Année scolaire 2022 – 2023

Projet de fin de semestre

D-SYNC

Application de
synchronisation de
répertoires

Nom : DOUMIT Geryes, PROUX Marc

Formation : 1A – IR – ENSISA

Table of Contents

Introduction**.....3**

Introduction

Dans le cadre de ce projet Java de fin de semestre, nous avons dû réaliser une Application de synchronisation de répertoires (local et LAN) avec son interface graphique.

Notre objectif était de créer une application grâce à laquelle les utilisateurs pourront synchroniser des répertoires tant sur un seul ordinateur que sur deux connectés au même réseau.

Le cahier des charges était le suivant :

- Cloner un dossier D et un dossier D'
- Synchroniser les deux dossier (si un élément du premier change, les mêmes modifications s'appliquent au second)
- Intégrer une fonctionnalité supplémentaire pouvant rendre l'application plus attractive
- La synchronisation peut se faire en réseau
- L'application doit disposer d'une interface graphique

Vous pouvez retrouver le code source, sa documentation ainsi que ce rapport sur notre dépôt GitHub.

Interface Graphique

Geryes s'est principalement occupé de l'interface graphique. Voici ses commentaires :

Il n'y a pas grand-chose à dire vis-à-vis de l'interface graphique, sachant qu'après avoir compris le fonctionnement de Swing, c'était juste une question d'esthétique et d'implémentation des fonctions de Dsync.java et Network.java (on en parle après dans ce rapport).
Cependant, j'ai quand même quelques points à noter :

1- Threading :

Pour avoir une interface graphique et un programme qui tourne en même temps, il faut absolument utiliser les fonctions Thread de java. En effet, si on ne met pas la synchronisation dans une thread, l'interface graphique se bloque jusqu'à ce que la synchronisation s'arrête, ce que n'arrive jamais parce qu'on est sensés l'arrêter avec cette interface graphique.
C'est pour cela que les classes Dsync.java et Network.java héritent de java.lang.Thread et redéfinissent la méthode run() de Thread. On appelle ensuite la méthode start() de Thread pour lancer leurs méthodes run() dans une nouvelle thread.

2- Thèmes de l'application :

Notre fonctionnalité bonus ! En effet, nous avons mis 4 thèmes par défaut pour l'application, tous basés sur la couleur de l'arrière-plan de cette dernière. De plus, vous pouvez choisir un thème personnalisé avec la couleur de votre choix !
N'hésitez pas à vous amuser avec le sélectionneur de thème, c'est assez cool de voir quelles couleurs rendent bien et lesquelles rendent l'application immonde !

3- Sauvegarde des données utilisateur :

Dans le répertoire d'exécution de l'application, je crée un fichier texte dans lequel je sauvegarde certaines données utilisateurs.
A chaque appui du bouton « Start Syncing » je sauvegarde :

- Les chemins rentrés
- L'adresse IP et le numéro de port rentrés
- La couleur du thème de l'application

Je charge ces données à la réouverture de l'application pour éviter de devoir retaper les mêmes informations ou de rechanger le thème à chaque fois.

