Application de Chat

Shane Moissonnier Théo Hermitte

Février 2022

Table des matières

1 Introduction **1**

2 Architecture du système **2**
2.1 Le serveur 2
2.2 Le client 2

3 Structure du code **3**
3.1 Le package *remoteInterfaces* 4
3.2 Le package *common* 4

4 Instructions d'utilisation **4**
4.1 Compilation et exécution 4

5 Dépôt github du projet **4**

1 Introduction

L'objectif de ce TP était d'utiliser l'API Java RMI pour concevoir une application de chat instantané. Nous devions concevoir le côté serveur et le côté client de l'application.

Nous avons implémenté l'entièreté des fonctionnalités exigées. Nous avons également implémenté une fonctionnalité optionnelle. Voici un récapitulatif des fonctionnalités que notre application propose :

— Les clients peuvent se connecter et se déconnecter.
— Les clients peuvent s'échanger des messages.
— Le serveur propose un historique des messages, ce qui permet à un nouveau client de consulter les messages qui ont été échangés avant son arrivée.
— L'historique des messages est persistant, il peut donc être retrouvé après un redémarrage du serveur.
— Le côté client de l'application propose une interface graphique.

2 Architecture du système

2.1 Le serveur

Le serveur sert d'intermédiaire pour que les clients puissent communiquer entre eux. Il propose un unique objet distant qui implémente l'interface *ChatService*. Le client récupère cet objet via le *registre RMI*, et l'utilise pour communiquer avec le serveur. C'est par l'intermédiaire de cet objet que le client se connecte, se déconnecte et envoie un message.

C'est également le serveur qui est chargé de tenir l'historique des messages, qui le charge lors de son démarrage, et qui le sauvegarde lorsqu'il s'arrête.

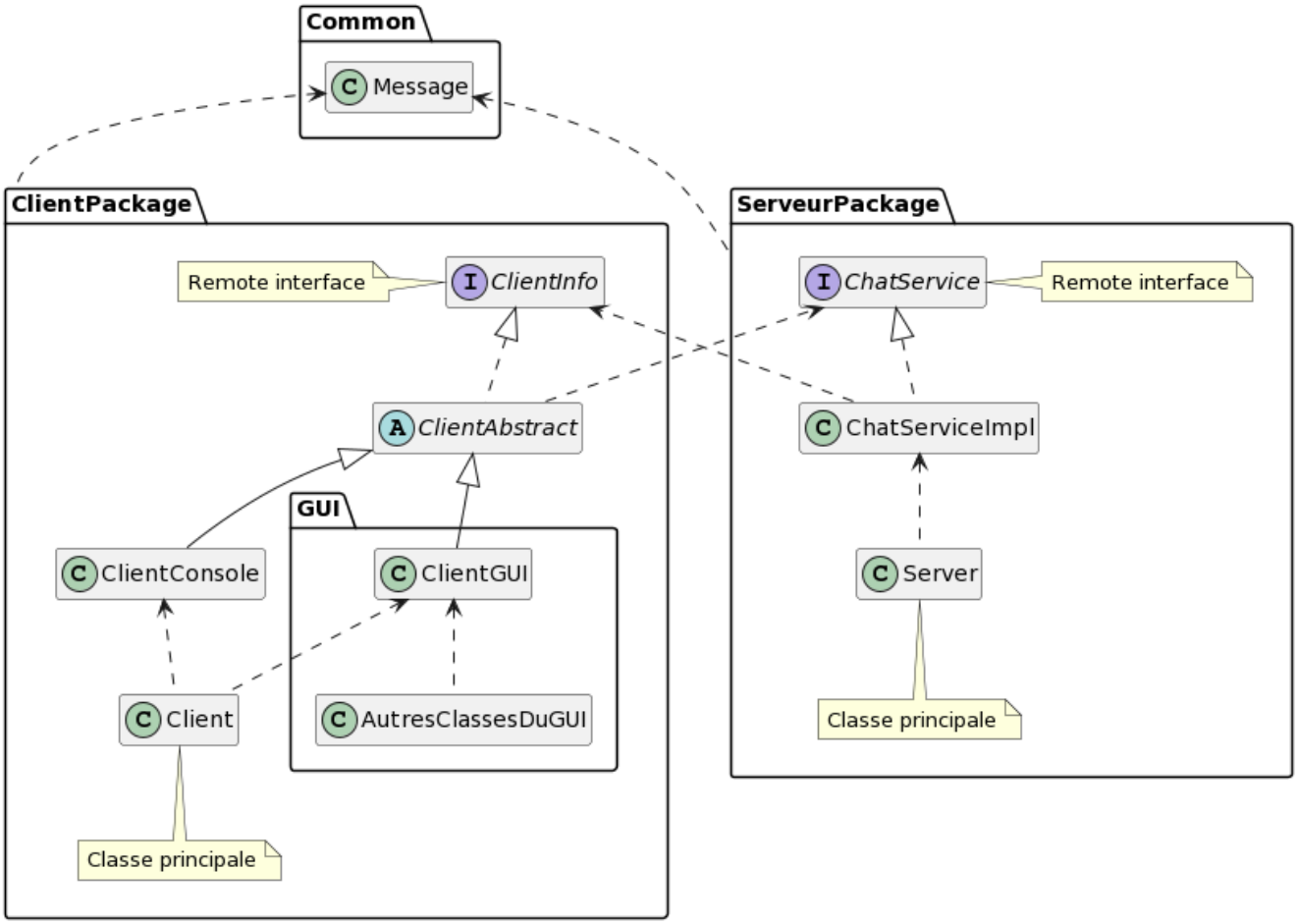

FIGURE 1 – Diagramme représentant l'architecture de notre système

2.2 Le client

Le client représente un utilisateur de l'application de chat. Il récupère l'objet implémentant l'interface distante *ChatService* grâce au *registre RMI*, et l'utilise pour interagir avec le serveur.

