RAPPORT PROJET REPARTITION : TELECHARGEMENT P2P

Louis ROYER, Jean-Sébastien AUGE, HAAS Flavien

STRI 2A, UPSSITECH, 2020

Table des matières

Lien du git :
https://git.flavien.ovh/flavien/Projet_JAVA_P2P_STRI2A

Lien de la vidéo de démonstration :
https://youtu.be/WFhnDIa11JY

Introduction

Le but de ce projet est de créer une application répartie en Java de téléchargement de fichier en mode P2P.
Nous avons réalisé toutes les étapes qui étaient proposées ainsi que 3 options sur les 4 qui étaient suggérées. Le lancement des applications est décrit dans le fichier README.md.

Conception

Le tracker gère l'accès au stockage des informations des fichiers et applications trackés en implémentant le modèle lecteurs/rédacteurs. Un processus vérifie périodiquement s'il y a un changement dans la liste des fichiers que l'application possède en local afin de les enregistrer auprès du tracker.

Pour la gestion du P2P coopératif, lorsqu'une application a reçu tous les blocs d'un fichier, elle met à jour auprès du tracker le nombre de blocs reçus par chaque autre application. Un processus met en cache le ratio des applications demandant le téléchargement de parties de fichier afin de limiter le nombre de requêtes vers le tracker. Lorsque le ratio est trop déséquilibré, les applications peuvent refuser aléatoirement de servir certains blocs (la probabilité de refus augmente avec le déséquilibre du ratio reçus/envoyés) afin de désavantager les applications qui ne font que demander des fichiers.

Afin de pouvoir gérer à la fois TCP et UDP, nous avons fait le choix d'utiliser les sockets plutôt que RMI. Nous avons donc créé un protocole que nous avons amélioré au fur et à mesure que le projet avançait. La documentation de notre protocole est disponible à l'adresse protocol.md.
Mais est également recopiée ci-dessous. Nous avons choisi de documenter tout notre projet en anglais, cette section du rapport ne sera donc pas traduite en français.

Protocole : P2P-JAVA-PROJECT version 1.2 (Binary protocol for step 2+)

All strings in the datagram are utf-8 encoded. All numerical values are big endian.

```
1 byte:  [(byte  0  ): VERSION(0x11, first quartet is major version, secon
d is minor)]
1 byte:  [(byte  1  ): REQUEST/RESPONSE CODE]
2 bytes: [(bytes 2-3): CHECKSUM (UDP only)]
4 bytes: [(bytes 4-7): PAYLOAD SIZE IN BYTES]
x bytes: [(bytes 8-?): PAYLOAD]
```

Requests and responses codes

- REQUESTS (msb is 0):
  - LIST (0x00)
  - LOAD (0x01)
  - HASH (0x02)
  - DISCOVER (0x03)
  - REGISTER (0x04)
  - UNREGISTER (0x05)
  - RATIO (0x06)
  - UPDATE RATIO (0x07)
  - SIZE (0x08)
- RESPONSES (msb is 1):
  - LIST (0x80)
  - LOAD (0x81)
  - HASH (0x82)
  - DISCOVER (0x83)
  - RATIO (0x86)
  - DENIED (0x87)
  - SIZE (0x88)
  - VERSION ERROR (0xC0)
  - PROTOCOL ERROR (0xC1)
  - INTERNAL ERROR (0xC2)
  - EMPTY DIRECTORY (0xC3)
  - NOT FOUND (0xC4)
  - EMPTY FILE (0xC5)
  - NOT A TRACKER (0xC6)
  - UNKNOWN HOST (0xC7)

List

Payload size for list request is always zero. Payload for list response is filenames separated by \n. Payload size for list response is never zero.

Empty directory

When directory is empty. Payload size for empty directory is always zero.

Load

Not found

Response when the file requested is not found on the server. Payload size for Not found is zero.

Load response

Payload contains

```
8 bytes: [(bytes  8-15): OFFSET OF FILE CONTENT IN BYTES]
4 bytes: [(bytes 24-27): FILENAME SIZE] (cannot be > to PAYLOAD_SIZE - 20
or be zero)
y bytes: [<FILENAME>]
z bytes: [PARTIAL CONTENT]
```

Load request

Payload contains

```
8 bytes: [(bytes  8-15): OFFSET OF FILE CONTENT IN BYTES]
8 bytes: [(bytes 16-23): MAX SIZE OF PARTIAL CONTENT (partial content in r
esponse should not excess this size, but this can be less (by example if e
ndoffile is reached or server doesn't have the full block requested)
4 bytes: [(bytes 24-27): FILENAME SIZE] (cannot be > to PAYLOAD_SIZE - 20
or be zero)
y bytes: [<FILENAME>]
2 bytes: port used to register on tracker
? bytes: hostname used to register on tracker
```

Possible responses: Load response, or Denied

Size

Size request

Payload contains

```
? bytes: filename
```

Possible responses: Size response, Empty file or Not found

Size response

Payload contains

```
8 bytes: file size
? bytes: filename
```

Hash

Hash request

Get hash of a file. Payload contains

```
4 bytes: [(bytes  8-11): FILENAME SIZE]
y bytes: [<FILENAME>]
z bytes: [ALGO_NAMES requested separated by \n] (ex.: SHA-256, MD5)
```

If file does not exists, a Not found can be responded.