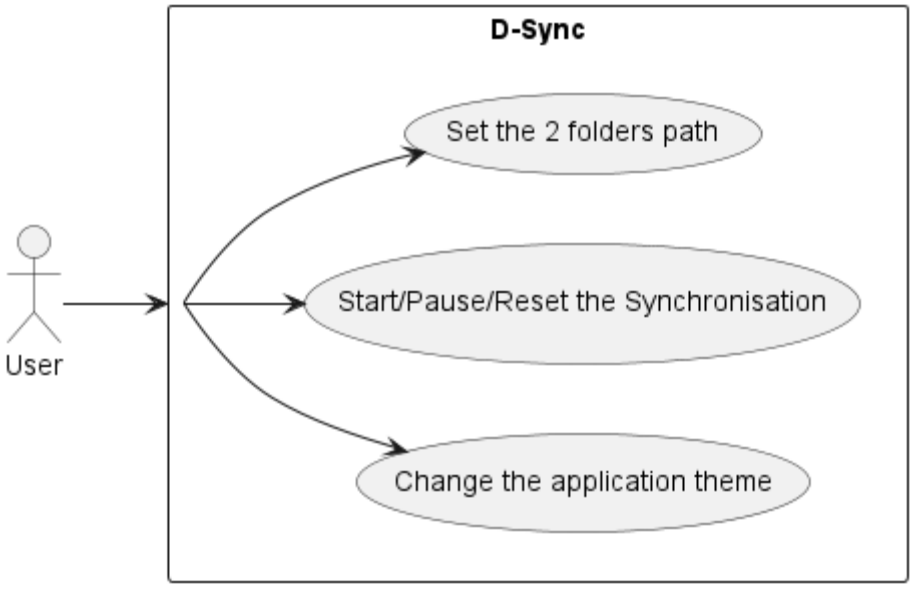
Avant de pouvoir lancer la synchronisation, on vérifie si les chemins existent, et si l'adresse ip et le port sont correctement écrites. En résonnement OCL, voici comment vérifier la conformité de l'ip et du port :

```
context String
inv validIpAddressFormat:
  self.matches('^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.' +
               '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.' +
               '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.' +
               '([01]?\\d\\d?|2[0-4]\\d|25[0-5])$')

context String
inv isNumber:
  self.toInteger() <> null or self.toFloat() <> null or self.toDouble() <> null
```

Synchronisation locale

Geryes s'est occupé de la synchronisation en local. Voici ses commentaires :
Pour la synchronisation en local, j'utilise principalement les chemins des fichiers. En effet, il y a plusieurs fonctions dans Dsync.java, DateAndName.java et Directory.java pour aider à la synchronisation.



1- Première synchronisation :

Au début de la synchronisation, je souhaite éviter de perdre des éléments. La fonction firstSync de Dsync.java (appelée deux fois parce qu'elle donne priorité à l'un des deux dossiers) fait en sorte que les deux dossiers se retrouvent avec les mêmes éléments.
Par exemple, si le dossier A contient a.txt et b.txt, et que le dossier B contient b.txt et c.txt, à la fin de la première synchronisation, les deux dossiers contiendront a.txt, b.txt et c.txt, avec la version la plus récente de b.txt.

2- Synchronisation normale :

Ensuite, nous devons gérer la suppression d'un fichier dans un des dossiers : nous devons le supprimer dans le second. Pour cela, J'utilise la fonction syncAndDelete qui va donner priorité à un dossier, et donc supprimer un fichier de l'autre s'il n'est pas présent dans le dossier prioritaire, ou le copier s'il est présent dans le dossier prioritaire.
Par exemple, si le dossier A contient a.txt et b.txt, et que le dossier B contient b.txt et c.txt, si on donne priorité au dossier B, les deux dossiers synchronisés contiendront b.txt et c.txt, avec la version la plus récente de b.txt.

Enfin, nous devons savoir quelles fonctions appeler et à qui donner la priorité. C'est là que DateAndName.java intervient ainsi que la variable lastState.
La variable lastState est une liste d'objets DateAndName, qui ne sont autre que des objets qui contiennent le nom ainsi que la date d'un fichier. Je l'utilise pour la simple et bonne raison que je ne souhaite que comparer le nom et la date des fichiers pour savoir s'ils diffèrent ou non.

Je donne donc priorité au dossier qui a été modifié par rapport au dernier état synchronisé (lastState), et tout ça se passe dans ma fonction syncLastState de Dsync.java.
Si les deux dossiers ont été modifiés entre deux synchronisations, on refait un fisrtSync pour éviter de perdre des

données.

3- Les fonctions de Directory.java :

Directory.java a un but très simple, c'est d'effectuer des opérations sur les dossiers. Il existe trois méthodes dans cette class, tous en statique :

