Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Licence en Informatique

Thème

Développement d'une application Client/serveur avec Java RMI

Réalisé par :

- *MAMMAR OSEMA*
- *BENAOUDA SID AHMED AMINE*

Présenté le 27 Juin 2013 devant la commission d'examination composée de MM.

- Mr BENMAMMAR BADR *(Encadreur)*
- Mr BELABED AMINE *(Examinateur)*
- Mr MERZOUG *(Examinateur)*

Année universitaire : 2012-2013

Remerciements

Nous remercions ALLAH de nous avoir données la santé et le courage afin de pouvoir réussir ce travail.

Ce travail est l'aboutissement d'un long cheminement au cours duquel nous avons bénéficié de l'encadrement, des encouragements et du soutien de plusieurs personnes, à qui nous tenons à dire profondément et sincèrement merci.

Nous exprimons notre grande gratitude à notre professeur encadrant Mr Badr Benmammar, d'avoir accepté de suivre notre travail et pour ses précieux conseils et ses orientations. Nous avons eu le privilège de travailler parmi votre équipe et d'apprécier vos qualités et vos valeurs. Votre sérieux, votre compétence.

Nous remercions toutes les personnes qui, d'une quelconque manière, nous ont apporté leur amitié, leur attention, leurs encouragements, leur appui et leur assistance pour que nous puissions mener à terme ce travail .

Nous tenons à exprimer nos sincères remerciements à tous les professeurs qui nous ont enseigné et qui par leurs compétences nous ont soutenu dans la poursuite de nos études.

Dédicaces

A
Mes chers parents

Pour les sacrifices déployés à nom *égard, pour leur patience, Leur encouragement et leur confiance à moi*

Ils ont tout fait pour mon bonheur et ma réussite.
Qu'ils trouvent dans ce modeste travail, le témoignage de mes Profondes affections et de mon attachement indéfectible. Nulle dédicace ne puisse exprimer ce que nous leur devons Que dieu leur réserve la bonne santé et une longue vie.

A
Mes chers grand parents, mes frères, mes sœurs à qui je souhaite la réussite dans les études.
A
Mon binôme Sid Ahmed Amine.
A
Tous mes camarades du Département d'Informatique spécialement à Omar, Ali et Soheyb … Merci pour tous ces moments qu'on avait vécus ensemble.
A
mes chers amis AEK, Mekki , Abd El Rahime et Mohamed qui m'ont tant aidé et soutenue pour effectuer ce travail.

A tous mes meilleurs amis.

Mammar Osema.

Dédicaces

A mes chers parents symbole de sacrifice, de tendresse, qui m'ont a éclairée mon chemin et qui m'ont encouragé et soutenue toute au long de mes études.

A ma grande sœur et son mari Omar à qui je souhaite beaucoup de bonheur dans la vie, à mes petites sœurs à qui je souhaite la réussite dans les études.
A tous les membres de ma famille, petits et grands, mes oncles, mes tantes.
A mon binôme Oussama
A tous mes camarades du Département d'Informatique spécialement à Fouad, Lout, Djamil, Kacem, les deux Abdelkader, Salah, Amine, Seddik Mohammed … Merci pour tous ces moments qu'on avait vécus ensemble.
A mes amis avec qui nous avons grandi Mustapha, Miloud, Radoine, Mourad, Abdelhak, Oussama, Boumediene.

A tous ceux qui m'aiment & que j'aime.

Sid Ahmed Amine

Table des matières

Chapitre II : Application Et Outils utilisés.

Table des Figures

Introduction Générale

La communication occupe une place très importante dans la vie humaine. Cette dernière est indispensable dans tous domaines : la vie sociale, professionnelle, civique, et personnelle.

À ce stade, il ne fait désormais plus aucun doute que les technologies de l'information et de la communication représentent la révolution la plus importante et la plus innovante qui a marqué la vie de l'humanité en ce siècle. En effet, loin d'être un éphémère phénomène de mode, ou une tendance passagère, ces technologies viennent nous apporter de multiples conforts à notre mode de vie, car ils ont révolutionné le travail des individus par leur capacité de traitement d'information, d'une part, et de rapprochement des dimensions espace/temps, d'une autre.

La progression des Technologies de la communication a créé l'interconnexion de réseaux de données solides qui sont profondément impact. Au début, les premiers réseaux étaient limites à échanger les informations reposant sur des caractères entre des systèmes informatique connectes, et avec les dernière générations d'outils de télécommunication, les réseaux sont en voie d'amélioration, ils ont prient en charge le transfert audio, des flux vidéo, du texte et des graphismes entre des périphériques de types très différents, la rétroaction devient plus aisée, et les messages se sont beaucoup enrichis.

La disponibilité du réseau permanente nécessite une technologie efficace et fiable. Dans ce cercle s'inscrit notre projet de fin d'études : « Développement d'une application client/serveur avec JAVA RMI ».

Ce projet s'articule autour de deux chapitres :

Chapitre 1 : Architecture Client-Serveur Et Java RMI

Dans ce chapitre, nous présentons les différentes architectures client/serveur, et JAVA RMI.

Chapitre 2 : Application Et Outils utilisés

Ce chapitre, s'intéresse à la présentation et réalisation de l'application, il contient les outilles utilisés, l'environnement du travail, le fonctionnement de l'application et les principales interfaces.

Chapitre I

Architecture Client-Serveur
Et
Java RMI

I. Introduction

Dans le but de centraliser les informations, ainsi que de les sécuriser le mieux possible, il devint nécessaire de créer une architecture de communication répondant à ces besoins. C'est en 1994 que l'architecture client-serveur s'implante sur le marché. Non seulement cette architecture rend-elle possible l'amélioration de la capacité de stockage des données, elle contribue à augmenter considérablement la rapidité de traitement des postes de travail. En effet, par la structure du serveur qui contient plusieurs unités de traitement (CPU), la capacité ainsi que la mémoire sont accrues. La progression de l'architecture client-serveur ne sera pas linéaire. En effet, cette technologie se développera à un rythme des plus accélérés. On peut affirmer qu'elle a connu une croissance exponentielle dans tous les domaines d'activités :

- Gestion de base de données.
- Les systèmes transactionnels.
- Les systèmes de messagerie, web, Internet.
- Les systèmes de partages des données.
- Les calcules scientifique.

II. Définition

De nombreuses applications fonctionnent selon un environnement client/serveur, cela signifie que des machines clientes (des machines faisant partie du réseau) contactent un serveur, une machine généralement très puissante en termes de capacités d'entrée-sortie, qui leur fournit des services. Ces services sont des programmes fournissant des données telles que l'heure, des fichiers, une connexion, etc.

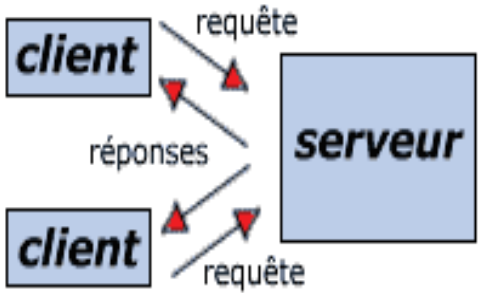

Figure 1.1 : Client-serveur

Les services sont exploités par des programmes, appelés programmes clients, s'exécutant sur les machines clientes. On parle ainsi de client (client FTP, client de messagerie, etc.) lorsque l'on désigne un programme tournant sur une machine cliente, capable de traiter des informations qu'il récupère auprès d'un serveur (dans le cas du client FTP il s'agit de fichiers, tandis que pour le client de messagerie il s'agit de courrier électronique).[1]

1. **Serveur :** ordinateurs spécialisé dans la fourniture et le stockage des ressources partagées des utilisateurs réseau.

Caractéristiques d'un serveur :

- Fournisseur de service ;
- Il est à l'écoute, prêt à répondre aux requêtes envoyées par des clients ;
- Dès qu'une requête lui parvient, il la traite et envoie une réponse ;
- Traitement de plusieurs clients simultanément.

2. **Clients :** ordinateurs qui accèdent aux ressources partagées fournies par un serveur du réseau.

Caractéristiques d'un client :

- Consommateur de service ;
- Il établit une connexion au serveur ;
- Il envoie des requêtes au serveur ;
- Il attend et reçoit les réponses du serveur.

