Lab 1 Assignment

CHAT SYSTEM

Catherine Nawire Rakama | ID2010 Programming of Interactive systems | 05/08/2018

INTRODUCTION

Most interactive systems are widely distributed and may be hosted on different webservers. For such systems to communicate and interact in the distributed environment which has different communication protocols diverse system specifications, there is a need of a communication paradigm. Communication paradigm can be direct (RMI) or Indirect (Message sending). Direct communication paradigm is good because its transparent - remote method call looks as local method call and developers don't need to worry about transport, network, data link or physical layer protocols. Its space uncoupled, the sender needs not to know about receivers IP and port number because RMI uses lookup service to get a reference to the remote object that it needs to communicate with. The down side are; RMI is time coupled, if remote object not found then the transaction is failure and does not guarantee persistence of messages, if both sender and receiver fail for instance. For these reasons majority of solutions today are implemented using message queues sending paradigm e.g. pervasive applications.

This lab specifically implements a client server application which relies on Jini infrastructure to discover available services in a distributed environment and RMI middleware to remotely invoke method of a program running on a different machine. To simulate the distributed environment, the lab contains codebases with several sub folders/directories. Distributed environment means that the programs are not installed directly on the host machine, instead they run on a web server or different web servers and a bootstrap program (JarRunner.jar) is executed to fetch the real program from the server. The applications (client and server) then use Jini look up service discover each other.

Lab requirements/Tasks:

- Enable all clients connected to the service to see whenever a client joins or leaves the chat.
- Add a user command to the client that allows the user to list users currently joined to the service.
- Consider the possibility that the list is longer than there are lines on the screen. Make this information meaningful to the maintainer of the service, so that it clearly shows information about the client, the user and (upon disconnect) session statistics.

IMPLEMNTATION AND EXPERIMENTS

I. Enable all clients connected to the service to see whenever a client joins or leaves the chat.

This requirement has been achieved by refactoring *register* and *unregister* server methods. A client instance through Jini's service discovery manager, has access to all server services that are running. The client requests to connect to a specific server using the command ***.connect <servername>,*** this involves registering a *RemoteEventListener* of this client to the server, in return the server responds by generating a notification event that is sent as a message through the event listener. The server maintains a HashMap which stores an event listener as key and Client name as Value. The map creates a list of clients that are active on this server. Clients are added to the list during register by issuing ***.connect command***, which eventually calls server ***register method*** defined in the server's interface/stub that is common between the server and client. The ***register*** method calls a local method to add client to the Map. The same process is followed when the client gets disconnected. Once added, the server generates a notification event, which is sent back to the client through the listeners registered in the Map. All methods accessing the Map are ***synchronized*** to prevent concurrent modification because different client threads will be accessing the map. Figure 1 below shows general view of client sessions at the server. More elaborated view recorded at each client's terminal can be seen in figure 2, figure 3 and figure 4 in the next session.

```
crakama@crakama-VirtualBox: ~/Projects/PIS/chatsystem/test/bin/unix$ ./chatserver.sh -n server1
Server ChatServer server1 on crakama-virtualbox started.
Server> Registered as Jini service 2a20e3eb-531a-438e-8ede-f1d57f29c9f8
Added client : client1
Added client : client2
Added client : client3
MSG#1:client3: Hello!
MSG#2:client1: Hello there!
MSG#3:client2: Good too see you all online
MSG#4:client3: sorry, I got to go. See you later. Cheers!
Removed client : client3
MSG#5:client1: soon, that's sad.
MSG#6:client1: Only the two of us left! Got to go too.
Removed client : client1
Removed client : client2
```

Figure 1: Session Statistics at the server terminal

II. Add a user command to the client that allows the user to list users currently joined to the service and Information about session statistics.

The command ***.listclients*** just like register method, it uses an active server service, through ***list*** method defined at the server stub to access local server method which retrieves list of all active agents and returns the list as a result. ***Figure 2*** and ***Figure*** 4 demonstrates.

```
crakama@crakama-VirtualBox: ~/Projects/PIS/chatsystem/test/bin/unix$ ./chatclient.sh
[Output from the client is in braces]
[Commands start with '.' (period). Try .help]
[When connected, type text and hit return to send]
Client> [Added server net.jini.core.lookup.ServiceItem@365e3fc7]
.name client1
Client> .connect server1
[Connecting to ChatServer server1 on crakama-virtualbox...0 : client1 Joined the Cha
t Group
ok]
Client> 0 : client2 Joined the Chat Group
0 : client3 Joined the Chat Group
1 : client3: Hello!
.text Hello there!
2 : client1: Hello there!
Client> 3 : client2: Good too see you all online
4 : client3: sorry, I got to go. See you later. Cheers!
4 : client3 Left the Chat Group
.text Leaving so soon, that's sad.
 Client> 5 : client1: soon, that's sad.
.listclients

List of Users Currently in the ChatGroup
client2
client1

Client> .text Only the two of us left! Got to go too.
Client> 6 : client1: Only the two of us left! Got to go too.
.disconnect
6 : client1 Left the Chat Group
[Disconnected from ChatServer server1 on crakama-virtualbox]
Client> [Removed server net.jini.core.lookup.ServiceItem@649519df]
[Added server net.jini.core.lookup.ServiceItem@3f3dee94]
[Removed server net.jini.core.lookup.ServiceItem@2cdc3cd4]
[Added server net.jini.core.lookup.ServiceItem@1eb59dd]
```

