Network and Complex Systems
ISSN 2224-610X (Paper) ISSN 2225-0603 (Online)
Vol.3, No.5, 2013

Use of Java RMI on Mobile Devices for Peer to Peer Computing

Christopher I. Eke*, Absalom E. Ezugwu[1], Henry F. Nweke[2], Mehmet.E Aydin[3]
1. *Department of Computer Science, Federal University, Lafia, Nigeria
2. Department of Computer Science, Ebonyi State University, Abakaliki, Nieria
3. Department of Computer Science and Technology, University of Bedfordshire, Luton, England
{ eke.ifeanyi, ezugwu.absalom}@fulafia.edu.ng, mehmet.aydin@beds.ac.uk

Abstract: In this paper, the use of Java RMI on mobile devices for peer-to-peer computing is presented. An overview of the commonly used distributed middleware systems are described by looking into remote procedure call (RPC) and object oriented middleware java remote method invocation (Java RMI). The differences between this middleware are equally detailed in this work. A review of some related literature was carried out and some of the features required for the proposed prototype were also extracted accordingly. This paper also provides an overview of peer-to-peer networking and some of the application areas linked to the platform implementation. Detailed design and implementation of the artifact for peer-to-peer network using Java 2 platform programming language were carried out. Finally, on the process of this research, three applications were developed and peered together to show that java RMI is a tool for peer-to-peer computing.
Keywords: - Remote method invocation, Remote procedure call, Stub, Skeleton, Peer-to-Peer

1. Introduction

The rapid increase and changes on portable devices, and the use of java technology for these devices facilitates the developers and researchers to have interest in investigating innovative applications in such areas. These mobile devices such as mobile phone, personal digital assistant (PDAs), mobile computers and others which are linked together through wireless network, their connection to service and other network relies on interoperability of hidden architecture in which they are operating upon. This hidden architecture is usually referred to as middleware. In [2, 3], "middleware is defined as a distributed platform that layered between the network operating system and application which facilitates development, deployment and management of distributed applications".

Java remote method invocation (RMI), an example of middleware system is a specification for building distributed object oriented application. Java remote method invocation is an object oriented middleware system in which a user is able to call methods on object residing in another java virtual machine which could also be located in a remote machine.

Distributed computing has become most popular in computing industries, academic research and even in technological paradigm. In a distributed system, the exchange of data between nodes or service access in between nodes can be achieved through the interaction among distributed components. Kortuem [10] maintained that this interaction may be built directly on top of network operating system primitives, which would be complex for many application developers. However, a middleware, which is layered between application and network operating system, can facilitate component interaction in distributed systems [1].

With the vast increase in Network deployment in the area of distributed computing, Middleware has become the building block for the development of future distributed system. Remote Procedure Call (RPC) is being proposed as a simple efficient and unifying paradigm to move data and control it flow to a remote source. Caporuscio, in [1] stated that, remote procedure call (RPC) is a protocol that allows components to invoke procedures executed on remote hosts without explicitly coding the details for this interaction. In traditional RPC, it consists of client and server side application. Each call to a remote interface is transferred from the client to the server for execution and the result is returned back to the client. In server side, a special RPC protocol compiler called *rpcgen* is used to generate stub and skeleton. The stub or stub interface need to be in a client address space at a compile time. Methods invoked on a remote object are intercepted by the client side stub that moves the call to the server side skeleton to give an illusion of a local invocation to the client application. RPC has been extended to support object oriented programming language by introducing object proxies, which forward calls from client to server.

Interface number and version numbers (a unique identification number) are used to reference a remote object in order to provide a strong contract between a client and a server. The RPC runtime system is used to hide the transport details from the client and server stubs manages the interface IDs and implements the scheduling policies for server side resources. RPC middleware offers many services such as

i. Generating Client/Server stub
ii. Marshalling and unmarshalling data (input parameters and return values)

* eke.ifeanyi@fulafia.edu.ng (Christopher I. Eke)

iii. Establishing synchronous communication and
iv. Assuming non-functional properties.

RPC middleware are usually synchronous, and this presents no potential for parallelism without using multiple threads. In the implementation of the RPC (see figure 1), there is an assumption that all the hosts in the network are stationary and always reachable with an exception of network failure and that the network is more or less freely available resource with no bandwidth constraints.

