International Journal of Emerging Technologies in Engineering Research (IJETER)
Volume 4, Issue 1, January (2016) www.ijeter.everscience.org

Transparency in Remote Method Invocation (RMI) for Distributed Systems: Middleware Layer

D.Madhavi
Assistant Professor, Andhra Loyala Institute of Engineering and Technology, India.

Abstract – Remote Method Invocation (RMI) is a mechanism that allows objects located in a different computers in a computer network to interact with each other. The main goal of RMI is to provide the full transparency regarding Distribution. This article gives the overall view of the working procedure of Remote Method Invocation and how it aims to be distribution transparency. Achieving distribution transparency is not an easy task in a distributed systems but this can be done through RMI. RMI is most important and currently most popular concept that need to be known by any one because it is used in so many applications such as internet search engines, social websites, distributed file systems, various mailing systems, online groups etc. This paper demonstrates clearly about RMI and its Transparency.

Index Terms – RMI, RMI Registry, Distributed Objects, Proxy, Skeleton, Marshalling.

1. INTRODUCTION

Distributed System is a collection or group of independent or autonomous systems that appears to its end-users as a single coherent system. The Structure of Distributed System [1] is shown in the below Fig. 1. It consists of user applications, middleware layer and local operating systems. User applications are presented on top of the middleware layer and consist of all the distributed system applications run by the different users. Middleware layer is logically placed in between user applications and local operating system. It acts as an interface between distributed applications and local operating systems. Examples of middleware layer are CORBA, DCOM etc. local operating systems of different computers in a distributed system, which is presented at the lower of all the layers. Different systems in a distributed system are having different operating system. Operating system provides its services supported by the middleware layer.

The three fundamental properties of a Distributed System are Transparency, Scalability and Openness. Transparency means hiding of resources, data, location, access, migration, relocation, replication, network, concurrency, failures occurred in a distributed system. Among all these distribution transparency is the important thing which aims to reach goal of a system. This can be achieved through middleware layer as shown in the Figure 1. Scalability is the ability of increasing or decreasing of resources or systems in a distributed system. It depends up on the capacity of a system without affecting its services. Openness is the syntaxes and semantics of a system can be able to open to the developer as well as user to access the services without knowing how they are implemented internally even in a heterogeneous environment.

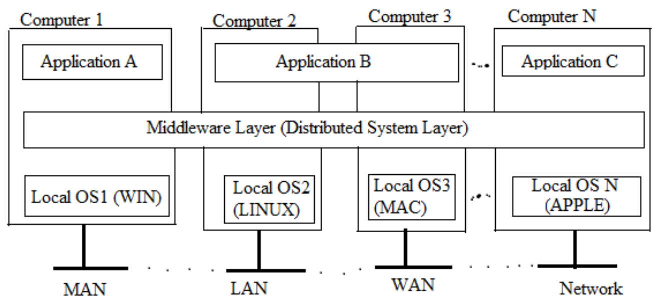


Figure 1 Structure of Distributed System

RMI is the most important part of distributed system placed in the middleware layer of it. If any one method that is called by a client system is not presented in it, but that method definition is presented in the other system on either client or system then we need an RMI. RMI is the one such concept without which the Distributed system is not designed to provide transparency especially distribution transparency. This paper discuss about what is the architecture of RMI, working of RMI, in which layer the RMI is presented in distributed system and how it is implemented with advantages.

RMI is a mechanism to allow access to the remote object which is presented on another remote machine connected in a network within a distributed system. Java RMI is quite similar to RPC (Remote Procedure Call). i.e RMI is java's version of RPC with its object orientated concepts. Basically RMI is only a version of CORBA (Common Object Request Broker Architecture) by supporting its remote object invocation concepts where as RPC does not allow object invocation it only calls the procedures which are presented on remote machines. The protocols used for implementation of RMI systems are JAVA RMI, CORBA IDL (Interface Definition Language), Microsoft DCOM (Distributed Component Object Model) /COM+, SOAP (Simple Object Access Protocol). Java RMI is implemented only in java object oriented concepts. CORBA uses CDL (Common Definition Language) [2, 3] to represents, call, reference remote objects and to support object oriented concepts. Sun Micro systems developed a DCOM model for supporting RMI. SOAP protocol presents RMI on top of HTTP. It is a part of middleware layer as shown in the Figure 2.

