Les Systèmes et Applications Réparties et leur Programmation

Samia Bouzefrane
Maître de Conférences
Laboratoire CEDRIC
Conservatoire National des Arts et Métiers
292 rue Saint Martin
75141 Paris Cédex 03

samia.bouzefrane@cnam.fr
http://cedric.cnam.fr/~bouzefra

Notion d'Intergiciel (Middleware)

Intergiciel : une terminologie pour une classe de logiciels systèmes qui permet d'implanter une approche répartie: exemple CORBA

- Fournit une API d'interactions de communication: interactions de haut niveau pour des applications réparties: exemple invocation distante de méthode pour des codes objets.
- Fournit un ensemble de services utiles pour des applications s'exécutant en environnement réparti: désignation, cycle de vie, sécurité, transactionnel, etc.
- Fonctionne en univers ouvert (supporte des applications tournant sur des plate-formes matérielles et logicielles différentes).

Les principales catégories d'Intergiciels

- Intergiciels à **messages**
  - **MOM**: **Message Oriented Middleware** IBM MQSeries, Microsoft Message Queues Server,DECmessageQ, etc.

- Intergiciels de **bases de données** (ODBC)

- Intergiciels à **appel de procédure distante** (DCE)

- Intergiciels à **objets répartis** (CORBA, JAVA RMI)

- Intergiciels à **composants** (EJB, CCM, Web services)

Intergiciels à Objets Répartis :

Java Remote Method Invocation (RMI)
ou les invocations de méthodes Java distantes

Rappel : Appel local (interface et objet)

```
public interface ReverseInterface {
String reverseString(String chaine);
}


public class Reverse implements ReverseInterface
{
public String reverseString (String ChaineOrigine){

int longueur=ChaineOrigine.length();
StringBuffer temp=new StringBuffer(longueur);
 for (int i=longueur; i>0; i--) {
   temp.append(ChaineOrigine.substring(i-1, i));}
 return temp.toString();
 }
}
```

Rappel : Appel local (programme appelant)

```
import ReverseInterface;

public class ReverseClient
{
    public static void main (String [] args)
    {  Reverse rev = new Reverse();
         String result = rev.reverseString (args [0]);
         System.out.println ("L'inverse de "+args[0]+" est
"+result);

    }
}

$javac *.java
$java ReverseClient Alice
L’inverse de Alice est ecilA
$
```

Qu’attend t-on d’un objet distribué ?

Un objet distribué doit pouvoir être vu comme un objet « normal ».

Soit la déclaration suivante :

```
ObjetDistribue monObjetDistribue;
```

• On doit pouvoir invoquer une méthode de cet objet situé sur une autre machine de la même façon qu’un objet local :

```
monObjetDisribue.uneMethodeDeLOD();
```

• On doit pouvoir utiliser cet objet distribué sans connaître sa localisation. On utilise pour cela un service sorte d'annuaire, qui doit nous renvoyer son adresse.

```
monObjetDistribue=
ServiceDeNoms.recherche('myDistributedObject');
```

• On doit pouvoir utiliser un objet distribué comme paramètre d'une méthode locale ou distante.

```
x=monObjetLocal.uneMethodeDeLOL(monObjetDistribue);

x= monObjetDistribue.uneMethodeDeLOD(autreObjetDistribue);
```

• On doit pouvoir récupérer le résultat d'un appel de méthode sous la forme d'un objet distribué.

```
monObjetDistribue=autreObjetDistribue.uneMethodeDeLOD();

monObjetDistribue=monObjetLocal.uneMethodeDeLOL();
```

Java RMI
Remote Method Invocation

permet la communication entre machines virtuelles Java (JVM)
qui peuvent se trouver physiquement sur la même machine
ou sur deux machines distinctes.

Présentation

RMI est un système d'objets distribués constitué uniquement d'objets java ;

- RMI est une **A**pplication **P**rogramming **I**nterface (intégrée au JDK 1.1 et plus) ;

- Développé par JavaSoft ;

• Mécanisme qui permet l'appel de méthodes entre objets Java qui s'exécutent éventuellement sur des JVM distinctes ;

