

Cours : SOA et services Web

Chapitre 1: Evolution des systèmes d'information

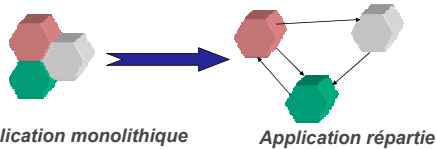
Présenté par : Sahbi BAHROUN

Qu'est ce qu'une application répartie ?

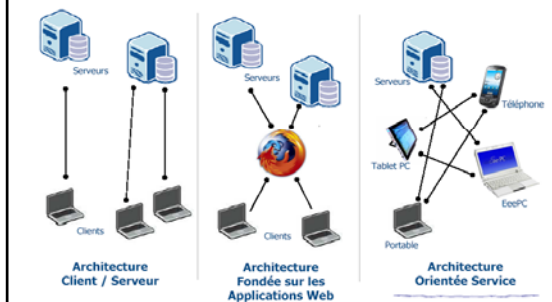
- Il s'agit d'une application découpée en plusieurs unités
 - Chaque unité peut être placée sur **une machine différente**
 - Chaque unité peut s'exécuter sur **un système différent**
 - Chaque unité peut être programmée dans **un langage différent**

Construction d'une application répartie

- **Identifier** les éléments fonctionnels de l'application pour les **regrouper** au sein d'unités
- **Estimer les interactions** entre unités
- Définir le **schéma d'organisation** de l'application



Evolution des applications réparties



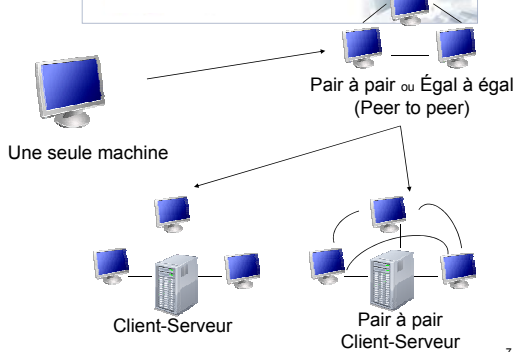
Avantages du réparti

- **Organisationnel**
 - Décentraliser les responsabilités
 - Découpage en unité
- **Fiabilité et disponibilité**
 - Individualisation des défaillances
 - Duplication des constituants de l'application
- **Performance**
 - Partage de la charge
- **Maintenance et évolution**

Inconvénients du réparti

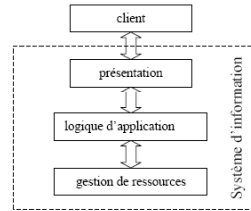
- **Une mise en œuvre plus délicate**
 - Gestion des erreurs
 - Suivi des exécutions
- **Pas de vision globale instantanée**
 - Délais des transmissions
- **Administration plus lourde**
 - Installation
 - Configuration
 - Surveillance
- **Coût**
 - Formation
 - Achat des environnements

Les SI: de la distribution à Internet



7

Conception d'un SI



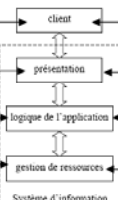
SOA et services web: 3^{ème} SIL

8

Conception d'un SI

Conception Top-down

1. définition du canal d'accès et de la plate-forme client.
2. définition de formats de présentation et de protocoles pour les clients sélectionnés.
3. définition des fonctionnalités nécessaires pour fournir le contenu à la couche de présentation.
4. définition de la source et de l'organisation des données nécessaires pour implémenter la logique de l'application.



Conception Bottom-up

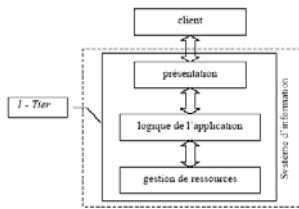
1. définition du canal d'accès et de la plate-forme client.
2. analyse des ressources existantes et de la fonctionnalité à fournir.
3. création d'une couche pour les ressources existantes travers une interface consistante.
4. adaptation du résultat de la logique de l'application de sorte qu'elle puisse être employée avec les canaux d'accès et les protocoles demandés par le client.



