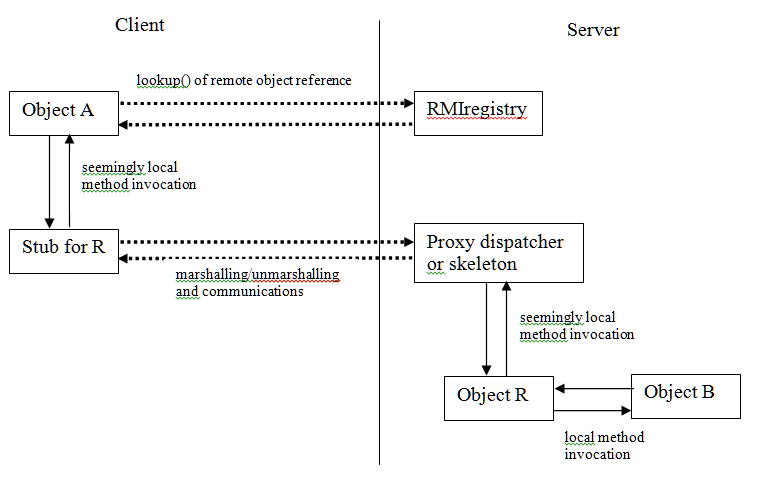
Project #2: Design and Implementation of a RMI Facility for Java

1. Clearly explain your design and illustrate its use, being sure to highlight any special features or abilities
   1. RMI Registry: The RMI registry maintains the remote reference information for the proxy dispatcher and for the SampleClient. The RMI registry will allow the proxy dispatcher to bind an object so that when the sample client looks up the remote reference with a “key name” it will provide the information which is needed. The information of the remote object will be packed in an object called RemoteRef. This will contain the class name, interface name, lookup name and the necessary IP address and the port number of the proxy dispatch. With this information the Proxy stub compiler will be able to create the stub object and remotely execute methods. The RMI registry will handle three types of actions which are “bind, rebind, and lookup”. The rebind will occur if there are registered remote references the in RMI registries with the same “key name”, if this occurs the RMI registry will send out an exception so that the Proxy dispatcher could send out the rebind signal. If the look up happens from the Sample Client the RMI registry will send out the RemoteRef back to the Sample Client
   2. Proxy Dispatcher: The proxy dispatch will perform two main actions. The first action will be to bind the remote reference objects to the RMI registry. The second action will be to handle the remote reference methods calls from the Simple Client. After the proxy dispatcher executes the method call, it will send back the results to the Simple Client.
   3. SampleClient: The sample client will connect to the RMI registry and looking up the remote reference. If there is a bind reference it will use the proxy stub compiler to create the object and perform actions.
   4. Proxy Stub Compiler: The stub compiler will use the information from the look up and generate the proxy object. (Add in more information )
   5. Signals
      1. Ack Look up Signal: This signal will be an acknowledgement from the look up signal. This will contain the remote reference information which will be used in the proxy stub compiler.
      2. Ack Signal: This signal will be an acknowledgement for the bind, rebind signal.
      3. Bind Signal: This signal will contain the remote reference information from the proxy dispatcher.
      4. Loop Up Signal. This signal will be a look up request from the Simple client.
      5. Rebind Signal: This signal will trigger a rebind on the RMI registry.
      6. Remote Exception Signal: This signal will handle the exceptions from the remote locations. If an exception is triggered the remote exception signal will be created with the exception information.
      7. Invoke Method Signal:
      8. Invocation Response Message:
   6. Helper Utility: This was design to handle the communication. It will have two methods which will be send and receive. This will be used in all components so that we will have a consistent method to communicate.
2. Describe the portions of the design that are correctly implemented, that have bugs, and that remain unimplemented.
   1. The design for the project two, we followed the basic idea that was introduced to use by in the project description with modification. One major this that is a bit different is that we separated the RMI registry from the server. To start the whole system, we need to boot up the RMI registry first. Then start up the proxy dispatcher, as the proxy dispatcher is started it will connect to the RMI registry and bind the necessary remote references. After binding everything, it will start to accept connection from the client so that it could execute the remote invocations. For the client side, it will start up and reach out to the RMI registry looks up the necessary remote reference so that the remote reference information could be used in the stub compiler. We are assuming that the interface file will be present on the client side so that we could initialize the interface and pass it to the stub compiler.



* 1. Not implemented
     1. Relook up from the proxy object is created: If they are connection exception while executing the remote methods, we do not have the ability to trigger another look up so that we could recreate the proxy object.
     2. Garbage collecting on the RMI registry: The RMI registry does not have the functionality to validate the liveness of the remote reference. The remote reference will only be updated with the rebind signal if the proxy dispatcher rebinds with the same “key name”.
     3. Automatic retrieval of the interface for the stub compiler on the client side.
     4. Distributed garbage collector

1. Tell us how to cleanly build, deploy, and run your project (need more work)
2. Highlight any dependencies and software or system requirements.
   1. There are no dependencies in the program. The only requirement will be the java 1.70 and the open ports for the connections.
3. Tell us how to run and test your framework with your two examples. (need more work)
   1. Happy path
      1. Start up the RMI registry following with a proxy dispatcher.
      2. Execute the simple client and valid the result.
   2. Rebind testing
      1. Start up the RMI registry with one proxy dispatcher and execute the simple client.
      2. Notice that the first proxy dispatcher will react to the remote method call.
      3. Start up another proxy dispatch. Notice that they will be an exception while binding and the second proxy dispatcher will send out the rebind.
      4. Start up a simple client again notice that the second proxy dispatcher will react to the remote method call.