Programmation réseau

Rapport projet ChatOs

Table des matières

Choix d’architecture :

Concernant l’architecture du projet, nous avons opté pour un large nombre de classes qui possèdent chacune leurs responsabilités classé par package.

`fr.uge.chatos` → Comprend les classes Client, Server et ClientList, défini comme étant les main classes du projet

`fr.uge.chatos.context` →Comprend tous les context selon l’utilisation

`fr.uge .chatos.core` →Comprend des classes permettant la lecture de certaines données présente dans les paquets, les interfaces définissante une trame de paquet ainsi que les différentes version de ces derniers, ainsi qu’une structure de données personnalisé permettant de stocker les paquets avec une taille limité.

`fr.uge.chatos.framereader` →Comprend les classes permettant la lecture des données reçu pour chacun des différents paquets.

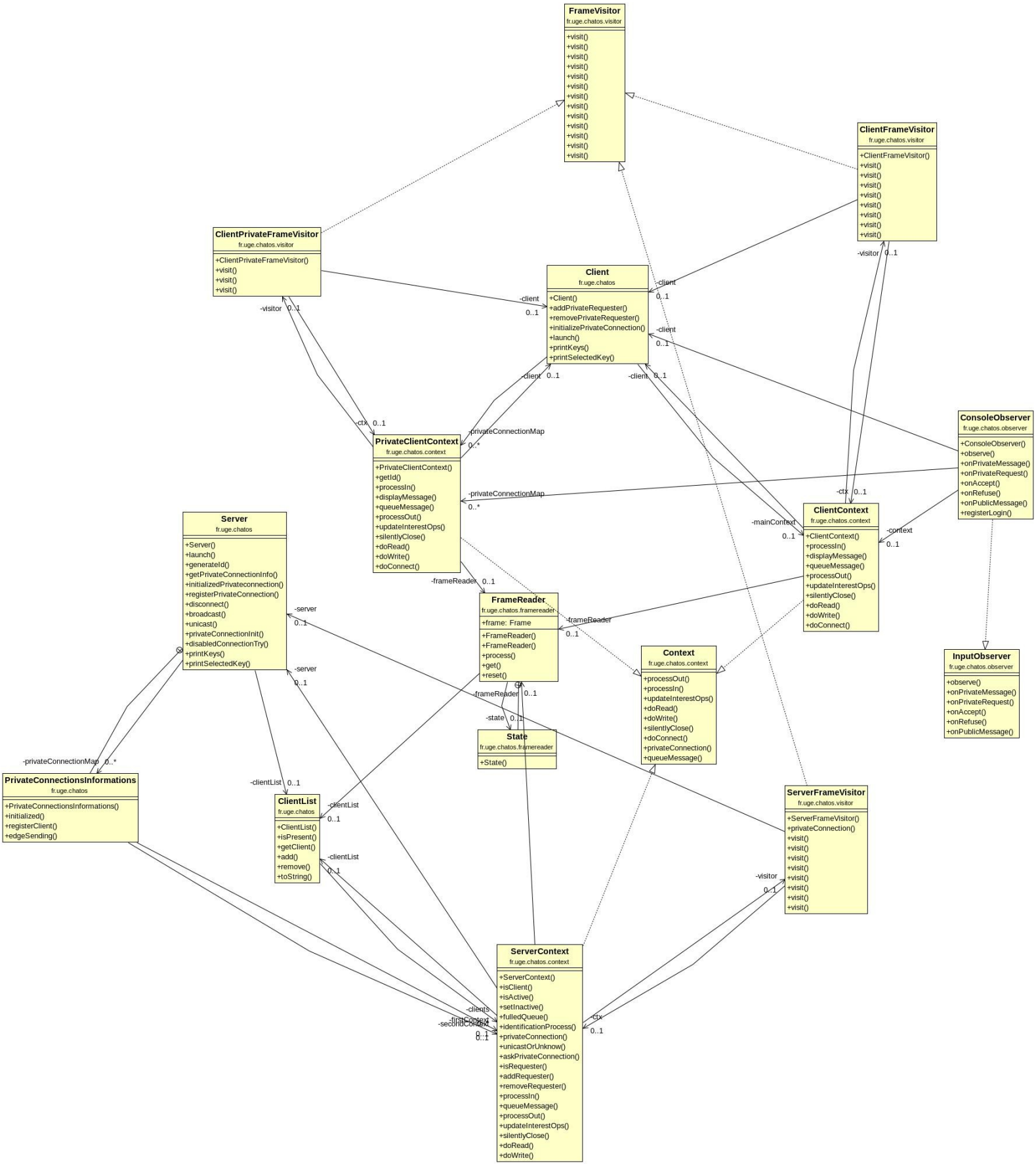
`fr.uge.chatos.frametypes` →Comprend les classes définissant tous les paquets envoyé/reçu.

