Chapitre 3 : Middleware orienté objet

Partie I: Java Remote Method Invocation (RMI)

1 Introduction

Le passage de l'orienté procédure à l'orienté objets amène plusieurs aspects intéressants :

- ✓ Peu ou pas de variables globales.
- ✓ Nouvelle façon de penser la décomposition de problèmes.
- ✓ Les programmes deviennent un ensemble d'objets faiblement couplé.
- ✓ Permet l'encapsulation (masquer la complexité des opérations).
- ✓ Construction de logicielle par assemblage de briques logicielles (Objets).
  - ➢ Tout cela rend le système plus flexible et les briques logicielles plus réutilisables.

2 Définition du Middleware orienté objet

Middleware orienté objet est un bus de communication spécifique entre un objet demandant l'exécution d'une opération sur un autre objet (serveur). Comme dans toutes les approches orientées objet (OO), le middleware OO se caractérise par une séparation Interface/Implémentation, un héritage qui vise la réutilisation et la communication entre objets distribués. La description de l'interface permet de convertir une application pour jouer le rôle d'un serveur, il suffit pour cela de la connecter à une interface objet.

Parmi les middlewares OO les plus connus on trouve '*Java Remote Method Invocation* - RMI' par Sun Microsystems, *Distributed Component Object Model* – DCOM par Microsoft, *Common Object Request Broker Architecture* – CORBA par l'OMG .etc.

3 Java Remote Method Invocation (RMI)

Java RMI est une abstraction du langage Java ou un modèle de programmation pour toute sorte de protocole à objets distribués. Le but de RMI est de créer des applications distribuées développées en Java. Il est apparu dès la version 1.1 du JDK (*Java Development Kit*).

RMI va plus loin que RPC puisqu'il permet non seulement l'envoi des données d'un objet, mais aussi de ses méthodes. Cela se fait en partie grâce à l'abstraction des objets. Avec RMI, les méthodes de certains objets (appelés objets distants) peuvent être invoquées depuis des JVM différentes (espaces d adressages distincts).

La répartition des objets sur différents processus et plateformes (ce qui donne une application répartie) est considérée comme une extension naturelle de l'approche basée objets. Souvent nous voulons dire par l'invocation de méthode le passage de message entre objets distribués. L'invocation de méthode est souvent une opération bloquante. En effet, RMI assure la communication entre objets clients et objets serveurs via TCP/IP et ce de manière transparente pour le développeur.

Les clients peuvent invoquer les méthodes d'autres objets (dits alors serveurs) sur le même site, on parle dans ce cas d'invocation locale, ou sur des sites distants, il s'agit alors du RMI.

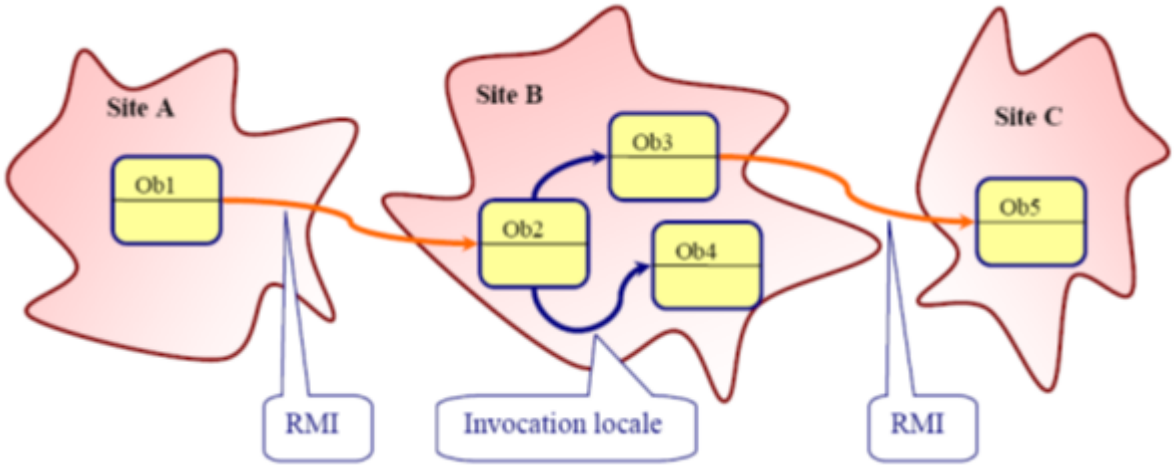

Figure 3.1- Invocation locale et invocation à distance.

