INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

SYSTEME DE CLAVARDAGE DISTRIBUE INTERACTIF MULTI-UTILISATEUR TEMPS REEL

ALFREDO Esmeralda

ENIREVEC Corentin

1-Introduction

Dans le cadre d'un projet de Programmation Orienté Objet, nous avons dû créer une application de clavardage distribué interactif et multi-utilisateur, en respectant certaines consignes.

L'objectif du projet était de réaliser un système de communication qui sert de support aux équipes et groupes d'une entreprise afin d'accroître leur efficacité. La création de cette application nécessite la connaissance de différentes phases de conception d'un système et la maîtrise du langage orienté objet.

Dans ce rapport, nous décrirons le processus de conception de l'application ainsi que son évolution, certains détails sur le choix des différentes technologies.

2- Cahier de Charges

Le système de clavardage va permettre la communication entre les utilisateurs par l'échange de message textuel, de type image ou un autre type de fichier.

Si les utilisateurs se connecte de l'intérieur de l'entreprise le système a un comportement distribué ce que veut dire que l'ensemble des ressources du système ne se retrouve sur la même machine et dans le cas où l'utilisateur se connecte d'extérieur le système repose sur l'architecture 3-tiers (Modèle - Vue - Controlleur).

Les principales exigences fonctionnelles sont :

- Authentification et gestion des login
- Découverte des utilisateurs de l'intérieur avec le système décentralisé ;
- Découverte des utilisateurs de l'extérieur avec le système centralisé ;
- Envoi/Réception des messages entre utilisateurs ;
- Gestion de l'historique des anciens messages.

D'ailleurs, il existe aussi d'autre exigences :

- Opérationnel : performance, ressources, et sûreté de fonctionnement
- Développement : management et gestion du projet, ergonomie et qualification

Le cahier de charge est bien spécifié dans le lien suivant :

https://moodle.insa-toulouse.fr/pluginfile.php/127921/mod_resource/content/1/INSA_COO_POO_URD_v3.1.pdf

3- Analyse

Après une analyse détaillée du besoin et les distinctes exigences présenter dans le cahier de charge nous avons effectué plusieurs choix afin de répondre aux attentes préciser dans le projet.

3.1-Authentification et gestion des logins

Pour permettre les utilisateurs de s'authentifier et utiliser un login ou pseudo unique nous avons créé un objet utilisateur qui est identifiée par son pseudo mais aussi par un ID unique que ne change pas. Au moment de la connexion un utilisateur au système il envoie un broadcast en UDP et tous les utilisateurs actives dans cet instant vont répondre sa requête en envoyant un PDU qui contient sont pseudo et son id qu'il va ajouter dans sa liste d'utilisateur actives. En disposant de la liste d'utilisateur actives, si un utilisateur x veut changer de pseudo, il a juste à tester si le nouveau pseudo qu'il veut adopter ne se trouve pas dans sa liste d'utilisateur active.

Dans le cas où l'utilisateur ne se trouve pas sur le réseau privé de l'entreprise il va récupérer la liste des utilisateurs active à partir un serveur qui appartient au réseau de l'entreprise.

3.2- Envoi/Réception des messages entre utilisateurs

Notre système permet l'envois et la réception des tous le types d'objet et cette émission et réception se fait par moyen des threads des qui sont responsable de gérer cette fonction. L'utilisation de threads permet la gestion de plusieurs communications au même moment. Comme le système est distribué nous avons choisi de faire la connexion entre deux utilisateurs en deux temps :

Tout d'abord, l'utilisateur x que veux discuter avec un utilisateur y, l'envois un PDU UDP à l'utilisateur y avec la demande de connexion pour une discussion. L'utilisateur y répond avec un PDU UDP avec le numéro de port qui va être utiliser pour la connexion en TCP, sur lequel l'utilisateur y va écouter en tant que serveur TCP.

