Research and Analysis (Remote Method Invocation: Mechanisms)

Navjot Singh, Email: Virksaabnavjot@gmail.com, National College of Ireland, Mayor Street, Dublin 1, Ireland

Introduction

A Distributed system *(DS)* is a model/collection of independent computers linked together through a network which produce an integrated computing facility using software programs *(middleware)*. Some widely-used DS are Word Wide Web *(WWW)*, Email, Cloud services like Google drive and Dropbox, teleconferencing services like Skype. In Distributed communication, services provided by a server can be accessed by multiple clients. "The goals in mind while implementing distributed systems are – scalability, reliability, openness, transparency and performance" (Lee , 2007). Inter-process communication is at the core of DS, and there are different ways to achieve that for example - Message Oriented or Stream Oriented communication but for this research report will focus primarily on Remote Method Invocation *(RMI)* which is a Java implementation of Remote Procedure Calls *(RPC)* that allows server and client software to communicate with each other.

Let's, look at high level details about RMI: Remote Method Invocation is an object-oriented Application Programming Interface which allows the creation of distributed applications using Java and this distributed environment supports/allows different computers running Java Virtual Machine *(JVM)* to communicate with each other using stub *(on client side)* and skeleton *(on server side)*. Stub and Skeleton are responsible for marshalling and un-marshalling data, the RMI allows an object on client to invoke methods/services on an object on the server running JVM. RMI does this in a way that the client application thinks its invoking a local Java object's methods.

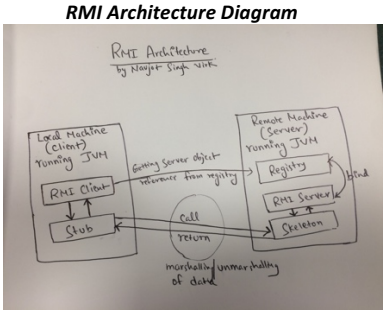
There are several RMI alternatives which can be used but Common Request Broker Architecture (CORBA) is a serious competitor to RMI that can also be used to create distributed systems using Java. For this report, we will not study CORBA in detail but some important points may need to be addressed.

CORBA wins from RMI in terms of performance but RMI is easy to implement and communication is straight forward through RMI's wire level protocol, the original, Remote Java Object communication in RMI occurs using the Java Remote Method Protocol *(JRMP).*But in CORBA communication happens over Internet Inter-Orb Protocol *(IIOP)* and it is necessary to keep in mind IIOP stubs are properly connected with ORB (Object Request Broker) before starting operations on IIOP stub whereas this is not required with JRMP. Java implementation of CORBA/IIOP is known as Java IDL *(Interface Definition Language)* and supports mapping for Java. Which is Java IDL, helps to define, implement and access CORBA object using Java. Previously, Java developers had to choose between RMI and Java IDL.

These days, RMI-IIOP is widely used to bring CORBA capabilities to Java platform with some limitations. RMI over IIOP provides Java developers the freedom to write CORBA applications without learning CORBA IDL.

So, far we understand that RMI allows communication between remote computers. And we also looked at RMI alternative CORBA. Well, it's hard to say one is better than the other. *Now, let's look at RMI Architecture Diagram and RMI mechanisms that allows back and forth communication between server and client.*

RMI Architecture Diagram



Mechanism involves generation of stub on the client-side which takes care of marshalling of parameters *(data)* and passing arguments to Skelton over the network when invoking a remote object method and un-marshal's data when receiving data from the skeleton which is generated on the server-side and is responsible for un-marshalling incoming arguments and marshalling results when returning values from the server object and passing it to client stub over network.

Next up let's look at some advantages and drawbacks of Java RMI –

Advantages	Drawbacks
• Easy implementation and portable across many platforms.	• Only available for platforms with Java support. Whereas CORBA, is language independent.
• Automatic Marshalling and un-marshalling, puts developers at ease.	• "Can be slower especially when compared to CORBA." (iupui.edu, Slide 56)

• Dynamic Interface creation is possible.	• "Remote methods are synchronous, which can cause problems when the network is down and the application may freeze." (DevX, 2000)
• Remote Objects look like they are local objects.	• "Security threats with remote code execution, and limitations on functionality enforced by security restrictions." (Reilly, 2006)
• "Ability to introduce new code to the foreign JVM's." (Reilly, 2006)	• No support for legacy systems

So far, we know the basics, advantages and drawbacks of RMI. Next, I will share my views, opinions and experience with RMI.

Discussion

As we know now, RMI is Java implementation of RPC, and it allows us to bring distributed system capabilities to our Java applications. Under the Introduction heading, we talked about Java IDL *(CORBA/IIOP)* and RMI-IIOP which can be used to expose Java Objects to CORBA ORBs. But for this report, we will focus on the original/basic RMI communication through JRMP used for Java to Java remote calls, and requires the client and server to use Java Objects.

