pouvoir lever **java.rmi.RemoteException**
- Exemple

```
package examples.hello;
public interface Hello extends java.rmi.Remote{
    public static final int EN=0;     // English
    public static final int FR=1;     // French
    public static final int ES=2;     // Spanish
    String sayHello() throws java.rmi.RemoteException;
    String sayHello(String name) throws java.rmi.RemoteException;
    String sayHello(String name, int lang) throws java.rmi.RemoteException;
}
```

Implémentation de l 'objet distant

La classe HelloImpl

- doit implémenter l 'interface distante Hello
- et étendre une des sous-classes de java.rmi.server.RemoteServer
  comme java.rmi.server.UnicastRemoteObject

java.rmi.server.UnicastRemoteObject

- sous classe le plus souvent utilisée

- offre toutes les fonctionnalités des classes distantes
- appelle la classe squelette pour la (dé)marshalisation
- utilise TCP/IP pour la couche transport

Implémentation de l 'objet distant

```
package examples.hello;
public class HelloImpl extends java.rmi.server.UnicastRemoteObject implements Hello {
   private int defaultlang;
   public HelloImpl(int defaultlang) throws java.rmi.RemoteException{
      super(); this.defaultlang=defaultlang; }
   public String sayHello() { return  sayHello(null, defaultlang); }
   public String sayHello(String name) { return  sayHello(name, defaultlang); }
   public String sayHello(String name, int lang) {
     switch(lang) {
        case Hello.EN : break; case Hello.FR : break; case Hello.ES : break;
        default : lang=Hello.EN; }
      switch(lang) {
        case Hello.EN : return "Hello " +((name==null) ? "World" : name) + " !";
        case Hello.FR : return "Bonjour " +((name==null) ? "tout le monde" : name) + " !";
        case Hello.ES : return "Hola " +((name==null) ? "todo el mundo" : name) + " !";
        default : return null;  }
   } // method String sayHello(String name, int lang)
} // class HelloImpl
```

La génération des Stub et Skeleton

- L'outil `rmic` génère
  - la classe souche **examples.hello.HelloImpl_Stub**
  - la classe squelette **examples.hello.HelloImpl_Skel**

  à partir de l'implémentation **examples.hello.HelloImpl**

- Exemple *(sous Win32)*

```
set CLASSPATH=%CLASSPATH%;./myclasses
javac -d .\myclasses Hello.java
javac -d .\myclasses HelloImpl.java
rmic -keepgenerated -d ./myclasses examples.hello.HelloImpl
```

Remarque: **-keepgenerated** à conserve les sources des Stub et Skeleton

Implémentation du serveur de l 'objet distant

- Crée un ou plusieurs objets distants (une ou plusieurs classes)
- Naming.bind() : les enregistre auprès du serveur de liaison `rmiregistry`

Exemple

```
package examples.hello;
public class HelloServer {
public static void main(String args[]) { // argument : l 'hôte/port du rmiregistry
      // Crée et Installe un Security Manager
      if (System.getSecurityManager() == null) {
         System.setSecurityManager(new java.rmi.RMISecurityManager()); }
      try {
         // instancie l 'objet
         HelloImpl obj = new HelloImpl(Hello.EN);
         // Enregistre l 'objet sous le nom "HelloObject" auprès de rmiregistry
         java.rmi.Naming.bind("//"+args[0]+"/HelloObject", obj);
      } catch (Exception e) { e.printStackTrace();  }
}}
```

- javac -d .\myclasses HelloServer.java

Implémentation du serveur de l 'objet distant

Remarques

- L 'objet distant servi peut être d 'une sous classe de HelloImpl

```
public class HelloESImpl extends HelloImpl {
    public HelloESImpl() throws java.rmi.RemoteException{ super(HelloES); }
}
// et dans HelloServer
java.rmi.Naming.rebind("//"+args[0]+"/HelloObject", new HelloESImpl());
```

- Le serveur peut crée et enregistré
  plusieurs objets appartenant à une ou plusieurs classes
- l 'enregistrement peut se faire auprès des plusieurs rmiregistry

Implémentation du serveur de l 'objet distant

Remarques

- il faut prévoir une procédure d 'arrêt (`shutdown`) du serveur
  - Demande d 'arrêt

    Unix : kill SIGQUIT 12345

    Win32 : ??
  - Arrêt des objets **UnicastRemoteObject**

    public static void Naming.unbind(String name)

    public static boolean UnicastRemoteObject.unexportObject(Remote, boolean force)

- Dans les exemples et tutoriels, le serveur est constitué par la méthode **main(String args[])** de l 'implémentation **HelloImpl**

Implémentation d 'un client invoquant l 'objet distant

- Demande un stub auprès du serveur de liaison `rmiregistry`
- invoque des méthodes sur le stub chargé

Exemple (Application)

```
package examples.client;
public class HelloClient {
public static void main(String args[]) {
    String message = "blank";
    try { // récupère le stub de l 'objet enregistré au nom de « HelloObject »
       Hello obj = (Hello) java.rmi.Naming.lookup("//" + args[0] + "/HelloObject");
      // invocation des 3 méthodes
       message = obj.sayHello();                    System.out.println(message);
       message = obj.sayHello(args[1]);             System.err.println(message);
       System.err.println( obj.sayHello(args[1], Integer.parseInt(args[2])));
    } catch (Exception e) { e.printStackTrace(); }
   } }
```