Hash response

Payload contains:

```
4 bytes: [(bytes  8-11): FILENAME SIZE]
y bytes: [<FILENAME>]
[[ multiple algo hashes bloc]]
```

A algo hash bloc contains:

```
4 bytes [ALGO_NAME size]
? [ALGO_NAME]
4 bytes: [HASH SIZE (bytes)] / or 0 if this hash algorithm is unsupported.
?? [HASH]
```

Tracker specific messages

Register

Used by a server to register itself on a tracker. Server may want to do a free `DISCOVER` to check if they have been registered. Payload contains:

```
2 bytes: [<PORT NUMBER>]
```

Unregister

Used by a server to unregister itself from a tracker. No error is raised if the server was not registered. Server may want to do a free `DISCOVER` to check if they have been unregistered. Payload contains:

```
2 bytes: [<PORT NUMBER>]
? bytes: [<HOSTNAME>]
```

Discover request

If payload size is null, lists all servers registered. If payload contains a filename, list all servers having this file in their list.

```
? bytes: [<FILENAME>]
? bytes: [<HOSTNAME>]
```

Discover response

Contains:

```
4 bytes: [(bytes  8-11): FILENAME SIZE]
y bytes: [<FILENAME>]
? bytes [multiple server blocks]
```

Server block is composed with:

```
2 bytes: [port]
? bytes: hostname
\n
```

Not a Tracker

This error is raised when receiving a Discover, a Register, or an Unregister request, but this application is not a tracker.

Ratio Request

Contains:

```
2 bytes: port
? bytes: hostname
```

Possible responses: Ratio response, or Unknown host

Ratio Response

Contains:

```
8 bytes: total sent bytes
8 bytes: total received bytes
2 bytes: port
? bytes: hostname
```

Update Ratio

Contains:

```
8 bytes: bytes sent by the server
2 bytes: server port
2 bytes: client* port
? bytes: server hostname followed by \n
? bytes: client* hostname
* note: used by client to register on tracker
```

Possible responses: No response, or Unknown host (if client is not registered or server is not registered)
Note: client must have verified hash before sending Update Ratio to tracker for each server which participated.

Unknown Host

Payload size is zero.

Denied

Contains

```
8 bytes: offset of content asked
? bytes: filename
```

Other response code (errors)

Version error

Response when datagram received use wrong version code.

Protocol error

Response when the request cannot be interpreted (but version is correct). Payload size for Protocol error is zero