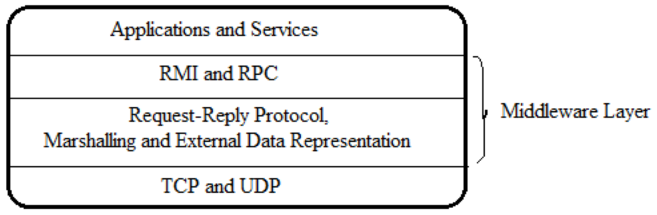

Applications and Services
RMI and RPC
Request-Reply Protocol, Marshalling and External Data Representation
TCP and UDP
Middleware Layer

Figure 2 RMI is in Middleware Layer

Distributed Systems uses Request-Reply protocol [2] for client-server communication as shown in the Figure 3. It is a part of middleware layer. This protocol supports three methods named as doOperation, getRequest and sendReply. It provides certain delivery guarantees. RMI passes a remote object reference in the request message and that method is invoked on the remote system.

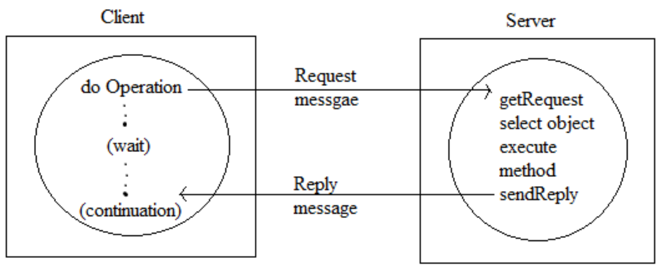


Figure 3 Request-Reply Communication

Multithreading concept is used to invoke a method on a remote object presented in remote machine. In practice, RMI is the client to hold references to remote objects that it can invoke methods on. These references should behave just like a local objects, but when invoked dispatch the method invocation to the remote object. Because need to refer to both and a method now, not just a function so it sends an object id across the socket, then the method and the arguments. The server then dispatches the method to the remote object based on id and method name, interface name in which the method defined. The flow of Communication mechanism during the execution of a remote method for RMI from client to server is

Client ←→ Interface ←→ Stub ←→ Skeleton ←→ Interface ←→ Server

The primary advantages of Java RMI [4, 5] are

- Object Oriented
- Mobile behavior
- Parallel computing
- Distributed Computing
- Write once run anywhere
- Easy to develop and use
- Dynamic class loading is very powerful
- Parallel computing
- Distributed Garbage Collection
- Security and Safety
- Connections to Legacy systems using JINI

The Disadvantages of Java RMI are

- Insecure while dynamic class loading
- Challenging and hard to implement call back mechanism over Internet.
- Overhead of Object serialization, marshalling and unmarshalling.
- Supports only JAVA RMI.

2. RELATED WORK

The first Java RMI [7] core library supports built-in functionality for distributed computing. Basically distributed systems are built on hertogenious environment so that they support first versions of java release, as time progresses new releases of java supports CORBA through RMI-IIOP. Later Sun Microsystems supports distribution capabilities for java and introduces Jini, in 1998 by Apache River. It supports secure distributed applications and services which are facilitated by Jini Extensible Remote Invocation Protocol (JERI) using JavaSpaces [9]. JERI is not equal to RMI but systemically similar to RMI. River also targeted towards to grid computing. It allows the Java Remote Method Protocol.

Cajo library to build distributed java virtual machines and uses RMI. It reduces network traffic, controls and agent objects exchanged between virtual machines. It uses multicast communication.

CloudSNAP [8] is a platform for decentralized systems on J2EE web applications. DAMON (Aspect Oriented Interception Middleware) is used to achieve distribution of objects. It uses the underlying architecture as P2P Network. It is better used for Cloud Enterprise web applications.

