CS5204 Operating System Programming Project Report

Flight Reservation System Using Java RMI

By Jing Ma and Jiang Shu

Fall, 2000

Table of Contents

Abstract

The system is designed for a Flight Reservation System using the Java Remote Method Invocation (RMI) as the underlying technology. The system consists of two servers and a client (Notice, multiple instances of client are implemented). In a nutshell, clients will submit their request via a graphics user interface (GUI) to Server1. Then after searching its database (MS Access) and consulting with Server2 (which will also look up its database), Server1 will either tell the client to revise his/her request to submit it again (due to 0 search result) or return all the hits and let the client choose his/her flight and make the reservation. Then acting upon the reservation request, Server1 will update its database accordingly. The objective of building this system is to get a better understanding of distributed system and Java RMI mechanism, and use Java RMI to build a practical distributed system.

Introduction

The system is trying to conduct a typical business function, a Flight Reservation System, in a distributed fashion. In this case, client GUIs, designed by using Java Swing Package, can be easily run at all kinds of sale representative machines no matter he/she is taking the order from their cubicles in the Main Flight Sale Department or from his/her own household to get the job done. (Of course, there are some other architecture options and design strategies that we can implement this system. We will come back to this topic in the Suggested Variations and Extensions section). People may wonder who gives us all these power and flexibilities. The answer is RMI. Since RMI allows an object (in our case, the client) running in a Java Virtual Machine (VM) to invoke methods on a remote object (in our case, Server1 and Server2) running in another Java VM. But the most important feature of this remote communication provided by RMI is its transparency. There is no difference between making a call to a local method and to a remote method when we are using RMI.

As part of the distributed system, Server1 and Server 2 can act as independent servers sitting in different departments. Server1 can sit in the Flight Control Department, which controls the flight information such as flight number, departure location, arrival location, number of passengers, departure date and return date. All these flight information are stored in a MS Access database. Server1 is using JDBC-ODBC to talk to the database to get desired flight information. A Writer Priority Monitor is implemented surrounding Server1's access to the Database since there are possibilities that multiple clients want to read the available flight information and reserve (write) flight simultaneously. By distributing Server1, we assume the Flight Control Department can update flight information (such as add/delete flights, change the flight departure date and time and so forth) independently without intervening the other parties' work.

Before Server1 responses to the client, it will also use a RMI call for Server2 to get the price of the flights, where we assume Server2 is sitting in another Department such as Finance Department, which can manage price changes independently such as changing the prices due to different seasons or competitions by giving promotion and coupons. Server2 also talks to its own database residing with it, upon requests from Server1.

After the client gets responses from Server1, it will display all the available flight combination including both the departure and return flights if there is any or ask the client to change the request due to 0 hits are found. When the client makes a reservation, Server1 will either take it if there are still empty seats available or reject it if possibly one of the multiple readers took up the last seat already. Please refer to the Figure1 – System Components and Interaction for a better understanding of the system.