- javac -d .\myclasses HelloClient.java

Implémentation d ’un client invoquant l ’objet distant

Exemple (Applet)

```
import java.rmi.Naming;
public class HelloApplet extends java.applet.Applet{
   String message = "blank";
   public void init() {
      try { // récupère le stub de l ’objet enregistré au nom de « HelloObject »
       // Remarque : rmiregistry et le serveur Web doit être sur la même machine (même #IP)
        Hello obj = (Hello)Naming.lookup("//" + getCodeBase().getHost() + "/HelloObject");
       // invocation d ’une des 3 méthodes
        message = obj.sayHello();
      } catch (Exception e) { // sortie d ’erreur sur la console
        System.out.println("HelloApplet exception:" + e.getMessage()); e.printStackTrace();
      }
   }
   public void paint(java.awt.Graphics g) {  g.drawString(message, 25, 50); }
}
```

- javac -d .\myclasses HelloApplet.java

L 'exécution Coté Serveur

Le serveur de liaison `rmiregistry`

- expose un objet distant serveur de liaisons (de noms)
  - le port de liaison par défaut est le port TCP 1099

```
hostreg> rmiregistry
^C
hostreg> rmiregistry 2001
^C
hostreg> rmiregistry 1099
```

- cet objet fait la correspondance entre nom et instance de Stub enregistré par le(s) serveur(s) avec Naming.bind()

L 'exécution Coté Serveur
L 'accès au serveur de liaison

La classe Naming

- encaspule le dialogue avec plusieurs objets serveur de liaison
- URL de liaison

```
rmi://hostreg:2001/Hello/World
rmi://:2001/Hello/World
//:2001/Hello/World
/Hello/World
```

- Méthodes Statiques
  - **bind(String url, Remote r), rebind(String url, Remote r), unbind(String url)**

    enregistre/désenregistre un objet auprès du serveur
  - **Remote lookup(String url)**

    retourne un stub
  - **String[] list()**

    liste les noms enregistrés

L 'exécution Coté Serveur
L implantation du serveur de liaison

- Les objets serveur de liaison
  - réalise la correspondance nom avec stub
  - Interface d 'un objet serveur de liaison
    - **sun.rmi.registry.Registry (extends java.rmi.Remote)**
  - Implémentation par défaut de l 'objet serveur de liaison
    - **sun.rmi.registry.RegistryImpl**
  - Méthodes non statiques
    - **bind(),rebind(),unbind(),lookup(),list()**
- La classe LocateRegistry
  - localise ou active un objet serveur de liaison
  - Méthodes Statiques
    - **Registry createRegistry(int port)**

      crée un objet serveur de liaison sur le port spécifié
    - **Registry getRegistry(int port)**

      récupère l 'objet serveur de liaison qui a été crée

L 'exécution Coté Serveur

Le serveur d 'objets distants

- *le Stub doit être dans le CLASSPATH ou chargeable via FS ou HTTP*
- *hello.policy autorise l 'usage du accept et du connect sur les Sockets*

```
hostser> java
    -Djava.security.policy=./hello.policy
    -Djava.rmi.server.codebase=http://hostwww/hello/myclasses/
    examples.hello.HelloServer hostreg:1099
```

- *Mais aussi*

```
-Djava.rmi.server.codebase=file://dev/hello/myclasses/ Unix
-Djava.rmi.server.codebase=file:/c:\dev\hello\myclasses/          Win32
```

- *./hello.policy*

```
grant { permission java.net.SocketPermission "*:1024-65535","connect,accept";
        permission java.net.SocketPermission "*:80", "connect";
        // permission java.security.AllPermission; /* ou autorise tout */ };
```

L ’exécution Coté Client

L ’application

- *le Stub doit être dans le CLASSPATH ou chargeable via FS ou HTTP*
- *hello.policy autorise l ’usage du connect sur les Sockets*

```
hostcli> java
    -Djava.security.policy=./client.policy
    -Djava.rmi.server.codebase=http://hostwww/hello/myclasses/
    examples.client.HelloClient hostreg:1099
```

- *./client.policy*

```
grant { permission java.net.SocketPermission "*:1024-65535","connect";
        permission java.net.SocketPermission "*:80", "connect"; };
```

L ’exécution Coté Client

L ’applet

- *l ’élément HTML applet doit spécifier le codebase*

```
<HTML><title>Hello World</title><center> <h1>Hello World</h1> </center>
The message from the HelloServer is:<br>
<applet codebase="myclasses/"
        code="examples.client.HelloApplet"  width=500 height=120>
</applet> </HTML>
```

- *seuls les sockets vers hostwww sont autorisés par la sandbox*
  - *donc hostreg = hostser = hostwww*

```
hostcli> appletviewer http://hostwww/hello/hello.html
```

Le passage de paramètres

- Les paramêtres des méthodes invoquées sur un objet distant sont soit:
  - une valeur de type Primitif
    - La valeur est passée
  - un objet d 'une classe qui implemente l 'interface Serialisable (ou l 'interface Externalizable)
    - l 'objet est sérialisé et envoyé à l 'objet distant : le paramétre est alors déserialisé pour l 'objet d istant l 'utilse
  - un objet d 'une classe qui implémente l 'interface Remote
    - c 'est l 'objet Stub qui est sérialisé et envoyé à l 'objet distant : quand l 'objet distant invoque un méthode sur le paramêtre, l 'invocation est distante.
- Une exception est levée
  si un paramêtre ne rentre pas dans ces trois cas.