SOA et services web: 3^{ème} SIL

9

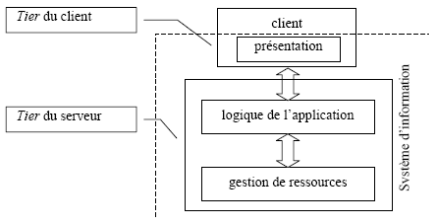
Architecture d'un SI: 1-tier



SOA et services web: 3^{ème} SIL

10

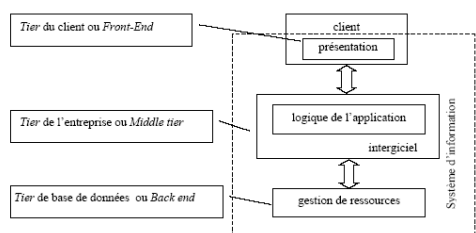
Architecture d'un SI: 2-tiers



SOA et services web: 3^{ème} SIL

11

Architecture d'un SI: 3-tiers



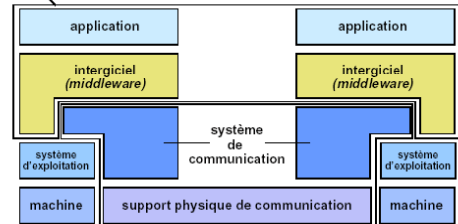
SOA et services web: 3^{ème} SIL

12

Middleware: Définition

Middleware: Mécanisme de base

applications réparties



Middleware:

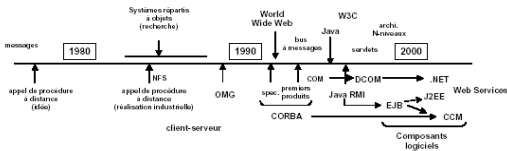
- En architecture informatique, un **middleware** est un logiciel tiers qui crée un réseau d'échange d'informations entre différentes applications informatiques. Le réseau est mis en œuvre par l'utilisation d'une même technique d'échange d'informations dans toutes les applications impliquées à l'aide de composants logiciels.
- Les composants logiciels du middleware assurent la communication entre les applications quels que soient les ordinateurs impliqués et quelles que soient les caractéristiques matérielles et logicielles des réseaux informatiques, des protocoles réseau, des systèmes d'exploitation impliqués.
- Les techniques les plus courantes d'échange d'informations sont l'échange de messages, l'appel de procédures à distance et la manipulation d'objets à distance.
- Les middleware sont typiquement utilisés comme *ciment* pour relier des applications informatiques disparates des systèmes d'information des entreprises et des institutions.

Middleware: Rôles de base

- Résoudre l'**Interopérabilité** : Unifier l'**accès** à des machines **distantes**
- Résoudre l'**Hétérogénéité** : Etre **indépendant** des systèmes d'exploitation et du **langage de programmation** des applications

Evolution des Middlewares

- Objets
 - CORBA (ORBIX, VisiBroker, OpenORB, ...)
 - DCOM
- Composant
 - J2EE (WebSphere, Weblogic, JBOSS)
 - .Net
- Web-Service



Classes de middlewares

■ Critères de classification

- Nature des entités qui communiquent
 - Objets
 - Composants
 - Autres
- Mode d'accès aux services
 - Synchrone (client-serveur)
 - Asynchrone (événements, messages)
 - Mixte
- Support de communication utilisé
 - Entités fixes / mobiles
 - Performances / qualité de service garanties ou non

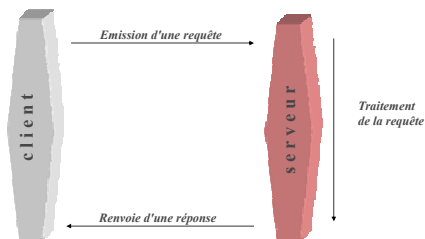
Pas de classification rigoureuse en raison de la diversité des réalisations

