Distributed Systems (H0N08A)

Report: JAVA RMI

Robin Geelen *(r0664431)*
Jochem Hoes *(r066420)*

October 29, 2020

How would a client complete one full cycle of the booking process, for both a successful and failed case? Figure 1 shows the function calls to the different parties and their responses for the example scenario.

When do classes need to be serializable? You may illustrate this with an example class. A class needs to be serializable if its objects need to be transmitted to an application running on a different machine. An example is the `Quote` class. When the client wants to make a reservation, a `Quote` object is created at a car rental company. Then this object is sent to the agency and eventually to the client.

When do classes need to be remotely accessible (Remote)? You may illustrate this with an example class. A class needs to be remote if its objects are located at the server, but the client needs to be able to invoke methods on it. An example is the `RentalAgency` class. When the client wants to print the available car types or create a reservation, he must be able to call certain methods on a rental agency which is located on a different machine.

What data has to be transmitted between client and server and back when requesting the number of reservations of a specific renter? At the method call, the name of the renter is transmitted to the server. Then the server will check the number of reservations for this renter and will transmit this number back to the client when the method returns.

What is the reasoning behind your distribution of remote objects over hosts? For our implementation, we chose to put as much functionality as possible on the servers of the car rental agency and the car rental companies. We kept all classes related to the cars and reservations on the car rental companies. The classes that aggregate this information for the client run on the rental agency. The client's class is rather simple and just performs the needed function calls on the car rental agency. Figure 2 shows how classes are divided over the client and server applications.

How have you implemented the naming service, and what role does the built-in RMI registry play? Why did you take this approach? The Java RMI register plays a major role as a naming service. The RMI registry itself is in essence a naming service. We used the following technique: when a car rental company comes online, it registers itself to the agency's RMI registry. When the manager of the agency decides they have valid agreement with the company, he can use the information from the registry to add the company to the agency. This technique has its limitations. For example, right now still everyone can simply register himself to the agency's RMI registry. This could result in security issues: a malicious party registering himself with another name will deny the service to the real car rental company. Another problem arises when the server of the car rental company crashes: then the value in the RMI registry is not longer valid, and the rental agency needs to add the company again when it has recovered.

Which approach did you take to achieve life cycle management of sessions? For the life cycle management of the sessions we chose a rather simple approach. Every time a client requests a session, a new session is created and the object is stored on the client side. We chose this approach over e.g. storing the sessions on the server, because of the simplicity of our sessions. They don't hold a lot of persistent information, and also don't need any authentication to access them. One disadvantage of our approach is that a client may need to recreate the quotes after a crash. A list of pending quotes is stored in the session, so when the client crashes, this information is lost. However, we think this disadvantage does not surpass the simplicity of our solution: no need for managing the session objects at the server side.

Why is a Java RMI application not thread-safe by default? How does your application of synchronization achieve thread-safety? Java RMI handles each remote method call, e.g. from two different clients, in a separate thread. The concurrency control problem is located at the set of reservations. If multiple threads access the set of reservations for a specific car simultaneously, inconsistencies can occur. Therefore, each thread that wants to access the set of reservations must synchronize on the car rental company until its transaction is complete. Note that it is not a good solution to just synchronize on a car when its set of reservations is used. After all there are transactions, like `getAvailableCars`, that use the set of reservations of multiple cars, and these accesses must be done as a whole.

How does your solution to concurrency control affect the scalability of your design? Could synchronization become a bottleneck? We think that synchronization will not become a bottleneck for a car rental company. All methods have low computational cost and the number of reservations for a car rental company is not that high in reality.