Le passage de paramètres

- Un exemple plus complexe

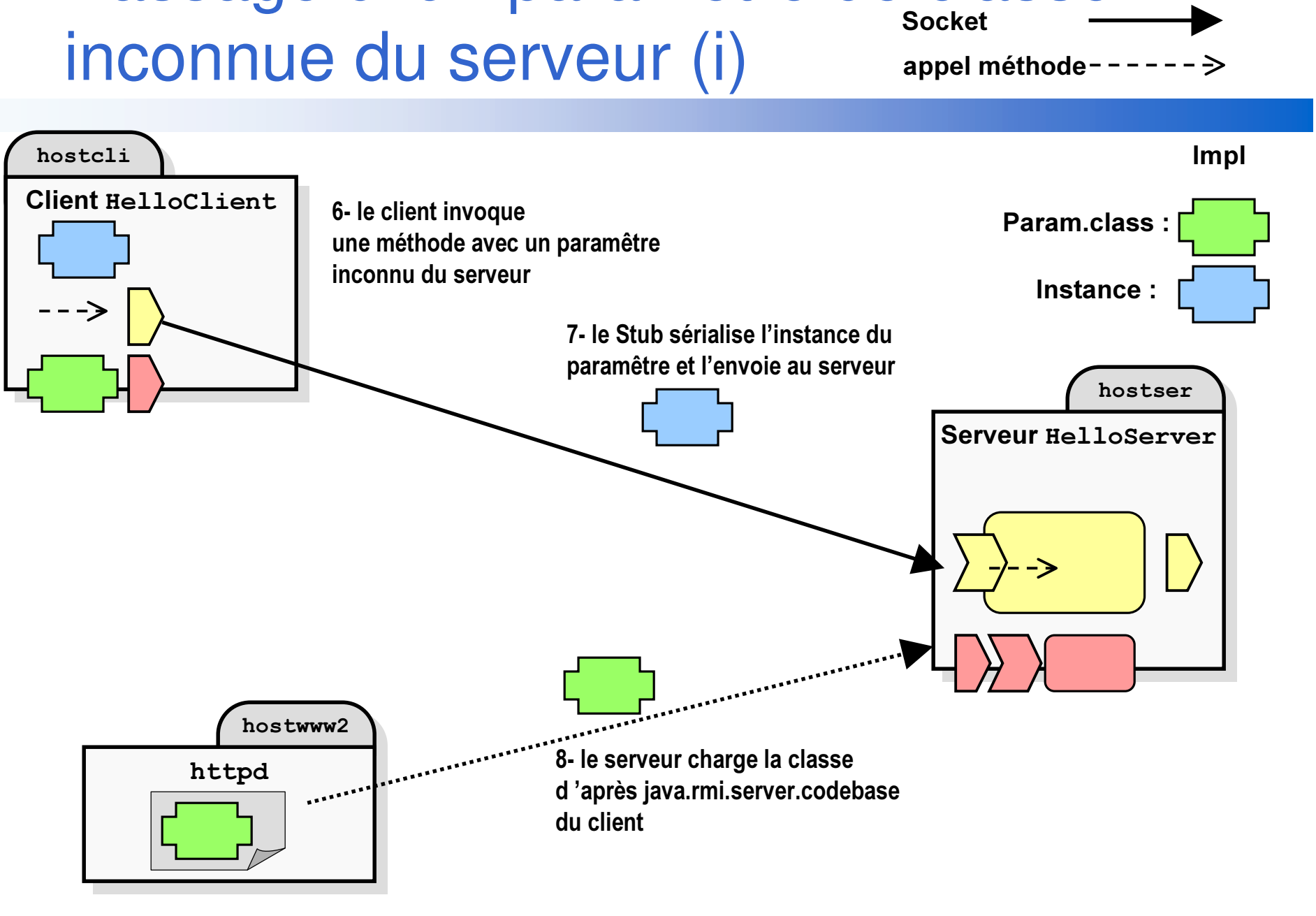
```
package examples.order;
public class OrderLine implements java.io.Serializable { // une classe sérialisable
    public String productname;
    public int quantity;
}

package examples.order;
import java.util.Vector; // une classe sérialisable
public interface Order extends java.rmi.Remote { // l 'interface distante
   public float price(String productname, int quantity) throws java.rmi.RemoteException;
   public float price(Vector orderlines) throws java.rmi.RemoteException;
   public float price(OrderLine[] orderlines) throws java.rmi.RemoteException;
   public Vector priceInDetail(Vector orderlines) throws java.rmi.RemoteException;
   public Vector priceInDetail(OrderLine[] orderlines) throws java.rmi.RemoteException;
}
```

