Javadirectory

Synchroniseur de dossiers local et en réseaux local

ENSISA MULHOUSE
UNIVERSITÉ
HAUTE ALSACE
1A INFORMATIQUE ET RÉSEAUX

Sommaire

Introduction

Cahier des charges :

Développer un outil permettant de synchroniser le contenu de deux dossiers distincts, c'est-à-dire :

Soient D et D' deux dossiers d'un système de fichier, si une modification est faite dans le dossier D celle-ci est reproduite dans le dossier D', de même si une modification est faite dans le dossier D' celle-ci est reproduite dans le dossier D. Une modification se caractérise par la création, la suppression ou la modification d'un fichier ou d'un dossier.

De plus, la modification doit se déclencher automatiquement sans que l'utilisateur de l'outil n'ait à le faire, pour faciliter le développement elle doit dans un premier temps se déclencher si une modification est réaliser dans D ou D' puis dans un second temps automatiquement.

En plus de ces spécifications, l'outil devra aussi pouvoir synchroniser deux dossiers originaires de deux machines différentes, et les interactions avec l'outils devront pouvoir se réaliser grâce à un interface graphique utilisateur.

N.B. : L'outil doit être utilisable et robuste et devra comporter un manuel d'utilisation et intégrer une fonctionnalité additionnelle non détaillé ci-dessus.

Interprétation.

Après avoir étudiée le cahier des charges nous nous sommes mis d'accord pour développer une structure de données qui réalisera, pour résumer de façon grossière, les opérations suivantes :

Soient deux dossiers D et D' d'un même système de fichiers. L'outil que nous allons développer identifiera l'un des dossiers comme étant le dossier source et l'autre comme étant le dossier cible (choix fait par l'utilisateur), puis synchronisera le dossiers cible sur le dossier source (notons que source et cible pourront être inversé à tout moment au choix de l'utilisateur) mais l'inverse ne sera pas possible.

Pour ce faire, l'outil devra analyser le contenu de la source et de la cible. Par la suite, l'outil va soit ajouter les documents manquants dans la cible (en le comparant avec les documents de la source), soit supprimer les documents présents dans le dossier cible et pas dans celui de la source, soit mettre à jour les dossiers qu'elle partage avec la cible en adoptant les modifications de la source.

Pour conclure, on a décidé de se donner une comme restriction que la synchronisation devra se faire seulement dans un sens : d'un dossier source vers un dossier cible. Ce qui n'a une importance que lors de la première synchronisation car lors des réitérations le dossier le moins récent est identifiée comme cible et le plus récent comme source.

Néanmoins pour la synchronisation réseaux ne se fait malheureusement que dans un sens car nous n'avons pas pu saisir tous les aspects de la programmation réseaux.

Fonctionnement

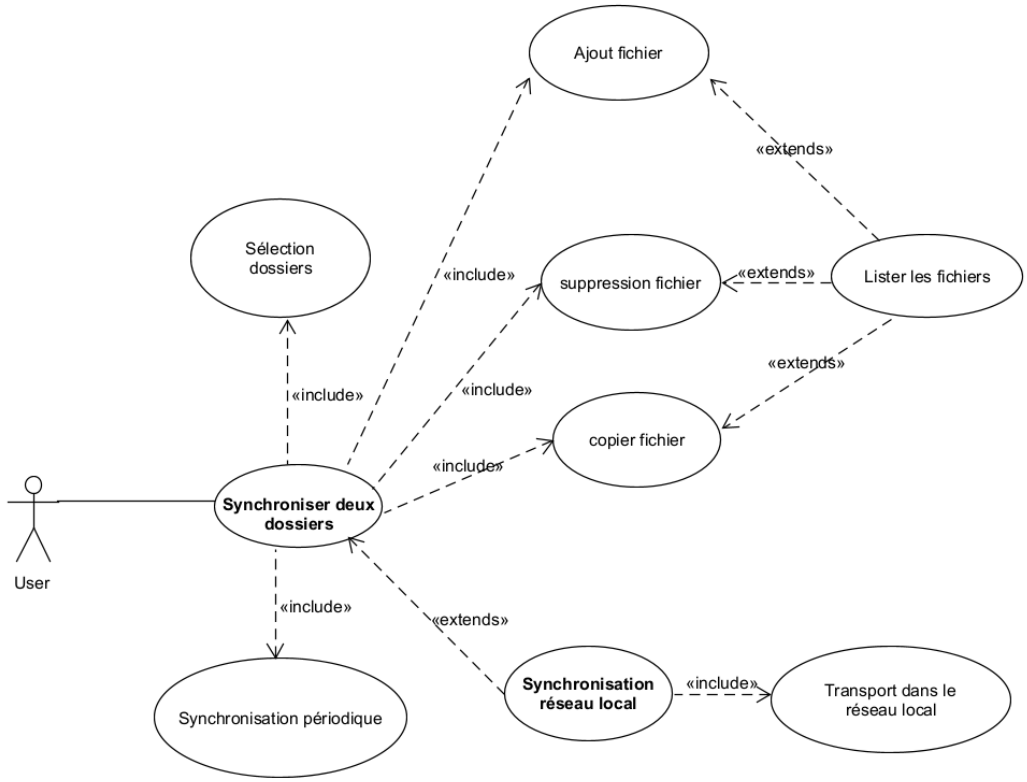


Figure 1 : Diagramme des cas d'utilisations

En local :

Au lancement de l'application l'utilisateur choisit quels sont les deux dossiers qu'il veut synchroniser puis il sélectionne la période T de synchronisation de ses dossiers.

