Distributed Systems (H0N08A)

# *Report:* Java RMI

Bruno Vandekerkhove *(s0216676)*
Afraz Salim *(r0439731)*

November 1, 2019

**Question 1** We start with a server that is set up by binding an agency and companies to the RMI registry. At the client side, an interface of the agency is acquired by doing a lookup on the registry. A manager registers the companies, and any client is then free to create sessions and book rental cars. Typical scenarios for subsequent booking sessions are displayed in the diagrams at the end of the report (figure 3, 4 and 5). A reservation only fails if the car type is not available.

**Question 2** Classes need to be made serializable when they need to be transferred *'by value'*. Their objects can be marshalled. The `ReservationConstraints` is one example ; it's used by clients to define constraints for the creation of a quote. When a client creates a quote, he passes an instance of this class as a parameter. The instance gets transferred over the network as-is, meaning that a whole copy of the object is brought to its destination. If the transferred object's state is altered, nothing happens with the original object.

**Question 3** When a user needs to alter the state of some remote object, then a server can provide a reference to that object by providing a remote interface. This allows the user to invoke methods of that object as if it were running on the same virtual machine. The car rental agency (`CarRentalAgency`) is an example ; it allows clients to create sessions (which are remote objects as well). When a client uses the agency, a reference is passed. When clients call methods on the reference (the stub), the state of the original agency may be altered.

**Question 4** Only integers and strings are passed around (such as the client's name or the number of reservations). Unless something went wrong ; in that case an exception may be sent over the network.

**Question 5** The centralised agency runs on some server. Clients (renters and managers) are running on other nodes. Each company run on its own server (though companies may share a server). Because of this, a company can run from anywhere, and a centralised agency makes it easy for clients to interact with any of them.

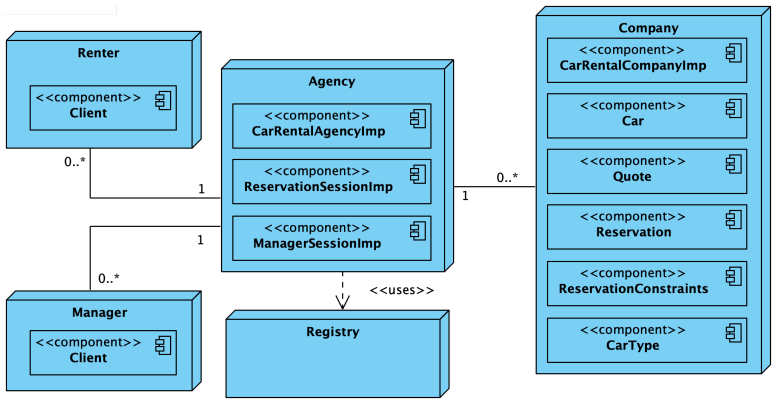



Figure 1: Deployment diagram.

**Question 6** We relied on the RMI registry itself since it is a naming service. There was no need to create a custom component.

**Question 7** All sessions are kept by the central agency (`CarRentalAgencyImp`). Any renter or manager may close his/her own session manually. We tried periodically removing *all sessions that haven't been in use for a while* (say, 30 minutes). Unfortunately the security manager prevents access to the `java.lang.reflect.Proxy` class so it didn't work remotely.

**Question 8** It's not a feature of Java RMI. The responsibility is delegated to the developer. So if instances are to be run concurrently, it's him who has to make sure no race conditions arise. The `synchronized` keyword (used in the `confirmQuote` method of the car rental companies) ensures that no two renters can confirm conflicting reservations at the same time. Multiple `confirmQuote` calls will not be interleaved..

**Question 9** If a rental company gets a lot of requests, then there may definitely be delays and the response speed may decrease substantially. Our strategy simply prevents interleaving at the level of the method, which could be improved upon by doing it at a lower level of granularity.