Passage d 'un paramètre de classe inconnue du serveur

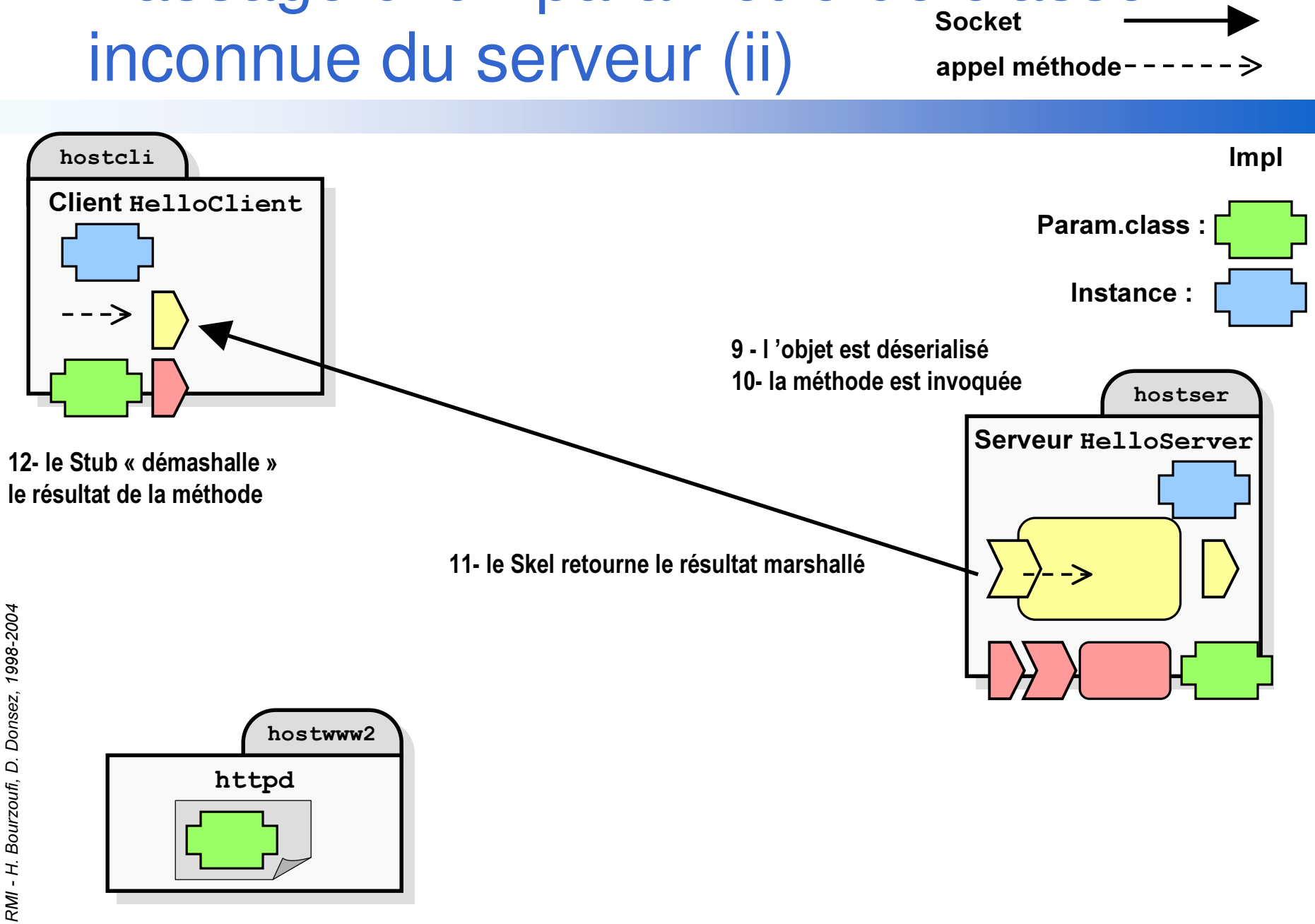
- La sous classe d'un paramètre est inconnue du serveur
  - le serveur charge la classe du sous-type inconnu
    à partir du java.rmi.server.codebase du client à défaut du CLASSPATH

- Voir : jdk1.2.2/docs/guide/rmi/codebase.htm

Passage d 'un paramètre de classe inconnue du serveur (i)



Passage d 'un paramètre de classe inconnue du serveur (ii)



Passage d 'un paramêtre de classe inconnue du serveur - Exemple

```
package examples.bank;
// l 'interface du 1er paramêtre de la méthode credit()
public interface Account extends Serializable { float getBalance(); void setBalance(float amount); }
// l 'interface de l 'objet distant
public interface Bank implements Remote { void credit(Account acc, float amount); }

// une sous-classe de Account inconnue du serveur
public class CheckingAccount implements examples.bank.Account {
    private String name; private float balance;
    public CheckingAccount(String name, float initamount) {
            this.name=name; this.balance= initamount; }
    public float getBalance(return balance;)
    public void setBalance(float amount) { balance=amount; }
}
// un client
examples.bank.Bank mybank = (Bank) java.rmi.Naming.lookup("//hostreg/MyBank");
examples.bank.Account accjohn = new CheckingAccount("john",1000.0);
mybank.credit(accjohn,2000.0);
```

Passage d ’un Stub en paramêtre

Méthode 1

- Le Stub est celui d ’un objet distant déjà servi

Méthode 2 : Objet Distant Temporaire

- But:
  - L ’objet distant est local au client et n ’est utilisé que pour la durée d ’une ou plusieurs invocations distantes depuis le client
- Solution
  - exporter l ’objet avec la méthode UnicastRemoteObject.exportObjet()
  - l ’objet doit implémenter l ’interface Unreferenced pour être ramassé par le GC

