



UNIVERSITÉ NATIONALE DU VIETNAM

INSTITUT FRANCOPHONE INTERNATIONAL

*Promotion : 24*

*O*ption : SIM (systèmes Intélligents et multimédias)

Conception et Architecture des réseaux

---

# Messagerie instantanée interactive sur le réseau local

---

*Rédigé par :*

**MBIAYA KWUITE Franck Anael**

**KANA NGUIMFACK Kévin**

**KAMBU MBUANGI Fabien**

**OUEDRAOGO Wend-Panga Jérémie**

**NGUYEN Truong Thinh**

*Enseignant :*

**Nguyen Hong Quang**

*Année Académique 2019/2020*

# SOMMAIRE

# INTRODUCTION

Le monde d'aujourd'hui qui nous entoure, est un monde connecté. À cet effet les réseaux informatiques ne cessent d'être concevoir dans le but de partager des informations, à l'instar des réseaux de chat comme Whatsapp, Messenger. Ces derniers permettent la réalisation des échanges sur la base des protocoles bien conçus respectant des normes bien définies. Certains protocoles sont ouverts à l'instar de **XMPP** (Extensible Messaging and Presence Protocol), **IRC** (Internet Relay Chat) et **SILC** (Secure Internet Live Conferencing). Et d'autres propriétaires comme, **WLM** (Window live Messenger) et le **protocole Skype**. C'est alors dans le cadre de notre module conception et architecture des réseaux, nous sommes amener à concevoir notre propre protocole et à Implementer une messagerie instantanée sur la base de ce protocole. Pour cette tâche nous definissons un ensemble de spécifications à respecter, puis nous allons concevoir notre protocole **CAR** et implementer l'application.

CHAPITRE 1

# CAHIER DE CHARGE

Notre futur application de chat devra respecté un ensemble de spécifications. elles sont définies comme suite :

- Les postes des utilisateurs se trouvent sur un réseau local.
- Les utilisateurs sont nomades (ils ne conservent pas leur adresses IP). Ils peuvent se connecter de n'importe où.
- La gestion du système est décentralisée ( pas de serveur centtrale).
- La communication par des messages asynchrones sous de chaînes de caractères de longueur variable.
- On a un seule groupe de discution sur le réseau.
- On a la possibilité d'éffectuée des communications groupés.
- Chaque utilisateur doit s'enregistrer sur le réseau lors de sa connexion en fournissant un identifiant.
- Chaque poste doit maintenir une liste des utilisateurs connectés au système de Messagerie ( gestion dynamique des connexions/déconnexions).
- La gestion des états des utilisateurs ( connecté, absent et occupé).

CHAPITRE 2

# CONCEPTION

## 2.1 Généralités

## 2.2 Liste des commandes

Notre protocole est defini à partir d'un ensemble de régles régissant la communication dans le réseau. Ces règles sont définies par un ensemble de commandes respectant un format bien défini.

### 2.2.1 Les commandes CONNECT et NUSER

Avant d'accéder à l'application, tout utilisateur doit fournir un pseudo qui permettra de l'identifier dans le réseau d'échange et il a également la possibilité de définir son statut **1 : disponible, 2 : occupé(e), 3 : absent**. La commande **CONNECT** permet de réaliser cette connexion au système. Lorsque le client exécute la commande **CONNECT**, le client scanne le réseau local pour trouver les autres utilisateurs connectés. Et tous ces derniers répondent au client nouvellement connecté avec leurs noms d'utilisateurs, leurs statuts et leurs adresses IP. Une fois cette opération effectuée, chaque utilisateur ajoute ce nouveau utlisateur et son adresse dans sa liste des utilisateurs connectés avec un statut par défaut à **"disponible"**. Par la suite, le nouvel utilsateur connecté, ajoute l'ensemble de clients qui ont répondu à sa requête **CONNECT** dans sa liste des utilisateurs.

#### Syntaxe

Toute connexion au réseau de chat doit respecter une syntaxe bien défine :

- **CONNECT name_user** : connexion de l'utilisateur name_user.
- **CONNECT name_user user_status** : connexion de l'utilisateur name_user avec definition de son statut.
- **NUSER user_name** : état connecté du user_name.

#### Réponse serveur

Les codes réponses possibles aux commande **CONNECT** et **NUSER** sont les suivantes :

- **code 2000** : connexion réussi.