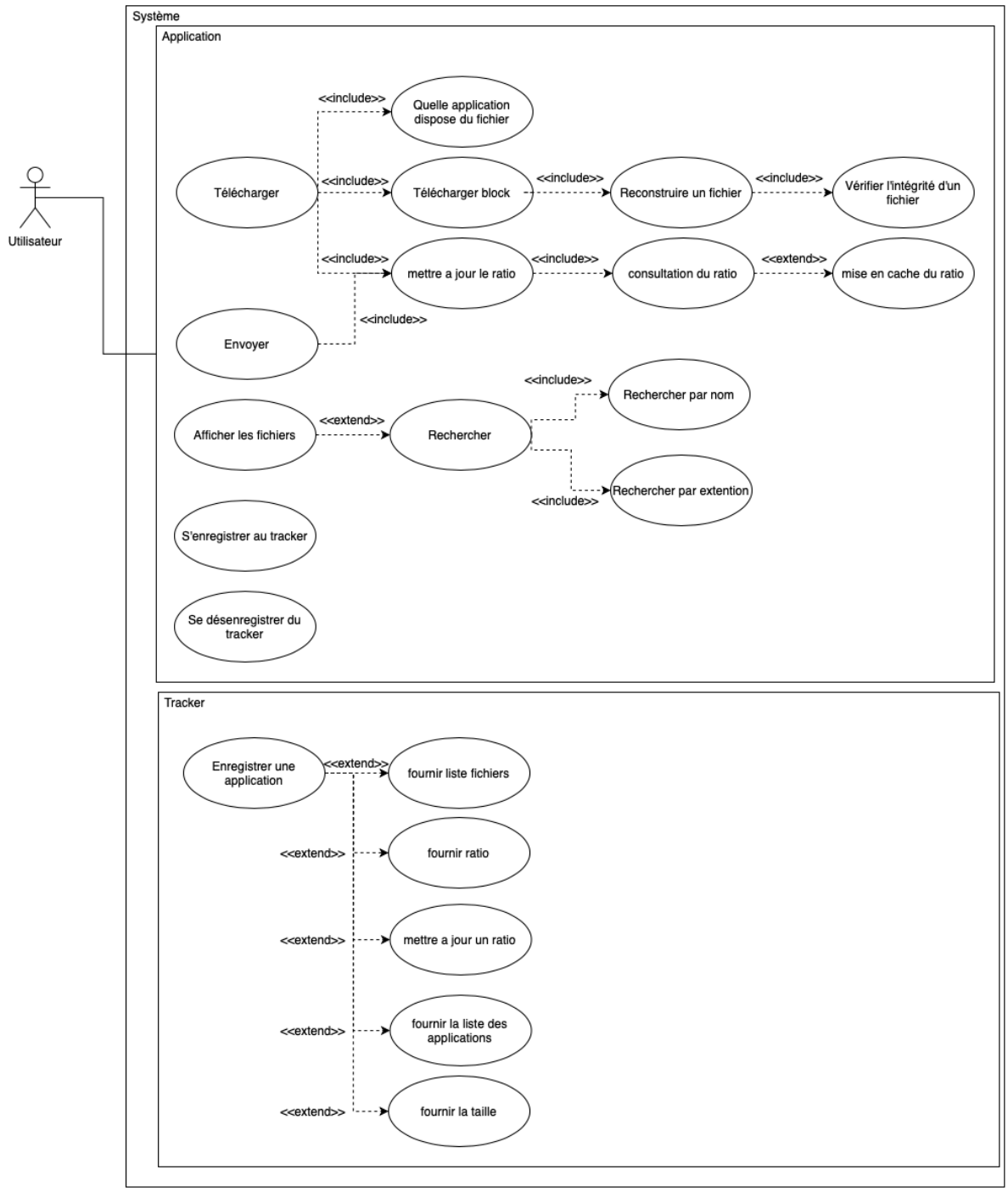
Internal error

Response in internal failure case. Payload size for Internal error is zero.

UML

Voici les diagrammes UML que nous avons utilisé pour modéliser notre projet.

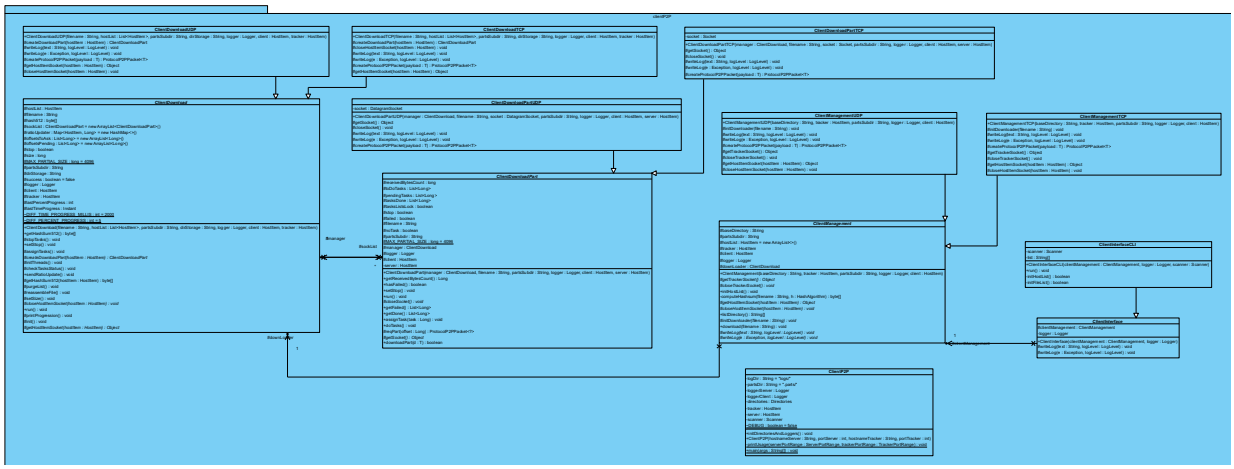
Diagramme de cas d'utilisations



Diagrammes de classes

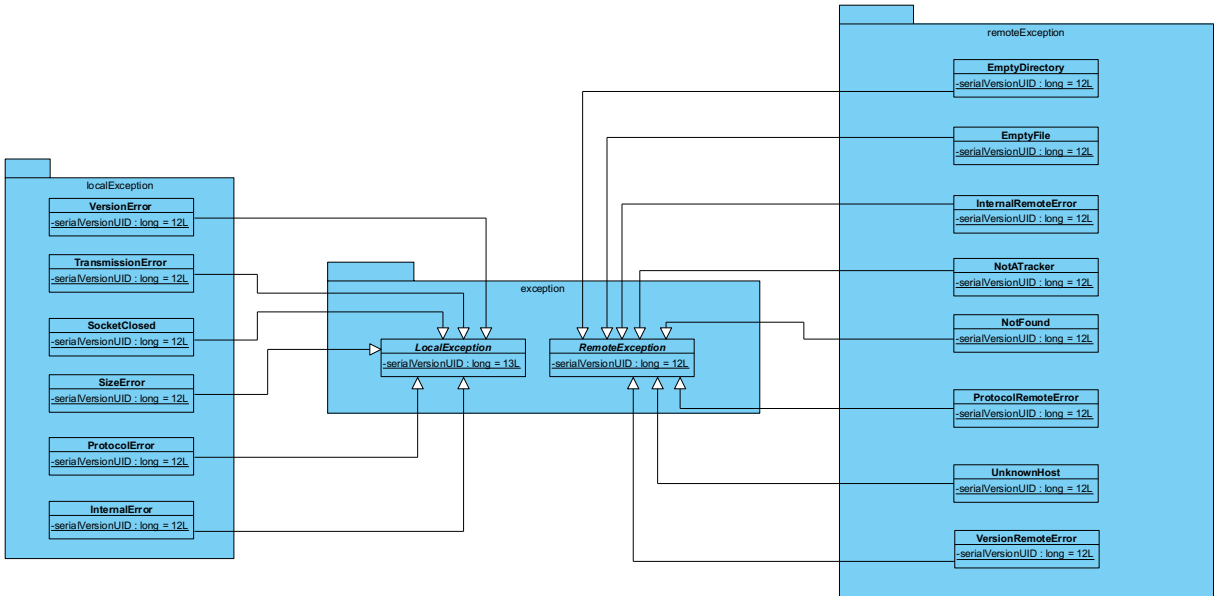
L'intégralité du diagramme de classe est disponible ici :
https://git.flavien.ovh/flavien/Projet_JAVA_P2P_STRI2A/src/branch/master/doc/Class%20Diagram.svg

