École Nationale Supérieure de Sciences Appliquées et de Technologie
6 rue de Kérampont
22300 Lannion
Tel : *02 96 46 90 00*
Fax : *02 96 37 01 99*
Courriel : *contact@enssat.fr*

Algorithmique distribuée - Rapport de projet - Enshare

Enshare est un logiciel d'édition de texte collaboratif. Dans le cadre de notre projet relatif au module d'algorithmique distribuée, il nous est demandé de modifier ce logiciel pour mettre en oeuvre les différents concepts étudiés en cours. Le présent rapport présente notre travail de réflexion et nos chois d'implémentation. Il vient en complément du code précédemment fourni.

Laurent Laluque
llaluque@enssat.fr
Baptiste Legeron
blegeron@enssat.fr
Javid Mougamadou
jmougama@enssat.fr
LSI 2
Le **24 mai 2015**

Table des matières

1 Introduction

Ce projet à eu lieu après le cours d'algorithmique distribuée dans le but de nous permettre une mise en oeuvre de nos connaissances.

Le but de ce projet était de passer d'un programme qui permettait de partager une liste de documents contenus dans un serveur à une situation aux possibilités identiques, mais sans le serveur. Le programme de départ permettait de modifier les documents en question de manière à ce que tous les autres clients voient en direct les changement faits. Cependant, ces changements étaient faits via des exclusions mutuelles gérées par le serveur. Notre but principal est donc de les décentraliser.

Pour réaliser ce projet en entier, nous devions passer par trois étapes, établissant des cas de réalisation successifs.

2 Le projet Enshare

2.1 Le projet dans sa globalité

Le projet *Enshare* consiste à mettre en application en Java les concepts étudiés lors du cours l'algorithmique distribuée. Il s'agit de prendre en main le code d'un logiciel d'édition de texte collaboratif, et d'en modifier l'architecture. En effet, l'exclusion mutuelle au niveau de l'écriture entre les clients est initialement gérée par un serveur. Le but de ce projet est de faire en sorte d'externaliser cette exclusion mutuelle et d'en permettre la gestion exclusivement par les clients.

2.2 Objectifs et avancement final

On distingue trois cas pour l'accomplissement du projet :

— le cas 1 consistant à mettre en place un mécanisme d'exclusion mutuelle sur les clients uniquement, document par document
— le cas 2 consistant à porter cette exclusion mutuelle ligne par ligne
— le cas 3 consitant à, en plus, faire en sorte que le serveur n'intervienne même plus pour la sauvegarde des documents

Notre objectif majeur était le cas 1, même si nous avons émis une réflexion pour les cas 2 et 3. Au final, le cas 1 est globalement fonctionnel, à quelques fonctionnalités près, détaillées ci-dessous. Nous n'avons pas pu mettre en oeuvre le cas 2, principalement car notre algorithme d'exclusion mutuelle est différent pour les deux cas.

2.3 Organisation

Au début, nous nous sommes tous concentrés sur la partie théorique du cas 1 avec une première ébauche CSP obtenu très rapidement grâce à une bonne analyse puis chacun devait s'occuper d'implementer chaque cas. Cependant, nous avons eu des problèmes de compréhension vis a vis du fonctionnement du code de l'*Enshare*, c'est pourquoi nous nous sommes tous refocalisés sur l'implémentation du cas 1. Lors des séances encadrées,

nous nous sommes tous les trois concentrés sur le code afin de le comprendre et de commencer à la modifier. Après les séances, nous avons dû tous les trois prendre du temps pour comprendre ce qu'il nous faudrait concrètement réaliser. Enfin, vers la fin du projet, Baptiste et Javid se sont occupées d'implementer le cas 1 à partir de l'analyse effectuée et Laurent a effectué les tests nécessaires à la validation de l'implémentation du cas 1. Il a également réfléchi à certains aspects théoriques, notamment l'algorithme en cas de déconnexion d'un client. Il a également fait la présentation Google Slides pour la démonstration et rédigé une grande partie du rapport.