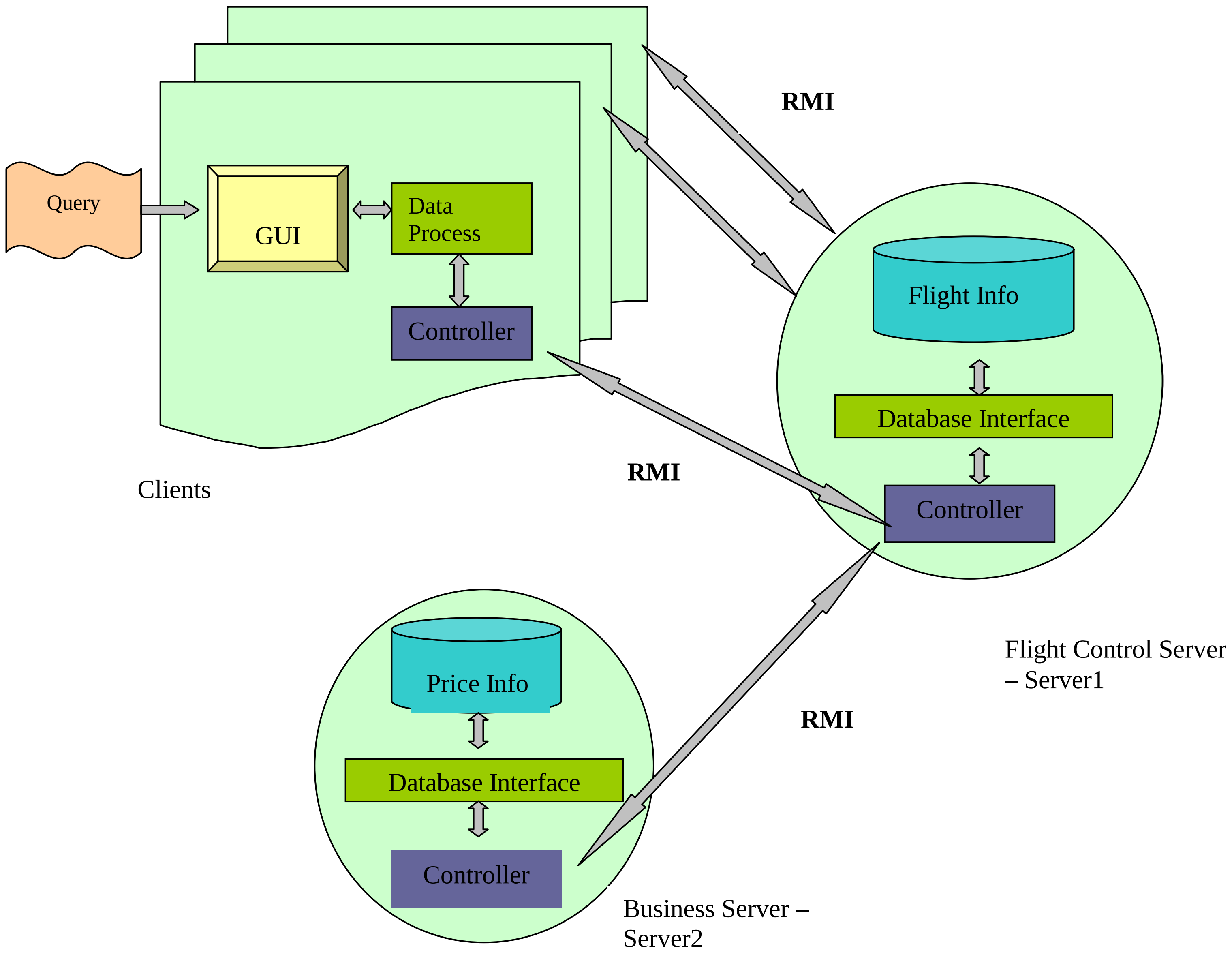


Figure1 – System Components and Interaction

System Design

- Interface

As we mentioned, by using Java RMI technology, we placed the interface files (Stubs and Skeletons) under a shared drive for all parties to access. The structure of the interface files – Flight Package is showed in the following Diagram:

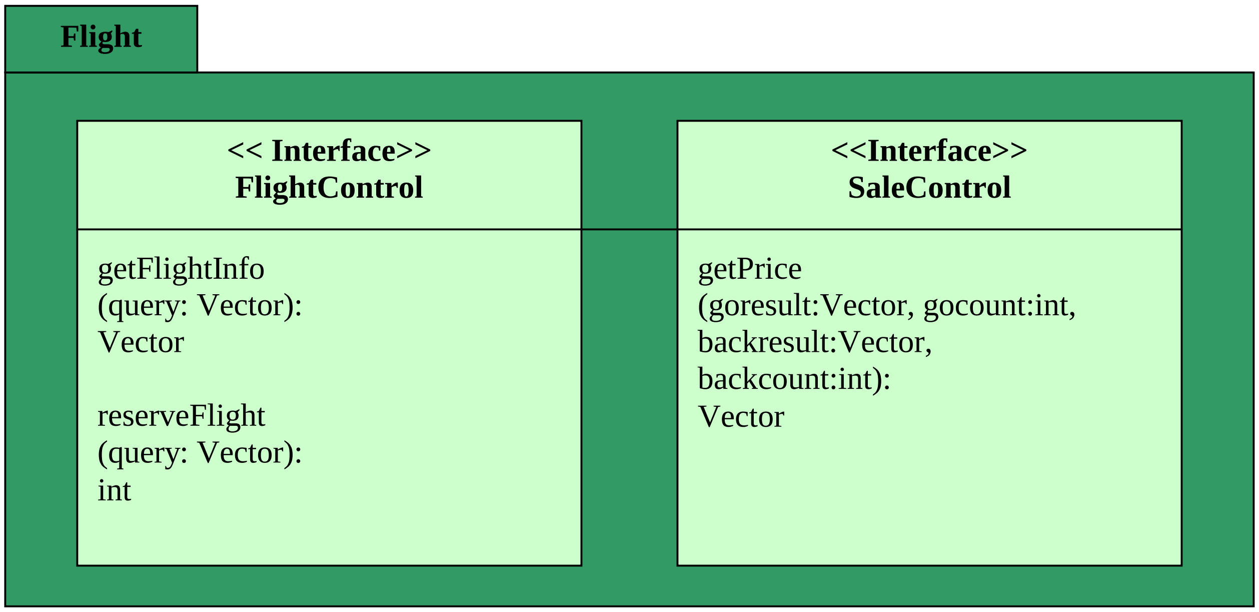


Figure 2 – Flight Package Diagram

The ***FlightControl*** interface connects clients and Server1 by using *getFlightInfo()* function call when the clients submit their requests to Server1. The ***SaleControl*** interface connects Server1 and Server2 by using *getPrice()* function to obtain corresponding flight prices.

- Client

Client calls *getFlightInfo()* to get flight information when user clicks the "submit " button in SaleClient interface, and display the search result to users by calling *ShowFlightInfo* class . When users want to make a reservation, the *reserveFlight()* function is activated. Please refer to Figure 3 – Client Package Diagram for details.

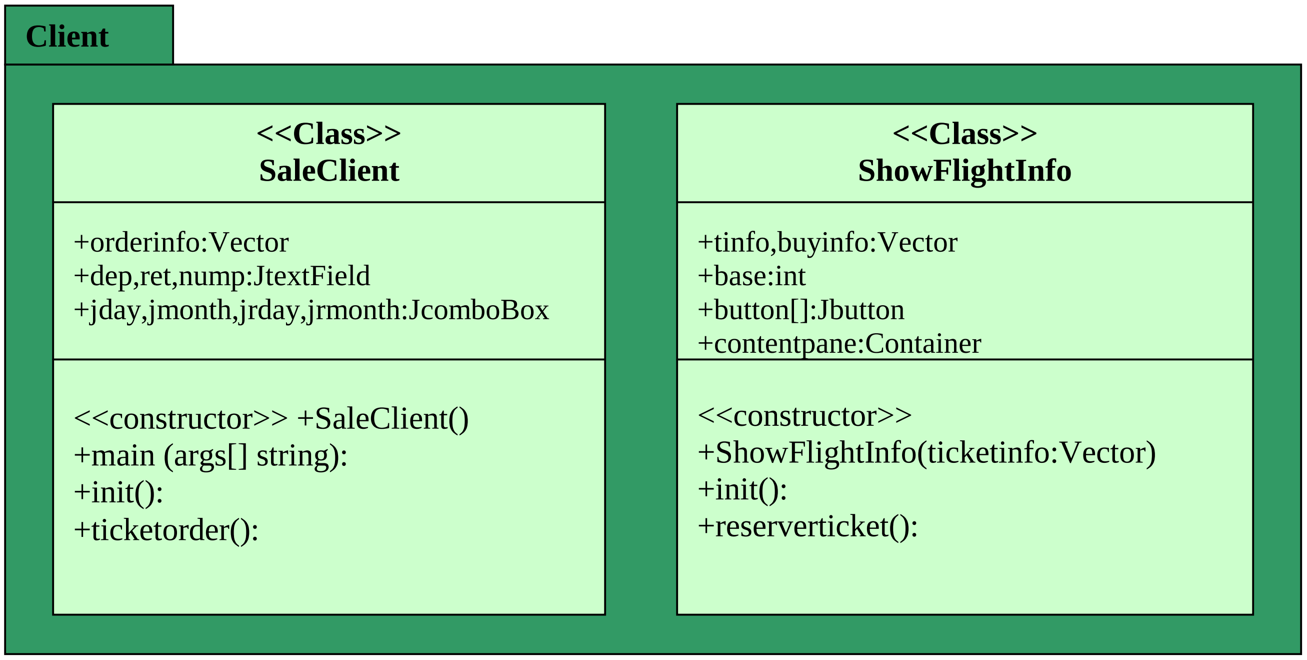


Figure 3 – Client Package Diagram

- Server1

Server1 resides in a machine other than the one that client is running. Acting upon client requests, it will search *flightcontrol* database to find available flights, call Server2 to get those flights' prices by using remote function call *getPrice*(), and send the available flight information back to the client . Server1 will also call *reserverFlight()* function to update the *flightcontrol* database when users want to make a reservation. A writer priority Monitor is implemented in Server1 to ensure exclusive database access and to reduce the network load by avoiding unnecessary reading and writing. The Writers-Priority Monitor will give the resource to a writer, if there are writer or both writers and readers are waiting. Only if there is no writer waiting, it will give its resource to readers if they are any. Figure 4 – Server1 Package Diagram illustrates the structure of the Server1 package.

- Server2

Server2 has the similar structure as Server1. It will communicate with Server1 and respond to Server1's invocation solely. It provides the price information by accessing a MS Access database *salecontrol.mdb*.

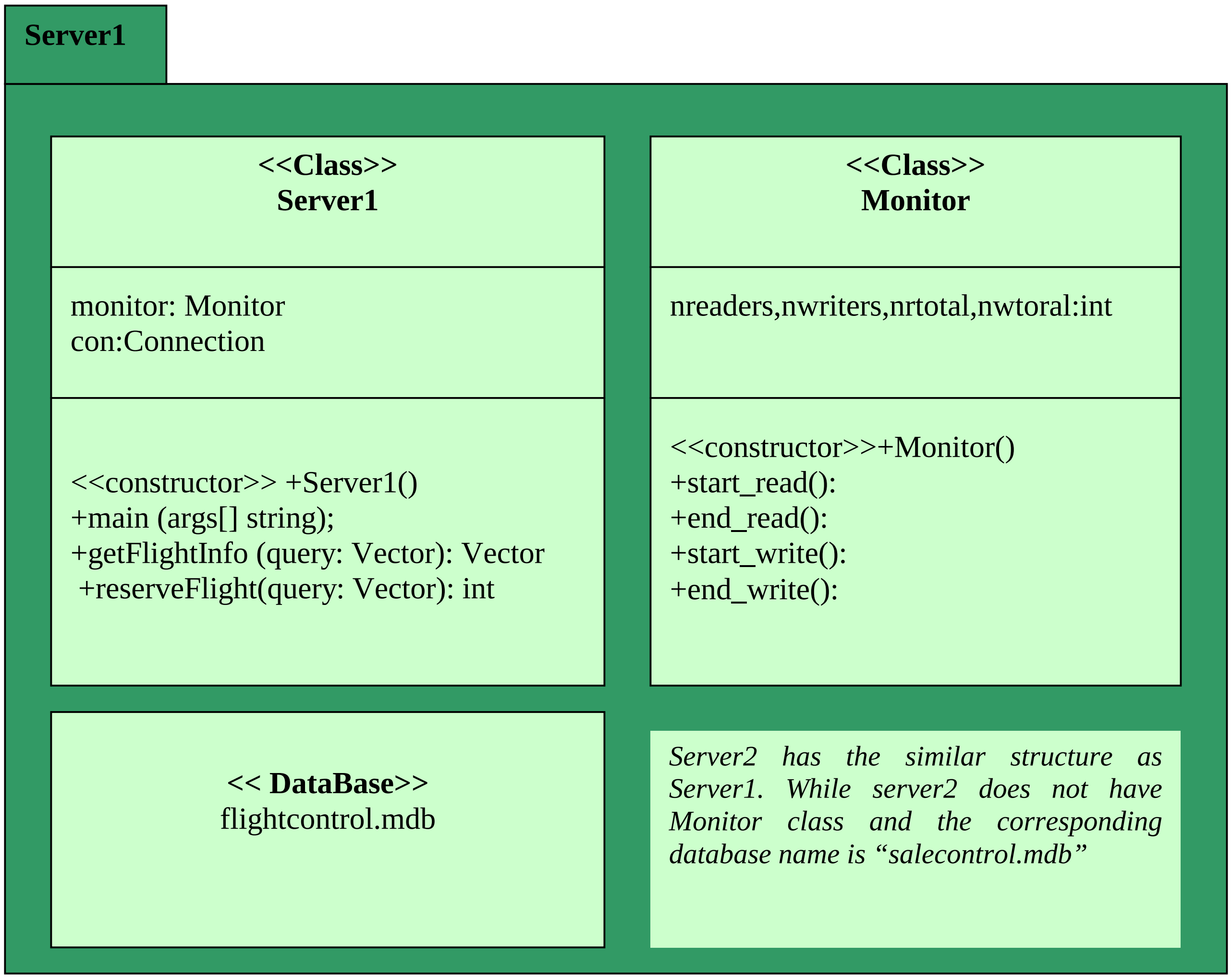

Server1
<<Class>>
Server1
monitor: Monitor
con:Connection
<<constructor>> +Server1()
+main (args[] string);
+getFlightInfo (query: Vector): Vector
+reserveFlight(query: Vector): int
<<Class>>
Monitor
nreaders,nwriters,nrtotal,nwtoral:int
<<constructor>>+Monitor()
+start_read():
+end_read():
+start_write():
+end_write():
<< DataBase>>
flightcontrol.mdb
Server2 has the similar structure as Server1. While server2 does not have Monitor class and the corresponding database name is "salecontrol.mdb"

Figure 4 – Server1 Package Diagram

Use Cases

A typical use case is a user fills out his/her request in a GUI interface, such as

From:	*Roanoke*
To:	*Chicago*
Total Passengers:	*2*
Departing Date:	*Nov 20*
Returning Date:	*Nov 24*

Then click the "*Submit*" button to send this query to Server1. User can also conveniently clear the current query and compose a new search by pressing the "Reset" button.

Server1 acting upon the request will look up its database – flightcontrol.mdb to find corresponding available flights' information by matching the locations, dates and available seats and extract those record sets. Then it will pass them to Server2 by calling *getPrice()* function to obtain flight price information. (Notice, the only thing needed to pass here is the flight number. But for the clarity sake, we pass all the information.)

Sever2 acting upon the request from Server1 will look up its database – salecontrol.mdb to find according flight prices and return them to Server1.

Then Server1 will pack all these information and return to the client sent out the request. Client will do a combination to the result (departure flight and return flight) and display all of them to users. By examining all the available flight information and prices, users can simply click the "Buy Now" button to place the reservation.

The possible feedback messages users will receive are

1. No requested flights are available. At this point, users have to change the request, such as destination and dates to try again.

2. Reservation failed message. Possibly anther client took the last seat already to fill up the requested flight. At this point, users have to also change their request.

Summary

This project involved:

- Application of principles of Distributed Operating System such as network transparency, monitor, distributed database management

- Application of Java RMI

- Application of Java Swing Package to design user interfaces

- Application of MS Access

- Application of JDBC-ODBC

By integrating all these technology, the project gave us a good opportunity to learn how to build a practical distributed system and also a chance to work as a team.

Suggested Variations and Extensions

There is also a dilemma how far we should go and how close we want to bring the system to a real life application while battling with the time constraints and scarce recourses. As we mentioned earlier, there are other architecture options and design strategies that we can implement this system such as imbed the system in web page and eliminate the middleman and give customers all the powers; or make the user interface more complicate to match the real life situation; or recompile the database to contain more detail data (such as seat type) and formulas (to calculate the price dynamically). But certainly, by doing so, the project will ask for more developing time and exploring more technology, which is not feasible for two full time students work load and one semester duration.

But obviously, the application developed serves a good foundation for a much more complex distributed applications.

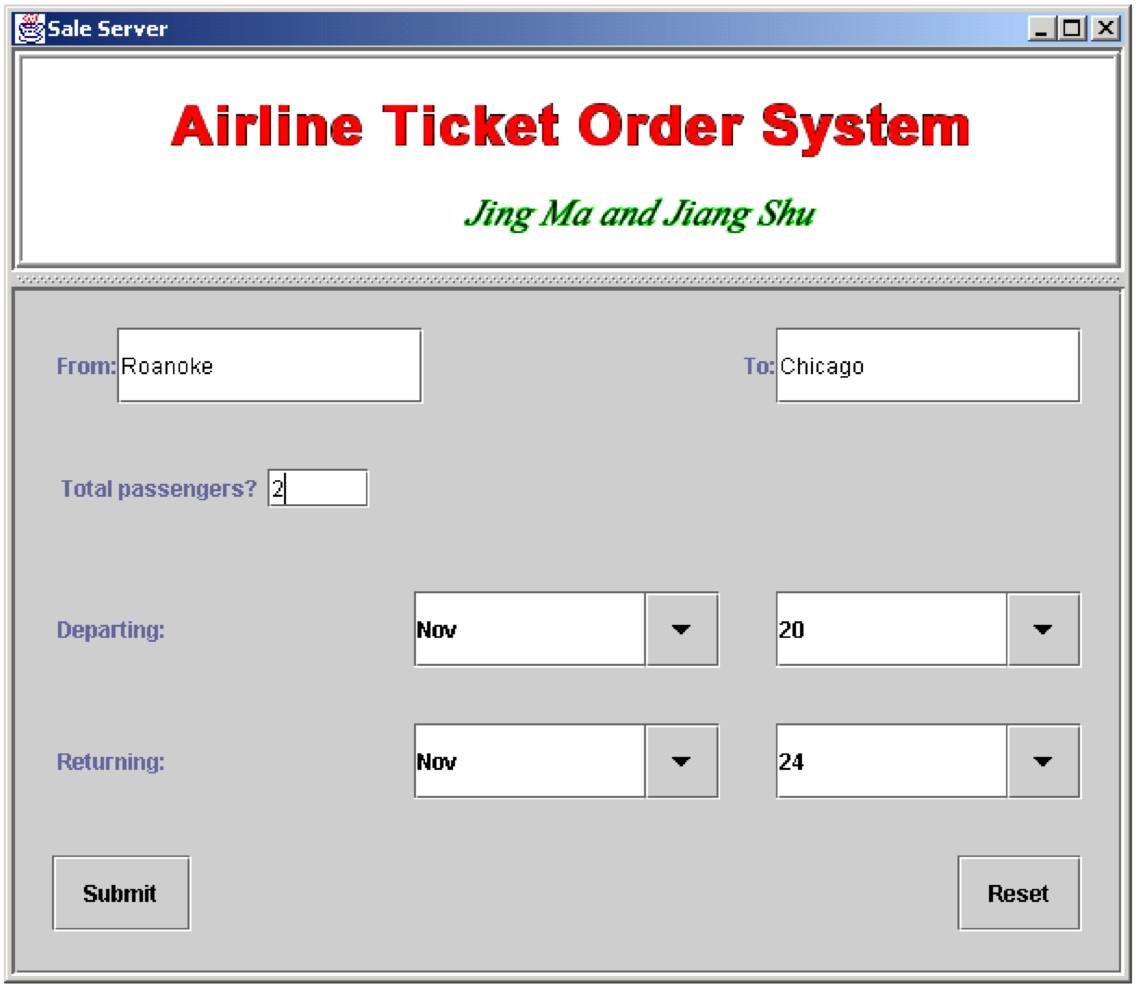
Reference

- Dr. Kafura's Lecture Notes and Class Discussions from CS5204 Operating System in fall of 2000

- JDBC Tutorial by Maydene Fisher
  http://java.sun.com/docs/books/tutorial/jdbc/index.html

- RMI Tutorial by Ann Wollrath and Jim Waldo
  http://java.sun.com/docs/books/tutorial/rmi/index.html

Screen Shots



From: Roanoke

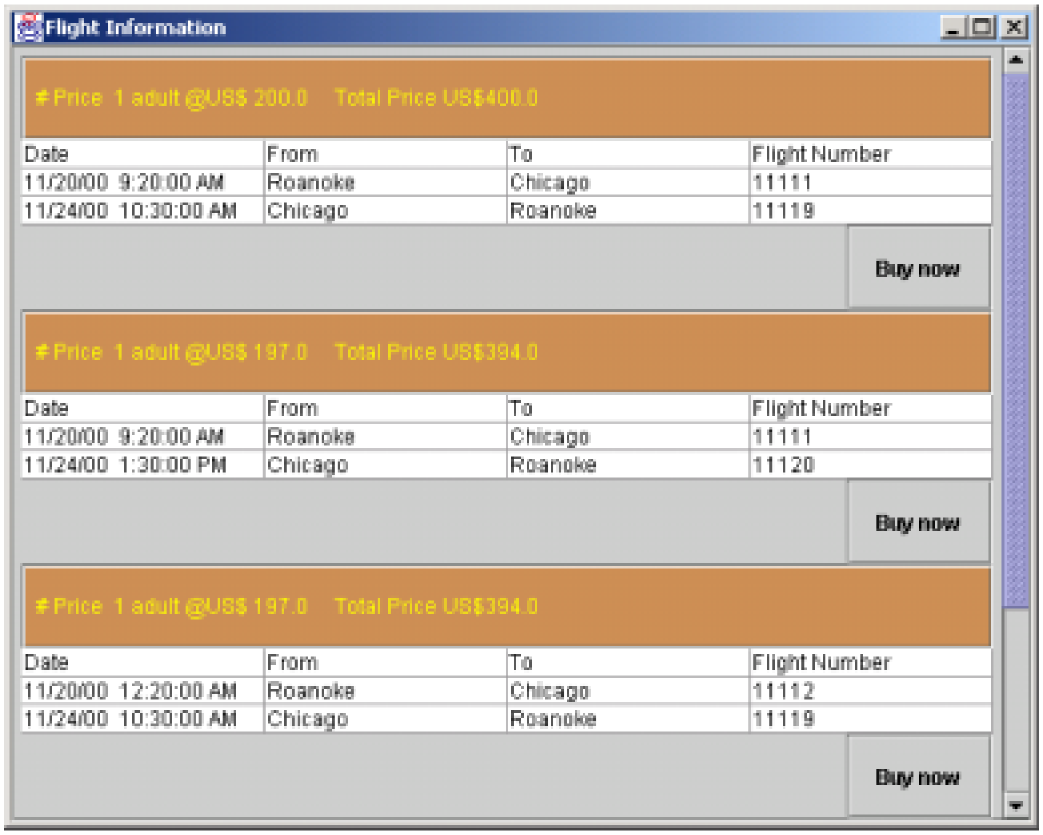
To: Chicago

Total passengers? 2

Departing: Nov 20

Returning: Nov 24

Figure 5- Flight Order Graphic User Interface



Price 1 adult @US$ 200.0 Total Price US$400.0

Date	From	To	Flight Number
11/20/00 9:20:00 AM	Roanoke	Chicago	11111
11/24/00 10:30:00 AM	Chicago	Roanoke	11119

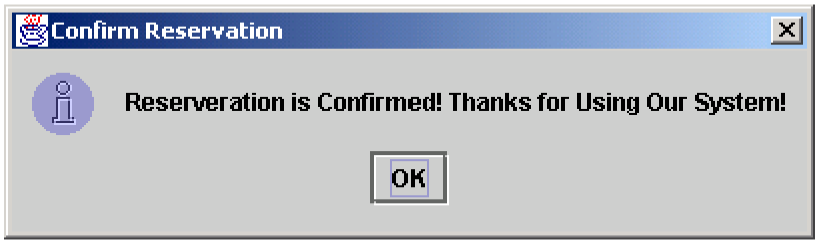
Price 1 adult @US$ 197.0 Total Price US$394.0

Date	From	To	Flight Number
11/20/00 9:20:00 AM	Roanoke	Chicago	11111
11/24/00 1:30:00 PM	Chicago	Roanoke	11120

Price 1 adult @US$ 197.0 Total Price US$394.0

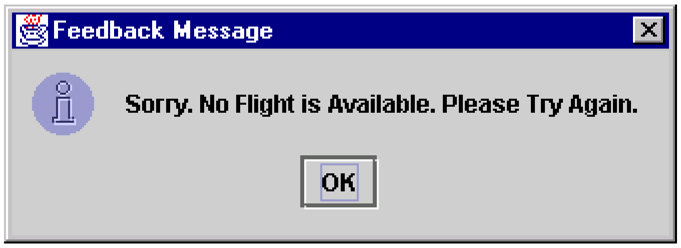
Date	From	To	Flight Number
11/20/00 12:20:00 AM	Roanoke	Chicago	11112
11/24/00 10:30:00 AM	Chicago	Roanoke	11119

Figure 6 - Flight Info Graphic User Interface



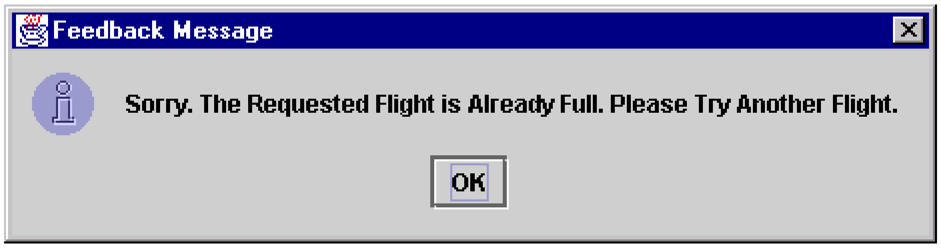
Reserveration is Confirmed! Thanks for Using Our System!

Figure 7-Reservation Confirmation Window



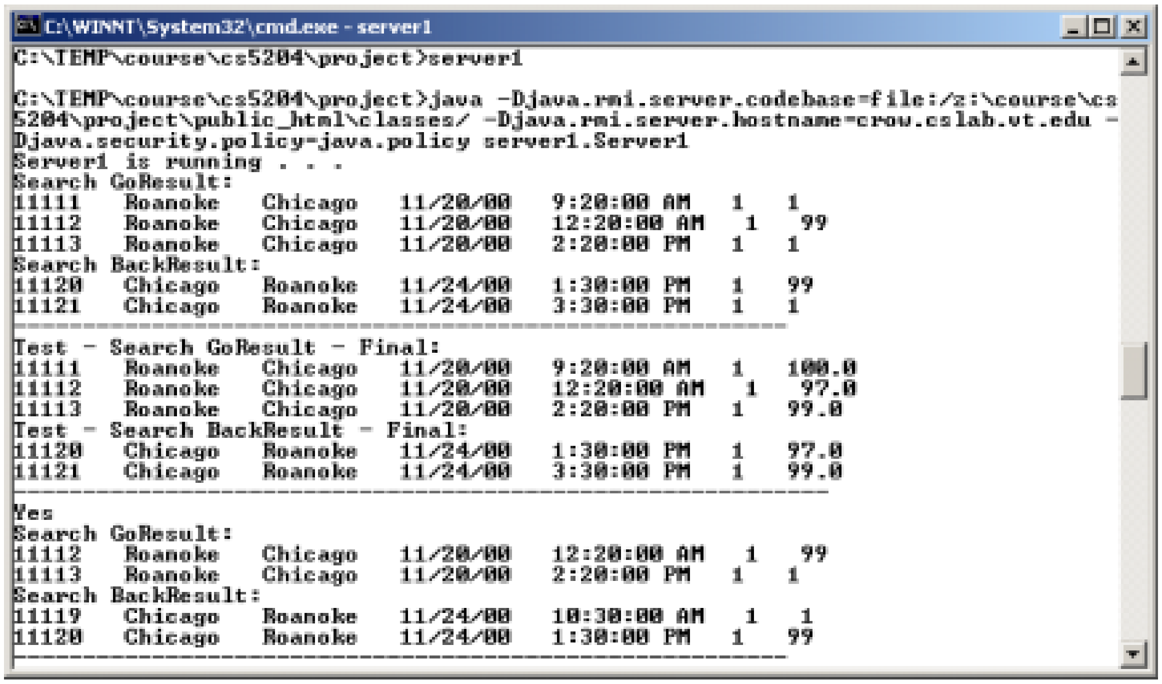
Sorry. No Flight is Available. Please Try Again.

Figure 8- Feedback Message (No Flight Available) Window



Sorry. The Requested Flight is Already Full. Please Try Another Flight.

Figure 9- Feedback Message (Requested Flight is Full) Window

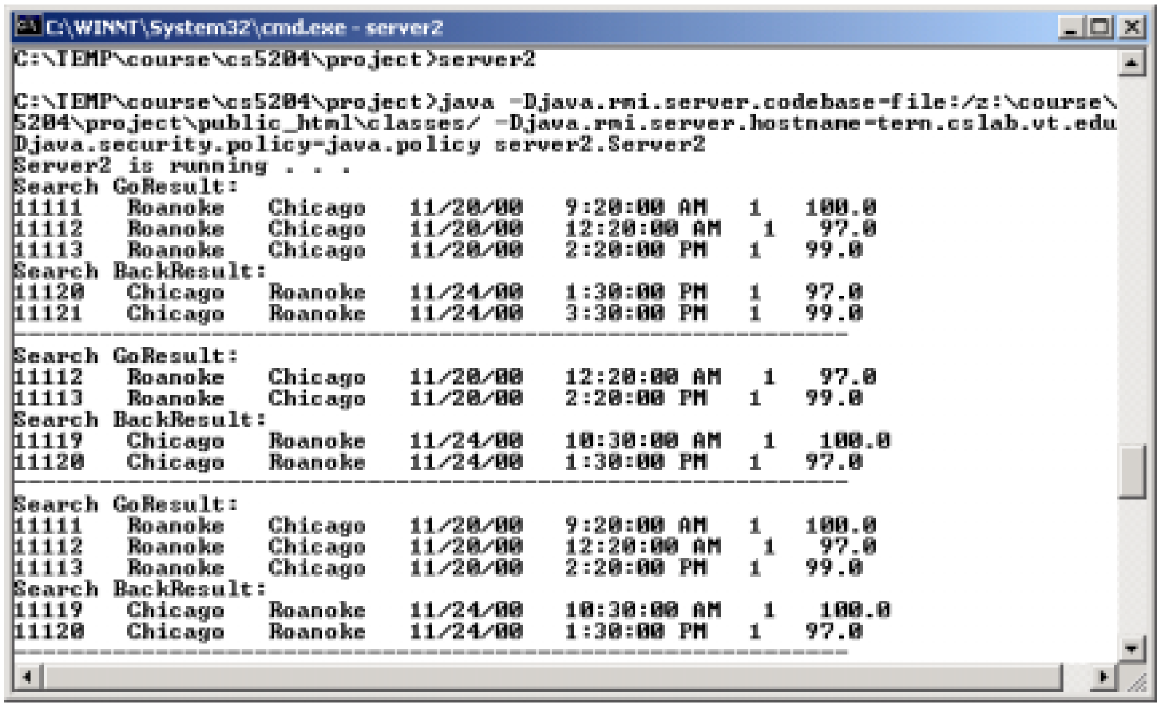


```
C:\TEMP\course\cs5204\project>server1

C:\TEMP\course\cs5204\project>java -Djava.rmi.server.codebase=file:/z:\course\cs
5204\project\public_html\classes/ -Djava.rmi.server.hostname=crow.cslab.vt.edu -
Djava.security.policy=java.policy server1.Server1
Server1 is running . . .
Search GoResult:
11111    Roanoke    Chicago    11/20/00    9:20:00 AM    1    1
11112    Roanoke    Chicago    11/20/00    12:20:00 AM    1    99
11113    Roanoke    Chicago    11/20/00    2:20:00 PM    1    1
Search BackResult:
11120    Chicago    Roanoke    11/24/00    1:30:00 PM    1    99
11121    Chicago    Roanoke    11/24/00    3:30:00 PM    1    1
========================================================
Test - Search GoResult - Final:
11111    Roanoke    Chicago    11/20/00    9:20:00 AM    1    100.0
11112    Roanoke    Chicago    11/20/00    12:20:00 AM    1    97.0
11113    Roanoke    Chicago    11/20/00    2:20:00 PM    1    99.0
Test - Search BackResult - Final:
11120    Chicago    Roanoke    11/24/00    1:30:00 PM    1    97.0
11121    Chicago    Roanoke    11/24/00    3:30:00 PM    1    99.0
-----------------------------------------------------------

Yes
Search GoResult:
11112    Roanoke    Chicago    11/20/00    12:20:00 AM    1    99
11113    Roanoke    Chicago    11/20/00    2:20:00 PM    1    1
Search BackResult:
11119    Chicago    Roanoke    11/24/00    10:30:00 AM    1    1
11120    Chicago    Roanoke    11/24/00    1:30:00 PM    1    99
--------------------------------------------------------
```

Figure 10- Screen Snapshot of Server1 output



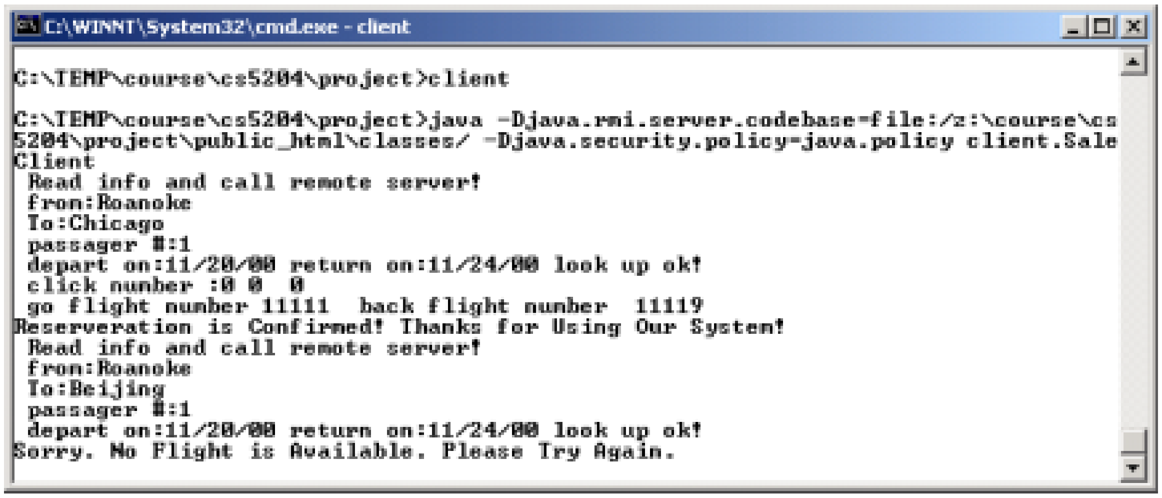
```
C:\WINNT\System32\cmd.exe - server2
C:\TEMP\course\cs5204\project>server2

C:\TEMP\course\cs5204\project>java -Djava.rmi.server.codebase=file:/z:\course\
5204\project\public_html\classes/ -Djava.rmi.server.hostname=tern.cslab.vt.edu
Djava.security.policy=java.policy server2.Server2
Server2 is running . . .
Search GoResult:
11111    Roanoke   Chicago   11/20/00   9:20:00 AM   1   100.0
11112    Roanoke   Chicago   11/20/00   12:20:00 AM   1   97.0
11113    Roanoke   Chicago   11/20/00   2:20:00 PM   1   99.0
Search BackResult:
11120    Chicago   Roanoke   11/24/00   1:30:00 PM   1   97.0
11121    Chicago   Roanoke   11/24/00   3:30:00 PM   1   99.0
------------------------------------------------------------------

Search GoResult:
11112    Roanoke   Chicago   11/20/00   12:20:00 AM   1   97.0
11113    Roanoke   Chicago   11/20/00   2:20:00 PM   1   99.0
Search BackResult:
11119    Chicago   Roanoke   11/24/00   10:30:00 AM   1   100.0
11120    Chicago   Roanoke   11/24/00   1:30:00 PM   1   97.0
------------------------------------------------------------------

Search GoResult:
11111    Roanoke   Chicago   11/20/00   9:20:00 AM   1   100.0
11112    Roanoke   Chicago   11/20/00   12:20:00 AM   1   97.0
11113    Roanoke   Chicago   11/20/00   2:20:00 PM   1   99.0
Search BackResult:
11119    Chicago   Roanoke   11/24/00   10:30:00 AM   1   100.0
11120    Chicago   Roanoke   11/24/00   1:30:00 PM   1   97.0
------------------------------------------------------------------
```

Figure 11- Screen Snapshot of Sevver2 output



```
C:\WINNT\System32\cmd.exe - client

C:\TEMP\course\cs5204\project>client

C:\TEMP\course\cs5204\project>java -Djava.rmi.server.codebase=file:/z:\course\cs
5204\project\public_html\classes/ -Djava.security.policy=java.policy client.Sale
Client
 Read info and call remote server!
 from:Roanoke
 To:Chicago
 passager #:1
 depart on:11/20/00 return on:11/24/00 look up ok!
 click number :0 0  0
 go flight number 11111  back flight number  11119
Reserveration is Confirmed! Thanks for Using Our System!
 Read info and call remote server!
 from:Roanoke
 To:Beijing
 passager #:1
 depart on:11/20/00 return on:11/24/00 look up ok!
Sorry. No Flight is Available. Please Try Again.
```

Figure 12- Screen Snapshot of Client output