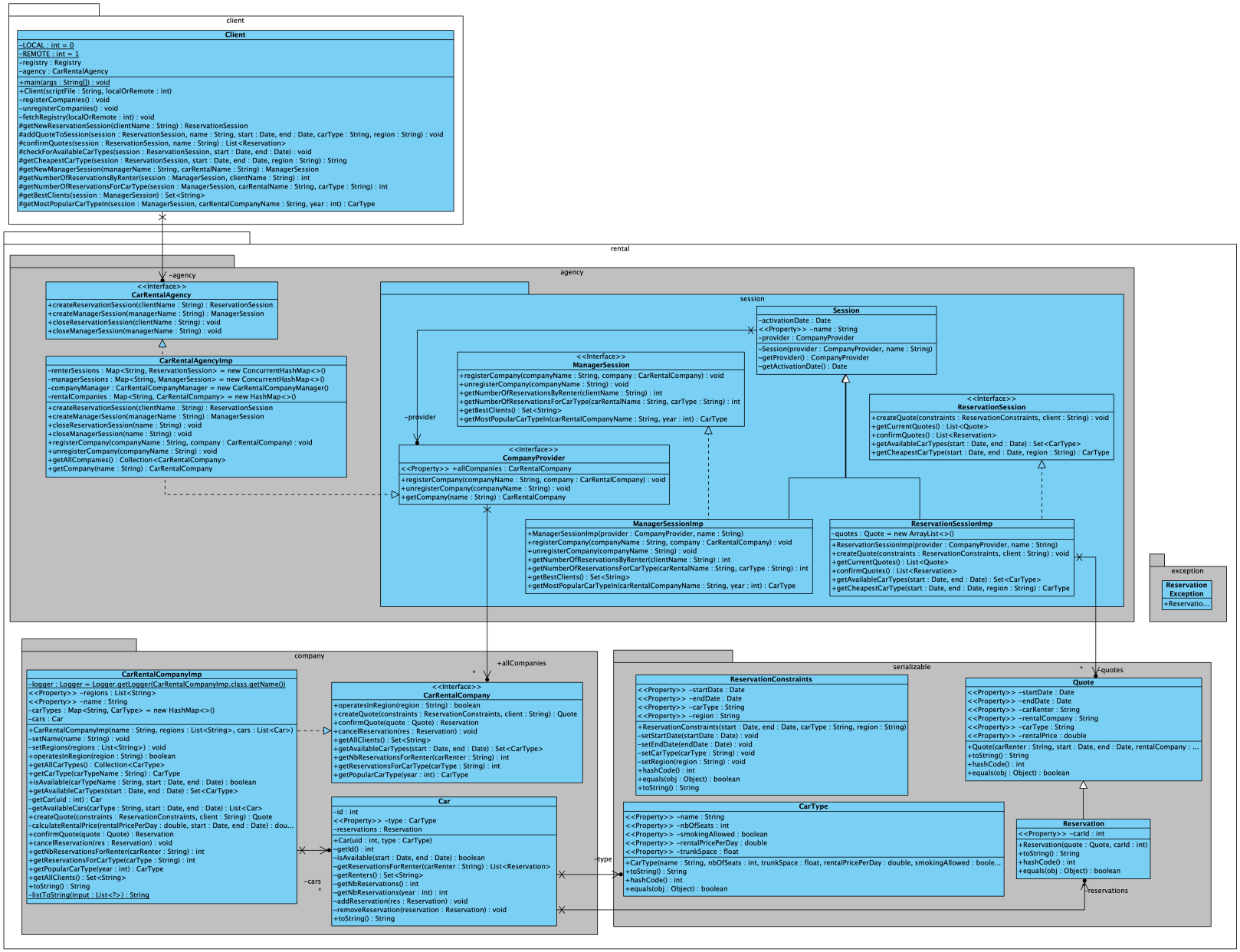
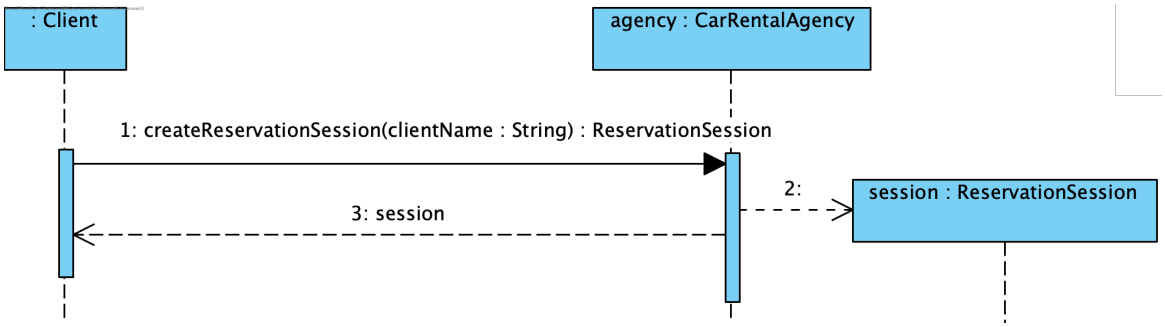
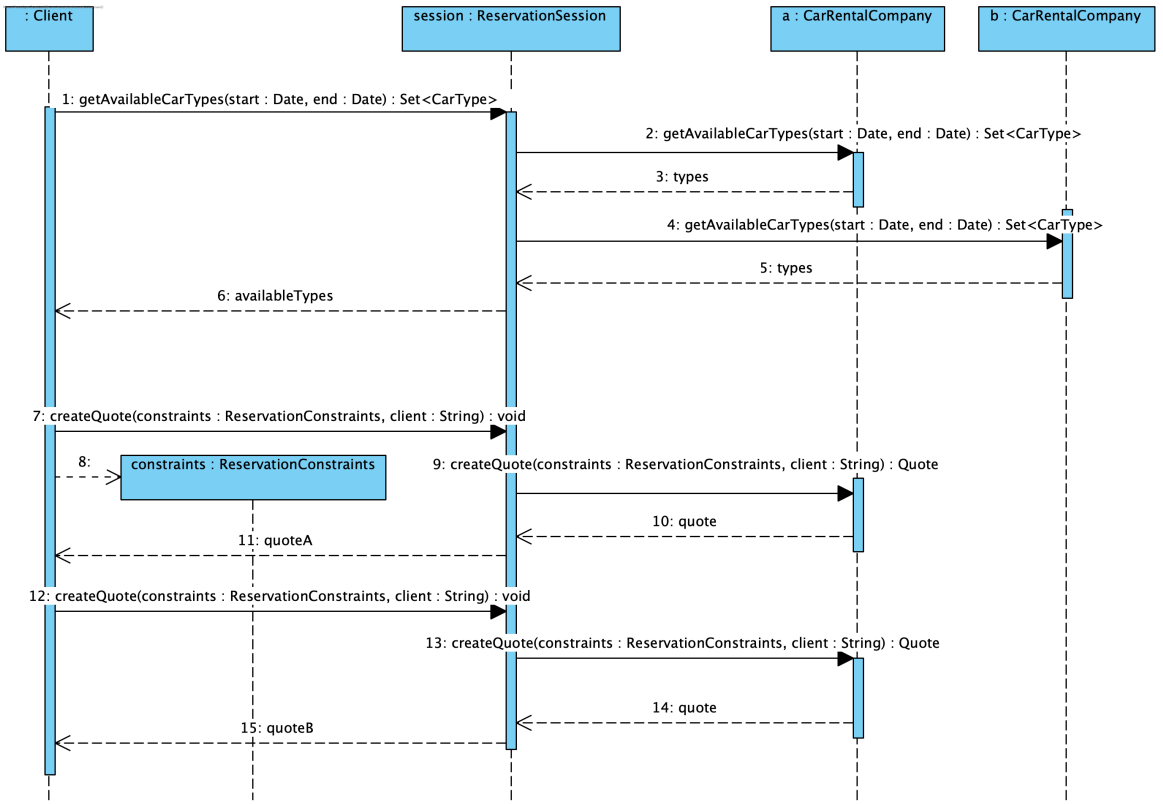