`fr.uge.chatos.typesreader` →Comprend les classes permettant la lecture des types primitif recu pour chacun des différents paquets.

`fr.chatos.visitor` → Comprend les classes permettant la réalisation du Design Pattern Visitor.

`fr.chatos.observer` → Comprend les classes permettant la réalisation du Design Pattern Observer.

Schéma UML du projet :


FrameVisitor
fr.uge.chatos.visitor
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
ClientFrameVisitor
fr.uge.chatos.visitor
+ClientFrameVisitor()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
-visitor 0..1
ClientPrivateFrameVisitor
fr.uge.chatos.visitor
+ClientPrivateFrameVisitor()
+visit()
+visit()
+visit()
-visitor 0..1
Client
fr.uge.chatos
+Client()
+addPrivateRequester()
+removePrivateRequester()
+initializePrivateConnection()
+launch()
+printKeys()
+printSelectedKey()
-client 0..1
-client 0..1
-client 0..1
-client 0..1
-client 0..1
-ctx 0..1
PrivateClientContext
fr.uge.chatos.context
+PrivateClientContext()
+getId()
+processIn()
+displayMessage()
+queueMessage()
+processOut()
+updateInterestOps()
+silentlyClose()
+doRead()
+doWrite()
+doConnect()
-privateConnectionMap 0..*
-privateConnectionMap 0..*
-frameReader 0..1
ConsoleObserver
fr.uge.chatos.observer
+ConsoleObserver()
+observe()
+onPrivateMessage()
+onPrivateRequest()
+onAccept()
+onRefuse()
+onPublicMessage()
+registerLogin()
-ctx 0..1
-mainContext 0..1
ClientContext
fr.uge.chatos.context
+ClientContext()
+processIn()
+displayMessage()
+queueMessage()
+processOut()
+updateInterestOps()
+silentlyClose()
+doRead()
+doWrite()
+doConnect()
-context 0..1
Server
fr.uge.chatos
+Server()
+launch()
+generateId()
+getPrivateConnectionInfo()
+initializedPrivateconnection()
+registerPrivateConnection()
+disconnect()
+broadcast()
+unicast()
+privateConnectionInit()
+disabledConnectionTry()
+printKeys()
+printSelectedKey()
-server 0..1
-server 0..1
FrameReader
fr.uge.chatos.framereader
+frame: Frame
+FrameReader()
+FrameReader()
+process()
+get()
+reset()
-frameReader 0..1
-frameReader 0..1
-state 0..1
Context
fr.uge.chatos.context
+processOut()
+processIn()
+updateInterestOps()
+doRead()
+doWrite()
+silentlyClose()
+doConnect()
+privateConnection()
+queueMessage()
InputObserver
fr.uge.chatos.observer
+observe()
+onPrivateMessage()
+onPrivateRequest()
+onAccept()
+onRefuse()
+onPublicMessage()
State
fr.uge.chatos.framereader
+State()
-privateConnectionMap 0..*
PrivateConnectionsInformations
fr.uge.chatos
+PrivateConnectionsInformations()
+initialized()
+registerClient()
+edgeSending()
-clientList 0..1
ClientList
fr.uge.chatos
+ClientList()
+isPresent()
+getClient()
+add()
+remove()
+toString()
-clientList 0..1
-clientList 0..1
ServerFrameVisitor
fr.uge.chatos.visitor
+ServerFrameVisitor()
+privateConnection()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
+visit()
-visitor 0..1
ServerContext
fr.uge.chatos.context
+ServerContext()
+isClient()
+isActive()
+setInactive()
+fulledQueue()
+identificationProcess()
+privateConnection()
+unicastOrUnknow()
+askPrivateConnection()
+isRequester()
+addRequester()
+removeRequester()
+processIn()
+queueMessage()
+processOut()
+updateInterestOps()
+silentlyClose()
+doRead()
+doWrite()
-clients
-firstContext
-secondContext
0..1
0..1
-ctx 0..1

Bugs connu :

Les différents bugs rencontré durant la phase de code on tous été corrigé (cf Difficultés rencontrées), suite a notre phase de test nous n'avons recensé aucun bug dans la version livrée.

