Distributed Systems (H0N08A)

Report: Java RMI

Elias Stalpaert *(r0669524)*
Mao-Jie Jimmy Chen *(r0637554)*

October 30, 2020

1. How would a client complete one full cycle of the booking process, for both a successful and failed case? Base yourself on the example scenarios in Figure 1. Create sequence drawings to illustrate this.

1. Client starts a new session
2. Client creates quotes for all reservations he wants to make
    - (a) Client asks session for a new quote with given constraints
    - (b) Session asks agency to contact companies for availability
        - i. Agency returns quote made at a company: session stores quote in his quotes-collection
        - ii. No companies had a suitable car: agency notifies the session which notifies the client that no quote is possible
3. Client confirms quotes (makes reservations)
    - (a) Client asks session to confirm all quotes
    - (b) Session asks the agency to try to confirm each quote with company
    - (c) Agency asks company to create reservation from given quote
        - i. Reservation succeeds, agency returns reservation to session
        - ii. Reservation fails because the car is now unavailable and returns failure to session
    - In case of failure of 1 reservation, session cancels all made reservations, removes all quotes and notifies client (our version of rollback)
    - In case of success, the session returns the made reservations to the client
4. Client closes the session
    - (a) Session asks agency to close the session
    - (b) Agency asks its SessionManager to remove given session

See sequence diagram in Figure 1

2. When do classes need to be serializable? You may illustrate this with an example class.

When objects need to be sent over the wire between client & server (as a return value or parameter), these objects need to be marshalled into a format suitable for transmission, so that unmarshalling at receiver-side will produce the same object. In Java, classes (and the types of their members in case they're not Java primitives) of which the objects need to support this functionality have to implement the java.io.Serializable-interface.

For example: when a client confirms their quotes, he wants to receive a list of Reservation objects, the Reservation-class is serializable because it inherits from the Quote-class which implements the java.io.Serializable-interface.

3. When do classes need to be remotely accessible (Remote)? You may illustrate this with an example class.

In cases where we want to have max. 1 instance of a class which needs to be accessible by other nodes in our distributed architecture. This could be for various reasons: consistency of the object (if object is returned by value, inconsistent states at different nodes are possible), size of the object (would unnecessarily clog traffic for ex.) etc. If the classes are made remote (implement an interface which extends java.rmi.Remote or java.rmi.Remote interface itself), they can be accessed via a remote reference after lookup using Java RMI registry (naming server of Java).

For example: a client's Session-object is kept server-side only because sending it over the wire with each request (small or big) would be using more network resources than is needed. We only want to have important information sent over the network (return values and parameters in this case). That's why they can only be accessed on the server through their remote interface.

4. What data has to be transmitted between client and server and back when requesting the number of reservations of a specific renter?

We assumed that each reservation goes through the agency (no reservations via other channels) so we decided to keep a number for each client in a map between client name and their number of reservations. This could however result in disk thrashing if the client decides to never return so there should be a timeout. (which is not implemented in this version) So, the only data that is transmitted between client and server is: String containing client name (parameter for the map) and an integer equal to the client's number of reservations (return value).

5. What is the reasoning behind your distribution of remote objects over hosts? Show which hosts execute which classes, if run in a real distributed deployment (not a lab deployment where everything runs on the same machine). Create a component/deployment diagram to illustrate this: highlight where the client and server are.

The central car agency has its own infrastructure for processing the reservations. This central agency will manage and interact with all the different registered car rental companies running on different hosts. This means that it acts as a middleman between client and companies. The clients should only interact with the central rental agency through a remote session and not directly with the car rental company.

See deployment diagram in Figure 2

6. How have you implemented the naming service, and what role does the built-in RMI registry play? Why did you take this approach?

We implemented the naming service as a module of CarRentalAgency (CompanyNamingService) containing a hash map linking company names to their respective remote references (ICarRental-Company). These references are added by a manager through ManagerSession where they're linked (using Naming.lookup-method) to the host running the rmiregistry for that specific company. Through this approach the CarRentalAgency can access each company's remote reference without requiring another rmi lookup call.
A disadvantage is found in the lack of flexibility when the host IP address changes. This could be mitigated by performing DNS lookup each time a company's reference is needed, but this is not the case in this particular implementation (hardcoded ip).