Package clientP2P



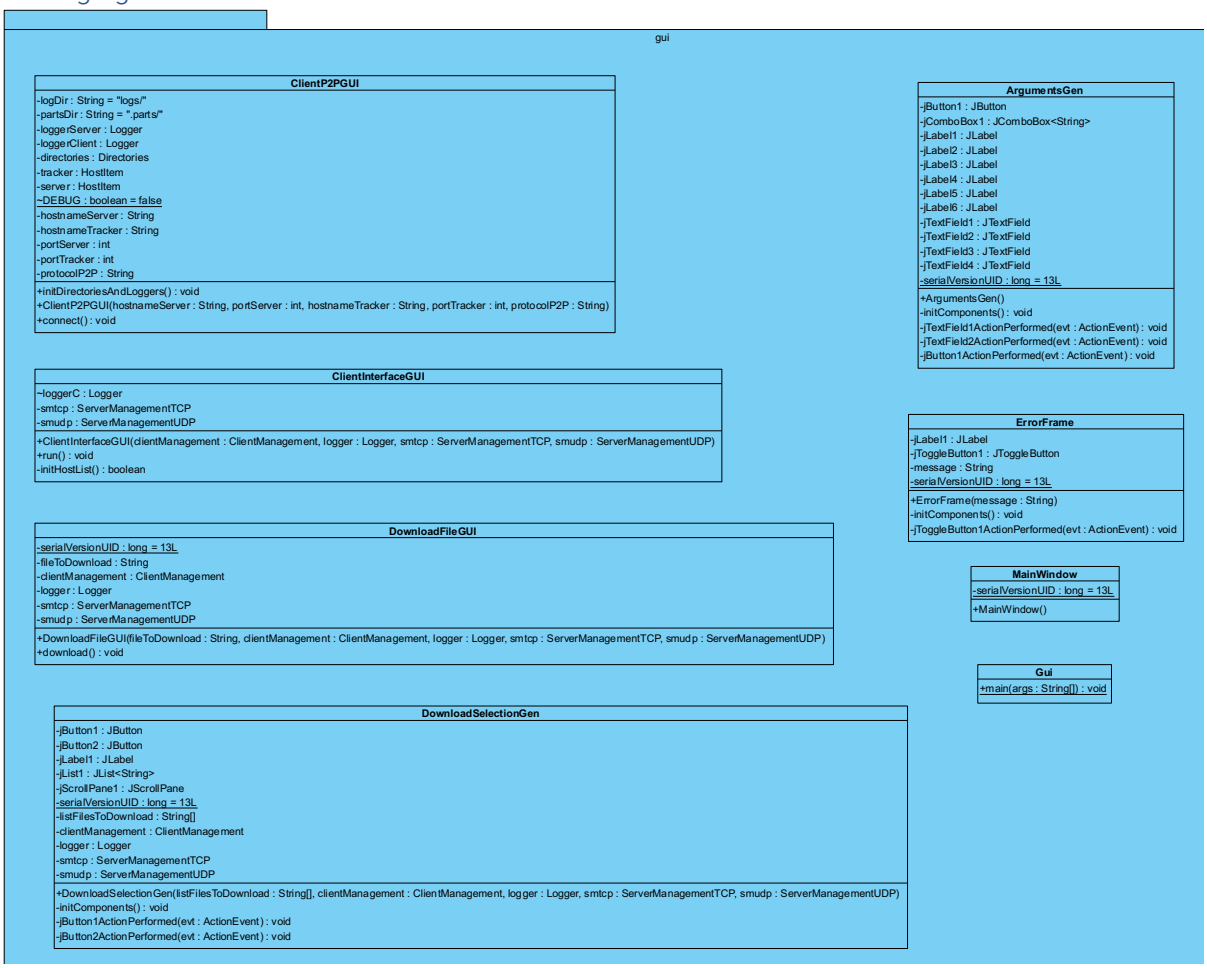
Ce package contient les classes relatives au fonctionnement du client et de ses fonctionnalitées. C'est ici que se trouve clientP2P.java, que l'on lance pour lancer un client en ligne de commande.