P2P-MPI [10] is a project to support Parallel P2P grid computing. This framework extends for resource sharing and Message passing interface Infrastructure. So that acts as a computational grid.

JXTA protocol provides definition and library to build P2P-based distribution of java objects. JXTA is independent.

All of the above does not provide full efficient transparent integration of simple object oriented approach but P2P RMI [6] provides this full distribution transparency.

3. PORPOSED MODELLING

The architecture of RMI is shown in the below Fig.2. Before performing the RMI operation specified in the architecture, first bind a client to a server is performed as shown in the Figure 4.

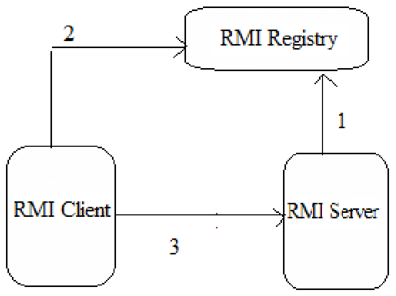


Figure 4 Binding a Client to the Server System

Binding a client system to the server is done in the three steps. They are

Step 1: Server must register the object's interface at RMI registry.

Step 2: If client wants to access the remote object then it looks up in the RMI registry.

Step 3: After getting the object's interfaces implementations into the client's address space, the actual RMI mechanism is taken place.

The RMI architecture and have the following steps as shown in the Figure 5.

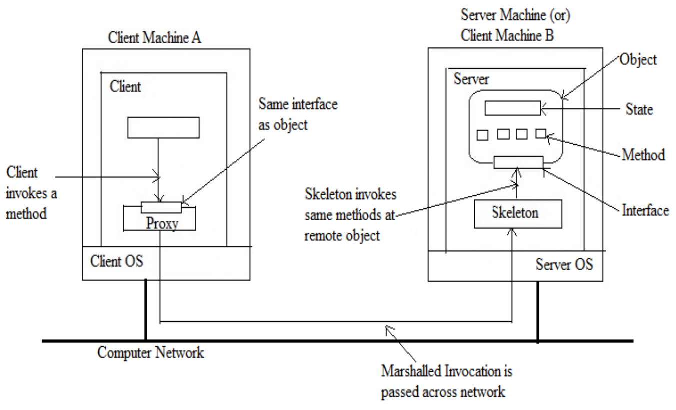


Figure 5 Simple Model / Architecture of RMI

Step 1: The server first registers their interfaces at the RMI registry i.e binding a client to a server.

Step 2: The client machine which it wanted to invoke a method on remote machine can generate the stub. It has objects id, method name, server reference be marshaled by the client stub is called the proxy.

Step 3: The proxy provides an implementation of interface as the server object which marshals method invocations into messages, sends into the target, waits and unmarshals reply messages to return the result of the method invocation to the client.

Step 4: Then proxy is converted that message into the required protocol format used for communication such as HTTP, SOAP… etc, this process of converting method arguments to message formats suitable for transmission in a computer network is called marshalling.

Step 5: After marshalling the data is given to Client Operating System which converts the data into binary format understandable by the Physical Layer.

Step 6: Server Operating System takes binary data and reformat and hand it over to the Server Stub called Skeleton.

Step 7: Skeleton unmarshals requests to proper method invocations at the object's interface at the server.

Step 8: The dispatcher selects the appropriate method in the Skeleton. Dispatcher has Object in which methods are defined.

Step 9: After Server does work, it returns result to the Skeleton.

Step 10: The Skeleton also marshals replies and forwards replies to the client-side Operating System through Computer network.

Step 11: Client OS takes and handover it to the Proxy, then it unmarshals the data

Step 12: The unmarshaled data is handover to the requested Client process.

The RMI structure offers full transparency regarding distribution.

Objects can be implemented in different ways. There are various categories of objects are there named as

- Compile time Objects
- Runtime Objects
- Distributed Objects
- Remote Objects
- Persistent Objects
- Transient Objects

3.1 Compile-Time Objects:

- Compile time objects are directly related to language level objects supported by Java and C++.
- The most obvious form of objects are known as compile time objects.
- They are easy to build distributed system applications using compile time objects.
- Drawback of compile time objects is dependent on particular programming language.