Remote Procedure Call RPC

Un exemple simple de middleware : Remote Procedure Call (RPC)

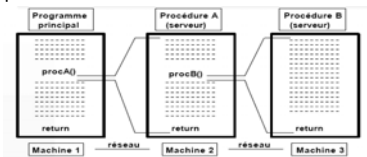
- Introduit par Birrell & Nelson (1984)
- Garder la sémantique de l'appel de procédure local ou *Local Procedure Call (LPC)*
- Fonctionnement synchrone
- Communication transparente entre le client et le serveur

Illustration du RPC

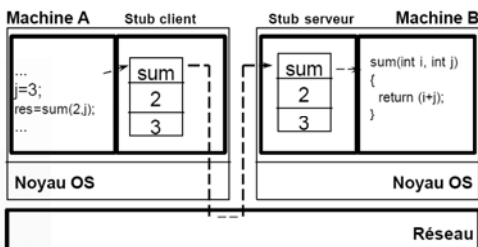


RPC: Principe

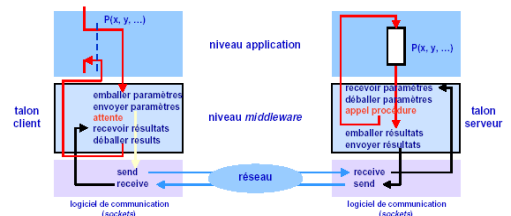
- Un client a une description (ou une interface) de procédures disponibles chez un serveur distant
- Le client invoque une procédure à distance
- Le serveur exécute localement cette procédure
- Le serveur renvoie au client le résultat (type de retour) de la procédure.



RPC : Exemple

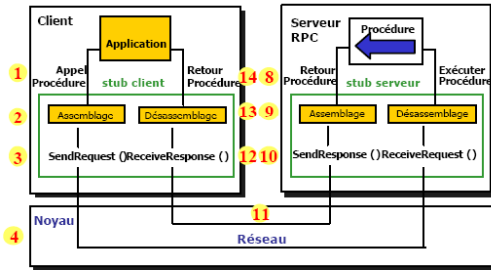


Un exemple simple de middleware : Remote Procedure Call (RPC)



■ Mise en œuvre de l'appel de procédure à distance

RPC : Fonctionnement



Problèmes

- Pas de passage de paramètres par adresse :
 - impossible de passer des pointeurs (ou références) en effet, les espaces d'adressage du client et du serveur sont différents donc aucun sens de passer une adresse
- La procédure distante n'a pas accès aux variables globales du client, aux périphériques d'E/S (affichage d'un message d'erreur !)
- Un appel de procédure obéit à un fonctionnement synchrone: une instruction suivant un appel de procédure ne peut pas s'exécuter tant que la procédure appelée n'est pas terminée

Remote Method Invocation RMI

RMI : Origine et Objectifs

- Solution (SUN) pour adapter le principe des RPC à la POO (à partir du JDK 1.1).
- Rendre transparent la manipulation d'objets situés dans un autre espace d'adressage
- Les appels doivent être transparents que l'objet soit local ou distant

```
Personne Opersonne = new Personne();
Int qi = Opersonne.calculerQi();
```

RMI : Généralités

- Principe:** mécanisme permettant l'appel de méthodes entre objets Java s'exécutant sur des machines virtuelles différentes (espaces d'adressage distincts), sur le même ordinateur ou sur des ordinateurs distants reliés par un réseau
 - utilise directement les socket
 - code ses échanges avec un protocole propriétaire : R.M.P. (Remote Method Protocol)
- Objectifs:** rendre transparent l'accès à des objets distribués sur un réseau faciliter la mise en œuvre et l'utilisation d'objets distants Java préserver la sécurité (inhérent à l'environnement Java)
 - RMI Security Manager
 - Distributed Garbage Collector (DGC)