En réseaux local :

Au lancement de l'application l'utilisateur choisit le dossier qu'il veut synchroniser puis, il doit décider s'il est le client ou le serveur. Car le client représente le dossier cible et le serveur le dossier source et le échanges ne se font que de la source vers la cible. Plus tard il aura la possibilité d'être le client s'il le veut.

Ensuite l'application s'exécute dans un processus infinie tournant en tâche de fond, si ensuite il veut l'arrêter il doit ouvrir l'application pour interrompre le processus.

Diagramme des classes

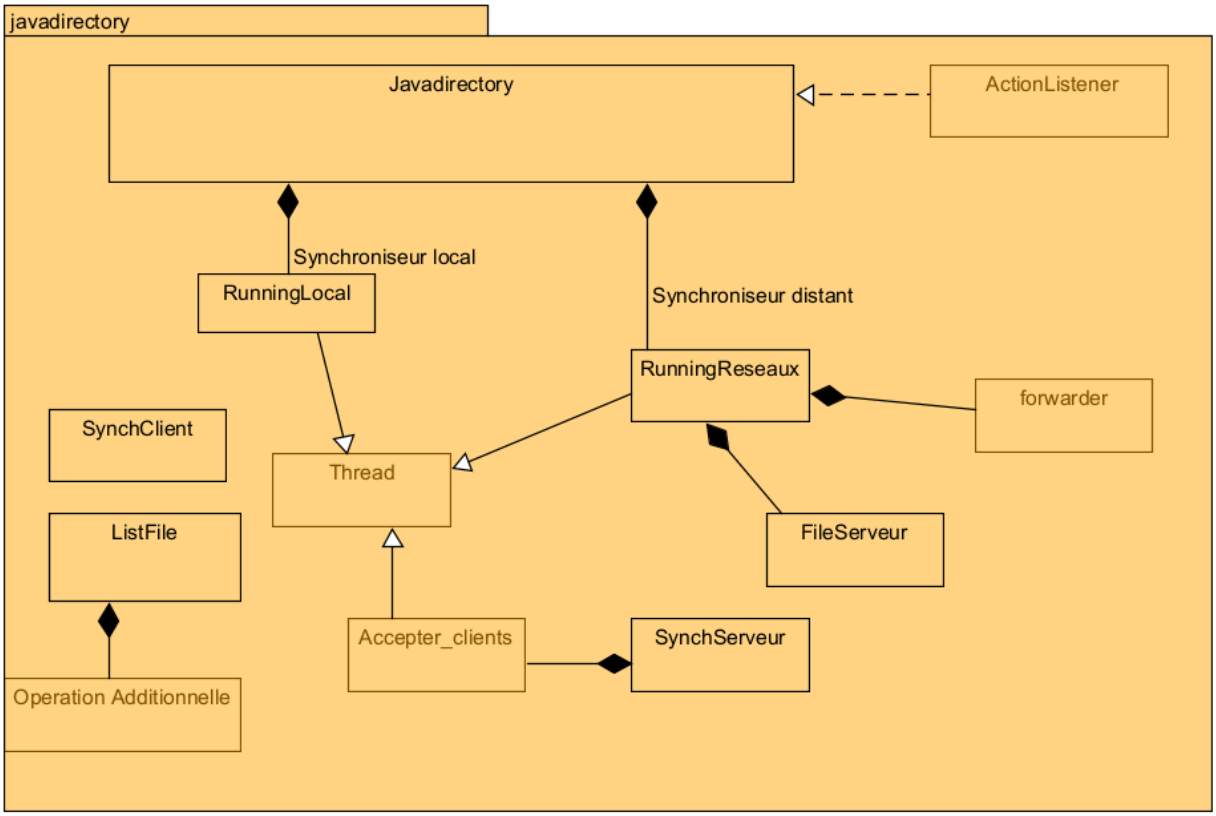


Figure 2 : diagramme des classes simplifié

Ce diagramme des classes simplifié répertorie l'ensemble des classes faisans fonctionner notre application et les relations entre elles.

Notons que l'on a décidé de séparer les opérations réseaux des opérations local.