- In compile time objects, objects are defined as an instance of a class.

3.2 Runtime Objects

- The implementation of Runtime objects are left open.
- This approach to Object-based Distributed Systems allows an application to be constructed from objects written in multiple languages.
- This approach may use object adapters that acts as wrappers that give implementations of an object appearance.

3.3 Distributed Objects

- It is an Object which publishes its interface on other machines
- The characteristic of the Distributed object is that their state is not distributed.
- It is constructed during explicitly runtime
- It is followed in many object based distributed systems, which are independent of programming languages.
- Applications can be constructed from objects written in multiple languages.

3.4 Remote Objects

- It is a distributed object whose state is encapsulated (Objects state is not distributed here).
- Objects implementation is left open. Object methods are invoked from remote systems.
- The remote object is based on id and method name.

3.5 Persistent Objects

- It continues to exist even if it is currently not contained in the address space of a server process.
- These objects are not dependent on its current server.
- Server manages this object then it stores into secondary memory and exit.

3.6 Transient Objects

- This object exists only as long as the server that manages the object.

Basically RMI is of two types they are Static RMI and Dynamic RMI.

Static RMI:

- After binding of a client to a server, it can invoke the object's methods through proxy
- The preferred interface definitions referred to as static invocation. Here interfaces of an object are known when the client application is being developed.
- If interfaces can change then the client application must be recompiled before it can make use of new interfaces.
- The preferred interface definitions are presented in CORBA's IDL.

Dynamic RMI:

- It can be able to compose a method invocation at runtime. It can also refer to as dynamic invocation.
- An application selects at runtime which method it will invoke at a remote object.

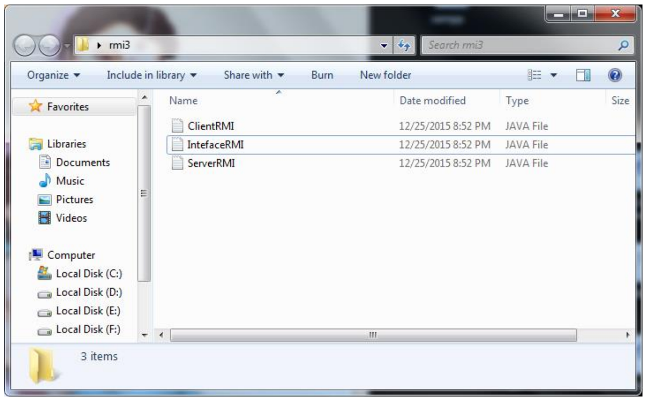
There is a drawback to RMI, it provides inherently synchronous nature. Even though it contributes to hiding the communication in Distributed System unfortunately this mechanism is not always appropriate. Client is blocked until its requests have been processed. There is a necessicity of using another mechanism called Message Oriented communication to avoid this drawback.

4. RESULTS AND DISCUSSIONS

RMI effectuation generates stub at client side and skeleton at server side on which actual implementation of an interface is defined. In the process of RMI implementation, Client machine uses RMIregistry as in Figure 4. to bind to a particular server in a network of Distributed System. A result of implementation of RMI using an object oriented language known as JAVA is shown in this section.

Consider a simple arithmetic operations whose method definitions are not available on the client system then it can invoke those required methods which are existed at server machine or another machine which is placed remotely in a Distributed System. Then this scenario uses the RMI mechanism. This paper gives an example for addition, subtraction, multiplication and division operations, whose method definitions are found on another machine.

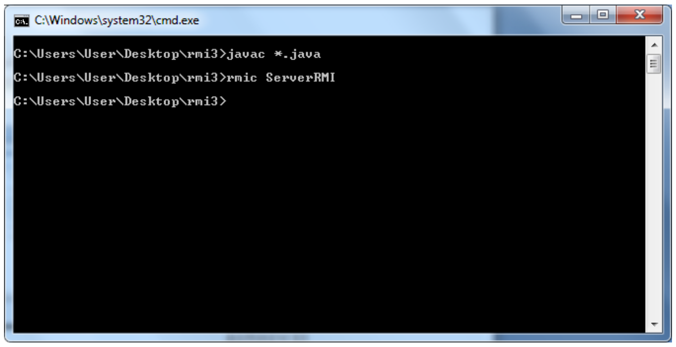
The Figure 6 Shows the three different Java files named as ClientRMI.java, to represent client side code, ServerRMI.java, represents server side object oriented code and InterfaceRMI.java, which specifies the interface definition.