3. **Requête :** message transmis par un client à un serveur décrivant l'opération à exécuter pour le compte du client.

4. **Réponse :** message transmis par un serveur à un client suite à l'exécution une opération, contenant le résultat de l'opération.

Le client et le serveur doivent bien sûr utiliser le même protocole de communication.

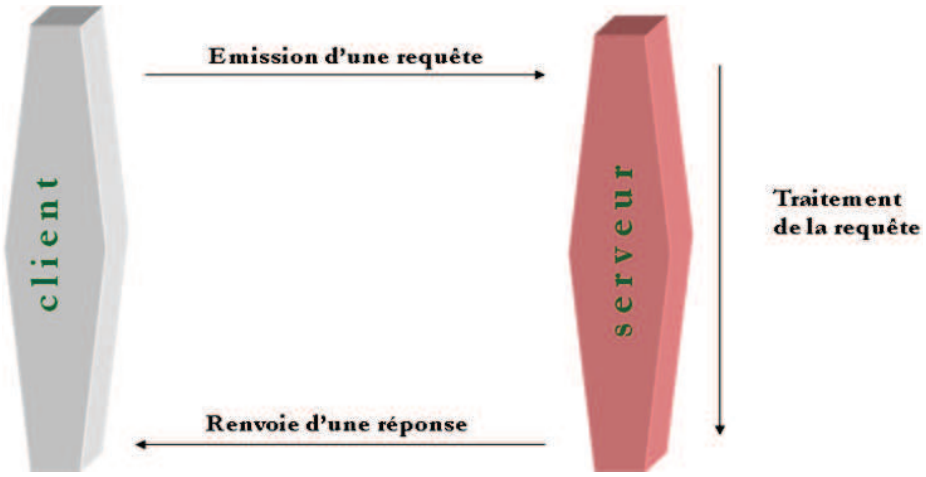

Figure I.2 : Modèle client –serveur [3].

III. Avantages de l'architecture client/serveur

Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux a sont :

- **Des ressources centralisées** : étant donné que le serveur est au centre du réseau, il peut gérer des ressources communes à tous les utilisateurs, comme par exemple une base de données centralisée, afin d'éviter les problèmes de redondance et de contradiction.
- **Une meilleure sécurité** : car le nombre de points d'entrée permettant l'accès aux données est moins important.
- **Une administration au niveau serveur** : les clients ayant peu d'importance dans ce modèle, ils ont moins besoin d'être administrés.
- **Un réseau évolutif** : grâce à cette architecture il est possible de supprimer ou rajouter des clients sans perturber le fonctionnement du réseau et sans modification majeure.

IV. Inconvénients du modèle client/serveur

L'architecture client/serveur a tout de même quelques lacunes parmi lesquelles :

- **un coût élevé** dû à la technicité du serveur.

- **un maillon faible** : le serveur est le seul maillon faible du réseau client/serveur, étant donné que tout le réseau est architecturé autour de lui. [1]

V. Les déférentes architectures client/serveur

Une application informatique peut être découpée en trois niveaux d'abstraction distincts: [9]

- **La couche de présentation :** permet l'interaction de l'application avec l'utilisateur. Cette couche gère les saisies au clavier, à la souris et la présentation des informations à l'écran.
- **la logique applicative, les traitements:** écrivant les travaux à réaliser par l'application. Ils peuvent être découpés en deux familles :
  * **les traitements locaux :** regroupant les contrôles effectues au niveau du dialogue avec l'IHM, visant essentiellement le contrôle et l'aide à la saisie.
  * **les traitements globaux:** constituant l'application elle-même, contient les règles internes qui régissent une entreprise donnée.
- **Les données :** l'accès aux données, regroupant l'ensemble des mécanismes permettant la gestion des informations stockées par l'application.

Ces trois niveaux peuvent être imbriqués ou repartis de différentes manières entre plusieurs machines physiques.

Le noyau de l'application est composé de la logique de l'affichage et la logique des traitements.

Le découpage et la répartition de ce noyau permettent de distinguer les architectures applicatives suivantes :

L'architecture 1-tiers

L'architecture 2-tiers

L'architecture 3-tiers

Les architectures n-tiers

1. Présentation de L'architecture 1-tiers

Dans une application un tiers les couches applicatives sont liées et s'exécutent sur le même ordinateur. On ne parle pas ici d'architecture client-serveur, mais d'informatique centralisée.

Dans ce contexte plusieurs utilisateurs se partagent des fichiers de données stockés sur un serveur commun.

2. Présentation de l'architecture 2-tiers

L'architecture à deux niveaux (aussi appelée architecture 2-tiers) caractérise les systèmes clients/serveurs pour lesquels le client demande une ressource et le serveur la lui fournit directement, en utilisant ses propres ressources. Cela signifie que le serveur ne fait pas appel à une autre application afin de fournir une partie du service.
L'échange de messages transite à traverse un réseau reliant les deux machines (client et serveur), il met en œuvre des mécanismes relativement complexe qui sont, en général, pris en charge par un middleware.

Le cas typique de cette architecture est une application de gestion fonctionnement sous Windows ou Linux et exploitant un SGBD centralisé. S'exécutant le plus souvent sur un serveur dédié, ce dernier est interrogé en utilisant un langage de requête comme SQL.

Ce type d'application permet de tirer une partie de la puissance des ordinateurs déployés en réseau pour fournir à l'utilisateur une interface riche, tout en garantissant la cohérence des données, qui restent gérées de façon centralisée.

L'architecture 2-tiers présente de nombreux avantages qui lui permettent de présenter un bilan globalement positif : elle permet l'appropriation des applications par l'utilisateur, l'utilisation d'une interface utilisateur riche et elle introduit la notion d'interopérabilité. [4]

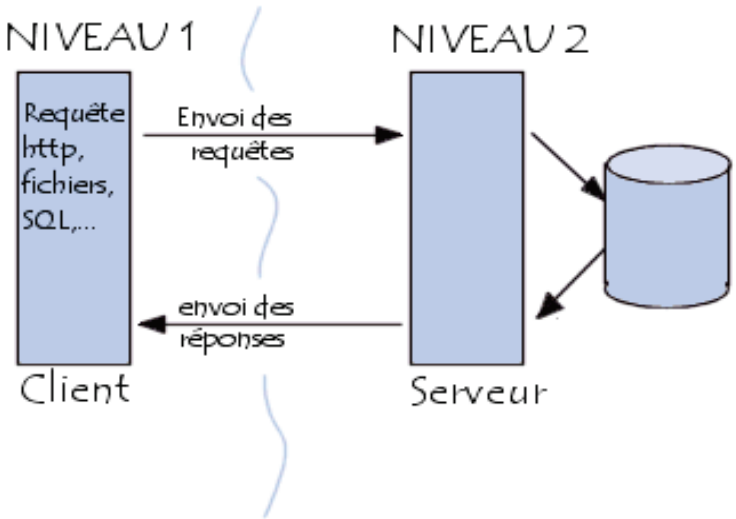

Figure I.3 :l'architecture à 2 niveaux. [1]

3. Présentation de l'architecture 3-tiers

Pour résoudre les limitations du client/serveur 2-tiers tout en conservant ses avantages, on a cherché une architecture plus évoluée, facilitant les forts déploiements à moindre coût, La réponse est apportée par les architectures distribuées, on utilisant un poste client simple communicant avec le serveur par le biais d'un protocole standard.
Cette architecture 3-tiers également appelée client-serveur de deuxième génération ou client-serveur distribué applique les principes suivants :

Les données sont toujours gérées de façon centralisée, la présentation est toujours pris par le post client, la logique applicative est pris en charge par un serveur intermédiaire.

