# ELC A-2 Application Concurrente Mobile

## Projet de messagerie instantanée Répartie

AMADEI Philippe

COLLIER Antoine

PERROTTE Camille

ZHANG Linchao


ÉCOLE CENTRALELYON

# Contenu



# Table des figures

ÉCOLE CENTRALELYON

# Introduction

Les cours d'électif sur le langage de programmation Java nous ont permis d'apprendre à réaliser des applications graphiques, mais également d'apprendre à créer des applications réparties. Afin de mettre ces deux aspects en commun, nous avons donc pour projet de réaliser une application graphique répartie de type messagerie instantanée.

# Cahier des charges

Nous souhaitons réaliser une application de messagerie instantanée permettant à des utilisateurs de chatter entre eux. Les modèles pour ce type d'application sont Skype, MSN Messenger, mais aussi iMessage. Nous tenterons de réaliser une application basique s'inspirant de ces différents logiciels.

Le cahier des charges minimal adéquat pour une application de messagerie est donc le suivant :

- Possibilité de choisir son pseudo
- Possibilité de choisir le destinataire d'un message
- Possibilité de voir les personnes connectées

Pour aller plus loin dans la création d'une application java, nous nous recherchons à faire une application plus évoluée. Notre cahier des charges comprend donc également :

- Base de données côté serveur répertoriant tous les utilisateurs enregistrés, leurs mots de passe (en clair pour simplifier), leurs contacts
- Possibilité de rajouter des contacts
- Possibilité de supprimer des contacts
- Seules deux personnes étant contact peuvent se voir en ligne

Et bien entendu afin de rester dans le cadre de ce cours :

- L'application sera répartie : le client et le serveur seront deux applications différentes. Le serveur sera le centre névralgique du système de messagerie.

Nous chercherons également à avoir une interface claire et épurée. Des alertes permettront à l'utilisateur de comprendre comment l'interface fonctionne.

# Notre Application

Nous avons commencé par réfléchir au design que prendrait notre application. Cette étape nous semblait importante pour pouvoir scinder le travail dans le groupe avec une partie interface utilisateur et une partie code effectif, avec à la fin un lien à faire entre les boutons de l'interface et les fonctions développées.

En nous inspirant des logiciels cités précédemment nous sommes arrivés à ce concept :