Name	Date modified	Type	Size
ClientRMI	12/25/2015 8:52 PM	JAVA File	
IntefaceRMI	12/25/2015 8:52 PM	JAVA File	
ServerRMI	12/25/2015 8:52 PM	JAVA File	

Figure 6 Creation of Client, Server and Interface Files

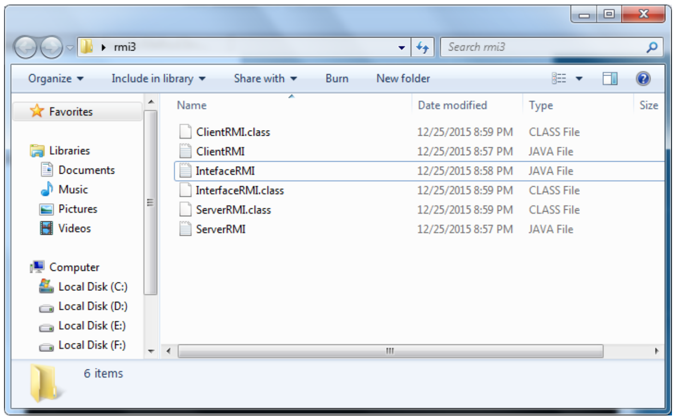
After creation of all the java files there is a need to compile all the created java files using Java compiler named as Javac. It creates .class files for all the java files as shown in the below Figure 7.



```
C:\Users\User\Desktop\rmi3>javac *.java
C:\Users\User\Desktop\rmi3>rmic ServerRMI
C:\Users\User\Desktop\rmi3>
```

Figure 7 Compilation of .java Files

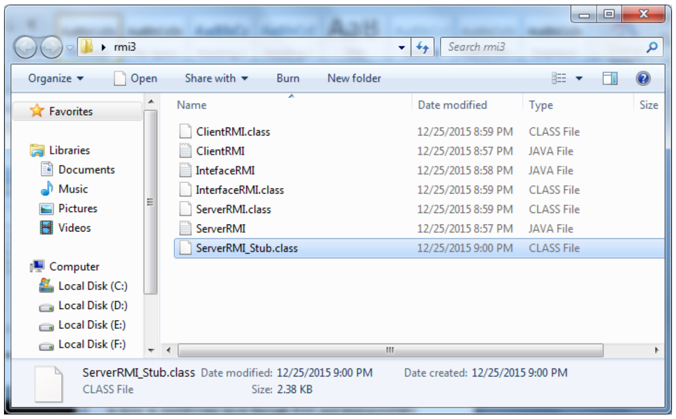
The java compiler compiles all the files and generates the .class files as shown in below Figure 8.



Name	Date modified	Type	Size
ClientRMI.class	12/25/2015 8:59 PM	CLASS File	
ClientRMI	12/25/2015 8:57 PM	JAVA File	
IntefaceRMI	12/25/2015 8:58 PM	JAVA File	
InterfaceRMI.class	12/25/2015 8:59 PM	CLASS File	
ServerRMI.class	12/25/2015 8:59 PM	CLASS File	
ServerRMI	12/25/2015 8:57 PM	JAVA File	