L'architecture 3-tiers sépare l'application en 3 niveaux de services distincts, conforme aux principes précédents :

- **Premier niveau** : l'affichage et les traitements locaux (contrôle de saisie, mise en forme de données…) sont pris en charge par le post client.
- **Deuxième niveau** : les traitements applicatifs globaux sont pris en charge par le service applicatif.
- **Troisième niveau** : les services de base de données sont pris en charge par un SGBD. [4]

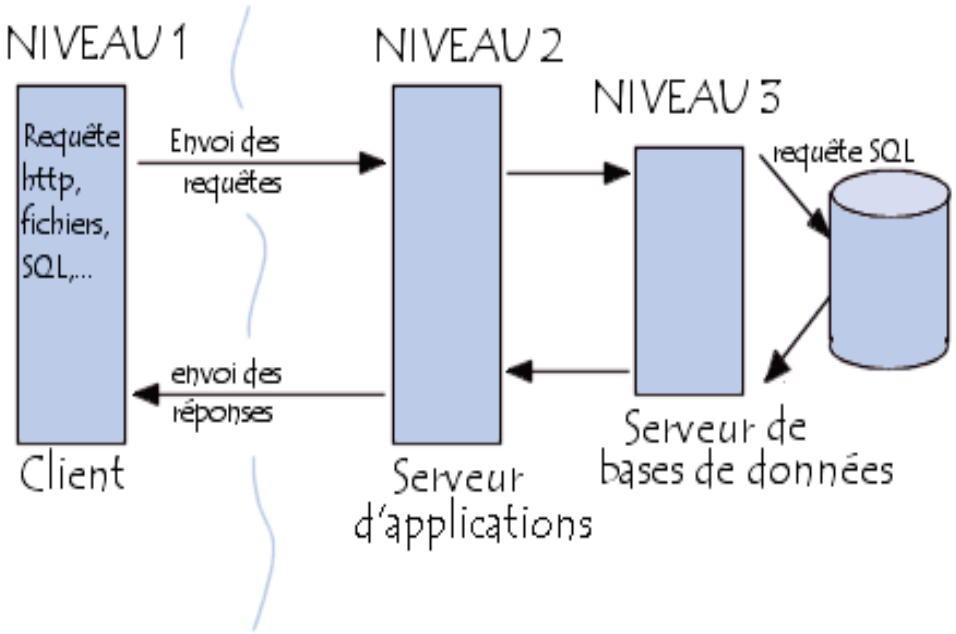

Figure I.4 :l'architecture à 3 niveaux. [1]

Comparaison des deux types d'architecture

L'architecture à deux niveaux est donc une architecture client/serveur dans laquelle le serveur est polyvalent, c'est-à-dire qu'il est capable de fournir directement l'ensemble des ressources demandées par le client. Dans l'architecture à trois niveaux par contre, les applications au niveau serveur sont délocalisées, c'est-à-dire que chaque serveur est spécialisé dans une tâche (serveur web/serveur de base de données par exemple).
L'architecture à trois niveaux permet :

- Une plus grande flexibilité/souplesse ;
- Une sécurité accrue car la sécurité peut être définie indépendamment pour chaque service, et à chaque niveau ;
- De meilleures performances, étant donné le partage des tâches entre les différents serveurs.

4. Les architectures n-tiers

L'architecture multi-niveaux a été pensée pour pallier aux limitations des architectures trois tiers et concevoir des applications puissantes et simples à maintenir.
Elle qualifie la distribution d'application entre de multiples services et non la multiplication des niveaux de services.

L'architecture n-tiers supprime tous les inconvénients des architectures précédentes :

- Elle permet l'utilisation d'interfaces utilisateurs riches,
- Elle sépare tous les niveaux de l'application,
- Elle offre des grandes capacités d'extension,
- Elle facilite la gestion des sessions.
- Elle est basée sur la programmation d'objet ainsi sur des communications standards entre application et le concept Middleware objet.

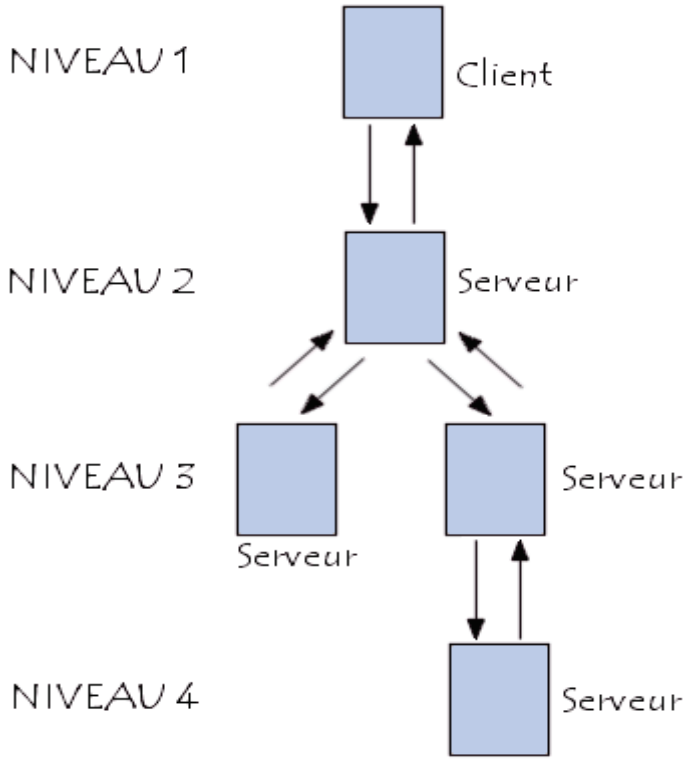

Figure I.5 *:* L'architecture multi-niveaux. [1]

VI. Notion de middleware

Le middleware est une couche intermédiaire (couche logiciel) qui s'intercale entre l'infrastructure de communication d'un réseau et les éléments de l'application distribuée.

Le middleware est un logiciel qui se compose d'un ensemble de services qui permettent à plusieurs entités (processus, objets etc.) résidents sur plusieurs ordinateurs, d'interagir à travers un réseau en dépit des différences dans les protocoles de communication, architectures des systèmes locaux, etc... [3].

VII. Remote Method Invocation (RMI)

1. Introduction

Avec l'évolution des langages de programmation et de l'orienté objet, la communication client-serveur s'est encoure améliorée et rendue plus accessible. Java RMI (Remote Method Invocation) est un exemple de technologie orienté-objet (grâce à java).

Le but de RMI est de créer un model objet distribuée java qui s'intègre naturellement au langage java et au model orienté objet local. Ce système étend la sécurité et la robustesse de java au mode applicatif distribuée. RMI apparaît avec Java 1.1 et est complètement intégré dans Java 1.1 donc, C'est un bon pas vers CORBA. C'est du CORBA allégé avec ses avantages (c'est plus simple) et ses inconvénients (tout doit être Java coté client comme coté serveur).

On dit généralement que RMI est une solution "tout Java", contrairement à la norme CORBA de l'OMG (Object Management Group) permettant de manipuler des objets à distance avec n'importe quel langage. CORBA est toutefois beaucoup plus compliqué à mettre en œuvre, contrairement au RMI qui permet aux développeurs de créer des applications distribuées de manière simple puisque la syntaxe et la sémantique reste la même que pour les applications habituellement, c'est la raison pour laquelle de nombreux développeurs se tournent généralement vers RMI. **[1]**

2. Définition

RMI est un ensemble de classes qui permettent la communication entre machines virtuelles Java (JVM) qui peuvent se trouvent physiquement sur la même machine ou sur deux machines distinctes: c.à.d. la manipulation des objets sur des machines distantes (objets distants) de manière similaire aux objets sur la machine locale (objet locaux).

Ces manipulations sont "relativement" transparentes. Pour cela RMI propose :

- un ramasse-miettes distribué.
- la gestion des représentants locaux d'objets distants et leur activation.
- la liaison avec les couches transport et l'ouverture de sockets appropriés.

- la syntaxe d'invocation, d'utilisation d'un objet distant est la même qu'un objet local.

En Java, une fois que l'environnement détermine qu'un objet n'était plus utile, le ramasse-miettes (en anglais, Garbage Collector) libère la mémoire qu'il occupe.

L'avantage est immense par rapport aux langages plus anciens, où le développeur devait connaître à tout instant ce qui se trouvait dans la mémoire, et gérer de manière efficace sa libération.

3. L'application RMI :

Une application RMI est composée d'une partie client et d'une partie serveur.

Le rôle d'un serveur est de créer des objets qu'on qualifie de « distants », de les rendre accessibles à distance et enfin d'accepter des connexions de clients vers ces objets.