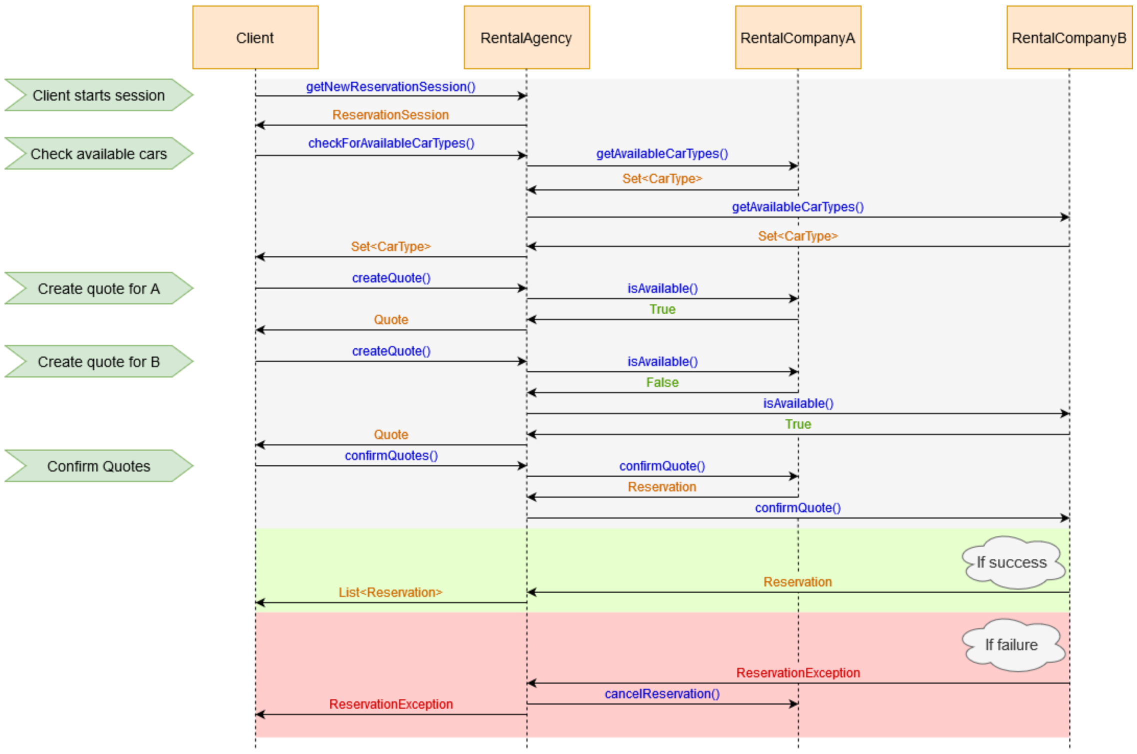

Figure 1: Sequence drawing of the two example scenarios of a full booking cycle. For clarity, the lookup queries to the RMI registry are not shown.

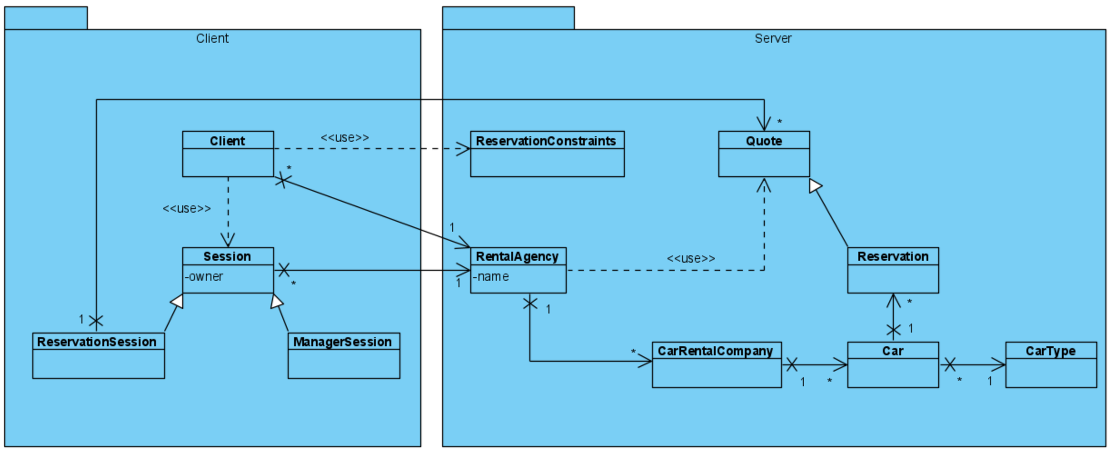

Figure 2: Diagram of the classes that were implemented on client and server side. These groups are not very strict, as there are also classes that are used by both the client and server applications.