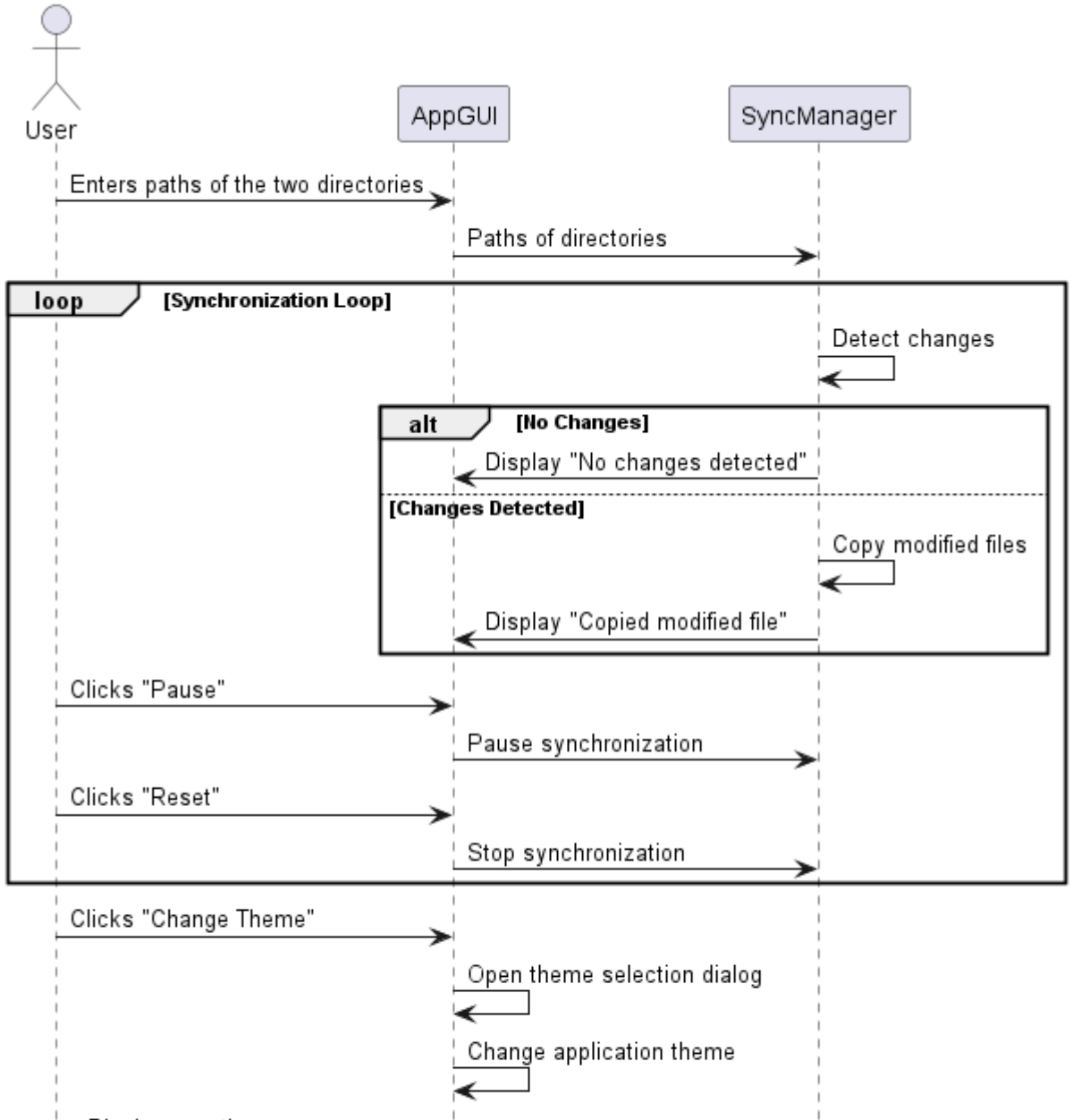
- **copyDirectory(source, destination)** qui copie un dossier et tout ce qu'il contient dans le chemin de destination.
- **deleteDirectory(chemin)** qui supprime un dossier et tous les éléments qu'il contient.
- **lastModifiedDate(fichier)** qui renvoie la date du fichier le plus récent présent dans un dossier, pour savoir quel dossier est plus récent.

4- Interactions de l'utilisateur :

Je vérifie avant chaque synchronisation si les dossiers à synchroniser existent encore, sinon l'utilisateur est notifié et la synchronisation s'arrête jusqu'à ce que le dossier soit remis, ou bien jusqu'à ce que l'utilisateur réinitialise le tout.

Il existe d'autres options pour l'utilisateur à travers le GUI, notamment mettre la synchronisation en pause.