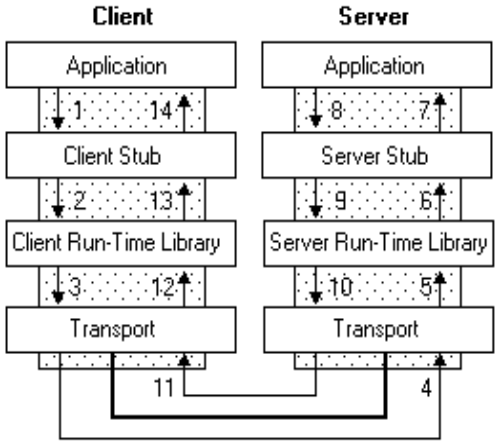

Figure 1: The Architecture of RPC.

Java is one of the object oriented programming language which is mostly used in building a web based application. The popularity of Java is based on its object oriented nature and platform independent. There are two different suitable distributed object that are used in java: these are the common object request broker architecture (CORBA) and remote method invocation (RMI). They simplifies the building and running of client-server based application. Even though CORBA and RMI provides the same functionality by hiding communication details of remote method invocation object interface which is seen to client, yet, they were built on different fundamental.

According to [7], "Java remote method invocation (RMI) is a java standard protocol that allows users to call methods on object located in another java virtual machine which in turn might be located in a remote node". The RMI architecture is based on behavioural definition and implementation of that behaviour. These two principles coded separately are to run on separate JVM. In java RMI, an object can be passed to remote method using different semantics. For instance, an integer value can be transferred using a *call-by- value* semantics. Call-by-copy semantics are used to pass java objects as well as non exported RMI objects to a remote method. Java RMI uses java serialization during the marshalling process; hence, objects being passed using call-by-copy semantics have to be serialized.

UnicastRemoteObject class makes it possible to export and un-export RMI objects. Call-by-reference semantics is used to remotely access export objects. Java RMI architecture is composed of three independent layers of architecture:

i. The stub and skeleton layer: this layer is an interface between the application layer and the rest of the RMI system. Stub and skeletons are generated as an interface for the client and server to the middleware. They facilitate the distribution –transparent invocations on the remote Objects and handle the marshalling and de-marshalling of objects. These two classes are generated from the object's remote interface which must be specified by the application programmer. The stub can be generated by running RMI compiler (RMIC) whereas the skeleton is automatically generated during the runtime execution time.
ii. Remote reference layer: this is the second layer of the RMI architecture. Its function is to carry out the semantics of the invocation. It gives the interpretation that a remote object call is made by the client. The remote reference layer of the client receives the method requests from the client and the request is passed to the remote reference layer of the server. Objects implementing remote call semantics have to implement RemoteRef interface which is defined in the RMI specification.
iii. Transport layer: the transport layer as the name implies is responsible for setup and handles communication between the different participating JVMs. Once this layer receives a request from the remote reference layer on client, it has to search for RMI server, before the establishment of connection to the server. RMI system is capable of using any streamed-based communication with the incorporation of socket factoring concept introduced in java 1.2.

The architectural diagram for Java RMI is illustrated in figure 2 below.

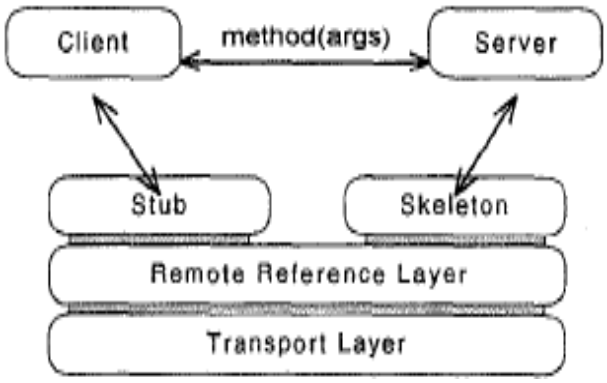

Figure 2: The Java RMI System Architecture

2. Review of Related Work

Several frameworks have been proposed to implement P2P computing on mobile devices using Java platform. In [11], JXTA hybrid peer (JHPeer) is design to facilitate the application software developed to run on mobile devices. JHPeer allows distributed access, concurrent querying and parallel computing for mobile information retrieval system over a large scale of mobile network. JHPeer is made up of two different peers (super peer and mobile peer). Super peer performs the duty of creating and monitoring overlay network, providing searching and directory services to other super peer and mobile peer. In contrast, mobile peer detects and join the network dynamically, offers information and permits context based information retrieval. The JHPeer framework consists of the following component parts with their operations:

i. Query processor :- It handle and process queries, allow context based retrieval.
ii. Context processor :- It manages and share context information in the peer group network.
iii. Indexing components :- It organizes information into index file for fast searching and it also saves temporary information to the database.
iv. Network monitor :– It monitor the peer network and check for online peers.
v. Application components – They are service components deployed in the system for providing functionality to users.

