Lab Exercise 2 – Simple Instant Messaging Implementation With RMI

CIS656 – Fall 2012

Due Date: October 4, 2012 (see below for description of deliverables.)
Grade Weight: 25 points (out of a total of 100 Lab points)

Objective

The objective of this project is to give hands-on experience building a simple approximation of a popular distributed Internet application: instant messaging. Successful completion of this lab will increase the students' understanding and appreciation of the topics covered in the lectures of this course: RPC/distributed objects, transient messaging protocols, processes, multithreading, and naming services.

Lab Description

Instant messaging is one particular application in a broader class of applications known as presence-based applications. Presence-based applications are inherently distributed applications that build on the notion of a user's (or computer's) virtual presence in a particular network context. While most popular instant messaging systems (AIM, Yahoo, MSN) have varying detailed feature sets, conceptually they are very similar.

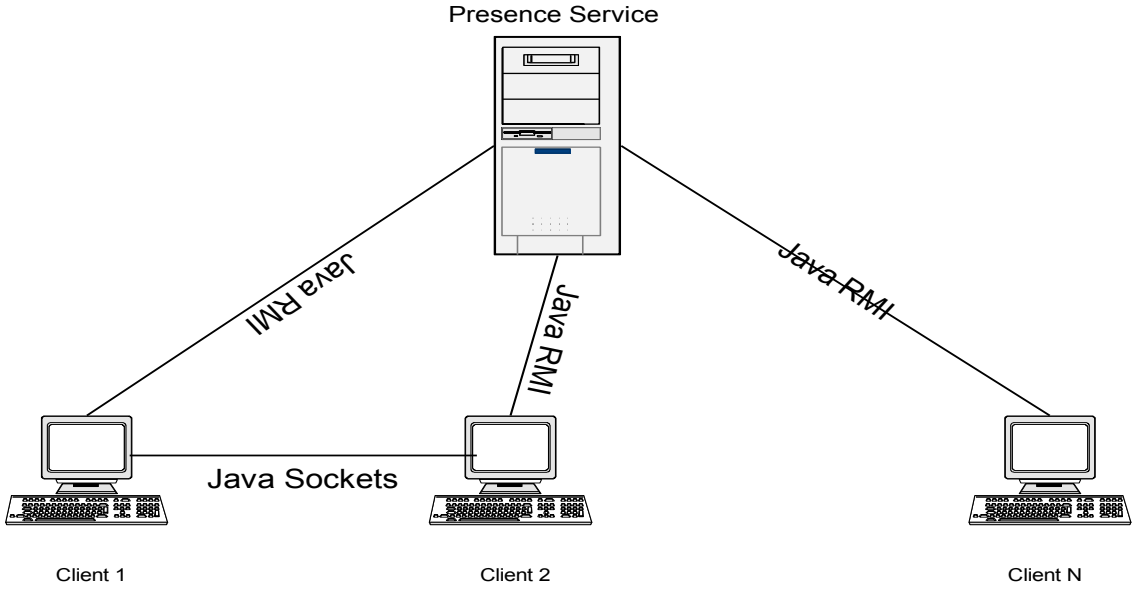

Figure 1 Instant Messaging Architecture

The architecture of the instant messaging system you will build is shown in Figure 1 above. There are essentially two major types of components in this system:

1. Presence Service – The Presence Service is essentially a remote naming service that is to be implemented using Java RMI. The abstract interface (PresenceService.java) and related data structures (RegistrationInfo.java) have been specified *a priori*, and can be downloaded from the class website. The Presence Service will run as a separate process (as will the rmiregistry service upon which it depends) and need not be on the same machine as the clients. The Presence Service is considered to be application infrastructure and only one instance will be present in the system.
2. ChatClient - The Chat Client is the application that end users will run when they wish to see who is on-line, and/or have on-line discussions with other users. The ChatClient will use Java RMI to determine the location of other ChatClients, but will exchange messages directly with other clients via Java Sockets.