How remote object invocation is possible? Any object that can be invoked remotely needs to implement Remote Interface *(serves to identify interfaces whose methods can be invoked from remote JVM).*

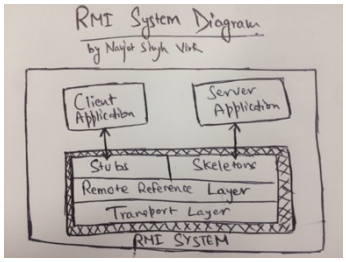
As a developer with 4 years' experience working with Java, I believe RMI put you at ease as it is easy to implement, RMI is a part of the Java core platform and *java.rmi* provides the RMI package since JDK 1.02. RMI's object-oriented nature makes it powerful. RMI can pass full complex objects as arguments *(like a hash map)* and return values unlike in some existing RCP implementations, where the client would have to dissolve the object into primitive data types then transfer and re-assemble the object on the server whereas RMI out of the box allows you to transfer objects across the wire with no extra client code putting developers at ease.

But the experience for other people may differ based on different factors like their understanding of Java language, goals and needs because it may or may not be efficient to implement RMI for creating your own distributed applications for that you will need to list out your needs and requirements including security and speed and others. RMI uses Java's security principles and when compared to CORBA it may not be the best choice when aiming for performance. Now, we know that RMI may not always suit all your needs and other alternatives like CORBA are available. Looking at if it suits your requirements how it will help - RMI allows to access remote objects in a distributed environment as if it's a local object, and pass more than one objects with the request and it automatically takes care of things like marshalling and un-marshalling so the developers don't have to deal working and writing complicated mechanisms to transfer data over the network.

Let's understand the mechanisms involved in RMI –
Basics: Server program creates remote objects and makes these objects accessible for clients to invoke methods on these objects and the Client program obtains a remote reference to remote object/s and invokes methods on them.

- Defining and Implementing Remote Interface
- Generation of Client Stub – stub translates calls from caller object and initiate communication towards server skeleton.
- Generation of Server Skeleton – skeleton is server side counterpart of stub and is responsible for accepting calls from stub.
- Starting RMI Registry – "A remote object registry is a bootstrap naming service that is used by RMI servers on the same host to bind remote objects to names. Clients on local and remote hosts can then look up remote objects and make remote method invocations." (rmiregistry, Oracle Website).
- Marshalling and Un-marshalling (at Client Stub and Server Skeleton) – marshalling the process of converting data into byte-stream and un-marshalling is the opposite converting byte-stream into arguments.

Diagram showing the 3 layers of RMI System (next page)



A typical RMI system contains three layers -

Stub/Skeleton layer: client stubs and server skeleton, *Remote reference layer:* responsible for interpretation of invocation. Transport layer: establish connection, remote object management and tracking.

Implementation

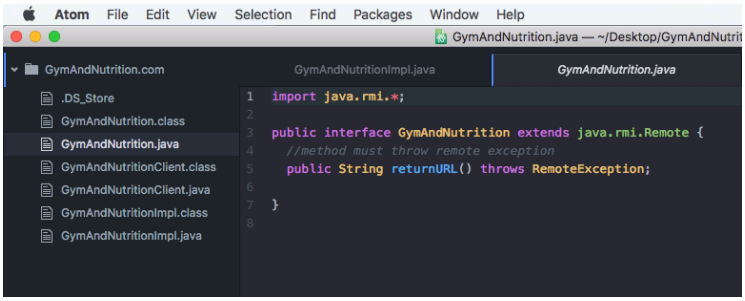
Note: For RMI to work client and server must be running JVM to communicate using JRMP.
Let's looks at the steps involved in the implementation (local implementation in this case) of simple RMI application. For this we will layout the steps and implement a sample "GymAndNutrition" application which will return the URL of the website when client makes a call to the remote object on the server.

