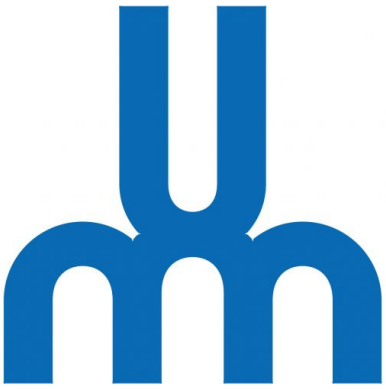
Devoir 2

Étienne Collin
20237904
Emeric Laberge
20220275

Dans le cadre du cours
IFT 3325



Département d'informatique et de recherche opérationnelle
Université de Montréal
Canada
29 novembre 2024

1 Diagramme de classes

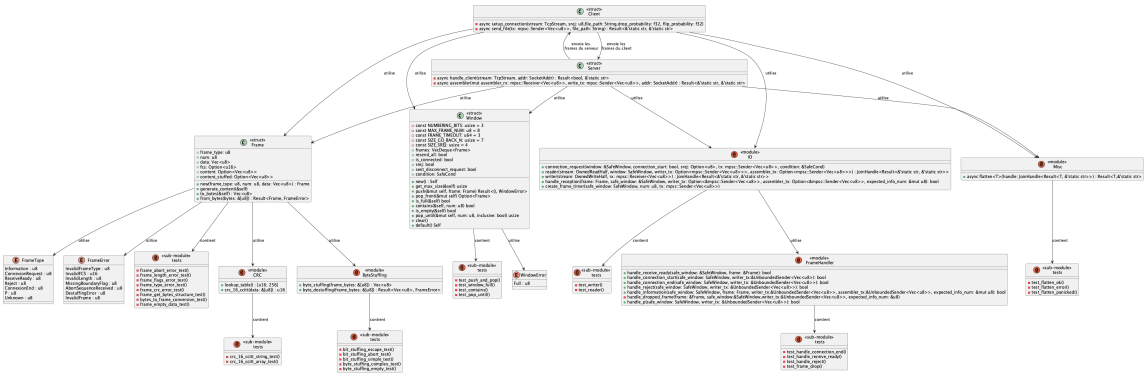


FIGURE 1 – Diagramme de classes

Veuillez consulter le fichier `class-diagram.svg` pour une version plus lisible du diagramme de classes.

Note : La documentation est présente à l'URL : https://ift3325.etiennecollin.com/

Notre projet se compose de deux modules principaux : le client et le serveur. Commençons par le module client. Ce dernier établit une connexion avec le serveur et initialise deux tâches asynchrones parallèles : le "reader" et le "writer". Chacune de ces tâches prend en charge une "moitié" de la connexion au serveur : le "reader" gère la partie lecture du socket, tandis que le "writer" s'occupe de l'écriture.

Le client envoie une demande de connexion au serveur et attend une réponse. Dès que la réponse est reçue, une troisième tâche asynchrone est lancée. Cette tâche lit le fichier demandé, le divise en morceaux, puis génère des trames à partir de ces morceaux. Les trames sont envoyées dans une fenêtre, tout en veillant à ne pas dépasser sa taille, et sont également transmises au "writer" qui les fait passer par le socket.

Étant donné que plusieurs tâches fonctionnent de manière asynchrone, des mécanismes de synchronisation et de communication sont mis en place pour garantir une bonne interaction entre les différentes parties du système. Pour plus de détails sur le fonctionnement du client, veuillez consulter ce lien.

Le serveur, quant à lui, attend une connexion sur le socket TCP. Une fois la connexion établie, il lance une tâche asynchrone dédiée au traitement du client, permettant ainsi la gestion simultanée de plusieurs clients. Pour chaque client, le serveur démarre deux tâches : "reader" et "writer", comme c'est le cas du côté client.

Cependant, le serveur active également une tâche supplémentaire, appelée "assembler", qui reçoit les trames lues et les réassemble. Ces trames sont lues par le "reader", qui s'assure de vérifier leur intégrité en détectant les trames corrompues ou perdues. Une fois la validité des trames confirmée, celles-ci sont envoyées au "frame handler". Ce dernier prend en charge les opérations spécifiques liées au type de trame reçue.

Lorsqu'une demande de déconnexion du client est reçue, le serveur sait que toutes les données ont été correctement transmises. Il assemble alors le fichier et le sauvegarde avant de fermer le

socket. Le serveur est ensuite prêt à accepter de nouvelles connexions. Pour plus de détails sur le fonctionnement du serveur, veuillez consulter ce lien.

Lors de l'utilisation du client et du serveur, une probabilité de perte de trame et de "bit flip" peuvent être spécifiées. Ces opérations servent à simuler une connexion imparfaite et sont appliquées au niveau de la tâche "writer".

Le module `Utils` regroupe plusieurs fonctions partagées entre le client et le serveur. Il comprend sept sous-modules, chacun dédié à des fonctions spécifiques. Pour des informations plus détaillées sur les membres du module `Utils`, consultez ce lien.

- **Byte Stuffing** : Ce sous-module fournit des fonctions pour le byte stuffing et le byte destuffing. Ces fonctions sont utilisées lors de l'envoi et de la réception de trames par les deux services pour garantir l'intégrité des données transmises.
- **CRC** : Implémentation de l'algorithme CRC-16 CCITT, utilisé pour calculer le CRC d'une trame avant son envoi. Le CRC calculé est ensuite ajouté à la fin de la trame afin de vérifier l'intégrité des données.
- **Frame** : Ce sous-module définit la structure des trames ainsi que des types associés, tels que l'énumération `FrameType` et `FrameError`. Il permet de gérer et de manipuler les trames de manière cohérente et typée.
- **FrameHandlers** : Ce sous-module est responsable du traitement des différents types de trames du côté du récepteur. Chaque type de trame (par exemple, RR, Reject, Connexion, Déconnexion, etc.) est géré par une fonction dédiée, assurant une gestion adéquate de tous les scénarios possibles.
- **IO** : Ce sous-module regroupe les fonctions relatives aux opérations d'entrée-sortie (IO), incluant la demande de connexion, la création de temporisations, la gestion de la réception des trames, ainsi que les fonctions reader et writer utilisées pour communiquer via le socket.
- **Misc** : Ce sous-module contient diverses fonctions utilitaires, dont la fonction `flatten` qui facilite la manipulation des objets `JoinHandle` asynchrones, permettant ainsi une gestion plus fluide des tâches parallèles.
- **Window** : Ce sous-module est entièrement dédié à la gestion des fenêtres d'envoi et de réception. Il contient tout le code nécessaire pour implémenter la logique des fenêtres, assurant le contrôle de flux et la gestion de la transmission des trames dans un ordre correct.