Rapport de Conception :

Tchat Client – Serveur

« Talk A Lot »

Le 17/11/2014

Par BIABIANY Emmanuel

Sommaire

Introduction

Ce document est le rapport de conception du projet de *« Système Avancé »* encadré par Mr Audrey ROBINEL, et réalisé par Mr BIABIANY Emmanuel.

Ce projet consiste en la réalisation d'un programme *Client – Serveur multithread* avec utilisation de *sockets*.

La conception et la réalisation du projet a duré environ un mois, et se termine par sa présentation le mercredi 19 Novembre 2014 dès 9h30.

Ce document cherche a explicité la conception du projet en définissant le protocole de communication ainsi qu'en expliquant le contenu du programme C TAL serveur et de l'application JAVA TAL client.

I. Protocole de communication

Nous avons défini un protocole de communication modulable entre le serveur et les clients de façon à savoir précisément si le message est pertinent ou non.

Pour commencer le message doit être typé, c'est en fait le première argument du message ceux qui suivent dépend entièrement de celui-ci.

Il existe deux types de typage et une fonction de formatage :

A. Le typage d'entré

Les messages envoyés par les clients au serveur. Voici la liste de ces types :

- ins : permet l'inscription d'un client.
  (ex : ins M NOM Prénom pnom motdepasse).
- d_ins : permet la suppression du compte du client.
- co : permet la connexion d'un utilisateur.
  (ex : co pnom motdepasse)
- d_co : permet la déconnexion d'un utilisateur.
  (ex : d_co pnom motdepasse)
- l_amis : permet à l'utilisateur de demander la liste des utilisateurs connectés. *(ex : l_amis pnom)*
- msg : permet d'envoyer un message à un autre utilisateur. *(ex : msg admin pnom Bienvenue sur Talk A Lot !!)*

B. Le typage de sortie

Les messages envoyés par le serveur aux clients. Voici la liste de ces types :

- info : renvoie un message de confirmation ou à titre informatif. *(ex :info util_co Prénom ou info mgs_delived)*
- err : renvoie un message d'erreur.
  (ex :err server_shutdown ou err mdpUtil_false)
- l_amis : renvoie la liste des utilisateurs connectés.
  (ex : l_amis ebiabian arobinel dpuzenat)

C. La fonction de formatage

La fonction de formatage est présente dans les deux programmes. Elle se compose principalement d'un *switch* et intervient au moment de la réception d'un message, le schéma suivant la résume assez bien :

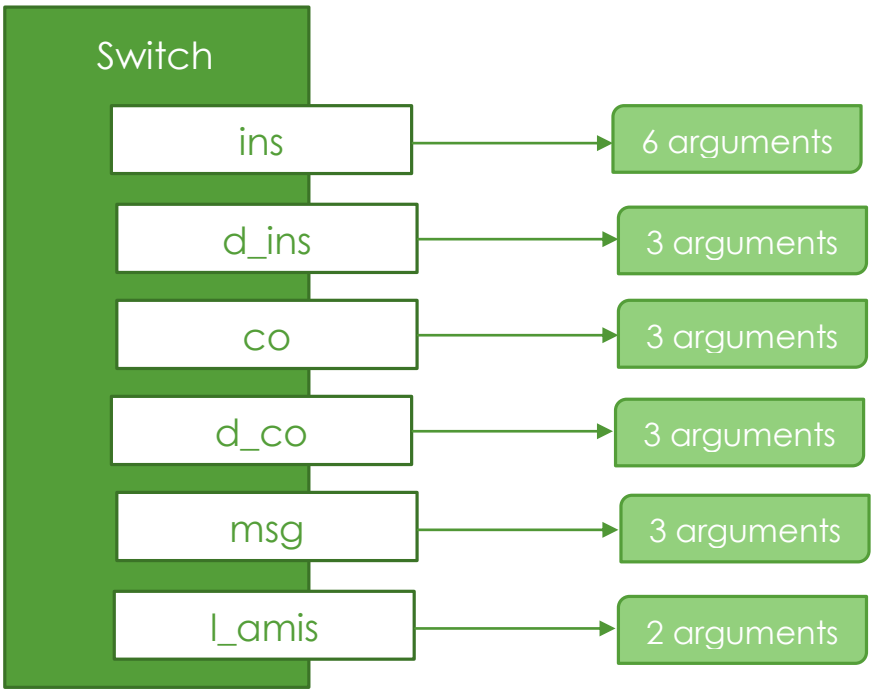

Figure 1: Représentation de la fonction formatage

II. Serveur en C

Le serveur est un programme C et *le thread principal (main)* de celui-ci commence par charger en mémoire la liste des clients puis celle des messages archivés puis il réserve le *port* choisi par l'administrateur *(par default c'est 3000)*.