3.1 Rappel sur les interfaces

- Classe abstraite : classe non instanciable, c'est à dire qu'elle n'admet pas d'instances directes. c.a.d Impossible de faire new ClasseAbstraite (…).
- Méthode abstrait : opération n'admettant pas d'implémentation au niveau de la classe dans laquelle elle est déclarée, on ne peut pas dire comment la réaliser.
- Une classe pour laquelle au moins une opération abstraite est déclarée est une classe abstraite (l'inverse n'est pas vrai).
- Une interface est une classe abstraite qui ne contient que des méthodes abstraites.
- Une classe peut hériter d'une seule classe et peut implémenter plusieurs interfaces.
- Implémenter une interface signifie qu'il faut redéfinir toutes les méthodes déclarées dans l'interface.
- Une interface est destinée à être "réalisée" (implémentée) par d'autres classes (celles-ci en héritent toutes les descriptions et concrétisent les opérations abstraites).

3.2 Architecture de RMI

L'architecture RMI est basée sur un principe important :

- La définition du comportement et l'exécution de ce comportement sont des concepts séparés.

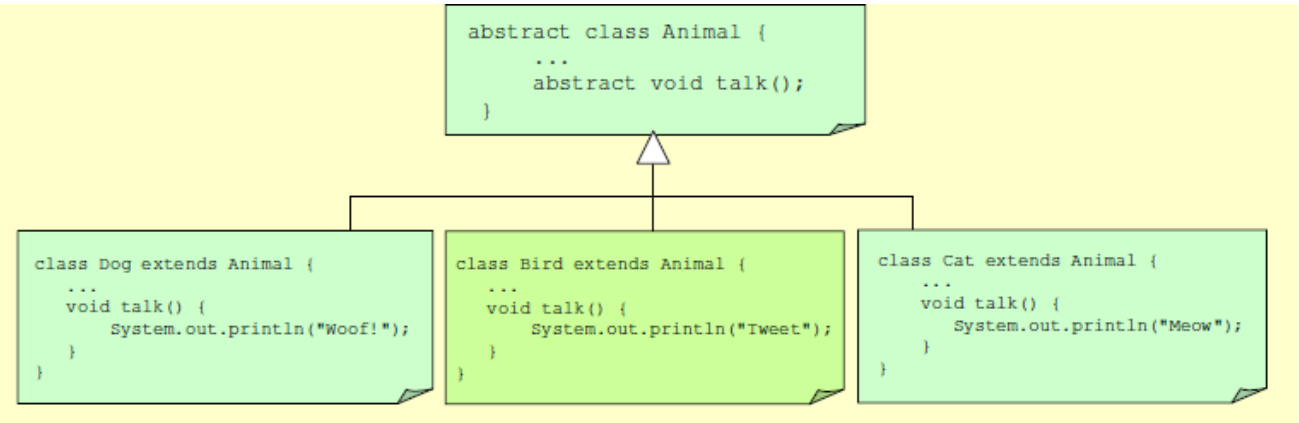

Figure 3.2- Classe abstraite et les classes concrètes (classes héritées).