Objet Distant Temporaire
Le client

```
public class AsyncHelloClient {
    public static void main( String[] args) {
        Hello hello = (Hello) Naming. lookup("//hostreg/HelloObject");
        Person person = new PersonImpl("Didier","Donsez");
        UnicastRemoteObject.exportObject(person);
        String hellomessage = hello.sayHello(person, Hello.EN);
        person = null; // force le GC local et le GC reparti
        System.println.out(hellomessage);
        ...
    }
}
```

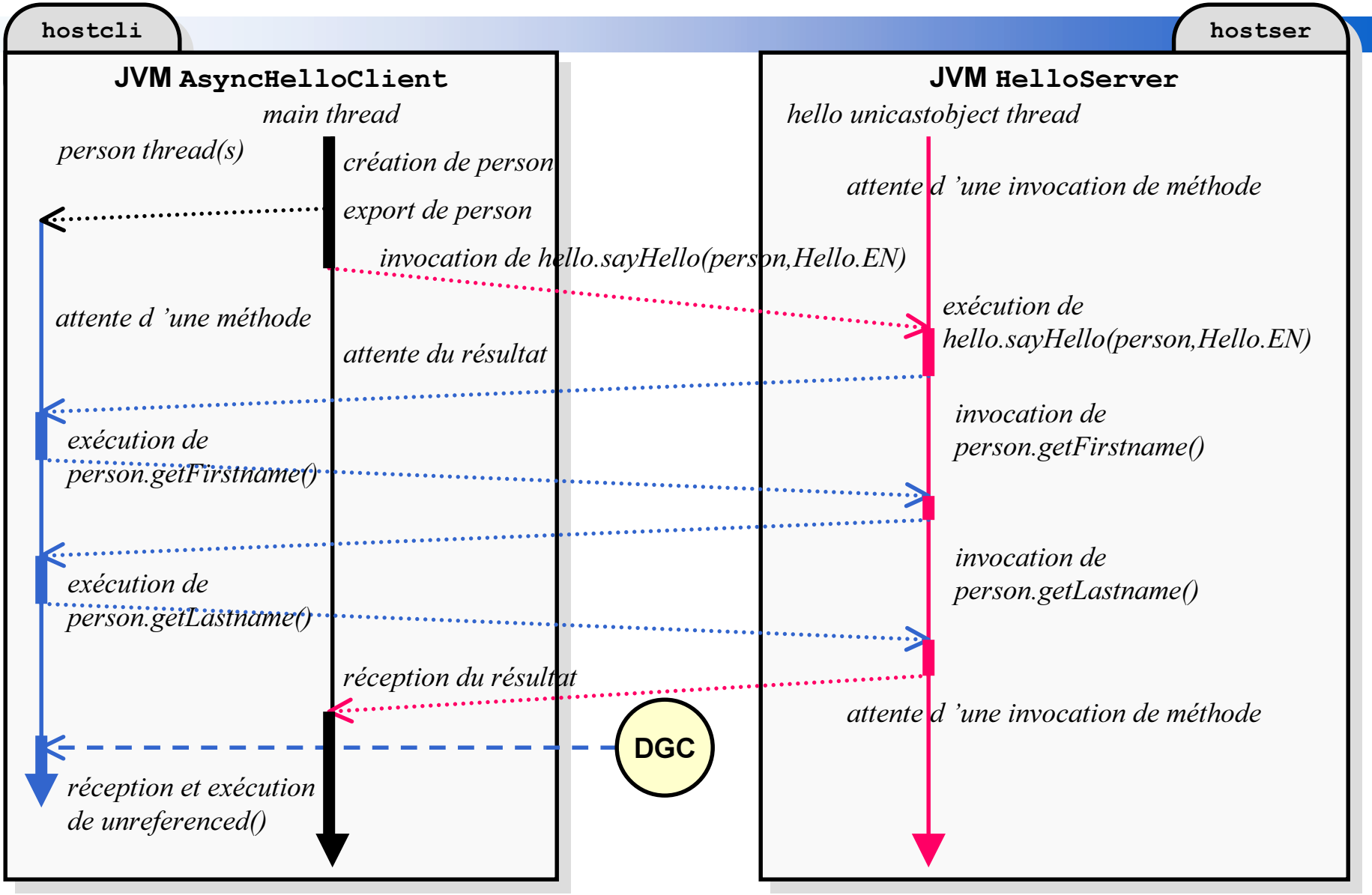
Objet Distant Temporaire
Le serveur temporaire

```
public interface Person extends Remote {
    String getFirstname() throws RemoteException;
    String getLastname() throws RemoteException;
}
public class PersonImpl implements Person, Unreferenced {
    private String lastname; private String firstname;
    PersonImpl(String lastname, String firstname) {
        this.lastname=lastname,this.firstname=firstname;
    }
    String getFirstname() throws RemoteException { return firstname; }
    String getLastname() throws RemoteException { return lastname; }
    void unreferenced() {
        ThreadGroup tg = Thread.currentThread().getThreadGroup();
        tg. stop()
    }
}
```

