Brian Vaesa, Christian Wilson

CNT4504 – Networks

4-17-2017

**ABSTRACT**

This project is based on, and makes use of the development of distributed application paradigms. It deals with the socket based approach (using TCP/IP) and with the RPC/RMI based approach (depending on the machine language chosen, C for RPC and Java for RMI). The goal of this project is to see how each side handles message passing through a process which sends a message that represents a request and then waits to receive back a response. The experiment conducted was to run three repetitions (back to back) of each request available on the provided client-side menu that only prints out the final [time] result (in milliseconds). The mean response time will then be calculated and displayed for the socket based approach and for the RPC/RMI approach.

**INTRODUCTION**

This client-server project deals with a socket based approach and a Remote Method Invocation approach (RMI). Sockets are a low-level interface created by a client to initiate a request to a server while the server must be prepared to listen for incoming requests in advance. This connection-oriented protocol allows two applications to send data back and forth keeping the connection open even if there is no communication. TCP/IP is used because Java only supports the Internet Protocol (IP) and the socket class only speaks Transmission Control Protocol (TCP). As for Remote Method Invocation, it allows an object residing in one system to access/invoke an object running on another system. RMI is used to build distributed applications­­; it provides remote communication between Java programs.

For both approaches the results will provide current system statuses from the server as requested by the client. The most common options are the system date and time, the number of running processes, the number of active socket connections, the time of the last system boot, the current users on the server, and an echo back from the server on what was sent from the client. These options/choices can be chosen from a text menu displayed to the user along with the option to quit. The goal of utilizing both the socket based approach and the RMI approach is to determine which one is better suited for speed.

**DISTRIBUTED APPLICATION DEVELOPMENT PARADIGMS**

A distributed application is one in which the application processing is divided among two or more machines. This division of processing also implies that the data involved is also distributed. Distributed application technologies such as RMI have evolved to meet requirements of the enterprise.

When using sockets, it provides the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to the server end. When the connection is made, the server creates a socket object on its end of the communication. After the connection is made, the client and the server can now communicate by writing to and reading from the socket.

There are two useful classes used, java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them. This is how communication occurs using sockets: The server instantiates a ServerSocket object, denoting which port number communication is to occur on. The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port. After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to. The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket. After the connections are established, communication can occur using I/O streams. Each socket has both an InputStream and an OutputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client). Inside the server program, a remote object is created and a reference of that object is made available for the client (using the registry). The client program requests the remote objects on the server and tries to invoke its methods. When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the client-side. When the client-side receives the request, it invokes a method and passes the request to the server side. The server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server. The result is then passed all the way back to the client. The main goals of RMI are to minimize the complexity of the application, to preserve type safety, distributed garbage collection and minimize the difference between working with local and remote objects.

**RESULTS AND COMPARISONS**

These results are the mean times (average) in milliseconds (ms) of running the processes three times in a row only showing the completed process times for each MC as we commented out the print statement for each iteration. Our tests were conducted wirelessly in Atlantic Beach, FL in the evening using a Linksys WRT1900ACS router on the Comcast/Xfinity cable network. From the results, you can see that all but MC6 were faster through RMI than through sockets. Additionally, MC4’s times were the closest in range during the three test runs. The reason for this is because RMI is an application of sockets used to transmit messages under a structured set of rules and conditions so that the underlying workings of the connections don’t have to be worried about.

|  |  |  |
| --- | --- | --- |
| MC’s | SOCKET | RMI |
| 1 - Server current Date & Time | 4105ms | 719ms |
| 2 - Number of running processes | 4781ms | 2015ms |
| 3 - Number of active socket connections | 5763ms | 5525ms |
| 4 - Server time of last boot | 4064ms | 502ms |
| 5 - Server current users | 4064ms | 517ms |
| 6 - Server echo back what is sent from client | 4ms | 507ms |

**CONCLUSIONS**

In conclusion, one can see that based on the results of the tests that RMI is pretty much the clear choice for distributed applications (if the application is written in Java) as it has all the low-level socket needs built in making it an overall faster choice and a bit easier to learn. The only major difference between the two applications was with the server echoing back what is sent from the client; those time differences where extremely shorter via sockets than RMI. In reality the needs of the business you’re in will determine whether sockets or RMI will work best for you.

**REFERENCES**

[1] A. Campeau, "Untitled", *Sce.carleton.ca*, 2017. [Online]. Available: <http://www.sce.carleton.ca/netmanage/simulator/rmi/RMIExplanation.htm>. [Accessed: 19- Apr- 2017].

[2]"Computer Networking - Lesson 6: Introduction to Client/Server Network Setup", *Functionx.com*, 2017. [Online]. Available: <http://www.functionx.com/networking/Lesson06.htm>. [Accessed: 19- Apr- 2017].

[3] P. Niemeyer and J. knudsen, "11. Network Programming with Sockets and RMI [Book]", *Safari*, 2017. [Online]. Available: <https://www.safaribooksonline.com/library/view/learning-java/1565927184/ch11.html>. [Accessed: 19- Apr- 2017].

[4]"Lesson 1: Socket Communications", *Oracle.com*, 2017. [Online]. Available: <http://www.oracle.com/technetwork/java/socket-140484.html>. [Accessed: 19- Apr- 2017].

[5]"Java Networking", *www.tutorialspoint.com*, 2017. [Online]. Available: <https://www.tutorialspoint.com/java/java_networking.htm>. [Accessed: 19- Apr- 2017].

[6]"Java RMI Tutorial", *www.tutorialspoint.com*, 2017. [Online]. Available: <https://www.tutorialspoint.com/java_rmi/index.htm>. [Accessed: 19- Apr- 2017].

[7] Ibrahim, Taneem (2004) "A Comparative Evaluation of .net Remoting and JAVA RMI," Inquiry: The University of Arkansas Undergraduate Research Journal: Vol. 5 , Article 12. Available at: <http://scholarworks.uark.edu/inquiry/vol5/iss1/12>

[8] I. Rozman, M. Juric, I. Golob and M. Hericko, "Qualitative and quantitative analysis and comparison of Java distributed architectures", *Wiley Interscience*, vol. 36, no. 17, pp. 1543-1562, 2006.