RMI : Origine et Objectifs

- invoker une méthode d'un objet se trouvant sur une autre machine exactement de la même manière que s'il se trouvait au sein de la même machine `objetDistant.methode();`
- utiliser un objet distant (OD), sans savoir où il se trouve, en demandant à un service « dédié » de renvoyer son adresse `objetDistant = ServiceDeNoms.recherche("monObjet");`
- pouvoir passer un OD en paramètre d'appel à une méthode locale ou distante `resultat = objetLocal.methode(objetDistant);`
`resultat = objetDistant.methode(autreObjetDistant);`
- pouvoir récupérer le résultat d'un appel distant sous forme d'un nouvel objet qui aurait été créé sur la machine distante
`NouvelObjetDistant = ObjetDistant.methode();`

RMI : Objets et invocations distantes

- **Objet distant (objet serveur)**
 - ses méthodes sont invoquées depuis une autre JVM
 - dans un processus différent (même machine)
 - dans une machine distante (via réseau)
 - son comportement est décrit par une interface (ou plus) distante Java
 - déclare les méthodes distantes utilisables par le client
 - se manipule comme un objet local
- **Invocation distante (RMI)**
 - action d'invoquer une méthode d'une interface distante d'un objet distant
 - même syntaxe qu'une invocation sur un objet local

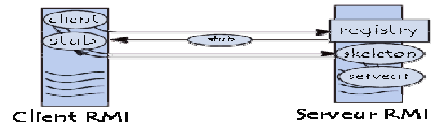
RMI : principes

- Outils pour :
 - la génération des stub/skeleton,
 - l'enregistrement par le nom,
 - l'activation
- Mono-langage et Multiplateforme: 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 paramètres peuvent être chargées dynamiquement via HTTP (<http://>) ou NFS (<file://>)

Structure des couches RMI

- **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
- **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

Fonctionnement RMI

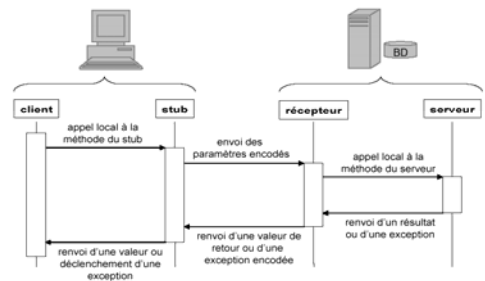


Lorsqu'un objet instancié sur une machine cliente désire accéder à des méthodes d'un objet distant, il effectue les opérations suivantes :

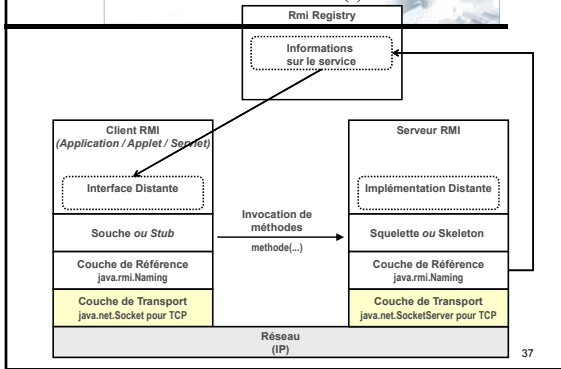
Fonctionnement RMI (2)

- 1) il localise l'objet distant grâce à un service de désignation : le registre RMI
- 2) il obtient dynamiquement une image virtuelle de l'objet distant (appelée stub ou *souche* en français). Le stub possède exactement la même interface que l'objet distant.
- 3) Le stub transforme l'appel de la méthode distante en une suite d'octets, c'est ce que l'on appelle la sérialisation, puis les transmet au serveur instanciant l'objet sous forme de flot de données. On dit que le stub "*marshalise*" les arguments de la méthode distante.
- 4) Le squelette instancié sur le serveur "désérialise" les données envoyées par le stub (on dit qu'il les "*démarshalise*"), puis appelle la méthode en local
- 5) Le squelette récupère les données renvoyées par la méthode (type de base, objet ou exception) puis les marshalise
- 6) le stub démarshalise les données provenant du squelette et les transmet à l'objet faisant l'appel de méthode à distance