Ensuite le programme passe en mode *« running »* et cela se traduit par le fait que l'on entre dans une boucle contenant un *accept*, le lancement d'une *routine* ainsi que de multiples tests.

Voici le schéma de classe modélisant les données utilisé par le système :

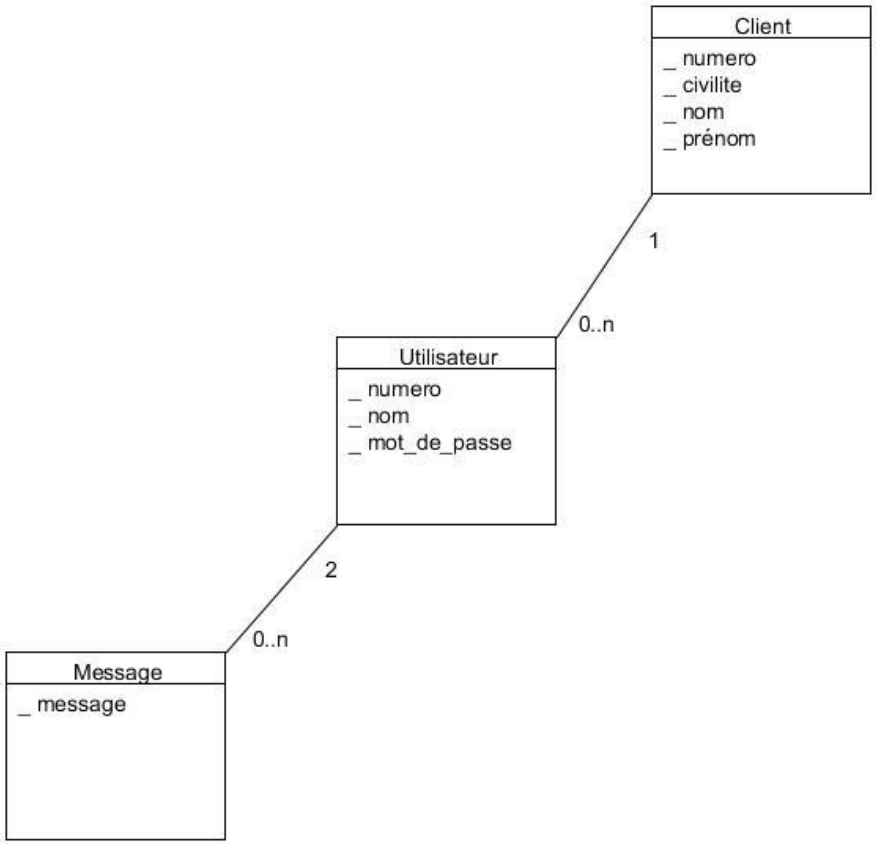

Figure 2: Diagramme de classe

A. La routine « Routine_Recieve »

La routine fait un *recv* de ce qui se trouve dans le socket du client, puis la fonction de formatage intervient et détermine le type du message.

Après le formatage du message la routine détermine les actions à exécutées pour répondre à la requête du client.

Quand la routine reçoit la commande « exit_c » ou alors -1 dû à une déconnexion brutal du client, elle lance une procédure de déconnexion dit « propre » de l'utilisateur en question avant de s'arrêté.

Quand la routine reçoit la commande « exit_s » elle vérifie que l'émetteur de la requête est bien l'administrateur, puis elle procède à l'arrête en douceur du serveur.

B. Les fonctions

Les fonctions sont multiples, on va donc les classées par famille :

- Les fonctions de vérification reconnaissable par le préfixe *Verif*.
- Les fonctions de communication reconnaissable par le préfixe *Send*.
- Les fonctions accesseur de donnée reconnaissable par le préfixe *get*.
- Les fonctions de gestion de la liste des utilisateurs connectés reconnaissable par le suffixe *Amis*.
- Les fonctions de sauvegarde et de récupérer en mémoire liste de données utilisé par le système reconnaissable par le suffixe *Data*.

Ensuite on la fonction *« StopServeur »* qui intervient lors de l'arrêt du serveur demandé par l'administrateur.

III. Client en JAVA

Le client est un programme JAVA composé d'un *thread principale* et un *thread secondaire*.

Le thread principale récupère dans un fichier *(ServerLocate.dll)* la localisation du serveur puis lance l'interface graphique. Ensuite il fait un *connect* pour communiquer avec le serveur.

Le *thread secondaire* lui fait un *read* sur le socket. Quand il reçoit un message il utilise la fonction de formatage qui détermine le type message dont il s'agit.

