Compte-rendu TP Réseaux

SOTIR Paul-Emmanuel
CHAPELLE Victoire
B3330

Introduction

Objectifs du TP : Notre mission, pour ce TP qui s'est déroulé sur 3 séances de 4h, était de réaliser un chat interactif que des participants peuvent rejoindre et y envoyer des messages. Deux variantes du chat étaient demandées : une réalisée avec l'interface de programmation RMI et l'autre basée sur les Sockets.
Suivant l'avancement de notre application nous pouvions aller plus ou moins loin dans notre programmation et rendre notre chat plus élaboré.
Une des améliorations possibles était la création d'une interface graphique du chat, qui nous permettra de rendre le chat plus utile et la démonstration plus intéressante.

Conception du système de chat

Nous avons conçu notre chat en choisissant de rajouter une contrainte au cahier des charges : la possibilité de créer des groupes (ou room) de chat. En effet, nous voulions réaliser un système d'échanges de messages plus élaboré afin de répondre au mieux aux attentes que pourraient avoir des utilisateurs d'un chat. Cette amélioration n'a pas été faîte au détriment du cahier des charges initial évidemment.
Les deux variantes de chat réalisées (RMI et Sockets) ont la même interface graphique, si bien qu'aucune différence n'est discernable hors de l'implémentation. En fait, grâce à l'architecture MVC et surtout au fait d'isoler l'accès aux données dans des DAL (Data Access Layer), seul quelques lignes diffèrent entre RMI et socket dans l'ensemble de la vue, des contrôleurs et du model.
Pour les deux réalisations, nous avons créé un historique des messages ainsi que le stockage des utilisateurs et groupes crées. Cet historique est persistant c'est-à-dire qu'il est sauvegardé même si le serveur est coupé.
Les deux variantes ont des DALs asynchrones exécutant les appels bloquants sur des threads à part. La récupération des résultats se fait grâce à des callbacks sous forme de lambdas.

Fonctionnement du chat

A l'ouverture de l'application client, l'utilisateur peut se connecter (Login_view) en utilisant son nom d'utilisateur et son mot de passe. Si nécessaire, il peut créer un compte en naviguant vers une vue dédiée (NewAccount_view). L'utilisateur rentre alors le nom d'utilisateur voulus et le serveur génère automatiquement le mot de passe si le nom d'utilisateur est valide et unique. Chaque nouvel utilisateur est enregistré sur le serveur qui conserve la liste de tous ses clients (sérialisés vers un fichier binaire tous comme les groupes et messages).
Une fois connecté, le client peut ensuite rejoindre ou créer un groupe (room) de chat pour pouvoir communiquer avec les autres. Une fois dans un groupe, il peut envoyer des messages et quitter le groupe quand il le souhaite.

A tout moment le client peut voir quels utilisateurs sont connectés dans le groupe et la liste de tous les groupes créés qu'il peut rejoindre. L'application gère les mises à jour automatiquement (pas de bouton 'refresh' pour actualiser la liste des utilisateurs du groupe ou la liste des groupes).
Voici un schéma montrant approximativement l'architecture du projet (certaines classes visibles à la fois par le serveur et le client dans le package shared sont omises) :

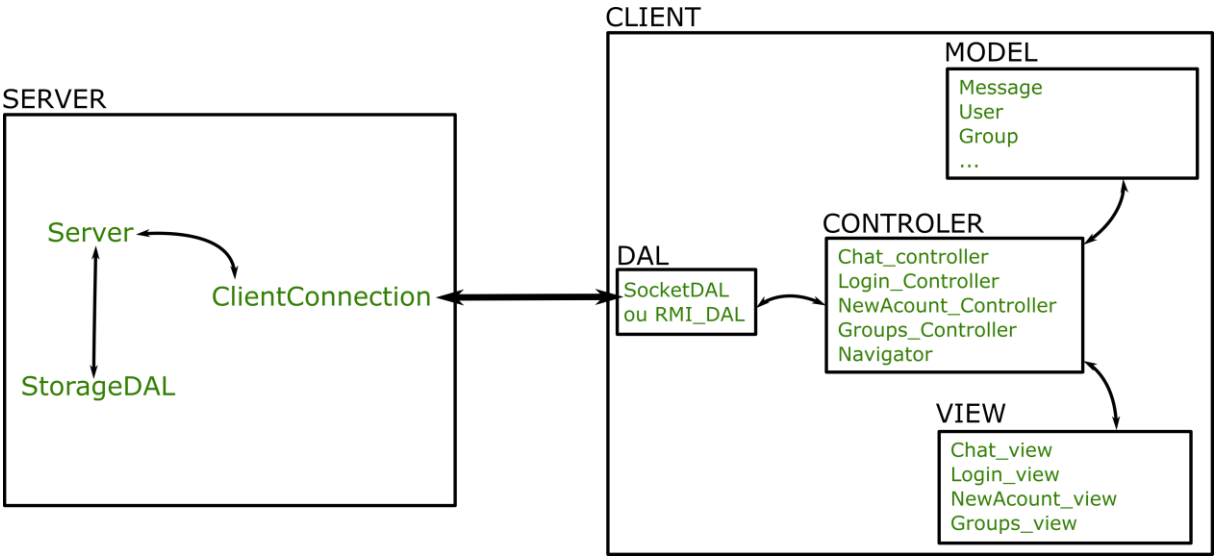

Schéma de l'application

Gestion des erreurs

Notre programme gère les erreurs qui pourraient se produire dans l'utilisation du chat. Par exemple, les erreurs dues à certaines actions de l'utilisateur : création de groupe sans nom, choix d'un nom d'utilisateur ou de groupe déjà utilisé ou connexion à un compte avec un mauvais mot de passe sont impossibles.
De plus, si le serveur se coupe de manière inopinée, l'historique des messages, groupes et utilisateurs est conservé en minimisant les risques de corruption.
Enfin, nous avons fait en sorte que l'état de l'application client reflète l'état du client au sein du serveur. Par exemple, pour quitter un groupe, l'application client attend la confirmation du serveur pour être sûr que le serveur à bien pu enlever l'utilisateur du groupe.

Améliorations possibles

Notre programme répond en parti aux attentes du cahier des charges mais peut cependant être perfectionné.
Nous n'avons pas fini la partie RMI par manque de temps. Une grosse majorité du travail est fait mais il manque cependant l'implémentation du DAL RMI côté client et une partie de l'implémentation du serveur (partie spécifique aux RMI).
Nous pourrions aussi améliorer l'interface graphique du chat en affichant plus de messages d'erreur quand ont entre un nom de groupe qui existe déjà ou quand on crée un groupe sans nom par exemple. Nous pourrions également afficher le créateur du groupe (déjà présent dans le model). Un autre élément du chat qui pourrait être amélioré est la connexion d'un client une fois qu'il a créé son nom d'utilisateur. En effet, à la place de demander le mot de passe de l'utilisateur à chaque fois, nous pourrions le retenir en local dans un fichier.