Assignment 3 – Design Document

Valerie Hayot-Sasson  
29605436  
COMP 6231  
November 27, 2016

**Changes to Previous Implementation**

The bulk of the design remains the same between the previous CORBA implementation and current iterations of this project as no new methods have been introduced to the interface and no new functionalities were added. Changes include:

* ManagerServer.java : this is the interface file and was generated automatically by the wsimport method
* ManagerServerImpl.java : This is the implementation of the interface. This file was used to automatically generate the interface
* ManagerServerImplServiceMTL/ ManagerServerImplServiceNDL/ ManagerServerImplServiceWST : These are the client views of the WebService. There’s one file that was generated for each server
* ManagerServerPublisher: This class is responsible for starting each server and enabling UDP communication between them. It also assigns an address to each server and publishes the servers, generating the wsdl files.

**Concurrency and Synchronization**

The Synchronization is only required on the server side of the application. Possible synchronization issues include:

* Creating two passenger records with the same last name initial departing from the same city.
  + Why? If executed concurrently, they might end up trying to write the same memory block, and thus, one record will be forever lost.
  + How it was handled: Added a synchronized block to the code in which a new passenger record was added to the list.
* Passengers booking the same flight
  + Why? As the seat needs to be decremented each time a passenger books a flight, if two passengers book the same flight concurrently, the systems might book more passengers than seats available.
  + How it was handled: used the flight isInUse variable to create a lock that would block flight from modification until seats were appropriately decremented
* Deleting a flight while the flight is being modified
* Deleting the passenger reservation before transfer is successfully executed.
* Passengers booking a flight as the number of seats are getting decremented or deleted by the manager.
  + Why? The passenger might end up booking the flight after it is deleted or is out of seats.
  + How it was handled: Synchronized block around the code where flight is getting delete or seats are getting decremented and the passengers are getting removed
* Creating multiple flights departing from same location and going to the same location.
  + Why? Same problem as having two passenger records. The flight records might end up fighting for the same spot in the List
  + How it was handled: Synchronized block around the area where the action is performed.
* Passenger booking flight and departure date being modified.
  + Why? Passenger might book a flight on an undesirable day
  + How it was handled: Synchronized block around the areas where the actions are being performed.

**Test cases**In order to determine if the program is functioning correctly, it is necessary to test whether the above synchronization errors occur or if they are properly handled. See methods testCases() in ManagerClient.java for test case details.

Additional testing was done to test if every method functioned properly in a sequential environment. See method basicTestCases in ManagerClient.java for test case details.

The test cases include:

* Passenger trying to book a flight that doesn’t exist
* Manager creating a flight with one seat and two passenger attempt to book that flight
* Increase the number of seats for the given flight, book more passengers
* Halve the number of seats, ensure the passengers have correctly been removed using the getBookedFlights() method
* Create a flight on another server
* Change the date and time of the flight to match that of another server’s and transfer reservation if there is a seat available
* Delete a flight
* Book passengers to different seat classes
* GetBookedFlight() of all classes.