Figure 2: Class diagram for the client & server.

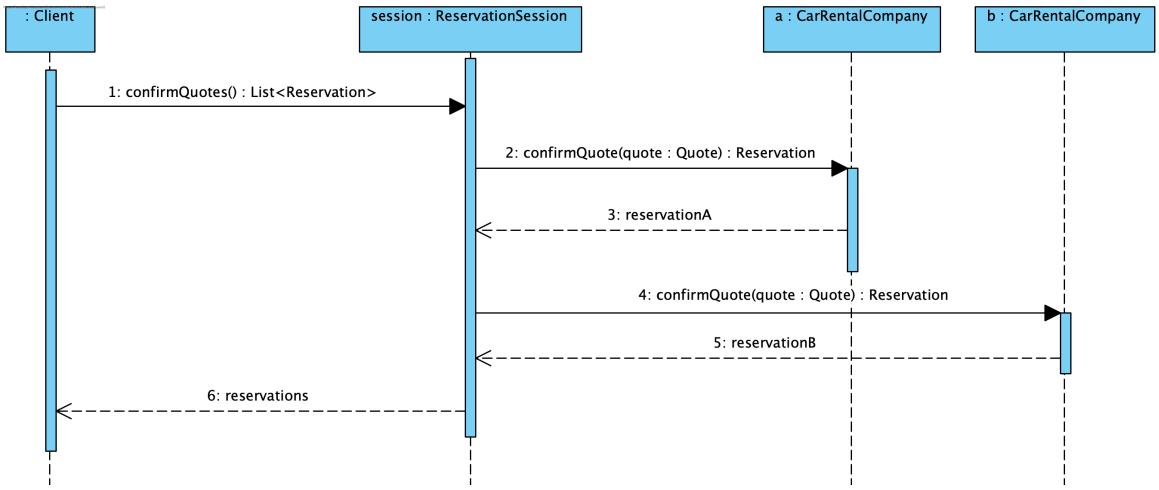



(a) The client creates a new session.

