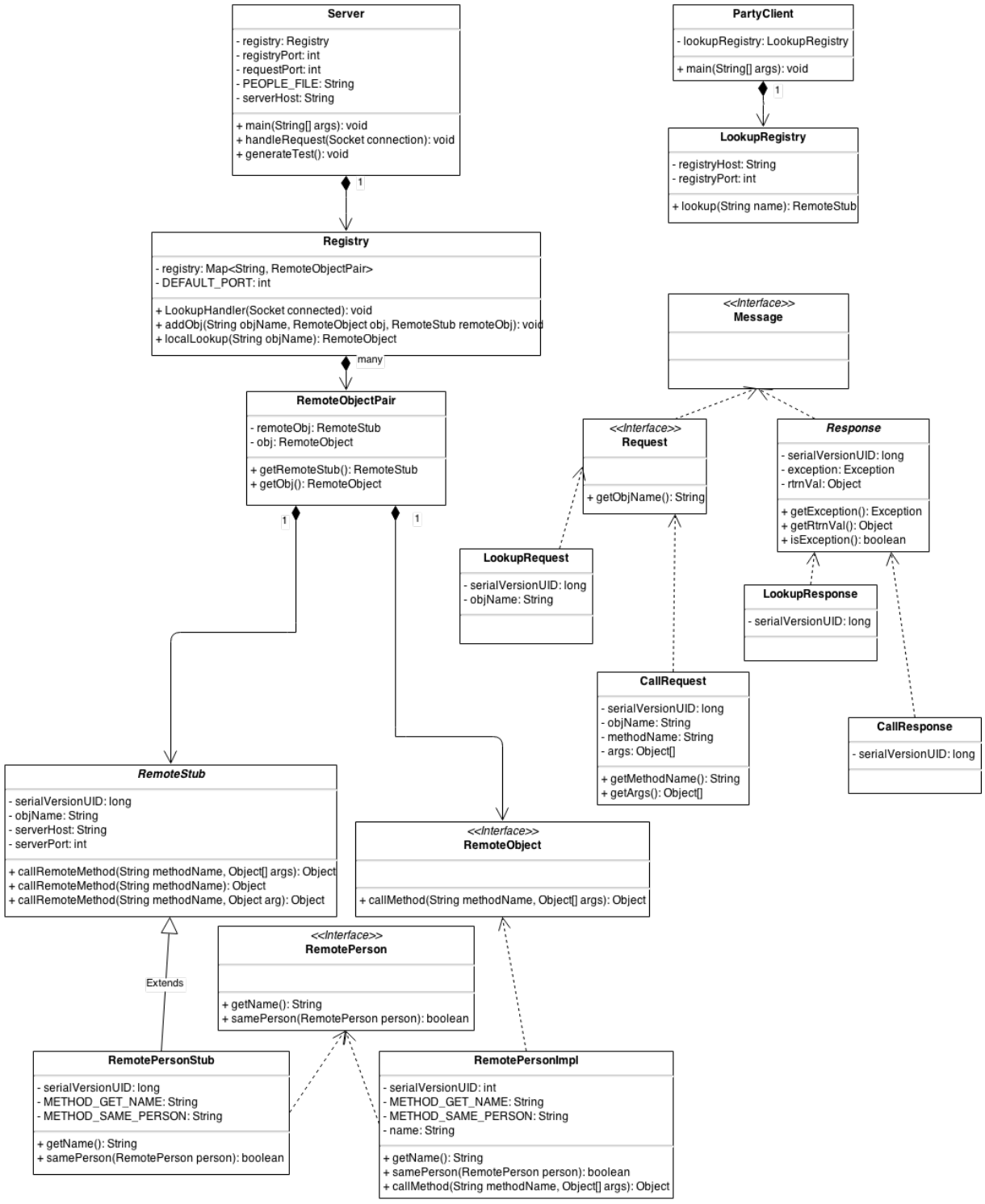
15-440: Lab 2

Spencer Barton (sebarton)
Emma Binns (ebinns)

