Distributed Systems – RMI 2&3

Fall 2018
Bart Claessens, Ondrej Sotolar

Design Overview

The application is split into two parts: Client and Rental. The Client represents the user application: this can be a rental company, or a user. The Client app uses Java RMI to communicate with the Rental app. The state between the Client and Rental parts is persisted in a session that's created for each Client instance. The sessions are managed by *SessionManager* which is bound to the RMI registry at startup. The client can look up the SessionManager to create either a *RentalSession* or a *ManagerSession*. When asked to create a new session, the *SessionManager* will create a new instance, bind it to the RMI registry and then return a reference to the Client. The Sessions are Remote classes, they represent the user on the server side and give the user an API.

The physical rental companies are stored in a single *RentalServer* that contains a list of all the CarRentalCompanies. Each *CarRentalCompany* contains all its cars and every car has all the final reservations made for it. Quotes are stored in the Session of the user until they are finalized into a reservation. When finalized they will be stored as reservation at the respective car.

Design decisions

Which classes are remotely accessible and why?

SessionManager: first contact point for the Client, the rmi name of the SessionManager is stored in the Client so it can be looked up when the Client first starts.
RentalSession: Client uses it for making reservations
ManagerSession: Manager uses it for getting statistics

Which classes are serializable and why?

CarType, Quote, Reservation, ReservationConstraints

- the classes are sent through the RMI channel, and therefore need to marshalled and unmarshalled

Which remote objects are located at the same host (or not) and why?

SessionManager, RentalSession, ManagerSession

- all the business logic is executed on the Rental server, Client uses it only through an API defined by these classes

Which remote objects are registered via the built-in RMI registry (or not) and why?

SessionManager, RentalSession, ManagerSession. These are the only classes that need to be accessible to the Client.

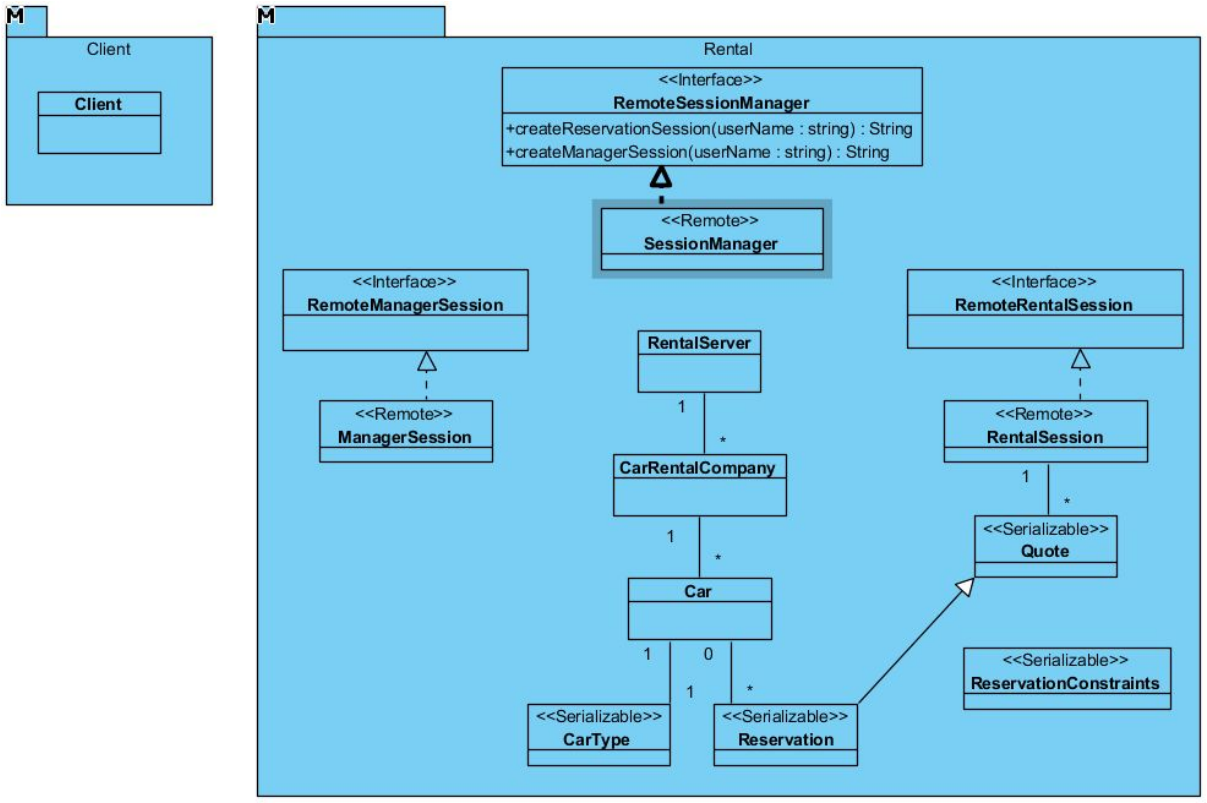
Briefly explain the approach you applied to achieve life cycle management of sessions.

SessionManager class manages the creation of Client and Manager sessions. It's accessible through the RMI registry to the user apps.

