**Morbois Antoine - Curtet Quentin - Pouget François - Le Bissonnais Gautier**

C++ Rapport TP – EPSI Hadopi

## Spécifications générales

Le but de ce projet est de pouvoir synchroniser des fichiers au travers d’un réseau local. On peut imaginer qu’il sera utilisé dans un réseau d’entreprise pour faciliter les échanges entre membres d’une même équipe d’un projet. Les échanges se feront donc uniquement via des câbles RJ45, tous les échanges sans fil ne seront pas pris en compte.

Le client doit donc dès le lancement du logiciel pouvoir synchroniser un répertoire de son ordinateur avec des ordinateurs distants. Pour fonctionner le logiciel doit être démarré sur chacun des postes que l’on souhaite synchroniser. Il n’y a pas de serveur qui centralise les échanges, par conséquent chaque poste devra donc jouer le rôle de serveur et de client.

Le client devra pouvoir récupérer les fichiers des voisins tout en transférant ces propres fichiers à ces voisins. Il pourra également voir la liste des fichiers transférés et récupérés. Le répertoire peut être choisi au lancement de l’application. Les transferts de fichiers se font de manière simultanée et indépendante.

Ce logiciel sera développé en C++ et devra utiliser un système de threading pour répartir les échanges et optimiser au maximum la vitesse de ces derniers. Les aspects réseaux, sécurité (hachage) seront gérés dans les librairies fournies avec le cahier des charges.

Le produit fini sera déployé uniquement sur des Windows x64.

En ce qui concerne le développement, notre équipe utilisera Visual Studio (2012 et 2013).

Seront utilisés également Google Code comme gestionnaire de version ainsi que Trello pour assigner les différentes tâches du TP.

## Spécifications détaillées

Pour des raisons de simplicité nous utiliserons des Design pattern :

1 singleton de config qui contient :

* Le nombre d’envois simultanés au client (nombre de thread)
* Le nombre de réceptions simultanées
* Le timeout du scan
* L’adresse du répertoire

1 singleton de config-interne :

* 1 tableau avec la liste des clients
* 1 tableau notant les fichiers

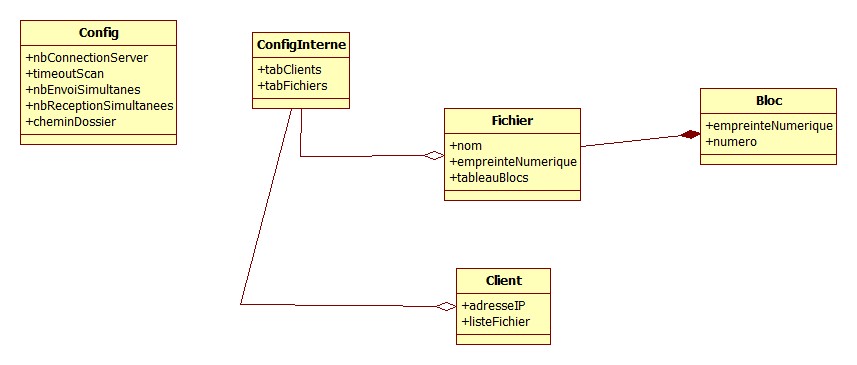
Découpage en différentes listes des tâches :

* Transfert de fichier avec l’utilisation des threads (~10h).
* **Tableau de fichiers** (Synchroniser liste des fichiers présent dans le dossier et dans le tableau contenant le nom et l’empreinte numérique (refresh toutes les 5 min idéal)) (2h)
* Vérification tableau nom + empreinte sur tous les serveurs (2h)
* **Tableau de blocs** (2h) (décomposition de chaque fichier en plusieurs blocs (16k max)
* Mise en attente en cas de surnombre de connexion (optimisation si temps)
* Mise en place de **log élaboré** dans un fichier .txt (thread)
* Mise en place « **log utilisateur** » détaillant quel utilisateur a récupéré quel fichier
* Stop synchronisation si **touche échap** (suppression bloc si fichier non entier) (~1h)
* Paramètre rentré manuellement dans un 1er temps puis demande paramètre avant lancement (30min) => paramètre console
* Même nom + empreinte différente? => la machine venant de se connecter renomme son fichier avant d’importer l’autre. (~2h)

Liste des exceptions possibles :

* N’arrive pas à créer le serveur.
* Deux fichiers portent le même nom
* La synchronisation ne se fait pas

