Réseau social de type Twitter

**2013**

Fabien Belli & Suzy Paeta



Sommaire

[1. Introduction 3](#_Toc356830213)

[2. Fonctionnalités de l’application 3](#_Toc356830214)

[3. Choix d’implémentation 3](#_Toc356830215)

[a. Noyau de l’application 3](#_Toc356830216)

[b. La sécurité 4](#_Toc356830217)

[c. Les images 4](#_Toc356830218)

[d. Sérialisation des Personnes et des Tweets 4](#_Toc356830219)

[e. Interface graphique 4](#_Toc356830220)

[4. Diagramme de classe 5](#_Toc356830221)

[5. Diagramme de séquence 6](#_Toc356830222)

[6. Les difficultés rencontrées 6](#_Toc356830223)

[7. Lancer l’application 6](#_Toc356830224)

[8. Conclusion 6](#_Toc356830225)

# Introduction

Le but de ce projet est de développer une application type Twitter.

# Fonctionnalités de l’application

Le client a accès à des fonctionnalités différentes s’il est authentifié ou non.

En mode non authentifié, le client peut :

* **lister les x derniers tweets à propos d’un topic**
* **lister les y derniers tweets émis, crées ou relayés par un compte utilisateur**
* **lister les topics existants**
* **lister les utilisateurs inscrits**

Un client peut **s’inscrire** et **se connecter** à son compte Twitter pour devenir un client authentifié dans le système. A la fin il peut bien sûr **se déconnecter**.

En mode authentifié, le client peut faire les mêmes actions que le client non authentifié et :

* **écrire un tweet**
* **écrire un tweet et y joindre une image**
* **relayer un tweet**
* **follower un utilisateur**
* **afficher son fil d’actualité** (afficher les tweets des personnes que le client follow)

# Choix d’implémentation

## Noyau de l’application

Pour ce projet, nous avons choisit de mettre en place une architecture client/serveur en utilisant l’API Java RMI. Le serveur est un objet distant qui gère les utilisateurs et centralise les données (tweets, comptes utilisateurs...). C’est lui qui offre des fonctionnalités aux clients. Ceux-ci grâce à RMI peuvent invoquer des méthodes du serveur à distance (ex : se connecter, écrire un tweet…).

Le client est lui aussi un objet distant et *extends* de *UnicastRemoteObject* afin que le serveur puisse obtenir une référence sur le client et donc appeler ses méthodes (ex : ).

Le client récupère un stub vers un objet distant côté serveur qui est différent selon qu’il soit connecté ou non. Pour cela nous avons créée deux interfaces nommées InterfacePrivee et InterfacePublic. Lorsqu’il n’est pas authentifié, le client n’a accès qu’aux méthodes de InterfacePublic, et lorsqu’il est connecté il a accès aux deux interfaces, il peut donc accéder à toutes les fonctionnalités de Twitter.

Vous pouvez donc constater que pour la gestion de l’authentification et des autorisations nous n’utilisons pas le framework Java JAAS. Nous avons cependant mis en place des sockets sécurisés.

## La sécurité

Le client devant s’identifier avec un login et un mot de passe, il est clair qu’il faut faire attention à la manière dont ces données sont envoyées sur le réseau. C’est pour cela que nous avons mis en place des sockets sécurisées, des sockets SSL. Cela permettra de sécuriser la transmission des données. Pour mettre en place ces sockets, nous avons créée deux classes « Factory » : RMISSLClientSocketFactory et RMISSLServerSocketFactory qui implémentent RMIClientSocketFactory respectivement RMIServerSocketFactory. Elles nous permettent de créer des sockets pour le serveur et pour le client. Deux instances de ces Factory sont passées en paramètre du constructeur du Serveur qui les passera au constructeur de sa classe mère : UnicatRemoteObject.

## Les images

Les tweets envoyés peuvent être accompagnés d’une image. Pour mettre en place cela, il a fallu choisir une classe Java gérant les images et qui implémente Serializable afin de pouvoir être transmise sur le réseau. Nous avons donc choisit que nos images seraient de type ImageIcon. On a donc une classe Twitt qui gère les tweets classiques et une classe dérivée TwittImage qui gère les tweets contenant des images.

## Sérialisation des Personnes et des Tweets

## Interface graphique

Même si l’interface graphique n’était pas vraiment demandée, nous en avons tout de même implémenté une, surtout pour faciliter les tests des méthodes et rendre l’application plus agréable. En revanche elle n’est pas complètement ergonomique ni esthétique.