Le rôle d'un client est donc d'accéder aux méthodes de ces objets, tout en considérant comme si ces objets étaient des objets locaux. RMI est très souvent utilisé en parallèle avec l'API d'annuaire JNDI (services de nommage). afin que les clients trouvent les services distants, ceux-ci doivent au préalable être enregistrés dans un annuaire, pour cela RMI fourni un rmiregistry.

rmiregistry : possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants.

Package RMI :

C'est un système d'objets distribués constitué uniquement d'objets Java, et qui permet et assure la communication entre machines virtuelles Java. [3]

Quelques classes :

java.rmi.Naming.
java.rmi.Remote.
java.rmi.RemoteException.
java.rmi.server.UnicastRemoteObject.
java.rmi.registry.LocateRegistry.
java.rmi.NotBoundException.

4. Architecture RMI

L'architecture RMI définit la manière dont se comportent les objets, comment et quand des exceptions peuvent se produit, comment gérer la mémoire et comment les méthodes appelées passent et reçoivent les paramètres.

Le système RMI contient 3 couches qui sont :

- La couche des amorces (stub/skelton).
- La couche des références (RRL).
- La couche de transport.

Chacun de ses couches est indépendante de l'autre et utilise un protocole spécifique, la transmission des objets utilise deux technique : [9]

- La sérialisation
- Le chargement dynamique du stub, permet au client de charger dynamiquement le stub quand il a seulement l'interface.

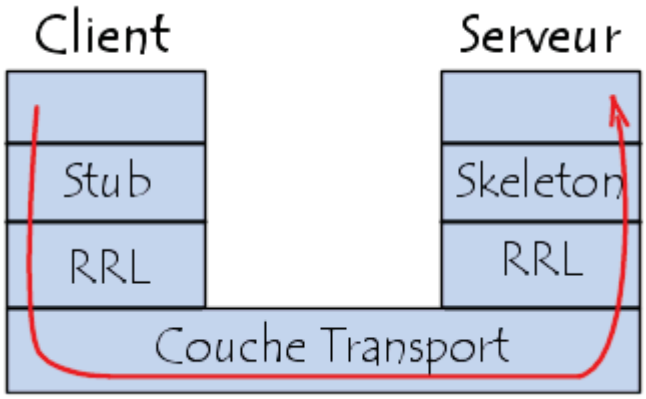

Figure 1.6 : Structure des couches RMI. [1]

1ère couche : Stub/Skeleton

Le stub (traduisez souche) et le skeleton (traduisez squelette), respectivement sur le client et le serveur, assurent la conversion des communications avec l'objet distant.

Stub (coté client) : est un mandataire de l'objet qui est chargé dans le client au moment de l'obtention de la référence, cette référence implémente les mêmes interfaces que l'objet distante, le stub a pour but de :

- Initie une connexion avec la JVM distante en transmettant l'invocation distante à la couche des références d'objets (RRL).
- Assemble les paramètres pour leur transfert à la JVM distante.
- Attend les résultats de l'invocation distante, désassemblent la valeur ou l'exception renvoyée, et renvoient la valeur à l'appelant.

Skeletons (coté serveur) : contient une méthode qui appelle les méthodes de l'objet distant.

Il a pour but de :

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

2ème couche : RRL (couche des références d'objets)

La couche de référence (RRL, Remote Reference Layer) est chargée du système de localisation afin de fournir un moyen aux objets d'obtenir une référence à l'objet distant. Elle est assurée par le package java.rmi.Naming. On l'appelle généralement registre RMI car elle référence les objets.

rmiregistry s'exécute sur chaque machine hébergeant des objets distants.

3ème couche : Transport

La couche de transport permet d'écouter les appels entrants ainsi que d'établir les connexions et le transport des données sur le réseau par l'intermédiaire du protocole TCP. Les packages java.net.Socket et java.net.SocketServer assurent implicitement cette fonction. Les connexions et les transferts de données dans RMI sont effectués par Java sur TCP/IP grâce à un protocole propriétaire JRMP, (Java Remote Method Protocol) sur le port 1099.

A partir de Java 2 version 1.3, les communications entre client et serveur s'effectuent grâce au protocole RMI-IIOP (Internet Inter-Orb Protocol), un protocole normalisé par l'OMG et utilisé dans l'architecture CORBA.

Généralement lors d'un appel d'une méthode, le Stub transmet l'appel à la couche RRL qui assure la connexion avec le serveur en point à point (unicast).

Cette couche transmet la requête à la couche transport qui utilise JRMP au dessus de TCP/IP et transfère la requête en remontant vers le skeleton qui appelle la méthode sur l'objet distant.

Avec Java 2, le skeleton est devenu obsolète (dépassé), une même classe skeleton générique est partagée par tous les objets distants. En plus, jusqu'à la version 5.0 du

J2SE (2005), il fallait utiliser un compilateur de stub appelé RMIC (Java RMI Compiler) pour générer les stub/skeleton avant tout enregistrement sur le registre RMI.

5. Principe de fonctionnement RMI. [2]

Coté Serveur :

A la création de l'objet, un stub et un skeleton (avec un port de communication) sont créés coté serveur.

1. L'objet serveur s'enregistre auprès d'un annuaire (rmiregistry) en utilisant la classe Naming (méthode rebind).
2. L'annuaire (rmiregistry) enregistre le stub de l'objet.
3. L'annuaire est prêt à donner des références à l'objet Serveur.

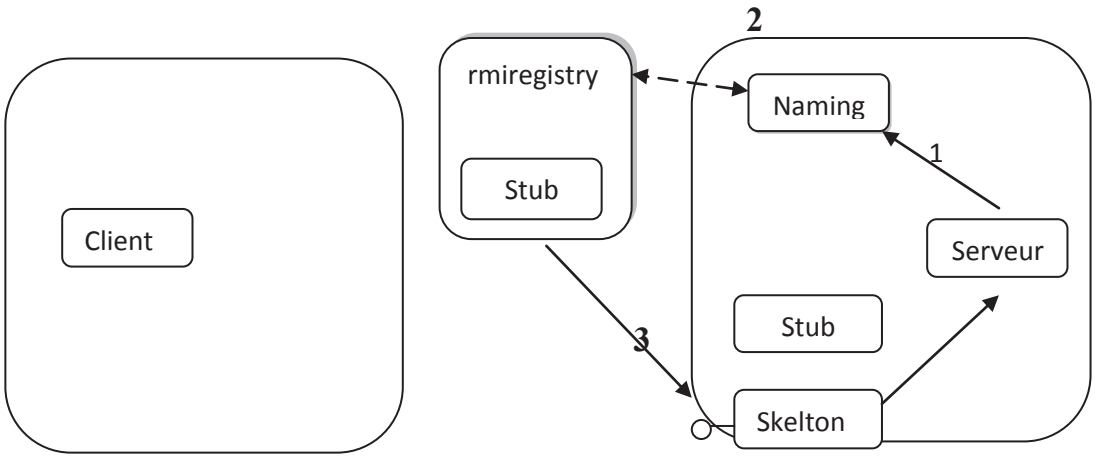

Figure 1.7 : Java RMI coté Serveur

Coté Client :

4. L'objet client fait appel à l'annuaire (rmiregistry) en utilisant la classe Naming pour localiser l'objet serveur (méthode lookup).
5. L'annuaire délivre une copie du stub.
6. L'objet stub est installé et sa référence est retournée au client.
7. Le client effectue l'appel à l'objet serveur par appel à l'objet stub.