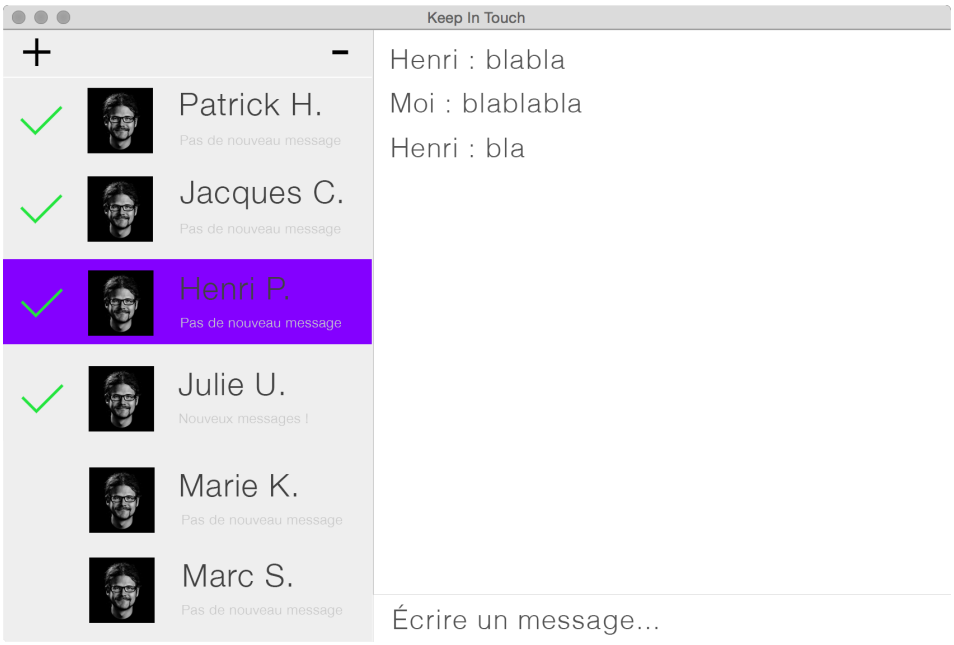


**Figure 1 : Concept d'application**

Nous avons cependant conscience au moment de la réalisation de concept que nous n'aurons probablement pas le temps tout inclure dans le projet rendu.

Nous avons donc travaillé sur les différents aspects du cahier des charges et avons rapidement constaté que des problèmes pouvaient survenir rapidement. Nous en détaillons certains dans la partie « Problèmes rencontrés ».

La réalisation de l'interface graphique a montré qu'avec les éléments de base de Swing il est difficile de faire des interfaces épurées (présence de bords sur tous les éléments notamment). Une solution est donc de customiser l'interface, ce que nous n'avons pas fait pour nous concentrer sur l'essentiel. Voici donc l'interface principale de notre application :

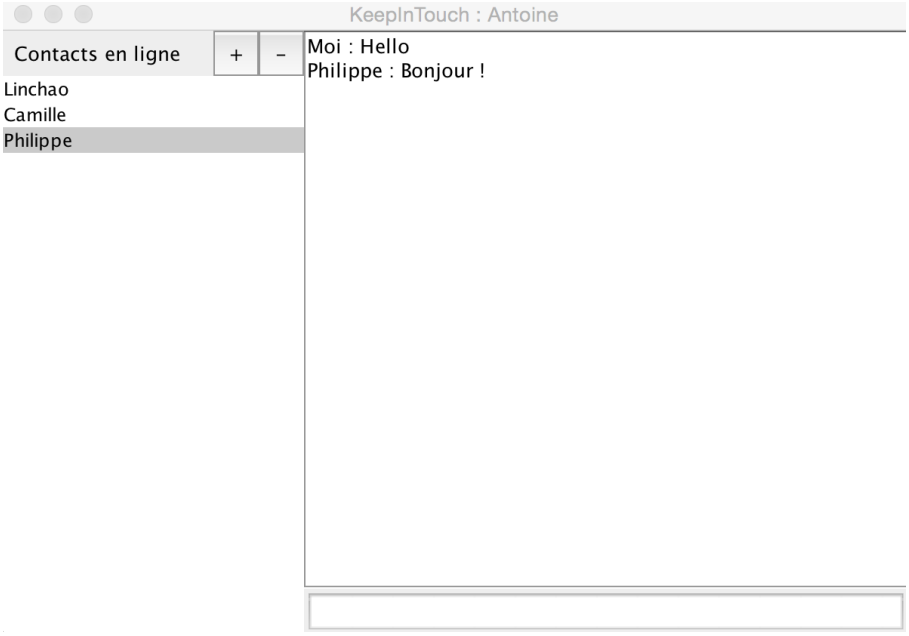


**Figure 2 : Interface de notre application**

Nous avons donc réussi à recréer globalement l'allure de notre concept.

## Installer notre application (développeur)

Rajouter les fichiers des projets « Client » et « Serveur » sous Eclipse. Nous n'avons pas créé de fichiers .jar car nous n'arrivions pas à inclure la base de données du serveur.

Afin de lancer notre application client, il faut préalablement lancer l'application serveur. Dans le cas contraire, un message d'erreur s'affichera sur le client comme suit :


Figure 3 : exemple d'erreur de connexion si le serveur n'a pas été lancé préalablement

Pour se connecter il suffit de donner une combinaison utilisateur-mot de passe correcte déjà présente dans la base de données ou de créer un nouveau pseudo (la connexion fait alors office d'inscription).

## Appendices de fonctionnement

Le lancement de l'application nous amène sur cette fenêtre de connexion :


Figure 4 : Fenêtre de connexion

On peut noter que la modification d'un jLabel informe l'utilisateur des informations sur l'état de la tentative de connexion.

Après une tentative de connexion réussie, la fenêtre de login disparaît au profit de l'interface de notre messagerie instantanée.

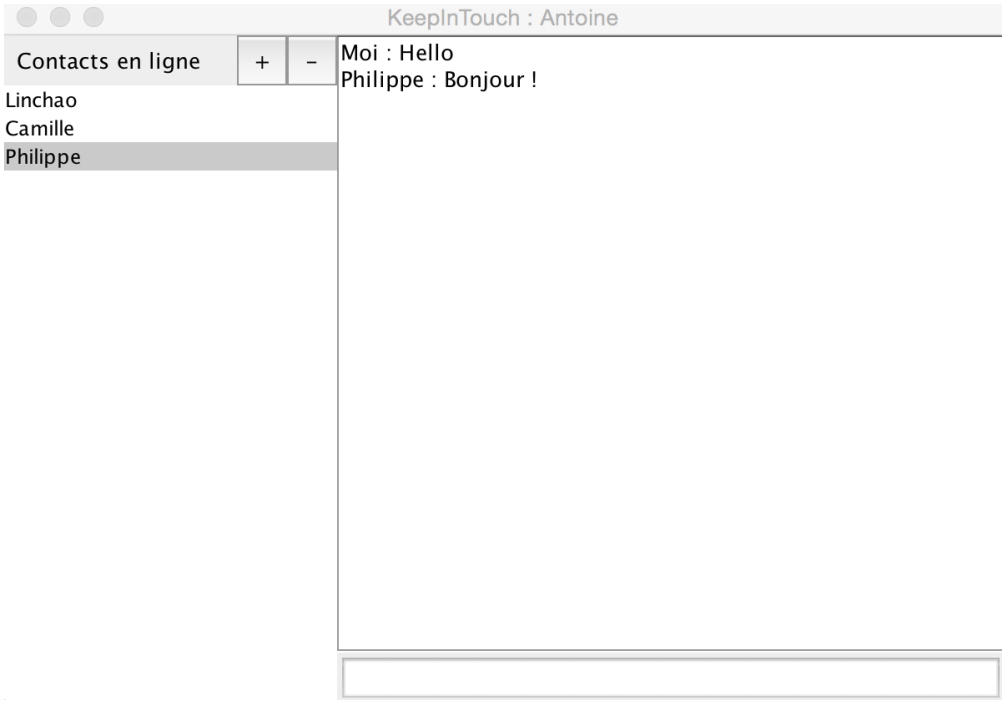


Figure 5 : L'interface principale

La liste sur le côté affiche les contacts en ligne. Dans l'exemple suivant, il y a trois contacts d'Antoine qui sont connectés.

Pour rajouter un contact, il suffit de cliquer sur le bouton « + » et de spécifier le pseudo de l'ami à rajouter


Figure 6 : Ajout d'un contact pour Philippe

Puisque Camille était en ligne, on constate que lorsqu'on l'a rajouté, elle apparaît directement dans la liste des personnes en ligne. On peut dès lors discuter avec le nouveau contact.

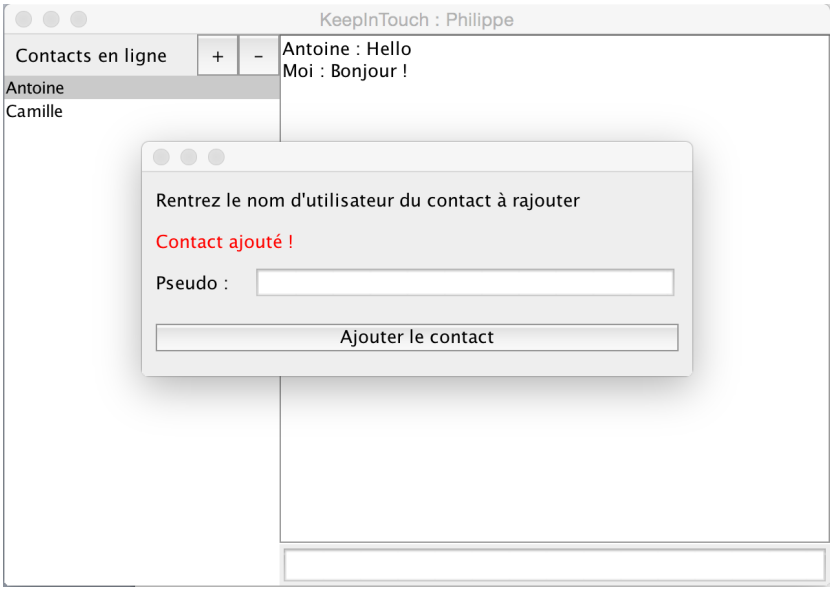


Figure 7 : Contact Camille rajouté

Pour supprimer un contact il faut que celui-ci soit en ligne. On le sélectionne, et on clique sur le bouton « - ». Une fenêtre vient alors confirmer l'utilisateur la suppression

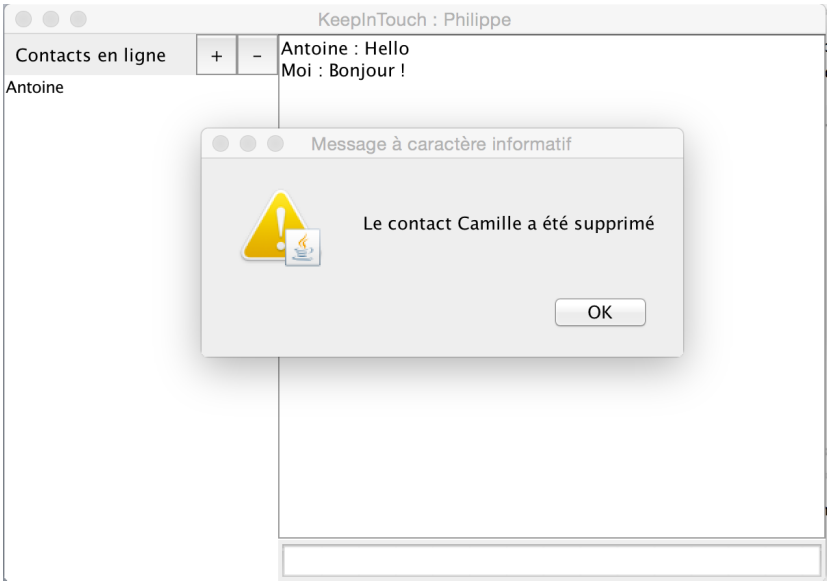


Figure 8 : Contact Camille Supprimé

Enfin, on peut remarquer que nous avions initialement prévu de permettre le changement de fenêtre de conversation en sélectionnant un correspondant dans la jList. Nous n'avons cependant pas reçu à intégrer une version totalement fonctionnelle et l'avons supprimé du livrable final. Si tant est que nous l'avions réussi, nous aurions pu débloquer les fonctionnalités suivantes qui auraient alors été pertinentes à coder :

- Stockage des conversations dans la base de données
- Affichage également des contacts déconnectés
- Possibilité d'envoyer des messages aux personnes déconnectées (qui les auront en chargeant une conversation une fois connectée)

La base de données a nécessité beaucoup de temps à la coder. Nous avons choisi, pour simplifier le traitement, d'utiliser une base de données au format XML. Mais le système de balise, bien que très visuel, n'est absolument pas simple à récupérer en Java, l'écriture devenant très rapidement lourde. Nous avons donc d'abord dû bien réfléchir à la structure à donner à notre base de données avant de nous lancer dans l'implémentation sous Java. Nous avons finalement choisi l'arborescence sous la forme suivante :

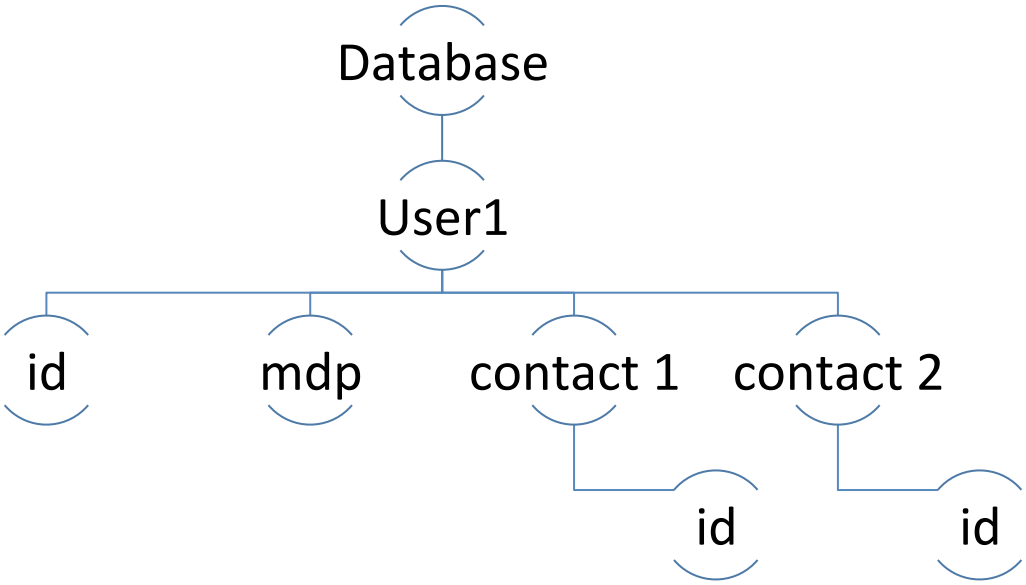


Figure 9 : Structure de la base de données XML

Cette base de données est ensuite importée au sein d'un tableau d'utilisateurs. Ces utilisateurs possèdent leurs attributs, et notamment un tableau de contacts. Ces contacts possèdent eux mêmes leurs attributs.

On a donc la possibilité de faire correspondre notre objet Java avec la base de données ! Pour résumer la situation, voici un diagramme de classe :

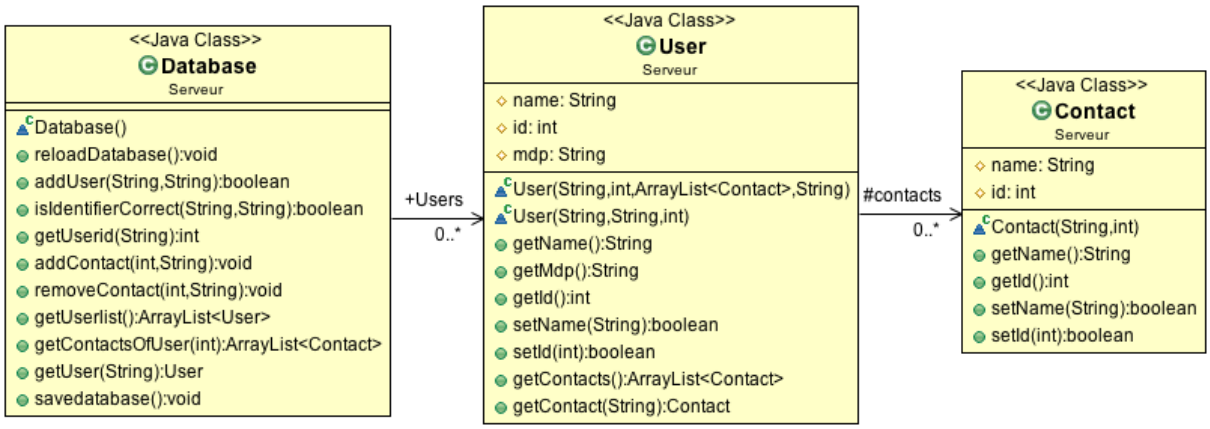


Figure 10 : Diagramme des classes de la base de données

La sauvegarde de la base de données XML et le rechargement de l'objet java correspondant se font ensuite à la volée lors d'un changement à effectuer (suppression d'un contact, ajout d'un contact, enregistrement d'un nouvel utilisateur …). On note l'utilisation de méthodes « synchronized » afin d'éviter de corrompre la base de données en faisant des changement simultanés.

## Problèmes rencontrés

1. Envoyer un message à l'utilisateur spécifié

On veut réaliser la fonction qui sert à envoyer un message à l'utilisateur qui a été sélectionné dans la liste des contacts. On enregistre le nom d'utilisateur quand on crée un <<clientObj>> à la connexion au serveur. Ensuite, on stocke les pseudos et les IClient correspondants dans deux listes séparées, mais avec des index qui se correspondent.

```
List<IClient> clients = new ArrayList<IClient>();

List<String> clientsName = new ArrayList<String>();
```

On peut alors jongler entre ces deux vecteurs pour réaliser la fonction attendue.

```
if(clientsName.get(i).equals(ToName))

{

 try

 {

  clients.get(i).MsgArrived(msg, FromUser);

 }

 catch(Exception e){…}

}
```

2. Changement de la fenêtre

On désire réaliser un changement de fenêtre de conversation lorsque l'on clique sur un utilisateur de la liste des contacts connectés. Lors d'un changement de fenpetre on veut que le contenu de la conversation soit rechargé. Il y aurait donc une fenêtre de conversation pour chaque contact en ligne.

On utilise donc le <<hashmap>> pour stocker les messages précédents. C'est ensuite aisé de les retrouver suivant l'utilisateur sélectionné.

```
private HashMap<String , List<String>> messages = new HashMap<String , List<String>>();
```

```
jList.addListSelectionListener(new ListListener());


private class ListListener implements ListSelectionListener{
        @Override
        public void valueChanged(ListSelectionEvent e) {
                // TODO Auto-generated method stub
                jTextArea1.setText("");
                String name =extracted(e).getSelectedValue().toString();
                if(messages.get(name).size() >0){
                for (int i=0; i<messages.get(name).size(); i++){
                        jTextArea1.append(messages.get(name).get(i));
                }
                }
        }
        private JList<String> extracted(ListSelectionEvent e) {
                return (JList<String>) e.getSource();
        }
}
```

Mais, il existe cependant des erreurs lorsque l'on lance l'application, jusqu'à présent, nous ne avons pas résolu ces problèmes.

## Perspectives d'évolution

Avec encore un peu de temps, et sachant que nous codons désormais plus vite, nous pourrions probablement réussir à intégrer les fonctionnalités suivantes :

- Rajouter les fonctionnalités décrites précédemment dans le cas où nous arrivons à faire un changement de fenêtre
- On pourrait améliorer la sécurité en ne stockant nul part le mot de passe en clair. On pourrait envisager de le hasher par exemple.

- Ajouter un son lors d'un nouveau message
- Ajout d'un marqueur distinctif dans la jList pour un nouveau message en provenance d'un utilisateur

Enfin, bien qu'impossible dans le temps imparti et avec nos connaissances en Java, nous pourrions imaginer rajouter une gestion plus fine de l'enregistrement d'un nouvel utilisateur (adresse mail, vérification de la difficulté du mot de passe), envoi de mails depuis un site internet pour vérifier les mails et gérer l'oubli de mot de passe, la possibilité d'utiliser des avatars, de s'envoyer des images et des fichiers dans une fenêtre de conversation etc.

## Conclusion

Bien qu'imparfaite, nous avons réussi à réaliser une application de messagerie répartie répondant à la quasi-totalité de notre cahier des charges. Elle permet de bien mettre en œuvre les procédés vus en cours et en TD, et nous a surtout permis d'apprendre réellement à coder en Java. Nous avons en effet bien réussi à nous partager les taches et chacun a ainsi pu progresser.