3 Aspects théoriques

3.1 Spécifications générales

D'une façon générale, le but de ce projet est de modifier le logiciel *Enshare*, éditeur de texte collaboratif. Le problème à traiter vis-à-vis de ce logiciel est l'accès en simultané par plusieurs utilisateurs à un bloc-note à modifier. On retrouve ici une problématique d'accès à une ressource partagée. Dans la version du logiciel proposée, cette exclusion mutuelle est résolue de façon centralisée (cas 0) : elle est gérée par le serveur qui fait le lien entre tous les clients. Très concrètement, l'IHM est pourvue d'un bouton *Verrouiller* que l'utilisateur doit presser avant de faire ses modifications. Si le document est déjà verrouillé par un autre client, alors le verrouillage est refusé, et le client devra retenter, de lui-même, plus tard. (Il n'y a pas de gestion de file d'attente des demandeurs.) Si le verrouillage est accepté, alors le client peut modifier librement le document et l'enregistrer.

3.2 Cas 1

Le but du cas 1 est de faire en sorte que l'exclusion mutuelle soit gérée entre les clients, et non plus par le serveur. Elle se fait toujours document par document, et c'est bien le serveur qui gère la sauvegarde des documents et leur récupération.

3.2.1 Algorithme choisi

L'une des spécifications de l'énoncé, pour ce projet, était de réduire au maximum le nombre de messages émis par les différents clients. Dans ce but, nous avons donc opté pour un algorithme de Naimi-Tréhel pour gérer l'exclusion mutuelle entre les clients. On rappelle brièvement ci-dessous le principe de l'algorithme de Naimi-Tréhel, et la manière dont nous l'avons adapté dans le cadre de ce projet :

Le principe est de construire un arbre constitué des clients utilisant *Enshare*. Le pouvoir d'accès à la ressource critique (le bloc-note), est représenté par un jeton. Lorsqu'un client veut le jeton, il en fait la demande à son père dans l'arbre, qui lui-même fait remonter la demande, jusqu'à trouver le client qui a le jeton. Si ce dernier est en train de modifier le bloc-note, alors il refuse de donner le jeton et la structure arborescente n'est pas modifiée. En revanche, si le jeton n'est pas utilisé, alors le processus demandeur

prend le jeton, et devient la racine de l'arbre. Il devient, au passage, père de tous ses ancêtres dans l'arbre. Chaque client connaît son père et ses fils. Chaque client est identifé par son URL. Dans nos représentations schématiques, les clients seront représentés par des nombres. Les comparaisons entre clients se font selon l'ordre lexicographique, entre chaînes de caractères constituant l'URL. Le schéma ci-dessous reprend de façon synthétique ces explications.