Figure 2: Client 1, ChatGroup Session

```
crakama@crakama-VirtualBox: ~/Projects/PIS/chatsystem/test/bin/unix$ ./chatclient.sh
[Output from the client is in braces]
[Commands start with '.' (period). Try .help]
[When connected, type text and hit return to send]
Client> [Added server net.jini.core.lookup.ServiceItem@1dce4a07]
.name client3
Client> .name client3
Client> .connect server1
[Connecting to ChatServer server1 on crakama-virtualbox...0 : client3 Joined the Chat Group
ok]
Client> .text Hello!
Client> 1 : client3: Hello!
2 : client1: Hello there!
3 : client2: Good too see you all online
.text sorry, I got to go. See you later. Cheers!
Client> 4 : client3: sorry, I got to go. See you later. Cheers!
.disconnect
4 : client3 Left the Chat Group
[Disconnected from ChatServer server1 on crakama-virtualbox]
Client> [Removed server net.jini.core.lookup.ServiceItem@21536e57]
[Added server net.jini.core.lookup.ServiceItem@7ed7395a]
[Removed server net.jini.core.lookup.ServiceItem@1576a5ce]
[Added server net.jini.core.lookup.ServiceItem@4474a62c]
```

Figure 3:Client 2, ChatGroup Session

```
crakama@crakama-VirtualBox: ~/Projects/PIS/chatsystem/test/bin/unix$ ./chatclient.sh
[Output from the client is in braces]
[Commands start with '.' (period). Try .help]
[When connected, type text and hit return to send]
Client> [Added server net.jini.core.lookup.ServiceItem@3a58cef1]
.name client2
Client> .connect server1
[Connecting to ChatServer server1 on crakama-virtualbox...0 : client2 Joined the Chat Group
ok]
Client> 0 : client3 Joined the Chat Group
.listclients

List of Users Currently in the ChatGroup
client2
client1
client3

Client> 1 : client3: Hello!
2 : client1: Hello there!
.text Good too see you all online
Client> 3 : client2: Good too see you all online
4 : client3: sorry, I got to go. See you later. Cheers!
4 : client3 Left the Chat Group
5 : client1: soon, that's sad.
.6 : client1: Only the two of us left! Got to go too.
6 : client1 Left the Chat Group
.text ok
[.text: unknown command]
Client> .disconnect
6 : client2 Left the Chat Group
[Disconnected from ChatServer server1 on crakama-virtualbox]
Client> [Removed server net.jini.core.lookup.ServiceItem@2f296e71]
[Added server net.jini.core.lookup.ServiceItem@ab317e3]
[Removed server net.jini.core.lookup.ServiceItem@4d4d7266]
[Added server net.jini.core.lookup.ServiceItem@2d6982af]
```

Figure 4: Client 2 ChatGroup Session

ANALYSIS AND CONCLUSION

One of the design decisions made for this lab is use of synchronize to manage threads access to HashMap data structure by use of monitor allocation. HashMap data structure was used to store event listers with their associated client name that is user friendly for display as shown in the experiments.

The implementation uses RPC asynchronous client-server model whereby message queue has been used to make the system ***time uncoupled,*** meaning the client drops messages in a queue and continue with other operations while waiting for server response.

The main goal of the lab was to learn how to develop a simple distributed and interactive system made of components that are autonomous, reactive and proactive. ***Reactivity*** property has been demonstrated by the server component having to generate a notification event and send to all registered clients upon receipt of a new chat message from one of the client instances. The component has perceived its environment (new message has arrived) and responds in a timely fashion (generate a notification event to all clients). ***Proactivity*** has been demonstrated by the lookup service which always takes the initiative to discover newly registered services or components. ***Autonomy*** property is mostly demonstrated in **lab 2 Assignment.** Link to source code on github