Objet Distant Temporaire
L 'objet distant et son serveur

```
public interface Hello extends Remote {
    String sayHello(Person person, int lang) throws RemoteException;
    String sayHello(String name, int lang) throws RemoteException;
}
public class HelloImpl extends java.rmi.server.UnicastRemoteObject implements Hello {
    String sayHello(Person person, int lang) throws RemoteException {
            String firstname= person.getFirstname();     // invocation distante sur le Stub
            String lastname= person.getLastname();       // invocation distante sur le Stub
            return sayHello(firstname + " " + lastname, lang);
    } ...}
public class HelloServer {
    public static void main(String args[]) {
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new java.rmi.RMISecurityManager()); }
    try {
            HelloImpl obj = new HelloImpl(Hello.EN);
            java.rmi.Naming.rebind("//hostreg/HelloObject", obj);
    } catch (Exception e) { e.printStackTrace();  } } }
```

Objet Distant Temporaire
Déroulement de l 'exécution



L ’activation d ’objets distants

- Rappel (JDK1.1)
  - l ’objet distant est actif au démarrage du serveur RMI
- Motivation
  - principe du démon `inetd` d ’Unix
    - Le démon `rmid` démarre une JVM qui sert l ’objet distant seulement au moment de l ’invocation d ’une méthode (à la demande) ou au reboot.
- JDK1.2 introduit
  - un nouveau package `java.rmi.activation`
    - un objet activable doit dériver de la classe `Activatable`
  - un démon RMI `rmid`
    - qui active les objets à la demande ou au reboot de la machine
- Info : jdk1.2.2\docs\guide\rmi\activation.html
- Remarque : l’activition est très utilisée par JINI !

Créer une implémentation activable

- La classe doit étendre **java.rmi.activation.Activable** et implémenter l 'interface distante
- La classe doit déclarer un constructeur avec 2 arguments
  java.rmi.activation.ActivationID, java.rmi.MarshalledObject

```
public class HelloActivatableImpl
      extends java.rmi.activation.Activatable implements Hello {
   private int defaultlang;
   public ActivatableImplementation(ActivationID id, MarshalledObject data)
     throws java.rmi.RemoteException {
                super(id, 0);
                this.defaultlang=((Integer)data.get()).intValue();
    }
     // implémentation des méthodes
    public String sayHello() throws java.rmi.RemoteException { ... } ...
}
```

Créer le programme d 'enregistrement (Setup)

- Rôle
  - passer l 'information nécessaire à l 'activation de l 'objet activable au démon **rmid**
    puis enregistrer l 'objet auprès de **rmiregistry**
- Descripteur
  - **ActivationDesc** : description des informations nécessaires à `rmid`
- Groupes d'objets activables
  - `rmid` active une JVM par groupe
  - Les objets du même groupe partagent la même JVM
    - **ActivationGroup** : représente un groupe d 'objets activables
    - **ActivationGroupDesc** : description d 'un groupe
    - **ActivationGroupID** : identifiant d 'un groupe

Créer le programme d 'enregistrement *Exemple (partie 1)*

```
import java.rmi.*; import java.rmi.activation.*; import java.util.Properties;
public class SetupActivHello {
    public static void main(String[] args) throws Exception {
     System.setSecurityManager(new RMISecurityManager());

// création d 'un groupe d 'objets activables
     Properties props = new Properties();  props.put("java.security.policy", "./helloactiv.policy");
     ActivationGroupDesc.CommandEnvironment ace = null;

// descripteur du groupe
     ActivationGroupDesc agroupdesc = new ActivationGroupDesc(props, ace);
//
     ActivationSystem asystem = ActivationGroup.getSystem();
     ActivationGroupID agi = asystem.registerGroup(agroupdesc);
// enregistrement du groupe
     ActivationGroup.createGroup(agi, agroupdesc , 0);
```

Créer le programme d 'enregistrement *Exemple (partie 2)*

```
// le descripteur doit contenir le codebase pour chercher les classes
    String classeslocation = "http:/hostwww/hello/myclasses/";
// le descripteur peut contenir un objet sérialisable pour l 'initialisation de l 'objet ; data peut
    être null

    MarshalledObject data = new MarshalledObject (new Integer(Home.ES));
// création d 'un descripteur pour l 'objet activable
    ActivationDesc adesc = new ActivationDesc
       (agi, "examples.hello.HelloActivatableImpl", classeslocation, data );


// enregistrement de l 'objet auprès du démon rmid : récupération d 'un stub
    Hello obj = (Hello)Activatable.register(adesc);
// enregistrement du stub auprès du rmiregistry
    Naming.rebind("//hostreg/HelloActiv", obj);
    System.exit(0);
  }
}
```

Personnaliser la couche Transport des RMI

Rappel

- Par défaut, TCP est utilisé par la couche de transport RMISocketFactory (classes java.net.Socket et java.net.SocketServer)
  - cependant dans les cas de Firewall/proxies, elle invoque les méthodes en utilisant la méthode POST de HTTP.
    - La propriété java.rmi.server.disableHttp=true désactive le tunneling HTTP

Motivation

- utiliser d 'autres classes (que Socket et SocketServer) basées sur
  - TCP pour de la compression, du chiffrage, …
    - Remarque : RMI over SSL utilise cette technique (voir l 'exemple)
  - ou non (UDP)

Info : jdk1.2.2\docs\guide\rmi\rmisocketfactory.doc.html

Personnaliser la couche Transport des RMI

- Etape 1

  Écrire 2 sous classes de

  java.rmi.RMIClientSocketFactory,

  java.rmi. RMIServerSocketFactory

  qui utilisent 2 autres classes de transport que

  java.net.Socket

  java.net.SocketServer

  (voir cours Socket)