Your implementation will simply have a text interface, like the early "talk" program on Unix that existed long before conventional IM solutions. It's also simple in the sense that we won't be supporting any sort of friend list, or authentication. In other words, every user becomes visible (and hence a friend) of every other user on the system as soon as they successfully register. There is also no user authentication, and user names are available on a first-come-first-served basis. When a user leaves the system, another user could claim the name.

The interactions between the components in Figure 1 are straight forward. When a user executes a ChatClient process, they will be expected to conform to the following conventions:

```
java [necessary java flags go here] ChatClient {user} [host[:port]]
```

where user refers to the name of the user invoking the client, and host:port refers to the host and port number associated with the RMI registry service that will bind the client to the presence service. The host and/or port parameters are optional, and if left unspecified the client will attempt to bind to a RMI registery service on the localhost and port 1099. When using EOS machines, to avoid conflicts, students are encouraged to stake out their own user level RMI port and run their registry instance on that rather than the default ports.

Upon startup, the ChatClient will register its presence with the Presence Service. If a user already exists with the given user name, the appropriate message should be printed, and the client should exit. The ChatClient will implement a very simple command-line-interface that interprets and executes six commands.

- friends – When this command is entered, the client determines (via the Presence Service) which users are registered with the presence server and prints out the users' names and whether they are available or not available (via the status field in the registration information).
- talk {username} {message} - When this command is entered, the client 1) first checks to see if the user is present *and available* (via the PresenceService). 2) If the user is registered and available, a connection to the target client is established,

based on their registration info, and they are sent the given message. Note that when a client receives a message, it will simply print it out to the console, and re-prompt the user to enter his/her next command. (i.e. you will need multiple threads of execution here within your client process.)

- broadcast {message} – The client broadcasts the message to every user that is currently registered and available. Clients should NOT broadcast the message to themselves.
- busy – The client updates its registration with the presence server, indicating it is not currently available. If the client is already in not available when this command is entered, nothing needs to be done, though it would be good to prompt the user and indicate they already are not available. A client that is busy should not receive any messages whether they be sent with the talk or the broadcast command.
- available – The client updates its registration information with the presence server, indicating it is now available. If the client is already available when this command is entered, nothing needs to be done, though it would be good to prompt the user and indicate they are already registered as available.
- exit – When this command is entered, the ChatClient will unregister itself with the PresenceService and terminate.

The Presence Service will conform to the following startup conventions:

```
java [necessary java flags go here] PresenceServiceImpl [port]
```

where port refers to the port the rmiregistry process is started on.

Deliverables

There are two deliverables required in order to receive full credit for this lab exercise.

Provide a Demo to Instructor

Provide a demo of the working system to the instructor during office hours before or on the designated due date.

Electronically Submit the Source Code

Archive all the source code needed to build and execute your implementation in a zip file and email it to the instructor. The timestamp on your submission must be no later than the due date to receive credit.

References

A good Java RMI tutorial can be found at:

http://download.oracle.com/javase/tutorial/rmi/index.html

If you feel a need to brush up on Java you can find a number of additional tutorials at:

http://download.oracle.com/javase/tutorial/index.html

If you are using the Eclipse IDE for your development, the RMI Plug-in for Eclipse is a handy component to install in your environment:

http://www.genady.net/rmi/index.html

You can get a free eval license, but the GVSU School of CIS already has a site license that you can use. The key for that license is:

oi14z0qkzeqlg9gb210mfojpcsrxmhlvpq2gy0jvdldw5849xg9nb2x
vr1sk02i9qoy804z4pwufsoyefgk04h5vz529q3yrxa1yogdpyqfolr
uhp3tmk9h12baa6bsw3mhe5h49n7r0ubir1qm8pmtfembu1b6bpk856
lwewpbi6quj4rfcvvc604db4q4gsqixcqszw5l896wuzc1ewr7utdpt
3o9263vm6xvu1njn9dxoxe6t8eygp0hcf9yhmrq6z2yrh3bjwi52cqw
45cri0r14ds66ftyfz5r10syc46s84b2ii

(Follow the instructions on the website for installation/configuration).