Rendu final de projet:
Apllications réseaux
Partie traitée: Fédération Simple

ABDEL WAHAB Ismail

10 Mai 2020

Sommaire

Chapter 1

Remise en forme du projet

1.1 Langage de programmation choisis

Le livrable intermédiaire est certe fonctionnel, mais ne respecte pas les pratiques de génie logiel, comme je l'aurais espéré. Mon premier but ici à donc été de refactor mon code afin d'y apporter un bon respect de ces pratiques.

Bien évidement, et pour les mêmes raisons que dans le livrable précendent, le langage choisi pour coder reste le Java pour les clients et les serveurs, et le python pour les tests.

1.2 Respect des pratiques de génie lociel

En relisant mon code je me suis apperçu qu'il y avait beaucoup de duplications, ou de code similaire qui pouvait être mis en commun. Il m'est donc venu l'initiative de créer le dossier ***Tools/*** .

Ce dossier contient des classes Java permettant de stocker des données *(Ex: ServerInfo)*, ou alors mettant à disposition des méthodes statiques de traitement de données *(Ex: FileParser.parseCfgFile())*

J'ai pu ainsi grandement alléger mon code, en délégant des actions répétitives à des classes dédiés à ces taches. Mon code est maintenant bien plus clair et je pense qu'il sera plus simple pour vous de regarder cette version que celle du livrable intermédiare.

1.3 Le protocol SERVERCONNECT

Le sujet du projet stipule que nous devons ajouter le protocole **SERVERCONNECT**. C'est donc pour répondre à cette demande que j'ai implémenté le client NormalClient que l'on peut retrouver dans:

`project_java_IRC/src/Clients/NormalClient/`

Ce client doit tout d'abord se connecter à un serveur à l'aide, par exemple, de la ligne suivante:

```
SERVERCONNECT localhost 12345
```

Suite à quoi ce client se comportera comme un SimpleClient, il devra donc par la suite se *LOGIN* puis parler en utilisant le protocol *MESSAGE*.

Chapter 2

Scripts mis à disposition

2.1 Script de compilation

Afin de faciliter la compilation du projet, un script à été ajouté à ce dernier: ***/src/compile_dir.sh***

Après avoir donné en paramètre de ce script bash un dossier, il compilera tous les fichiers ".java" du dossier en question, ainsi que des sous-dossiers.

2.1.1 Compilation du projet

A la reception de mon projet, placez-vous dans:

```
project_java_IRC/src/
```

Et executez le script de la manière suivante:

```
./compile_dir.sh .
```

Cela compilera l'ensemble des classes java du projet.

2.2 Suppression des *.class

Un petit script vous permettera également de supprimer tous les fichiers compilés java:

```
./remove_class_files.sh
```

2.3 Lancer un client ou un serveur

Dans le dossier *src/Clients*(resp. *src/Servers*) vous trouverez le script ***client.sh***(resp. ***server.sh***) l'execution d'un client(resp. serveur) est simple:

```
./client.sh <nom_du_client_à_lancer> [<serveur_port>]
(Exception pour NormalClient qui utilise le protocol SERVERCONNECT)
./client.sh <NormalClient>

./server.sh <nom_du_serveur_à_lancer> [<port_spécifique_à_utiliser>]
```

Les clients se connecterons automatiquement vers "localhost", et le port 12345 si aucun port n'est précisé.
Les paramètres entre crochets sont optionnels.

2.4 Localisation des tests

Les scripts python de tests sont localisés dans:

```
/project_java_IRC/Tests/*.py
```

Une simple execution du script lancera le tests, qui vous révèlerons aussi ce qui est attendu comme affichage durant ce dernier.

Chapter 3

Fédération simple

3.1 Stockage des clients et serveurs connectés

En effet un serveur peut se comporter comme un client par rapport à un autre. Néanmoins il faut faire attention, lors d'une fédération à au moins trois serveurs, qu'un message ne transite pas à l'infini entre ces derniers:

3.1.1 Execution problématique avec trois serveurs

Soit trois serveurs A, B et C.

Situation: Le serveur A reçoit un message d'un client. A renvoie le message à ses clients ainsi qu'aux serveurs B et C, par consequent:

1. Les clients de A reçoivent le message
2. Les serveurs B et C retransmettent le message à leurs clients respectifs
3. Le serveur B(resp. C) NE DOIT PAS transférer le message au serveur C(resp. B). Le message à déjà été donné par A.

C'est ce troisième point qui rend l'impementation un peu plus délicate que prévu.

3.1.2 Solution utilisée

Pour palier à cela, les messages produits et consommés dans des ArrayBlockingQueue seront répartis sur deux HashMap. Une pour les clients et une autre dédiées au serveurs. Dans mon code on retrouvera donc:

```
clientsQueues <- Queues des clients connectés au serveur courrant
serversQueues <- Queues des serveurs connectés à celui-ci
```

C'est grâce à ces structures de données que je peux controller d'une façon plus précise les mouvements des messages du chat.

Si on reprend l'exemple problématique ci-dessus, un message arrivant en A, sera retransmit à toutes les queues des clients de A, ainsi qu'à toutes les queues des serveurs lui sont connectés. Ainsi B et C consommerons une seul fois dans leurs queues le message et le donnerons à lire à leur clients.

3.2 Connexions entre serveurs

A leurs lancement, les serveurs ChatamuFederated se connecterons automatiquement entre eux à l'aide du fichier:

```
/Ressources/peers/peers.cfg
```

qui contient une liste d'ip et de port des serveurs de la fédération.

Mais cela n'est pas suffisant, en effet si A est un serveur seul, puis B se connecte, alors dans ce cas B sera connecté à A mais pas l'inverse!

Donc à **chaque nouvelle connexion** (que ce soit un client ou un serveur) le serveur courrant retentera de se connecter à l'ensemble des serveurs de la fédération **auxquels il n'est pas déjà connecté**.

3.3 Test du chat fédéré (3 serveurs && 4 clients)

Le test a été codé en python et est disponible:

`/Tests/chatamuFederatedTest.py`

Le rôle de ce test est de:

1. Lance trois serveurs sur les ports 12345, 12346 et 12347
2. Lance quatre clients tel que:
   (a) Sur le serveur de port 12345: soit connecté client0 ET client1
   (b) Sur le serveur de port 12346: soit connecté client2
   (c) Sur le serveur de port 12347: soit connecté client3
3. Fait communiquer les clients à l'aide des serveurs fédérés
4. Affiche l'output d'un des trois serveurs ***aléatoirement***

Vous pouvez donc lancer plusieurs fois le test pour bien vous assurer du bon fonctionnement des serveurs.

A noter que la gestion des tempêtes de broadcast n'as pas été implementée, je me suis arrêté juste avant dans la continuité du projet.

Chapter 4

Source alternative et retours sur le projet

4.1 Git du projet

L'ensemble du projet est aussi disponnible sur mon git personnel en cas de probleme vous pouvez y accéder à l'aide du lien suivant:

`https://github.com/ismailAbdelwahab/project_java_IRC`

4.2 Remerciements et fin de projet

Si vous avez du temps libre, je serais intéressé par un court retour par mail sur les points positifs et négatifs de ce projet.

`ismail.ABDEL-WAHAB@etu.univ-amu.fr`

En ces temps complexes, l'enseignement de la matière n'as pas du être de tout repos pour vous. Et c'est pour cela que je tiens à vous remercier.

Malgrès un projet qui n'est pas entièrement aboutit, j'ai grandement appris en essayant par moi-même tout en étant guidé par vos cours, TD et TP de la matière.

Encore merci, en espérant que vous allez bien.