Difficultés rencontrées :

La première difficulté rencontré lors de notre projet a été lors de la réalisation de la connexion privée notamment pour le changement de context.
En effet une fois la connexion privée établie pour un client ce dernier doit changer de context pour un ClientPrivateContext disposant de méthodes propre a la connexion privée.
Malgré plusieurs tentatives nous nous sommes heurté a un refus de mise a jour de certains sélecteur résultant a une mauvaise réception de certain paquets par le client une fois la connexion privée établie.
Pour outrepassé ce problème nous avons opté pour un boolean qui est mis a jour lors de l'établissement d'une connexion privée.

La réalisation d'un pont pour une connexion privé nous a demandé de revoir une grande partie de l'implémentation de notre code pour une fois la connexion établie pouvoir différencier la phase publique de la phase privée.
Nous avons pour cela du rajouter différents context pour gerer au mieux les responsabilité et avoir une meilleur vision sur la gestion de cet aspect.

La dernière difficulté rencontrée a été l'implémentation des Design Pattern Visitor et Observer, cela nous a permis de revoir nos cours du semestre dernier pour implémenter ces deux fonctionnalités, mais malgré cela ça n'a pas toujours été évident, ils nous a fallu nous y prendre a plusieurs tentatives.

Évolutions demandées :

1. Ajouter une taille max pour la queue dans le cas d'un utilisateur inactif

   Pour cela nous n'avons pas pu utiliser une LinkedList traditionnel en Java car cette structure ne permet pas de limiter le nombre d'objet au sein de la liste.

   Nous avons donc opté pour une structure personnel qui implémente LinkedList et nous permet de contrôler le nombre maximal de paquets pouvant être recu.

2. Mettre un enum pour process les paquets plutot que de passer par l'opcode

   Notre méthode parsePacket dans la classe FrameReader permet d'appeler les méthodes correspondantes a la lecture des différents paquets en fonction de l'opcode recu, pour cela

nous passons l’opcode en paramètre de la méthode puis nous appliquons un switch correspondant au noms des différents paquets faisant référence a un opcode dans la classe FrameReader.

3. Rajouter un timeout

   Nous avons deux système de timeout, par le temps, qui déconnecte un utilisateur inactif au bout de 5 minutes et également un timeout dans le cas ou un utilisateur reçois un trop grand nombre de paquet sans réponse, ce dernier point est géré grâce a la structure LimitedQueue

4. Une vue pour les clients a la récéption d'un packet

   Ici le principe est qu’après un broadcast chaque context de chaque client doit envoyer les memes données aux autre clients.

   Ces données sous forme de ByteBuffer sont construite a partir d’objet Paquet mais l’action est exécuté autant de fois qu’il y a de clients ce qui consomme inutilement du CPU et de la mémoire.

   Une solution aurait été de produire le ByteBuffer avant de faire le broadcast et donc d’envoyer a tout le monde le même ByteBuffer.

   Une autre solution aurait été d’avoir pour chaque context une vue c’est a dire une Read Only ByteBuffer qui fait référence au vrai ByteBuffer construit.

   Malheureusement par contrainte de temps, aucune des solutions n’a pu etre implémenté malgré le bénéfice évident que cela représente.

5. Ajout Design Pattern visiteur

   L’ajout du Design Pattern Visitor pour la la réception des paquets pour les différents context nous a permis une factorisation de notre code ainsi qu’une meilleure lisibilité non négligeable.

   Partant du même principe nous avons pris la liberté d’implémenter un Design Pattern Observer pour nous aider a mieux parser les commandes entré par l’utilisateur, et pour permettre une meilleur flexibilité par rapport aux éventuels changement et évolutions du projet.