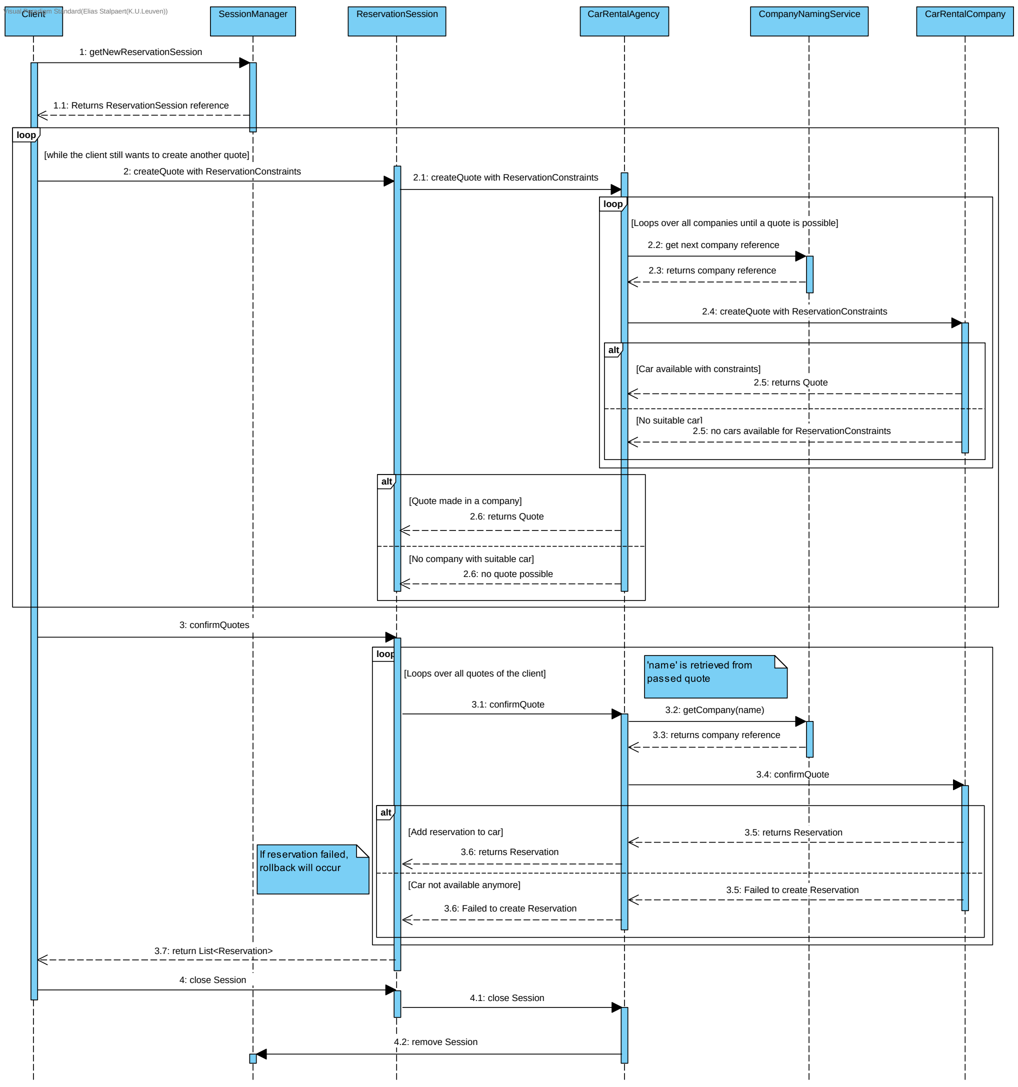
7. Which approach did you take to achieve life cycle management of sessions? Indicate why you picked this approach, in particular where you store the sessions.

Life cycle management of sessions is delegated to a SessionManager which stores the current Session-instances. SessionManager is able to instantiate all types of Sessions and returns remote references of them to the clients requesting them via its remote interface. Closing a session is also done through an explicit call to closeSession on the remote interface of CarRentalAgency which redirects the call to the removeSession in the SessionManager.

8. Why is a Java RMI application not thread-safe by default? How does your application of synchronization achieve thread-safety?
Server-side implementation where the dispatcher calls the appropriate method may not be single-threaded. If the remote method implementation isn't thread-safe, we cannot guarantee thread-safety because the dispatcher which calls the remote method may do this in a multi-threaded way.
In our application, we focused on mitigating double bookings of the same car during the same period. This is done by ensuring confirmQuote can only be run by 1 thread at once in each company's JVM.
Creating quotes however, could still give a false notion of availability of a car. But this is not a problem because during confirmQuote (which is thus synchronized) the company checks whether a car is still available (no reservations in given period) and can thus still prevent double bookings. Making createQuote synchronized would result in a much bigger bottleneck. (especially if it synchronizes on the company instance)

