Distributed Systems: Java RMI report

Tobias De Locht, Dries De Backker

3 November, 2017

1 Design decisions

The set up of this project consist of 2 packages, the client and the rental server. Figure 1 shows the class diagram of the project. For real life application a web server is needed so the client can download the necessary implementation classes. This deployment is visualized in figure 4.
When starting up the rentalserver, it's main method creates a new namingService object. This object is mainly responsible for registering car rental companies and looking up their objects by name. The main method also creates a new remotely accessible sessionManager object. This object is in charge of creating new sessions for clients as well as managers. So two type of session classes are available with each it's corresponding remote interface class. Figure 2 shows a sequence diagram for a session creation by the client.
The manager sessions are stateless while the reservation sessions are stateful. The reservation sessions are used by clients to check car availabilities and make or cancel reservations. The responsibilities of the manager is registering companies and obtaining statistics about the rental companies. The sequence diagram for making a reservation is shown in figure 3.
For the session objects no life cycle management was implemented. This means the session for a specific client or manager is not removed after he closes it's application. For future use it is best to implement a life cycle management for these sessions.

2 Remote objects

There are 3 remote classes, each with there corresponding remote interface. The sessionManager, reservationSession and mangerSession classes. These objects are remote because the client only needs to call methods from these objects. The sessionManager object is stored in the RMI registry, the two session objects are not. These objects are stored in the sessionManager object because they are made on demand of the client. All the remote objects are stored on the rental server because their implementation requires data and methods from this server.
Because some objects need to be returned over the remote connection, these

classes need to be serialized. For example, the reservation class is a return value in some of the client's methods and needs to be serialized. The class diagram shows wich of these classes are serializable.

3 Synchronization

With the current deployment at use, it is possible multiple clients are concurrently sending method requests to the server. These request are then executed in interleaving threads. This way some data can become inconsisten. To solve this problem some methods at the server are defined as synchronized. This will make it impossible for these methods to interleave their thread executions and execute sequential instead. The methods to made synchronized are in the carRentalCompany class and are the ones that can be requested directly by one of the sessions objects.

4 Diagrams

Figure 1: Class Diagram

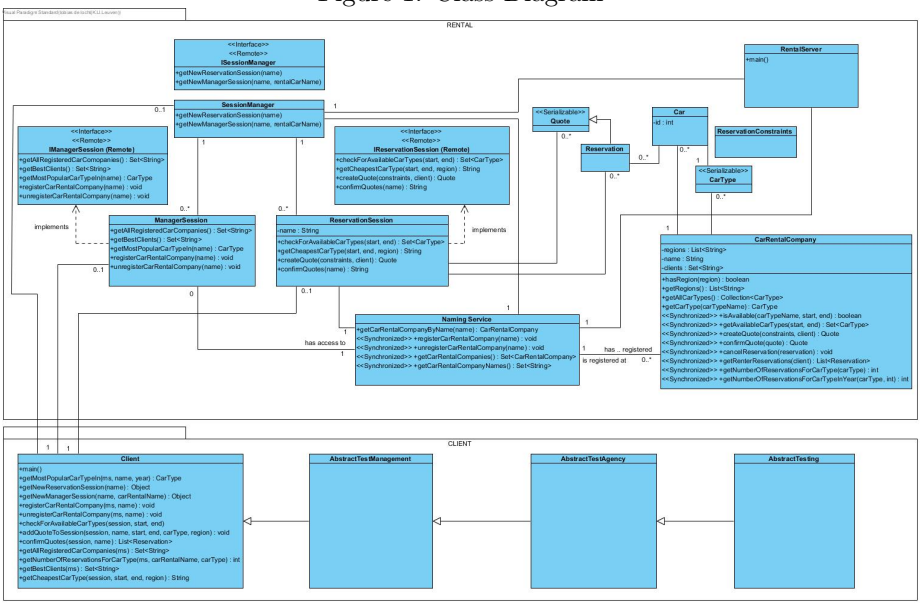

Figure 2: Client session creation

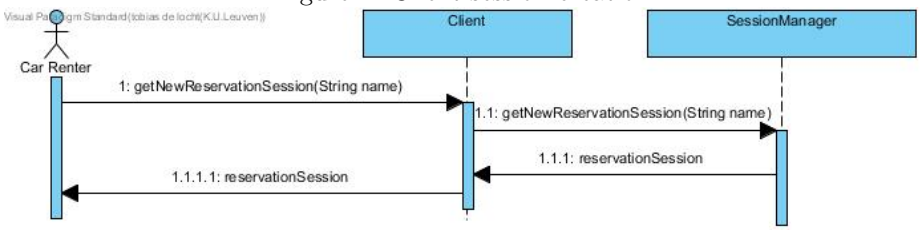

Figure 3: Reservation

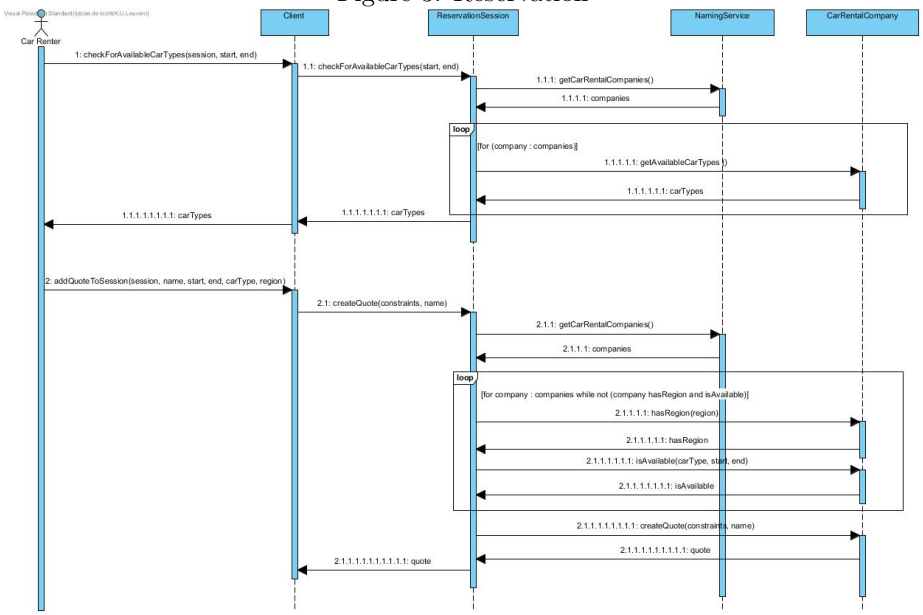

Figure 4: Deployment