Package Exceptions/localException/remoteException



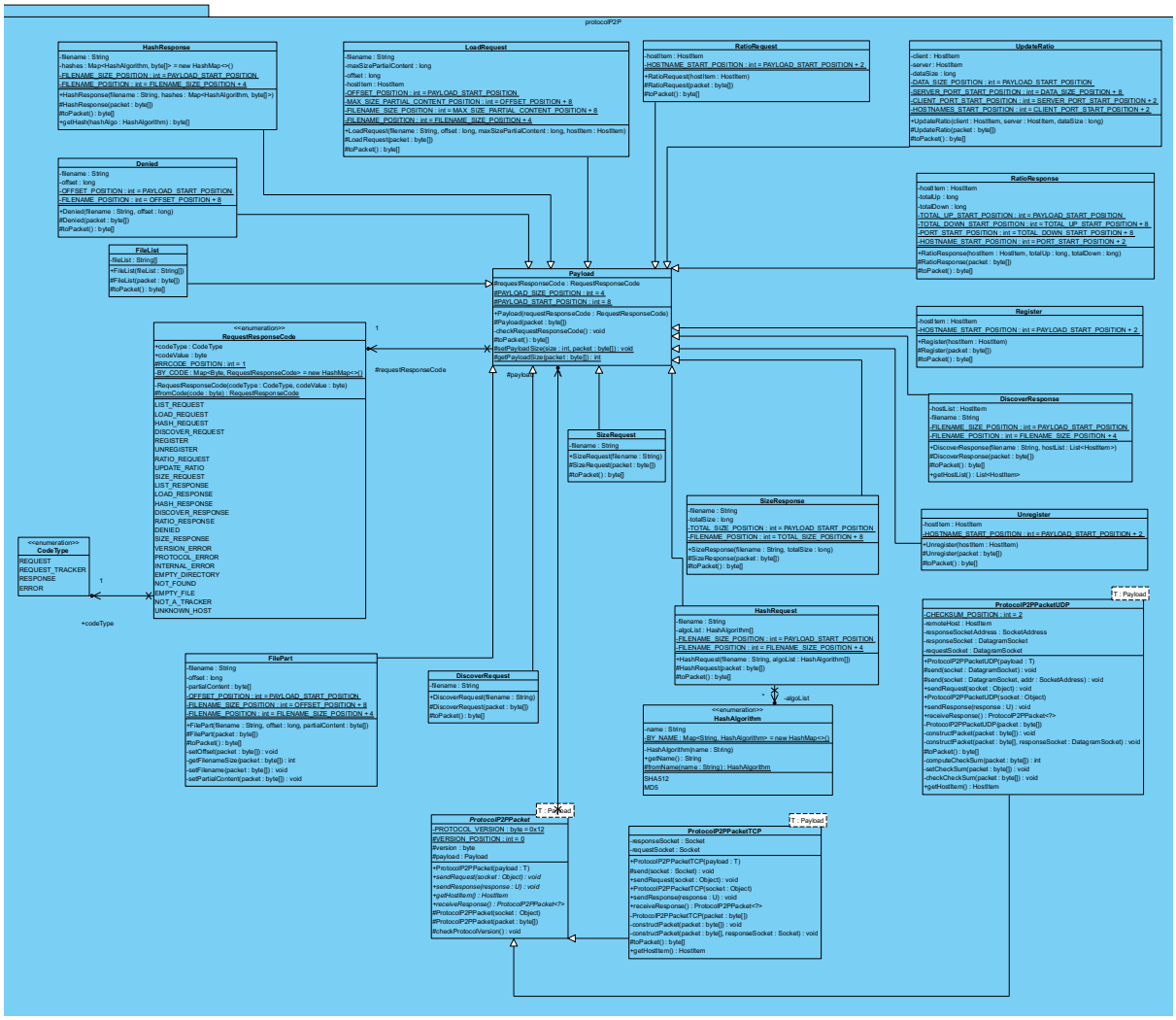
Contient les deux types d'exceptions mères dont les exceptions customs héritent (LocalException et RemoteException).

