Brief Explanation of the Implementation and Tests for RMI Coursework

Fayimora Femi-Balogun

November 26, 2013

The first section of this document provides an explanation of the code, mainly what it does and how it does it. The second section describes findings from testing ltency and bandwidth between the client and server.

1 Implementation

The code comprises of 7 source files. The main source files include:

1. **BankServerImpl:** This is the main server class that implements the given interface, `BankServer`. This class provides it's own main method from which it binds to a location for a client to connect to. It also exposes a few methods to the client for manipulating an account. They include `int deposit(int amount)`, `int withdraw(int amount)`, `int balance()` and `int getAccNo()`.

2. **AccountFactory:** This class provides two static methods. They include `BankServer newAccount()` which creates a new account for a client and `BankServer getAccount(int accountNumber)` which returns a setup for the client. The methods are static because we do not wan to have multiple instances. Every client should use the same factory. This class also keeps track of all open accounts while the server is running. If an account cannot be found, a unique `AccountNotFoundException` is thrown.

3. **Client:** This class makes use of the classes above to run user operations and create/setup accounts, respectively. It also has a nice command line user interface. An example run of the client can be found in the submitted zip archive.

2 Testing the server

In order to test the time it takes the server to respond, I created a class called `TestRMI` which connects to the server and times a few operations. I made use of `System.nanoTime()` provided in the Java library to test the latency. I saved

the time before and after the operation, then took the difference. The results were in nano seconds.

I noticed that when performing the same operation on the server, the time it takes does not really vary much. This was something I expected. However, when I increased the complexity of the computation as well as the size of the data being returned, my system froze. Initially, I thought this was due to the complexity of the function but running the function on it's own, that is, without returning the result to a client over a network, did not freeze the computer. This might be because I was not properly setup in someway or there was a problem while transmitting the very large result over the network.