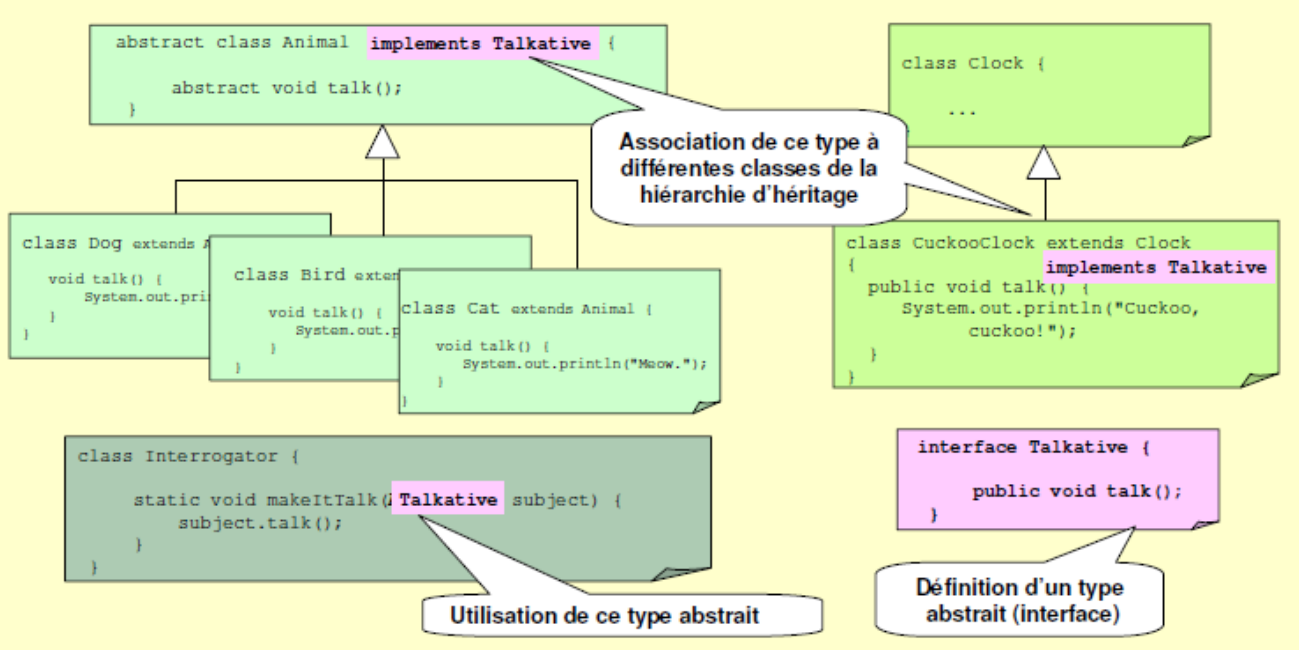

Figure 3.3- Un exemple d'une interface.

- ❑ La définition d'un service distant est codée en utilisant une interface Java.
- ❑ L'implémentation de ce service distant est codée dans une classe.

RMI est essentiellement construit sur une abstraction en trois couches.

- ■ Stubs et Skeletons (Souche et Squelette)
- ■ Remote Reference Layer (Couche de référencement distante)
- ■ Couche Transport

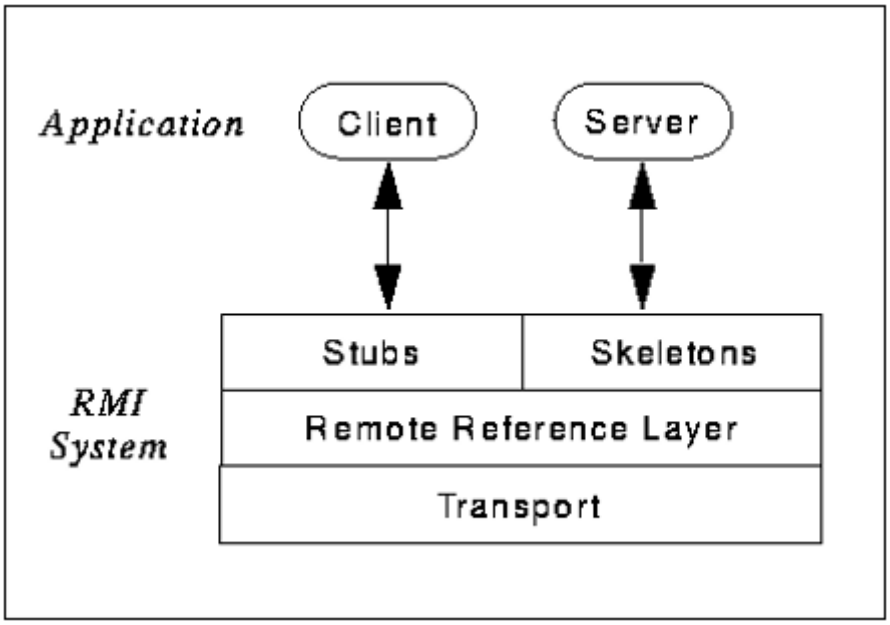

Figure 3.4- Architecture de RMI

3.2.1 Stubs et Skeletons

- Lorsque notre client fera appel à une méthode distante, cet appel sera transféré au stub.
- Le stub est donc un relais du côté du client.
- Il est le représentant local de l'objet distant.
- Il emballe les arguments de la méthode distante et les envoie dans un flux de données vers le service RMI distant.
- D'autre part, il déballe la valeur ou l'objet de retour de la méthode distante.
- Il communique avec l'objet distant par l'intermédiaire d'un skeleton.
- Le skéleton est lui aussi un relais mais du côté serveur.
- Il déballe les paramètres de la méthode distante, les transmet à l'objet local et emballe les valeurs de retours à renvoyer au client.
- Les stubs et les skeletons sont donc des intermédiaires entre le client et le serveur qui gèrent le transfert distant des données.
- On utilise le compilateur 'rmic' pour la génération des stubs et des skeletons avec le JDK
- Depuis la version 2 de Java, le skeleton n'existe plus. Seul le stub est nécessaire du côté du client et aussi du côté serveur.