Package gui



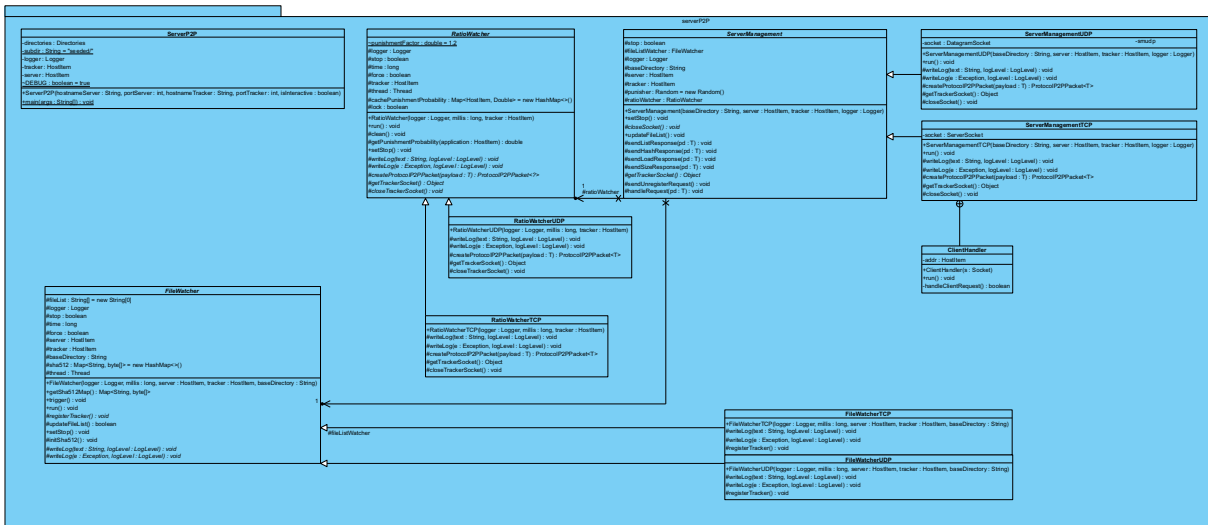
Contient les classes relatives à l'interface graphique et à son fonctionnement (avec classes ClientP2PGUI et classes de téléchargement adaptées par rapport à celles présentent dans clientP2P).

Package ProtocolP2P



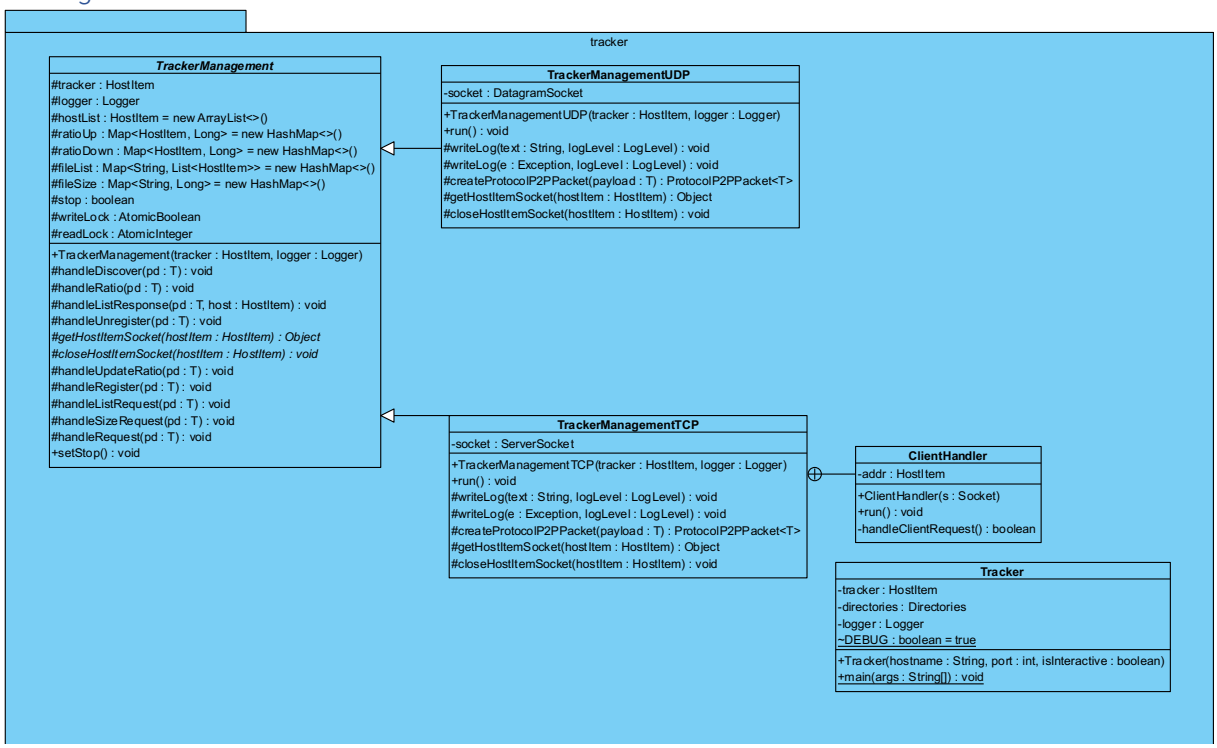
Contient les classes et les méthodes relatives au fonctionnement du protocole créé.

Package serverP2P



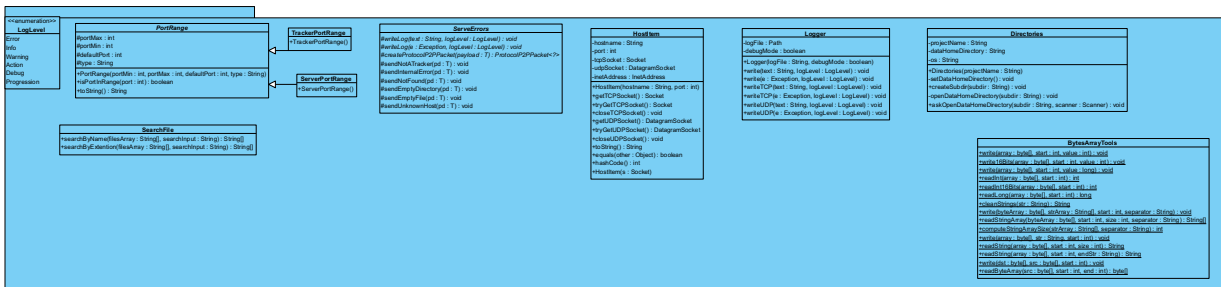
Contient les classes relatives au fonctionnement d'un serveur. On trouve dans ce package la classe ServerP2P qui permet de lancer un serveur autonome en ligne de commandes afin d'effectuer des tests, en plus des méthodes servant à lancer serveur de chaque client.