— **code 2001** : format non respecté.
— **code 8000** : utilisateur connecté.

### 2.2.2 La commande QUIT

Pour quitter l'application de chat, l'utilisateur doit saisir la commande **QUIT**. Lorsqu'il exécute la commande **QUIT**, tous les autres utilisateurs recoivent le nom et l'adresse IP de ce dernier. Chaque utilisateur le supprime de sa liste. Une fois cette opération effectuée, l'utilisateur est déconnecté du système.

#### Syntaxe

Toute déconnexion au réseau de chat doit respecter la syntaxe suivante :
**QUIT** : fermer l'application.

#### Réponse serveur

Les codes de Responses possibles à la commande **QUIT** sont les suivantes :

— **code 7000** : fermeture de l'application réussite.
— **code 7001** : échec de fermeture de l'application.

### 2.2.3 La commande USERS

L'utilisateur demande la Liste des connectés avec leur statut à l'aide de la commande **USERS** et l'application lui retourne cette liste des utilisateurs connectés.

#### Syntaxe

la demande de la liste d'utilisateurs au réseau de chat se fait de la manière suivante :

— **USERS** : demander la liste des utilisateurs.
— **USERS list_users** : liste des utilisateurs connectés.

#### Réponse serveur

Le code de Response à la commande **USERS** est le suivant :
**code 3000** : succès.

### 2.2.4 La commande STATUS

La commande **SSTATUS** retourne la liste des utilisateurs avec leur statut. Les statuts possibles sont : **1 : disponible, 2 : occupé(e), 3 : absent** .

#### Syntaxe

les commandes sont décrites de la manière suivante :

— **STATUS** : retourne les utilisateur avec leur statut.
— **STATUS user_name1 user_name2** : retourne le status des utilisateurs spécifiés.

#### Réponse serveur

Le code réponse à la commande **STATUS** est suivant :

**code 9000** : opération effectuée avec succès.

### 2.2.5 La commande SENDTO

La commande **SENDTO** est utilisée pour envoyer les messages privés ou groupés. La commande contient le mot clé **SENDTO**, les noms d'utilisateur et le contenu du message.

#### Syntaxe

la demande se définit la manière suivante :
**SENDTO user_name1 user_name2 ... user_nameN** *suivi d'un retour à ligne*
**leMessageAEnvoyer**

#### Réponse serveur

Les codes Réponses à la commande **SENDTO** sont les suivants :

- **code 1200** : envoi avec succès.
- **code 1201** : format incorrect.
- **code 1202** : Aucun destinataires.

### 2.2.6 La commande BROADCAST

La commande **BROADCAST** peut être utilisé pour envoyer un message à tous les utilisateurs actifs du système, toutefois le message est précédé des caractères **@@@**.

#### Syntaxe

la demande se définit la manière suivante :
**BROADCAST @@@leMessageAEnvoyer** diffuse un message dans le réseau.

#### Réponse serveur

Les codes de Réponse à la commande **BROADCAST** sont les suivants :

- **code 4000** : diffusion avec succès.
- **code 4001** : format incorrect.

# CHAPITRE 3

# IMPLEMENTATION

Le projet est composé de trois fichiers, que sont :

- **ClientTop.java**
- **Car_protocol.java**
- **Users.java**

### 3.0.1 spécifications Car_protocol.java

Cette classe utilise quartre attributs nécessaires pour les différents traitements :

- **requete(String)** : recupère les requêtes de l'utilisateur et procède à une vérification syntaxique.
- **connect()** : réalise la connexion de l'utilisateur sur le réseau de chat.
- **users()** : permet d'obtenir la liste des utilisateurs connectés.
- **status()** : donne le statut des utilisateurs.
- **sstatus()** : permet de changer le statut d'un utilisateur.
- **quiit()** : permet la fermeture de l'application.
- **who()** : permet d'identifier un utilisateur.
- **id()** : la Réponse à la commande who.
- **broadcast()** : permet la diffusion du message dans le réseau.
- **sendto** : permet l'envoi de messages à un groupe privé.

### 3.0.2 spécifications Users.java

Cette classe est utilisé pour représenter un utlisateur du système. Il est identifié par quartre attributs **son adresse IP**, **son nom**, **son port de connexion** et **son statut**.