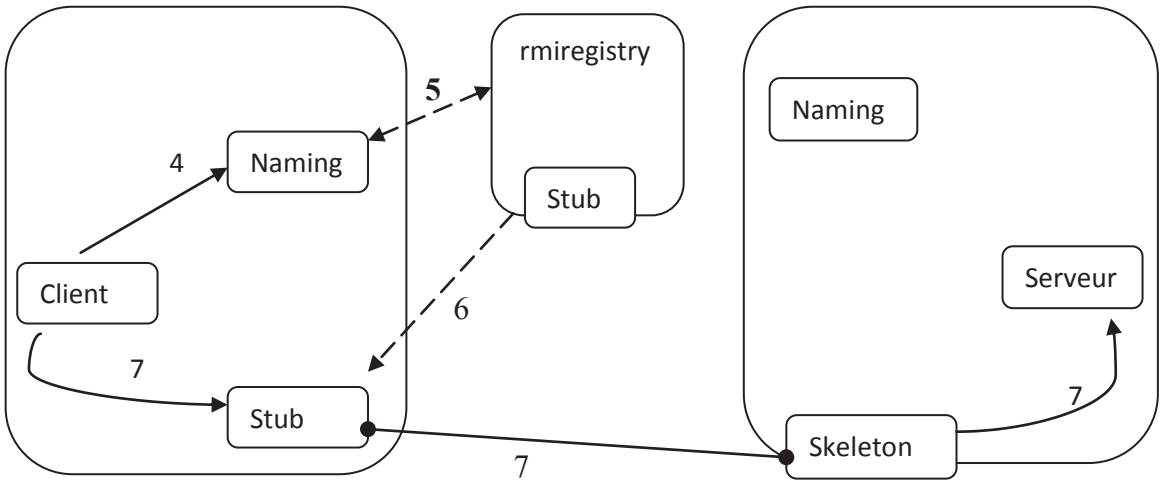

Figure 1.8: Java RMI coté Client

6. Déploiement de RMI

L'utilisation d'interfaces est un principe important dans RMI, En effet, la définition d'une fonctionnalité et l'implémentation de celle-ci sont deux concepts séparés, Une interface décrit l'interaction entre client et fournisseur du service.

En RMI, on utilise une interface pour coder la définition d'un service et une classe pour coder son implémentation (située à distance), une même interface est implémentée par deux classes : une classe implémente le service (skeleton) sur le serveur, et une classe implémente un proxy (stub), pour le service distant sur le client.

Pour créer une application avec RMI il y a :

- Coté serveur :

La définition d'une interface qui contient les méthodes qui peuvent être appelées à distance (l'interface est partagée avec le client), celle-ci doit étendre java.rmi.Remote, et déclarer les méthodes public globales de l'objet, c'est-à-dire les méthodes partageables, de plus ces méthodes doivent lancer une exception de type java.rmi.RemoteException.

L'écriture d'une classe qui implémente cette interface, cette classe doit dériver de java.rmi.server.UnicastRemoteObject.

L'écriture d'une classe (serveur) qui instanciera l'objet et l'enregistrera en lui affectant un nom dans RMI Registry.

- Côté client :

L'écriture d'une classe (client) capables d'accéder aux méthodes d'un objet sur le serveur grâce à la méthode Naming.lookup(nom_de_l'objet), cette classe permet :

L'obtention d'une référence sur l'objet distant à partir de son nom.

L'appel à la méthode à partir de cette référence. [3]

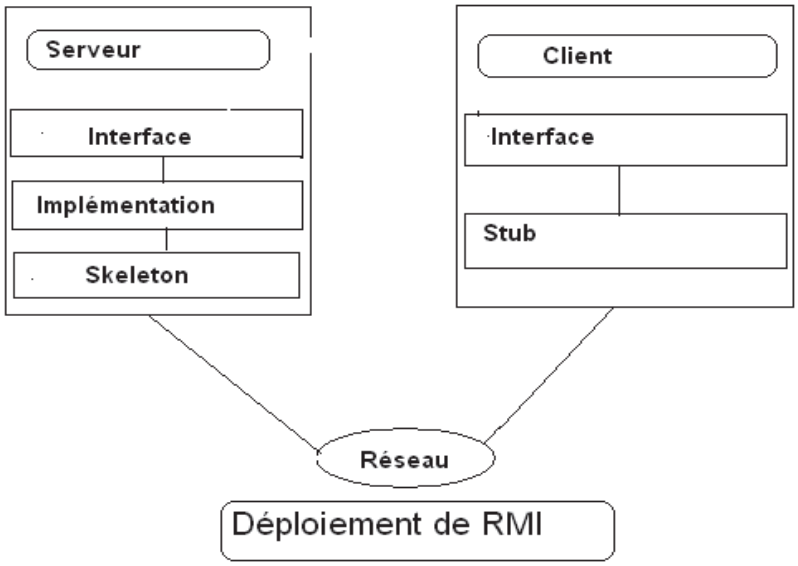

Figure 1.9: Déploiement de RMI.

VIII. Conclusion :

Nous avons abordé dans ce premier chapitre les différents types de l'architecture client/serveur .ainsi nous avons expliqué les principes de l'application RMI qui permet la communication entre machines virtuelles Java sur des machines distantes. En peut l'utiliser pour développer des applications client/serveur comme la communication basée sur la discussion instantanée.

Chapitre II

Application Et Outils utilisés

I. Introduction

Il nous reste à mettre les concepts de RMI en pratique sur une application clients/serveur de « TCHAT » pour cela nous avons choisis le langage JAVA avec l'environnement Netbeans, et nous avons utilisés ``Swing`` pour développer l'interface graphique, ainsi MySQL pour le but de crée une Table des clients, on utilise EasyPHP.

II. L'ENVIONEMENT DE TRAVAIL

1. JAVA

Java intègre les concepts les plus intéressants des technologies informatiques récentes dans une plate-forme de développement riche et homogène,

Java a été développée par SUN au début des années 90 dans une optique de plus grand portabilité d'une machine à une autre, et d'une grande fiabilité, il s'agit donc d'un environnement de Programmation orientée objet .

Il est possible d'utiliser Java pour créer des logiciels dans des environnements très diversifiés :

- Applications sur client lourd (JFC) ;
- Applications Web, côté serveur (servlets,JSP,Struts,JSF) ;
- Applications réparties (EJB) ;
- Applications sur carte à puce (JavaCard).

Ces applications contiennent des nombreuses fonctionnalités :

- Accès à des bases de données (JDBC) ;
- Traitement XML ;
- Web services (JAXM) ;
- Accès à des annuaires (JNDI).

- Caractéristiques du JAVA :

* Orienté Objet.
* Distribué.
* Multithread.
* Robuste.
* Dynamique
* Portable.
* Haute performance.

- Les composantes de java :

***Langage orienté objet JAVA.**

***JVM (machine virtuelle Java) :** permette d'interpréter et d'exécuter le bytecode java

*** API (Application Programming Interface) :** un ensemble de classes standards (bibliothèque standard).

***JRE (Java Runtime Environement)** : l'environnement d'exécution
Java désigne un ensemble d'outils permettant l'exécution de programmes Java sur toutes les plates-formes supportées.
JRE est constitue de la JVM et de la Bibliothèque Standard(API).

***Java Development Kit (JDK) :** le nouveau terme c'est SDK (Standard Développent Kit) qui est l'environnement dans lequel le code Java est compilé pour être transformé en bytecode afin que la machine virtuelle Java (JVM) puisse l'interpréter.
JDK est composé de :
Javac : le compilateur.
Javadoc : le générateur de documentation.
Jar : L'archiver.
Jdb : Le débogueur.
JRE : Environnement d'exécution java. D'après : [3]

Figure II.1 : Java

2. NetBeans

2.1. Historique :

En 1997, NetBeans naît de Xelfi, un projet d'étudiants dirigé par la Faculté de Mathématiques et de physique de l'Université Charles de Prague. Plus tard, une société se forme autour du projet et édite des versions commerciales de l'EDI NetBeans, jusqu'à ce qu'il soit acheté par Sun en 1999. Sun place le projet sous double licence CDDL (Common Development and Distribution License) et GPL v2 en juin de l'année suivante.

2.2. Définition :

NetBeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000. En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML.

Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web).

Netbeans est une plateforme, vous permettant d'écrire vos propres applications Swing. Sa conception est complètement modulaire : Tout est module, même la plateforme. Ce qui fait de Netbeans une boite à outils facilement améliorable ou modifiable.