3.2.2 Couche de référence distante

- Ce service est assuré par le lancement du programme ‘rmiregistry’ du côté du serveur
- Le serveur doit enregistrer la référence de chaque objet distant dans le service rmiregistry en attribuant un nom à cet objet distant.
- Du côté du client, cette couche permet l’obtention d’une référence de l’objet distant à partir de la référence locale (le stub) en utilisant le même service (rmiregsitry).

3.2.3 Couche transport

- La couche transport est basée sur les connexions TCP/IP entre les machines.
- Elle fournit la connectivité de base entre les 2 JVM.
- De plus, cette couche fournit des stratégies pour passer les firewalls.
- Elle suit les connexions en cours.
- Elle construit une table des objets distants disponibles.
- Elle écoute et répond aux invocations.
- Cette couche utilise les classes Socket et ServerSocket.
- Elle utilise aussi un protocole propriétaire R.M.P. (Remote Method Protocol)**.**

3.3 Démarches de RMI

étape 1. Créer les interfaces des objets distants

étape 2. Créer les implémentations des objets distants

étape 3. Générer les stubs et skeletons (Ce n'est pas nécessaire depuis JDK 11)

étape 4. Créer le serveur RMI

étape 5. Créer le client RMI

étape 6. Déploiement Lancement (Serveur et Client)

3.4 Récapitulation

1. On exécute rmiregistry
2. On exécute le Serveur
3. On crée l’objet distant
4. On l’enregistre dans l’annuaire avec la méthode Naming.bind()

5. On exécute le Client

6. On recherche la référence de l' objet distant avec la méthode Naming.lookup()

7. L'annuaire renvoie une référence de l'objet distant

8. On invoque une méthode distante par l'intermédiaire du STUB

9. On reçoit une valeur de retour envoyée par le SKELETON

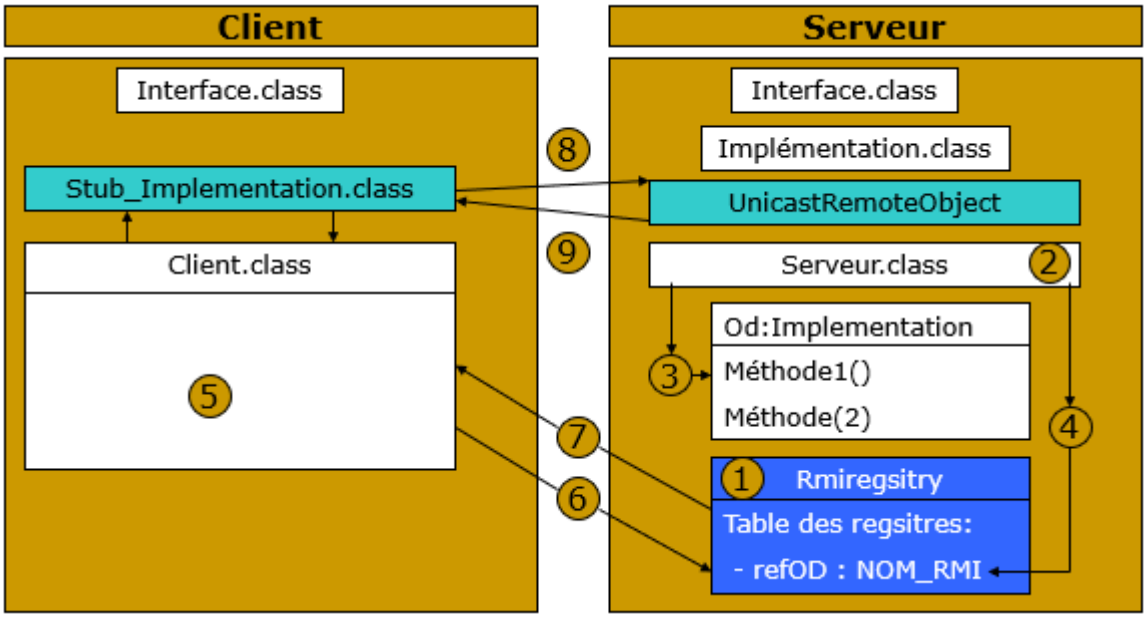

Figure 3.5- Synthèse de l'RMI