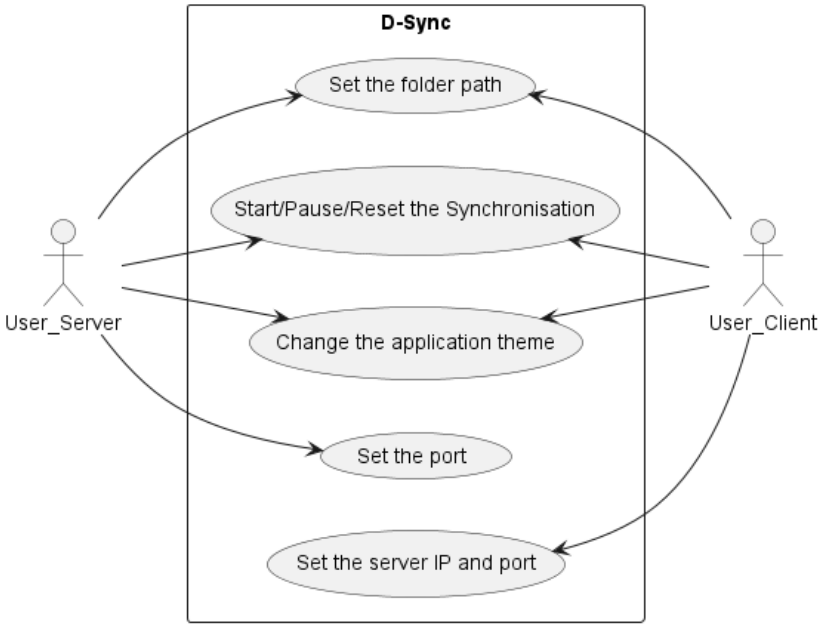
Voici un diagramme de séquence qui indique l'interaction entre l'utilisateur, l'interface graphique, et l'objet qui synchronise les fichiers :