2.3. Environnement de base

L'environnement de base comprend les fonctions générales suivantes:

- Configuration et gestion de l'interface graphique des utilisateurs ;
- Support de différents langages de programmation ;
- Traitement du code source (édition, navigation, formatage, inspection..) ;
- Fonctions d'import/export depuis et vers d'autres IDE, tels qu'Eclipse ou JBuild ;
- Accès et gestion de bases de données, serveurs Web, ressources partagées ;
- Gestion de tâches ;
- Documentation intégrée. [5]

3. AWT et SWING

La première API pour développer des interfaces graphiques portables d'un système à un autre en Java est AWT. Cette API repose sur les composants graphiques du système

sous-jacent ce qui lui assure de bonnes performances. Malheureusement, ces composants sont limités dans leur fonctionnalité car ils représentent le plus petit dénominateur commun des différents systèmes concernés. Pour pallier ce problème, Sun a proposé une nouvelle API, Swing. Swing fait partie de la bibliothèque Java Foundation Classes (JFC). C'est une API dont le but est similaire à celui de l'API AWT mais dont les modes de fonctionnement et d'utilisation sont complètement différents. Swing a été intégré au JDK depuis sa version 1.2. Cette bibliothèque existe séparément pour le JDK 1.1.

Swing utilise la même infrastructure de classes qu'AWT, ce qui permet de mélanger des composants Swing et AWT dans la même interface. Il est toutefois recommandé d'éviter de les utiliser simultanément car certains peuvent ne pas être restitués correctement. Les principales différences sont :

- Utilisation du double buffering qui améliore les rafraichissements.
- Utilisation d'un panneau de contenu (contentPane) pour insérer des composants.

Le double-buffering est le fait d'avoir un buffer (voir l'image se tracer) où l'on dessine en mémoire et un buffer qui s'occupe de l'affichage.

Principe est : dessiner sur le buffer mémoire (des images, des textes, des dessins...), ensuite, dessiner le buffer mémoire dans le buffer d'affichage (séparation entre le chargement et l'affichage).

Les composants Swing ne sont plus insérés directement au JFrame mais à l'objet contentPane (de la classe JRootPane) qui lui est associé. [3]

Quelques composants de Swing utilisées dans notre application :

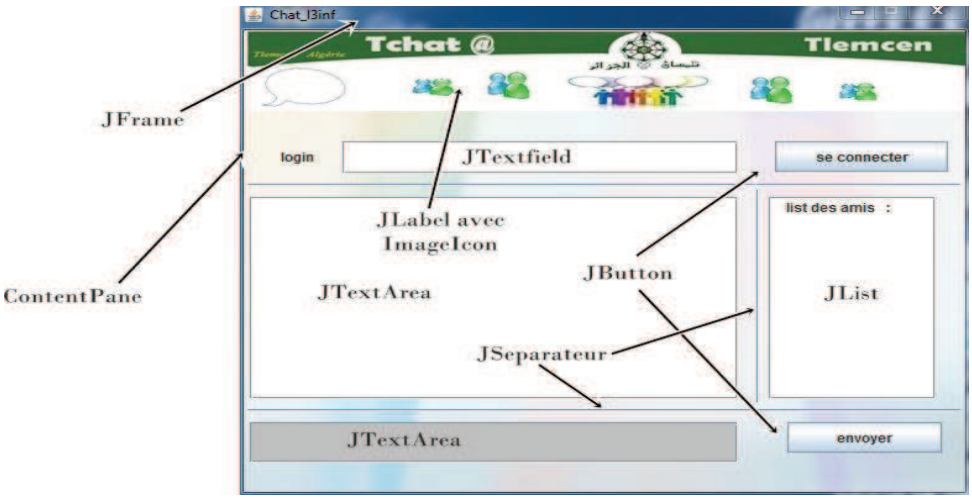

Figure II.2: Swing

4. JDBC (Java DataBase Connectivity)

JDBC est une API fournie avec Java (depuis sa version 1.1) permettant de se connecter à des bases de données, c'est-à-dire que JDBC constitue un ensemble de classes permettant de développer des applications capables de se connecter à des serveurs de bases de données (SGBD).

L'API JDBC a été développée de telle façon à permettre à un programme de se connecter à n'importe quelle base de données en utilisant la même syntaxe, c'est-à-dire que l'API JDBC est indépendante du SGBD.

De plus, JDBC bénéficie des avantages de Java, dont la portabilité du code, ce qui lui vaut en plus d'être indépendant de la base de données d'être indépendant de la plate-forme sur laquelle elle s'exécute. [6]

Principaux éléments de JDBC

JDBC Driver API

L'interface java.sql.Driver est destinée aux développeurs de drivers (pilotes) désirant interfacer un SGBD à Java en utilisant JDBC. La programmation d'un driver JDBC consiste à implanter les éléments définis dans les interfaces abstraites de l'API JDBC.

DriverManager

L'interface java.sql.DriverManager gère la liste des drivers JDBC chargés dans la machine virtuelle Java et la création des connexions TCP. Il sert également à mettre en correspondance les URL utilisés par les connexions avec les drivers à disposition. Ainsi, le DriverManager décortique les URL afin d'en extraire le sous-protocole et le service, puis recherche dans sa liste de drivers celui (ou ceux) capable(s) d'établir la connexion.

java.sql.Connection Gère les connexions existantes, crée les requêtes, gère les transactions.

Une fois créée, une Connection va servir à envoyer des requêtes au SGBD et à récupérer les résultats, ces deux tâches étant effectuées à l'aide des interfaces Statement (requête) et ResultSet (ensemble résultat).

JDBC et SQL

SQL (sigle de Structured Query Language, en français langage de requête structurée) est un langage informatique normalisé servant à effectuer des opérations sur des bases de données relationnelles. La partie langage de manipulation de données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

Créé en 1974, normalisé depuis 1986, le langage est reconnu par la grande majorité des systèmes de gestion de bases de données relationnelle (abrégé SGBDR) du marché. SQL fait partie de la même famille que les langages SEQUEL (dont il est le descendant), QUEL ou MySQL est un gestionnaire de base de données libre. Il est très utilisé dans les projets libres et dans le milieu industriel. [7]

Requêtes SQL dans JDBC

Les requêtes SQL sont représentées dans JDBC à l'aide des interfaces Statement, PreparedStatement et CallableStatement. Chacune de ces interfaces est spécialisée dans un type particulier de requêtes,

Une fois obtenue auprès d'une Connection, une requête peut être exécutée, son résultat prenant la forme d'un ResultSet, dans le cas d'un SELECT. Ce résultat peut alors être vérifié (valeur NULL, ensemble vide, etc.) et parcouru ligne par ligne. Les ordres DELETE, INSERT, UPDATE quant à eux retournent un entier.

La méthode executeQuery() exécute une requête et retourne le résultat sous forme d'un ResultSet. Un ResultSet est un ensemble de lignes (chaque ligne représentant un tuple de la relation résultat); chaque ligne comporte le même nombre de colonnes (chaque colonne représentant un attribut de la relation résultat). [8]

III. Présentation de l'application

Notre projet consiste à réaliser une application Client-Serveur par Java RMI, dans un réseau locale, cette application est une application de Tchat : Messagerie instantanée

c.-à-d échange des messages textuels en temps réel.

Pour réaliser ce projet on a crée les classe suivante :

1. Coté serveur :

- La définition de l'interface Notify qui contient les méthodes nécessaires pour synchroniser les échanges des messages entre les clients :

```
public void joinMessage (String name) throws RemoteException;
public void sendMessage(String name, String message) throws RemoteException;
public void exitMessage (String name) throws RemoteException;
public String getName() throws RemoteException;
public void setName(String s) throws RemoteException;
```

- La création de la classe Message qui implémente l'interface Notify, i.e. contient l'implémentation des méthodes déclarées en Notify.
- La définition de l'interface ChatInterface qui contient les méthodes qui peuvent être invoqués à distance :

  public void rejoindre(Notify n) throws RemoteException;

  public void parler(Notify n, String s) throws RemoteException;

  public void laisser(Notify n) throws RemoteException.

- La création de la classe ChatServer, dans cette classe on a regroupés l'implémentation de l'interface ChatInterface, et la création du serveur :

  Tous d'abord on a crée une liste des collections nommé ThreadList de type ArrayList pour enregistrer les objets de type Notify.

  Dans les méthodes rejoindre on a parcourir tout la liste ``ThreadList`` pour envoyer le nom de nouveau client (de type Notify) à tous les autres clients connectées, dans cette méthode on fait appel à la méthode joinMessage.

  On fait la même chose pour les autres méthodes sauf que, pour la méthode parler on fait appel à la méthode sendMessage, et pour la méthode laisser on fait appel à la méthode exitMessage.