Figure 8 Creation of .class Files

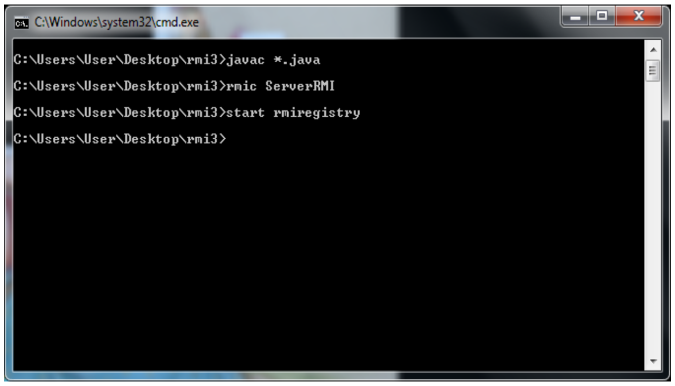
RMIRegistry is started on a system which generates the Stub class named as 'ServerRMI_Stub.class' for the interface as shown in the below figures Figure 9 and Figure 10 and as a result of RMI registry the service of registry started by opening a new command window as shown in Figure 11.



Name	Date modified	Type	Size
ClientRMI.class	12/25/2015 8:59 PM	CLASS File	
ClientRMI	12/25/2015 8:57 PM	JAVA File	
IntefaceRMI	12/25/2015 8:58 PM	JAVA File	
InterfaceRMI.class	12/25/2015 8:59 PM	CLASS File	
ServerRMI.class	12/25/2015 8:59 PM	CLASS File	
ServerRMI	12/25/2015 8:57 PM	JAVA File	
ServerRMI_Stub.class	12/25/2015 9:00 PM	CLASS File	

ServerRMI_Stub.class Date modified: 12/25/2015 9:00 PM Date created: 12/25/2015 9:00 PM
CLASS File Size: 2.38 KB

Figure 9 Generation of Stub Class, ServerRMI_Stub.class



```
C:\Users\User\Desktop\rmi3>javac *.java
C:\Users\User\Desktop\rmi3>rmic ServerRMI
C:\Users\User\Desktop\rmi3>start rmiregistry
C:\Users\User\Desktop\rmi3>
```

Figure 10 Executing the Command start rmiregistry

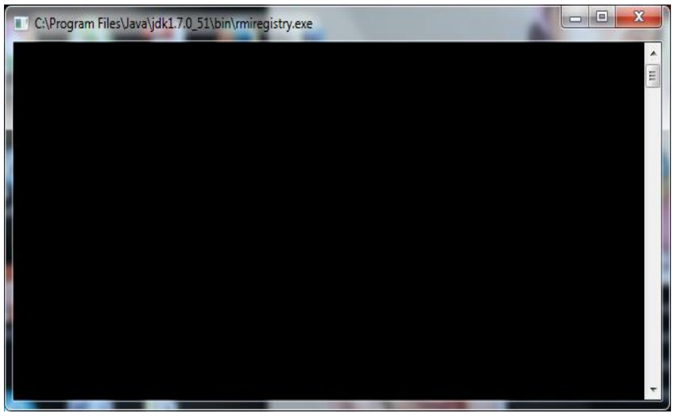
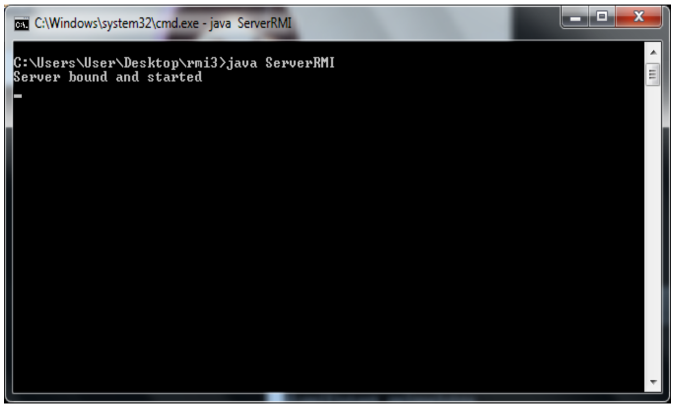


Figure 11 Starting RMI Registry Service

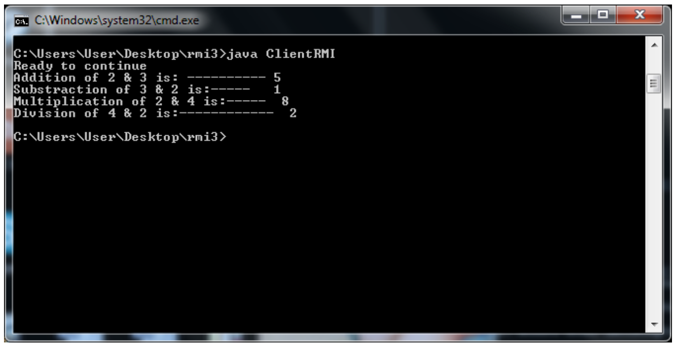
Open a new separate environments for both server and client side execution of programs by running server and client program consecutively as shown in the below Figure 12 and Figure 13.