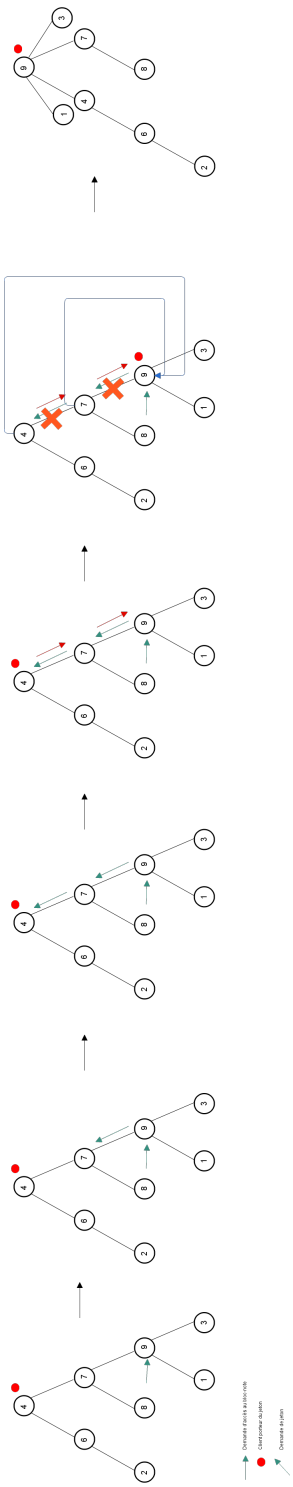

FIGURE 1 – Exclusion mutuelle avec Naimi-Tréhel

3.2.2 Cas particuliers

Déconnexion d'un client

Lorsqu'un client se déconnecte, un problème se manifeste alors : celui du maintien de la structure arborescente de Naimi-Tréhel. En effet, si l'on reprend l'exemple du schéma précédent, si le noeud 7 se déconnecte, ses descendants doivent continuer à figurer dans l'arbre, pour assurer la bonne transmission des demandes de jeton.

Nous avons ainsi mis en oeuvre la solution suivante lors de la déconnexion :

— le client qui souhaite se déconnecter procède à l'élection d'un leader parmi ses fils directs. Le critère de sélection est le suivant : est leader le client qui possède le nom le plus petit.
— le client se déconnectant annonce le leader à son père et à ses fils.
— les processus avertis se lient au leader de la façon suivante : le père du client se déconnectant devient le père du leader, et les fils du client se déconnectant deviennent les fils du leader. Les liens avec le client se déconnectant sont ainsi rompus.
— le client se déconnectant peut ainsi quitter l'application en toute sécurité pour la structure arborescente.

Le schéma suivant reprend de façon synthétique ces explications :

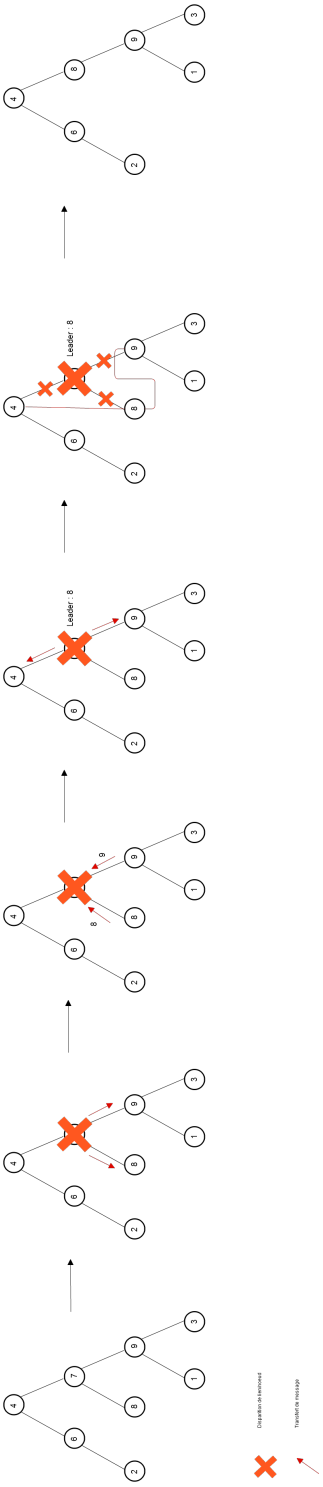

FIGURE 2 – Déconnexion avec Naimi-Tréhel

Déconnexion brutale d'un client

Dans le cas où un client se déconnecte brutalement et n'effectue pas les étapes décrites ci-dessus, il peut y avoir un problème au niveau du maintien de la structure arborescente. En effet, les fils d'un client qui se déconnecte brutalement se retrouveraient alors totalement séparés de l'arbre, sans aucune possibilité de communiquer avec les clients qui y sont toujours. Il s'agit là du défaut majeur de notre algorithme. Nous avons pensé à une solution passant par le serveur : lorsqu'un noeud détecte qu'il est orphelin, c'est-à-dire que son client père ne lui envoie plus les messages attendus, alors il en notifie le serveur, et lui demande de le raccorder à un des clients qui est bien connecté. Le serveur rattache alors le noeud orphelin à un autre client et ce noeud peut exprimer sa requête de nouveau. Nous n'avons cependant pas eu le temps d'implémenter cette solution à une déconnexion brutale.