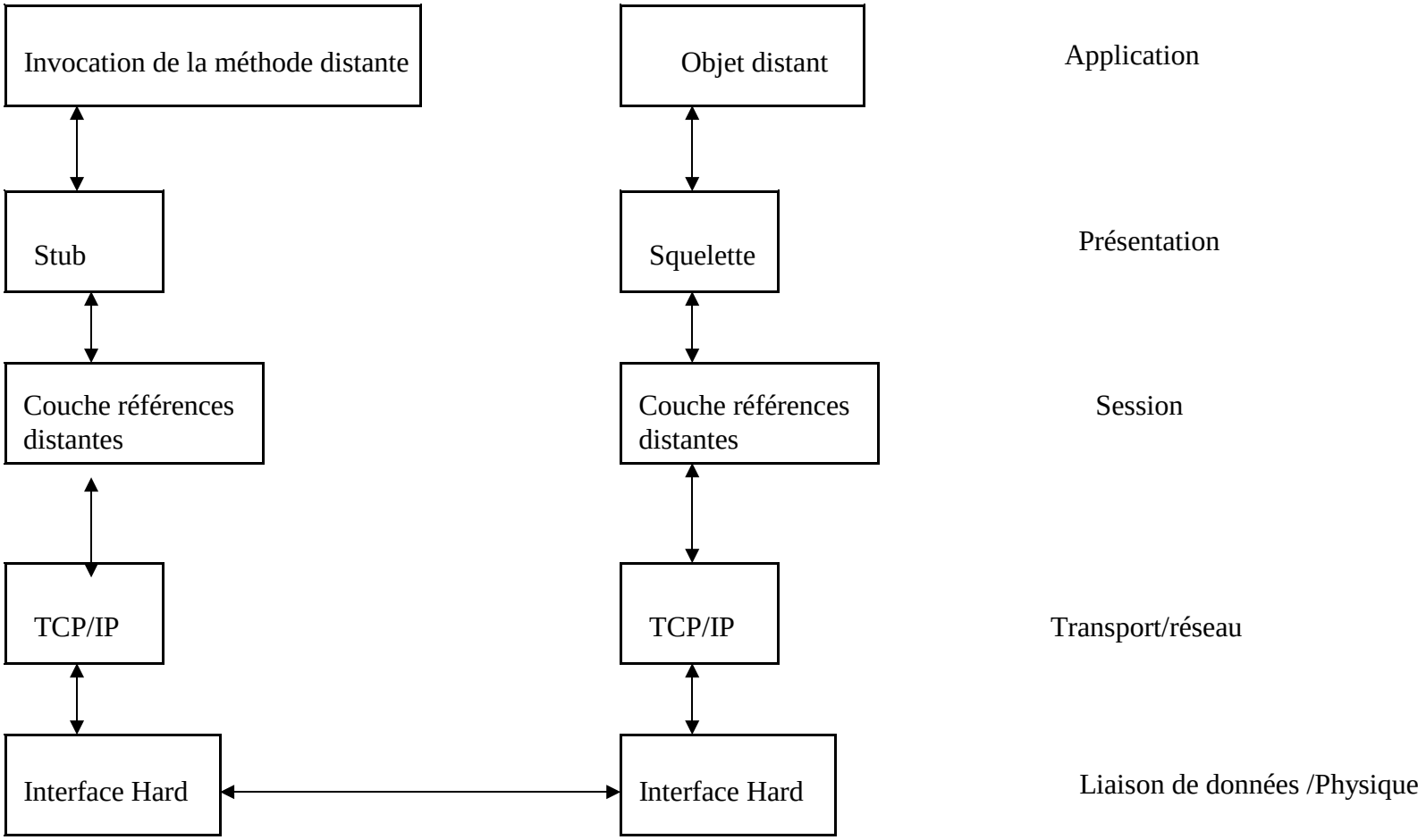
• L 'appel peut se faire sur la même machine ou bien sur des machines connectées sur un réseau ;

• Utilise les sockets ;