### 3.0.3 spécifications ClientTop.java

Cette classe utilise quartre attributs nécessaires pour les différents traitements :

- **buildInterface()** : utilise pour le graphisme.
- **actionPerformed(ActionEvent)** : gère les évenements de l'interface .
- **rechercheConnecte()** : appelé une et une seule fois au lancement de l'application.
- **est_utilisateur** : vérification de l'existence des utilisateurs.
- **est_utilisateurIP()** : permet de verifier l'appartenance d'un utilisateur au réseau à partir de son IP.

— **getUser(String)** : permet de recupérer les informations d'un utilisateur à partir de son nom.
— **getUserIP(String)** : permet de recupérer les informations d'un à partir de son adresse IP .
— **portIsOpen(ExecutorService, String,int)** : verifier l'ouverture du port sur une adresse IP.
— **sendToAll()** : permet d'envoyer un message à tous les utilisateurs, utiser par la commande BROADCAST.
— **manageuser()** : une classe interne qui hérite de la classe Thread, chaque que socket créé utilise une instance de cette classe.

# CHAPITRE 4

# TESTS


FIGURE 4.1 – Connexion à l'application de chat.

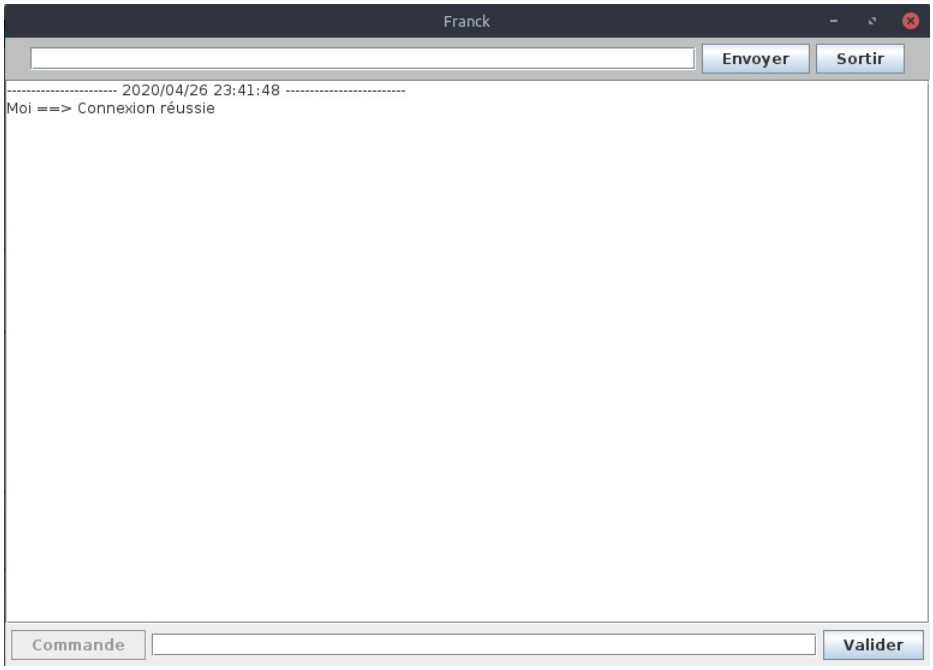


FIGURE 4.2 – utiisateur connecté au chat.