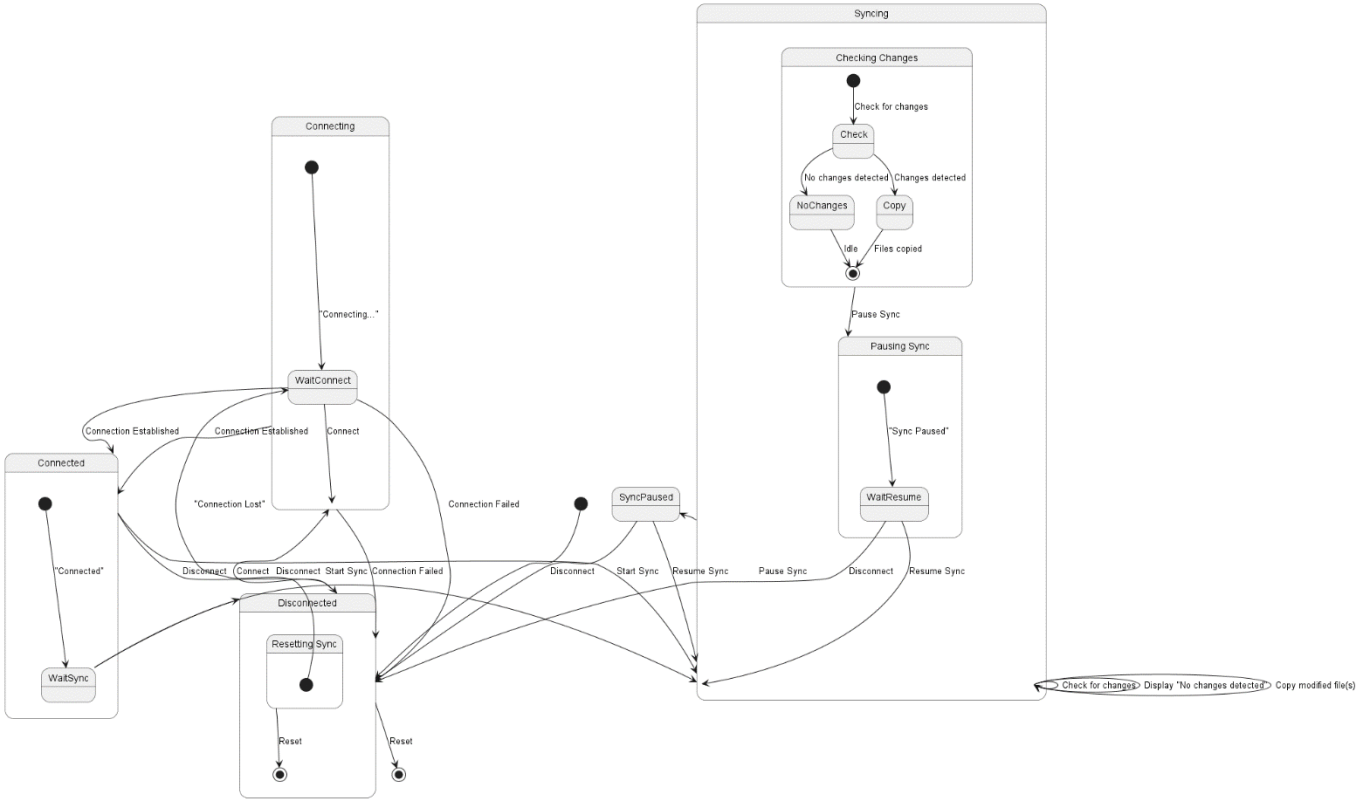


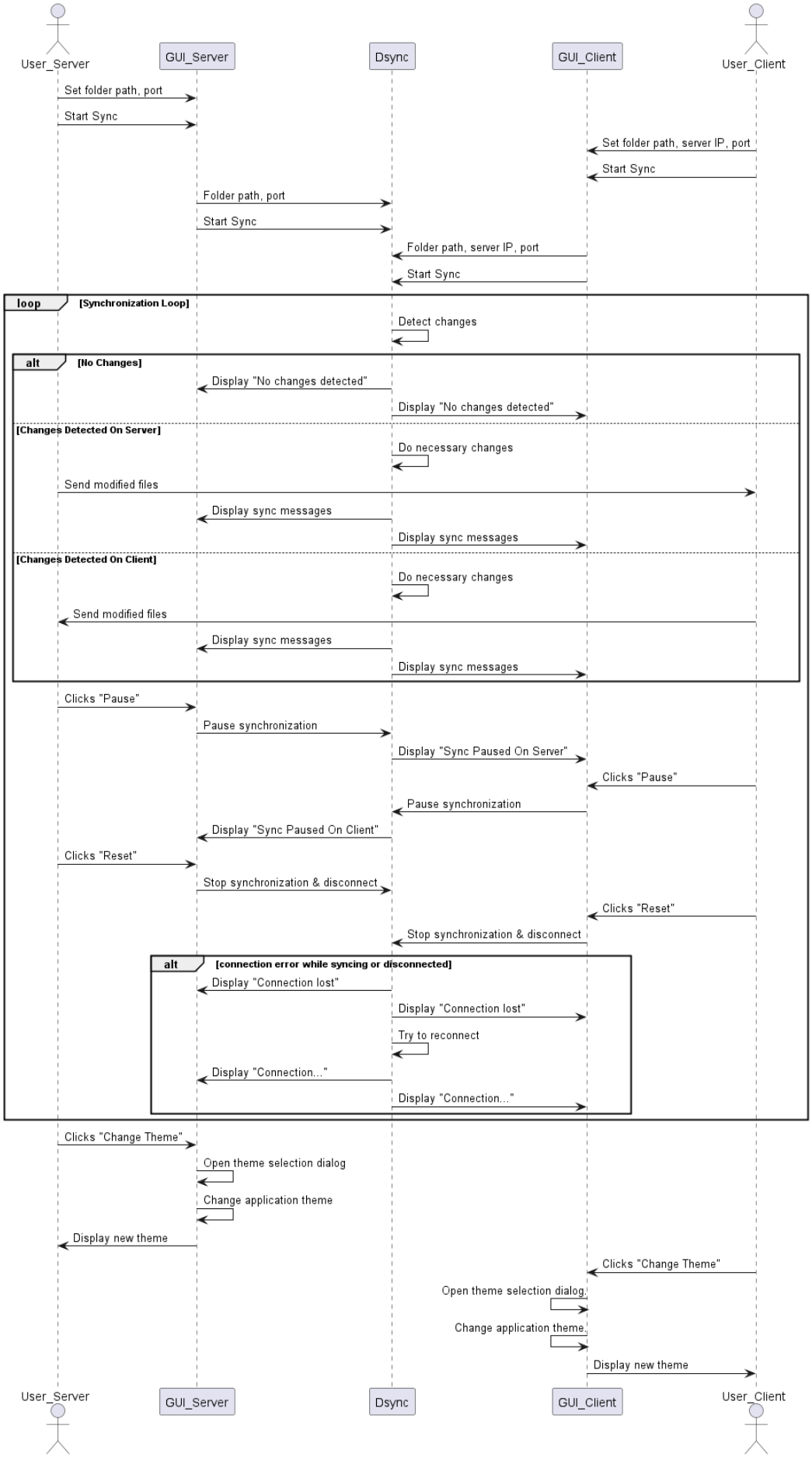
Synchronisation réseau

La synchronisation en réseau de notre application à pour but de connecter deux ordinateurs entre eux et d'échanger les fichiers présents dans leur répertoire de travail afin de les synchroniser et ce, tant que l'application est lancée.

Elle s'exécutera de la manière montrée par les diagramme de cas d'utilisation d'état et de séquence suivants :



