Programmation d'applications distribuées et en réseau projet 1 : Anagrammes

MARTINOT Romain
STREIGNARD Rémi

Octobre 2020

1 Introduction

Ce rapport décrit le premier projet développé dans le cadre du cours de "Programmation d'applications distribuées et en réseau". Celui-ci avait comme focus le fonctionnement du RMI, l'utilisation des threads, l'intégration de la librairie Swing ainsi que l'implémentation de celui-ci en Java.

2 Choix d'implémentation

Nous avons dû faire plusieurs choix lors de réalisation de ce projet. Ceux-ci sont décrits ci-dessous.

2.1 Structure du projet

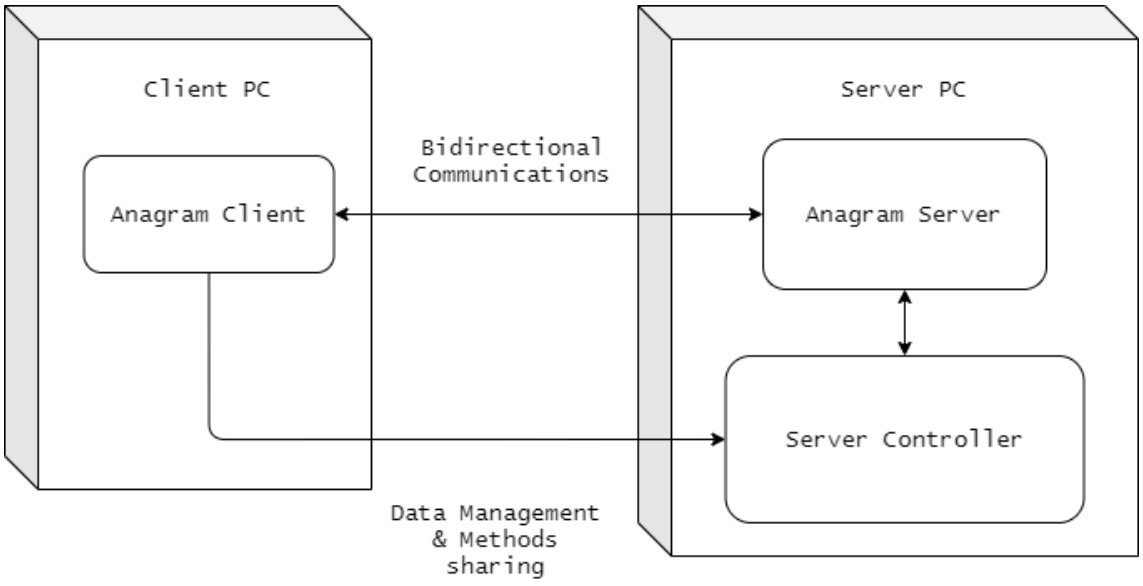

Figure 1: Structure RMI du projet

Dans une structure RMI classique, la communication est à sens unique. Le client, grâce à l'interface qu'implémente le serveur, sait quelles méthodes lui sont disponibles. Ici, il était nécessaire d'établir une connexion bidirectionnelle permettant au client ainsi qu'au serveur de communiquer l'un avec l'autre.

Ainsi, en plus de la structure classique du RMI, nous avons choisi d'adopter un semblant de modèle MVC. Ainsi, le serveur et le client n'interagissent jamais directement et doivent passer par le ServerController.