Some of the applications that can be integrated on JHPeer are described below.

i. Tourism :– This involves automatic detection and location of information by a system based on the geographical location as soon as user starts this application.
ii. Mobile news :– News service is another benefit that this application offers to mobile users. Important news are published automatically in real-time, alerting users with interesting topics using user's preference and contexts.
iii. Mobile discussion board :– Mobile discussion board is a means of providing information and sharing such information among peers (users) in order to fulfil the user's demand to satisfaction.
iv. Mobile blog :– Mobile blog enables mobile users to shear their feelings about what is happening at a particular moment. A particular mobile user can post a blog and other mobile users within that location can read it.

JXTA (Juxtapose) application is aP2P framework built at sun Microsystems aimed at providing peer to peer exchange of data and resources which does not depend on any operating system for its functionality. It allow any connected device on the network that ranges from PDAs to servers to communicate and interact with each other as peers. JXTA is made up of core, which offers security, monitoring, and group service among peers. It consist of application programming interface and certain protocol for the purpose of peer to peer collaboration. It also provides decentralized environment that aids user's application to synchronize in ad-hoc mode. In JXTA application, XML descriptors enable users advertise information and services. Peer discovery protocol can be used to find these descriptors whereas JXTA service layer provides authorization access to these services. JXTA can be implemented in variety of programming languages but most cases in Java and does not depend on network connections. JXTA is quite broad and it is meant for communication which covers P2P application with good number of hardware platforms.

Proxy Lady [4] is an application prototype that is aimed at supporting familiar and direct collaboration among people. Proxy lady is applicable for group of people who always meet together and requires exchanging messages whenever the need arises. This prototype has an intimate cooperation support through disclosing their location and the item of the message (like e-mail and files). The mobile device holds the contents of these messages when peering with other network gadgets. The approach about proxy lady is that this message item presumed can be used as source for familiar collaboration. Users may want to chat with message item they holds,

which means that users of these system have each other's knowledge and have stored message item on mobile gadgets bearing in mind of their recent inter-communication.

JINI prototype: In [5], "Jini is a name given to a distributed computing environment, which can offer "network plug and play" and user friendly access to information." In JINI technology, a gadget can join a network and all the services in that network can locate it and exchange the resources without any central system coordination. JINI is applicable in a wireless mobile network where members join and leave the network within a short period of time. In implementing a JINI , three parties are involved namely; the service that is made available, the client that will have an access to such service and the lookup registry which is used to find the service. The lookup service forces JINI to have a central server located somewhere in the network which fails the decentralized objectives of the proposed system.

Proem [9], is a prototype which is java platform dependent peer-to-peer designed to operate on both mobile and fixed wired network. Proem architecture consists of three structured layer, the peers, peerlets and the message. Peers are different gadgets from the platform in which it operates on and it is composed of P2P network. On the order hand, the peerlets are the applications that proem peer to peer communicate with. They use protocol defined by the designer for communication. Proem is designed to promote execution of P2P application in one peer. However, Proem fails to meet up with the proposed system as it is designated to operate on the fixed wired network. It also pay more attention on person to person communication which is mostly used in ad-hoc environment with constant change in network topology.

ProMoCoTo framework [6], is a P2P application embedded on mobile devices that brings about a mobile dedicated tool. The pro-active form of the device involves finding a dedicated collaborator for interaction and formation of decentralized network in which the simple association could be carried out. The prime objective of ProMoCoTo is to advance the interaction which can occur when co-staffs interacts together in simple, open collaboration and message exchange. The application stores information about owner's profile such as their bio-data and information regarding their job experience. The mobile devices of two or more co-worker will discover each other when they meet without their arbitration. Their gadget is now referred to as peers in decentralized P2P network which will instantly begin to interchange messages. According to [6], the message exchange is done in the following manner:

> "*A query is sent which is run against the local profile of each peer,*
> *If a match is found, the profile will be sent to remote peer who sent the query*.
> *When a profile has been received, the user is informed, and become aware of the possibility for interaction and communication with nearby user*"

3. System Architecture

In [8] it is stated that, "The beginning of wisdom for a computer programmer is to recognize the difference between getting it right". Therefore, when developing a complex computer based system, it is often very useful to breakdown the programming effort into subsystems and component modules during the design stages (Top down design).

