Middleware : Vente aux enchères

Léo CASSIAU Geoffrey DESBROSSES
Jean-Christophe GUERIN Ugo MAHEY

18 décembre 2016

1 Introduction

Ce document résume le travail effectué au cours du projet de Service de 2e année du master ALMA 2016 - 2017. Ce projet est mené par des étudiants de la faculté des sciences et techniques de Nantes, et le sujet est proposé par M.MOSTEFAOUI.

L'objectif de ce projet est de comprendre comment fonctionne les applications distribuées. Pour cela, nous avons dû développer une application avec un serveur et un client qui communiquent. Le serveur doit impérativement pouvoir gérer le multi-clients, et le moyen de communication doit être Java RMI.

Le sujet du projet est de construire une plateforme de vente aux enchères. Plusieurs clients doivent pouvoir s'authentifier puis participer aux différentes enchères proposées par le serveur.

Les principales problématiques que soulève ce sujet tournent autour de la synchronisation entre les différentes fonctions et les différents clients. En effet, il faut réfléchir à un ordre d'appels et coment mettre en place un middleware.

Ce document commence par présenter les différentes fonctionnalités proposées, puis présenter l'implémentation du serveur et du client pour enfin voir comment ils communiquent entre eux sans se gêner.

2 Développement de l'application

2.1 Les fonctionnalités

L'application que nous avons dû développer est une plateforme de vente aux enchères. Nous avons choisi de développer les fonctionnalités essentielles de ce genre de système.

La plateforme doit pouvoir présenter des objets à vendre, puis pour chaque objet, choisir un unique gagnant qui remporte l'enchère. Une fois tous les objets d'une vente sont vendus, la plateforme doit pouvoir recréer une vente avec de nouveaux objets. De plus, les participants peuvent proposer leurs propres objets à vendre qui seront ajoutés à la vente en cours.

Au niveau des clients, ils doivent pouvoir enchérir sur un objet ou passer leur tour. Un chrono doit être implémenté afin de passer son tour automatiquement à partir d'un certain temps. De plus, si deux clients proposent un même prix, c'est celui qui aura répondu le plus rapidement qui remportera l'enchère (donc celui avec le chrono le plus haut).

2.2 Choix d'implémentation

Pour implémenter cette application, il nous a fallu développer un client et un serveur. Ceux-ci dialoguent via la technologie Java RMI.

2.2.1 Le client

Le client est composé d'une interface Acheteur, ce sont les méthodes de cette interfaces que le serveur va appeler. Cette interface possède les fonctions suivantes :

- objetVendu(String gagnant) : Cette méthode permet au client de savoir que l'objet courant a été remporté par quelqu'un.
- nouveauPrix(int prix, Acheteur gagnant) : Cette méthode est appelée lorsqu'un tour est passé et que l'enchère sur l'objet courant a été modifiée (prix différent et/ou gagnant potentiel différent).
- finEnchere() : Cette méthode est appelée lorsque tous les objets sont vendus et que l'enchère est finie.

Le client possède également ses propres méthodes que le serveur ne peut pas appeler. Parmi celles-ci, on retrouve connexionServeur qui permet de se connecter au serveur grâce à son adresse IP. La méthode encherir permet de donner un prix de l'objet courant puis de l'envoyer au serveur. La méthode nouvelleSoumission permet au client d'ajouter un objet aux enchères. Enfin, la méthode inscription permet au client de s'authentifier puis d'attendre la prochaine vente pour pouvoir y participer.

Ces méthodes ne font que changer les paramètres du client (changer l'IHM, l'état du client, le chrono...) puis d'appeler les méthodes du serveur correspondantes via Java RMI.

Le client possède un chrono qui se déclenche lorsque l'on le notifie. Si le client n'enchérit pas avant la fin du chrono alors la méthode encherir est appelée avec la valeur -1. Si le client enchérit avant, alors il utilise la méthode encherir avec le montant qu'il a renseigné.

Une IHM développée en swing est également présente dans la classe VueClient. Celle-ci est actualisée régulièrement selon les évènements qui se produisent.

2.2.2 Le serveur

Le serveur doit gérer les appels des différents clients. Pour cela, il expose des méthodes sur son interface Vente :

- inscriptionAcheteur(String login, Acheteur acheteur) : Cette méthode permet à un client de s'inscrire à la vente en cours. Cependant, celui-ci doit attendre dans une file d'attente si un objet est déjà en cours

de vente. Il rejoindra les autres acheteurs lorsque l'objet courant aura changé.
— rencherir(int nouveauPrix, Acheteur acheteur) : Cette méthode est appelée lorsqu'un client enchérit sur l'objet courant. On attend toutes les réponses de tous les clients qui participent puis on modifie le prix de l'objet courant puis le gagnant potentiel. Ensuite on renvoi cette information aux clients en appelant leur méthode nouveauPrix(). Si tous les acheteurs passent leur tour (enchérissent -1) alors on change d'objet courant.
— ajouterObjet(Objet objet) : Cette méthode permet simplement d'ajouter un objet à la liste des objets à vendre.

Le serveur possède une classe Donnees, celle-ci mock une vraie base de données dans laquelle elle prend des objets à vendre et les proposes aux echères. A la fin de chaque enchère, on appelle la méthode init() pour recréer une nouvelle vente proposée directement aux différents clients.

2.3 La communication client/serveur

Comme dit précédemment, nous avons utilisé Java RMI pour communiquer entre le client et le serveur. Cependant, certaines méthodes ne doivent être accessibles par plusieurs clients en même temps. Il faut qu'elles soient synchronisées. Dans notre application, certaines méthodes possèdent l'attribut synchronized pour gérer ce genre de problème.

Avec java RMI, ce sont les méthodes du client qui appellent celles du serveur, et vice-versa. Ainsi, il est facile de voir l'enchainement des différentes méthodes appelées.

3 Conclusion

Notre plateforme de vente aux enchères est fonctionnelle. Les actions présentent dans le cahier des charges sont toutes présentes. Ainsi, nous avons un serveur qui gère le multiclient et pouvant vendre des objets.

Lors de ce projet, nous avons vu plusieurs aspects des applications distribuées. Il faut pouvoir gérer des problèmes de concurrence et pouvoir dialoguer entre le serveur et le client. L'important est de réfléchir avant de développer l'application, se demander comment structurer le programme.

Ici nous avons réalisé notre application en Java, avec une interface graphique et des technologies qui lui sont propres (Swing et Java RMI). Cependant, nous aurions pu la développer dans d'autres technologies qui sont beaucoup utilisées aujourd'hui, par exemple gérer la communication via HTTP avec une interface web. Java RMI n'est qu'un protocole qui nous a permis de mieux comprendre coment fonctionnent les applications distribuées.