Lab 2 – RMI Distributed Messaging

CC5212-1 – March 16, 2022

We will build an instant messaging service using Java RMI (Remote Method Invocation). The idea of RMI is that you can call Java methods that run on other people's computers. Of course they must have a Java server installed to implement/run those methods and you have to find them using their hostname/IP[1] and a port.

This lab will cater to two options. For those attending in person, we may get to message each other's machines directly (if the network permits). For those at home, this will not be possible without complex configurations for fixed IPs, etc. In case you cannot message from your machine due to network issues, we will use a centralised machine to emulate the service. These instructions assume the latter more general case. However, in the class we may give some specific instructions to have the messaging work within the room.

The goal is to set up an RMI application on the machine to receive and print messages on a port that you choose. Anyone in the class can find your application and print you a message. But to find you and connect to your application, they need your IP/port. How can we manage that? Do we share all our ports with each other? That would be $O(n^2)$ messages.

There are a few more efficient ways. The one we will use is a central server that runs an RMI directory of usernames to ports/IPs. Everyone will upload their username, IP and port to this one directory and view those of others. Once you have the details of all users in the directory, you can choose whom to message by their username (if they are still online), or you can spam everyone. The messaging part is peer-to-peer: messages are sent directly, not through the central directory.

- The machine we will use is accessible through SSH at `cm.dcc.uchile.cl:220`. To log in on a terminal (in Windows, Mac, Unix), you can run `ssh -p 220 uhadoop@cm.dcc.uchile.cl`.[2] I will give the password during the session.
- Type `pwd` to see the current folder. It should be `/data/2022/uhadoop/`. Create a new folder for yourself (or your group): `mkdir` `FOLDER-NAME`.
- On this server, I am running a central directory.[3] You can look at the log of the directory by running
  `more /data/2022/uhadoop/directory/directory.log`
  or if you want to follow updates in a separate terminal, use
  `tail -f /data/2022/uhadoop/directory/directory.log`
- Download `mdp-lab02.zip` from u-cursos and import it into your IDE of choice (Eclipse, IntelliJ, etc.).
- In the class `StartRegistryAndServer.java` you will find an example of how to start a registry and bind a skeleton/implementation to it on the server side.[4] In `UserDirectoryStub` you will find an interface shared by the client and server. In `UserDirectoryServer` you will find an implementation of that interface running on the server. This is what the central directory I have running uses.
- We will work on two class: `InstantMessagingApp.java` and `InstantMessagingServer.java`. There's some partial code left there that you will have to complete. Before we start coding, however, run `InstantMessagingApp.java` with the argument `-n 192.80.24.222` (this is the IP of the machine we logged into before with the central directory). Add a username, add your real name, add the IP of your machine (or `locahost` if not sure)[5] and add a port of your choice in the range. If you enter `add`, this will try to add you to the directory and you will see in the `directory.log` that it tried to add you. This will fail because parts of the code are not implemented yet, and even when they are it will probably still fail: although you can connect to the machine of the directory and call methods on it (it has a static/external IP), it cannot connect to you to call methods on your machine (you likely don't have a static/external IP and the necessary ports open). To get around this issue, once we have coded everything, we will eventually run this application on the same machine as the directory.

[1] We can use hostnames or IP addresses. A hostname might be something like `localhost` or `cm.dcc.uchile.cl`. An IP address might be something like `64.90.49.28`. These can be used interchangeably (if a hostname is used, DNS will give the IP).

[2] A useful application for Windows is PuTTy.

[3] You can run a directory elsewhere with `java -jar mdp-lab02.jar StartRegistryAndServer -r -s 1 -n` `IP-OR-HOST-OF-MACHINE` `-sp`

[4] One line – `.newinstance()` – is deprecated, but this is to maintain compatibility in case some people run an older version of Java.

[5] On Windows, run `cmd` and type `ipconfig` (look for the value on the `IPv4 Address` line). On Unix, open a terminal and put `ip addr show`.