If a programmer can produce a simple program module then the programmer can develop a subsystems comprised entirely of program module by taking the modules one at a time. Therefore, if a complex system is first divide into subsystem and each subsystem is further divided into modules then anyone who can develop a simple program module can also design a complex software system following the " divide and conquer" techniques. The design of the proposed system is illustrated in figure 3 below.

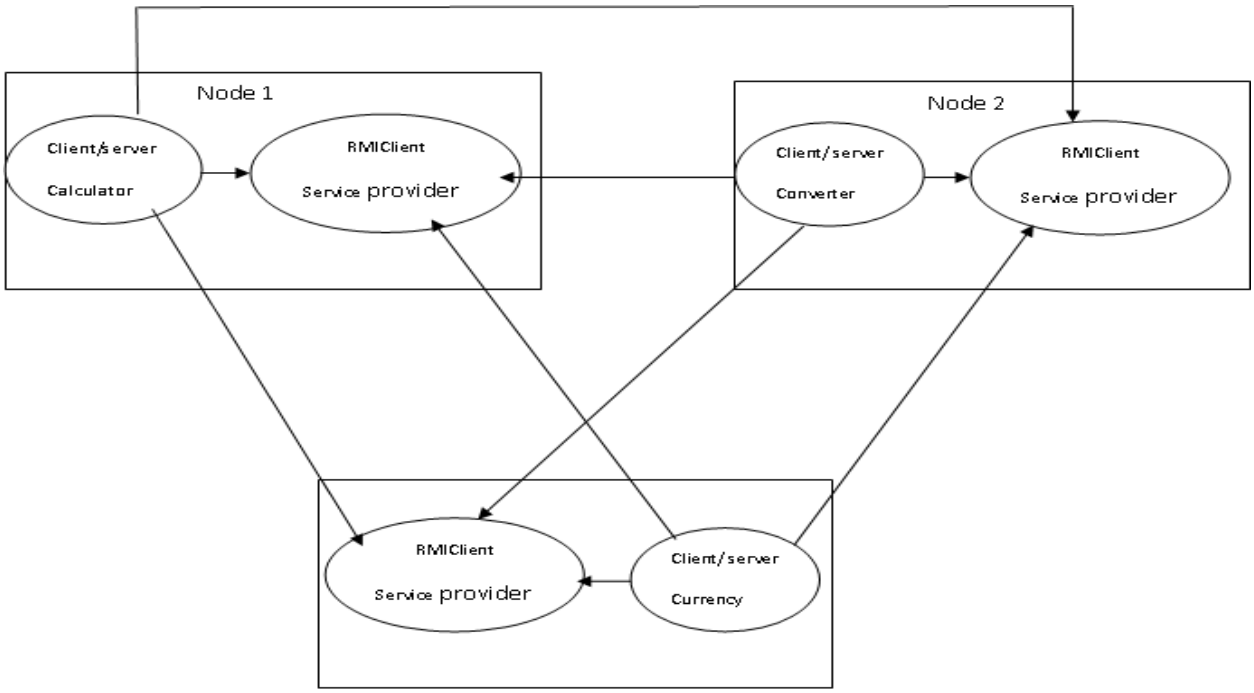

Figure 3: P2P RMI Network design.

Node 1:- consists of a calculator application and RMIClient
Node 2:- consists of converter application and RMIClient
Node 3:- consists of currency exchange application and RMIClient

Node1: Simple calculator: In this node, the calculator application is a simple RMI application that does calculation based on the imputed data and displays the result. The computation is actually done by server and the result is invoked by the client. In this node, there is also an ordinary client called RMIClient. This client serves as a service provider. In RMIClient application, a user is allowed to make a choice of the service and RMIClient invokes the service requested from the node that has the service using the IP address of that node. In order word, the virtual machine communication among the nodes uses an IP address.

