Auction System

Distributed Algorithms and Systems 4

Adrián Gómez Llorente - 0904327G

1. Introduction

An auction system will be explained through this document. To make it easy to follow, next all chapters are showed.

1. Design of the system
2. Implementation of the Auction system
3. Testing of the application
4. Performance
5. Potential extensions
6. User manual

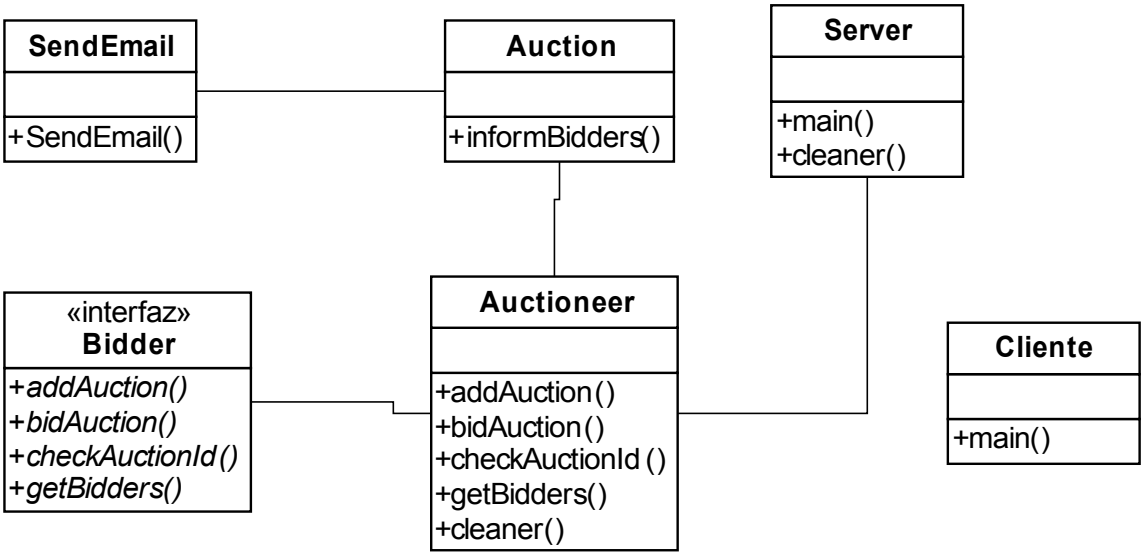
2. Design of the system

2.1 Introduction

The design of the Auction system is based in RMI (Remote Method Interface). This means that it's a server-client application with a server side that includes all software required to maintain auctions, bidders and bids, and a client side that implements all of the client operations.

2.2 Design of the application

Next, a class diagram shows how the application has been implemented, parameters and some methods like, get/set and constructors, are not shown to simplify the diagram.

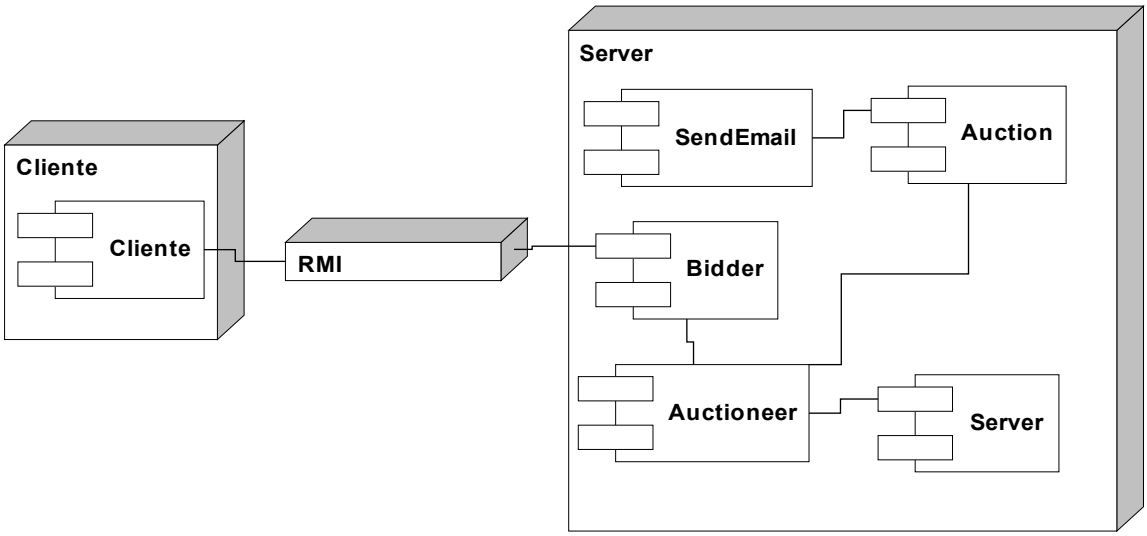


- SendEmail: Is a class that gives methods to send emails to specified directions.
- Auction: Is a class that represents an auction and all the methods needed like get/set and

inform bidders when the auctions has finished.