September 11, 2014

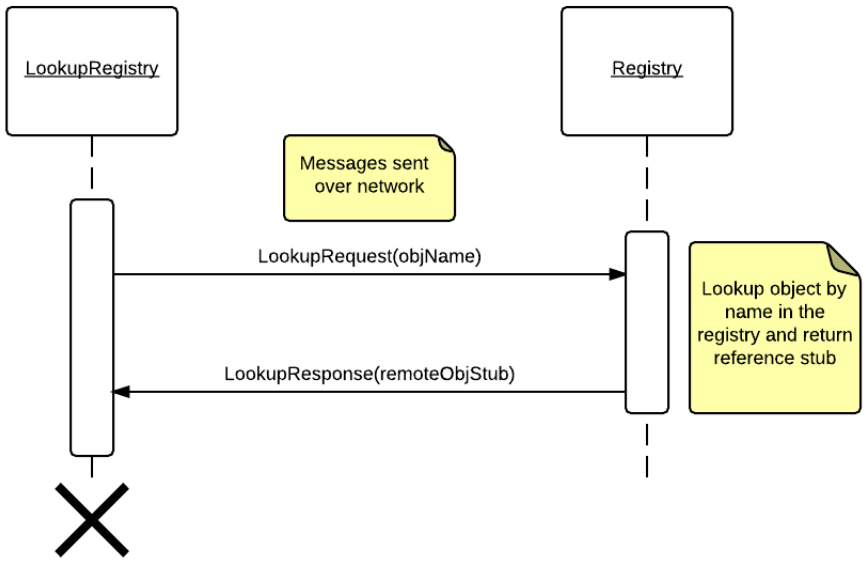
1 Design

Here is a UML diagram to illustrate our design:



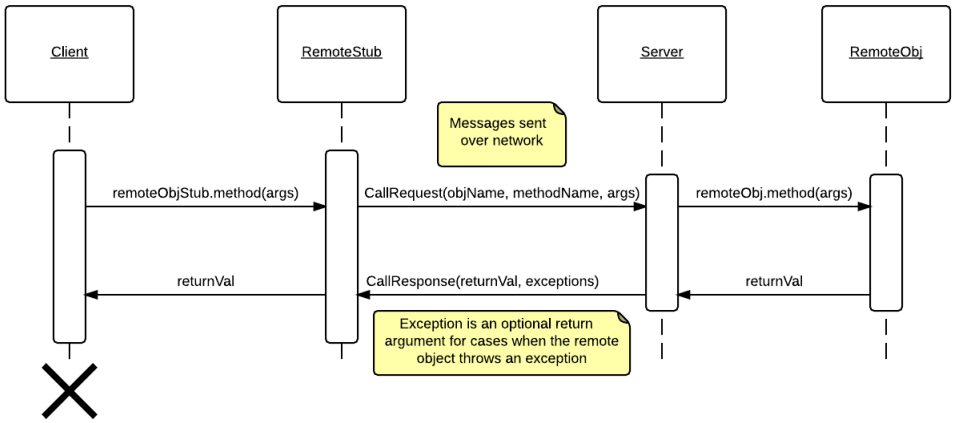
We decided not to implement any server skeletons and instead to incorporate the functionality of a dispatcher into our server. We chose to do this because, as stated in the handout, the process to handle remote method calls is not dependent on the specific object the method is being called on. Therefore, it made more sense to us for our server to include the functionality of a dispatcher rather than having each stub require a corresponding skeleton.

Registry Lookup



In order to lookup a remote object the client uses their `LookupRegistry` to lookup an object by name string. The `LookupRegistry` creates a `LookupRequest` with the object name. This is sent to the server which looks up the object by name in its registry. If the object exists, a `RemoteObjectStub` is returned through a `LookupResponse` else the `LookupResponse` comes back with an exception.