  Pour le serveur :

- La création de rmiregistry
- La creation de l'objet server et l'enregistrer auprès rmiregistery avec la méthode rebind, cet objet est une instanciation de la classe chatServer.

```
ChatServeur server = new ChatServeur();
Naming.rebind("rmichat", server);
```

Dans le but que chaque client entrant le tchat doit savoir les membres du groupe participant à la discussion, on a crée une base de données sur lequel on peut enregistrer les pseudos des clients, l'insertion des pseudos se fait à partir de l'interface principale de client.

Figure II.3: Base de données

Apres la création de base de donnée avec l'outil EasyPHP MySQL, on a configuré et connecter l'outil JDBC/ODBC avec la base qu'on a créé, tout a fait on utilise La classe TestConnectionMySQL qui contient trois méthode :

AfficherTousLesPersonnes : dans cette méthode on a exécuté des requêtes SQL qui retourne un résultat de type Resultset: ce résultat retourne une liste des pseudos enregistrés dans la base de données.

InsérerPersonne : permet d'insérer le pseudo dans la liste.

SupprimerPersonne : permet de supprimer le pseudo de la liste.

Lorsque on connecte la Base de Donnée avec Netbeans 6.9.1,on peut voir la mise à jour effectuée a notre table comme ci-dessus :

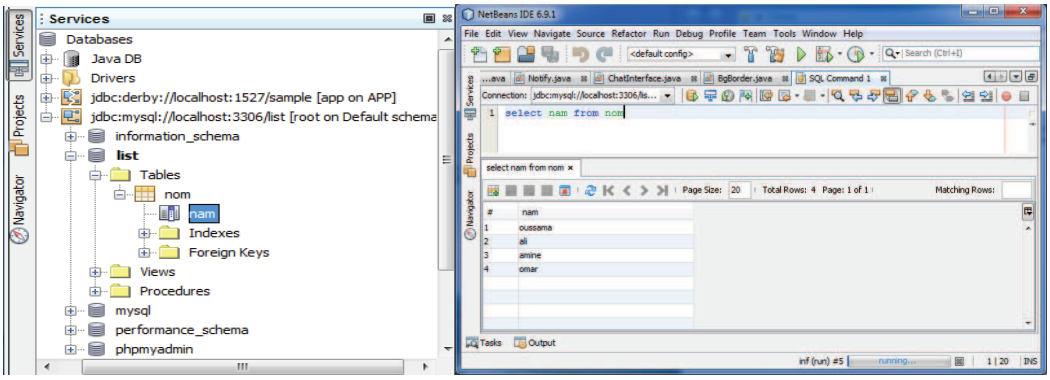

Figure II.4 : Table de client sous Netbeans.

2. Coté Client :

– la définition de l'interface ChatInterface qui contient les méthodes nécessaire pour invoquer les méthodes de l'objet distant.

- La création de la classe ChatClient qui permette d'accéder à des méthodes de l'objet distant on utilisant la méthode Lookup :

```
Remote remoteObject = Naming.lookup("rmichat");
        if (remoteObject instanceof ChatInterface) {
            chatServer = (ChatInterface)remoteObject ;
        displayChat = new Message(otherText,model);  }
```

- La création de l'interface home machine on utilisant AWT et Swing .
- L'instanciation et création d'un objet de la classe TestConnectionMuSQL pour afficher liste des clients participant à la discussion public.

IV. Fonctionnement de l'application

Pour exécuter notre application, il faut d'abord lancer le serveur et EasyPHP de coté serveur, et après lancer les clients.

Nous présentons l'interface de notre application de Tchat, nommée Tchat@Tlemcen.

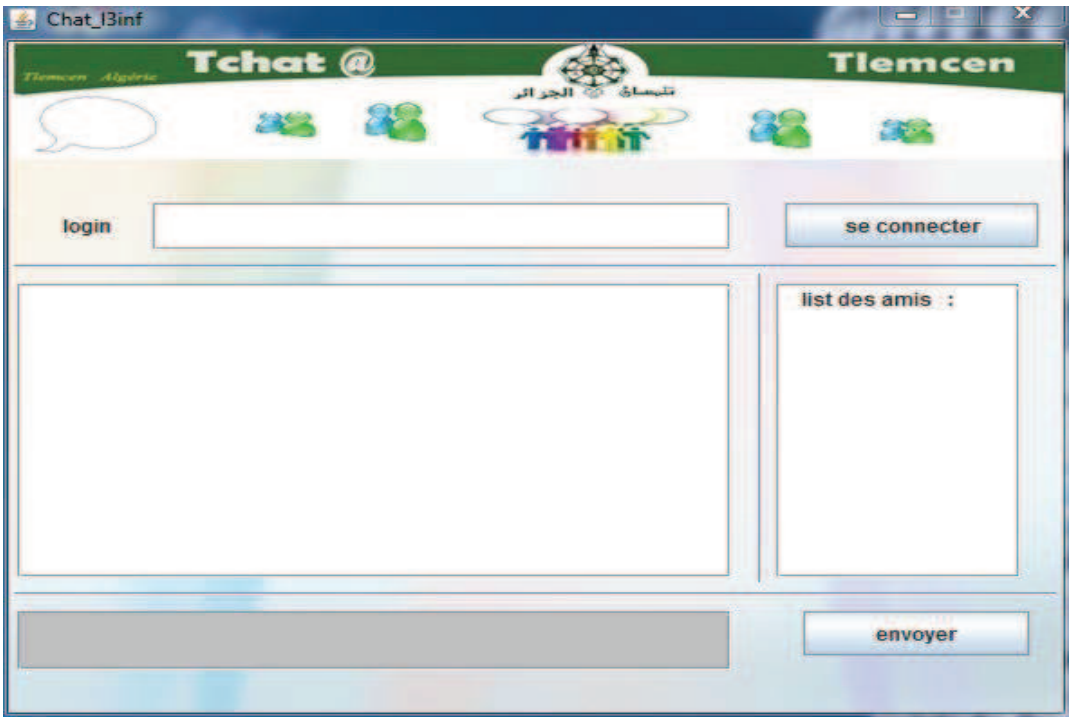

Figure II.5 : L'interface Principale

1. L'utilisation

Pour se connecter, il faut d'abord avoir un login, c'est le nom sur lequel, on peut joindre le tchat, Dans la Zone de texte, on donne notre pseudo de tchat.

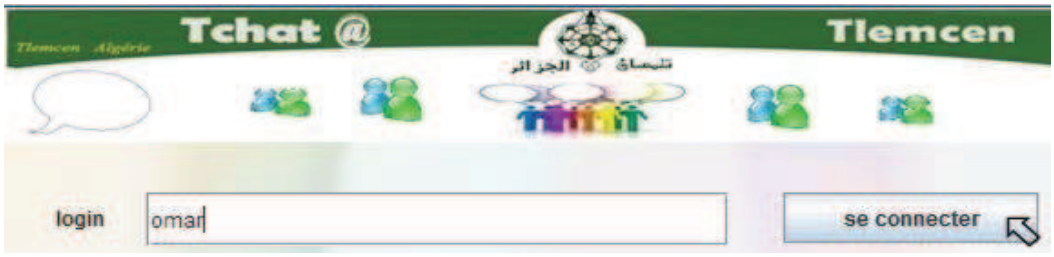

Figure II.6 : Entrer le login

Une fois vous êtres connectés, votre nom apparait dans la liste chez toutes les clients et vous pouvez joindre le groupe de Tchat pour envoyer et recevez des messages. Votre pseudo sera affiché au dessus de la fenêtre avec l'heure de connexion.