- Server: Is a class that contains a main method to turn on the server-side application.
- Auctioneer: Is the most important class in the server side. It controls all the operations related with the system like adding an auction, bidding it, get bidders, etc. Moreover, it has a method to clean all ended auctions.
- Bidder: Is an interface that make some of the Auctioneer's method remote, is used by the client.
- Client: Is a class that uses RMI to connect with Bidder and controls all the operations in the client-side, like showing data, menus, etc.

We have seen all the classes that complete the application but this is an RMI application show a component diagram is needed to represent how all the classes are connected. Next, one can be seen:



In the previous diagram we can see that the class Client is in the client-side and is connected using RMI to the class Bidder that is server-side.

More than 1 client can use the server at the same time with only the creation of other Client object in other PC.

3. Implementation of the Auction system

How the application has been created is show in the previous chapter but right now we are a going to take a look to what operations can be done using this auction system.

- Create an auction: All the auctions have the same duration and only a name and a price have to be filled to add one.
- View auctions: Every client can see what are the current auctions.
- Make a bid: If a client wants to make a bid for an auction he only has to put an email to be informed about the status and his offer.

- Inform bidders: All bidders are informed when an auction has finished with information like, minimum reached and winner. Why by email? To keep information save despite of disconnected situations.
- Clean ended auctions: All auctions are deleted from the system when the end date is reached.
- Many clients supported: This system supports many clients simultaneously thanks to synchronized methods.

4. Testing of the application

The application has been tested to check that all offered operations are made correctly. To complete these tests I have completed some system tests and results have been collected. Next, all tests and results can be seen:

1. Create an auction
    - *Test:* Client an auction from a client and see if it's added to the current auction list from other different client.
    - *Result:* The auction is created and added to the current auction list, this way the other client is capable of seeing the auction.
2. View auctions
    - *Test:* One client try to see auctions when the server doesn't have auctions hosted and when other client has added one.
    - *Result:* One client has tried to see auctions when there is not auctions created and he has gotten an empty list instead of an error. Moreover, one client is capable of seeing auctions created by other client.
3. Make a bid
    - *Test:* One client make a bid on an auction and other client can see it in the auction's bid list
    - *Result:* One client make a bid and the other client see it in the auction's bid list.
4. Inform bidders
    - *Test:* Various clients bid on an auction and check if they have received email with information.
    - *Result:* All the clients have received the email with the information.
5. Clean ended auctions
    - *Test:* One client creates an auction and show the list until check if it's gone.
    - *Result:* The auction was gone since it has ended.
6. Many clients supported
    - *Test:* One server is launched and many clients created.

- *Result:* All the clients were created and they were capable of working together and bidding in the same auctions.

5. Performance

5.1 Using HashMap

Server is always maintaining a list of the current auctions and Auctions are maintaining a list of the bidders and their higher bid. These to lists are using HashMap for indexing both auctions and bidders:

- Auctions are indexed using their unique ID
- Bidders are indexed using their email

So every auction and bidder is hashed into a HashMap, so that, every access is really fast because they are all indexed using a hash code and this is better than using an ArrayList.

5.2 Using daemons

The application uses daemons for some actions, for example for cleaning ended auctions. This daemon is always running in background and does not interfere with other methods. Moreover, this daemon might be installed in other server for cleaning auctions.

5.3 Avoiding bottlenecks

Most known bottleneck in RMI is object serialization. This application try to avoid this by using simple parameters like String, int, float, etc. to be sent between server and client. This way all operations are made in the server-side and only results are sent to client, moreover, security is increased because all operations depend of the server.

6. Potential extensions

This application can improve the fault tolerance and availability with some extensions like:

- Web based interface: This way application will be more available because it might be used 24 hours a day through the whole year.
- A database: A database would make the application more scalable and would improve its integrity. For example a cleaner wouldn't be needed to delete ended auctions and a history of bidders and auctions might be saved.

7. User manual

7.1 Running the application

Follow next steps to run the application:

- Windows
  1. A compressed file called *0904327G*.tar contains all the files needed to run the application.
  1. Uncompress it with winrar, for example, right click, extract it here.
  2. Be sure port 1099 is available
  3. Go to AuctionSystem/execute folder and double click on server.bat and client.bat for running server and client.
- Linux
  1. A compressed file called *0904327G*.tar contains all the files needed to run the application.
  1. Open a shell and go to the folder that contains *0904327G*.tar
  1. Uncompress it with: *tar -xvf ./ ./0904327G.tar*
  2. Be sure port 1099 is available
  3. Go to AuctionSystem/execute folder and run server and client with these commands:
     - java -jar Server.jar & (to run it in background)
     - java -jar Client.jar