RMI : un RPC à objets



Structure des couches RMI (i) : architecture



37

Structure des couches RMI

- 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



SOA et services web: 3^{ème} SIL

38

Mise en œuvre RMI

Pour créer une application avec RMI il suffit de procéder comme suit :

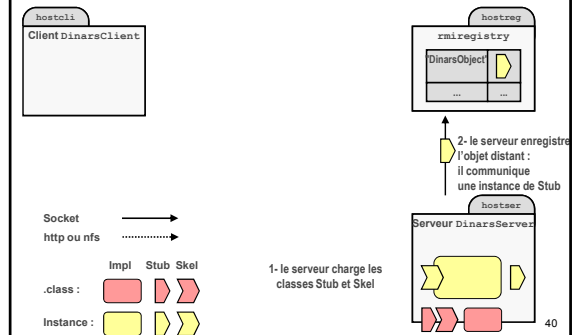
- définir la classe distante. Celle-ci doit dériver de `java.rmi.server.UnicastRemoteObject` (utilisant elle-même les classes `Socket` et `SocketServer`, permettant la communication par protocole TCP)
- définir l'interface pour la classe distante. Celle-ci doit implémenter l'interface `java.rmi.Remote` et déclarer les méthodes publiques globales de l'objet, c'est-à-dire les méthodes partageables. De plus ces méthodes doivent pouvoir lancer une exception de type `java.rmi.RemoteException`.
- créer les classes pour le stub et le squelette grâce à la commande `rmic`
- Lancer le registre RMI et lancer l'application serveur, c'est-à-dire instancier l'objet distant. Celui-ci lors de l'instanciation créera un lien avec le registre
- Créer un programme client capable d'accéder aux méthodes d'un objet sur le serveur grâce à la méthode `Naming.lookup()`
- Compiler l'application cliente
- Instancier le client



SOA et services web: 3^{ème} SIL

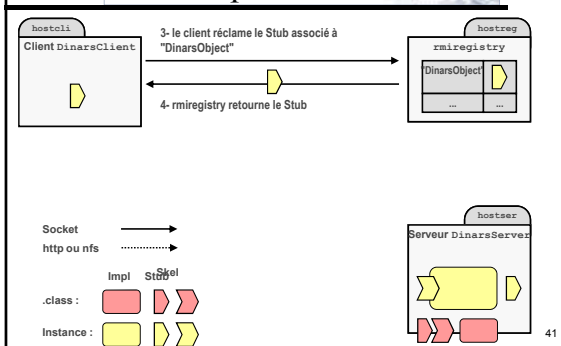
39

L'enregistrement de l'objet



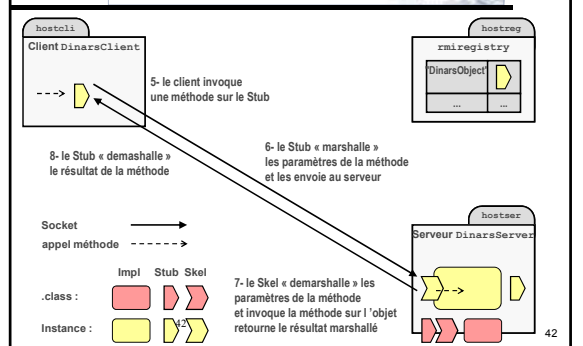
40

La récupération du Stub



41

Invocation d'une méthode



42

Etapes de création : serveur

Interface objet distant

```
1 package testrmi ;
2 import java.rmi.*;
3 public interface CalculInterface extends java.rmi.Remote {
4     public int additionner (int a , int b ) throws RemoteException ;
5     public int soustraire (int a, int b) throws RemoteException;
6 }
```