# Diagramme de classe

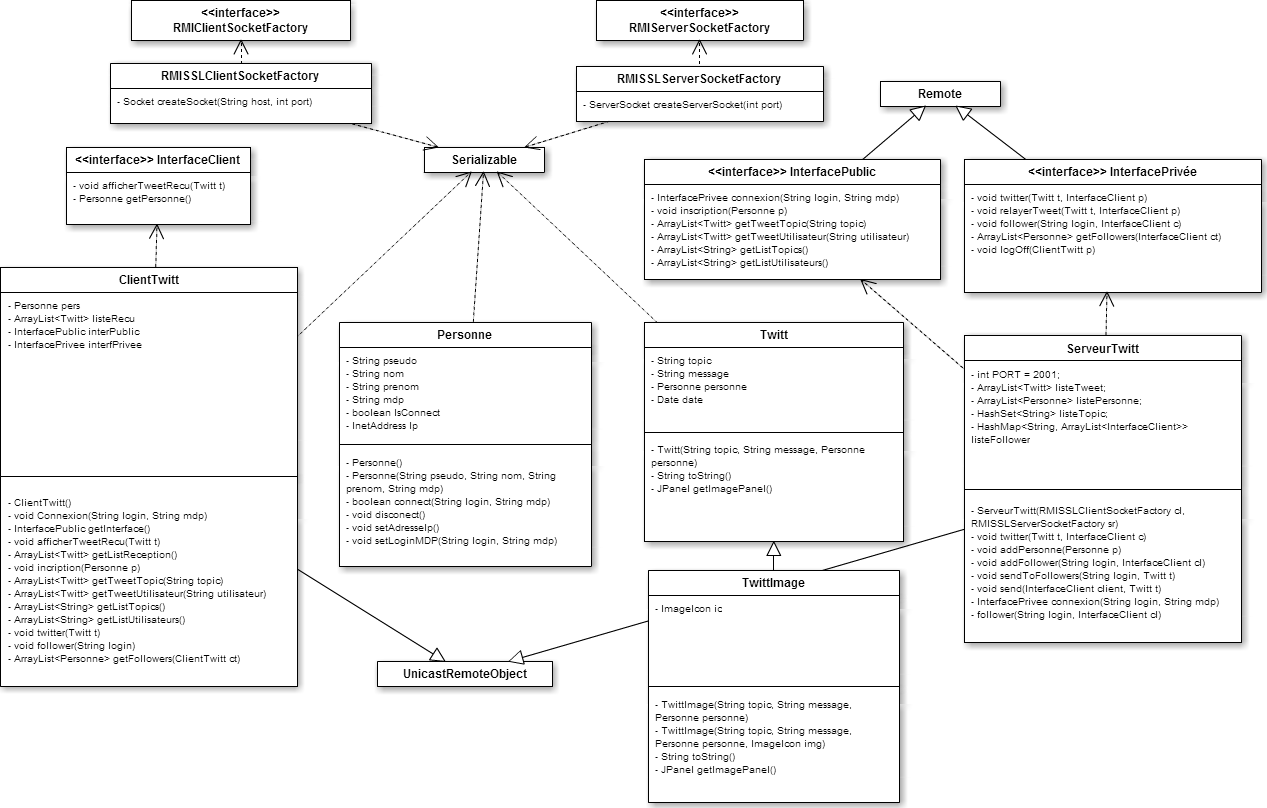


Figure 1 : Diagramme de classe de notre application Twitter

# Diagramme de séquence

# Les difficultés rencontrées

# Lancer l’application

L’application présente une interface graphique minimaliste afin de pouvoir tester toutes les fonctionnalités implémentées. Il faut cependant lancer le serveur et les clients en console comme décrit ci-dessous :

# Conclusion

….

Vous pourrez aussi remarquer l’effort particulier que nous avons fournit pour commenter notre code.

1. pourquoi on a une classe personne qui est dans le package Serveur ?

la classe personne permet d'identifier un client par son nom, prenom, mdp...

elle se trouve normalement dans le client et le serveur, c'est mieux que de faire un HashMap contenant login et mdp, enfin je trouvais...

 2) A quoi servent les classes RMISSLServerSocketFactory  et RMISSLClientSocketFactory

ces dernières servent pour des  connexions SSL securisés (fin du TP 4)

3) a quoi ça sert d'avoir 2 interfaces, une privée et une publique

**Fabien:** C'est la même chose que pour Jaas, quand le login a fonctionné, il retourne un personal que le client est le seul a avoir, ce personal lui donne des privilèges et il ne peut executer que les métthodes qui contiennent doAsPrivilège

**moi:** d'accord mais j'ai pas fait jaas encore lol

oui par contre je suis toujours sceptique pour la classe personne ^^

pour toi c'est une personne qui a un compte twitter ?

**Fabien:** Oui c'est comme si tu mettais un login, tu es d'accord que c'est une personne qui emet un twitt

**moi:** en fait j'ai du mal a savoir pourquoi on a un clienttwitt ET une personne

**Fabien:** je ne peut pas stocker des references (ClientTwitt) car elle ne seraient jamais à jour quand le client se connect et se deconnect