List of Steps involved in the implementation of RMI application –

1.	*Defining Remote interface*
2.	*Implementing the Remote Interface*
3.	*Implementing the Server*
4.	*Implementing Client that makes use of our defined Remote Interface*
5.	*Generating Stubs and Skeletons*
6.	*Starting RMI Registry ($ rmiregistry &)*
7.	*Starting Server (server should always be started before Client)*
8.	*Starting Client*

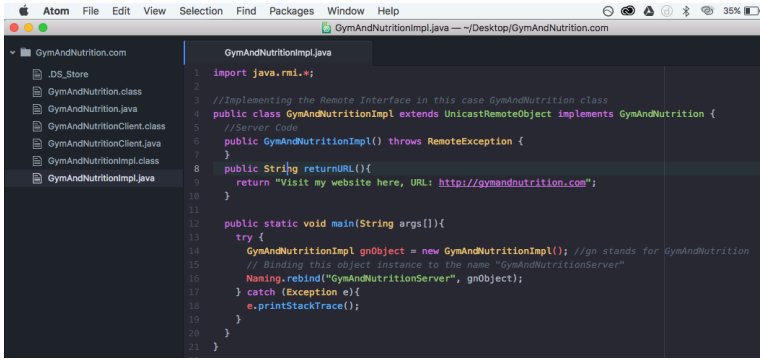
Below is the implementation of GymAndNutrition RMI Application using the above steps *(the code sample used here available at: https://github.com/virksaabnavjot is done with the help of resources made available at https://moodle.ncirl.ie by National College of Ireland)*:

Step 1: Defining Remote interface by extending (java.rmi.remote)



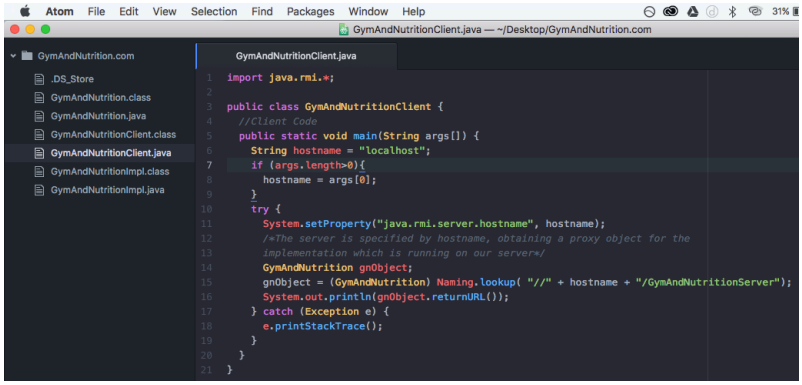
Since RMI, is Java only the interface object interfaces are written in Java. Stub and Skeleton are generated from this interface. And, it's an requirement that all the methods to object interface must throw Remote Exception (Package: java.rmi.RemoteException)

Step 2 and 3: Implementing the Remote Interface and Server



When the remote object's interface is de-fined, we can proceed with a server implementation of this interface. Also, the server typically extends the *java.rmi.server.UnicastRemoteObject* class. UnicastRemoteObject is an extension of the RemoteServer class, and function as a base class for server implementations of objects in Java RMI. Here, gnObject at line 14 (refer code screenshot), is our remote object. And at line 16, we bind this object instance to the name "GymAndNutritionServer" which our client will look for through the registry.

Step 4: Implementing the Client



Implementing the RMI client which will get reference to the remote object (gnObject) by connecting to the remote RMI registry and ask object by name (in this case - the name we specified earlier in the server "GymAndNutritionServer". Using the line 14-15 in the code implementation (refer screenshot).

Step 5: Generating Stubs and Skeletons

Open terminal on your computer (command line for Windows)

```
Navs-MacBook-Pro:gymandnutrition.com navNav$ ls
GymAndNutrition.java            GymAndNutritionImpl.java
GymAndNutritionClient.java
Navs-MacBook-Pro:gymandnutrition.com navNav$ javac *.java
Navs-MacBook-Pro:gymandnutrition.com navNav$
```

Compiling all our java files using – **javac *.java** (files can also be compiled individually if preferred – javac filename.java), stub and skeleton will be generated by compiling the interface and server implementation in bytecode using javac compiler.

Step 6: Starting RMI Registry

```
Navs-MacBook-Pro:gymandnutrition.com navNav$ rmiregistry &
[1] 6132
Navs-MacBook-Pro:gymandnutrition.com navNav$
```

We will start the RMI Registry through terminal using this command – ***Rmiregistry &*** (registry can also be started from code). In RMI registry serves as an Object manager and Naming service, its only required on server side. now a registered class can be located by client by using the lookup() method on the Naming interface.

Step 7: Starting Server

Start Server using – ***java GymAndNutritionImpl*** (or java YourServerFileName)

```
Navs-MacBook-Pro:gymandnutrition.com navNav$ java GymAndNutritionImpl
```

Server must be started before the client, this class GymAndNutritionImpl registers itself under the name GymAndNutritionServer as specified in the code (it can be any name you specify), which the RMI Client will search for.