Remote Method Invocation



Remote object invocation begins with the client calling a method on a `RemoteObjectStub`. This call is then abstracted and a `CallRequest` message is generated with the object name, method name and arguments list. This message travels over the network to the server where the actual object lives. The relevant method is call and the return values is sent back to the client and returned from the original stub method invocation.

2 Implementation State

We completed the basic requirements.

The extras that were not added include a skeleton, compiler and loading class files. We chose not to implement skeletons as we were able to add that the necessary methods within `RemoteObject`.

In order to add a compiler the design would not need to change. When performing a registry lookup a remote object stub is returned. Currently this remote object stub must be instantiated earlier by the server. With a compiler this would simply be replaced by a call to the compiler during an object lookup.

Finally support for loading class files would require a little more work since another server socket would need to be implemented on the host the .class files. Beyond that the remote object stub would handle a .class request and another type of message could be implemented to handle this communication.

3 Running the Project

Here are the instructions to run our project on the Andrew machines using the provided example client/remote object code for testing:

1. Navigate in a terminal to the directory containing our project, then navigate to the "src" folder within our project.
2. Run the command: `python SETUP.py`
   (This command should print out commands to run the server and client.)
3. From this "src" folder, run the server using the command: `java server/Server`
   (This command should print out: `Server online at <hostname:port>`)
4. In another terminal window (connected to a different server or the same one, whichever you prefer), navigate to the "src" folder as you did before. Then run the command: `java client/PartyClient <hostname>` where `<hostname>` is the hostname printed out when you ran the server above (this is the hostname corresponding to where you're running the server; ex: unix1.andrew.cmu.edu)

When you run the client, several lines should print out, both in the server window and the client window. These are the results of built in tests we've written that run automatically when you start the client. To check that our project works, we've provided thorough print statements of what is occurring at each step of the process. If our project functions correctly, this is what should print out:

In the client window:

```
Creating local people

Getting remote people
Sending lookup request from client to registry...
Lookup request result:  <specific RemotePersonStub object>
Sending lookup request from client to registry...
Lookup request result:  <specific RemotePersonStub>
Sending lookup request from client to registry...
Lookup request exception:  java.lang.NullPointerException
Good - unable to find non-existant remote person

These people are at the party:
Remote method call:  calling getName on elain
Remote method call result:  elain
Remote method call:  calling getName on robert
Remote method call result:  robert
sally ben jim robert elain robert

Remote method call:  calling getName on robert
Remote method call result:  robert
Was Robert double counted (should be true)?  true
Remote method call:  calling samePerson on robert
Remote method call result:  true
To check again, was Robert double counted (should be true)?  true
Remote method call:  calling samePerson on robert
Remote method call result:  false
Are Elain and Robert the same person (should be false)?  false

Remote method call:  calling samePerson on robert
Remote method call exception:  java.lang.NullPointerException
Good we got an exception null
```

In the server window:

```
Server received registry lookup request
Registry is performing lookup.
Registry lookup:  elain
Server received registry lookup request
Registry is performing lookup.
Registry lookup:  robert
Server received registry lookup request
Registry is performing lookup.
Registry lookup exception:  java.lang.NullPointerException
Server received method call request
Remote method call request:  elain.getName(null)
Server received method call request
Remote method call request:  robert.getName(null)
Server received method call request
Remote method call request:  robert.getName(null)
Server received method call request
Remote method call request:  robert.samePerson(<specific Object>)
Server received method call request
Remote method call request:  robert.samePerson(<specific Object>)
Remote method call:  calling getName on elain
Server received method call request
Remote method call request:  elain.getName(null)
Remote method call result:  elain
Server received method call request
Remote method call request:  robert.samePerson(<specific Object>)
```

4 Dependencies

There are no dependencies.

5 Testing

Our test in built into `PartyClient`. It automatically runs when `PartyClient` is run. The test runs each of the methods in the `Person` object for both a remote and local implementation. `Person.samePerson` is run using both remote and local objects as arguments to check that the server-side remote object implementation can handle either object type as an argument.