3.3 Cas 2

Le but du cas 2 est de garder les conditions du cas 1, et en plus de rajouter une exclusion mutuelle qui se fait non plus document par document, mais ligne par ligne. C'est toujours le serveur qui contient les documents et qui gère la sauvegarde et la récupération.

3.3.1 Algorithme choisi

L'algorithme de Naimi-Tréhel ne convenait plus : il aurait fallu un arbre par ligne, ce qui alourdissait énormément le programme. Nous avons choisi l'algorithme du jeton dans un anneau, modifié pour notre cas. En effet il faut, non pas un jeton simple qui désigne le client en section critique, mais un jeton qui est un vecteur contenant des booléens correspondant à chaque ligne du document vérifiant si elle sont verrouillées lorsque le client demmande à rentrer en section critique. Ainsi, le jeton ne reste pas chez un client, mais navigue sans cesse de demandeur en demandeur renseignant chaque client. Le déverrouillage marche comme pour le verrouillage sauf qu'il n'y a pas de possibilité de refus.

Afin de gérer les cas particuliers, nous avions décidé que chaque utilisateur connaîtrait son client suivant ainsi que son client précédent. Cela a pour but de faciliter la reconstitution de la chaîne lors de la déconnexion d'un client. Dans ce cas, le client se déconnectant prévient le client suivant pour qu'il se rattache au client précedent. Lors d'une déconnexion brutale, on peut rétablir la chaine en la parcourant dans l'autre sens afin de faire correspondre le client suivant et le client précédent de celui qui ne répond plus. Cette solution de marche pas si deux clients sont déconnectés brutalement en même temps.

Voici un schéma montrant le principe énoncé précédemment :

