RAPPORT CHATOS

Sommaire

I. Développement de l’application ChatOS

On distingue deux parties dans l’application ChatOS : le client et le serveur.

Client *: Se connecte au serveur ChatOS. Sa représentation est la suivante :*

ClientOS : Classe permettant la représentation d’un client de l’application ChatOS.

Context : Interface représentant les actions possibles pour un client concernant une trame.

PublicContext : Classe représentant le contexte publique d’un client lorsqu’il est au connecté au serveur.

PrivateContext : Classe représentant le contexte privé d’un client lorsqu’il a établie une connexion privée avec un autre client.

CommonContext : Classe abstraite représentant les actions communes du contexte d’un client qu’il soit publique ou privé.

ProcessCommands : Classe représentant les actions d’exécution des commandes d’un client.

ClientFrameVisitor : Classe reprenant le patron Visitor pour les trames côté client. L’intérêt d’utiliser ce pattern permet à une classe extérieure d’être informée du type exact d’une trame. Il va permettre au code source des objets utilisés de ne pas être ajusté constamment.

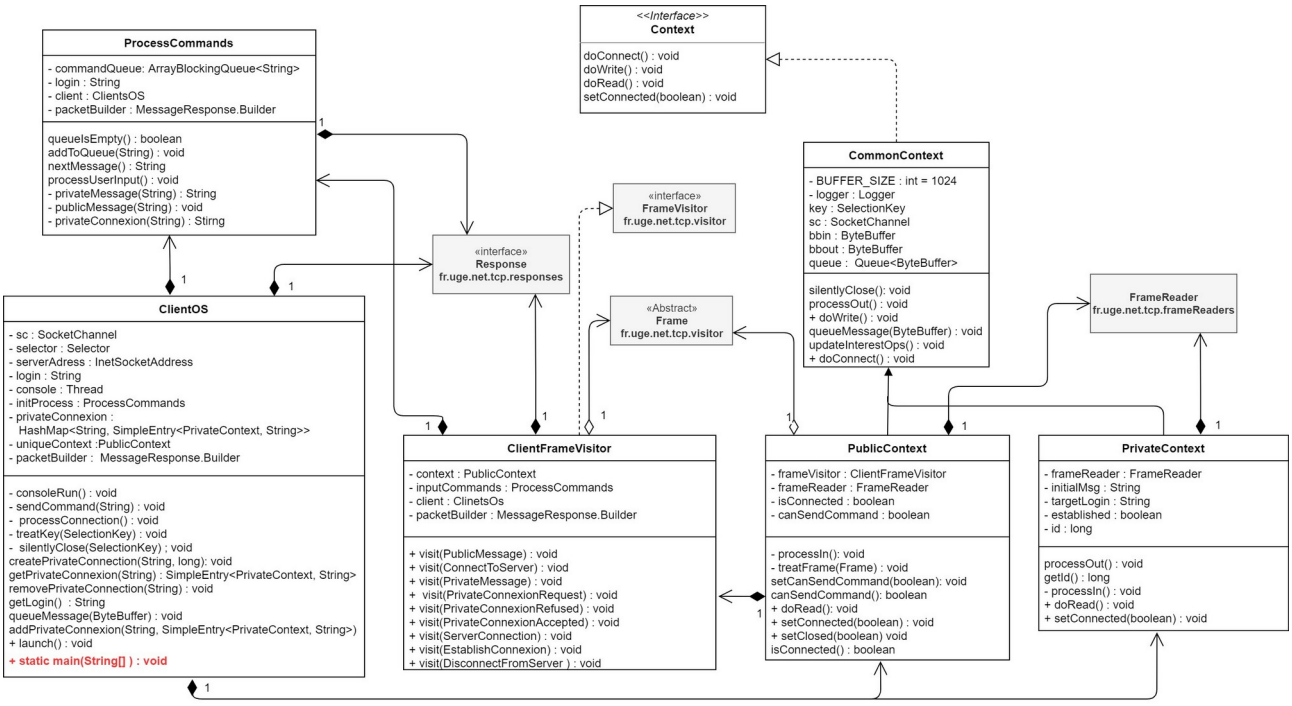

Schéma UML du client ChatOS

Serveur *: Serveur de l'application ChatOS sur lequel des clients peuvent s'envoyer des messages entre eux une fois qu'ils sont connectés. Sa représentation est la suivante :*

ServerChatOS : Classe principale de l'application ChatOS permettant de lancer le serveur.

