Lab 2: Remote Method Invocation

Kota, Yasaswy
ykota@andrew.cmu.edu

Venkataraman, Adwaith
adwaithv@andrew.cmu.edu

1 Implementation

Remote Method Invocation is a framework where we can remotely invoke methods and get the results for the said methods without having to spend computational power ourselves. This can be done by having a simple Server - Client setup. The Server will host a number of different classes and their respective methods. Whichever classes the Server wishes to expose to others will be registered in a *registry*.Each registry is associated with only one server. This *registry* will contain all the necessary information about the class so that it's methods can be invoked remotely. This information is called the *RemoteObjectReference*. In our implementation, the *RemoteObjectReference* contains the host name, the port number and name of the interface it implements. When the Client needs to invoke a remote method. It queries all the registries it knows and sees if any of them have the required method. If there is a hit, it gets the *RemoteObjectReference* and generates a *ProxyObject* using it.

A *ProxyObject* is a object that is associated with a *InvocationHandler*. This *InvocationHandler* is called whenever a method of a *ProxyObject* is called. This *InvocationHandler* acts as the middleman between the Server and Client. It gets the class name, the method name and arguments of the method that calls the *InvocationHandler* in the first place and sends that across a network to the server that has the definition to the method. The host name and the port number of the server is got by the *RemoteObjectReference*. Once the Server finishes the computation, it sends the result back to the requesting Client. This result is got by the *InvocationHandler* and sent to the object that caused the *InvocationHandler* to be called. So the *ProxyObject* thinks that a normal method call gave that result. According to the *ProxyObject*, it the made a local call.

The *.class files are not explicitly transferred over the network, assuming that we use the andrew file system.

No known bugs.

2 Compiling and Building

To compile and build, *cd* into the folder containing the **.java* files, the *RMI* folder. In terminal, type the following commands:

```
1 $ make clean
2 $ make
```

This will produce **.class* files which is used for running the Client and Server. A user *must* start the Server first, then the user can start the Client. To start the Server, type the following into the terminal:

```
1 $ ./run_server.sh PortNumber
```

To start a Client:

```
1 $ ./run_client.sh ServerHostName ServerPortNumber
```

3 Usage + Examples

There are a total of 6 remote methods that we've written as examples. These belong to 2 classes, the *Calc* class and the *Case* class. The *Calc* class has 4 arithmetic methods and the *Case* class has 2 String methods.

The available methods are:

- add
- sub
- mul

- div
- upper
- lower

The Client program also has a *help* option to aid the user to utilize RMI. We can invoke the methods as follows:

```
$ > add 1.3 4.1
5.4
$ > sub 3.9 - 6.9
-3
$ > mul -8.9 -1
8.9
$ > div 12 3
4
$ > upper ThIs Is a test
THIS IS A TEST
$ > lower ThIs Is a test
this is a test
```