Package tracker



Contient les classes et les méthodes relatives au fonctionnement du tracker. C'est ici que l'on trouve Tracker.java, que l'on utilise pour lancer le tracker en lignes de commandes

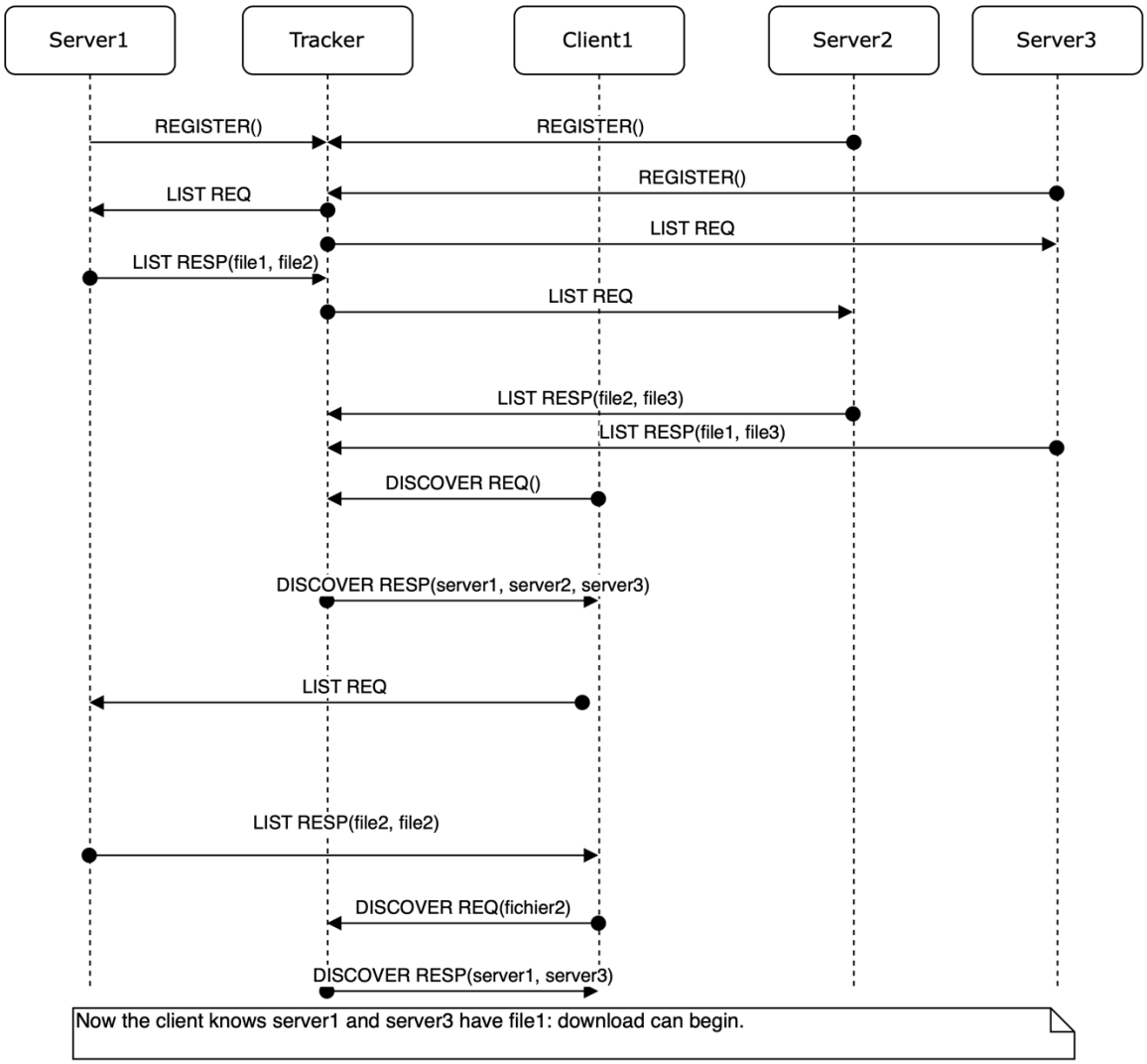
Package tools



Contient plusieurs classes comprenant des méthodes outils, utilisées dans les autres classes (par exemple pour tester la validité d'un port, ou effectuer une recherche).

