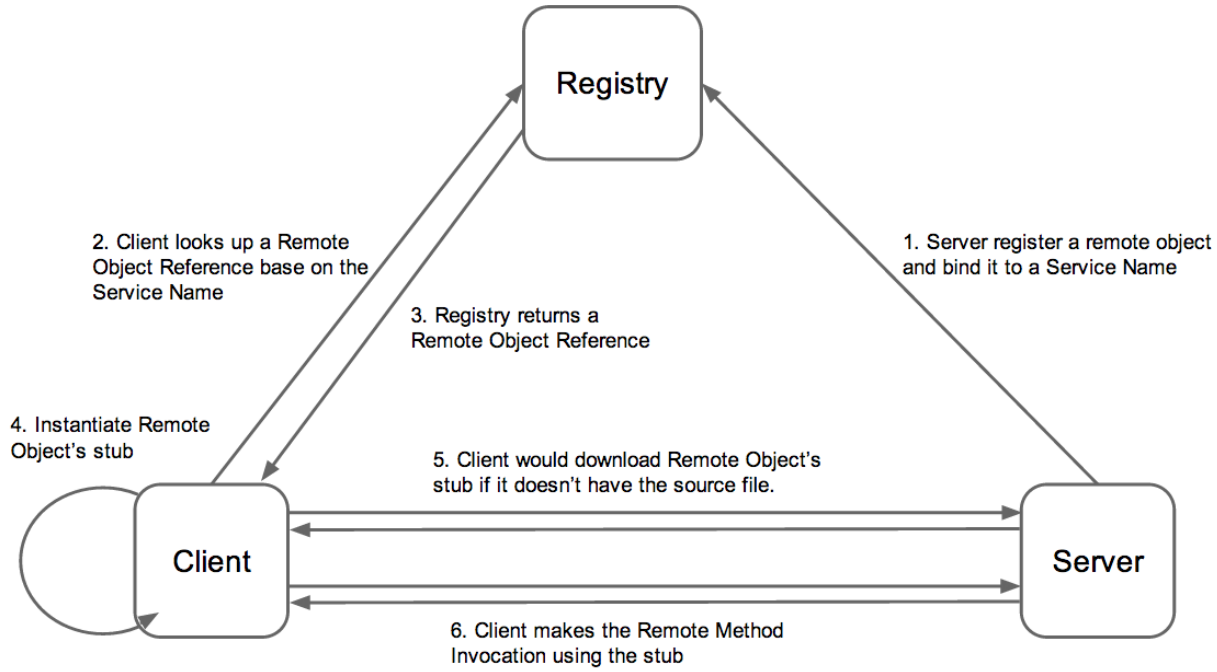
15-640 Lab 2

Yin Xu (yinxu), Zheng Kou (zhengk)

1. **Design**:



Our goal is to implement a system that offer the ability to let one Java Virtual Machine (JVM) invoke method on objects that on another but network accessible JVM.

Registry:
Registry maintains a hashmap of Object Service Name and Remote Object Reference. Server can create an object and bind a Service Name to it, then register the entry into the Registry. On the other side, client is able to look up the Remote Object Reference using the Service Name, so that it can make the remote method invocation later on.

Client:
Client should provide an interface to end user which encapsulate the communications between the registry and the server. In other words, the end user only need to two things to invoke a remote method: 1. Instantiate a remote object's stub using the Service Name; 2. Make the remote method invocation using this stub.

For task 1, client first get a Remote Object Reference from the registry which contains the remote interface name to instantiate the stub, server ip address and port number to connect to and URL to download stub file if needed. Secondly, client can download the source file, if needed, from server and then instantiate the stub locally.

For task 2, client create a RMI Message containing the Remote Object Reference and

marshalled arguments and send it to the designated server. Then get the return value from the server.

Server:
The server offers three functionalities:
1. Bind the Remote Object Reference to a Service Name, then register them to the registry
2. Receive RMI Invocation message, invoke on remote object and send back the return value using the original message.
3. Provide stub file downloading service to client.

2. **Implementation:**
RMIMessage:
There are basically three kinds of communication in our system: 1. Server and Registry; 2. Client and Registry; 3. Client to Server. So we decide to use RMIMessage as the carrier of the all three kinds of communications: 1. RMIMessageReg; 2. RMIMessageLookup; 3. RMIMessageInvoke. All three kinds of messages inherit the same superclass, RMIMessage, which contains exception and remote object reference to avoid code duplication. Further, we can improve the robustness of our system to use RMIMessage objects as carrier instead of Strings.

RMINaming:
RMINaming runs on behalf of client and encapsulate the communications between the registry and the server. It first lookup the remote object reference from the registry and then localise the stub based on the interface name.

Client Side Socket Cache:
Since it is very likely that client and server talk frequently to each other, it would be essential to maintain a socket cache to avoid resource operation (unnecessarily open and close socket and input/output streams) overhead. We use a HashMap to implement the socket cache in which socket ip address and port number as the key, socket itself and input / output stream as the value of hashmap. If the socket cache reach the maximum size, we kick out the least recently used socket and close this socket before we add new socket into the cache.

Server Side Socket Cache:
Server doesn't maintain any data structure for socket cache. Instead, it won't close the socket until the server thinks the client associate with that socket has been inactive for a certain amount of time.

The Automatic Retrieval of .class Files for Stubs:
If the client doesn't have the necessary .class files for stubs, it will catch the ClassNotFound Exception, and then automatically make a HTTP request to the server based on the stub URL contained in the RemoteObjectReference object. On the server side, when its communication module starts, a DownloadThread also begins running and listening for the class file download

request.

3. **Instructions (Build, Deploy and Run)**:

- **Build**
  - go to the src/ folder
  - make clean
  - make build
- **Deploy**
  - start the registry first
    make runreg port=<input port number of registry>
    example: make runreg port=15441
  - start the RMI server
    make runser port=<port number of registry>
    example: make runser port=15441
- **Run**
  - run the client program
    make runcli host=<registry hostname> port=<port number of registry>
    example: make runcli host=blueshark.ics.cs.cmu.edu port=15441

4. **System requirements:**

Java Virtual Machine

5. Test examples:

We only implemented one remote object and one method in this object. The only method is to take the String argument and return a String that attach “Hello hehe “ before the input argument.

Example 1: Ordinary Test

Make the remote method invocation ten times in a row. And print out the result on console.

Example 2: Server Timeout Test

We set the server time out time to be **one second**, the sleep time between each (and the same) remote method invocation to be **three second.** So after each RMI, the server will close the socket and the socket in the client side socket cache will be invalid. This example behave as server side socket expiration.

Our system shows client will reconnect the socket and makes the RMI run normally.

Example 3: MyRemoteException Catch Test

We send “exception” to the server side method invocation, and hard-coded throw a MyRemotException to test that the system is able to successfully handle remote exception.

Test Downloading .class file for stubs

- in src/ folder, start the registry and the server (if they are running, no need to restart)
  make runreg
  make runser
- go to DownloadTest/bin folder. The stub class has been deleted here.
- run the client
  make runcli
- The client program will automatically download the stub class file then continue the remote invocation.