Ensuite l'utilisateur x commence une connexion TCP au port qu'il a reçu de y comme client TCP par moyen d'un nouveau thread. Après la phase d'établissement de connexion les deux entités peuvent communiquer.

3.3- Gestion de l'historique des anciens messages

Etant donnée que notre système est décentralisé, pour le stockage de l'historique des messages échangés nous avons choisi d'utiliser le SQLite qui sera intégré directement dans notre application localement. De cette façon nous avons un le temps d'accès aux informations stocké moindre.

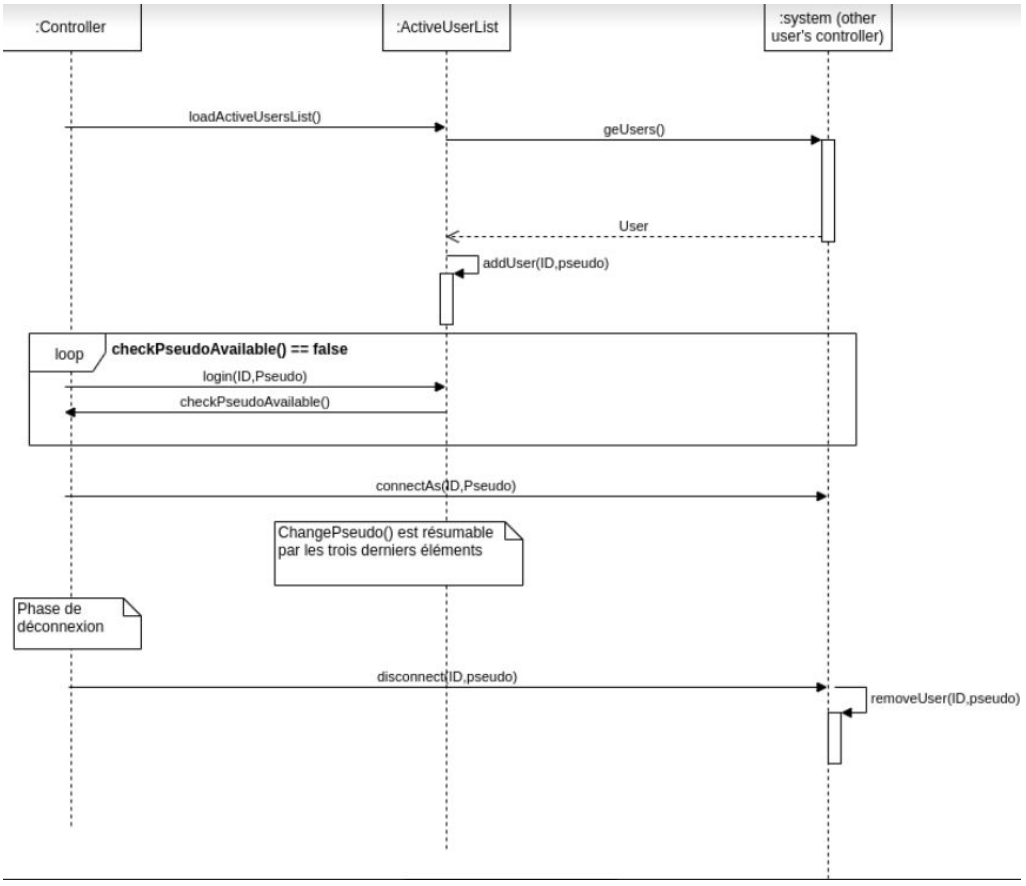
4- Conception

Après la prise de décision des différents choix qui nous semble pertinent et après plusieurs itérations de nos modèles, nous sommes arrivées à de diagrammes qui représente notre système.

