SYSTEM DESIGN DOCUMENT

Distributed Auction Service Using CORBA

Shikhar Vats

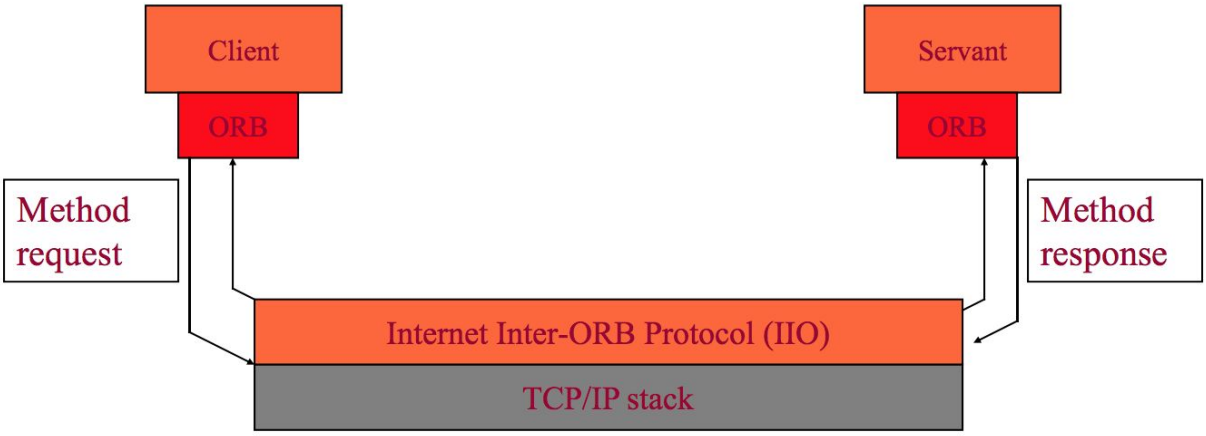
1 INTRODUCTION

1.1 Purpose and Scope

The objective of this machine problem is to use CORBA to build a Distributed Auction Service, which allows the buying and selling of individual items. For this project we will be using an English auction protocol (increasing price, current price visible to all parties).

1.2 Project Executive Summary

1.2.1 System Overview



Simple Architecture of CORBA

Our application system consists of two basic modules the client and the server:

It includes the following features:

Client side:

- The command line User Interface
- Browsing the active auctions
- Bid on the active item
- Creating a new auction
- Checking the status of an auction/bid
- Checking the highest bidder

Server side:

- Create a new auction
- Update an item upon the status change
    - sell, create, bid

Data associated with each bid:

- User ID, item ID, bi price

Data associated with each new auction:

- Item name, seller ID, start price(optional)

Implementation:

- CS-Communication: CORBA. Interface defined in a .idl file
- Client: Java

The Interface file for the project

```
module DistributedAuctionApp{
    interface DistributedAuction{
        exception IncorrectOfferException{
        string description;
        };
        exception IncorrectSellException{
        string description;
        };
        exception IncorrectStatusException{
        string description;
        };
        exception IncorrectBidException{
        string description;
        };

    typedef struct AuctionStatus{
        string userID;
        string itemName;
        double bidPrice;
    } Status;

    boolean offerItem(in string itemName, in string userID, in double startPrice) raises (IncorrectOfferException);
    string viewHighBidder(in string userID) raises (IncorrectSellException);
    Status viewAuctionStatus(in string userID) raises (IncorrectStatusException);
    string viewBidStatus(in string userID) raises (IncorrectBidException);
    boolean bid(in string userID, in double price) raises (IncorrectBidException);
    boolean sell(in string userID) raises(IncorrectSellException);
    };
};
```

2 SYSTEM ARCHITECTURE

The system architecture has three main components:

1. The client side: DistributedAuctionClient.java
2. The server side: DistributedAuctionServer.java

3. The seller side: DistributedAuctionSeller.java

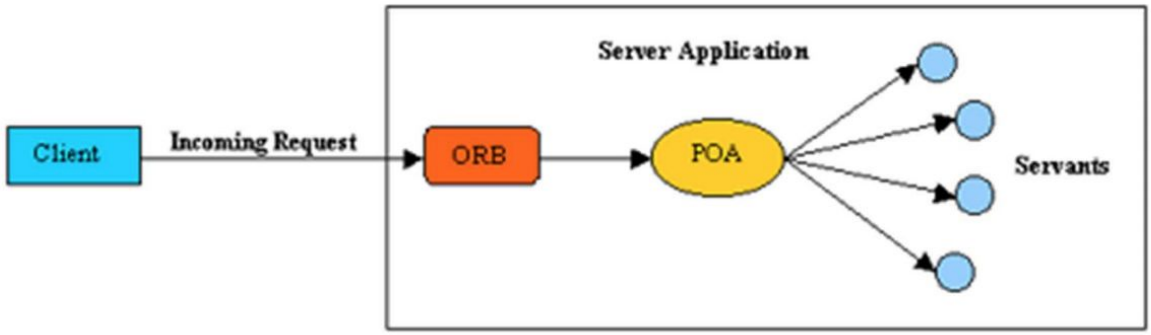
The server and the servant part of the application operate out of the same directory, while the client operates from a different directory.

2.1 Internal Communications Architecture

Following procedure is followed while establishing a communication:

The Object Request Broker:

1. Uses Object Reference to identify and locate objects
    a. Object Reference: A handle to an object that a client must hold in order to access the object
2. Delivers request to objects
3. Returns output values back to client
4. Services necessary to accomplish the tasks are completely transparent to the client



3 HUMAN-MACHINE INTERFACE

The interface for the Distributed Auction program is completely command-line based.

3.1 Inputs

The program asks the user for an input after displaying a menu.

For the client who is a seller:

The input can be any number ranging from 1-5.

1. Create an auction item
2. Get the highest bidder of the current item
3. Get the highest bidder on the current item
4. Get the status of the auction
5. Quit

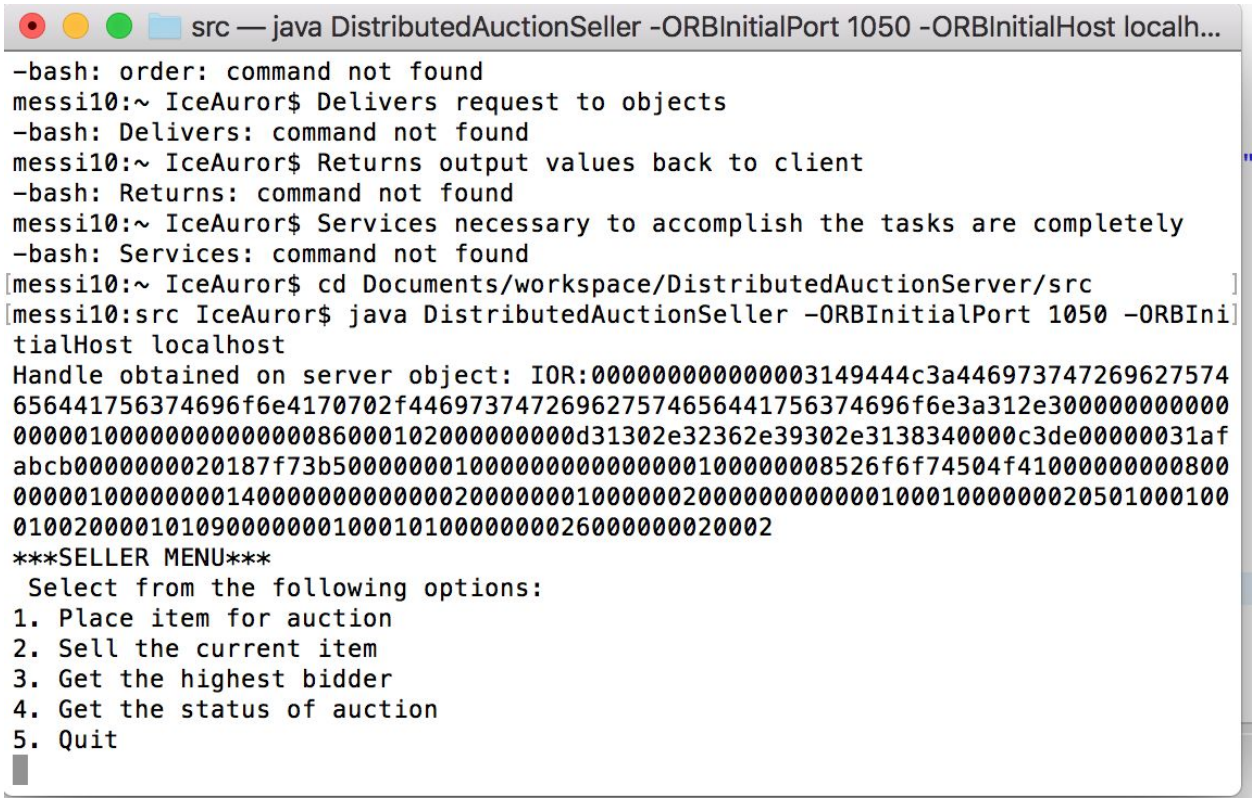
For the client who is a buyer:

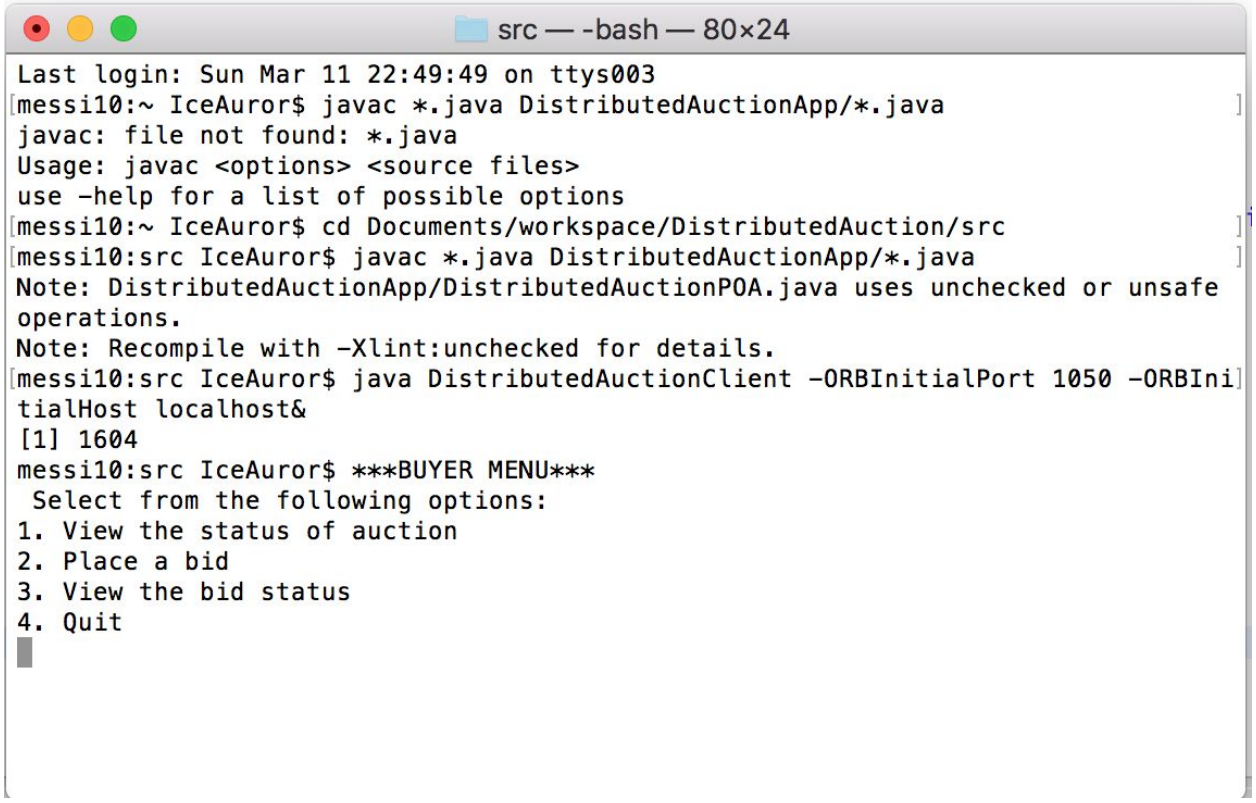
The input can be any number ranging from 1-5.

1. Get the status of the auction
2. Place a bid on the item
3. Check if he/she is the highest bidder
4. Quit

3.2 Outputs

Based on the input, the program returns the relevant output from the server side.


src — java DistributedAuctionSeller -ORBInitialPort 1050 -ORBInitialHost localh...
-bash: order: command not found
messi10:~ IceAuror$ Delivers request to objects
-bash: Delivers: command not found
messi10:~ IceAuror$ Returns output values back to client
-bash: Returns: command not found
messi10:~ IceAuror$ Services necessary to accomplish the tasks are completely
-bash: Services: command not found
messi10:~ IceAuror$ cd Documents/workspace/DistributedAuctionServer/src
messi10:src IceAuror$ java DistributedAuctionSeller -ORBInitialPort 1050 -ORBIni
tialHost localhost
Handle obtained on server object: IOR:000000000000003149444c3a446973747269627574
6564417563746 96f6e4170702f4469737472696275746564175637469 6f6e3a312e300000000000
0000010000000000000086000102000000000d31302e32362e39302e3138340000c3de00000031af
abcb0000000020187f73b500000001000000000000000100000008526f6f74504f41000000000800
0000010000000014000000000000020000000100000020000000000001000100000020501000100
0100200001010900000001000101000000026000000020002
SELLER MENU
 Select from the following options:
1. Place item for auction
2. Sell the current item
3. Get the highest bidder
4. Get the status of auction
5. Quit


src — -bash — 80×24
Last login: Sun Mar 11 22:49:49 on ttys003
messi10:~ IceAuror$ javac *.java DistributedAuctionApp/*.java
javac: file not found: *.java
Usage: javac <options> <source files>
use -help for a list of possible options
messi10:~ IceAuror$ cd Documents/workspace/DistributedAuction/src
messi10:src IceAuror$ javac *.java DistributedAuctionApp/*.java
Note: DistributedAuctionApp/DistributedAuctionPOA.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for details.
messi10:src IceAuror$ java DistributedAuctionClient -ORBInitialPort 1050 -ORBIni
tialHost localhost&
[1] 1604
messi10:src IceAuror$ ***BUYER MENU***
 Select from the following options:
1. View the status of auction
2. Place a bid
3. View the bid status
4. Quit