Server : Classe représentant le serveur de l'application ChatOS.

Context : Classe représentant les actions possibles pour un serveur concernant une trame.

ServerOperations : Classe représentant les différentes actions possibles pour un serveur.

ServerFrameVisitor : Classe reprenant le patron Visitor pour les trames côté serveur. L'intérêt est le même que celui expliqué pour le visiteur côté client.

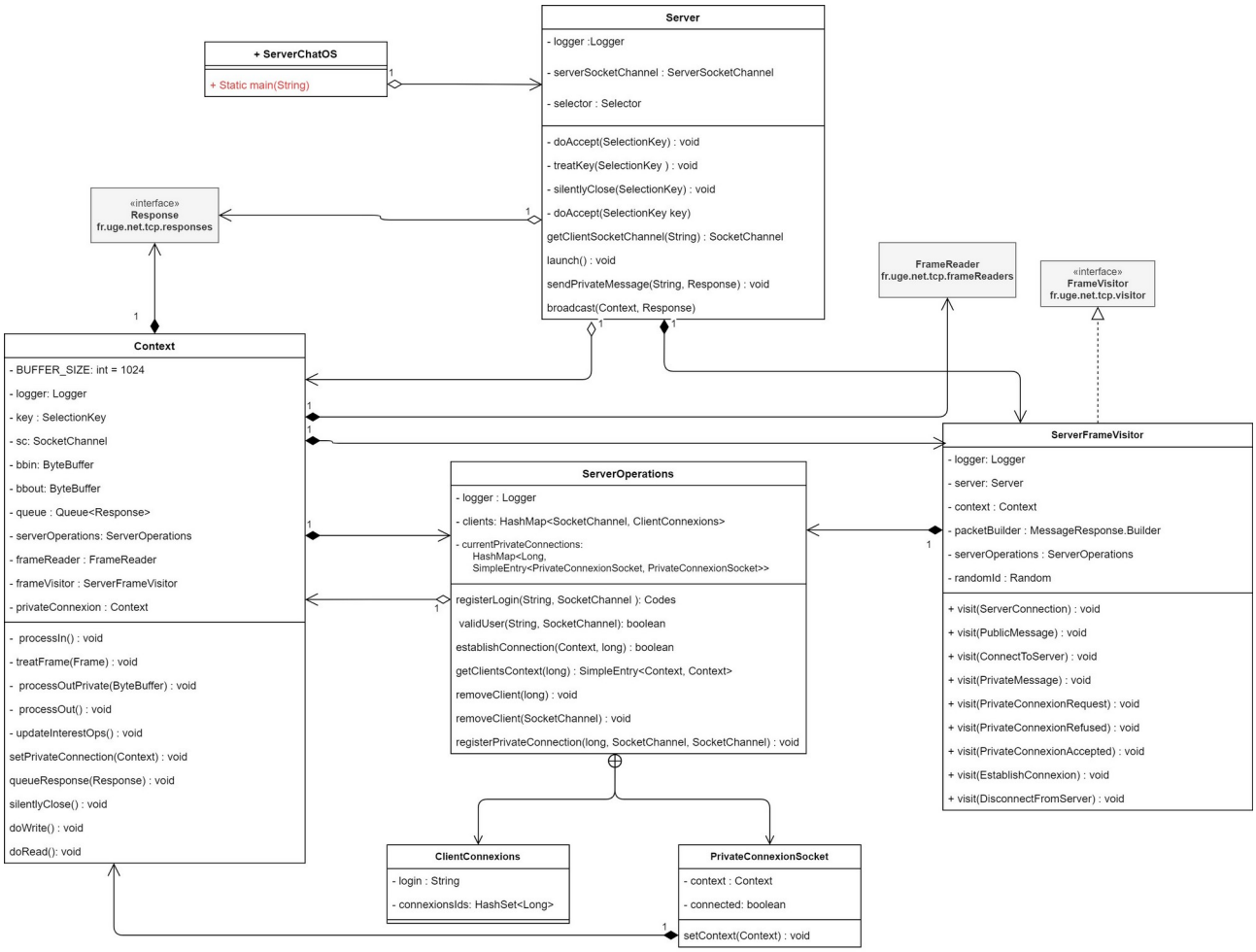

Schéma UML du serveur ChatOS

En dehors des parties principales de l'application ChatOS, nous avons également représenté nos paquets par la classe ***MessageResponse***. Celle-ci implémente une interface Response où les différents codes pour un paquet donné sont représentés. De plus, nous avons pour cette classe utilisée le patron **Builder**. L'intérêt de ce patron pour notre représentation d'un paquet permet de

faire varier les champs à renseigner selon le paquet envoyé, ce que l'on a trouvé intéressant à exploiter car les paquets peuvent être différents selon le code reçu.

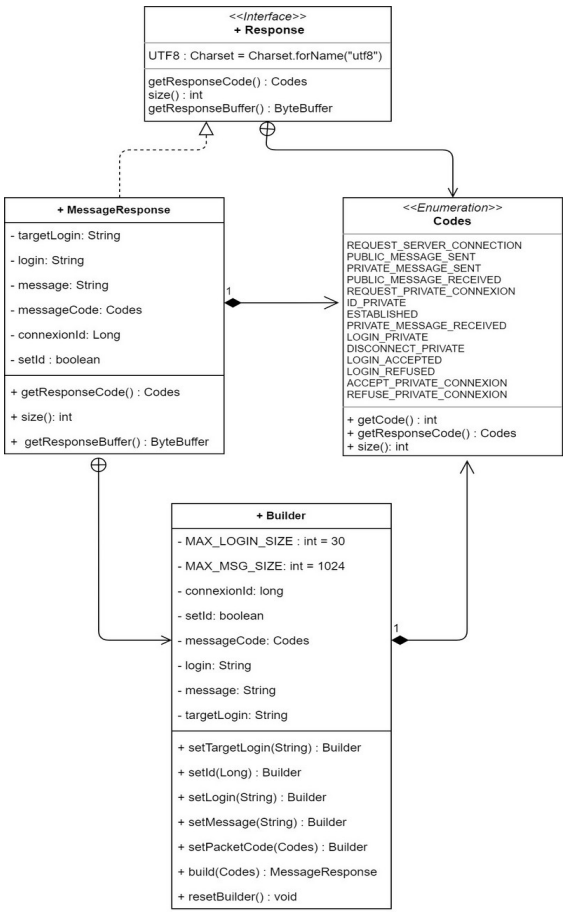

Schéma UML de Response

