Distributed Systems (H0N08A)

Report: Java RMI

Filipe Borralho *(r0825061)*
Razvan Roatis *(r0825322)*

October 30, 2020

Question 1 When a client starts a booking the first step is always creating a reservation session. Then the client can check which car types are available within a specified start and end date, this is done using the method checkForAvailableCarTypes. Checking the available car types isn't necessary if the client already knows which car type he pretends. Afterwards, the client will create as many quotes has he wants for each car type he wants. The final step is confirming the quotes, though here there are two possible outcomes:

- Confirmation succeeds
- Confirmation fails

Case the confirmation succeeds then the client can just close is reservation session. If the confirmation fails it means one of the quotes wasn't successfully turned into a reservation. In this case, the client will be notified and either can try again with other car type or give up and close his session.
Check figure 1.

Question 2 Some objects need to be sent through the network from the server to the client, so the client can use these. That way all classes that we want accessible by the client must implement Serializable. This will convert the state of an object into a byte stream, which then can be sent over the network to any other machine. Below we have an use case example:

```
1   import java.rmi.Remote;
2
3   interface IRemoteServer extends Remote {
4       Car sendCar(String id) throws RemoteException;
5   }
6
7   import java.io.Serializable;
8
9   class Car implements Serializable {
10      private String id;
11      private String brand;
12      private int power;
13
14      // Constructor, getters and setters ...
15  }
16
```

If the client remotely calls the IRemoteServer and wants to retrieve a Car via the sendCar(String id) method, then the Car class must implement Serializable in order for the server to be able to send it via the network.

Question 3 A remote object is an object whose method can be invoked from another JVM (i.e. remotely accessible). Classes need to be remotely accessible whenever the client needs to interact with it directly. We illustrate via the following example:

```
1   import java.rmi.Remote;
2
3   interface IRemoteServer extends Remote {
4       Car sendCar(String id) throws RemoteException;
5   }
6
```

Here, IRemoteServer provides the signatures of the methods which can be invoked remotely. This means that in our project IAgency needs to extend Remote so the client can utilize all available methods.

Question 4 When the manager asks for the number of reservations of a specific renter, the manager has to provide his manager session, this way only a manager is able to use this method, and the renter's name. After the manager calls numberOfReservationsByRenter, the server will iterate through all car rental companies he has a contract with and retrieve the number of reservations the client made. The server will then return to the client the sum of all reservations.

Question 5 On a fully remote and distributed setup the car rental companies would be running on different servers and would have to upload their data to the car rental agency then the agency would interact with the client which one in their one machine. So Hertz and Dockx are both CarRentalCompany but running separately from each other and from the Client and the RentalServer. We can see this in figure 2

Question 6 We implemented the naming service by creating a map with the name of the companies and their respective ICarRentalCompany on the agency side. The client then interacts only with the agency, if a manager wants to unregister or register a company on the server he just has to call the specific methods and then the agency will clear that company's data from the map in case of unregister or add it in case of register. We decided to take this approach because this way the client only interacts with the agency and never with the car rental companies but also because this allows the manager to easily add and remove companies from the agency's "database".

Question 7 There are 2 types of session, the manager session and the reservation session. The manager session is first created when the client uses the method getManagerSession, this method creates a manager session with the name of the manager and then the server stores it in a set of manager sessions. When the manager is about to leave, he closes his session using the method endManagerSession which basically will remove his session from the set. The reservation session works in a pretty similar way, it's first created by using the method getReservationSession and it's stored in a map by the server. We opted for a map instead of a set because of this way whenever the client adds a quote to his session that information is also uploaded to the map. To clean the session the process is the same as the manager, using the method endReservationSession.

This approach allows the server to only store the active sessions and to easily add or remove sessions so it seems appropriate since this way the only computational concerns the server will be related with active users.

Question 8 A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread. The RMI runtime makes no guarantees concerning mapping remote object invocations to threads. Since remote method invocation on the same remote object may execute concurrently, the application is not thread-safe by default. In order to achieve thread-safety, all of the methods inside RentalAgency class have been defined using the keyword synchronized. Therefore, all methods are synchronized to the object and only one thread can execute a function at a time.

Question 9 By defining all the methods inside RentalAgency with the keyword synchronized the server will act as a completely serial one (i.e. there will not be any parallelization whatsoever).

Therefore, if the number of clients would increase drastically, the waiting time for each one will increase because they would have to wait for one another. Yes, synchronization implemented in this manner represents a serious bottleneck.

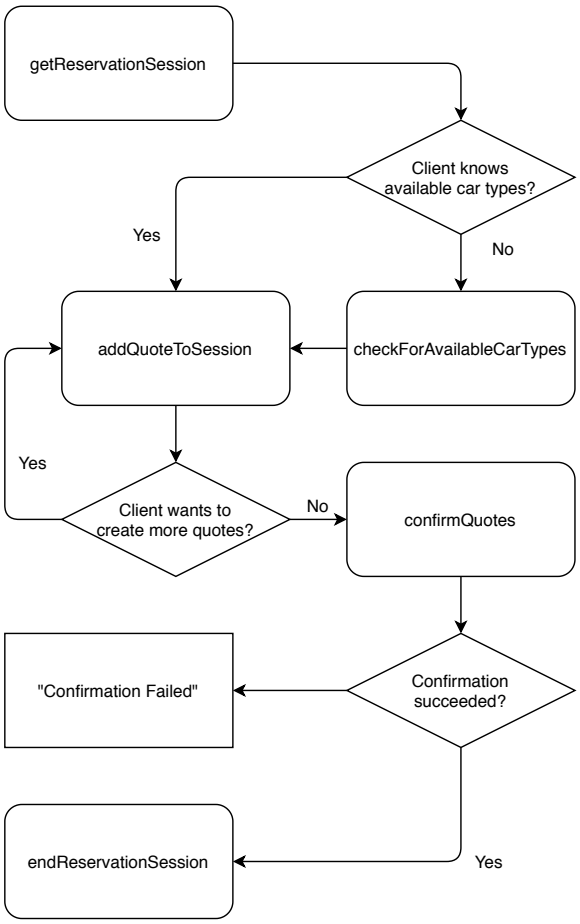

Figure 1: Full Booking Cycle

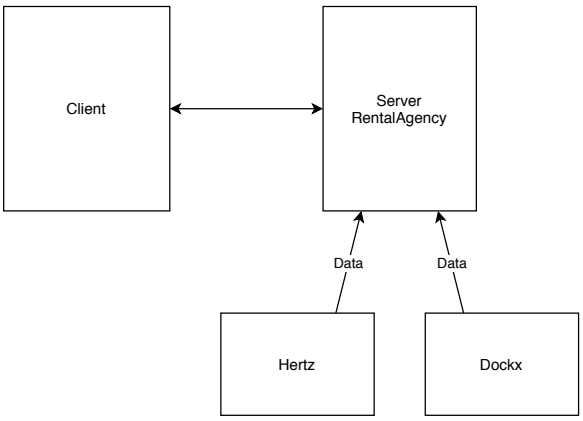

Figure 2: Component/Deployment Diagram