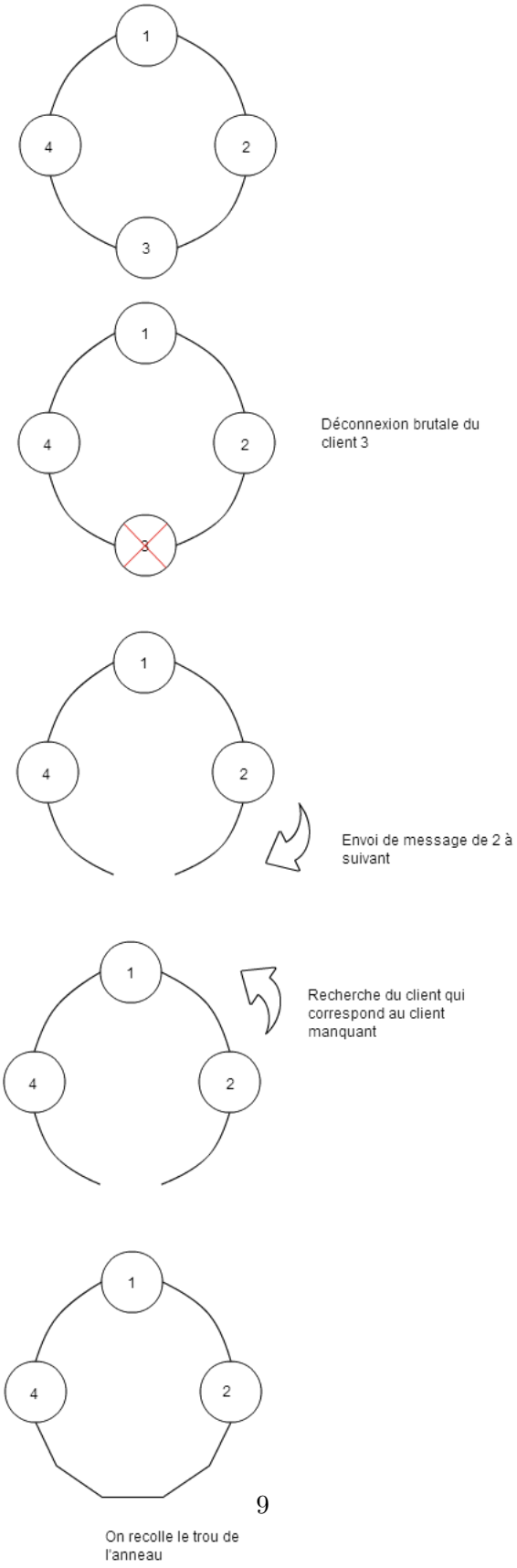

FIGURE 3 – Exclusion mutuelle avec un algorithme adapté de jeton circulant dans un anneau

3.4 Cas 3

Le but du cas 3 est de garder les conditions de cas 2 tout en enlevant complétement le serveur. C'est donc les client qui gère la sauvegarde ou la récupération.

3.4.1 Algorithme choisi

A l'instar du cas 2 l'algorithme du jeton dans un anneau peut-être appliquer si on le modifie légèrement. Dans le cas présent nus pouvons penser à un anneau qui ne contiendrait pas tout les client utilisant un même documents mais tout les clients utilisant le programme. Dans ce cas si le jeton contiendrait les documents utilisable, et donc si un nouveau client ce connecte il pourrait avoir accès à tout les documents présent dans le jeton. Pour rajouter des documents dans le jeton lors de la connection le client mettrait à disposition tout les documents qu'il possède en lien avec le programme et le jeton les copirait si il ne sont pas déjà présent. La mise à jour ce fait lorsqu'un client demmande la sauvegarde du document. Lorsqu'un client ce connecte à un document il reçoit une notification des modifications faite hors sauvegarde pour qu'il puisse avoir un document à jour. La gestions des cas particulier ce fait de la même manière que précédemment et on recontre donc les même problème.

4 Présentation technique

4.1 Analyse et CSP

Nous considérons un client et un serveur de documents. Comme illustrés sur la figure ci-dessous, voici les événements possibles (communications et changements d'état) suivants :

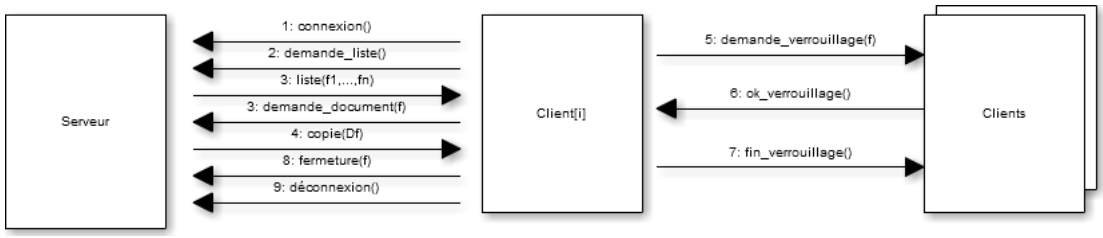

FIGURE 4 – Diagramme de Communication du Cas 1

A partir de ce diagramme de communication et de l'algorithme de Naimi-Tréhel, nous en déduisons un code CSP pour le *Serveur* et le *Client*.

