Distributed Systems (H0N08A)

Report: Java EE

Dries Janse *(r0627054)*
Steven Ghekiere *(r0626062)*

November 20, 2019

1. Outline the different tiers of your application, and indicate where classes are located.

Our application has 3 main tiers: The application client, the EJB container and the database server. The classes are located as follows:

TODO diagram

- The Main class is on the *application client* side, with the getSession methods and other client-side functionalities.
- The CarRentalSession and ManagerSession are the Enterprise beans inside the *EJB container*. These classes contains the business logic of the application, and the client invokes these methods so they can access the services offered by these beans.
- The *database server* manages and provides access to the data. The enterprise beans uses JPA and JDBC to manage the data inside a Derby database.

2. Why are client and manager session beans stateful and stateless respectively?

Both the manager and client session beans are server-side (EJB) Enterprise Java Bean components. The component is a session bean, because it represents a session with the client. The client session is a stateful session bean, this means that it retains state between method invocations. The (conversational) state consists of the values of the instance variables. The client session is stateful because it keeps the conversational state, it contains the name of the renter and the list of quotes of this renter. The manager session is a stateless session bean, this means that it does not retains the conversational state. In our application there is no need to make the manager session stateful because all instances of the bean are equivalent. This allows the EJB container to assign an instance to any manager.

3. How does dependency injection compare to the RMI registry of the RMI assignment?

The RMI registry of the RMI assignment is a simplified name service that allows clients to get a reference to a remote object. The @EJB annotation can be applied on fields or methods to inject dependencies. The EJB client container will first perform a JNDI lookup to locate the dependency. JNDI stands for Java Naming and Directory Interface, it allows distributed application to look up services in an abstract way. With JEE we don't have to explicitly instantiate, with the "new" keyword, the objects of which we need a reference. JEE provides these injection mechanisms. We only have to declare the needed resources with the annotations (@EJB) that denotes the injection point to the compiler. The container will then provide an instance of the required resource at runtime. The advantage of using dependency injection is that it simplifies our code because the container automatically provides these instances.

4. JPQL persistence queries without application logic are the recommended approach for retrieving rental statistics. Can you explain why this is more efficient?

JPQL uses the entity object model instead of querying actual tables in the database. This makes it more easy for the Java developer. To answer on the question; there are multiple advantages of retrieving rental statistics without using application logic.

For example, retrieving the number of reservations made by a particular car renter. With only a JPQL query, one number is returned. Only one number is send from the database server back to the Jave EE Server. The JEE server then only has to create an integer and store the result. If we used application logic to calculate the number of reservations for a particular car renter, we first had to request all the reservations from the database. This results in a lot more network traffic between the database server and the Java EE Server. For each reservation an object is created on the Java EE server, which results in a big memory usage. The application now has to make operations on the list of reservations, which can take substantially longer than a database. This is because the database can be indexed.

5. How does your solution compare with the Java RMI assignment in terms of resilience against server crashes?

The resilience against server crashes depends on which component crashes on which server. This is because we are talking about a distributed application. In the Java RMI assignment all the car rental companies store their data in memory. So if a server crashes which contains a car rental company all the data is lost when the server reboots. If the server with contains the Naming Service component crashes and reboots, all the car rental company references are lost, this is because they are stored in memory. If the server with Rental Agency crashes and reboots all the sessions are lost with the client.

The biggest difference with this is assignment is that all the data is stored in a database. So if the server crashes which contains the CarRentalSession and ManagerSession classes all the data will be persistent. Only the data of the sessions will be lost because they are impersistent.

6. How does the Java EE middle ware reduce the effort of migrating to another database engine?

The entities in the application are persisted in the same manner, and the only change is the location and name of the database. So the only configuration needed to change is inside the persistence unit (persistence.xml). JPA and JDBC automatically generates the relations and the tables of the entities. For the actual migration to another database, the copying of the data from one database to another, is done by the database administrators. They have to set up a migration strategy for retrieving the database from the first database and inserting it into the new database.

7. How does your solution to concurrency prevent race conditions?

Thread handling is done by EJB container and JSE-level thread handling is discouraged to use by the developer before JEE 7. This means that for example the use of the 'synchronized' keyword, that we used in the RMI lab, is discouraged.

TODO: misschien hier nog een beetje bij?

8. How do transactions compare to synchronization in Java RMI in terms of the scalability of your application?

Transactions make sure the required data modifications are either all saved (committed) or rolled back. If any of the statements fail of our transaction, the transaction will roll back and all the statements are undone. Since the EJB container handles the rollback, we do not need to implement the reverse operation. We only need to raise an exception telling the container there went something wrong.

Within Java RMI we cannot make use of the rollback functions and thus we needed to implement the rollback functionality ourselves. After we've undone our statements, we raise an exception to the client as expected.

9. How do you ensure that only users that have specifically been assigned a manager role can open a ManagerSession and access the manager functionality?

By the use of server side authorization, we check if a user calling the ManagerSession methods has the Manager role. On our GlassFish server, we map certain users in the required Manager group. These users are stored inside a realm, which uses a policy with its own method of authentication and authorization. Our application uses this realm with it's policy, and so the users in the Manager group are able to log in using the default authentication prompt when using the ManagerSession's methods.

10. Why would someone choose a Java EE solution over a regular Java SE application with Java RMI?

Java EE offers a wide variety of functionalities that could be used for building a large scale distributed system, which would be hard to do in Java SE. Java EE provides an API for a lot of useful libraries (JDBC, JPA, dependency injection, servlets, ...) which makes creating these kind of applications easier. It wouldn't be impossible to create the same application using Java SE, but it would require a lot more work and testing.