Binding a client to a server can be done by executing rmi registry. After that execution of server program can be done Figure 12.



```
C:\Users\User\Desktop\rmi3>java ServerRMI
Server bound and started
_
```

Figure 12 Execution of ServerRMI



C:\Windows\system32\cmd.exe

```
C:\Users\User\Desktop\rmi3>java ClientRMI
Ready to continue
Addition of 2 & 3 is: ---------- 5
Substraction of 3 & 2 is:-----   1
Multiplication of 2 & 4 is:-----  8
Division of 4 & 2 is:-----------  2

C:\Users\User\Desktop\rmi3>
```

Figure 13 Execution of ClientRMI

5. CONCLUSION

Remote Method Invocation is presented in the Middleware layer of a Distributed System. This layer is heart or central part of all the systems in a Distributed System. The important property of distribution transparency is provided by this Layer with the help of RMI. That is implementation of transparency is done in middleware layer through RMI and interoperability issues are also provided as discussed in the above section of this paper. Transparency is the main goal of RMI. Without transparency there is no Distributed System. It places the major role of design and implementation of any Distributed System. Achieving full transparency is not possible but maximum extent it will be achieved depending upon the application.

REFERENCES

[1] Andrew S. Tanenbaum, Maarten Van Steen, "Distributed Systems: Principles and Paradigms", Eastern Economy Edition, PHI Learning Private Limited, Second Edition. ISBN-978-81-203-3498-4.

[2] George Coulouris, Jean Dollimore and Tim Kindberg, "Distributed Systems Concepts and Design", Pearson education, Fourth Edition.

[3] Hanumant Pawar, Sujeet Patil, Sourabh Karche, Udit Upadgayay, Mahesh Channaram, "Distributed Object Computing Using Java Remote Method Invocation", International Journal of Emerging Research and Technology, Volume 3, Issue 5, May 2015, PP 68-76, ISSN 2349-4395 (Print) and ISSN 2349-4409 (Online).

[4] Integrated Cloud Applications & Platform Services, "Java Remote Method Invocation- Distributed Computing for Java", ORACLE Technology Network, Technologies.

[5] Kapep, Rafalmag, Rob Lachlan, "Benefits and Disadvantages of using java rmi", Stack Overflow Community, Stack Exchange Inc., Licensed under cc by-sa3.0, 6th June 2014, 30th Oct 2012, 26th Feb 2016 respectively.

[6] Thomas Zink, Oliver Haase, Jurgen Wasch and Marcel Waldvogel, "P2P-RMI: Transparent Distribution of remote java Objects", International Journal of Computer Networks & Communications (IJCNC) Vol. 4, No. 5, September 2012.

[7] Ann Wollrath, Roger Riggs, and Jim Waldo, "A Distributed Object Model for the JAVA System", USENIX Computing Systems, 9, 1996.

[8] Ruben Mondejar, Pedro Garcia-Lopez, Carles Pairot, and Lluis Pamies-Juarez, "CloudSNAP: A Transparent Infrfastructure for Decentralized web deployment using Distributed Interception", Future Generation Computer Systems, (0), 2011.

[9] Qusay H.Mamoud, "Getting Started With JavSpaces Technology: Beyond Conventional Distributed Programming Paradigms", July 2005.

[10] Stephane Genaud and Choopan Rattanapoka,"A Peer-to-Peer Framework for Message Passing Parallel Programs", Advances in Parallel Computing, Volume 17, pages 118-147. IOS Press, June 2009.