Voici le schéma de classe modélisant les données utilisé par le système :

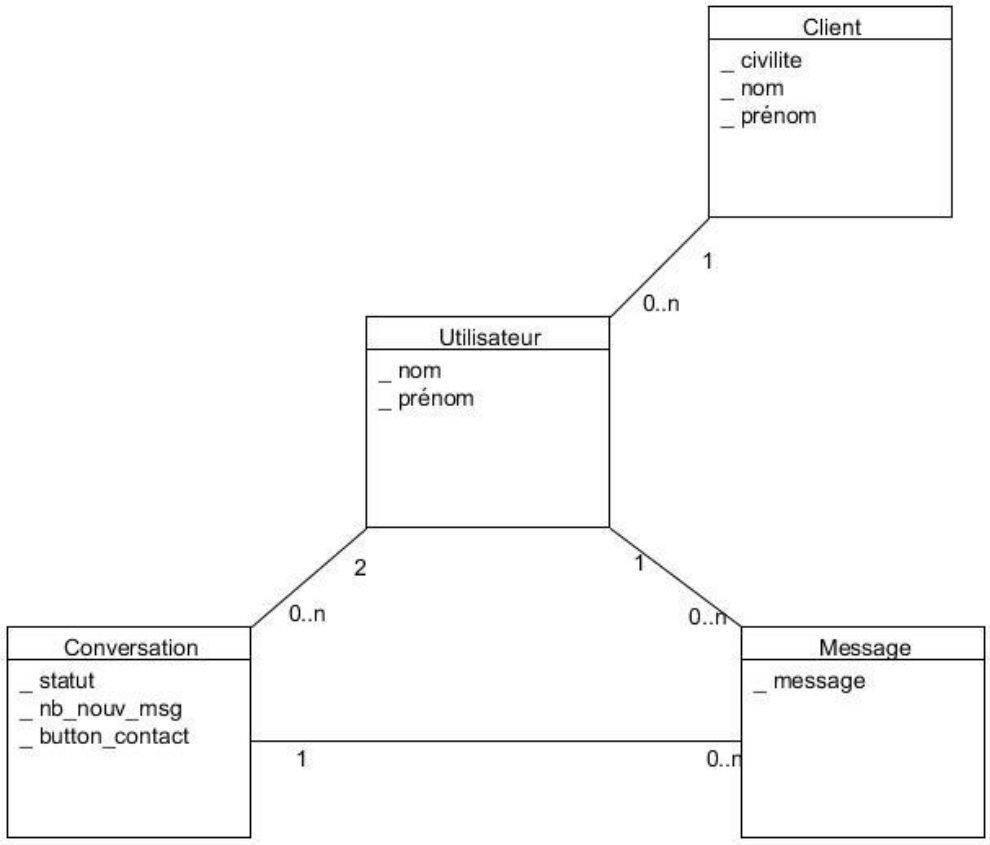

Figure 3: Diagramme de classe

A. Package « tal »

Les classes présentes dans ce package sont principalement mis en avant dans le diagramme de classe *(figure3)*, ne reste que la classe *Interface* qui génère l'interface graphique et la classe *Application* qui est en fait le *thread principale (main)* qui n'y figure pas.

B. Package « tal.gestion »

Les classes présentes dans ce package sont la classe *Reception* qui est en fait le thread secondaire qui gère l'arrivé des messages envoyés par le serveur ainsi que la classe *Notification* qui est un thread et qui déclenche un effet sonore lors de la réception d'un message de type *« msg »*.

C. Package « tal.graphics »

Les classes présentes dans ce package sont tous des éléments graphique que j'ai conçu spécialement pour cette application.

Pour commencer les classes *HintTextField* et *HintPasswordField* qui permet d'avoir un texte pré-écrit dans les curseurs et automatiquement effacé lorsque l'utilisateur clique sur le composant.

La classe *BordArrondie* qui permet tout simplement d'arrondir les bords d'un composant.

La classe *Fond* qui gère l'affichage et les transitions de du fond de la fenêtre ainsi que la transparence de celle-ci.

La classe *Contact_Button* qui affiche et gère indépendamment du reste de l'affichage, le statut de la conversation et le nombre de nouveau message reçu.

La classe *Space_Converse* qui est enfait un bloc qui affiche et gère indépendamment du reste de l'affichage la conversation en cous ainsi que les messages reçu attribué à cette conversation.

La classe *Message_Bloc* est en fait un bloc qui contient le nom de l'auteur d'un message et du message en question. Celui est créé à chaque fois qu'un nouveau message est arrivé dans la conversation en cours. Il vient se greffer à la liste des messages dans le bloc *Space_Converse*.

Conclusion

Tal A Lot est une application Client-Serveur de tchat utilisant les threads ainsi que les sockets, ce qui fait de lui une application utilisant des moyens de communiquer efficace.

Le design de sa partie client JAVA se veut intuitif afin de le rendre accessible au grand public.

Ce projet est encore en cours de développement..