Un autre point intéressant de l'application ChatOS qui sont les traitements de trames représentées dans notre application par la classe ***FrameReader***. En effet, celle-ci représente l'objet principal concernant les lectures de trames car c'est dans cette classe que la méthode process() permettant le traitement des différents paquets selon le code se trouve. Adapté sous la forme d'un patron **Visitor**, on distingue les différents reader pouvant être utilisé selon le code reçu :

- ***EstablishConnexionReader***
- ***FrameReader***
- ***PrivateConnexionAcceptedReader***
- ***PrivateConnexionReader***
- ***PrivateMessageReader***
- ***PublicMessageReader***

Ces différents reader permettent la lecture des différents objets possibles du *package fr.uge.net.tcp.visitor*. En effet, afin de faire évoluer au mieux l'architecture de notre application ChatOS, nous avons mis en place le patron Visiteur. Trois visiteurs ont donc été mis en place :

- Visiteur pour les clients

- Visiteur pour le serveur

- Visiteur pour le traitement des trames

L'intérêt est le même pour ces trois visiteurs : permettre de connaître l'objet visité avec une méthode visit() et utiliser ses méthodes pour traiter les informations nécessaires grâce à la méthode accept(). Nous avons donc pour chacun les visiteurs respectifs ClientFrameVisitor, ServerFrameVisitor et FrameVisitor.