Serveur : :

$f_1, ..., f_N$ *: documents*

Client[i] ? connexion() ;

Client[i] !! ok_connexion() ;

Client[i] ? demande_liste() ;

Client[i] ! liste($f_1, ..., f_N$) ;
Client[i] ? demande_document(f_j) ;
Client[i] ! copie(D_j) ;
Client[i] ? enregistrement(D_j) ;
f_j := D_j ;
Client[i] ? fermeture(f_j) ;
Client[i] ? deconnexion() ;

Voici le code CSP détaillé du client :

Client[1,..,N] : :
dernier : entier ; – c'est le client père qui est dans l'arbre
fils : liste d'entier ; – ce sont les client fils qui sont dans l'arbre
demande : booleen := faux ;
jeton : booleen ;
D_j : document ;
Serveur ! connexion() ;
Serveur ? ok_connexion() ;
- - Initialisation de dernier et jeton
[*i = 1 − >*
dernier :=0 ;
jeton :=vrai ;
[] *i != 1 − >*
dernier :=1 ;
Client[1] ! ajouter(fils,i) ;
jeton :=faux ;
] ;
Serveur ! demande_liste() ;
Serveur ? liste($f_1, ..., f_N$) ;
Serveur ! demande_document(f_j) ;
demande := vrai ;
Serveur ! copie(D_j) ;
- - Demande d'entrée en section critique
*[*!demande ; A[i] ? besoin_SC() − >*
[*dernier !=0 − >*
Client[dernier] ! dem_SC(i) ;
dernier :=0 ;
[]*dernier = 0 ; jeton − >*
A[i] ! dem_SC() ;
[]*dernier = 0 ; !jeton − >*
skip
] ;
demande := vrai
- - Fin de section critique

```
[] demande ; jeton ; A[i] ? fin_SC() − >
    demande := faux
- - Reception d'une demande sans être racine
[] dernier != 0 ; Client[k] ? dem_SC(j) − >
    Client[dernier] ! dem_SC(j) ;
    dernier := j ;
[] dernier = 0 ; Client[k] ? dem_SC(j) − >
    [ !demande − >
        Client[j] ! jeton() ;
        jeton := faux ;
    [] demande − >
        skip
    ] ;
    dernier := j ;
- - Reception du Jeton
[] Client[k] ? jeton() − >
    jeton := vrai ;
    A[i] ! debut_SC()
] ;
Serveur ! enregistrement(D_j) ;
Serveur ! fermeture(f_j) ;
Client[1,..,N] ! ! election_leader(dernier,fils) ;
Serveur ! deconnexion() ;
```

4.2 Choix d'implémentation

Pour implémenter le cas 1, nous avons repris le code d'origine *Enshare* qui utilise RMI, permettant d'appeler des méthodes distantes d'une classe (ici *Client* et *Serveur*). Nous avons remplacé la classe *CentralizedClientController* par *DistributedClientController*, ajouté quelques variables pour cet algotihme et nous avons modifié essentiellement les méthodes suivantes :

- *child* : il s'agit d'un *ArrayList* contenant tous les fils du client. Cette variable est nécessaire pour l'élection d'un leader.
- *demand* : il s'agit d'un booléen permettant de savoir si le client a fait une demande.
- *lastURL* : c'est l'URL du père du client dans l'arbre.
- *locked* : il s'agit d'un booléen permettant de savoir si le client a effectué un verrouillage.
- *token* : il s'agit d'un booléen permettant de savoir si le client possède le jeton.
- *opendocument(String fileName)*
  Cette méthode permet d'ouvrir un document. Nous avons modifié cette fonction pour qu'elle ajoute le client à l'arbre correspondant au fichier qu'il vient d'ouvrir. S'il est le seul client sur cette arbre, alors il est placé à la racine et possède le jeton. Dans le cas contraire, il devient fils du premier client connecté à l'arbre.

— *trylock()*
Cette méthode verrouille le document courant. Nous avons modifié d'une part le fait que le verouillage se fait dans le *Client*et non dans le *Serveur*.D'autre part nous utilisons l'algorithme de Naimi-Tréhel , donc cette méthode permet au client de faire une demande d'entrée en section critique. Pour cela, nous utilisons une fonction annexe *demandToken*, qui demande le jeton à la racine de l'arbre, et si celui-ci n'a pas déjà verrouillé le fichier, alors la demande est acceptée et il reçoit le jeton. Sinon, la demande d'entrée en section critique échoue.
— *closeDocument()*
Cette méthode permet au client de fermer le document courant. Celui-ci est retiré de l'arbre et nous utilisons une fonction annexe *electionNextTokenClient* qui va élire un nouveau leader dans le but de reconstruire l'arbre.

