

Distributed systems

Java RMI 2-3 Exercise Session Report

Academic year 2014-2015 Bavo Goosens & Michiel Vandendriessche

**Real world deployment**

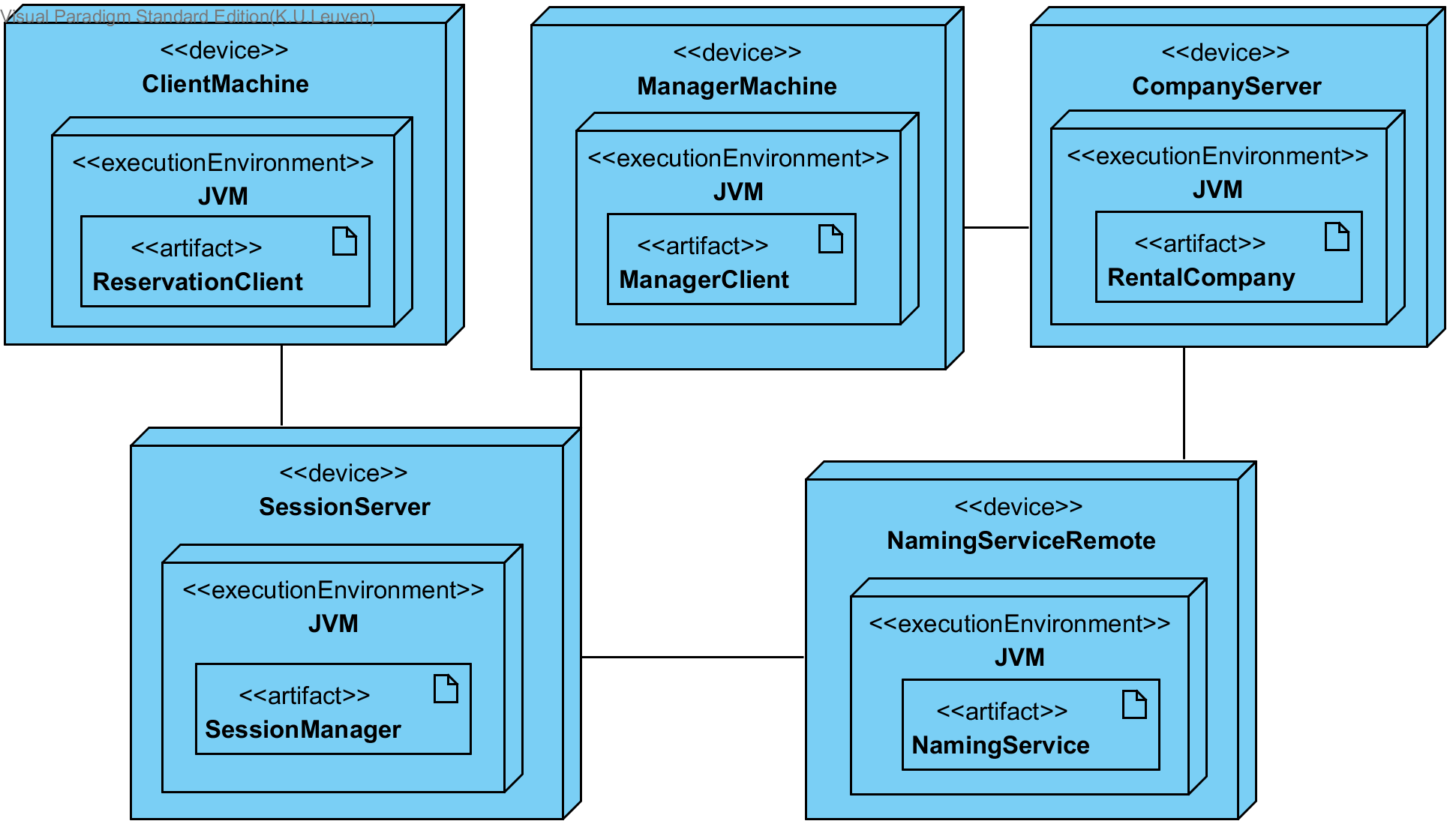


Figure 1: Deployment diagram

Figure 1 shows how our application would be deployed in a real world situation. First of all we have two types of client applications. One that can be used by the manager to start up a new rental company or remove a registered company. Furthermore it offers several useful methods a manager would use in a day to day working environment. The ManagerClient artifact runs in a different JVM (Java Virtual Machine) than the actual rental company server. Initially this will not be the case. When the manager wants to start and register a new rental company on his local server this will be done via the same JVM, but as soon as the manager exits the client the rental company will keep running on that JVM. The client can then be used to access other information via a different JVM for example via the personal computer of the manager instead of a server. The second type of client is a reservation client that offers useful methods to clients who want to order cars or want to check up on the progress of their reservations.

A different server will run a SessionManager artifact. This session manager offers two types of session. A stateful one for the ReservationClient and a stateless one for the ManagerClient. This could be a place where multithreading could be used to increase the performance of the overall system. Since the clients need to know where the SessionManager is running, this information will need to be provided to the client artefacts. All the interaction a client of the system needs will be offered through these sessions. This makes the application transparent to the users and gives us more freedom in how we internally keep track of all the companies.

The last important artefact is the NamingService. It is used to register and unregister the rental companies. It will however never be accessible to a RentalCompany. The SessionManager knows where to find and use the NamingService artefact.

**The following classes are remotely accessible:**

**NamingServiceRemote:**

This remote interface is used by all sessions. Through this interface sessions can register new companies, unregister registered companies, acquire a company by name or acquire all registered companies from the naming service.

**ManagerSessionRemote:**

This remote interface is used by the manager clients. Through this interface manager clients can invoke methods on their manager sessions.