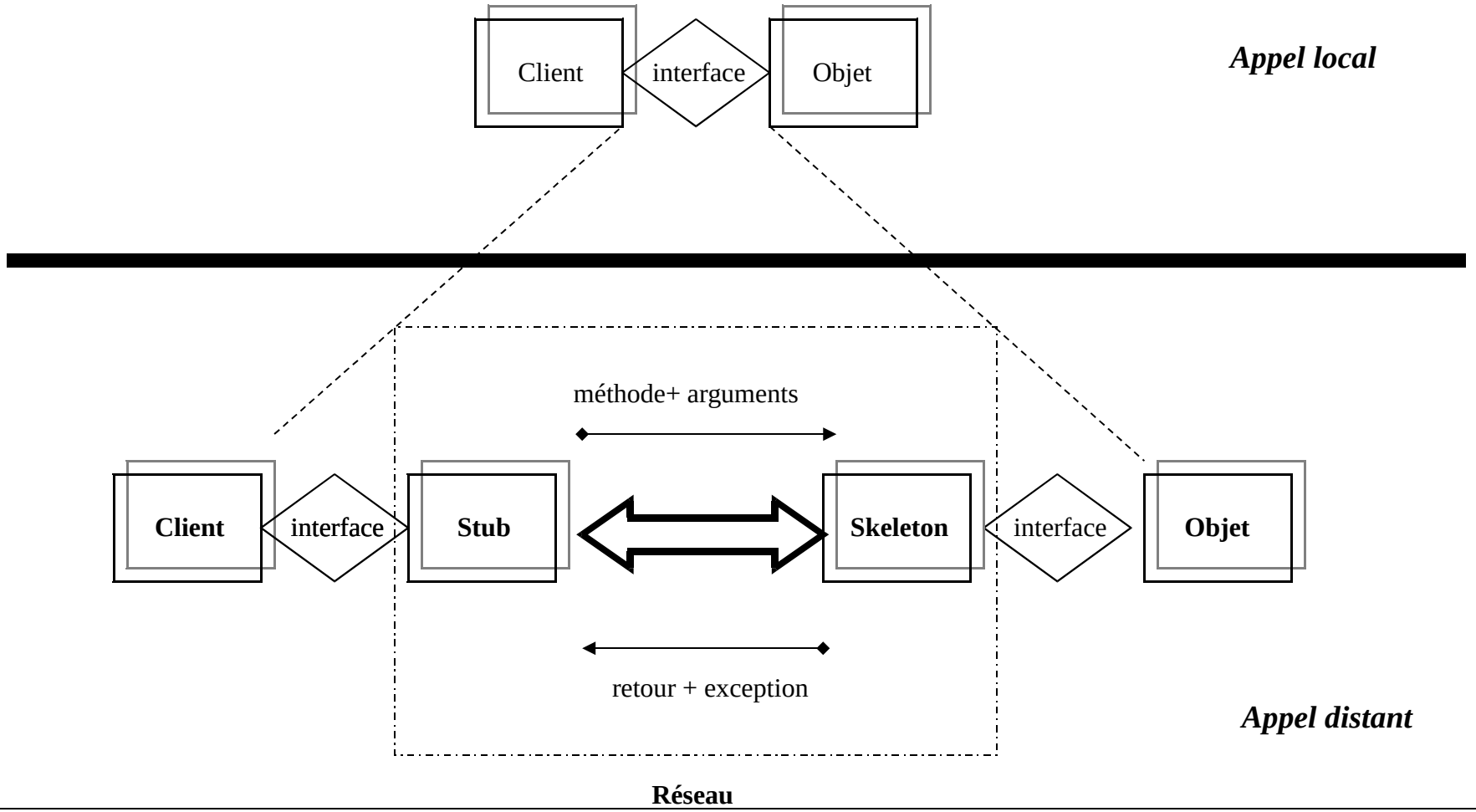
• Les échanges respectent un protocole propriétaire : **R**emote **M**ethod **P**rotocol ;

• RMI repose sur les classes de sérialisation.

Architecture



Appel local versus Appel à distance



Les amorces (Stub/Skeleton)

• Elles assurent le rôle d'adaptateurs pour le transport des appels distants

• Elles réalisent les appels sur la couche réseau

• Elles réalisent l'assemblage et le désassemblage des paramètres (*marshalling*, *unmarshalling*)

• Une référence d'objets distribué correspond à une référence d'amorce

• Les amorces sont créées par le générateur **`rmic`**.

Les Stubs

- Représentants locaux de l'objet distribué ;

- Initient une connexion avec la JVM distante en transmettant l'invocation distante à la couche des références d'objets ;

- Assemblent les paramètres pour leur transfert à la JVM distante ;

- Attendent les résultats de l'invocation distante ;

- Désassemblent la valeur ou l'exception renvoyée ;

- Renvoient la valeur à l'appelant ;

- S'appuient sur la sérialisation.

Les squelettes

- Désassemblent les paramètres pour la méthode distante ;
- Font appel à la méthode demandée ;
- Assemblage du résultat (valeur renvoyée ou exception) à destination de l'appelant.

La couche des références d'objets
Remote Reference Layer

• Permet d'obtenir une référence d'objet distribué à partir de la référence locale au stub ;

• Cette fonction est assurée grâce à un service de noms **`rmiregister`** (qui possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants) ;

• Un unique **`rmiregister`** par JVM ;

• **`rmiregister`** s'exécute sur chaque machine hébergeant des objets distants ;

• **`rmiregister`** accepte des demandes de service sur le port 1099;

La couche transport

- réalise les connexions réseau basées sur les flux entre les JVM

- emploie un protocole de communication propriétaire (**JRMP**: Java Remote Method Invocation) basé sur TCP/IP

- Le protocole JRMP a été modifié afin de supprimer la nécessité des squelettes car depuis la version 1.2 de Java, une même classe skeleton générique est partagée par tous les objets distants.

Etapes d'un appel de méthode distante