**RMI**

According to the project description, we made an architecture including 3 servers, a middleware, and multiple clients. Each of the mentioned servers is responsible for managing one of the flights, cars, and room resources. The middleware, which is in charge of sending each client request to its corresponding resource manager, behaves as a client for the resource managers and as a server for the clients and uses RMI protocol to do so.

* **The roadmap of the RMI part is the following:**

To implement the system with RMI, the main implemented class is RMImiddleware. In this class, there are 3 remote method stubs, flight\_RM\_stub, car\_RM\_stub, room\_RM\_stub, which are created to connect to the flight, car, and room resourcemanager, respectively. The connectServer method is responsible for connecting to the servers and obtaining a reference to the IResourceManager object. By doing so, we can call all functions related to resource management (including add, delete, query for all resources,etc). Since all resource managers need to know the customers and their reserved resources, these stubs are called while running the commands newCustomer, deleteCustomer and queryCustomerInfo.

This class also implements the functions that the a client can ask for. Besides, there is another function called **bundle** by which the client can ask for multiple reservations simultaniousely.

The method takes in several parameters: customerID (an integer), flightNumbers (a vector of strings containing flight numbers), location (a string representing the location), and two booleans indicating whether a reservation should be made for a car and a room respectively.

If car is true, it attempts to reserve a car for the customer using the car\_RM\_stub.reserveCar method. The result is stored in the carReservationResult variable, and a message indicating the car reservation result is printed. If the car reservation fails (i.e., carReservationResult is false), it returns false immediately.

If room is true, it attempts to reserve a room for the customer using the room\_RM\_stub.reserveRoom method. The result is stored in the roomReservationResult variable, and a message indicating the room reservation result is printed. If the room reservation fails, it checks if car is true to determine if a car reservation was made. If so, it cancels the car reservation using the car\_RM\_stub.UndoreserveCar method and prints a message indicating the cancellation. Then it returns false immediately.

If both the car and room reservations are successful, it enters a loop that iterates over the flight numbers in the flightNumbers vector. For each flight number, it attempts to reserve a flight using the flight\_RM\_stub.reserveFlight method. The result is stored in the flightReservationResult variable. If the flight reservation fails, it cancels the car and room reservations (if they were made) and any previously reserved flights. The cancellations are performed using the corresponding Undoreserve methods from the respective stubs. Messages indicating the cancellations are printed.

Finally, if all reservations (car, room, and flights) are successful, it returns true.

The main method expects command-line arguments containing three host names which are the resource managers. If the number of command-line arguments is less than three, an error message is printed, indicating the correct usage of the program, and the program exits.

An instance of the RMIMiddleware class is created, passing the three host names to the constructor. The RMIMiddleware object is exported as a remote object, creating a stub (client proxy) using the UnicastRemoteObject.exportObject method.

A registry is created or obtained on port 4010 using the LocateRegistry.createRegistry or LocateRegistry.getRegistry methods, respectively. The stub is bound to the registry using the Registry.rebind method. A shutdown hook is added to handle unbinding the stub from the registry when the program is terminated. If an exception occurs during the setup process, an error message is printed, the exception is printed, and the program exits. If everything is successful, a message is printed indicating that the resource manager server is ready and bound to the specified RMI name.

**Test**

In Senario, 2 flights, 2 customers, and some cars and rooms are added. Then a bundle reservation is requested including a wrong location and the client receives ‘Bundle could not be reserved’. In the following another correct bundle request is entered and done successfully. At last the number of flights, rooms and cars is queried showing that the bundle function either reserves all resources or non of them (if it cannot reserve one of the resources it roles back the reserved ones.) It also shows that the bundle function can reserve just one of the resources car and room if the customer prefers so. The last two commands show that the code can handle the error if the command or the number of its arguments are wrong.

Then following paper includes the the commands. (The order is from left to right)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AddFlight,14,1,100  Adding a new flight  -Flight Number: 14  -Flight Seats: 1  -Flight Price: 100  Flight added | AddFlight,15,2,100  Adding a new flight  -Flight Number: 15  -Flight Seats: 2  -Flight Price: 100  Flight added | AddCars,Montreal,4,40  Adding new cars  -Car Location: Montreal  -Number of Cars: 4  -Car Price: 40  Cars added | AddRooms, Montreal,4,50  Adding new rooms  -Room Location: Montreal  -Number of Rooms: 4  -Room Price: 50  Rooms added | AddCustomerID,3  Adding a new customer  -Customer ID: 3  Add customer ID: 3 |
| AddCustomerID,4  Adding a new customer  -Customer ID: 4  Add customer ID: 4 | Bundle,3,14,15,Mon,0,1  Reserving an bundle  -Customer ID: 3  -Flight Number: 14  -Flight Number: 15  -Location for Car/Room: Mon  -Book Car: 0  -Book Room: 1  Bundle could not be reserved | Bundle,3,14,15,Montreal,0,1  Reserving an bundle  -Customer ID: 3  -Flight Number: 14  -Flight Number: 15  -Location for Car/Room: Montreal  -Book Car: 0  -Book Room: 1  Bundle Reserved | QueryFlight,15  Querying a flight  -Flight Number: 15  Number of seats available: 1 | QueryFlight,14  Querying a flight  -Flight Number: 14  Number of seats available: 0 |
| QueryRooms, Montreal  Querying rooms location  -Room Location: Montreal  Number of rooms at this location: 3 | QueryCars,Montreal  Querying cars location  -Car Location: Montreal  Number of cars at this location: 4 | AddRooms, Toronto,3  **Command exception:** Invalid number of arguments. Expected 3, received 2. Location "help,<CommandName>" to check usage of this command | QueryRoo,Toronto  **Command exception:** Command QueryRoo not found |  |