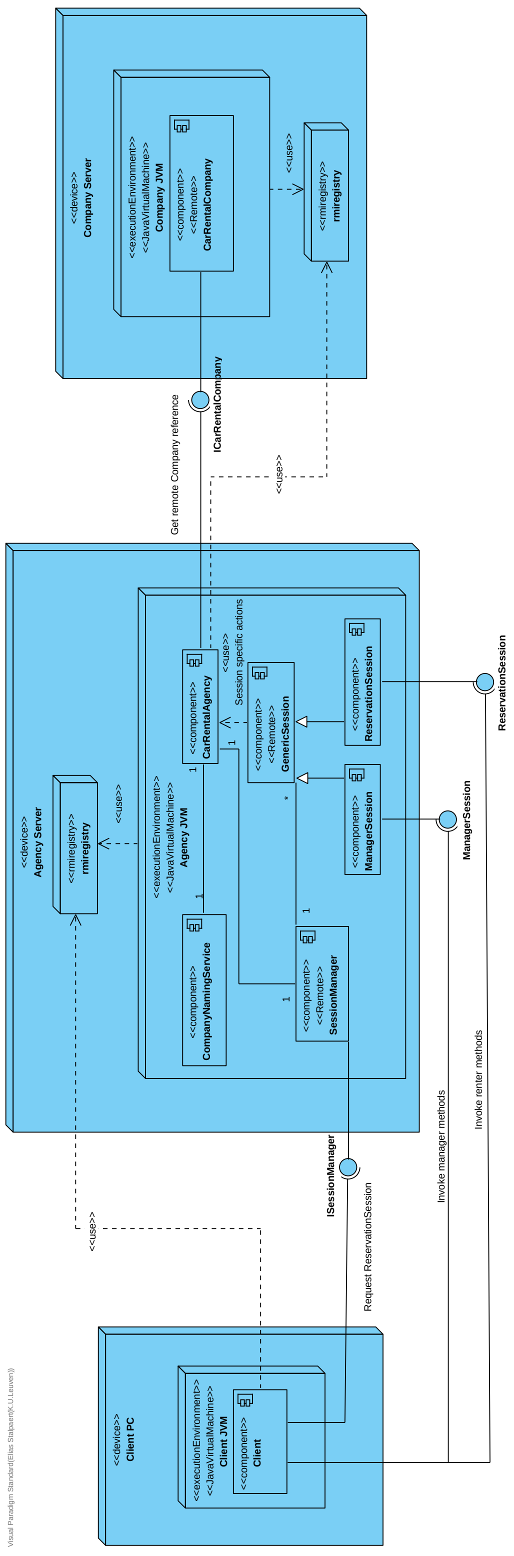
9. How does your solution to concurrency control affect the scalability of your design? Could synchronization become a bottleneck?
Our type of synchronization is coarse, by making confirmQuote synchronized, only 1 reservation could be made at the same time in the whole company! A fine-grained, less bottlenecking approach is found in synchronizing on Car-instances instead of the CarRentalCompany-instance, because while a reservation is made for 1 car, we could concurrently make a reservation for another different car in the company.

Figure 1: Booking Sequence Diagram (zooming is required to be readable (vector image))



Visual Paradigm Standard(Elias Stalpaert(K.U.Leuven))
Client
SessionManager
ReservationSession
CarRentalAgency
CompanyNamingService
CarRentalCompany
1: getNewReservationSession
1.1: Returns ReservationSession reference
loop
[while the client still wants to create another quote]
2: createQuote with ReservationConstraints
2.1: createQuote with ReservationConstraints
loop
[Loops over all companies until a quote is possible]
2.2: get next company reference
2.3: returns company reference
2.4: createQuote with ReservationConstraints
alt
[Car available with constraints]
2.5: returns Quote
[No suitable car]
2.5: no cars available for ReservationConstraints
alt
[Quote made in a company]
2.6: returns Quote
[No company with suitable car]
2.6: no quote possible
3: confirmQuotes
loop
[Loops over all quotes of the client]
'name' is retrieved from passed quote
3.1: confirmQuote
3.2: getCompany(name)
3.3: returns company reference
3.4: confirmQuote
alt
[Add reservation to car]
3.5: returns Reservation
3.6: returns Reservation
If reservation failed, rollback will occur
[Car not available anymore]
3.5: Failed to create Reservation
3.6: Failed to create Reservation
3.7: return List<Reservation>
4: close Session
4.1: close Session
4.2: remove Session

Figure 2: Deployment Diagram (zooming is required to be readable (vector image))



<<device>>
Company Server
<<executionEnvironment>>
<<JavaVirtualMachine>>
Company JVM
<<component>>
<<Remote>>
CarRentalCompany
<<use>>
<<rmiregistry>>
rmiregistry
ICarRentalCompany
Get remote Company reference
<<use>>
<<device>>
Agency Server
<<rmiregistry>>
rmiregistry
<<use>>
<<executionEnvironment>>
<<JavaVirtualMachine>>
Agency JVM
<<component>>
CarRentalAgency
<<use>>
Session specific actions
<<component>>
<<Remote>>
GenericSession
<<component>>
ReservationSession
<<component>>
ManagerSession
<<component>>
CompanyNamingService
<<component>>
<<Remote>>
SessionManager
1
1
1
*
1
1
ReservationSession
ManagerSession
ISessionManager
Request ReservationSession
Invoke manager methods
Invoke renter methods
<<use>>
<<device>>
Client PC
<<executionEnvironment>>
<<JavaVirtualMachine>>
Client JVM
<<component>>
Client
Visual Paradigm Standard(Elias Stalpaert(K.U.Leuven))