Personnaliser la couche Transport des RMI

```
package examples.rmisocfac;
import java.io.*; import java.net.*; import java.rmi.server.*;

public class CompressionClientSocketFactory
        implements RMIClientSocketFactory, Serializable {
   public Socket createSocket(String host, int port) throws IOException {
      return new CompressionSocket(host, port);
   } }

public class CompressionServerSocketFactory
      implements RMIServerSocketFactory, Serializable {
   public ServerSocket createServerSocket(int port) throws IOException {
      return new CompressionServerSocket(port);
   } }
```

Personnaliser la couche Transport des RMI

- Etape 2

  Spécifier les factories dans le constructeur de l 'objet distant qui hérite de la classe **UnicastRemoteObject**

```
protected UnicastRemoteObject( int port,
                               RMIClientSocketFactory csf,
                               RMIServerSocketFactory ssf)
```

- Exemple:

```
public AccountImpl() throws RemoteException {
   super( 0,          new CompressionClientSocketFactory(),
                      new CompressionServerSocketFactory()
   );
   ... }
```

RMI et SSL

- Couche de transport custom
  - SSLClientSocketFactory, SSLServerSocketFactory
  - utilisent SSLSocket, SSLServerSocket (www.phaos.com)

- voir
  - jdk1.2.2\docs\guide\rmi\SSLInfo.html
  - jdk1.2.2\docs\guide\rmi\PhaosExample.html

RMI over SSL

RMIClientSocketFactory et RMIServerSocketFactory

```
package examples.rmissl;
import java.io.*; import java.net.*; import java.rmi.server.*;
import crypsec.SSL.*; // importe des classes de SSL (www.phaos.com)

public class SSLClientSocketFactory implements RMIClientSocketFactory, Serializable {
    public Socket createSocket(String host, int port) throws IOException {
        return ((Socket) new SSLSocket(host, port, new SSLParams()));
} }

public class SSLServerSocketFactory implements RMIServerSocketFactory, Serializable {
    transient protected SSLParams params;
    public SSLServerSocketFactory() { this(new SSLParams()); }
    public SSLServerSocketFactory(SSLParams p) { params = p; }
    public ServerSocket createServerSocket(int port) throws IOException {
        return new SSLServerSocket(port);
  } }
```

RMI over SSL
l 'objet distant

```
package examples.rmissl;
import java.io.*; import java.net.*; import java.rmi.*; import java.rmi.server.*;
public class SSLHelloImpl extends UnicastRemoteObject implements Hello {
  private int defaultlang ;
  public SSLHelloImpl(SSLServerSocketFactory ssf, int defaultlang)
                throws RemoteException {
     super(0, new SSLClientSocketFactory(), ssf);
     this.defaultlang = defaultlang ;
  }

  public String sayHello() throws RemoteException { ... }
  ...
}
```

RMI over SSL
le serveur

```
package examples.rmissl;
import java.io.*; import java.net.*; import java.rmi.*; import java.rmi.server.*;
public class SSLHelloServer {
   public static void main(String args[]) {  try {
// initialize server certificate
      SSLCertificate cert = new SSLCertificate();
      cert.certificateList = new Vector();
      cert.certificateList.addElement(new X509(new File("server-cert.der")));
      cert.certificateList.addElement(new X509(new File("ca-cert.der")));
      SSLParams params = new SSLParams(); // initialize SSL context object
      params.setServerCert(cert);
      params.setRequestClientCert(true); // require client authentication
      System.setSecurityManager(new RMISecurityManager());
// secure server socket factory to use in remote objects
      SSLServerSocketFactory ssf = new SSLServerSocketFactory(params);
// create a secure rmiregistry
      Registry registry = LocateRegistry.createRegistry(1099,new SSLClientSocketFactory(), ssf);
// create a remote object that will use a secure client/server socket pair
      SSLHelloImpl o = new SSLHelloImpl(ssf, Hello.FR);
      registry.rebind("/SSLHelloObject", o);
     } catch (Exception e) { System.err.println(e.getMessage()); e.printStackTrace(); }
 } }
```

RMI over SSL
le client

```
package examples.rmissl;
import java.io.*; import java.net.*; import java.rmi.*; import java.rmi.server.*;
public class SSLHelloClient {
   public static void main(String args[]) {
     try {
        if (args.length < 1) { System.err.println("Usage: <hostName>"); System.exit(1); }
        System.setSecurityManager(new RMISecurityManager());

 // Create a secure rmiregistry
        Registry registry = LocateRegistry.
             getRegistry(args[0], 1099, new SSLClientSocketFactory());

// Obtain a reference to the HelloObject
        Hello hello = (Hello) registry.lookup("/SSLHelloObject");
        System.out.println("Message: " + hello.sayHello());
     } catch (Exception e) {
        e.printStackTrace();
     }
   }
}
```

Le glaneur de cellules distibué des RMI
`java.rmi.dgc`

- Motivation pour un GC réparti
  - ramasser les objets distants
    qui ne sont plus référencés (i.e. plus de stub sur des clients)
- Principe
  - basé sur le comptage de références
  - interagit avec les GCs locaux de toutes les JVM
    - maintient des *weaks references* pour éviter le ramassage par le GC local
- Remarque:
  - en cas de partition du réseau, les objets peuvent être prématurément ramassés

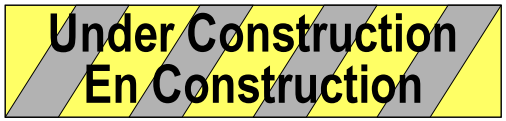
Le glaneur de cellules distibué des RMI
java.rmi.dgc

- Interface java.rmi.server.Unreferenced

```
package java.rmi.server;
public interface Unreferenced {     public void unreferenced();    }
```

- Un objet distant peut être notifié quand il n 'est plus référencé
- il doit implémenter l 'interface **java.rmi.server.Unreferenced**
- la méthode void **unreferenced()** est appelée par le DGC quand il n 'y a plus de référence.