2.2 Interfaces utilisateur

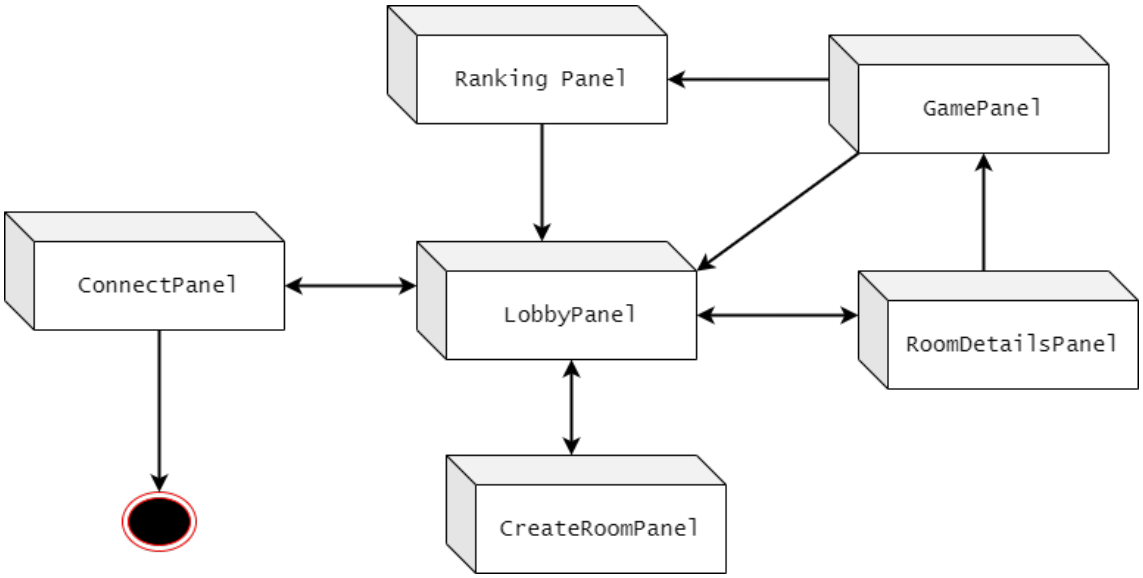

Figure 2: Interactions possibles entre les différentes interfaces

L'application cliente dispose d'une interface composée de plusieurs panneaux qui sont interchangés et chargés dans une fenêtre unique. L'utilisateur peut ainsi se déplacer à sa guise à travers le programme.

2.3 Persistance des données

L'énoncé sous-entendais que dès le lancement du serveur des utilisateurs (joueurs) et des rooms soient déjà créés. Les fichiers players.txt et rooms.txt se chargent de stocker ces structures de données.

De plus, nous avons décidé de stocker les données des joueurs d'une connexion à l'autre. Ainsi, le fichier players.txt verra son contenu modifié au fil des parties des joueurs.

3 Compilation

L'utilisation d'un fichier makefile permet de compiler les différentes parties du projet. Voici, ci-dessous, les règles de compilation et d'exécution qu'il contient :

- cc-all : Permet la compilation de tout le projet, autant de la partie cliente que seveur.
- cc-server & cc-client : Permet la compilation de la partie spécifiée.
- run-server & run-client : Permet l'exécution de la partie spécifiée, une fois compilé. L'adresse du serveur peut être donnée en argument au client. e.g. : IP="192.168.0.5"
- java-doc : Permet la création de la documentation propre à chaque partie.
- clean-java & clean-class : Permet la suppression des fichiers non désirés.

4 IDE et environnement de développement

Nous avons tous les deux utilisés l'IDE IntelliJ pour développer sur nos machines Windows.

Afin de vérifier le bon fonctionnement du makefile ainsi que celui du programme, un environnement Linux a été déployé. Celui-ci a été déployé en tant que machine virtuelle sur un environnement Debian 10. Aucun problème n'a été constaté, mise à part l'intégration des polices.

5 Difficultés rencontrées

Plusieurs difficultés ont été rencontrées tout au long du projet. La plus importante reste néanmoins la persistance et la bonne synchronisaton des données à travers les différentes structures de données (ArrayList, HashMap; etc).

Ceci est expliqué par le fait que parfois, Java passe les objets par copie (ex : un getter) et parfois non (une méthode). L'utilisation d'un SGBD et par extension la centralisation des données aurait ainsi permis de pallier à ce problème tout en en facilitant la gestion, en plus d'y apporter un meilleur contrôle lors des accès concurrents.