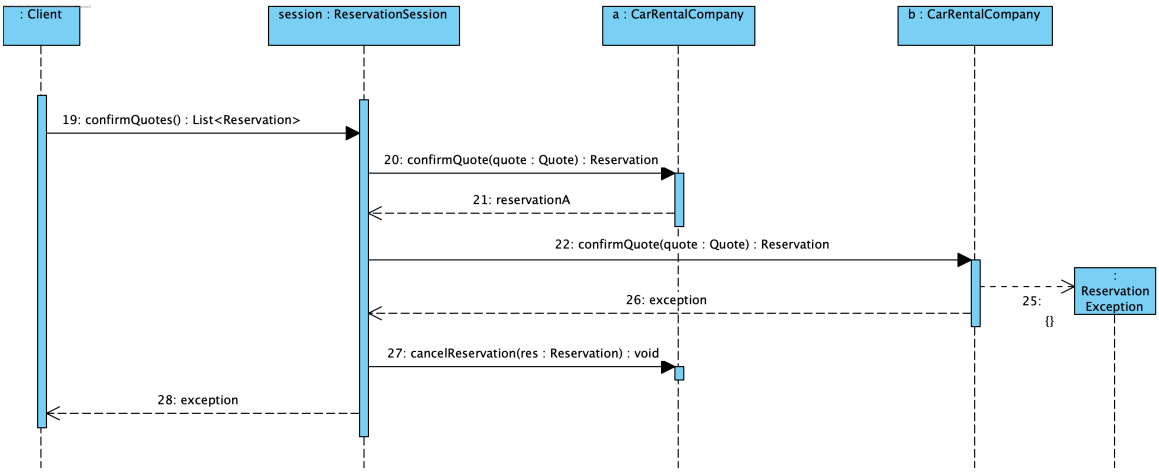



(b) The client creates two quotes.

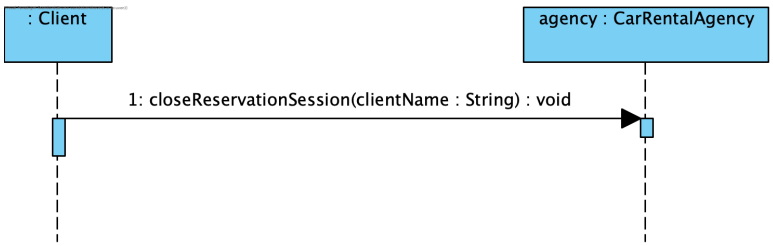
Figure 3: Relevant sequence diagrams for a failed or successful booking.




(a) The client confirms the quotes. The confirmation succeeds.




(b) The client confirms the quotes. The confirmation fails.




(c) The renter manually closes a session.

Figure 4: Relevant sequence diagrams for a failed or successful booking (continued).

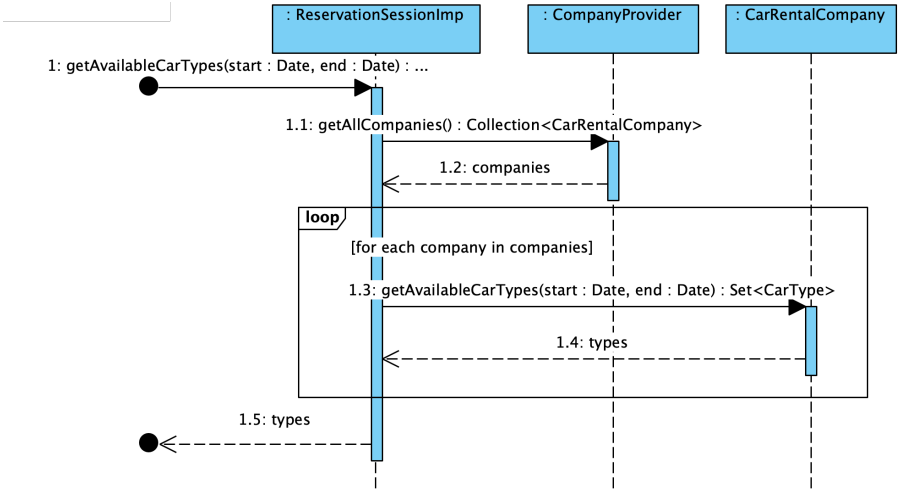



Figure 5: Getting available car types through a session. Each session has a provider implementing the `CompanyProvider` interface. Currently it's the agency that implements it. The provider can (un)register companies and keeps track of all companies such that each session can interact with them.