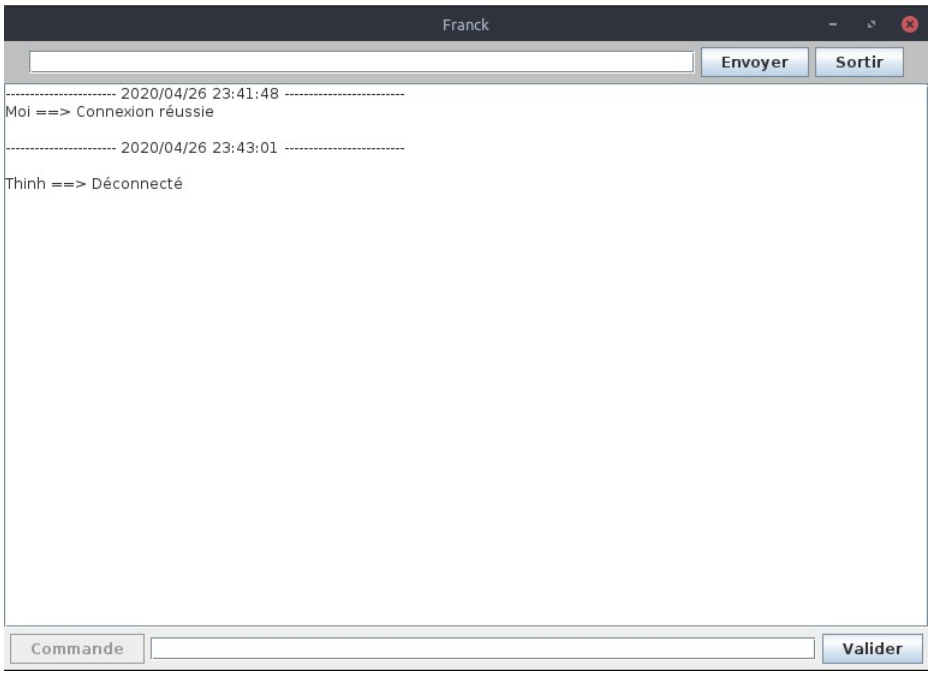


FIGURE 4.3 – utilisateur déconnecté au chat.

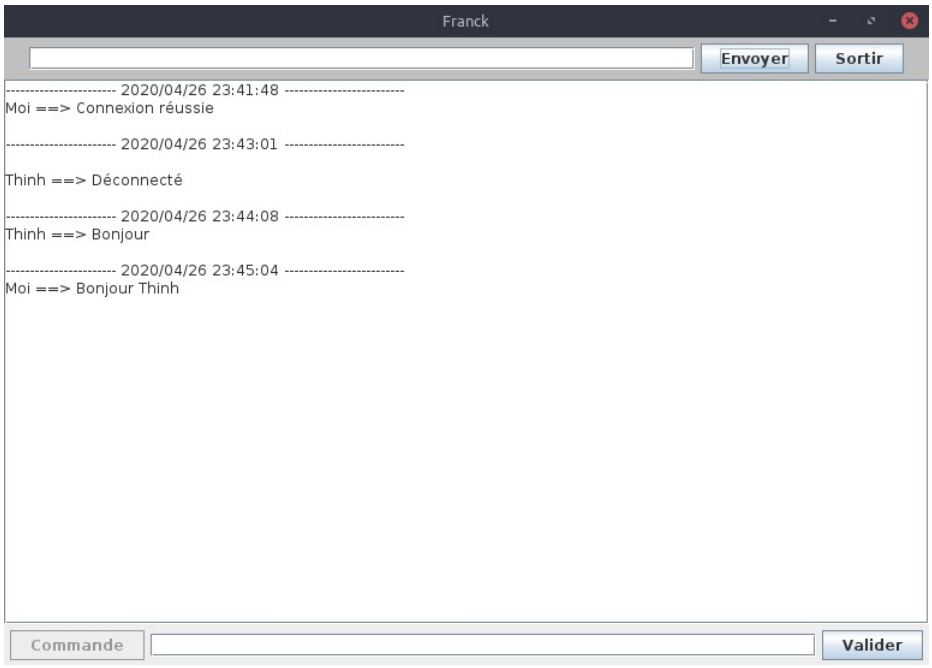


FIGURE 4.4 – echange sur le réseau chat.