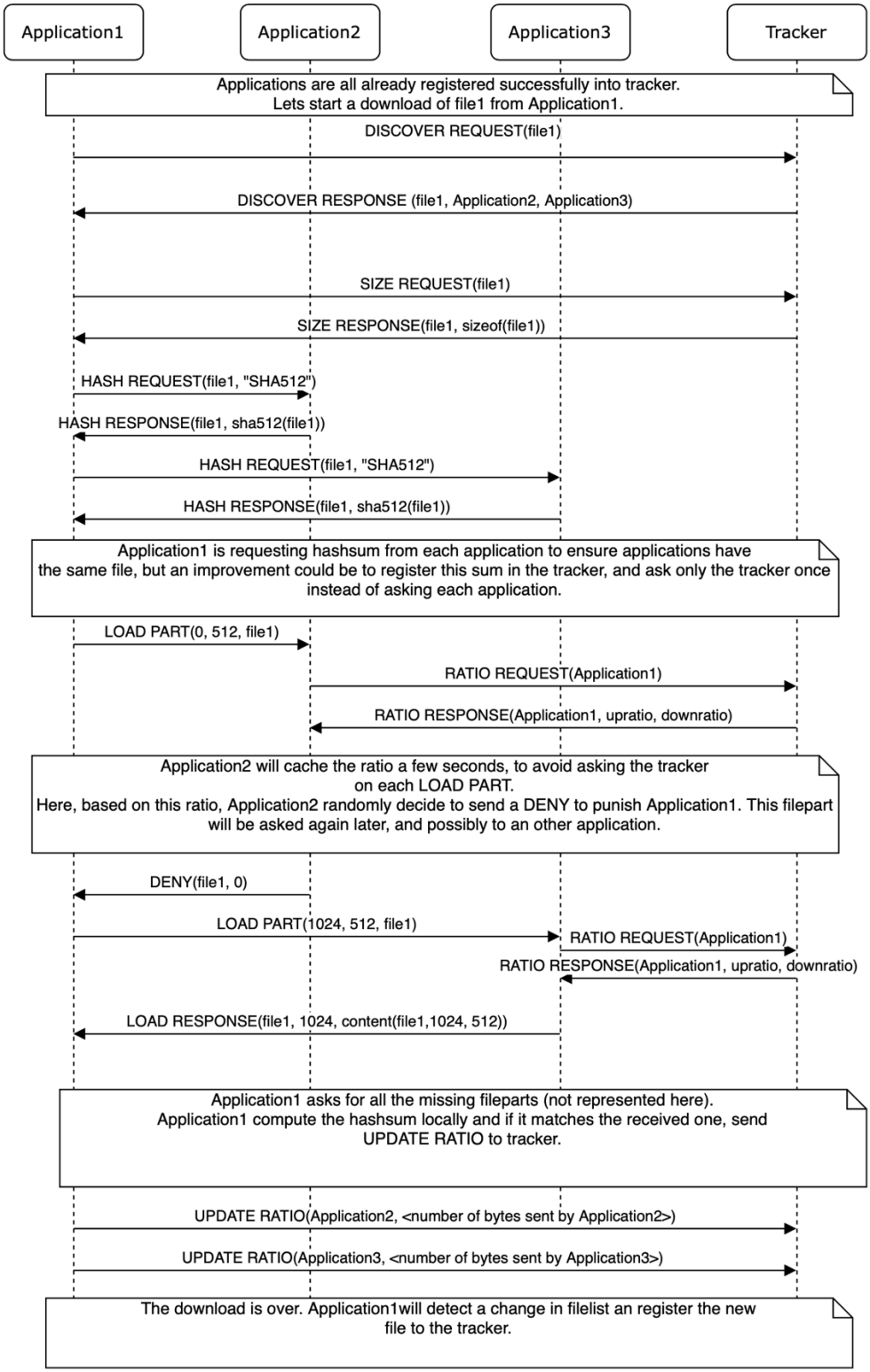
Diagrammes de séquence

Séquence tracker



Ce diagramme représente l'enchainement des actions permettant la connexion a un tracker.

Séquence download



Ce diagramme représente l'enchainement des actions durant le téléchargement d'un fichier.

Conclusion

Notre projet répond aux critères qui étaient énoncés dans le sujet, et nous avons même 3 fonctionnalités optionnelles :

- Créer une IHM graphique pour les applications avec Swing.
- Gérer à la fois des communications UDP et TCP.
- Permettre la recherche de fichiers à partir de leur nom ou de toute autre caractéristique. À l'issu de la recherche on devra pouvoir connaître un ensemble d'application possédant le fichier et commencer le téléchargement.

Cependant, notre programme pourrait encore être amélioré, notamment par :

- une amélioration de l'interface graphique (actuellement elle ne propose pas encore autant de choses que l'interface en lignes de commandes)
- une amélioration de l'interface en lignes de commandes (on aimerait ajouter plus de détails sur les fichiers dans la liste affichée)
- des améliorations protocolaires (on voudrait que le tracker réunisse plus d'informations sur les fichiers traqués pour diminuer le nombre de requêtes nécessaires)

Nous avons également d'autres pistes d'améliorations qui sont disponibles à l'adresse https://git.flavien.ovh/flavien/Projet_JAVA_P2P_STRI2A/issues