Step 8: Starting Client

Open new/different terminal window. Start Client using – ***java GymAndNutritionClient***

```
Navs-MacBook-Pro:gymandnutrition.com navNav$ java GymAndNutritionClient
Visit my website here, URL: http://gymandnutrition.com
Navs-MacBook-Pro:gymandnutrition.com navNav$
```

Here, in the above screenshot we can see the results returned by our RMI Server that is the URL for the Gymandnutrition.com website. We have successfully implemented a basic RMI.

References

The research report was only possible because of the sources listed below and I would like to thank the authors of the work that was utilised to successfully complete the report. The research report utilizes different sources including but not limited to books, websites and reports published by reputed educational institutions and more.

Cs.iupui.edu. (n.d.). *Disadvantages of RMI*. [online] Available at: http://cs.iupui.edu/~aharris/cgi-bin/slides/langs56.html [Accessed 12 Apr. 2017].

Devx.com. (n.d.). *To RMI or Not to RMI*. [online] Available at: http://www.devx.com/tips/Tip/25531 [Accessed 10 Apr. 2017].

Docs.oracle.com. (n.d.). *An Overview of RMI Applications (The Java™ Tutorials > RMI)*. [online] Available at: https://docs.oracle.com/javase/tutorial/rmi/overview.html [Accessed 9 Apr. 2017].

Docs.oracle.com. (n.d.). *Getting Started with Java IDL*. [online] Available at: http://docs.oracle.com/javase/8/docs/technotes/guides/idl/GShome.html?cm_mc_uid=72983530074714919152354&cm_mc_sid_50200000=1491915235 [Accessed 7 Apr. 2017].

Docs.oracle.com. (n.d.). *java.rmi (Java Platform SE 8)*. [online] Available at: https://docs.oracle.com/javase/8/docs/api/java/rmi/package-summary.html [Accessed 13 Apr. 2017].

Fuhrer, P. (n.d.). *RMI*. [online] pp.6-9. Available at: http://diuf.unifr.ch/drupal/sites/diuf.unifr.ch.drupal.softeng/files/file/publications/others/RMI.pdf [Accessed 12 Apr. 2017].

German, D. (n.d.). *RMI: Observing the Distributed Pattern*. [online] Computer Science Department, Indiana University Bloomington, p.1. Available at: http://www.cs.indiana.edu/~dgerman/tutorials/fie2004.pdf [Accessed 11 Apr. 2017].

Ibm.com. (2017). *What are RMI, IIOP, and RMI-IIOP?*. [online] Available at: https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.hybrid.80.doc/rmi-iiop/overview.html [Accessed 7 Apr. 2017].

Javacoffeebreak.com. (n.d.). *Java RMI & CORBA - a comparison of competing technologies*. [online] Available at: http://www.javacoffeebreak.com/articles/rmi_corba/#reillyd_corba [Accessed 13 Apr. 2017].

Lee, I. (2007). *CIS 505: Software Systems Introduction to Distributed Systems*. [online] University of Pennsylvania. Available at: https://www.cis.upenn.edu/~lee/07cis505/Lec/lec-ch1-DistSys-v4.pdf [Accessed 4 Apr. 2017].

Oracle.com. (2017). *Advantages of RMI (Java Remote Method Invocation) - Distributed Computing for Java*. [online] Available at: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html#close [Accessed 9 Apr. 2017].

Reilly, D. (2006). *Introduction to Java RMI*. [online] Javacoffeebreak.com. Available at: http://www.javacoffeebreak.com/articles/javarmi/javarmi.html [Accessed 12 Apr. 2017].

Standford University. (n.d.). *RMI basics - How RMI Works?*. [online] Available at: http://infolab.stanford.edu/CHAIMS/Doc/Details/Protocols/rmi/rmi_description.html [Accessed 7 Apr. 2017].

Tanenbaum, A. and Steen, M. (2007). *Distributed systems - Principles and Paradigms*. 2nd ed. pp.115-116.
Villanova University, United States. (n.d.). *Distributed Systems -Introduction (What is Distributed Systems?)*. [online] Available at: http://www.csc.villanova.edu/~schragge/CSC8530/Intro.html [Accessed 6 Apr. 2017].

www.javatpoint.com. (n.d.). *Remote Method Invocation (RMI) - javatpoint*. [online] Available at: http://www.javatpoint.com/RMI [Accessed 5 Apr. 2017].

The research report also utilizes Moodle resources available on NCI college website (mainly the implementation code sample to demonstrate the implementation steps).
https://moodle.ncirl.ie

The images used in this report, 1- The hand drawn images - are my own work which I have drawn from my own understanding on RMI mechanisms learned through the research, and may not be 100% accurate please refer to RMI's official documentation in case you require more accuracy. 2- Any other images - are screenshots of my work, the development environment used to write Java code is Atom Code Editor and MacBook's terminal is used to execute commands.