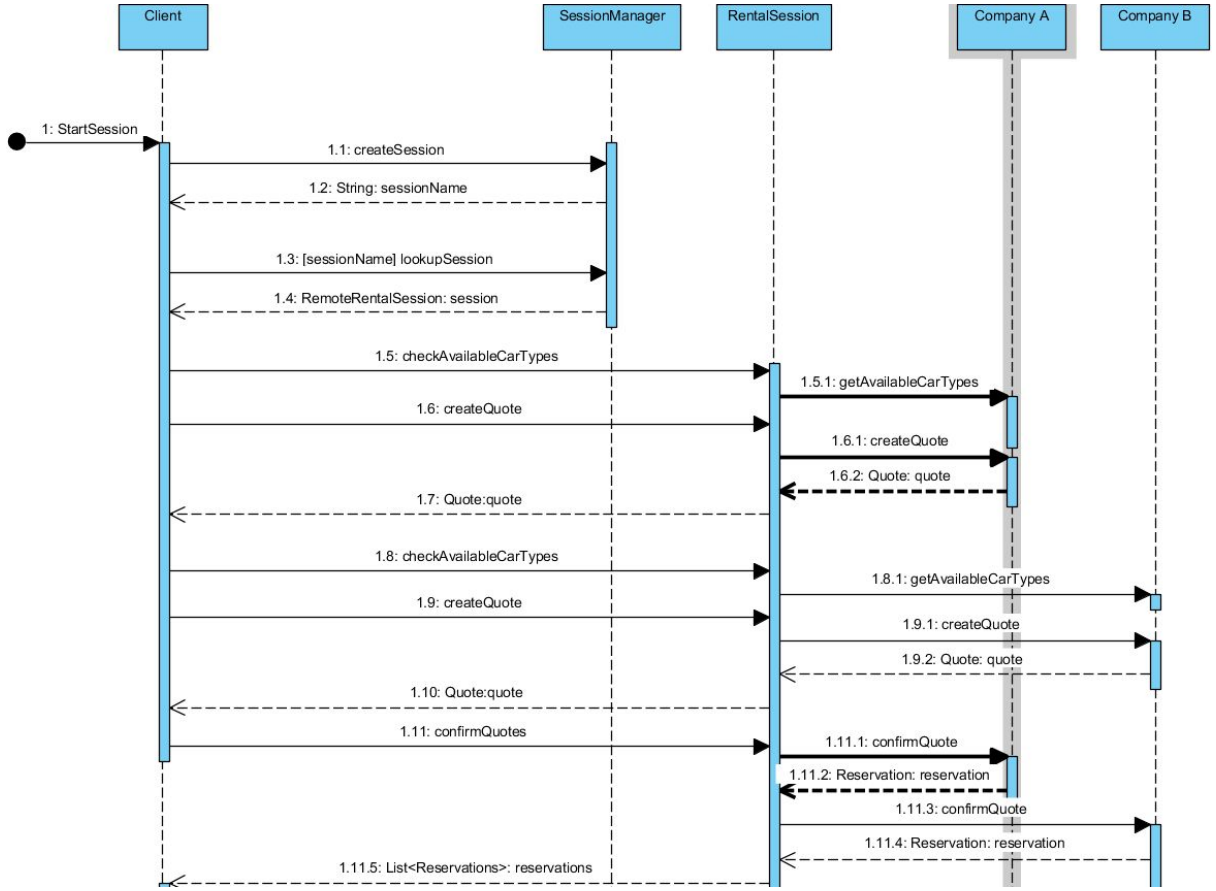
At which places is synchronization necessary to achieve thread-safety? Will those places become a bottleneck by applying synchronization?

We implemented synchronization in the *CarRentalCompany.confirmQuote()* method and added a rollback to the *RentalSession.confirmQuotes()* method in the case that one quote finalization fails. Because we only applied synchronization on the single *confirmQuote()* method, this won't become a bottleneck.

Class diagram

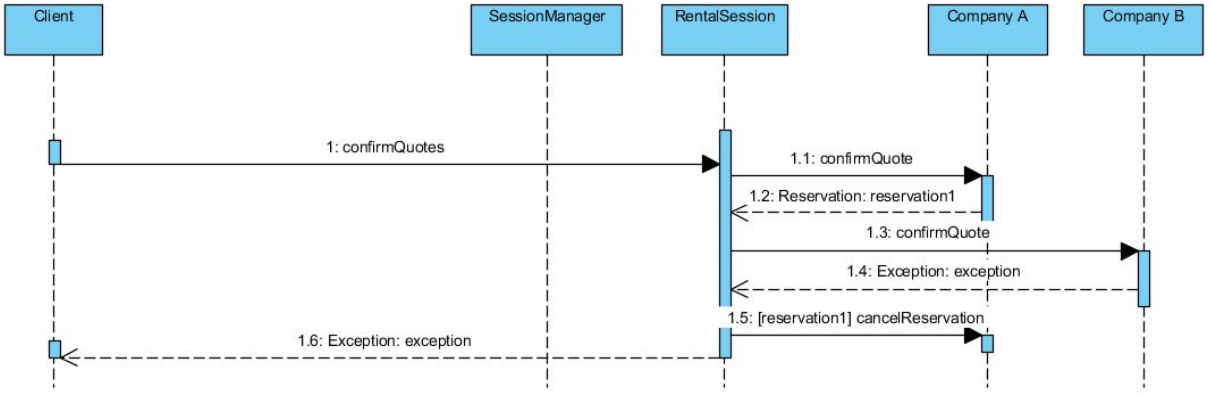


Sequence diagrams



scenario 1-2-3A-4A

Alternative:



scenario 3B-4B

Deployment diagram