Java RMI Design Report

Jakub Adam Matteo Di Pirro

November 3, 2017

1 Design overview

The main components of the `CarRental`'s distributed design are located in the `session` package. It comprises the three remotely accessible interfaces: `SessionManagerI`, `RentalSessionI`, and `ManagerSessionI`. The first manages the association between renters and their rental session. A client may create a new rental session only via this manager. Clients are identified by their name, which is supposed to be unique. This class maintain a conversational state with the corresponding client. This is the reason why it is a stateful session. Lastly, `ManagerSessionI`, provides a stateless session for managing purposes. It is stateless because there is no need for conversational state. The intended behavior of these components is as follows. The renter asks the session manager for a unique rental session. This rental session is made persistent and returned to the client. From now on a binding <client, session> is created and maintained until the client terminates the session. A manager session, on the other hand, can be open on the fly without the need of the aforementioned binding. A manager uses this session for managing purposes without storing any data.

The `rental` package implements the application's business logic. There is only one addition to the provided code: `RentalStore`. This class implements the naming service and provides company registration, unregistration and lookup.

2 Design decisions

2.1 Remote and Serializable classes

The three interfaces defined in the `session` package are remotely accessible (i.e. they are stereotyped with <<`Remote`>>) in the class diagram. This design choice follows the Java RMI's standard mandate. After defining the interfaces as <<`Remote`>> we can register their concrete implementation to the `rmiregister`. This allows the client to ignore any implementation details and to act only considering the remote interface. The actual code can be dynamically downloaded or distributed as a `.jar` file.

We made serializable (i.e. stereotyped as <<`Serializable`>>) only the classes intended to be marshaled and unmarshaled. These classes do not include the remote accessible ones, since for them Java RMI sends a remote reference and not an object copy. Below is a list of the serializable classes.

- `ReservationException`
- `ReservationConstraint`
- `CarType`
- `Quote`
- `Reservation`

Instances of these classes are used as parameters or return values by remote object's methods. As mentioned before, Java RMI uses a pass-by-value semantics for non-remote objects, which involves marshaling and unmarshaling. Since instances of the other classes in the `rental` package are not sent over the network, those classes do not implement the `Serializable` interface.

2.2 Location and registration of remote objects

2.3 Sessions life time management

2.4 Thread synchronization