Fonctionnement temporel

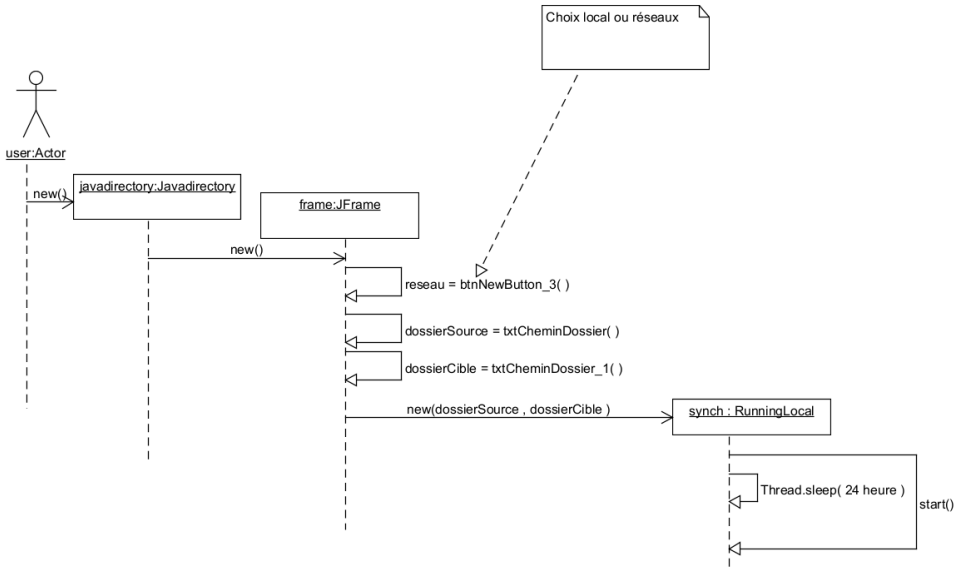


Figure 3 : Utilisation en local

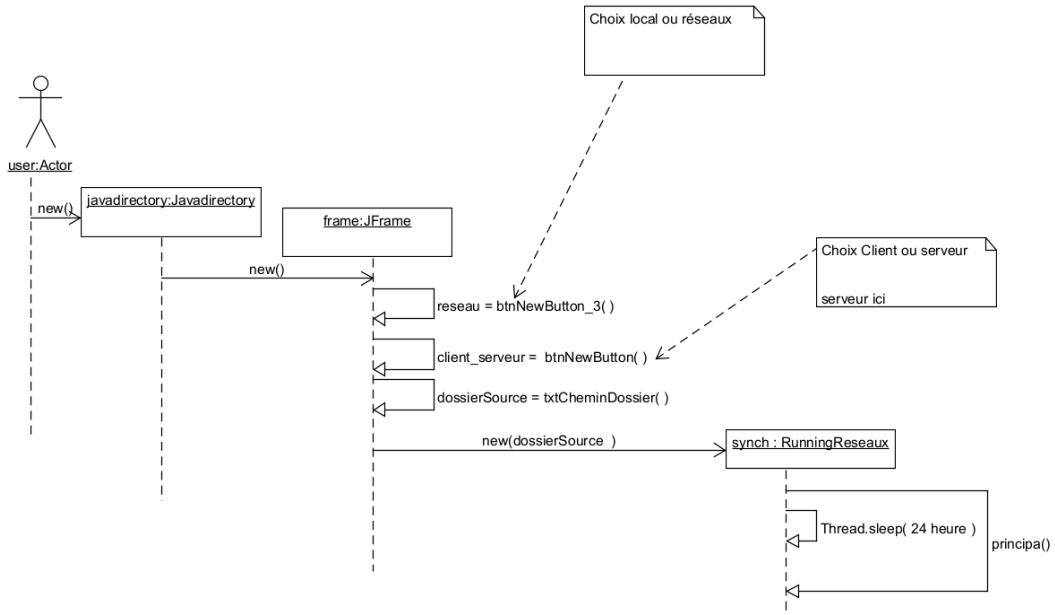


Figure 4 : Utilisation serveur en réseau local

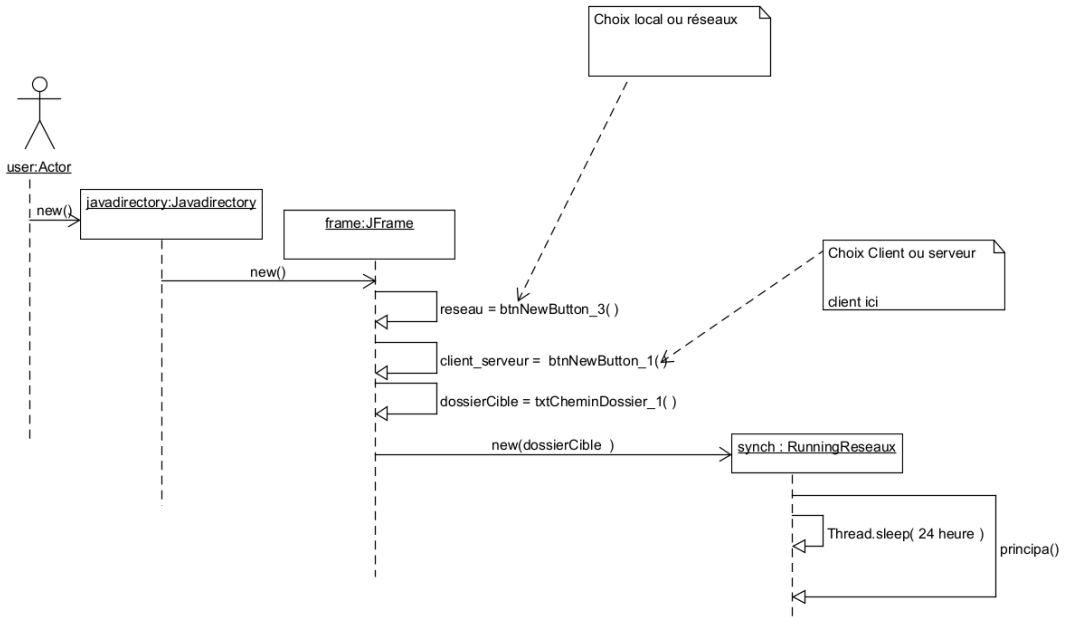


Figure 5 : Utilisation client en réseau local

La figure 3 décrit le fonctionnement global de l'application s'exécutant en local dans un processus infinie. Les figures 4 et 5 décrivent les fonctionnements en réseaux local du serveur et du client.