**ReservationSessionRemote:**

This remote interface is used by the reservation clients. Through this interface reservation clients can invoke methods on their reservation sessions.

**SessionManagerRemote:**

This remote interface is used by all clients. Through this interface clients can acquire new manager sessions or new reservation sessions

**ICarRentalCompany:**

This remote interface is used by all sessions. Through this interface sessions can invoke methods on registered companies.

**The following remote objects are serializable:**

ManagerSession, ReservationSession and SessionManager objects are all located at the same host.

…

CarRentalCompany objects are located at a different host

…

NamingService object is located at yet another host.

…

**The following remote objects are registered via the built-in RMI registry:**

**NamingService:**

The naming service is registered via the built-in RMI registry. That way the host that runs all sessions can acquire a proxy object of the naming service so that the sessions can invoke methods on it (via the NamingServiceRemote interface).

**SessionManager:**

The session manager is registered via the built-in RMI registry. That way the host that runs the client code can acquire a proxy object of the session manager so that the client can invoke methods on it and acquire sessions from it (via the SessionManagerRemote interface).

**Life cycle management of sessions:**

Sessions are created when a client invokes the getManagerSessionRemote or getReservationSessionRemote methods on the session manager. Sessions only live on the host where they are created (there is a separate host which runs all sessions). Only proxy objects for sessions are sent over the wire and returned to the client.

Sessions are closed when the client invokes the closeManagerSession or closeReservationSession method on the session manager. When these methods are invoked, the session manager removes all references to that session.

We assume the client closes the session itself. Although, we can easily implement an automatic session closing system which uses a time out.

**Synchronization:**

Synchronization is necessary at the confirmQuote method in a CarRentalCompany. After all, multiple clients can all have their own individual session at the same time. That session can invoke methods on the same CarRentalCompany-object. When the company tries to confirm a quote, it checks whether it is able to do that. When multiple clients invoke this method at the same time, the integrity of the system is compromised (imagine two clients making the same reservation at the same time). Because of that reason, the method needs to be synchronized.