- Next we are going to implement the message client/server you will run. Go to `InstantMessagingServer` and implement the `message(.,.)` method. *What should it do?*

- Next head to `InstantMessagingApp`. This application puts everything together: (1) it can register your information in the CENTRAL DIRECTORY and fetch other peoples information from there; (2) it can create a registry on your local machine and bind your MESSAGE SERVER that other people can call to message you; (3) it will implement a MESSAGE CLIENT that can connect to other user's message server using details from the central directory and send them a message you write. A bunch of input/output stuff is done for you; please have a look at what this does. What you need to do is implement the methods at the bottom:

  - CENTRAL SERVER: In `connectToDirectory`, we open the remote registry of the central directory, find the stub for the directory class, and return it. The code is already done. With this stub, in the main method, you can add yourself to the central directory, or request other users' details.
  - MESSAGE SERVER: You already implemented the message server in INSTANTMESSAGINGSERVER. Now you need to set it up so other users can call it given your details in the central directory ...
    * First implement `startRegistry(.)`, which opens an RMI registry on your local machine on the given port.
    * Now we need to implement `registerMessageSkeleton(.)`. Remote users cannot directly call the local Java class `InstantMessagingServer`. You need to create a *skeleton* – a remotely callable version – of an instance of `InstantMessagingServer`. Once you have that skeleton, you need to bind it to your registry with a key so other users can find it by the same key and call methods on it. Finally, you should return the skeleton.[6]
  - MESSAGE CLIENT: The final piece of the puzzle is the message client. Other users will have set up message server just like you, so now we need a local client to connect to their remote servers. The central directory will give you a list of their IPs/hostnames and ports to choose from.
    * In `messageUser(.,.,.)`, open up a user's registry with the given IP/hostname and port. Retrieve from that registry an `InstantMessagingStub` interface using the same key under which you registered your own server (they should have used the same key). Using that interface, call `message(.,.)` to message that user. Return the time returned by their server (which is the time they received the message).

- You can try to run `InstantMessagingServer` as before, but even though you can call methods on the directory, it's quite likely that the directory will not add you as it will not be able to reach your machine (this is a feature to avoid the directory filling up with users that are no longer online or contactable). Instead you will have to build the jar, copy it to the server, and then run it.

  - To build the jar, in your IDE, right-click on `build.xml` > `Run As` > `Ant Build`. This should compile and create a new `mdp-lab02.jar` Jar file in your `dist` folder (fit F5 to refresh).
  - To copy to the machine, use (with the same password as before)
    `scp -P 220` PATH`/dist/mdp-lab02.jar uhadoop@cm.dcc.uchile.cl:/data/2022/uhadoop/` FOLDER-NAME
  - To run the messaging application, login to the same machine again via SSH and run:
    `java -jar /data/2022/uhadoop/` FOLDER-NAME`/mdp-lab02.jar InstantMessagingApp -n 192.80.24.222`
  - As your hostname/IP you can put `localhost`. Thereafter try `add` to add yourself to the directory (it should work this time). Try `list` to see the available users. Try `msg` to send someone a message. You can even message yourself!

- OPTIONAL: In `messageUser(.,.,.)` it opens a connection every time you message. Maybe you can cache connections instead?

- QUESTIONS: What is the role of the central server here? What effect would it have if the central server died? If you were to send a nasty message about me to someone, could I read that from the central server later and feel sad about it? What effect would it have if a peer died? What happens as the number of peers grows? How could we change the architecture of the overall distributed system to avoid a central server entirely? How could we use such a distributed system to count n-grams?

- Submit the `InstantMessagingApp` and `InstantMessagingServer` classes to ucursos with the names of your group members before the deadline.

[6] We don't actually use the skeleton outside, but it we do not return the skeleton, the reference will be lost and the garbage collector will eat the skeleton before users can call it.