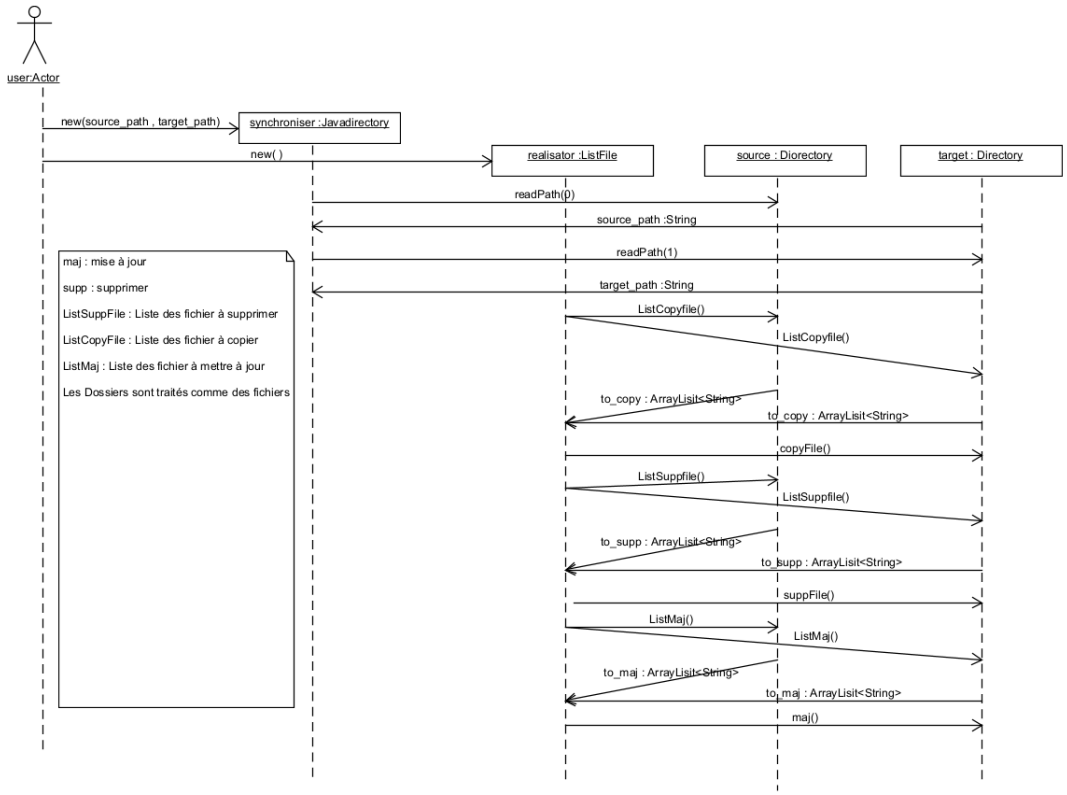


Figure 6 : synchronisation en locale

La figure 6 est une description un peu plus précise des opérations s'exécutant en local pour synchroniser les fichiers.

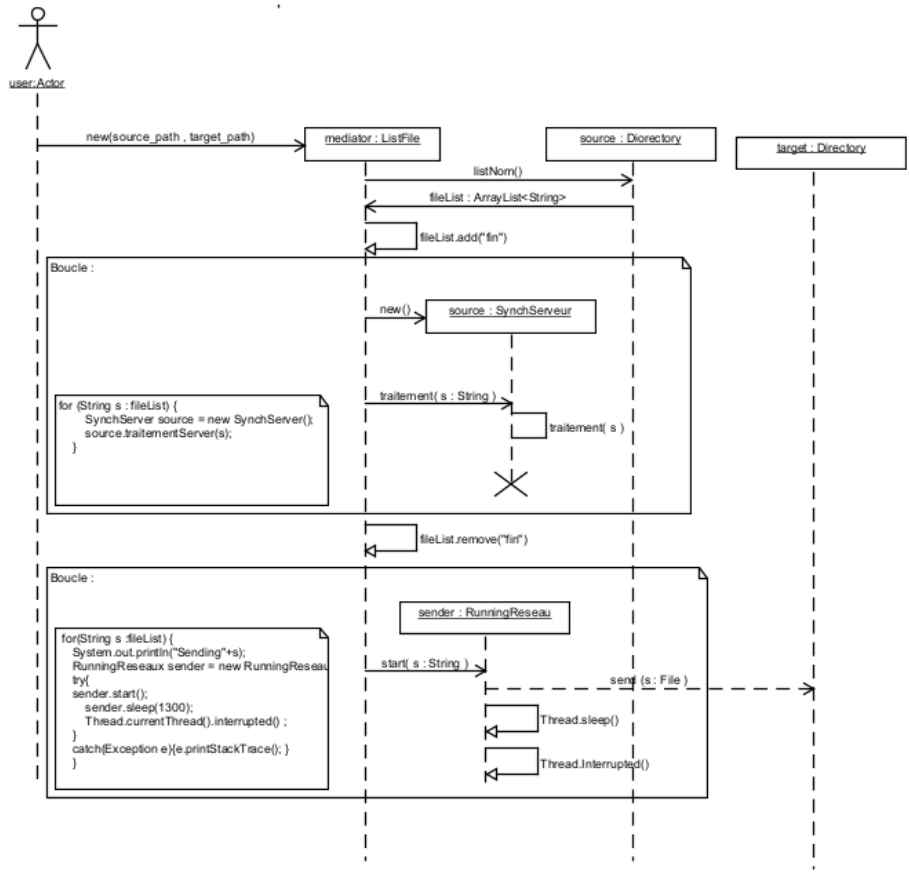


Figure 7 : Synchronisation serveur en réseau Local

La figure 6 est une description un peu plus précise des opérations s'exécutant pour synchroniser 2 dossiers en réseaux local du point de vue du serveur.