4.3 Outils utilisés et normes respectées

Tout d'abord, nous avons utilisé l'environnement de développement Eclipse sous Linux. Il nous a permis de développer notre application de façon simplifiée, notamment concernant la gestion des erreurs. De plus, Eclipse nous permet de gérer la totalité du projet en explorant les répertoires et fichiers appropriés au développement.

Afin de partager le code, nous avons choisi de créer un répertoire sur github, pour de permettre un développement collaboratif simplifié. De plus, ce système nous permet également d'avoir un gestionnaire de versions, et de pouvoir retrouver notre code depuis n'importe quelle machine. Enfn, le système de commit et de merge nous permettait une mise à jour régulière et commune du code de l'application.

Finalement, nous avons utilisé la syntaxe Doxygen afin de documenter notre code, ce qui nous a permis de le structurer au fur et à mesure, ainsi que d'avoir une documentation nous permettant de reprendre le développement avec un code commenté de façon rigoureuse, ce qui nous a fait gagner du temps.

4.4 Tests

Tous nos tests ont été effectué avec l'interface graphique car cela nous permettait d'avoir un résultat clair, visible et rapide. Nous avons testé de manière progressive, *i.e.* au fur et à mesure que nous implémentions nos services :

1. ouverture d'un document
2. verrouillage d'un document
3. déverrouillage d'un document
4. sauvegarde et notification d'un document aux clients
5. fermeture d'un document

Nous avons également pris en compte plusieurs scénarios possibles :

- un seul client se connectant au serveur
- deux clients se connectant au serveur
- verrouillage d'un document déjà verrouillé par un autre client
- déconnexion d'un client
- *etc...*

Pour la déconnexion brutale d'un client, nous avons dû utiliser quelques lignes de commandes pour effectuer des tests. Toutefois, nous avons rencontré un problème à la réouverture d'un document préalablement verrouillé, sauvegardé et fermé par un client. Le *GUI* affiche alors un document vide, bien que celui-ci soit non vide en réalité.

5 Bilan du projet

Tout d'abord, nous avons eu énormément de mal à comprendre ce que nous devions modifier et comment le faire, ce qui nous à fait perdre énormément de temps. Ainsi, même si le côté théorique n'a pas pris beaucoup de temps ensuite, le perte de temps engendrée par ce manque de compréhension a été trop important pour que l'on puisse finir la totalité du sujet, c'est-à-dire finir le cas 3.

Du côté technique, le contrat a été rempli pour le cas 1, mis à part quelques exeptions, comme le cas de la déconnexion brutale, ou de la réouverture immédiate d'un fichier tout juste fermé que nous n'avons pas pu régler faute de temps.

Cependant, si l'on devait recommencer un sujet similaire avec les connaissances actuelles, notre projet serait réalisé de manière bien plus efficace, car ce travail nous a permis de mieux comprendre l'algorithmiquee distribuée.

Il s'agissait aussi du premier projet relativement conséquent où nous avions à reprendre un code déjà existant pour le modifier, ce qui fut

6 Conclusion

Ce projet nous a permis de mettre en pratique les différents algorithmes d'exclusion mutuelle vus en cours. Lors de sa réalisation, nous avons été conscients de l'importance de la phase d'analyse CSP, qui a été notre base de réflexion. Toutefois, nous n'avons pas réussi à intégrer rapidement le code du logiciel *Enshare* fourni, ce qui a limité l'implémentation des autres cas. Bien que nous ayions rendu un code satisfaisant uniquement le cas 1, ce projet nous a permis de cerner nos propres difficultés.