4.1-Diagramme de Cas d'Utilisation

4.2-Diagramme de Séquence

Changement de pseudo



Connexion / Déconnexion

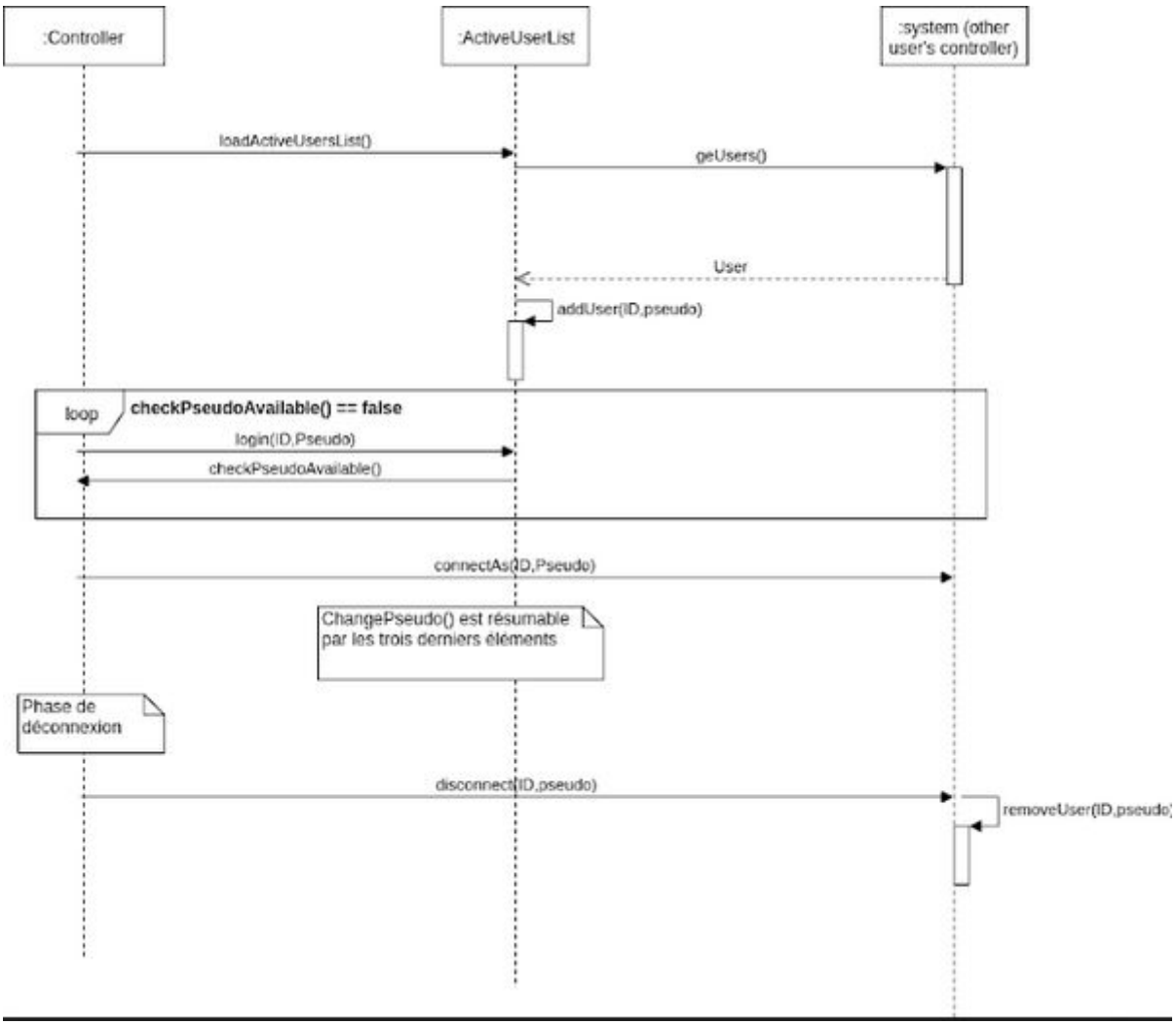
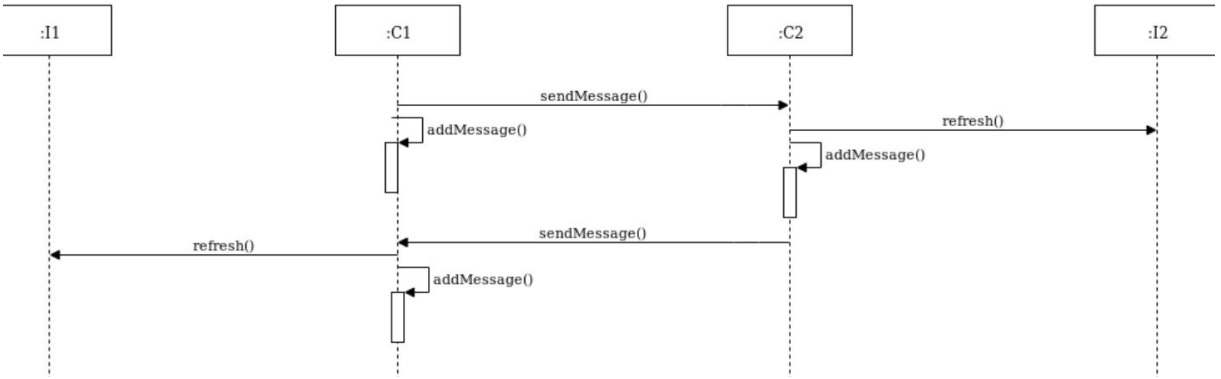