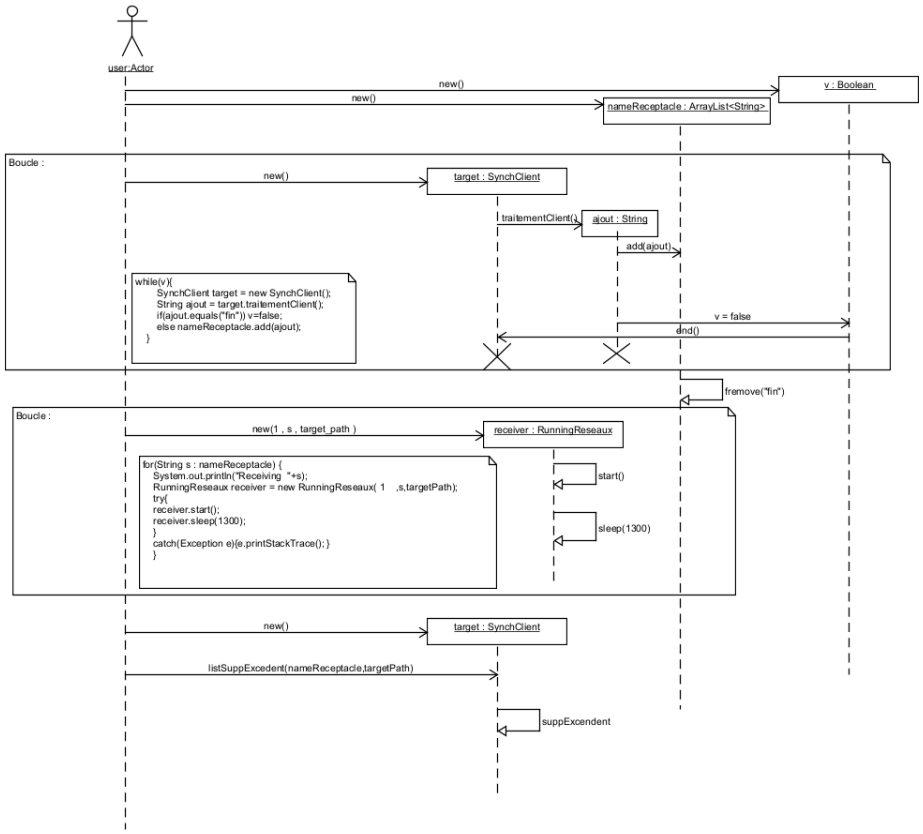
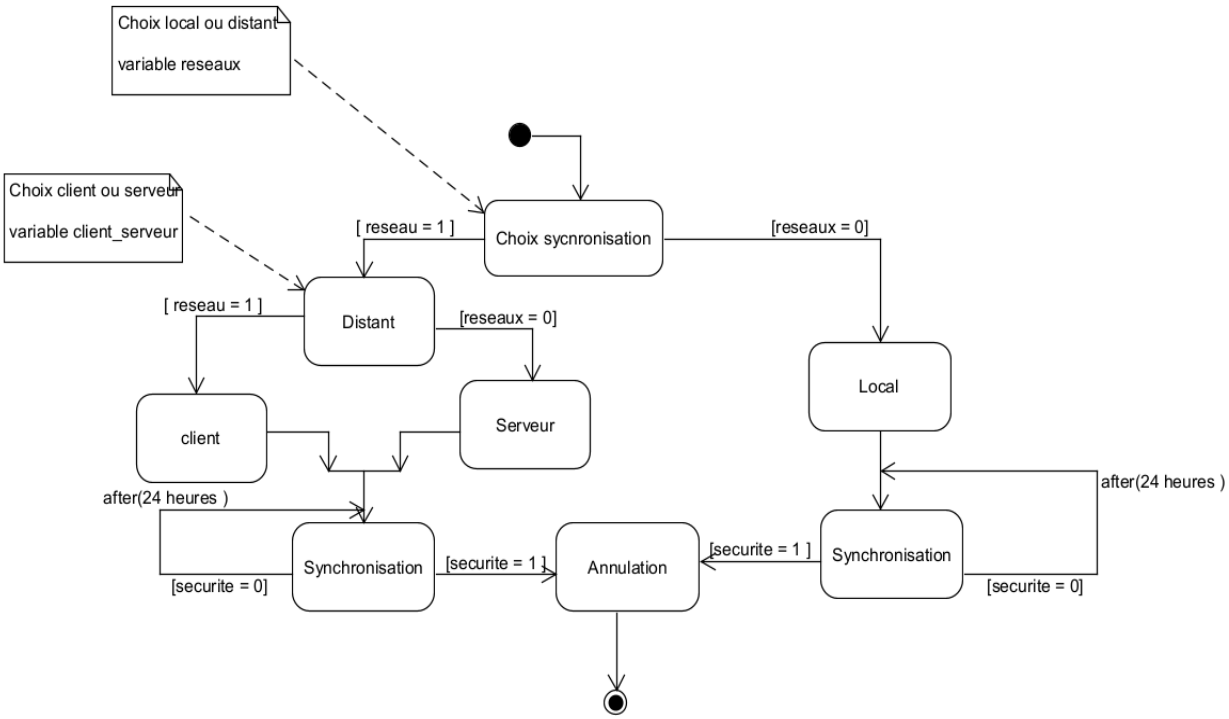


Figure 7 : Synchronisation client en réseau Local

La figure 7 est une description un peu plus précise des opérations s'exécutant pour synchroniser 2 dossiers en réseaux local du point de vue du serveur.