Les classes ***IntReader, LongReader*** et ***StringReader*** sont également présentes pour les lectures respectives d'un entier, d'un long et d'une chaîne de caractères.
Toutes ces classes implémentent l'interface ***Reader*** attendant un type T pour effectuer le traitement de celui-ci.

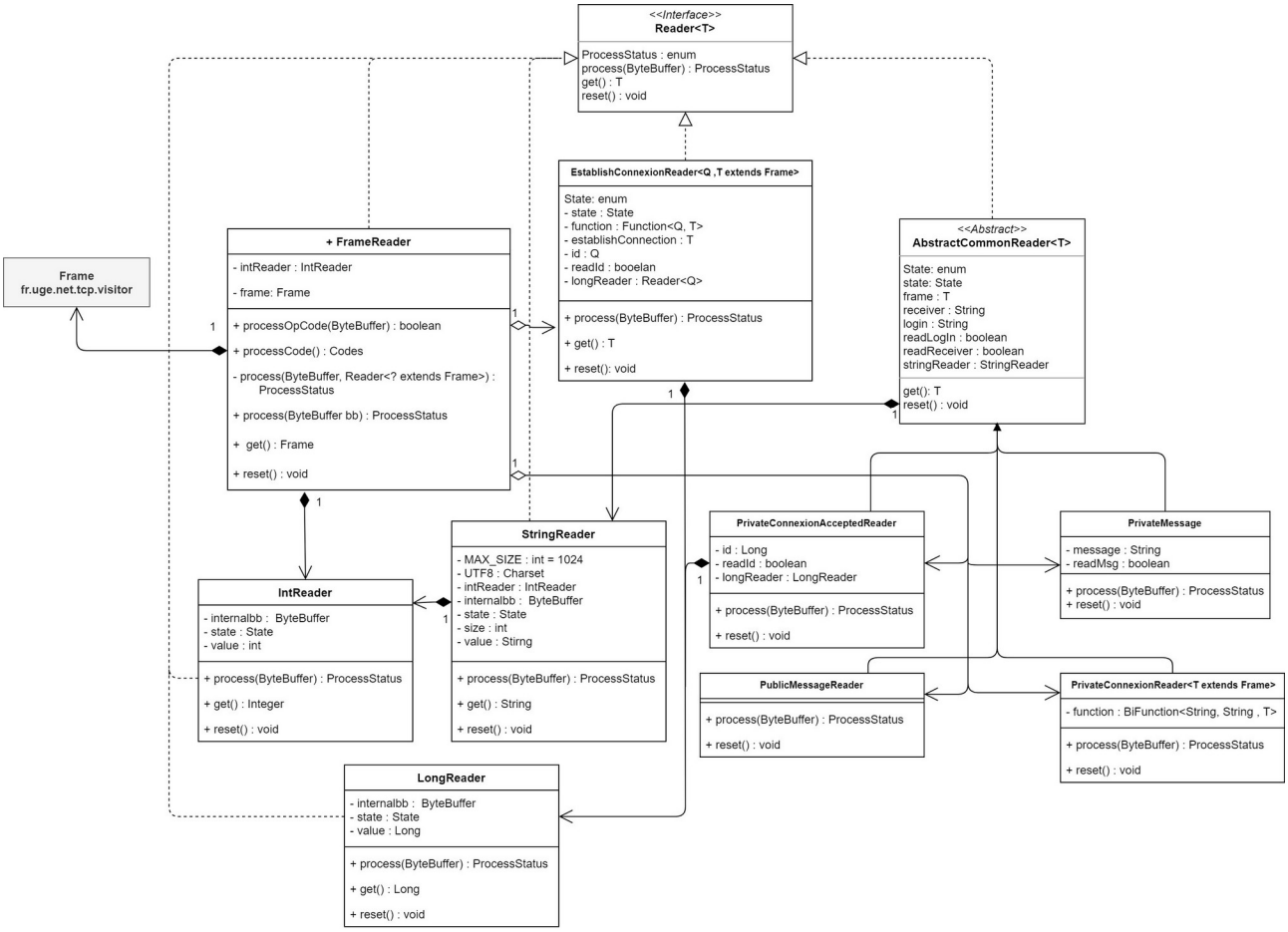

Schéma UML de FrameReader

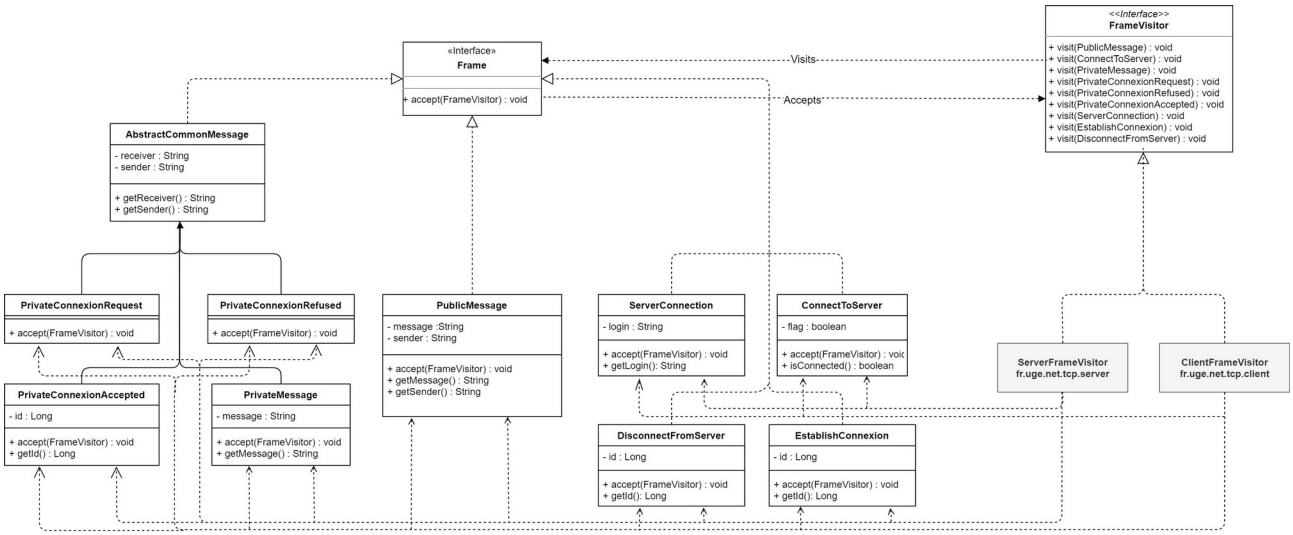

Schéma UML du patron Visiteur ChatOS

II. Soutenance bêta : Points abordés et retour

Divers points ont été abordés durant notre soutenances bêta du vendredi 11 avril tels que :

- La mise en place d’une trame reader

- La mise en place de visiteur(s)

- La maintenance d’une connexion privée d’un client si celui-ci déconnecte sa connexion publique

Suite à celle-ci, nous avons donc pu mettre en place la trame reader et des visiteurs comme demandés et qui a nécessité une revue importante du code concernant l’architecture. Ayant beaucoup travaillé le deuxième point, nous n’avons pas pu implémenter le troisième point concernant la maintenance d’une connexion privée. Une idée à laquelle nous avons pensé était de mettre en place une commande pour le client (exemple : « cp login_client ») qui lui permettrait de couper sa connexion publique tout en maintenant sa connexion privée existante avec un autre client.

III. Conclusion

Pour conclure, le projet ChatOS nous aura permis de mieux comprendre l’implémentation d’un chat en TCP et de travailler les notions vu au cours du semestre concernant l’écriture/lecture d’un bytebuffer, la manipulation d’un socket channel et surtout la manipulation d’un serveur et d’un

client en mode non bloquant. Le projet fut pour notre part intéressant à travailler malgré la difficulté à visualiser la bonne architecture de celui-ci qui pouvait devenir compliqué, notamment pour la connexion privée. En résultat, le projet a donc tout au long de son développement évolué et nous avons pu mettre en place les fonctionnalités demandées de l'application.