Diagramme de sequence pour la Comunication

Pour ce cas nous considérons que la session de connexion est déjà faite et que l'user1 dispose de sa liste de users actives et l'historique avec un autre user2 avec le quel il veut discuter



4.3-Diagramme de Classe

5- Implémentation

Le langage orienté objet que nous avons utilisé c'est le java. Nous avons créé un projet java EE en utilisant l'IDE NetBeans. Notre projet est constitué par différentes packages et classes que permettent d'avoir une bonne organisation du code.

5.1 - Idées de conception

Message UDP

Chaque agent dispose d'un serveur prêt à recevoir des messages UDP (voir tableau ci-après) sur le port 1025. Chaque message est récupérer et traiter par un thread en charge de son exploitation.

Message UDP émis	Description
connected	Notifier la connexion d'un nouvel utilisateur
connecting	Utilisateur en cours de connexion, l'objectif de ce message est de recevoir les informations
createChatServer	Demander à un des client l'ouverture d'une connexion
endChating	Terminer un connexion en cours
disconnect	Notifier la déconnexion d'un utilisateur
changePseudo	Notifier le changement de pseudo
serverDisconnect	Notifier la déconnexion d'un serveur

Idées pour le design

Pour le design nous avons cherché à avoir une interface propre, épurée , cohérente avec cette de l'OS et avons cherché à la simplifier au maximum.

Les éléments les moins fréquemment utilisés de l'écran de messagerie sont ainsi placés dans le menu option afin d'alléger l'interface.

5.2 - Fonctionnement des communications

Pour le mode distant (hors réseau local) comme le mode local (dans le réseau local), la communication s'effectue par l'envoi préalable d'un paquet createChatServeur pour enclencher la création du Serveur TCP auquel l'initiateur se connectera pour l'envoi de messages

Un serveur à aussi été développé pour permettre les utilisations disantes de l'application, il est en charge du relayage des messages.

5.3 Fonctions intéressantes pour la stabilité et le confort d'utilisation

- L'utilisateur est informé si le serveur n'est plus accessible et est invité à quitter l'application
- Les paramètres renseignés sont sauvegardés afin de faire gagner du temps pour les connexions ultérieures
- Le nom de l'utilisateur n'est pas affiché en cas d'envoi de messages successifs
- L'utilisateur est informé et ne peut utiliser l'application si aucune adresse n'est définie sur l'interface proposée

5.4 -Fonctions qui n'ont pas pu être implémentées

Des messages UDP propre au serveur

Il était prévu la possibilité d'opérer avec plusieurs serveurs qui pourrait permettre d'avoir de la redondance. Un serveur maître serait élu et serait le serveur disposant de l'historique des messages reçus à la fois des utilisateurs locaux et distants.

Message UDP émis	Description
challengeServer	Lancer un tirage au sort permettant d'établir le serveur maître
newServer	Notifier l'entrée d'un nouveau serveur
defaultServer	Notifier le serveur maître

Message avec fichiers et Images

À cause d'une contrainte de temps, nous n'avons pas été en mesure d'implémenter la fonction de message non textuel. Cependant, il avait été pensé , l'ajout d'un attribut permettant de savoir sous quelle forme doit être envoyé le message reçu. Tous ces messages seront récupérés avec la fonction Message.getContent();

Message.type	Texte	Image	Fichier
Description	Message textuel	Message avec image	Fichier de divers type
Ce qui serait fait	Afficher le texte (implémenté)	Télécharger l'image Afficher l'image	Télécharger le fichier

Stockage sur base de données

L'idée ici était de permettre le stockage de tous les messages émis en base de données, les messages seraient envoyés au serveur lors de la déconnexion.

6-Guide d'utilisation

Comment lancer ClavardApp

Pour démarrer ClavardApp en mode client, il faut utiliser la commande **java Controller.Application nomDeL'Interface**

avec nomDeL'interface l'interface sur le réseau local où est déployé le système ClavardApp

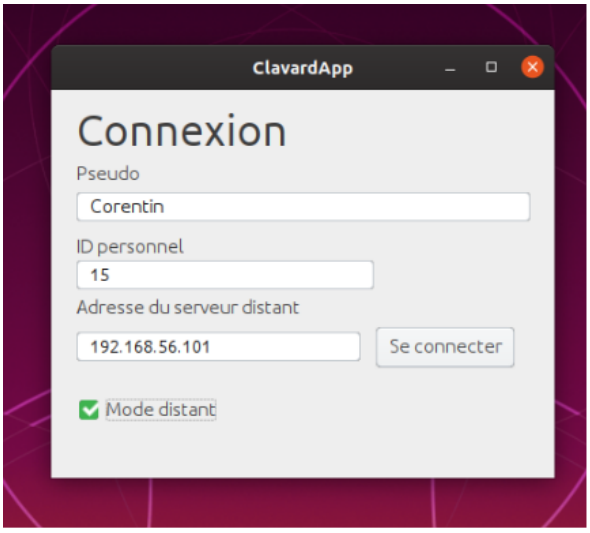
(exemple java Controller.Application eth0)

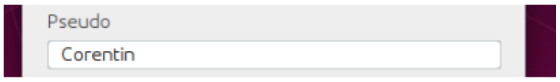
Pour démarrer ClavardApp en mode client, il faut utiliser la commande **java Server.Server nomDeL'Interface** , dans ce mode aucune fenêtre ne s'affichera mais vous pourrez voir les logs de l'application.

avec nomDeL'interface l'interface sur le réseau local où est déployé le système ClavardApp

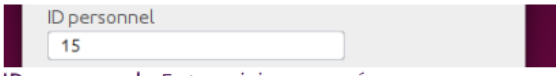
(exemple java Server.Server eth0)

Écran de connexion



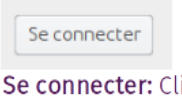


Pseudo : Entrez-y le pseudo souhaité le temps de cette session, il ne doit pas être déjà utilisé sur le réseau ClavardApp.
Il vous sera indiqué l'utilisation par autre utilisateur via un popup en bas de fenêtre



ID personnel : Entrez-ici un numéro d'identifiant, il vous identifiera en tant qu'utilisateur unique sur tout le réseau

Adresse du serveur distant :
Entrez ici l'adresse IP ou nom de domaine du serveur auquel vous voulez vous connecter en mode distant (uniquement disponible si mode distant est coché)

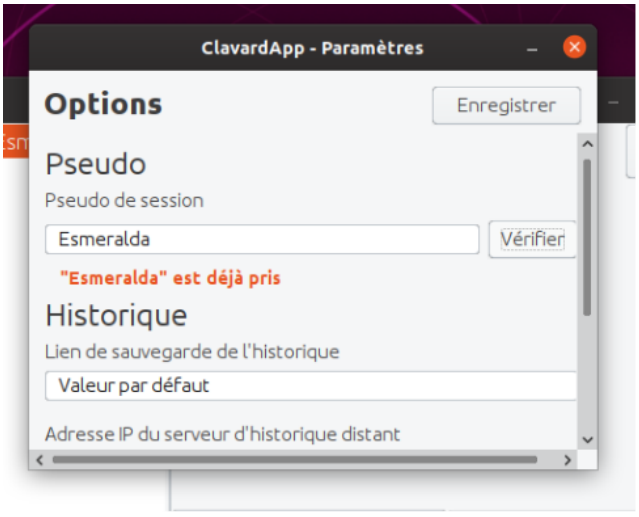


Se connecter: Cliquez ensuite sur « Se connecter » pour rejoindre le réseau

Écran de messagerie

1 : Liste des utilisateurs connectés
La liste des utilisateurs connectés, cliquez sur l'un des utilisateurs pour pouvoir entamer un discussion avec lui ou voir les messages qui vous sont destinés
2 : Panneau messages
Les messages reçus et émis à votre interlocuteur
(l'aspect peut changer selon la version)
3 : Rédaction de messages
La zone où vous pouvez entrer votre messages, cliquez sur « Envoyer » après rédaction du message
4 : Menu Option
Accédez aux paramètres réglables de l'application, (nom de pseudo, IP serveur distant, etc.)

Écran d'options



Voici le menu Option.
Toutes les options sont réglables à l'aide des champs proposés, le données ne sont prise en comptes que si clique à la fin sur « Enregistrer »
Pseudo : Changer de pseudo durant la session

Lien de sauvegarde de l'historique:
Chemin vers le fichier contenant les messages précédents
Adresse IP serveur d'historique :
Adresse du serveur en charge de l'enregistrement des messages

Adresse IP de serveur distant
Adresse du serveur sur lequel on se connectera pour accéder au réseau en mode distant

7- Conclusion

Ce projet a avant tout été l'occasion d'expérimenter, de découvrir, de comprendre un cahier des charges afin de savoir quoi faire et comment faire. Sur ce projet nous devions concevoir à partir d'un

cahier des charges, les objets, les communications, l'interface et le contrôle des différentes fonctionnalités du projet. Pour ce faire nous avons réalisé une application basée sur une architecture 3-tiers. Nous avons dû nous organiser, et trouver comment expliquer pour permettre de poursuivre et aussi de ne pas refaire les tâches déjà faites.

Nous avons à travers ce projet essayé de récupérer le moins possible de code déjà fait d'Internet afin de plus réfléchir au fonctionnement et mieux comprendre le fonctionnement et donc ce que l'on peut et ce que l'on ne peut pas faire avec les différents objets Java, mais aussi comment les modifier pour les adaptés le plus possible à nos besoins. Ça a été l'occasion de se familiariser avec Swing et la conception d'interface graphique en Java. Une des parties sur lesquels nous avons passé beaucoup de temps est la conception du mode de fonctionnement du client distant afin qu'il puisse réutiliser tous les objets et exclusivement les objets du client local. Par la suite, pour la conception de la fonction de relayage, nous avons fait face à beaucoup de bugs et découvert de nouveaux besoins liés à la création du mode serveur.

Ce que l'on ferait mieux ou pour accélérer le développement

- Assurer une meilleure répartition des tâches
- Plus se tenir au courant de ce qui a été fait et de comment cela fonctionne par l'intermédiaire de débrieffing en début de séances
- Mieux documenter la liste des tâches à faire

Certaines parties d'implémentations n'ont pas pu être faites, mais il est sûr qu'avec les compétences acquises en cette fin de projet, nous aurions été en mesure de le terminer et de le peaufiner.