Diagramme d'état



Classes détaillées et restriction

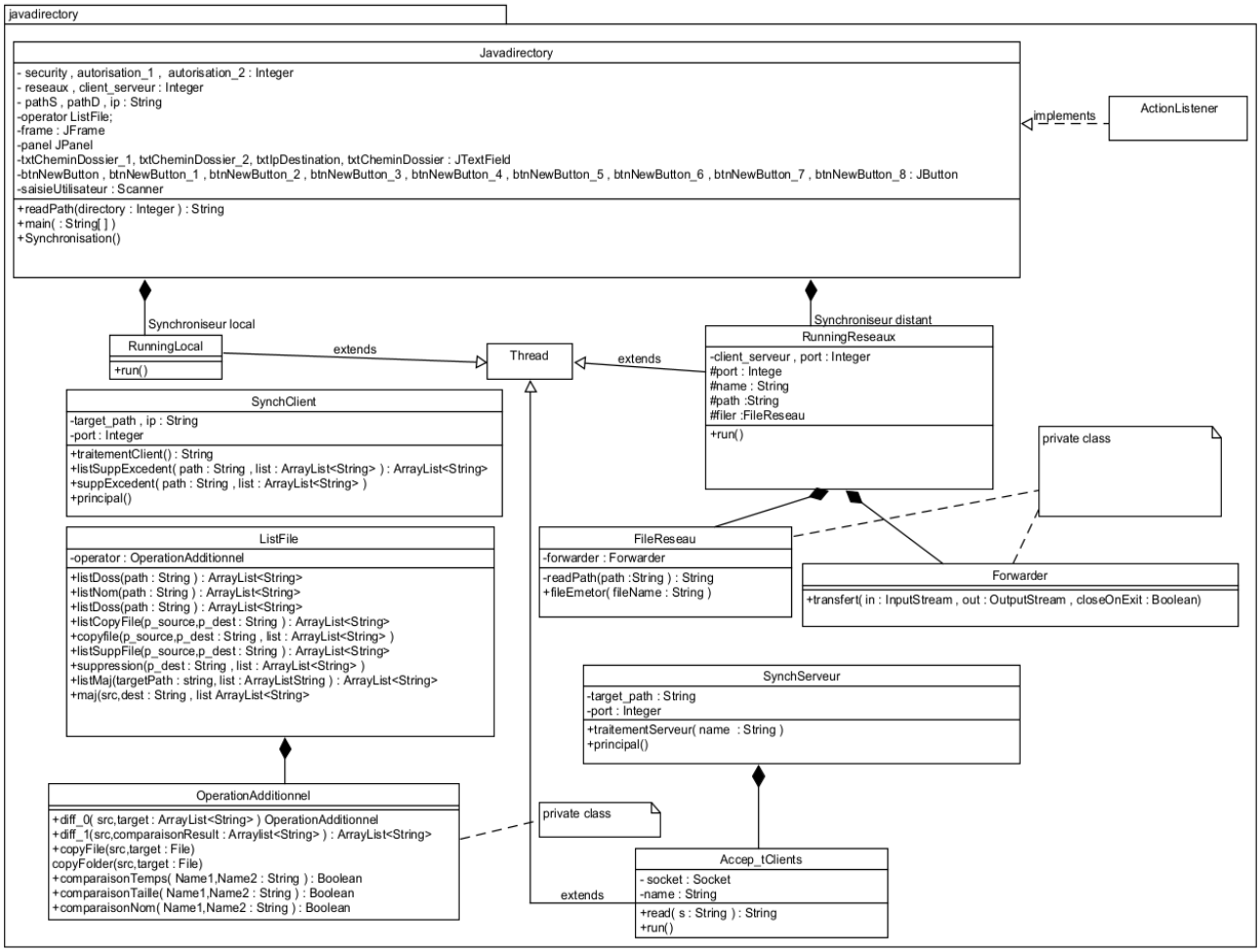


Figure : Diagramme des classes détaillé (cf annexes)

La figure est le diagramme des classes détaillé. On y retrouve aussi toutes les relations existantes entres les classes.

N.B. : La classe OperationAdditionnel est une classe privée encapsulée dans ListFile de même pour Accept_Clients qui est encapsulée dans SynchServeur.

N.B. : Les classes Running Local et RunnintReseaux étendent la classe Thread car on y a surchargé l'opération run pour y implémenter les processus principaux de la synchronisation.

De plus, si la classe Acceptes_Clients étend la classe thread c'est parce que l'envoi des fichier est une branche à part (processus).