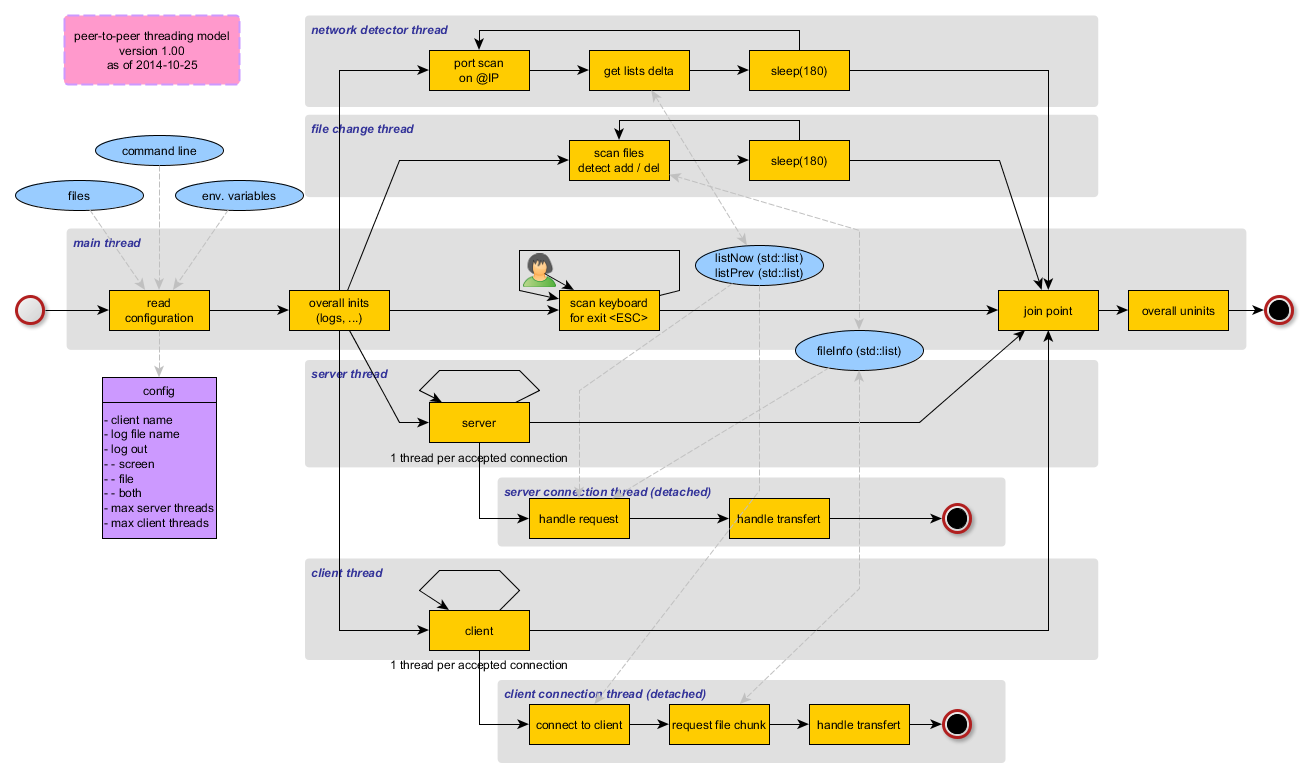
Diagramme de classe :



## Bilan de réalisation

Pour la réalisation de ce TP, nous nous sommes servis du schéma ci-dessous afin de comprendre comment fonctionnait le TP et afin de savoir comment développer ce dernier en plusieurs modules.

On en a défini les différents singletons à réaliser et les différents modules comme l’écriture des fichiers de log, les communications entre les différents postes, …



Notre ressenti sur ce TP, est qu’il était assez complexe à mettre en œuvre mais le principe est assez simple (un logiciel de synchronisation de fichier avec gestion des conflits lorsque plusieurs fichiers portent le même nom). Nous avons eu beaucoup de mal avec le découpage des différents fichiers en bloc pour les transférer un par un. Mais une fois que cela était fait, il fallait encore faire les échanges c’est-à-dire copier les blocs un à un d’un ordinateur à un autre.

Lors de la réalisation de ce projet, nous nous sommes rendus compte que le temps que nous avions estimé pour certaines tâches, par exemple pour le tableau de blocs(nous avions prévu deux heures de temps pour réaliser ceci) s’est avéré largement sous-estimé.

En ce qui concerne les points faibles, le plus important est la complexité du projet qui est trop long par rapport au temps qui nous était donné ensuite, la mauvaise gestion du temps, nous n’avons pas respecté les durées des tâches que nous nous étions fixées au départ et donc le projet n’a pas pu avancer dans les meilleures conditions et nous nous sommes retrouvés coincés sur certains points.

En ce qui concerne les points positifs, nous avons utilisé un système de versioning (Tortoise SVN) qui permet à l’ensemble de l’équipe de suivre l’avancée du projet et donc de pouvoir aider les autres si jamais une personne bloque mais ce n’est pas toujours évident. Ensuite, nous étions conscients que le projet allait être conséquent et donc nous avons dû nous retrouver en dehors des cours prévus à cet effet afin de pouvoir mener le projet à son terme.

Ce qui marche dans notre projet :

* L’écriture dans les fichiers de log
* Le découpage d’un fichier en une liste de bloc
* La configuration des serveurs pour le nombre de connexions, d’envois et de réceptions simultanés.
* L’utilisation de singletons pour les deux configurations (nombre de connexions, etc) ainsi que la liste des fichiers et des serveurs
* L’application s’arrête dès lors qu’on appuie sur echap.

A COMPLETER