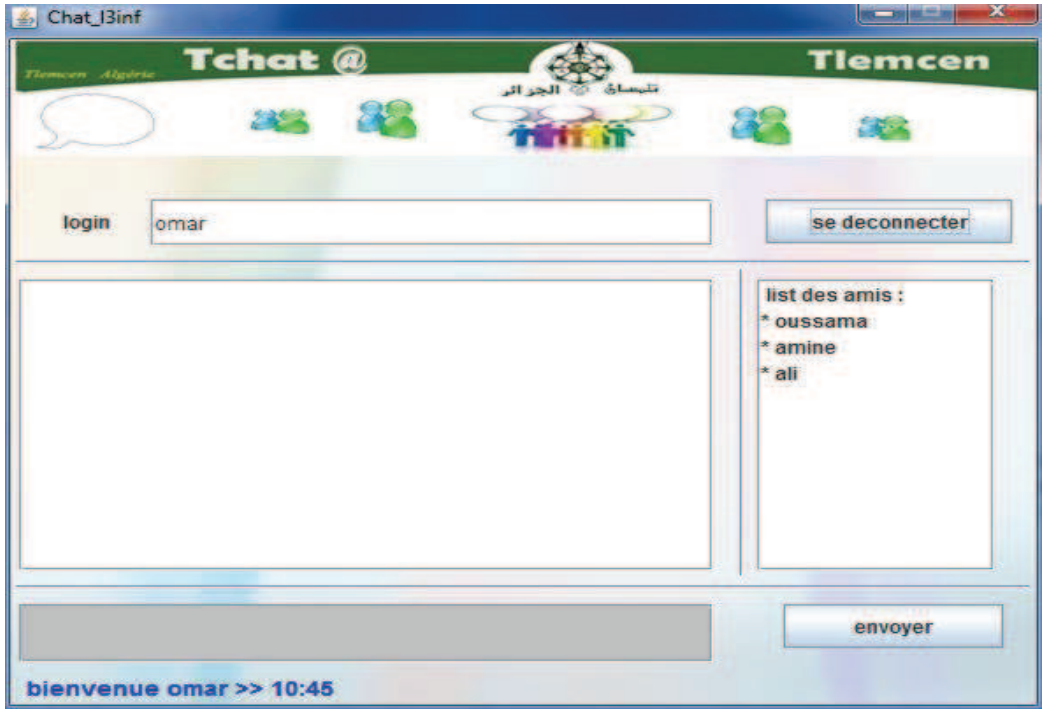

Figure II.7 : Omar se connecte au Tchat

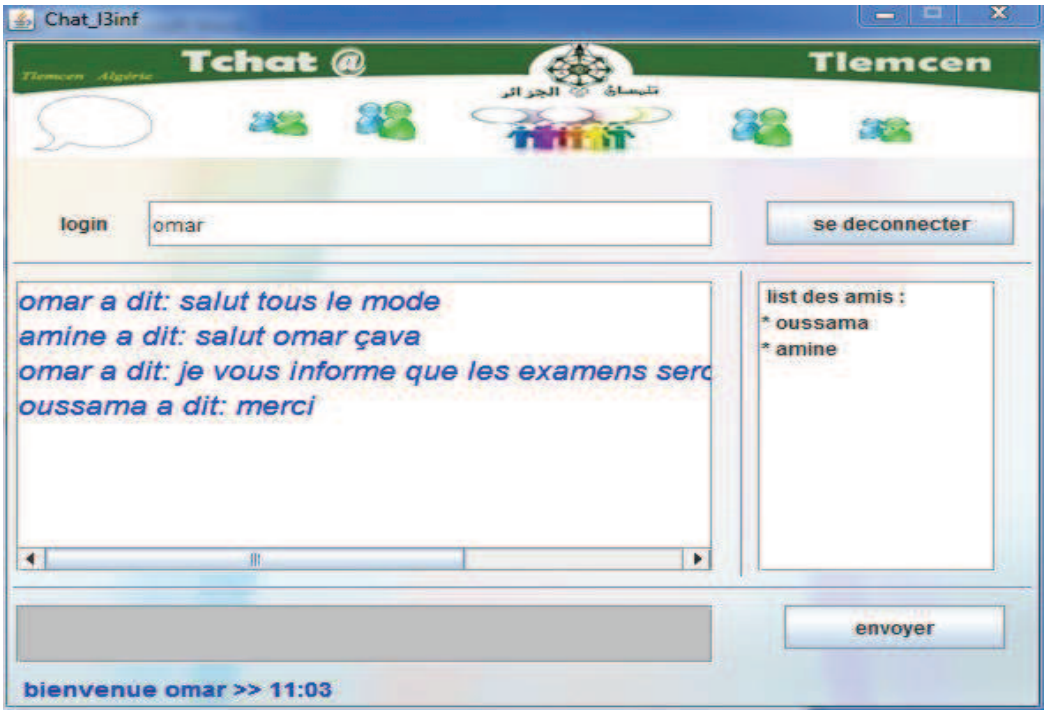

Figure II.8 : discussion 1

Figure II.9 : discussion 2

Pour se déconnecter et quitter la discussion on clique sur le bouton ‘ se déconnecter ‘.

Figure II.10 : déconnexion

Lorsqu’on clique sur l’icone de fermeture de la fenêtre, une boite de confirmation s’affiche come ci-dessus.

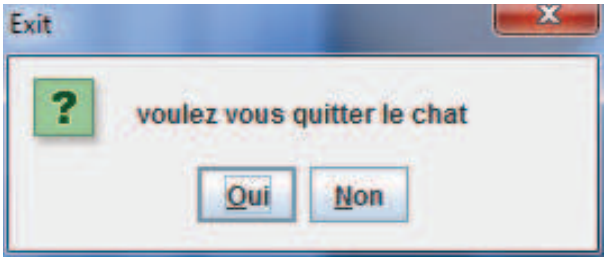

Figure II 11: déconnecter

V. CONCLUSION

Nous avons montrés dans ce dernier chapitre l’interface et les taches que nous avons accomplies dans notre projet de TCHAT.

Conclusion

Ce projet nous a donné la chance de découvrir les différentes architectures distribuées, en particulier l'architecture de type client/serveur et de se familiariser aussi avec l'outil Java et en particulier avec le middleware java RMI.

Nous avons appris comment utiliser l'architecture client /serveur avec la méthode Java RMI pour développer notre application de tchat i.e. la messagerie instantanée dans un réseau local avec l'intégration d'une base de données MySQL.

Dans ce rapport on a présenté au début les supports nécessaires au fonctionnement des communications, ensuite on a abordé les différentes étapes du développement de notre application sous l'IDE Netbeans.

.

BIBLIOGRAPHIE

Sites web:

[1] http://www.Commentçamarche.net/cs.htm, Avril 2013

[2] Java Remote Method Invocation (Java RMI) http://java.sun.com/products/jdk/rmi/

[3] Badr Benmammar, Introduction aux objets répartis Java RMI, Cours M2 RSD (Master Réseaux et Systèmes Distribués). 2013.

[4] Chap.-12- Le client-serveur.doc.pdf http://www.framasoft.net/article3991.html

[5] NetBeans - Wikipédia.htm.

[6] **http**://www.JDBC.pdf.com juin 2013

[7] Structured Query Language-Wikipédia.htm.

[8] Développons en Java-JDBC (Java DataBase Connectivity).htm

Theses:

[9] Ghefir Mohamed El Amine et Bendoukha Sidi Ahmed, Développement d'une Application Distribuée par la méthode java RMI, Juillet 2006.

Résumé

Ce projet a comme objectif de développer une application client /serveur par la méthode Java RMI, nous avons réalisé un serveur de messagerie instantanée «**TCHAT**» connectant plusieurs clients à la fois.

Cette application permet l'échange des messages, éventuellement, entre plusieurs ordinateurs connectés au même réseau local ou internet. Ce moyen de communication est caractérisé par le fait que les messages s'affichent en temps réel.

Abstract

The object of this project is to develop an example of client/server application model using Java RMI. We have performed an instant messaging server named «**CHAT**» connecting multiple customers of the same organism.

This application allows the instant exchange of text messages between computers connected to the same local network or Internet.
This way of communication is characterized by the fact that messages are displayed in near real time.

ملخص

الهدف من المشروع هو تطوير تطبيقات العميل الزبون / الخادم. أجرينا خادم التراسل الفوري اسمه « الدردشة » التي تربط الزبائن من الهيئة نفسها.

هده العملية الفورية تسمح بالتبادل اللحظي للرسائل النصية بين أجهزة الكمبيوتر الموصول بنفس الشبكة أو الانترنت,تتميز هده الوسيلة الاتصالية بميزة عرض الرسائل في وقت شبه حقيقي .