Diagramme de communication

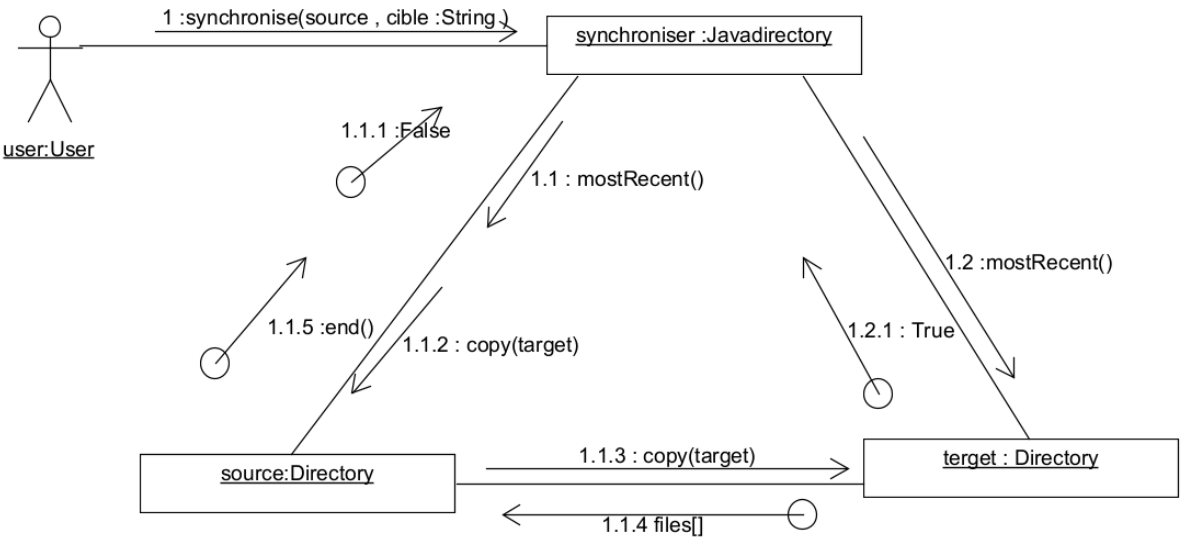


Diagramme de déploiement

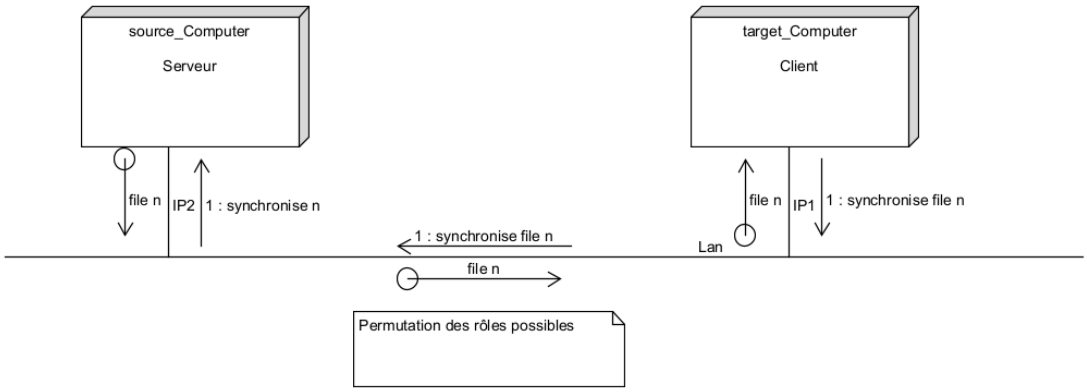


Figure : Diagramme de déploiement

La figure caractérise dans les grandes lignes le fonctionnement de l'application en réseaux. Le client connaissant le nom du fichier n il demande au serveur de lui envoyer sa dernière version ce que celui-ci fait si ce fichier existe dans le répertoire cible et ce jusqu'à ce que l'ensemble des fichiers soit traité.

Puis le client demande au serveur la liste des fichiers qu'elle doit supprimer (c'est-à-dire les fichiers existant côté serveur mais pas côté client) et les supprime.

Difficultés rencontrées

1. L'envoi de fichier en réseau local :

La principale difficulté s'est manifestée lors de cette étape du développement. Lorsque l'on envoyait des fichiers à une autre machine par un port unique le premier fichier s'envoyait correctement puis le port étant occupé, plus rien ne pouvait transiter par celui-ci. On a d'abord pensé à créer un processus pour l'envoi de chaque fichier puis kill celui-ci à la fin de l'envoi mais le même problème se manifeste toujours. Puis on a trouvé une solution grossière que l'on a décidé de garder, on a décidé d'ouvrir un port par envoi de fichier. Ainsi si le dossier considéré compte 100 fichiers on ouvrira 100 ports pour transférer les fichiers ce qui en soit n'est pas un problème sachant qu'un ordinateur possède plus de 60 000 ports.

Manuel d’utilisation

Notes sur l’écriture du manuel d’utilisation :

Lors de l’ouverture de l’outil, l’utilisateur doit sélectionner le type de synchronisation qu’il veut : en réseaux, en local, en tant que client ou en tant que source (serveur). Par la suite il faut ajouter le chemin du dossier correspondant en remplaçant le texte présent.

Par exemple si l’utilisateur clique sur le bouton “Client”, il pourra par la suite choisir si c’est en réseau ou en local. Dans ce cas, il devra seulement y ajouter le chemin du dossier cible. De même, si l’utilisateur clique sur le bouton “Serveur”, il devra seulement y ajouter le chemin du dossier source.

