Lab 5-6: Distributed Use Case Demonstrators

Regulations as in previous lab.

Problem and tasks

Simplest task of software design of a distributed OO systems is designing a two-tier (client-server, C-S) system. Most simple distribution technologies are Java RMI (educational only for simplicity, lab5) and Apache ActiveMQ (message-oriented middleware, MOM, lab 6).

The goals are:

- a conceptual understanding of how distributed object systems are engineered,
- starting from refined implementations of lab 4 UC demonstrators, design your Client-Server(CS)-distributed UC demonstrators that involves both components, robot system and mobile app component,
- lab 5: technical understanding of Java RMI, implementation, test and documentation of the designed RMI-distributed UC demonstrator,
- lab 6: extending design (explicite proxy classes), implementation, test and documentation of your lab 5 demonstrator as a MOM-distributed application.

The app teams no-A are asked to extend their lab-4 UC-cluster demonstrator to include some robot-side classes that they either implement themselves or that they get by integrating (part of) classes developed by the robot team no B of their quartet.

The firmware teams no-B are asked to extend their lab-4 UC-cluster demonstrator to include some app-side classes that they either implement themselves or that they get by integrating (part of) classes developed by the app team no A of their quartet.

Complexity required is

1. C-S contract shall consist of at least 2 interface classes, one implemented by the app, one by the robot, and at least one of your data-classes for CS-transport as method argument or as return value.
2. Your design patterns must be kept, either on your original side or CS-distributed
3. Your demonstrator uses remote method invocations (RMI) that trigger, after the first RMI, object state changes on both C&S sides.

For lab 5 preparation:

- Complete, refine and document your lab-4 UC-cluster demonstrator with design patterns if not yet achieved (executable Java code in gitlab!),
- Conceptual and technical understanding as described below,
- Design of your distributed demonstrators (points 1 and 2 above).

For lab 5 and lab-5 report

- Remaining work, in particular point 3 above.

Lab 5 additional details

1. Conceptual understanding:

- Why do C-S contracts of distributed systems define interfaces and classes? Refer to the lecture slide sets!
- What is the role of the interfaces and of the classes in the contract of a distributed object system (procedural RMI and message-based)? Refer to lecture code examples!

2. Technical Understanding of Java RMi

- Use the provided code examples, explain in your report what it means and under what conditions you declare an interface or class or object as
   a. java.rmi.Remote (give corresponding examples!)
   b. java.io.Serializable (give corresponding examples!)
   c. java.rmi.UnicastRemoteObject (give corresponding examples!)
   d. to be exported by calling rebind (give corresponding examples!)
- and if you can do that for interfaces/classes/objects on client and/or server side.
- Explain which methods require to throw a java.rmi.RemoteException
- Espain the role of the registry. Remark concerning your singleton pattern if present: The proxy object looked-up in the examples is automatically kind of singleton object.

3. Design of the your C-S-demonstrator

- Explain why the app and the robot take the client or server part in your demonstrator and what their functionality related to UCs in labs 1-2 will be.
- Explain what interfaces you have to introduce in your design and which classes need to be declared as one of a. – d. and for what reason (by color-coded class diagram as shown in lecture 9 slide set).

Lab 6 additional details based on lecture 10

Task:

- Prerequisite: Use lab 6 template extension! Ensure that your RMI-Demonstrator fulfills the complexity requirements given above, otherwise refine it accordingly!
- Prepare a demonstration of it on two computers connected to a mobile hotspot!
- Replace RMI middleware by MOM using JMS together with **Apache ActiveMQ 5-18-6 (required!)** while keeping the functionality of your (refined) demonstrator!

1. Conceptual understanding:

- What is the role of the message queue (MQ) in MOM?
- Why is Class-Design of the JMS-distributed demonstrator very similar to that of the RMI-distributed demonstrator?
   - What is identical? What are the differences? Name four of them!
- Name two MOM services! Which of them is best for which remote method calls?

2. Technical Understanding of JMS

- Package javax.jms does not contain any executable code. Why is it enough to standardize Interfaces in JMS? How are objects generated then? Give five examples of JMS-based object creation in my JMS-distributed examples, one line of code each!
- Which components are JMS client, which JMS server? In which sequence do you need to start the components?
- How do you inspect sending and delivery of your messages in the message queue?

Now consider "SE 09 Renz DistributedSystems2 MOM" lecture slide set and the RMI- and JMS-Laborkurs example in my "Client-Server Examples Renz 5-18-6"!

- Which classes define here the C-S contract?
- Why does Caller implement the LaborkursInterface? Which is the corresponding class in RMI-Laborkurs? If it is not visible, how could you make it visible? Cf. RMI slide set!
- What is the reason to place the Student class in a serobjs package for JMS? What happens if you remove serobjs package from trusted packages in JMSClient class?
- Which mechanism is used in class Responder for defining specific message receivers each responsible for one of the remote message calls defined in the C-S interface? Hint: Check section 2 of the above slide set, which describes the MQ!
- What do the Lambda-Expressions in messageConsumer.setMessageListener() define and which Interface and method do they implement?
- What is the difference between receiving messages in the JMS-Eliza example and receiving messages in the JMS-Laborkurs?
- Consider the JMS Message structure as given in the lecture slide set! How is the above Responder-side mechanism supported in defining messages in the Caller?
- Why is there only a client side Caller and a server side Responder here?

3. Design of the your MOM-based C-S-demonstrator

- Which is your server side, the App or the Robot Firmware?
- Which classes make your C-S contract? Give the Interface definitions and corresponding MessageKeys! Name the DTO class(es) with instance variables, i.e. object properties!
- Which are your caller and responder classes for each of the Interfaces? Be aware of the fact that such classes can implement more than one Interface!
- Edit your BlueJ class diagram so that server side is left, client side right and common classes in the middle, or the other way round.