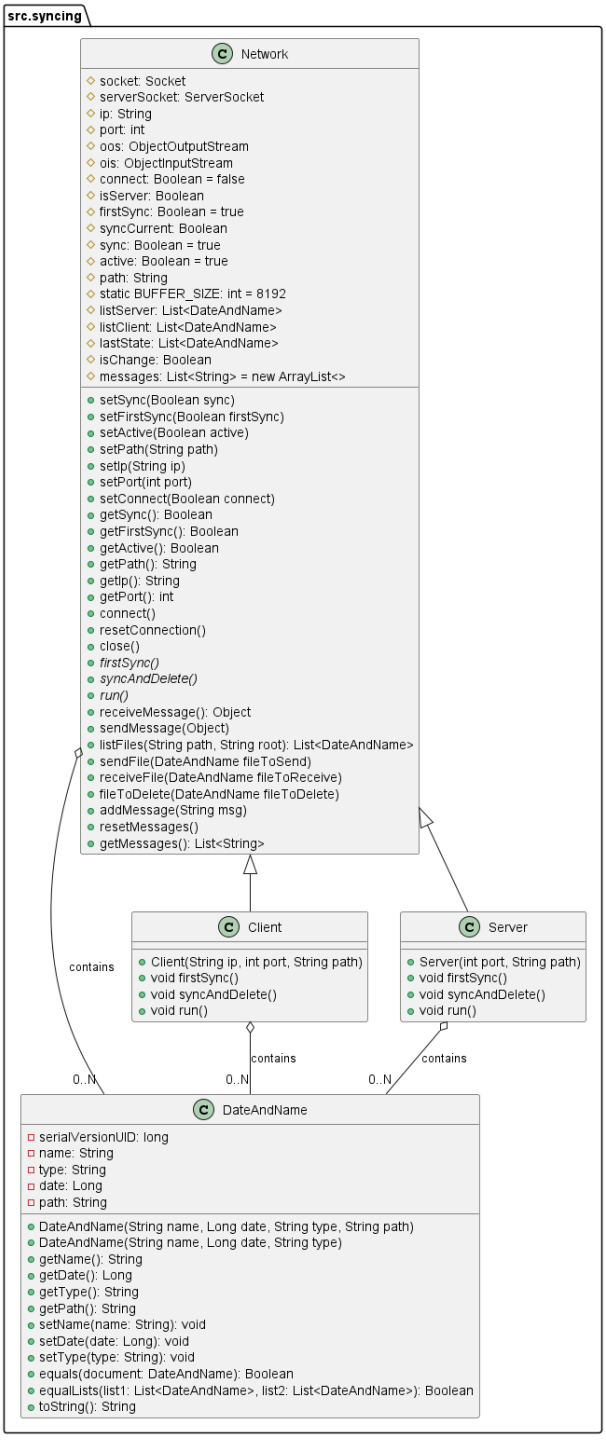



User_Server
GUI_Server
Dsync
GUI_Client
User_Client
Set folder path, port
Start Sync
Set folder path, server IP, port
Start Sync
Folder path, port
Start Sync
Folder path, server IP, port
Start Sync
loop
[Synchronization Loop]
Detect changes
alt
[No Changes]
Display "No changes detected"
Display "No changes detected"
[Changes Detected On Server]
Do necessary changes
Send modified files
Display sync messages
Display sync messages
[Changes Detected On Client]
Do necessary changes
Send modified files
Display sync messages
Display sync messages
Clicks "Pause"
Pause synchronization
Display "Sync Paused On Server"
Clicks "Pause"
Pause synchronization
Display "Sync Paused On Client"
Clicks "Reset"
Stop synchronization & disconnect
Clicks "Reset"
Stop synchronization & disconnect
alt
[connection error while syncing or disconnected]
Display "Connection lost"
Display "Connection lost"
Try to reconnect
Display "Connection..."
Display "Connection..."
Clicks "Change Theme"
Open theme selection dialog
Change application theme
Display new theme
Clicks "Change Theme"
Open theme selection dialog
Change application theme
Display new theme
User_Server
GUI_Server
Dsync
GUI_Client
User_Client

Pour ce faire, j'ai dû créer 3 classes :

- Une classe abstraite : Network
- Deux classes qui héritent de Network : Client et Server

Elles sont représentées dans le diagramme suivant :



1. Network :

La classe Network est la classe « mère » de Client et Server. C'est elle qui contiendra les fonctions nécessaires au bon fonctionnement de la partie réseau.

C'est elle qui contient les fonctions d'envoie et de réception de message ou de fichier ou encore la fonction qui permettra de lister l'ensemble des éléments présents dans le dossier de travaille de chaque machine.

Elle contient également trois fonctions abstraites firstSync, syncAndDelete et run. Bien que leur Override dans les classe Server et Client soient très similaire, j'ai préféré les Override car il y a un système de chassé-croisé dans entre les appels des fonctions sendMessage et receiveFile (quand une machine envoie l'autre doit recevoir). Ces méthodes seront détaillées dans les partie Server et Client.

Vous verrez qu'après chaque appelle du binôme Envoie – Reception mon code fait appelle à la méthode resetConnection(). Sont but est de réinitialiser les buffers afin que l'élément envoyé ne soit pas écrit les uns sur les autres.

Enfin, comme l'a expliqué Geryes dans sa partie sur l'interface graphique, Network hérite de la classe Thread afin que son code puisse s'exécuter en parallèle à celui de l'interface graphique et ainsi ne pas la bloquer durant la synchronisation.

2. Client et Server

Ces deux classes qui héritent de Network sont chargés du bon fonctionnement de la synchronisation. Le choix de quelle classe utiliser entre les deux dépend bien sûr de si la machine est paramétrée en serveur ou en client sur l'interface graphique.

Tout comme dans la partie locale, nous retrouvons les méthodes firstSync(), syncAndDelete() et run(). FirstSync, appelé lors de la première synchronisation. Elle ne supprimera aucun élément. Elle se contente de fusionner les deux répertoires et choisissant les éléments les plus récents si par exemple deux fichiers on le même nom.

SyncAndDelete est la méthode appelé durant tout le reste de la synchronisation. Elle, contrairement à firstSync, va supprimer des éléments si ce dernier est absent dans un des deux répertoires mais présente dans une liste de sauvegarde du dernier état.

La fonction run elle est chargée de faire tourner la synchronisation. Elle boucle sur elle-même seulement si les appareils sont connectés, que la synchronisation n'est pas en pause et que les répertoires de travails existent bien. Elle interrompt la synchronisation et tente une reconnexion si cette dernière est interrompue.

La mise en œuvre de ce système peut se faire en suivant le diagramme de déploiement suivant :

User_Server
Folder path, port
Start Sync
Sync_Manager
Stop synchronization & disconnect
Display sync messages
Display
Sync_Server
Parse
Reset
GUI_Server
Sync Parsed On Client
Open theme selection dialog
Change application theme
User_Client
Display new theme
Set video path and Start Sync
Sync_Client
Parse
Reset
GUI_Client
Sync Parsed On Server
Open theme selection dialog
Change application theme