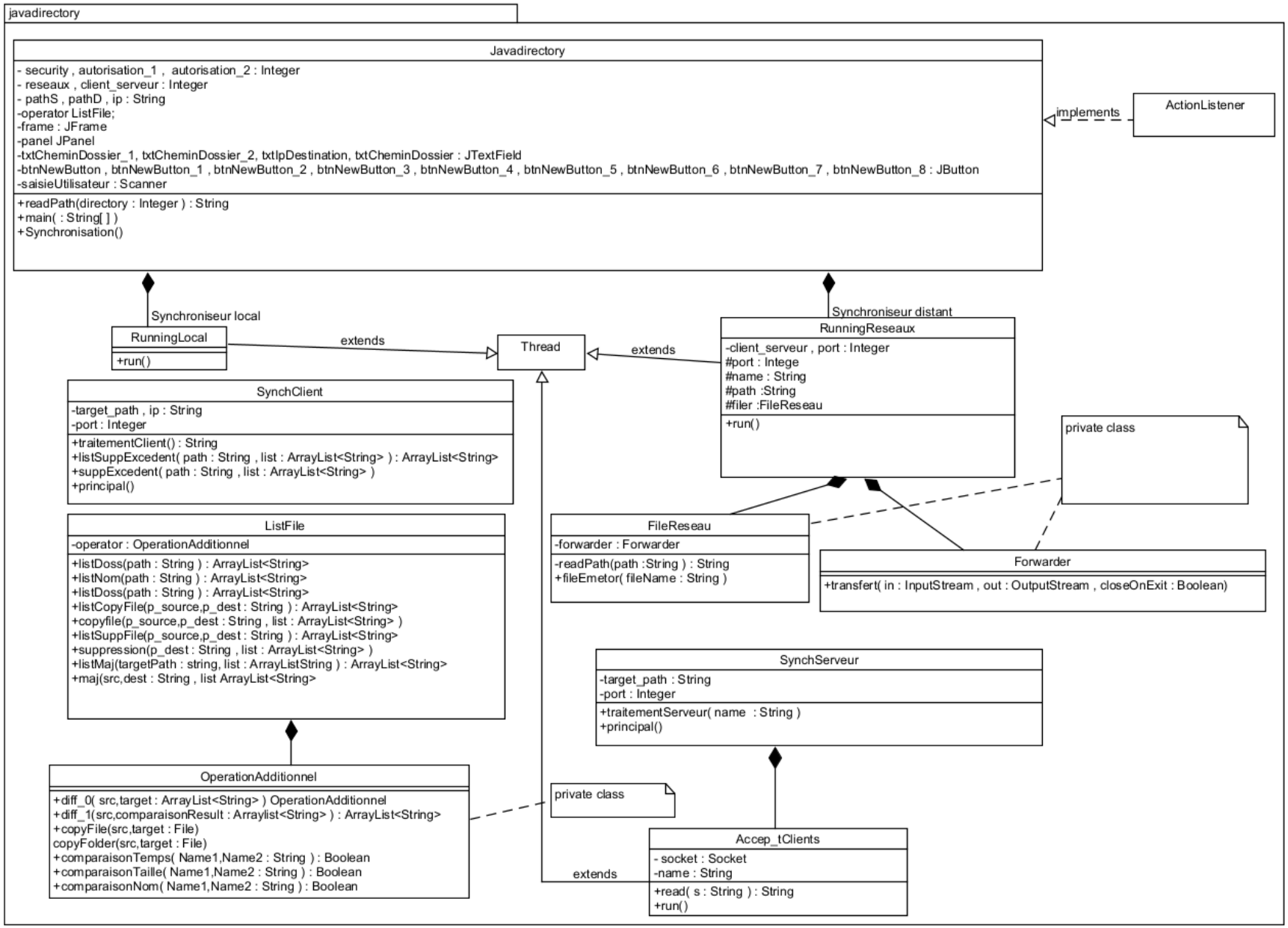
En cliquant sur le bouton “Réseaux” ou “Local”, l’utilisateur devra indiquer sur le champ du haut le chemin du dossier source puis cliquer sur le bouton “Valider” juste en dessous. Puis l’utilisateur devra indiquer sur le champ en dessous le chemin du dossier cible et cliquer sur le bouton “Valider”. Pour commencer la synchronisation, il faut ensuite cliquer sur le bouton “Start”. Concernant la partie Réseaux, il faut bien préciser l’ipv4 de destination et le confirmer en cliquant sur le bouton “Valider” à droite du champ.

Si l’utilisateur souhaite changer de mode (passer du réseaux en local, passer de client à source,…) il faudra fermer puis ouvrir de nouveau l’application.

Conclusion

L'application final respecte les fonctionnalités essentielles demandée par le client et réécrite dans le cahier des charges (cf page 2). A chaque étape, la structure de données interagit avec le client pour lui permettre de s'orienter vers ce qu'il veut et exécute le programme de la meilleure et de la plus confortable des façons.

Annexes


javadirectory
Javadirectory
- security , autorisation_1 , autorisation_2 : Integer
- reseaux , client_serveur : Integer
- pathS , pathD , ip : String
-operator ListFile;
-frame : JFrame
-panel JPanel
-txtCheminDossier_1, txtCheminDossier_2, txtIpDestination, txtCheminDossier : JTextField
-btnNewButton , btnNewButton_1 , btnNewButton_2 , btnNewButton_3 , btnNewButton_4 , btnNewButton_5 , btnNewButton_6 , btnNewButton_7 , btnNewButton_8 : JButton
-saisieUtilisateur : Scanner
+readPath(directory : Integer) : String
+main(: String[ ])
+Synchronisation()
implements
ActionListener
Synchroniseur local
Synchroniseur distant
RunningLocal
+run()
extends
Thread
extends
RunningReseaux
-client_serveur , port : Integer
#port : Intege
#name : String
#path :String
#filer :FileReseau
+run()
SynchClient
-target_path , ip : String
-port : Integer
+traitementClient() : String
+listSuppExcedent(path : String , list : ArrayList<String>) : ArrayList<String>
+suppExcedent(path : String , list : ArrayList<String>)
+principal()
private class
ListFile
-operator : OperationAdditionnel
+listDoss(path : String) : ArrayList<String>
+listNom(path : String) : ArrayList<String>
+listDoss(path : String) : ArrayList<String>
+listCopyFile(p_source,p_dest : String) : ArrayList<String>
+copyfile(p_source,p_dest : String , list : ArrayList<String>)
+listSuppFile(p_source,p_dest : String) : ArrayList<String>
+suppression(p_dest : String , list : ArrayList<String>)
+listMaj(targetPath : string, list : ArrayListString) : ArrayList<String>
+maj(src,dest : String , list ArrayList<String>
FileReseau
-forwarder : Forwarder
-readPath(path :String) : String
+fileEmetor(fileName : String)
Forwarder
+transfert(in : InputStream , out : OutputStream , closeOnExit : Boolean)
SynchServeur
-target_path : String
-port : Integer
+traitementServeur(name : String)
+principal()
OperationAdditionnel
+diff_0(src,target : ArrayList<String>) OperationAdditionnel
+diff_1(src,comparaisonResult : Arraylist<String>) : ArrayList<String>
+copyFile(src,target : File)
copyFolder(src,target : File)
+comparaisonTemps(Name1,Name2 : String) : Boolean
+comparaisonTaille(Name1,Name2 : String) : Boolean
+comparaisonNom(Name1,Name2 : String) : Boolean
private class
Accep_tClients
- socket : Socket
-name : String
+read(s : String) : String
+run()
extends