Node 2: Temperature conversion: this node provides the service of converting a temperature in Celsius degree into its equivalent Fahrenheit degree and vice versa. The node is also made up of the client and server that communicate to each other for information exchange. The temperature conversion is done by the server and the result is invoked by the client. In this node, there is also an RMIClient which acts as a service provider. It can remotely call the service on each node using their IP addresses.

Node 3: Currency exchange service: Node three is dedicated to carry out the service of exchanging from one currency to another. This node is comprised of the client and server which communicate to each other and pass information back and forth. The exchange computation is done by the server and the result is invoked by the client. RMIClient in this node also communicate to the servers in node 1 and node 2 through their IP address and provides the service requested by the user and that makes the three nodes to peer with one another.

4. Platform Implementation

The result of this write up can be generated after the implementation of the design model stated above. Implementation involves transforming the design into the working code. Based on the aim of this write up which is a test whether RMI is or not a protocol for peer to peer computing on mobile devices, an RMI protocol was used for the implementation. The implementation starts by getting the system ready and setting up the library that is required for building of the product components. Next, the coding stage follows. In any RMI system, the following steps are involved in application development.

i. Declaring the interface of the server object: The interface of the server object is used as an intermediary between the server and its client. The sample code illustration is shown as follow:

```
public interface ServerInterface extends Remote {
public void service1(...) throws RemoteException;
// Other methods
}
```

Note that java.rmi.Remote interface must be extended by a server object.

ii. Define the implementation of server object: This is a java class that implements the server object interface. This class must extend java.rmi.server.UnicastRemoteObject for the implementation to be successful. This is because it uses TCP streams to facilitate point-to-point active object reference. The illustration is shown as follow:

```
public class ServerInterfaceImpl extends
UnicastRemoteObject
implements ServerInterface {
public void service1(...) throws RemoteException {
// Implement it
}
// Implement other methods
}
```

iii. Develop and register the server object: The server object is created from the server implementation class and registered with an RMI registry as is shown in the following code snapshot:

```
ServerInterface server = new ServerInterfaceImpl(...);
Registry registry = LocateRegistry.getRegistry();
registry.rebind("RemoteObjectName", obj);
```

iv. Develop client program: Developing a client program helps to locates a remote object and invokes its method as illustrated here.

```
Registry registry = LocateRegistry.getRegistry(host);
ServerInterface server = (ServerInterfaceImpl)
registry.lookup("RemoteObjectName");
server.service1(...);
```

v. Create a security policy file: Security policy file need to be created to enable the authorization of RMI to access the network resources since it is a distributed system and runs on a network under the check of security manager. This file must have a policy name extension.

In the proposed prototype system, three different applications were developed using RMI protocol which will be in three nodes as shown in the design structure in figure 3. They are calculator application, temperature converter application and currency exchange application. In any RMI application, there are four java source file that are needed to be created. These file must be in place before the success implementation of RMI system. They are the Interface, Implementation, Server and Client source file. Their implementation is explained below.

i. Simple calculator application: This application is developed to provide a service of simple calculation. The Implementation of this application requires four java source file as mentioned above to be created. The files are (*calculatorInterface, calculatorImpl, calculatorServer and calculatorClient*) which are saved with an extension dot java (.java). The *calculatorImpl* is used to implement calculator interface. The *calculatorServer* keeps all the files and gets ready to accept the client call. The RMI server must create a security manager, create one or more remote object instances and should be able to use an RMI registry to register one or more remote objects created for the success reception of the client call by *calculatorServer.* The c*alculatorClient* on the other hand makes a call to the registry in order to get a reference to the remote object and calls its method. This can be done by first assigning a security manager. Next the files are compiled with java compiler *javac* to generate the class file for each of the source file. *Stubs and skeleton* class files are used for the client and server communication.

ii. Temperature converter application: This node application is implemented in such a way that the system allows the user to select the operation he wants to perform such as converting temperature in Celsius degree to it Fahrenheit equivalent or vice versa. When the user selects a choice, the system performs the operation and displays the output result on the screen. This is implemented using the following source files: *TemperatureInterface, TemperatureImpl, TemperatureServer and TemperatureClient* which were created in development environment and saved as a java file with ***.java*** extension. This file performs exactly the same function as mentioned in simple calculation application. The files are then complied with java compiler *javac* in order to generate the class file used for client server communication. The client class file is referred to as ***stub*** and the server class file is referred to as *skeleton*.