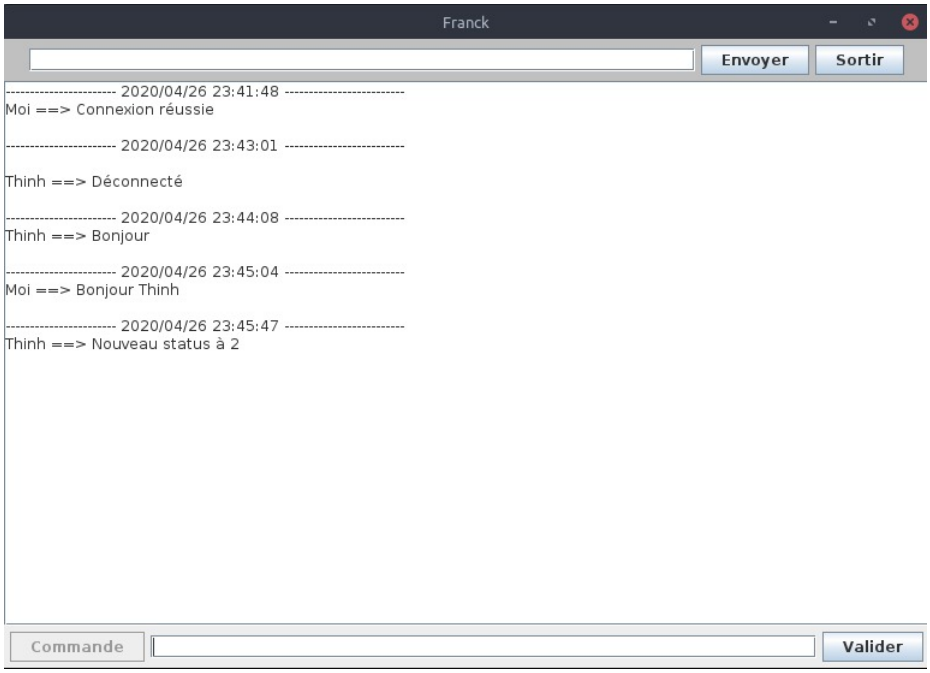


FIGURE 4.5 – changement statut de l'utilisateur.

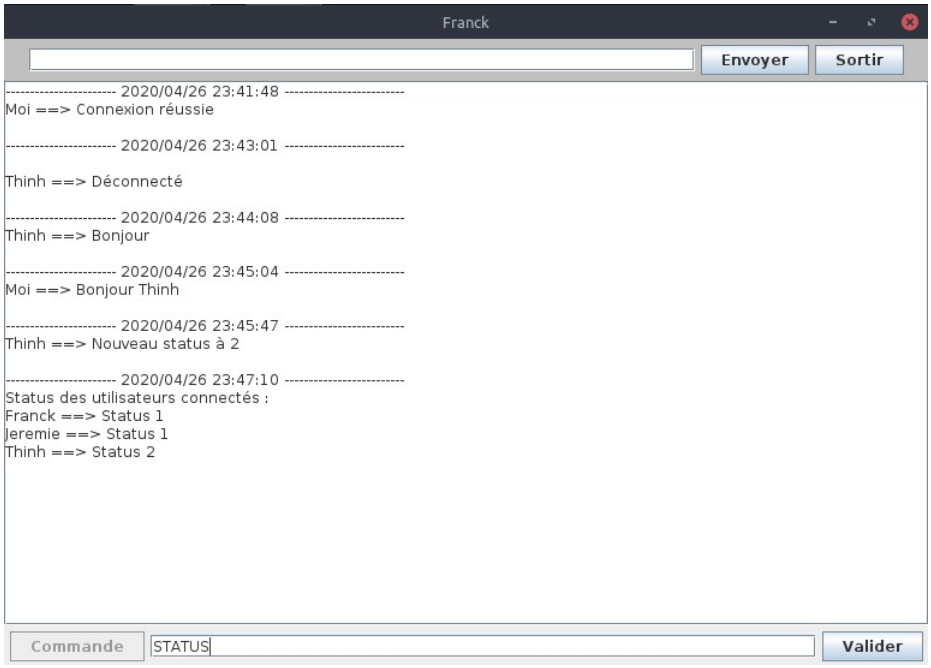


FIGURE 4.6 – liste utilisateurs sur le réseau et leur statut.


FIGURE 4.7 – réseau chat en mode client telnet.

# CONCLUSION

Le monde d'aujourd'hui qui nous entoure, est un monde connecté. À cet effet les réseaux informatiques ne cessent d'être concevoir dans le but de partager des informations, à l'instar des réseaux de chat comme Whatsapp, Messenger. Ces derniers permettent la réalisation des échanges sur la base des protocoles bien conçus respectant des normes bien définies. Certains protocoles sont ouverts à l'instar de **XMPP** (Extensible Messaging and Presence Protocol), **IRC** (Internet Relay Chat) et **SILC** (Secure Internet Live Conferencing). Et d'autres propriétaires comme, **WLM** (Window live Messenger) et le **protocole Skype**. C'est alors dans le cadre de notre module conception et architecture des réseaux, nous sommes amener à concevoir notre propre protocole et à Implementer une messagerie instantanée sur la base de ce protocole. Pour cette tâche nous definissons un ensemble de spécifications à respecter, puis nous allons concevoir le protocole et implementer l'application.