Du point de vue du serveur, le client est représenté par l'interface distante *ClientInfo*. L'implémentation de cette interface n'est jamais attachée au registre RMI, mais est transmise au serveur par référence lors de la connexion. Cette interface propose des méthodes de *callback*, permettant au serveur d'informer les clients lorsqu'un évènement se produit. Par exemple, lorsqu'un client envoie

un message au serveur, ce dernier transmet ce message à tous les autres clients connectés via la méthode *messageReceivedCallback* de l'interface *ClientInfo*.

On remarque que l'interface distante *ClientInfo* est implémentée deux fois (par les classes *ClientConsole* et *ClientGUI*). Cela permet de choisir, selon l'implémentation que l'on instancie, si l'on souhaite utiliser l'interface graphique ou l'interface textuelle.

3 Structure du code

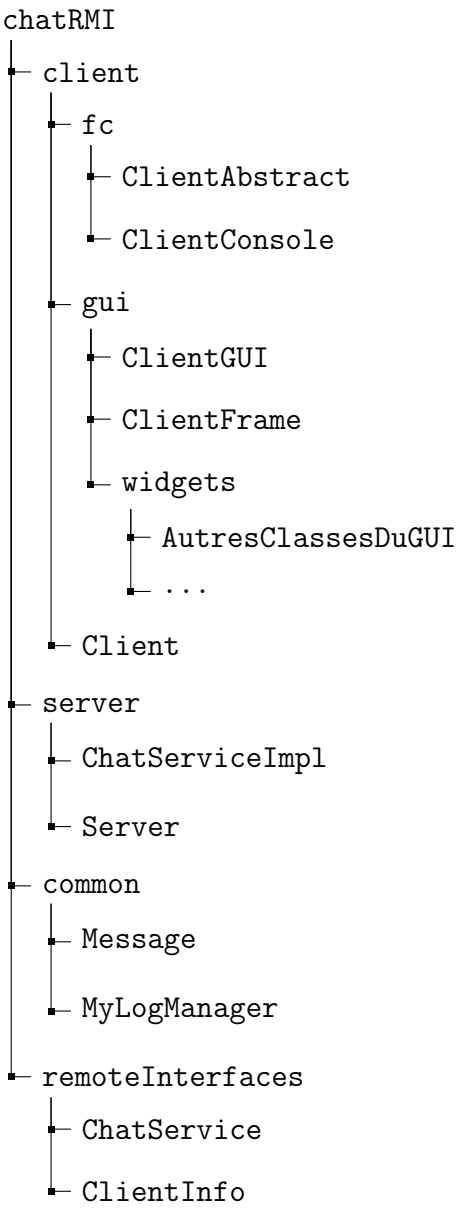

FIGURE 2 – Arborescence du code final

3.1 Le package *remoteInterfaces*

La figure 2 montre l'arborescence du code telle qu'elle est dans le produit final. On peut voir que, contrairement à ce qu'on observe dans le diagramme UML en figure 1, toutes les interfaces distantes (*ChatService* et *ClientInfo*) ont été sorties dans un package *remoteInterfaces*. Nous avons fait cela pour permettre une meilleure séparation du serveur et du client.

En effet, les interfaces distantes mises à part, le client et le serveur sont parfaitement indépendants l'un de l'autre. Tout ce que l'un a besoin de savoir de l'autre, c'est comment interagir avec, c'est-à-dire quelles interfaces distantes sont implémentées.

3.2 Le package *common*

Le package *common* rassemble des classes qui sont communes aux deux parties de l'application.

La classe *Message* représente un message textuel envoyé par un client. Comme un message est envoyé par un client au serveur, les deux doivent nécessairement avoir connaissance de la classe *Message*.

La classe *MyLogManager* est une extension de la classe *LogManager*. Nous utilisons *java.util.logging* pour afficher des messages système durant l'exécution de notre application. La classe *MyLogManager* permet de modifier le comportement du logger que nous utilisons lors de l'arrêt de la machine virtuelle Java, afin de garantir un logging fonctionnel jusqu'à l'arrêt complet du système. Puisque cette classe ne concerne que le logging, et n'a aucun impact sur l'aspect fonctionnel ou sur l'aspect distribué de notre application, nous ne l'avons pas faite figurer sur le diagramme en figure 1.

Il n'est nullement nécessaire que cette classe soit parfaitement identique côté client et côté serveur. Nous l'avons uniquement mise dans le package *common* afin d'éviter d'avoir une classe dupliquée dans les deux côtés de notre application.

4 Instructions d'utilisation

4.1 Compilation et exécution

Afin de compiler le projet :

```
$ make clean
$ make
```

Pour lancer le serveur :

```
$ make launch-server
```

Pour lancer un client :

```
$ make launch-client
```

5 Dépôt github du projet

Le projet ainsi que son code source est accessible en ligne à l'adresse suivante :
`https://github.com/ShaneMoissonnier/ChatRMI`