iii. Currency exchange application: This application is used to exchange currencies from one currency to another. It accepts input value for one currency and computes the equivalent values for other currency. It consists of the following source file: *currencyInterface, currencyImpl, currencyServer and currencyClient.* These files are developed in java development environment and compiled with java compiler ***javac*** to generate the equivalent class files. Next, the currency implementation is compiled with rmic to generate the server skeleton file.

iv. RMIClient: This is an ordinary Client that actually does the work of peering the nodes together. It serves as service provider and could reside in any of the three nodes. The interface is developed in such a way that all the application can be invoked in any of the machine. In order word, it maps all the three services together using a *HashMap put function*. In this scenario, each of the service is mapped with an IP address corresponding to its node. RMIClient uses the IP address of each node to invoke the service corresponds the node. However, it gives room for a user to make a choice of the service and the invocation is carried out based on the user's choice.

5. Experimental Results and Testing

Any software developed must be tested to ensure the logical correctness of the software. Testing stage is necessary to make sure that the result obtained meets the expected outcome. This section gives an overall experience for executing the peer-to-peer application. The three application were developed in three separate machine referred to as node 1, node 2 and node 3. Each of the application was tested separately as described below before being moved into the P2P testing stage.

A. Simple calculator: Ensure that all the four source files are saved in one folder before the testing commence. Next, Launch the window DOS command prompt and follow the steps below.
   i. Make the directory that contains the calculator source file as your current directory.
   ii. Compile calculator interface with javac *calculator.java* to generate class file *calculator.class*.
   iii. Compile calculator implementation with javac *calculatorImpl.java* to generate class file *calculatorImpl.class*.
   iv. Run *rmic* to generate the stub file.
   v. Compile calculator server with javac *calculatorServer.java* to generate class file *calculatorServer.class*.
   vi. Compile calculator client with javac *calculatorClient.java* to generarte class file *calculatorClient.class*.
   vii. Create two folders, for *rmiServer* and *rmiClient*.
   viii. Copy *calculator.class*, *calculatorImpl.class* and *calculatorServer.class* file to *rmiServer* folder.
   ix. Copy *calculator.class* and *calculatorClient.class* files to *rmiClient* folder.
   x. Call *rmiServer* directory with cd rmiServer and execute server program by issuing command java calculatorServer.
   xi. This execution creates java *rmi* registry.
   xii. Open another command prompt and set rmiClient as a current directory.
   xiii. Execute the client program with java *calculatorClient*.
   xiv. The output result is then generated as shown in figure 4.

B. Temperature converter : Having saved all the four source files that are needed to run *rmi* application for temperature converter in one folder, the Dos command prompt is launched and the following steps were followed.
   i. Call the directory that contain the Temperature converter source file and make it a current directory.
   ii. Compile the four source files using a java compiler *javac* to generate the equivalent class file.
   iii. Run *rmic* to generate the stub file.
   iv. Create two folders, for *rmiServer* and *rmiClient*.
   v. Copy *converterInterface.class*, *converterImpl.class* and *converterServer.class* file to *rmiServer* folder.
   vi. Copy *converterInterface.class* and *converterClient.class* files to *rmiClient* folder.
   vii. Call *rmiServer* directory with cd *rmiServer* and execute server program by issuing command java *ConverterServer*.
   viii. This execution creates java *rmi* registry.
   ix. Open another command prompt and set *rmiClient* as a current directory.
   x. Execute the client program with java *ConverterClient*.
   xi. The output result is generated as shown in figure 5.

C. Currency converter: On Dos command prompt, the following steps were used to test java RMI currency converter application:
   i. Call the directory that contains the four source files for currency converter application and set it as your current directory.
   ii. Compile the four source files to generate their equivalent class file.
   iii. Run *rmic* to generate the stub file.
   iv. Create two folders, for *rmiServer* and *rmiClient*.
   v. Copy *currencyInterface.class*, *currencyImpl.class* and *currencyServer.class* file to *rmiServer* folder.

vi. Copy *currency.class* and *currencyClient.class* files to *rmiClient* folder.
vii. Call *rmiServer* directory with cd *rmiServer* and execute server program by issuing a command java *currencyServer*.
viii. This execution creates java *rmi* registry.
ix. Open another command prompt and set *rmiClient* as a current directory.
x. Execute the client program with java *currencyClient*.