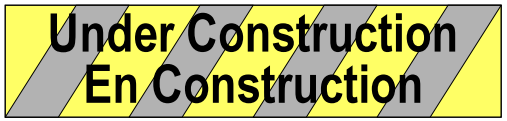
Détail d 'implantation



Coté serveur

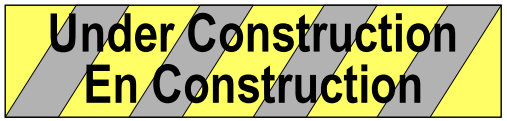
- Une thread par appel d 'un client
  - + la thread d 'écoute des nouveaux appels sur le socket
- L 'appel concurrent aux méthodes n 'est pas synchronisé
  - La synchronisation est à la charge du développeur
- Un couple <hôte,port> pour tous les objets par une JVM

RMI over IIOP



- Couche de transport : IIOP (www.omg.org)

RMI et EJB



RMI et JINI

JINI

- mécanisme de courtage distribué (*discovery-join-lookup*)
  - Courtage tolérant aux pannes (redondance)
  - Recherche par rapport

    à l'interface du service (nom+signature méthodes)

    à des attributs qualifiants (Entry)

    à des groupes
- Le client récupère la liste des objets distants (stub RMI) potentiellement actifs ou activables offrant ce service

JERI : JINI Extensible Remote Invocation

- Rendre extensible les couches transfort et sérialisation des RMI
  - http://pandonia.canberra.edu.au/java/jini/tutorial/Jeri.html

Ajout du J2SE 1.4

- java.rmi.server.RMIClassLoaderSpi
  - Délégation du chargement dynamique des classes
- Support du POA (Portable Object Adapter) pour rmic
  - rmic -iiop -poa …
- Génération d'IDL CORBA
  - Pour l'interopérabilité avec des serveurs CORBA via IIOP
  - rmic -idl …

Ajout du J2SE 1.5

- Nouveautés dans le langage Java : les annotations
  - Avant 1.5

```
public interface IHello extends Remote {
    public String sayHello(String name) throws RemoteException;
}
public class Hello implements IHello {
    public String sayHello(String name) throws RemoteException {
      return "Hello "+name;
}}
```

  - Avec les annotations du 1.5

```
public class Hello { // with 1.5 annotations
   public @remote String sayHello(String name) {
      return "Hello "+name;
}}
```

- Impact sur les outils de génération comme rmic

Portable Interceptor & RMI

- TODO

- Voir CAROL d'ObjectWeb

Comparaison

	RPC	RMI	CORBA	.NET Remoting	SOAP
Qui	SUN/OSF	SUN	OMG	MicroSoft/ECMA	W3C
Plate-formes	Multi	Multi	Multi	Win32, FreeBSD, Linux	Multi
Langages de Programmation	C, C++, …	Java	Multi	C#, VB, J#, …	Multi
Langages de Définition de Service	RPCGEN	Java	IDL	CLR	XML
Réseau	TCP, UDP	TCP, HTTP, IIOP customisable	GIOP, IIOP, Pluggable Transport Layer	TCP,HTTP, *IIOP*	RPC,HTTP
Marshalling		Sérialisation Java	Représentation IIOP	Formatteurs Binaire, SOAP	SOAP
Nommage	IP+Port	RMI, JNDI,JINI	CosNaming	IP+Nom	IP+Port, URL
Intercepteur	Non	depuis 1.4	Oui	Oui CallContext	Extension applicative dans le header
Extra		Chargement dynamique des classes	Services Communs Services Sectoriels	Pas de Chargement dynamique des classes	

Bibliographie

- Spécification des RMI
  - java.sun.com et jdk1.2.2/docs/guide/rmi/spec
- Guide de la doc du JDK
  - jdk1.2.2/docs/guide/rmi
- Tutorial Java
  - Trail sur les RMI et sur JINI (Objet activables)
  - Online Training
    - http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/rmi.html
- Autres
  - Jeremie : une implémentation OpenSource des RMI
    - http://www.objectweb.org

Bibliographie

- Gilles Roussel, Étienne Duris, "Java et Internet, Concepts et programmation", Ed Vuibert, 01/2000, ISBN : 2-7117-8654-4
  - le chapitre 11 détaille bien les RMI du JDK1.2 et notamment les objets activables
- Elliotte Rusty Harold, "Programmation Réseau avec Java", Ed O Reilly, 1997, ISBN 2-84177-034-6
  - date un peu
- Java Distributed Computing, Ed Oreilly existe en français
  - date un peu
- Robert Orfali, Dan Harkey, “ Client/Server Programming with Java and Corba ”, 2ème édition, 1998, Ed Wiley, ISBN 0-471-24578-X.
  - survol rapide
  - comparaison intéressante avec les autres techniques (DCOM, CORBA, …)