COMP 6231

Distributed System Design

Fall 2016

Assignment #2

Jerome Charriere

ID: 40003247

Table of Contents

1. Implementation of transferReservation remote method.

1.1 What the method does.

This remote method takes 3 arguments: A passenger ID (integer), The city for which the passenger record is location and the city for which the passenger is to be transferred to.

This method can transfer a passenger from a certain city to another city. Note that the city for which the passenger is to be moved from may be different than the one that the manager belong to. Hence giving the ability to the manager to move a record located anywhere in the distributed system, regardless of the city server which he belongs to.

1.2 Internal overview of the remote invocation.

Upon invocation of the this remote method, the server will first check if the source location for the passenger record happens to be the one managed by itself. If so, the server will directly proceed to the transfer. Otherwise, the server will verify if the given source location is a city served by the distributed system and If it is, it will proceed to contact the city server to inform it that it should attempt the transfer.

To do that, the contact will be made through UDP. The server will send to the other distant server, where the passenger's record is assumed to be located, a UDP message containing the passenger's id and the name of the city where the passenger should be moved to.

Upon reception of this message or if the record source location is the same as the server which received the remote invocation, the system will first search the entire passenger records looking to match the provided passenger ID. If no match is found, the server will return an error, informing that the passenger record could not be found. Otherwise, the server will proceed to contact the destination server, by passing a UDP request containing passenger information. Passenger information will be modelized as the actual passenger record object. To perform this communication over the network, the object will be marshalled and unmarshalled upon reception of the message. No further actions will be carried out by the source server until reception of the response from the destination server.

The destination server for the transfer will verify that there exist a flight holding the necessary booking requirements for the new passenger. If such is the case, the passenger will be added to the passenger records and the server will inform the status of the operation to the source server. Upon reception of the reply, the source server will proceed to delete the passenger record from its records. However, if no flights were found, the server will inform the source server of the transfer failure. Upon reception of this reply, the source server will take no further action on the passenger records hence satisfying the atomicity requirements of the transfer operation.

The source server will finally inform the server which initiated the remote invocation of the status of the operation (in the case where the source server and the server which initiated the call are different, otherwise the server will return the status of the operation directly to the client).

1.3 How concurrency was handled.

Most importantly, when designing a thread-safe implementation of the transferReservation, we have to keep in mind that the transferReservation need to be able to prevent the passenger record (that is meant to be transferred) from being removed. The only possible operations that may attempt a deletion on a passenger record are:

- The deletion of the passenger's associated flight.
- An update of the number of seat that requires the removal of the passenger.

Both these operations are guarded with synchronization performed on the entire flight records. Hence, to ensure the liveness of the passenger record until the termination of the transfer operation, we only need to synchronize the flight records. With this operation, we will prevent both the removal of the passenger record from the flight as well as the removal of the passenger record from the global passenger record Hashmap.

2. Testing

The directory test2/ contains files that represent input to feed to the console. Each of those input files are stored in different folders. For each of those folders, the program will run multiple threads to read and execute the command in the files within the folder.

As we are looking to test the functionality of the transferReservation method, the testing will first, in this order, create multiple flights, book passengers on those flights and perform transfer of passenger records between servers.

The testing of the transferReservation method will include the following test cases:

- Two managers from WST will attempt to transfer the same passenger record from the MTL server to the WST server. This test case is meant to verify that the servers are able to transfer a single record with two concurrent identic requests.

- A manger from NDL will attempt to transfer a record from the NDL server to the MTL server. This test case is meant to ensure that the NDL server won't attempt to contact itself through UDP and will initiate the transfer request to MTL locally.

3. How to run

To launch the different servers, the user must execute the ServerInit.java class. This class will take care of the CORBA related initialization of the 3 different servers.

The file COMP6231_Assign2_Main.java represents the main program that enables a set of interaction with the distributed system.

If no arguments are provided upon execution, the program will run in normal mode. This implies that the user will be able to have direct interaction with one of the available servers through a command line interface.

The argument "-multitest" can be provided to test the distributed system. Tests are represented as directories containing files with console inputs to be sent. if the program is executed with this argument each of the testing directories will be processed to perform the testing. A second argument may also be defined to specify a single testing directory that we would like to perform the testing on.

Finally, the servers can be stopped by specifying the argument "-killservers".