Peer-to-Peer testing is the heart of this paper. An ordinary client interface called *RMIClient* was created for the P2P purpose. This interface performs the function of service provider. In this interface, IP addresses of each machine were mapped with the service offered by that machine. This interface can reside in any of the three machines and is capable of calling methods on the other remote machines. However, before testing commence, the servers in the three machines must have started running. Once the servers' starts running in the three machines, then *RMIClient* can now be executed in any node by following the steps listed below:

i. Open another terminal on Dos command prompt on any one of the machine.
ii. Call a directory that contains the *RMIClient* source file and make it your current directory.
iii. Compile the RMIClient source file with *javac* to generate the class file.
iv. Run the *RMIClient* by issuing a command java *RMIClient.*

The execution starts by displaying a user interface prompting the user to enter the choice of the service required (see figure 6) and subsequent execution continues. The sample result of this test is shown in figures 4, 5, 6 and 7.

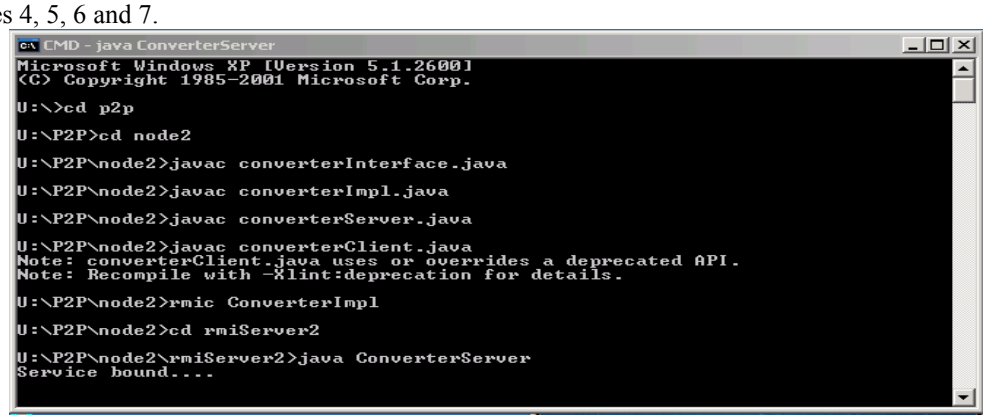

Figure4: Java Converter Server Interface

Figure 5: Java RMIClient

Figure 6: Interface for Selecting RMIClient Services

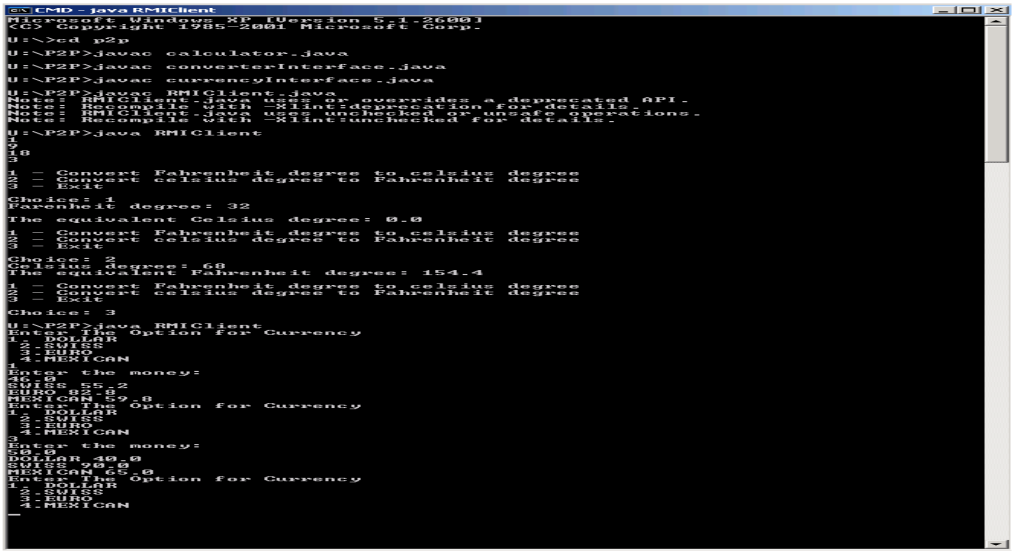

Figure 7: Java RMIClient Interface displaying the three services (including Calculator, Currency Converter, and Temperature Converter services).