CalculInterface.java

```
import java.rmi.RemoteException;
import java.rmi.Naming;
public class CalculImpl extends java.rmi.server.UnicastRemoteObject
    implements CalculInterface {
    public CalculImpl () throws RemoteException {
        super();
    }
    public int additionner(int a, int b) throws RemoteException {
        return a + b;
    }
    public int soustraire(int a, int b) throws RemoteException {
        return a - b;
    }
    public static void main (String args[])
    {
        try {
            System.out.println("Création objet distant...\n");
            CalculImpl serveurCalcul = new CalculImpl();
            System.out.println("Création succes...\n");
            System.out.println("Enregistrement objet distant");
            Naming.rebind("rmi://localhost:1099/Calcul", serveurCalcul);
            System.out.println("Enregistrement réussi");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

CalculImpl.java

Explications

- Le service de nommage : **Java.rmi.Naming**

Fonction	Rôle
bind (name, obj)	Lie l'objet distant (remote object) à un nom spécifique
rebind (name, obj)	Lie l'objet distant même s'il existe déjà
unbind (name)	Retire l'association entre un nom et un objet distant
Obj lookup (url)	Renvoie l'objet distant associé à une URL
String [] list(url)	Renvoie la liste des associations sur la registry spécifiée dans l'URL

Etapes de création : serveur

Génération des stub et skeleton

- Compilation des fichiers : **Javac**
- Génération des stub et skeleton

> **Rmic testrmi.calculImpl**

```
C:\>"C:\Program Files\Java\jdk1.6.0_03\bin\rmic" testrmi.CalculImpl
C:\>cd testrmi
C:\testrmi>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 66E9-BDD1

Répertoire de C:\testrmi
24/09/2008 10:41 <REP>
24/09/2008 10:41 <REP>
24/09/2008 10:40 424 CalculImpl.class
24/09/2008 10:20 379 CalculImpl.java
24/09/2008 10:41 2 855 CalculImpl_Stub.class
24/09/2008 10:40 235 CalculInterface.class
24/09/2008 10:25 233 CalculInterface.java
```

Lancement du serveur

- Lancement du service de nommage :

> **rmiregistry <port>**

- Lancement du serveur

```
java -Djava.rmi.server.codebase=<URL>
-Djava.rmi.server.hostname=<hôte>
-Djava.security.policy=java.policy <serveur>.
```

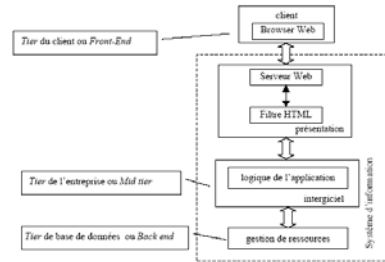
Ecriture de la classe client

```
1 public class ClientCalcul{
2     public static void main(String args[])
3     {
4         try {
5             CalculInterface objc = (CalculInterface)java.rmi.Naming.lookup
6             ("rmi://localhost:1099/Calcul");
7             int i = objc.soustraire(5,3);
8             int j = objc.additionner(5,3);
9             System.out.println(i); System.out.println(j);
10            }catch (Exception e) {
11                e.printStackTrace();
12            }
13        }
14    }
```


Lancement du client

```
>java -Djava.rmi.server.codebase=<URL>
-Djava.security.policy=java.policy
<client>
>2
>10
```

Architecture d'un SI: 3-tiers avec client léger



Service web et SOA

- Limites des middlewares "traditionnels"
 - Mono-langage : Java RMI,
 - Mono-plateforme : DCOM sous Windows,
 - Complexe à mettre en œuvre : CORBA.
- Adaptation des architectures réparties au contexte de l'Internet où **le Web est considéré comme un nouveau middleware**.
 - Multi-langage, Multi-plateforme,
 - Spécification garantie par un organisme indépendant,
 - Simple à mettre en œuvre.

Service web et SOA

Fournir une architecture générale pour les applications réparties sur internet :

- inter-opérables :
 - basé sur des standards ouverts
 - sans composant spécifique à un langage ou un système d'exploitation
- faiblement couplées :
 - limiter au maximum les contraintes imposées sur le modèle de programmation des différents éléments de l'application
 - par exemple ne pas imposer un modèle objet supportant la montée en charge : par exemple en n'imposant pas un modèle de type RPC