Laboratoire P2P

Matière	ASI
Professeur	Christian buchs
Assistant	Roand Koszali
Classe	IL-2008
Auteurs	Simon Hintermann, Romain de Wolff

Table des matières

1 Introduction....3

2 Choix de l'architecture....3

3 Spécifications....3

4 Les scénarios....4

4.1 La connection au serveur....4

4.1.1Scénario principal....4

4.1.2Scénario alternatif....4

4.2 Déconnexion d'un client....4

4.2.1Scénario principal....5

4.2.2Scénario alternatif....5

4.3 La demande de chargement d'un fichier....5

4.3.1Scénario principal....5

Scénarios alternatifs....5

4.4 Vérification d'existence des clients....6

4.4.1Scénario principal....6

5 Les cas-tests....7

1 Introduction

Le but de ce laboratoire est de créer un protocole applicatif permettant de faire des échanges peer-to-peer entre 2 ou plusieurs personnes. L'application ne doit pas faire grand-chose, mais il faut mettre l'accent sur la conception.

2 Choix de l'architecture

Notre architecture sera de type « serveur centralisé », avec le premier utilisateur connecté comme serveur-annuaire. C'est simplement lui qui fera office de distributeur des demandes.

Ce choix nous semblait le plus adapté pour une application de cette ampleur, car il faudrait implémenter des structures de routage assez complexes dans le cas d'un architecture semi-distribuée ou complétement ditribuée. De plus, l'architecture complétement distribuée génère beaucoup de messages, donc des protocoles de communication plus compliqués.

3 Spécifications

Numéro	*Description*
1	L'interface graphique doit être très simple
2	L'application doit pouvoir se connecter et se déconnecter à l'aide d'un seul bouton
3	L'application doit se fermer via un bouton dédié
4	La fermeture de l'application doit se faire proprement
5	Lors du lancement, le nom de l'utilisateur est passé en paramètre
6	Le nom de l'utilisateur est affiché dans l'application
7	Le répertoire contenant l'application doit aussi contenir deux dossiers: 'share' et 'download'
8	Tous les fichiers présents dans le dossier 'share' sont partagés pour les autres utilisateurs connectés
9	Tous les types de fichier doivent pouvoir être partagés
10	L'application va partager les fichiers de tous les utilisateurs en affichant les fichiers à disposition avec le nom de l'utilisateur qui en est propriétaire
11	Un utilisateur ne verra pas la liste de ses propres fichiers à télécharger
12	Les utilisateurs ne peuvent télécharger qu'un fichier à la fois
13	Les fichiers téléchargés sont déposés par l'application dans le dossier 'download'
14	Si une communication est interrompue, le fichier est considéré comme perdu, il faudra relancer la transaction
15	Si une transaction est interrompue, l'application le signale à l'utilisateur
16	Aucune communication de broadcast ne doit être utilisée

17	Les temps de réponses de l'application doivent se situer dans une fourchette de 1- 5 secondes
18	Si le dossier 'share' n'existe pas, l'application ne se lance pas
19	Si le dossier 'download' n'existe pas, l'application ne se lance pas
20	L'application doit être robuste, transfert de fichiers fiable
21	

4 Les cas d'utilisation

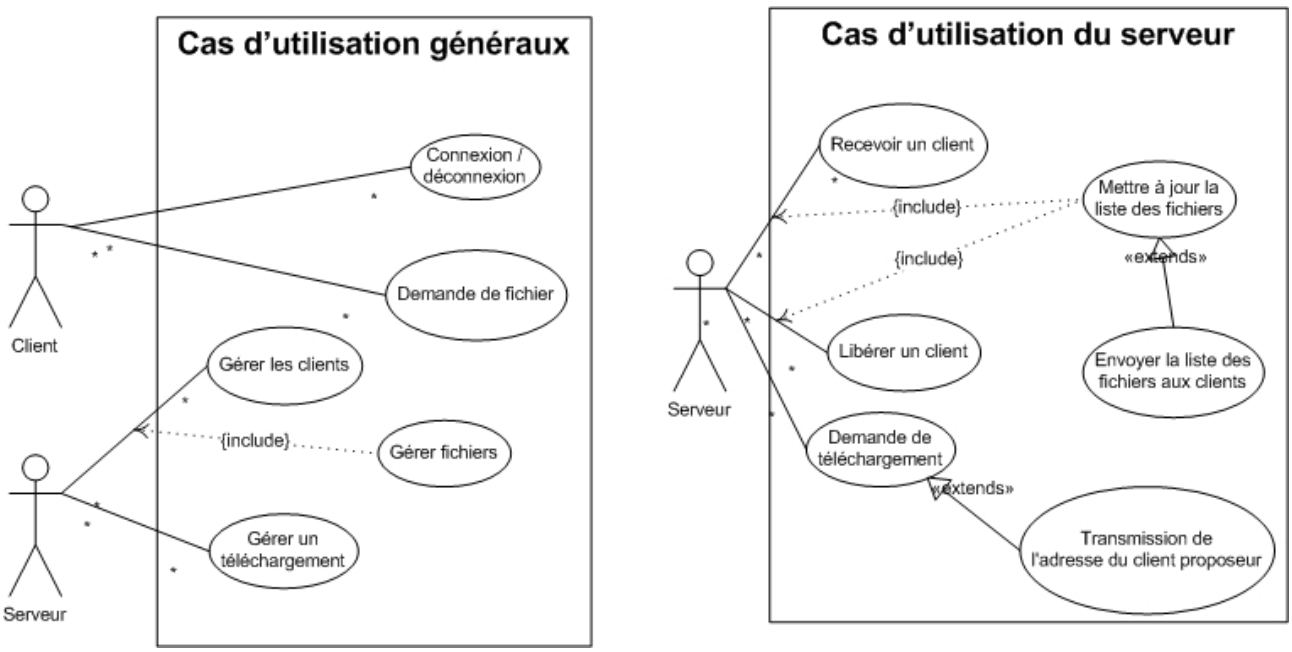

Illustration 1: Les cas d'utilisation

La figure 1 présente les cas d'utilisation de notre application. La plus grosse partie de gestion se trouve du côté du serveur, comme on le voit, c'est lui qui va gérer toutes les transactions et la cohérence des clients.

Comme on le voit, le travail principal du serveur est de lier les clients entre eux afin qu'ils puissent s'échanger des fichiers, mais il doit aussi (et surtout) maintenir la cohérence des informations fournies au clients. Effectivement, il est primordial que le client ait bien la liste des fichiers pouvant être téléchargés, et non

Ces cas d'utilisation seront expliqués ci-après dans le rapport. Nous allons faire état des scénarios du déroulement de l'application, puis des schémas d'activité, des schémas de séquence (messages échangés), et enfin, du diagramme de classes.

5 Les scénarios

5.1 La connection au serveur

Ce scénario représente la volonté du client de se connecter au serveur pour obtenir la liste des fichiers partagés par le réseau peer-to-peer, ainsi que proposer sa propre liste de fichiers.

5.1.1 Scénario principal

1 Un client se connecte au serveur centralisé du réseau peer-to-peer.

2 Le serveur enregistre le client dans sa liste de clients

3 Le serveur met à jour sa liste des fichiers partagés

4 Le serveur renvoie au client la liste des fichiers partagés avec les utilisateurs correspondant, sauf les fichiers appartenant à ce client

5 Le client affiche tous les fichiers partagés

6 Le serveur boucle sur tous les autres clients enregistrés et pour chacun, il réitère les points 4 et 5

5.1.2 Scénario alternatif

1.1 Le serveur n'existe pas, impossible de se connecter

1.2 Le client renvoie une erreur à l'utilisateur l'avertissant que le serveur n'est pas disponible

5.2 Déconnexion d'un client

C'est ici le cas où, volontairement, un client désire se retirer du réseau peer-to-peer. Cette déconnexion est la déconnexion « propre », nous verrons plus loin que faire d'un plantage de l'application, ou d'une coupure de réseau.

5.2.1 Scénario principal

1 Le client se déconnecte du serveur

2 Le serveur met à jour sa liste de clients

3 Le serveur met à jour sa liste de fichiers partagés

4 Le serveur renvoie les fichiers partagés à tous ses clients

5 Le serveur renvoie un ACK au client déconnecté

6 Le client déconnecté avertit l'utilisateur que tout s'est bien passé

5.2.2 Scénario alternatif

1.1 Le serveur n'existe pas, impossible de se connecter

1.2 Le client annonce que la déconnexion est faite, mais que le serveur n'existe plus

5.3 La demande de chargement d'un fichier

Le client désire charger un fichier. Il lui suffira de choisir un des fichiers proposés dans sa liste mise à jour par le serveur, puis d'entrer le numéro correspondant au fichier voulu.

Cette phase implique que le serveur soit sûr que le client propose toujours le fichier. Il va donc envoyer une demande au client proposeur avec le nom du fichier voulu. Le client proposeur va alors renvoyer au serveur une réponse positive ou négative.

5.3.1 Scénario principal

1 Le client demande un fichier de la liste au serveur

2 Le serveur teste si le clients proposant le fichier existe toujours

3 Le serveur renvoie l'adresse d'un client qui propose le fichier

4 Le client demandeur se connecte au client proposeur avec le nom du fichier voulu

5 Le client proposeur transfère le fichier au client demandeur

6 Le fichier est enregistré dans le dossier 'download'

Scénarios alternatifs

1 Le client demande un fichier de la liste au serveur

1.1 Le serveur n'existe plus, impossible de se connecter

1.2 Le client avertit l'utilisateur qu'il est déconnecté, que le serveur n'existe plus

2 Le serveur teste si le clients proposant le fichier existe toujours

2.1 Le client n'existe plus, ou ne propose plus le fichier

2.2 Une erreur est renvoyée au client comme quoi le fichier voulu n'est plus disponible

3 ...

4 ...

5 Le client proposeur transfère le fichier au client demandeur

5.1 Le transfert a été interrompu

5.2 Le fichier est considéré comme perdu, le client affiche une erreur

5.4 Vérification d'existence des clients

La vérification des clients doit être faite afin d'être sûr que les informations que tous les clients toujours connectés sont cohérentes. Si un client plante son application, le serveur n'en sera pas averti, et les autres clients ne seraient pas au courant que les fichier correspondant au client déconnecté ne sont plus disponibles.

En réalité, on pourrait ne faire cette vérification que quand un client demande un fichier, mais il nous semble plus approprié de savoir assez tôt si un client n'existe plus. De cette manière, le serveur garde un état cohérent des fichiers partagés pratiquement en permanence.

5.4.1 Scénario principal

1 Le serveur, toutes les minutes environ, va demander aux clients de sa liste s'ils existent toujours

1.1 Si un des clients n'existe plus, le serveur va mettre à jour sa liste de clients et sa liste de fichiers

2 Une fois tous les clients contactés, s'il y a eu des déconnexions non-déclarées, le serveur renvoie sa liste de fichiers partagés à tous les clients encore existants

6 Les cas-tests

Numéro	No Spec	Procédure	Résultat attendu	OK	KO	Commentaire
1	2	Faire un essai de connexion si le serveur existe	Le client doit se connecter au serveur (message de bienvenue?)			
2	2	Faire un essai de connexion alors que le serveur ne tourne plus	Le client doit afficher une erreur, puis quitter			
3	2	Faire un essai de déconnexion si le serveur existe toujours	Le client doit afficher un message comme quoi la déconnexion s'est bien passée			
4	2	Faire un essai de déconnexion si le serveur n'existe plus	Le client doit prévenir l'utilisateur que le serveur n'existe plus, mais qu'il est déconnecté			
5	3	Essayer de fermer l'application sans s'être déconnecté	Le client doit prévenir l'utilisateur que la déconnexion va d'abord être effectuée			
6	3	Essayer de fermer l'application	Le client affiche une boîte modale de confirmation			
7	4	Essayer de fermer l'application sans s'être préalablement déconnecté	Le(s) port(s) utilisé(s) par l'application doivent être libres après la fermeture de l'application			
8	5	Essayer de lancer l'application sans nom d'utilisateur	Le client refuse de se lancer et affiche un message d'utilisation correcte			
9	6	Lancer le client avec un nom d'utilisateur	Le nom d'utilisateur passé en paramètre doit être affiché dans le client			
10	7	Lancer le client sans avoir le dossier	Le client doit afficher une erreur,			

		'share'	puis quitter			
11	7	Lancer le client sans avoir le dossier 'download'	Le client doit afficher une erreur puis quitter			
12	8	Lancer deux clients normalement, chacun avec des fichier sauvegardés dans leur dossier 'share'	Le client 1 doit voir les fichiers partagés par le client 2, et inversément			
13	9	Télécharger une dizaine de fichiers différents d'un client à un autre	Les fichiers doivent tous pouvoir être téléchargeables et utilisables			
14	10	Lancer le client normalement	Le client doit afficher la liste des fichiers partagés par le réseau			
15	10	Fermer l'application, ou se déconnecter	Les autres clients doivent recevoir une mise à jour de l'affichage des fichier partagés qui ne comprend plus les fichiers du client déconnecté			
16	10	Connecter trois clients avec des fichiers partagés	Chaque client doit voir les fichiers des deux autres clients avec: 'nom du fichier' - 'propriétaire'			
17	11	Lancer le client normalement	Le client ne doit pas voir ses propres fichiers partagés			
18	12	Télécharger un fichier avec un client	Le client ne doit plus pouvoir refaire la demande avec un autre fichier			
19	13	Télécharger un fichier avec un client	Le fichier doit apparaître dans le dossier 'download' à la fin du téléchargement			
20	14	Télécharger un fichier. Pendant le téléchargement, couper le client qui propose le fichier	Le fichier en téléchargement doit être considéré comme étant perdu			
21	15	Interrompre une transaction	L'erreur doit être signalée au client demandeur			
22	16	Connecter un client tout en observant	Les trames envoyées par			

		le trafic réseau avec Wireshark	l'application ne doivent en aucun cas être du broadcast			
23	18	Supprimer le dossier 'share', et essayer de lancer l'application	L'application affiche un message d'erreur, puis se ferme			
24	18	L'ancer l'application, supprimer le dossier 'share', faire une demande de téléchargement d'un autre client pour un des fichiers de ce 'share'	Le serveur doit signaler l'erreur au client demandeur, le client proposeur doit être quitté avec un message d'erreur			
25	19	Supprimer le dossier 'share', et essayer de lancer l'application	L'application affiche un message d'erreur, puis se ferme			
26	19	Lancer l'application, supprimer le dossier 'download', puis télécharger un fichier	L'application doit émettre un message d'erreur, couper la connexion avec le serveur, et enfin quitter			
27	20					

7 Références

8 Logiciels utilisés

1. Microsoft Visio: pour tous les schémas de conception
2. Open Office: pour réaliser ce rapport