5. Conclusion

In the presented work, it was observed that Java RMI is a tool for peer to peer computing on mobile devices. Useful information has been provided for researchers who are embarking on peer to peer application on hand-held devices. This paper also gave an overview of the Java 2 platform from sun Microsystems which enable any java developer to make the right choice of API and runtime environment for developing any java based application.

The initial goal of this paper which is to ascertain whether Java RMI is an appropriate protocol on mobile devices for peer to peer networking were fully achieved. Three peer to peer applications have been created to demonstrate the possibility of peer to peer RMI on mobile devices. The applications were tested in computer lab as well as on laptops (mobile devices) and the developed systems worked satisfactorily. However, for the application to work successfully on those devices java runtime environment must have been installed on that machine and IP address must have been configured either statically or dynamically on each of the device.

Finally, a java distributed programs have been developed with the same semantics and syntax used for non-distributed programs through mapping of java classes and objects to work in a distributed (multiple JVM) computing environment. The result obtained thus aids in minimizing the complexity of distributed programming language as well as recognizing the differences between remote call and local call.

References

[1] Caporuscio, M., Georgantas. N. and Issarny. V., (2007) “A Perspective on the Future of Middleware-based Software Engineering”, *Future of Software Engineering (FOSE '07)*. pp. 244-258. IEEE computer society.

[2] Capra, L. , Emmerich, W. and Mascolo, C. (2003)"CARISMA: Context-Aware Reflective middleware System for Mobile Applications," IEEE Transactions *on Software Engineering,* vol. 29, no. 10, pp. 929-945, Oct. 2003.

[3] Capra, L., Emmerich, W. and Mascolo, C. (2002) “A Micro-Economic Approach to Conflict Resolution in Mobile Computing” *Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering. (SIGSOFT '02/FSE-10),* pp. 31-40.

[4] Dehlberg, P., Ljungberg, F., and Sanneblad. J (2000) “Supporting Opportunistic Communication in Mobile setting”, *Extended abstract on Human factors in computer system (CHI’00),* pp. 111-112. ACM.

[5] Demartini, C., Montrucchio, B., Sanna, A. and Zunino.C. (2001) “ A Distributed Visualization Environment for Scientific Visualization based on Jini Technology”, *17th Spring Conference on Computer Graphics (SCCG ‘01),* Budmerice, Slovakia. Pp. 0095. IEEE Computer Society.

[6] Fossum, T., Sorensen, C.F., and Wang, A.I (2005). “Mobile Peer-to- Peer Technology used to promote spontaneous collaboration”, *International Symposium on collaborative Technologies and Systems (ISCST ‘05)* pp.48-55. St Louis, MO, USA. IEEE Computer Society.

[7] Hauck, F.J., Kapitza, R., Kirstein, M. and Schmidt, H. (2005) “FORMI: an RMI extension for adaptive

applications", *Proceedings of the 4th workshop on Reflective and adaptive middleware systems (ARM '05)*, pp 1-6. ACM.

[8] Jackson, M. A., (1983) "System Development" Englewood Cliffs, N.J.; London: Prentice-Hall International,

[10] Kortuem, G. (2002) "A methodology and Software platform for building wearable communities", PHD thesis, University of Oregon.

[11] Mintian. Z, and Qilin. L, (2010) "The State of the Art in Middleware," *2010 International Forum on Information Technology and Applications*, vol. 1, pp.83-85, IEEE computer society.

[12] Yang, Y. and Yeung, K. (2009) " Mobile information Retrieval in a Hybrid P2P Environment", *Proceedings of the 6th International Conference on Mobile Technology, Application & System(Mobility '09)*.ACM.

This academic article was published by The International Institute for Science, Technology and Education (IISTE). The IISTE is a pioneer in the Open Access Publishing service based in the U.S. and Europe. The aim of the institute is Accelerating Global Knowledge Sharing.

More information about the publisher can be found in the IISTE's homepage: http://www.iiste.org

CALL FOR PAPERS

The IISTE is currently hosting more than 30 peer-reviewed academic journals and collaborating with academic institutions around the world. There's no deadline for submission. **Prospective authors of IISTE journals can find the submission instruction on the following page:** http://www.iiste.org/Journals/

The IISTE editorial team promises to the review and publish all the qualified submissions in a **fast** manner. All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Printed version of the journals is also available upon request of readers and authors.

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar