LIVE HELP DESK AN ONLINE CUSTOMER SUPPORT SYSTEM

By

SGT Malik Kashif Mudassar

PC Omer Moeen

TC Fahd Niazi

PC Ghazali Farooq

Submitted to Faculty of Computer Science, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of BE Degree in Computer Software Engineering

April 2008

ABSTRACT

LIVE HELP DESK

The objective of our project is to implement an Online Customer Support Communication Platform for managing all online interactions so that visitors of a website can communicate with sales person /agent through their web browsers without need of any additional software required to be installed on the client side. Live chat is usually a text-based communication that happens online between a customer and a merchant. It's a kind to the instant messaging you might do with a friend through Apple's iChat or AOL's or any other Instant Messenger. Live chat extends communication opportunities between buyer and seller while maximizing the customer's overall satisfaction with their website experience.

JAVA Remote Method Invocation (RMI) technology is used to access remote database that is written using Microsoft Access.

The main objectives are Call Waiting, Forwarding and Transferring, Load Balancing and TimeStamping, Emailing, File/Snapshot Transferring, Customers Continuous Feedbacks/Online Surveys, Technical Support in Sales and Marketing, Reports Generation.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of any other award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose unflinching support and unstinting cooperation, a work of this magnitude would not have been possible

ACKNOWLEDGMENTS

We are eternally grateful to Almighty Allah for bestowing us with the strength and resolve to undertake and complete the project.

We gratefully recognize the continuous supervision and motivation provided to us by our Project Supervisor, Asst. Prof Asim Elahi .

We would also like to thank our Co-Supervisor, Maj. Ather Mohsin Zaidi for his constant guidance and help he extended to us whenever we needed it.

We are especially grateful to HoD (CS Dept) Lt Col. Naveed Sarfraz Khattak for his heartfelt support in fulfilling our project goals.

We deeply treasure the unparallel support and tolerance that we received from our friends for their useful suggestions that helped us in completion of this project. We are also deeply obliged to our families for their never ending patience and support for our mental peace and to our parents for the strength that they gave us through their prayers.

A word of thanks to the Military College of Signals (MCS) as it had been our foundation and has made us capable to undertake the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Preface

The Report covers comprehensively all the basic building blocks of our project. The JAVA RMI and related stuff is explained in accordance with our Implementation strategy. The necessary Architectures including Java-RMI architecture, messaging architecture, applet implementation and sequence diagrams are elaborated to get a detailed understanding of our approach and the advantages that comes with it. Necessary information for this Draft was gathered from Internet.

1.2 Introduction

The basic objective of our project is to implement Online Customer Support service. This would be accomplished through the incorporation of RMI Interface on the host side. It communicates with the customer by downloading an applet on it as the customer clicks the chat logo on the website. Accomplishing the above mentioned objective would result in an efficient online messaging which has numerous advantages. The objective of this Software Requirement Specification document is to provide key software functionalities of our project as well as the applications of our software along with its scope.

1.3 Background

In the past Online Help Desks have been implemented using various platforms like Delphi, Visual C, XML, SQL and even Java. But the use of JAVA-RMI with a remote access database is new idea that provides a higher level of abstraction than socket-level programming. RMI programs are much easier to maintain than socket-

level programs. An RMI server can be modified or moved to another host without the need to change the client application (apart from resetting the URL for locating the server). In the conventional client–server mechanism, a client sends a message to the server that replies with a result. The reverse is not possible: a server cannot invoke the methods on a client. However, the RMI mechanism supports the idea of callbacks in which the server invokes methods on the client. This facility enables interactive distributed applications to be developed. Secondly the project helps make the website more interactive as the viewers can get instant help through chatting live with the support or marketing personnel.

1.4 Scope

The design and lookup of the live help application would be similar to the live help software available. It would prompt the user to select kind of help he wants i-e marketing or troubleshooting etc. After the selection the customer will be guided to chat with the concerned desk. The application software would be providing smooth communication in the form of messages. We would not be implementing VOIP related applications in our software .So our software will not include features like Voice Chat and Video Conferencing.

1.5 Assumptions and Dependencies

We would be using Java RMI (Remote Method Invocation) in our project. We would be utilizing all the java techniques that determine our project scope. All our work would be done on java platforms available that are compatible with Windows OS.

1.6 Organization of SRS

Chapter 3 highlights the key characteristics of SRS and explains how it is organized in our project.

Chapter 2

Literature Review

2.1 Online Support Community

An Online Support Community is generally a discussion board for questions and answers. Ideally, it is specifically designed to allow users to easily and quickly find answers to questions already asked and answered by others or, if they can't find an answer, to ask their questions for expert customers to answer. A good board allows quick escalation of more difficult issues to your own support representative. It is also designed to motivate other users to answer questions fast and accurately. The point of an online support community is Questions and Answers not opinion, not commentary, not gossip, and not personal conversations. It is about solving problems, learning, and sharing ideas.

A Help Desk is basically an open source live support solution that helps customer support with live help functionality that can be proactively pushed to visitors to your site or requested by the consumer. It includes a large range of features to allow multiple operators, multiple departments and multiple languages to be used.

2.2 Benefits of Customer-to-Customer Support

Customer support demands continue to escalate for most companies, and delivering support is becoming more complex. Avoiding costly contacts should be top on every customer support executive agenda.

Few companies, however, successfully leverage one of the most valuable company assets, the loyal, dedicated, and expert customers in your installed base. Your customers represent a huge, untapped, and virtually free support resource. Often they

are also your strongest advocates. Among the few firms that recognize this resource and attempt to leverage it by creating an online support community, precious few implement appropriate plans. Ensuring the customer support community opportunity ensures your company with the following different benefits:

2.2.1 **Cost Avoidance**

For many companies, a 5% improvement in call avoidance can save $1 million or more in expenses a year. You gain huge leverage when you understand how to use your own expert customers to help answer questions. Achieving such improvement requires very little investment on your part.

2.2.2 **Increased Value**

Customers now demand service in Internet time. Where companies once prided themselves in handling 99% of all questions within 24 to 48 hours, even an hour wait can become an eternity in today's world of instant communication. Most companies can not afford to deliver such support without "breaking the bank." Expert customers allow you to deliver faster support without significant increases in cost. Plus, customers are often available when your team is not 24 hours per day, 7 days per week and 365 days per year.

2.2.3 **Greater Customer Satisfaction**

Most companies measure the performance of their support representatives by how many calls they handle per day, as well as the length of time of each call. These metrics encourage representative to spend as little time as possible with each customer. By contrast, expert customers will often spend as much time as someone needs. Those seeking help often get a more complete solution from your customers while being taught to be more self-sufficient. Moreover, expert customers have real

world experiences that your support representative does not have offering better insight and understanding about customer problems.

2.2.4 **Increased Revenue**

A vibrant online support community allows new and prospective clients to gain greater confidence with your products more quickly and easily. Customers have an easy way to network with each other, getting answers to questions, along with advice and recommendations. From whom would it be better for your new and prospective customers to learn than your most loyal, dedicated, and expert enthusiasts?

2.3 Features of a good Customer Support System

i. The system should store all its information in a relational database.
ii. Customer calls to the support desk should be logged.
iii. It should be possible to add and search for information on problems that have already been addressed, and which are decided for inclusion in a coming version of the product.
iv. It should be possible for several GUI clients to use the support system at the same time.
v. It should be possible to view statistics on various data in the support database.

Table 2-1 clearly mentions the most winning features of a good Customer Support System.

Table 2-1 A good Support System Features

Efficient	One agent can support multiple customers
Anonymous	Customers do not have to reveal personal information
Call Tracking	Ability to monitor support sessions in real time
Business Knowledge	Knowledge base automatically grows and stays up to date
Easy complex task resolution	Customers can easily get a support transcript to review the step by step processes for complex tasks
Competitive IT edge	Rapidly becoming a standard in eCommerce industry
A New Standard	Customers now expect live chat support
Customer Satisfaction	Improves customer impressions of your customer service
Customer Loyalty	Customers will return because of your customer service
Longevity	Proven track record

Requirements Analysis

3.1 Specific Requirements

Live Help Desk is a standardized java interface for online customer interactions. Then, we will develop an applet that will be downloaded from to the client through which communication will take place. The task stated will be the essential input to our system. After a thorough understanding of above task we would be able to write a java code that supports the above mentioned requirements. The essential outcome will be a Customer Support System with functionalities stated earlier in this report.

3.1.1 External Interface Requirements

We would be utilizing java software development kit for our project. Besides this all the hardware interface requirements for communication over an internet would be fulfilled. This would include identification of source and destination involved in communication along with the port numbers being utilized. Data encryption and security along the communication medium also needs to be implemented through these interfaces. Databases will be used to incorporate various features in the system.

3.1.1.1 Software Interfaces

The software that is required in the successful incorporation of our Customer Support System include

a) Microsoft Windows

b) Java IDE i.e., JBuilder 5

c) Java Runtime Environment version. 1.5

3.1.1.2. Hardware Interfaces

As it is defined that Live Help Desk service can be used any where in the world. The thing that needs is the connectivity between the nodes and that is established through internet. But in our case we would be implementing with Transmission Control Protocol as the basic transmission protocol. The connecting wire would be RS-45.

3.1.1.3. Communication Interfaces

Instant Messaging with the client will be implemented just like any other IM client available in the market. It will be providing you many facilities that is required in making this communication smoother like sales and marketing and related technical support.

3.2 Performance Requirements

Performance requirements (PR) are necessary for system design and development. The Performance Requirements of Live Help Desk are shown in Table 3-1

Table 3-1 Performance requirements

ID	Performance Requirements	Criticality
PR-01	The system shall support source party to get involved in communication with any of the support desk.	High
PR-02	There should be no considerable delay from applications perspective	High
PR-03	The system shall cope all the security related issues and data encryption	High

3.3 Functional Requirements

In software engineering, a functional requirement defines a function of a software-system or its component. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that show how a use case is to be fulfilled. Table 3-2 shows functional requirements of Live Help Desk:

Table 3-2 Functional requirements

ID	Functional Requirements	Type
FR01	The system should download an applet as the user clicks the chat logo.	Normal
FR02	The system must prompt option of choosing the type of support required i-e marketing or support	Exciting
FR03	The system shall provide a secure and reliable communication	Normal
FR04	The system shall close the session once the communicating parties have terminated	Normal
FR05	The system should check for the validation of information at the destination end.	Expected
FR06	The system should provide smooth communication with very little delay incurred.	Expected

3.4 Security/Privacy Requirements

Security and privacy is a key ingredient of any communication and in our implementation too RMI addresses security issues by use of RMI Security manager and Security files.

3.5 Reliability Requirements

There needs to be no loss or interruption in communication by a stranger. If a new customer wants to chat it should not interrupt the present session. The communication mechanism needs to be fully secure and reliable. This reliability requirement is fulfilled by TCP.

3.6 Design Constraints

The security and reliability is a key issue related to design constraints. It limits the options since it requires looking into numerous no of issues for secure communication.

Remote Method Invocation

4.1 Introduction

Java Remote Method Invocation (RMI) allows programmer to execute remote function class using the same semantics as local functions calls. The server must first bind its name to the registry. The client lookup the server name in the registry to establish remote references. The Stub serializing the parameters to skeleton, the skeleton invoking the remote method and serializing the result back to the stub.

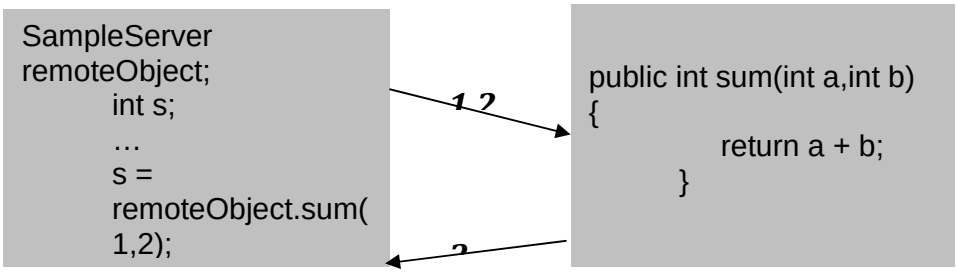


Figure 4-1 Remote function calls using Local function calls

Here is general **RMI architecture**:

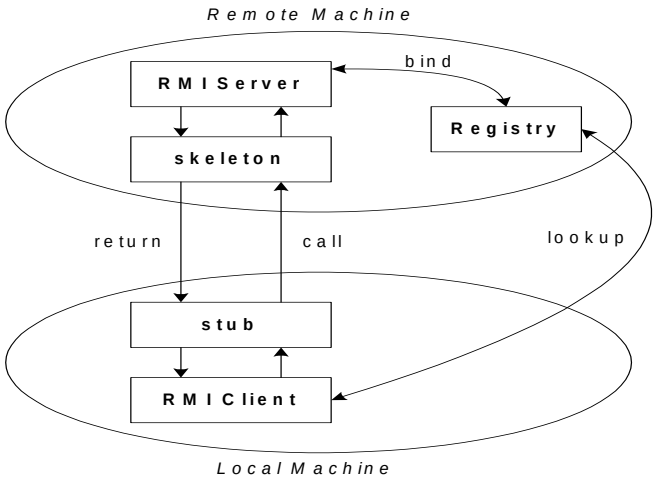


Figure 4-2 RMI architecture

4.2 RMI Interfaces

The RMI architecture is based on one important principle: the definition of behaviour and the implementation of that behaviour are separate concepts. RMI allows the code that defines the behaviour and the code that implements the behaviour to remain separate and to run on separate JVMs. This fits nicely with the needs of a distributed system where clients are concerned about the definition of a service and servers are focused on providing the service.

Specifically, in RMI, the definition of a remote service is coded using a Java interface. The implementation of the remote service is coded in a class. Therefore, the key to understanding RMI is to remember that interfaces define behaviour and classes define implementation.

While the Figure 4.3 illustrates this separation, remember that a Java interface does not contain executable code.

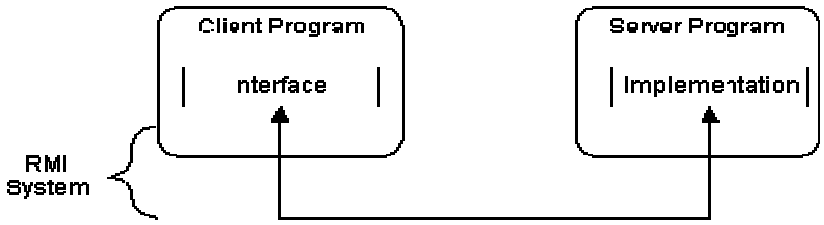


Figure 4-3 RMI Interface

RMI supports two classes that implement the same interface. The first class is the implementation of the behaviour, and it runs on the server. The second class acts as a proxy for the remote service and it runs on the client. This is shown in the following diagram.

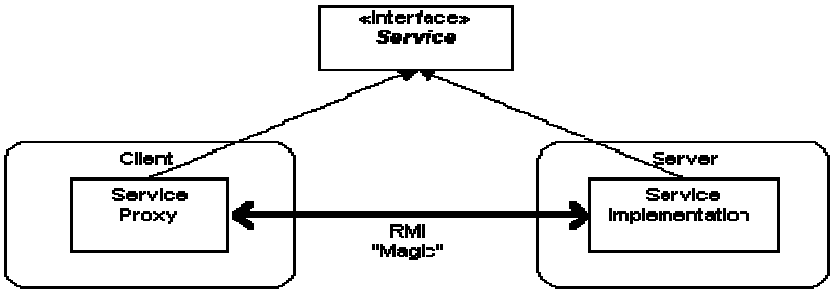


Figure 4-4 RMI Service Proxy and Implementation

A client program makes method calls on the proxy object, RMI sends the request to the remote JVM, and forwards it to the implementation. Any return values provided by the implementation are sent back to the proxy and then to the client's program.

4.3 RMI Architecture Layers

The RMI implementation is essentially built from three abstraction layers. The first is the Stub and Skeleton layer, which lies just beneath the view of the developer. This layer intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI service.

The next layer is the Remote Reference Layer. This layer understands how to interpret and manage references made from clients to the remote service objects. In JDK 1.1, this layer connects clients to remote service objects that are running and exported on a server. The connection is a one-to-one (unicast) link. In the Java 2 SDK, this layer was enhanced to support the activation of dormant remote service objects via Remote Object Activation.

The transport layer is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

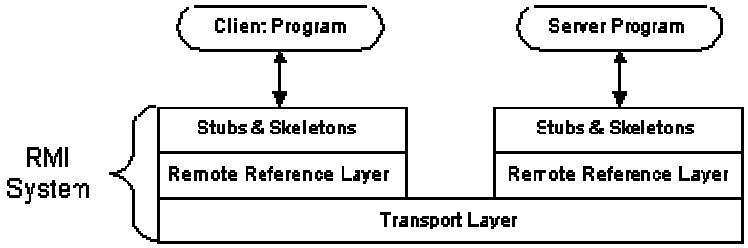


Figure 4-5 RMI Architectural layers

By using a layered architecture each of the layers could be enhanced or replaced without affecting the rest of the system. For example, the transport layer could be replaced by a UDP/IP layer without affecting the upper layers.

4.3.1 Stub and Skeleton Layer

The stub and skeleton layer of RMI lie just beneath the view of the Java developer.

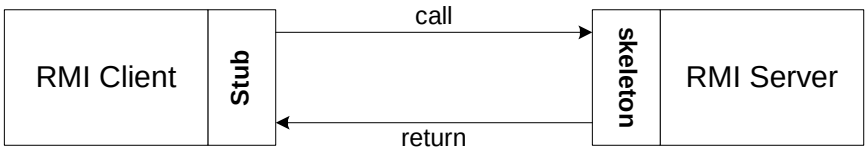


Figure 4-6 RMI Stub and Skeleton Layer

A client invokes a remote method, the call is first forwarded to stub. The **stub** is responsible for sending the remote call over to the server-side skeleton. The stub opening a socket to the remote server, marshaling the object parameters and forwarding the data stream to the skeleton. The stub class plays the role of the proxy.

A **skeleton** contains a method that receives the remote calls, unmarshals the parameters, and invokes the actual remote object implementation. A skeleton is a helper class that is generated for RMI to use. The skeleton understands how to communicate with the stub across the RMI link. The skeleton carries on a

conversation with the stub; it reads the parameters for the method call from the link, makes the call to the remote service implementation object, accepts the return value, and then writes the return value back to the stub.

4.3.2 Remote Reference Layer

The Remote Reference Layers defines and supports the invocation semantics of the RMI connection. This layer provides a `RemoteRef` object that represents the link to the remote service implementation object.

The stub objects use the `invoke()` method in `RemoteRef` to forward the method call. The `RemoteRef` object understands the invocation semantics for remote services.

The JDK 1.1 implementation of RMI provides only one way for clients to connect to remote service implementations: a unicast, point-to-point connection. Before a client can use a remote service, the remote service must be instantiated on the server and exported to the RMI system. (If it is the primary service, it must also be named and registered in the RMI Registry).

The Java 2 SDK implementation of RMI adds a new semantic for the client-server connection. In this version, RMI supports activatable remote objects. When a method call is made to the proxy for an activatable object, RMI determines if the remote service implementation object is dormant. If it is dormant, RMI will instantiate the object and restore its state from a disk file. Once an activatable object is in memory, it behaves just like JDK 1.1 remote service implementation objects.

Other types of connection semantics are possible. For example, with multicast, a single proxy could send a method request to multiple implementations simultaneously and

accept the first reply (this improves response time and possibly improves availability). In the future, Sun may add additional invocation semantics to RMI.

4.3.3 Transport Layer

The Transport Layer makes the connection between JVMs. All connections are stream-based network connections that use TCP/IP. Even if two JVMs are running on the same physical computer, they connect through their host computer's TCP/IP network protocol stack. The following diagram shows the tolerant use of TCP/IP connections between JVMs.

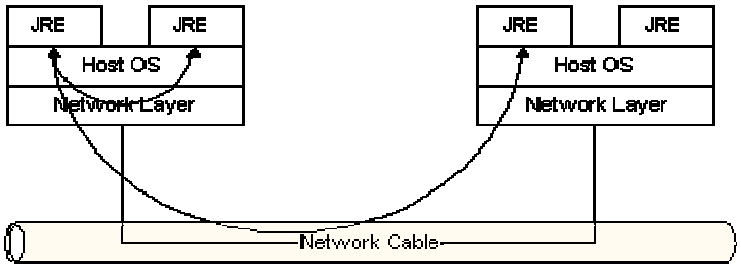


Figure 4-7 RMI Transport Layer

As you know, TCP/IP provides a persistent, stream-based connection between two machines based on an IP address and port number at each end. Usually a DNS name is used instead of an IP address. In the current release of RMI, TCP/IP connections are used as the foundation for all machine-to-machine connections.

On top of TCP/IP, RMI uses a wire level protocol called Java Remote Method Protocol (JRMP). JRMP is a proprietary, stream-based protocol that is only partially specified is now in two versions. The first version was released with the JDK 1.1 version of RMI and required the use of Skeleton classes on the server. The second version was released with the Java 2 SDK. It has been optimized for performance and

does not require skeleton classes. Some other changes with the Java 2 SDK are that RMI service interfaces are not required to extend from `java.rmi.Remote` and their service methods do not necessarily throw `RemoteException`.

The RMI transport layer is designed to make a connection between clients and server, even in the face of networking obstacles.

While the transport layer prefers to use multiple TCP/IP connections, some network configurations only allow a single TCP/IP connection between a client and server (some browsers restrict applets to a single network connection back to their hosting server).

In this case, the transport layer multiplexes multiple virtual connections within a single TCP/IP connection.

4.4

Steps of developing an RMI System:

1. Define the remote interface
2. Develop the remote object by implementing the remote interface.
3. Develop the client program.
4. Compile the Java source files.
5. Generate the client stubs and server skeletons.
6. Start the RMI registry.
7. Start the remote server objects.
8. Run the client

4.4.1. Defining the Remote Interface

To create an RMI application, the first step is defining of a remote interface between the client and server objects.

```
/* SampleServer.java */
import java.rmi.*;

public interface SampleServer extends Remote
{
  public int sum(int a,int b) throws RemoteException;
}
```

4.4.2. Develop the remote object by implement the remote interface

The server is a simple unicast remote server. Create server by extending java.rmi.server.UnicastRemoteObject.

The server uses the RMISecurityManager to protect its resources while engaging in remote communication.

```
/* SampleServerImpl.java */
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class SampleServerImpl extends UnicastRemoteObject
                        implements SampleServer
{  SampleServerImpl() throws RemoteException
  {
    super();}
```

The server must bind its name to the registry, the client will look up the server name. Use java.rmi.Naming class to bind the server name to registry. In this example the name call “SAMPLE-SERVER”. In the main method of your server object, the RMI security manager is created and installed.

```
/* SampleServerImpl.java */
  public static void main(String args[])
  {
     try
     {
 System.setSecurityManager(new RMISecurityManager());  //set the security manager
    //create a local instance of the object
    SampleServerImpl Server = new SampleServerImpl();
    //put the local instance in the registry
   Naming.rebind("SAMPLE-SERVER" , Server);
   System.out.println("Server waiting.....");
     }
 catch (java.net.MalformedURLException me)      {
 System.out.println("Malformed URL: " + me.toString());   }
 catch (RemoteException re)  {
 System.out.println("Remote exception: " + re.toString());  }  }
```

Implement the remote mehtods

```
/* SampleServerImpl.java */

 public int sum(int a,int b) throws RemoteException

 {  return a + b;  } }
```

4.4.3. Develop the client program

In order for the client object to invoke methods on the server, it must first look up the name of server in the registry. You use the java.rmi.Naming class to lookup the server name. The server name is specified as URL in the from (rmi:// host:port/name). Default RMI port is 1099. The name specified in the URL must exactly match the name that the server has bound to the registry. In this example, the name is "SAMPLE-SERVER" The remote method invocation is programmed using the remote interface name (remoteObject) as prefix and the remote method name (sum) as suffix.

```
import java.rmi.*;
import java.rmi.server.*;
public class SampleClient
{
  public static void main(String[]  args)
  {
   // set the security manager for the client
   System.setSecurityManager(new RMISecurityManager());

   //get the remote object from the registry
     try
      {
```

```
System.out.println("Security Manager loaded");
String url = "//localhost/SAMPLE-SERVER";

SampleServer remoteObject = (SampleServer)Naming.lookup(url);
System.out.println("Got remote object");

System.out.println(" 1 + 2 = " + remoteObject.sum(1,2) );
  }
catch (RemoteException exc) {
System.out.println("Error in lookup: " + exc.toString()); }
catch (java.net.MalformedURLException exc) {
System.out.println("Malformed URL: " + exc.toString());   }
catch (java.rmi.NotBoundException exc)  {
System.out.println("NotBound: " + exc.toString());
  }  }  }
```

4.4.4. Compile the Java source files & generate the client stubs and server skeletons

Assume the program compile and executing at elpis on ~/rmi. Once the interface is completed, you need to generate stubs and skeleton code. The RMI system provides an RMI compiler (rmic) that takes your generated interface class and procedures stub code on its self.

```
elpis:~/rmi> set CLASSPATH=”~/rmi”
elpis:~/rmi> javac SampleServer.java
elpis:~/rmi> javac SampleServerImpl.java
elpis:~/rmi> rmic SampleServerImpl
```

elpis:~/rmi> javac SampleClient.java

4.4.5 Start the RMI registry

The RMI applications need install to Registry. And the Registry must start manual by call rmiregisty. The RMI registry is a Java program named rmiregistry which, when it is executing, maintains a list of references to objects that have been registered with it. An object is registered by a server using the rebind method from the class Naming (part of the java.rmi package). The first argument of the rebind method is a String that specifies: where the registry is located — by quoting the IP address of its host (localhost can be used when the registry is on the same host as the server) and the port on which the registry listens for communications (by default this is 1099, and can be omitted); a programmer defined name by which clients can gain access to the object. The RMI registry listens to a given port on the remote machine. The default port of RMI is **1099**.

To launch the RMI registry type: rmiregistry & or, if you wish to listen to a port other than the default one, simple provide the port number:rmiregistry 1234 & keep in mind that the RMI registry will continue to work, even when your session terminates, so be kind and kill the process before you leave. The rmiregistry us uses port 1099 by default. You can also bind rmiregistry to a different port by indicating the new port number as **:** rmiregistry <new port>

elpis:~/rmi> rmiregistry

remark: On Windows, you have to type in from the command line:

> *start rmiregistry*

4.4.6 Start the remote server objects & Run the client

Once the Registry is started, the server can be started and will be able to store itself in the Registry.Because of the grained security model in Java 2.0, you must setup a security policy for RMI by set java.security.policy to the file policy.all

elpis:~/rmi> java –Djava.security.policy=policy.all SampleServerImpl

elpis:~/rmi> java –Djava.security.policy=policy.all SampleClient

remark: Java 2 Policy Files

In Java 2, the java application must first obtain information regarding its privileges. It can obtain the security policy through a policy file. In above example, we allow Java code to have all permissions, the contains of the policy file policy.all is:

```
grant {
    permission java.security.AllPermission;  };
```

Now, we given an example for assigning resource permissions:

```
grant {
permission java.io.filePermission “/tmp/*”, “read”, “write”;
permission java.net.SocketPermission “somehost.somedomain.com:999”,”connect”;
permission java.net.SocketPermission “*:1024-65535”,”connect,request”;
permission java.net.SocketPermission “*:80”,”connect”;
};
```

4.5 Comparison of RMI with the client–server socket mechanism

RMI is a higher level of abstraction than socket-level programming. It enables the details of socket servers, sockets and data streams to be hidden. Although we have not explicitly mentioned it, RMI uses a hidden multithreading system that would otherwise have to be implemented in a socket-layer.

RMI clients can invoke a server method directly but socket-level programming allows only values to be passed that must then be decoded and turned into a method call by the server. This decoding is performed automatically by RMI stubs (marshalling).

RMI programs are much easier to maintain than socket-level programs. An RMI server can be modified or moved to another host without the need to change the client application (apart from resetting the URL for locating the server).

RMI is implemented using socket-level programming. Socket-level programming is viewed as a primitive mechanism that is prone to error and you should use RMI in preference. This is similar to the relationship between semaphores and higher level constructs such as monitors.

In the conventional client–server mechanism, a client sends a message to the server that replies with a result. The reverse is not possible: a server cannot invoke the methods on a client. However, the RMI mechanism supports the idea of callbacks in which the server invokes methods on the client. This facility enables interactive distributed applications to be developed.

While the details of this mechanism are outside the scope of this unit, the following example illustrates what has to happen. The Internet is becoming popular for game playing among groups of people. Each player in a group runs a client program that sends remote method invocations to a central client. The central client's purpose is to coordinate the players' activities by maintaining the state of the game and communicating each player's moves to the other players. As part of this process, the server invokes client methods. However, a server cannot invoke a client's methods directly because the client is not a remote object. However, through Java's callback mechanism, each client passes a stub to the server. The stub contains an instance of the client so that the server can invoke the client's methods.

JAVA Database Connectivity

5.1 Introduction

Java Database Connectivity (JDBC) is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

The Java 2 Platform, Standard Edition, version 1.4 (J2SE) includes the JDBC 3.0 API together with a reference implementation JDBC-to-ODBC Bridge, enabling connections to any ODBC-accessible data source in the JVM host environment. This Bridge is native code (not Java), closed source, and only appropriate for experimental use and for situations in which no other driver is available, not least because it provides only a limited subset of the JDBC 3.0 API, as it was originally built and shipped with JDBC 1.0 for use with old ODBC v2.0 drivers.

5.2 Overview

JDBC has been part of the Java Standard Edition since the release of JDK 1.1. The JDBC classes are contained in the Java package **java.sql**. Starting with version 3.0, JDBC has been developed under the Java Community Process. JSR 54 specifies JDBC 3.0, JSR 114 specifies the JDBC Rowset additions, and JSR 221 is the specification of JDBC 4.0. JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may

be query statements such as SELECT. Additionally, stored procedures may be invoked through a JDBC connection. JDBC represents statements using one of the following classes:

Statement – the statement is sent to the database server each and every time.

PreparedStatement – the statement is cached and then the execution path is pre determined on the database server allowing it to be executed multiple times in an efficient manner.

CallableStatement – used for executing stored procedures on the database.

Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many rows were affected in the database. These statements do not return any other information.

5.3 Key Features

The Key Features Java Database Connectivity has the following key features:

5.3.1 Full Access to Metadata

The JDBC API provides metadata access that enables the development of sophisticated applications that need to understand the underlying facilities and capabilities of a specific database connection.

5.3.2 No Installation

A pure JDBC technology-based driver does not require special installation; it is automatically downloaded as part of the applet that makes the JDBC calls.

5.3.3 Database Connection Identified by URL

JDBC technology exploits the advantages of Internet-standard URLs to identify database connections. The JDBC API includes an even better way to identify and connect to a data source, using a DataSource object that makes code even more portable and easier to maintain.

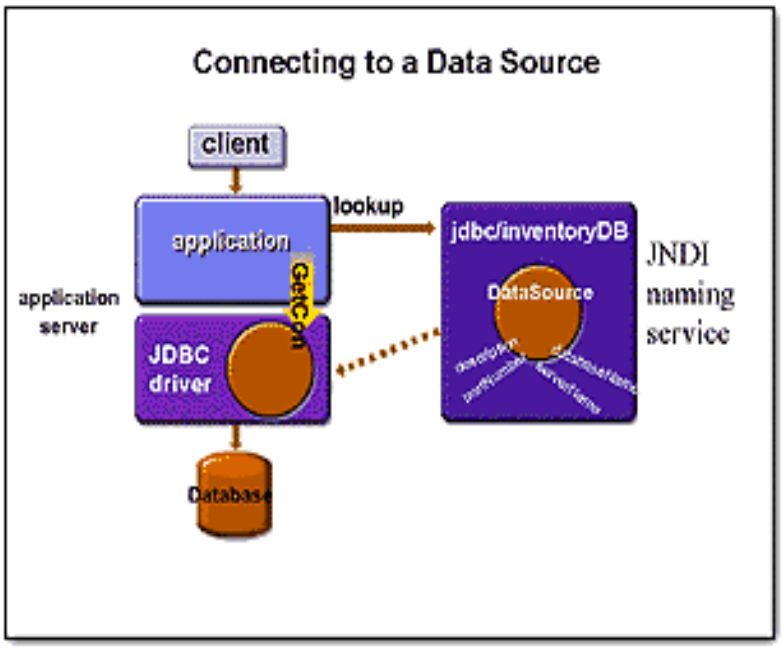


Figure 5-1 Connecting to Data Source

In addition to this important advantage, DataSource objects can provide connection pooling and distributed transactions, essential for enterprise database computing. This functionality is provided transparently to the programmer.

5.3.4 Included in the Java Platform

As a core part of the Java 2 Platform, the JDBC API is available anywhere that the platform is. This means that your applications can truly write database applications once and access data anywhere. The JDBC API is included in both the Java 2 Platform, Standard Edition (J2SE) and the Java 2 Platform, Enterprise Edition (J2EE), providing server-side functionality for industrial strength scalability.

5.4 Advantages of JDBC Technology

Java Database Connectivity has the following different advantages which make it useful in DataBase Applications:

5.4.1 Leverage Existing Enterprise Data

With JDBC technology, businesses are not locked in any proprietary architecture, and can continue to use their installed databases and access information easily even if it is stored on different database management systems.

5.4.2 Simplified Enterprise Development

The combination of the Java API and the JDBC API makes application development easy and economical. The JDBC API is simple to learn, easy to deploy, and inexpensive to maintain.

5.4.3 Zero Configurations for Network Computers

With the JDBC API, no configuration is required on the client side. With a driver written in the Java programming language, all the information needed to make a connection is completely defined by the JDBC URL or by a DataSource object registered with a Java Naming and Directory Interface (JNDI) naming service. Zero configurations for clients support the network computing paradigm and centralize software maintenance.

5.5 Accessing MS Access from Java

In JAVA the Microsoft DataBase Access can be accessed in a stepwise convenient method described below:

5.5.1 Prerequisites

An installed and licensed Easysoft JDBC-ODBC Bridge (JOB) server on a supported Windows platform that has Microsoft Office installed. An existing Access database file (.mdb) on the Windows machine.

5.5.2 Configuring the Microsoft Access ODBC Data Source

You will find the ODBC Administrator within Administrative Tools from your Control Panel.

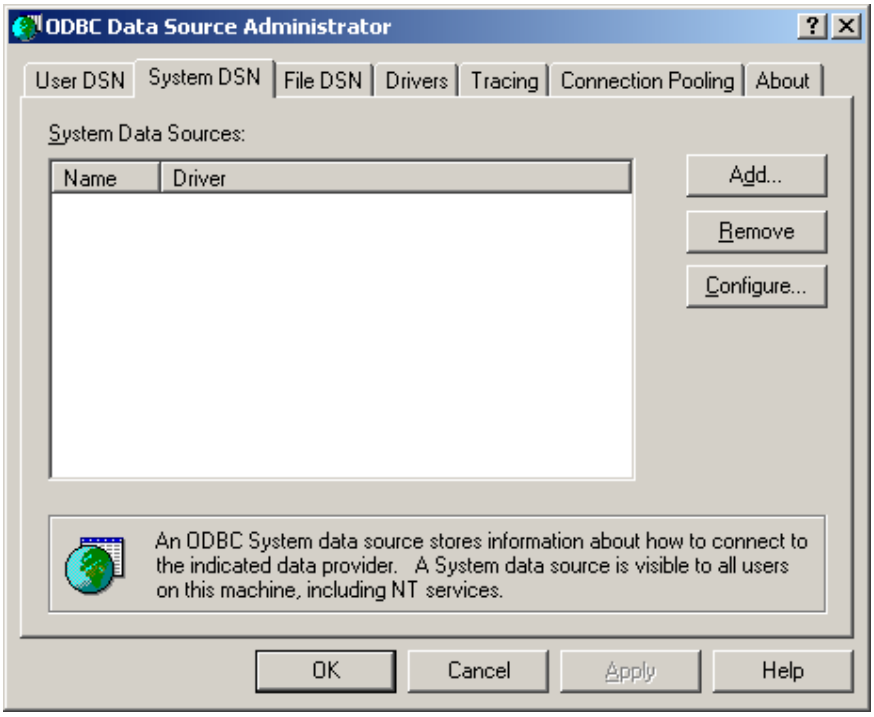


Figure 5-2 ODBC Data Source Administrator

From here you will create your new System DSN. Click the Add button and then select the Microsoft Access Driver and click Finish.

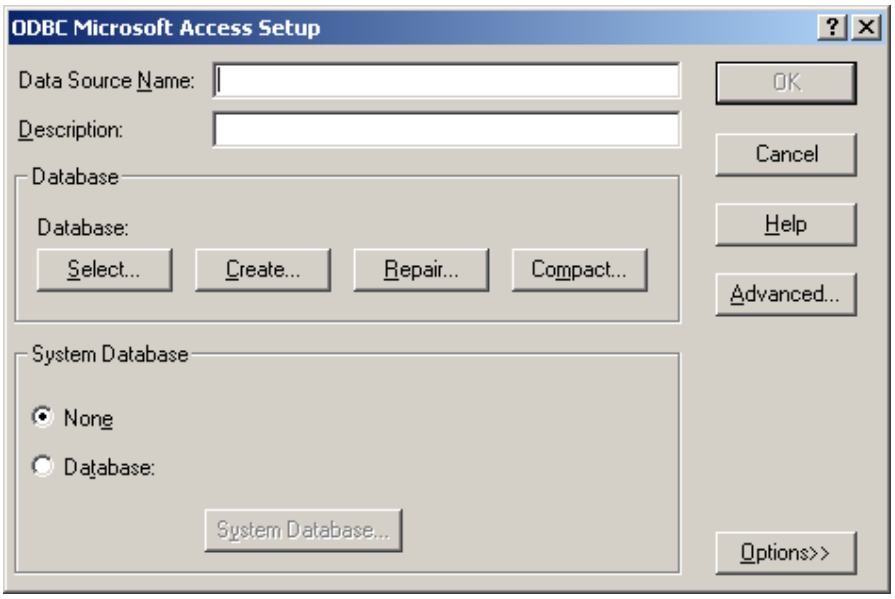


Figure 5-3 ODBC Microsoft Access Setup

Give your DSN a meaningful name and click Select. Here you will be able to browse to your existing **.mdb** file.

Once you have selected the database you want to access, click OK and then OK again to exit the dialog box. You have now created your data source.

The Live Help Desk

6.1 Features

Live Help Desk is an online customer support system that provides the following different functionalities both on the Client and Agent side.

6.1.1 Agent Side Functionalities

- Sign in/Sign out
- Agent picture display
- TimeStamping and Waiting Queue
- Chatting
- Technical Support in Sales and Marketing
- Reports Generation

6.1.2 Client Side Functionalities

- Department Selection
- Chatting
- Emailing Sessions
- Feedback

6.2 Using the RMI Server

The RMI server can be used in our system by a step wise process that is explained below:

6.2.1 The Server Interface

The server Interface not only extends the Remote Interface but also provide methods that throw Remote Exception. In our system it is provided by ServerInterface.java file.

6.2.2 The Server Implementation

The Server Implementation provides classes that implement methods defined in interface. It is provided in the ServerImpl.java file.

6.2.3 The Server

The Server.java file given is meant to bind the ServerImpl object with the RMI Registry Service.

6.3 RMI REGEISTRY

The RMI Registry is created using three steps

- Object Creaton
- URL of Client
- Binding Object with the URL

6.4 Messaging Architecture

The messaging architecture of the system is briefly explained by the Figure 6-1. It consist of TCP listeners at the port 6060. We can bind the TCP Listener at any port we wish. Every user agent is check against the database to for authentication purposes through the RMI server.

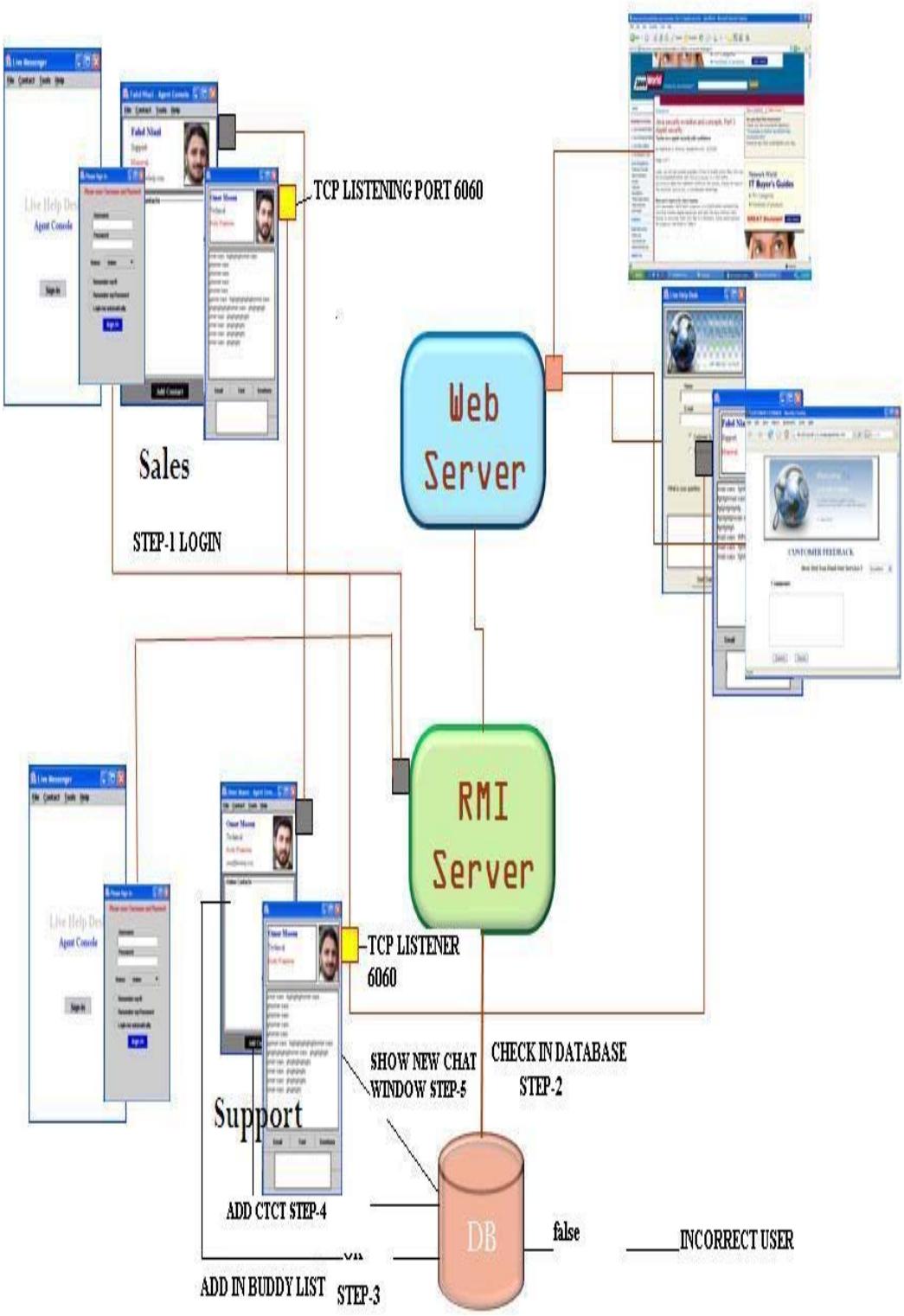


Figure 6-1 Messaging Architecture

6.5 The User Interface

Live Help Desk is a user friendly GUI oriented system that provides enhanced interfaces for different functionalities like sign-in capability, viewing online users, emailing, feedback etc. All these features along with the User Interfaces are discussed here.

6.5.1 User Sign-in

The first step in this system is to know the port at which we wish to start the server. At this port we will have to bind our server.

Figure 6-2 Server's Port Address

As soon as the system run it'll display an Agent Console window where the user agent can sign-in. It will ask for the name and password of the user-agent. For this it will have to contact the RMI Registry. The RMI Registry will make a reference to a remote object.

```
ServerInterface ref=(ServerInterface)Naming.lookup(remoteRegURL);


        if (ref.login(userName, passWord)){
          String agentName=ref.getAgentName();
          String agentDept=ref.getDept();
```

String agentLoc=ref.getLoc();

String agentCon=ref.getCont();

ref.regUnreg(IP, "online", userName);

String[] onContact=ref.onlineContact(userName)

JOptionPane.showMessageDialog(null, "Welcome You Are Connected!!");
loggedInFrame(userName,agentName, agentDept,agentLoc,agentCon, onContact);}

Figure 6-3 User-Agent Sign-In Window

When the Sign In button is clicked it'll ask for the IP address and the port at which server is running. For this the Remote user is checked against the IP address.

ServerInterface ref = (ServerInterface) Naming.lookup(remoteRegURL);

String IP=JOptionPane.showInputDialog("Enter IP Address");

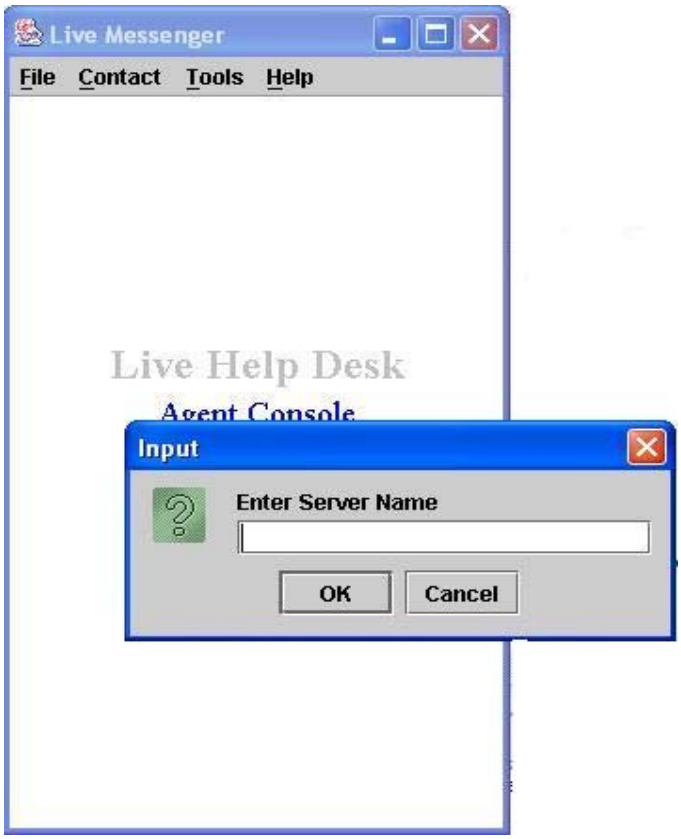


Figure 6-4 Server Name for Sign-in

After that the Remote user is checked against the IP.

if(ref.RemoteUser(IP)){showNewChatWindow(ref.getRemoteUserName(),

ref.getRemoteAgentName(),

ref.getRemoteDept(), ref.getRemoteLoc(), IP);

6.5.2 User Login

Now the user needs to login and is verified through the database. Database is updated as soon as the user logs in.

ref.regUnreg(IP, "online", userName);

String[] onContact=ref.onlineContact(userName);

JOptionPane.showMessageDialog(null, "Welcome You Are Connected!!
loggedInFrame(userName,agentName, agentDept,agentLoc,agentCon, onContact);

Figure 6-5 Log-in Window

If the user is not verified in the database, an error message will be displayed.

JOptionPane.showMessageDialog(null, "Please Enter Username and Password",

"ERROR!!!", JOptionPane.ERROR_MESSAGE);

Figure 6-6 Error Message in Log-in

Once the user has logged in a login window is displayed.

loggedInFrame(userName,agentName, agentDept,agentLoc,agentCon, onContact);

The user-agent picture is also displayed in the Log-in window.

```
LoadPicture(String path) {
        picture = new JLabel(new ImageIcon(path));
        picture.setPreferredSize(new Dimension(70, 100));
        picture.setBorder(BorderFactory.createMatteBorder(
                        2, 1, 2, 2, Color.black));
        picturePanel = new JPanel(new GridBagLayout());
```

picturePanel.setBackground(Color.LIGHT_GRAY);

picturePanel.add(picture); }

The online contact are also displayed in a buddy list.

String[] onContact=ref.onlineContact(userName);

Figure 6-7 User-Agent Logged-in window

6.5.3 Adding Contacts

When the Add Contact button is pressed in the log-in frame a Remote reference is made to the user to view any online contacts that are to be added.

if(ref.RemoteUser(IP)){

showNewChatWindow(ref.getRemoteUserName(), ref.getRemoteAgentName(),

ref.getRemoteDept(), ref.getRemoteLoc(), IP);

If there are no users online an appropriate error message is displayed.

Figure 6-8 No User Online

However if there is some user then the user is contacted by it's IP address.

Figure 6-9 Accessing User by IP address

6.5.4 Chat Windows

Once the user has been found the chat window for that user is displayed on the callers side.

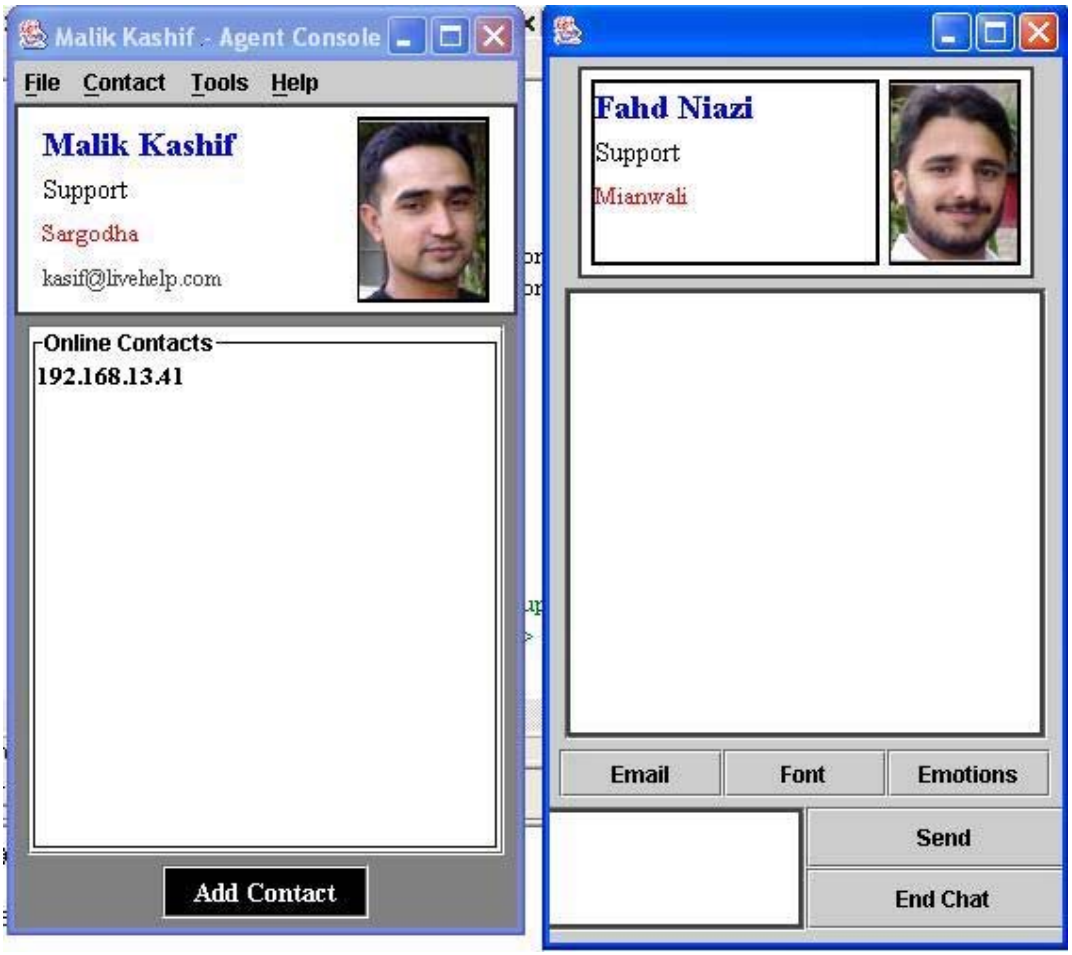


Figure 6-10 User Window on Callers side

Similarly a chat window of the caller will also be displayed on the other side also. This will display the chat window of the callee on the callers side.

Figure 6-11 User Window on Callee's side

After this both the user can have live chat and the messages will be displayed in their chat windows.

Figure 6-12 Messaging

6.6 Client-Agent Interaction

The client can contact the user-agents by an applet provided. The client window contains the clients name as well as the E-mail id where the client will receive complete session description as a mail in his account.

Figure 6-13 Client Window

If there is no user-agent online an appropriate message is displayed to put the client the waiting queue.

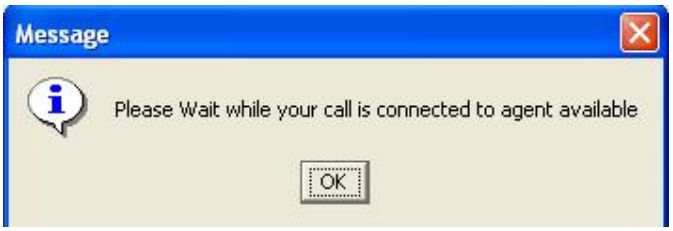


Figure 6-14 No User-Agent Available

6.6.1 Clients Feeback

Once the user has completed the chat with the user-agent he can send a feedback that will be stored in the database of the server.

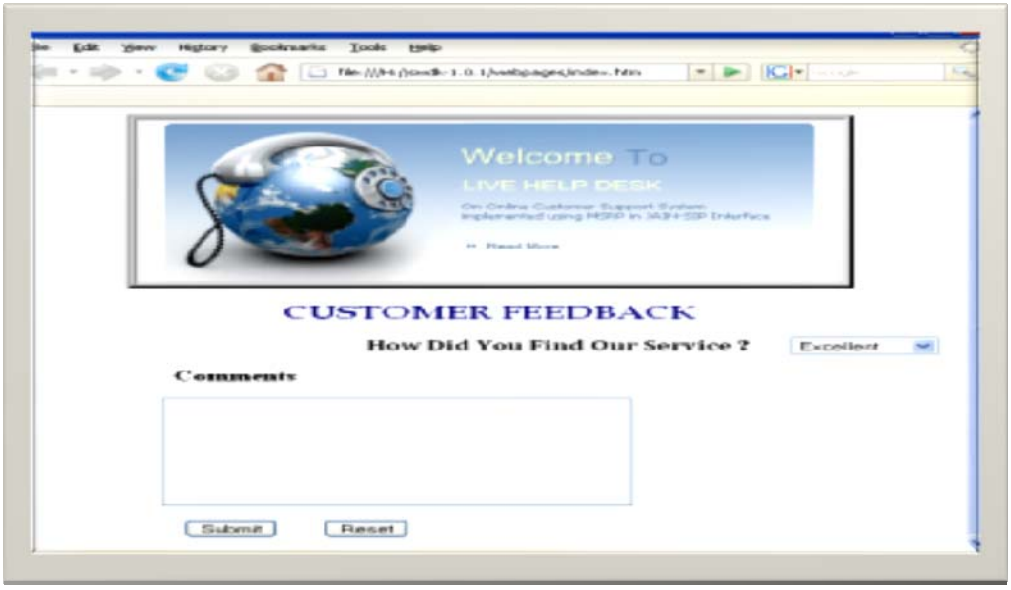


Figure 6-15 Customer FeedBack

6.7 The Database

The database for all interactions that happen between user agents pr those among clients and user agents are stored in different database tables that are updated remotely.

6.7.1 The Agent Data

The database for agent's data is kept in the AgentData table the contains information about agentName, agwntDept, agentLocation, username, password, IPAddress,status and TimeStamping.

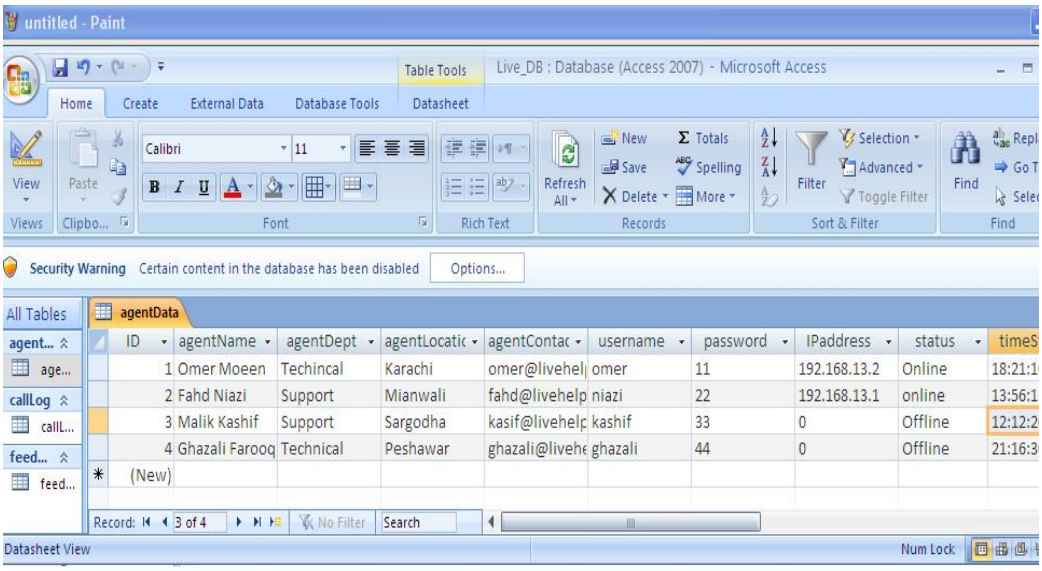


Figure 6-16 User-Agent Database

6.7.2 Call Log

Similarly a log is maintained for all calls in the Call Log table. It contain information like agentName, customerName, customerEmail, startTime, endTime, operatingSystem and complete chat.

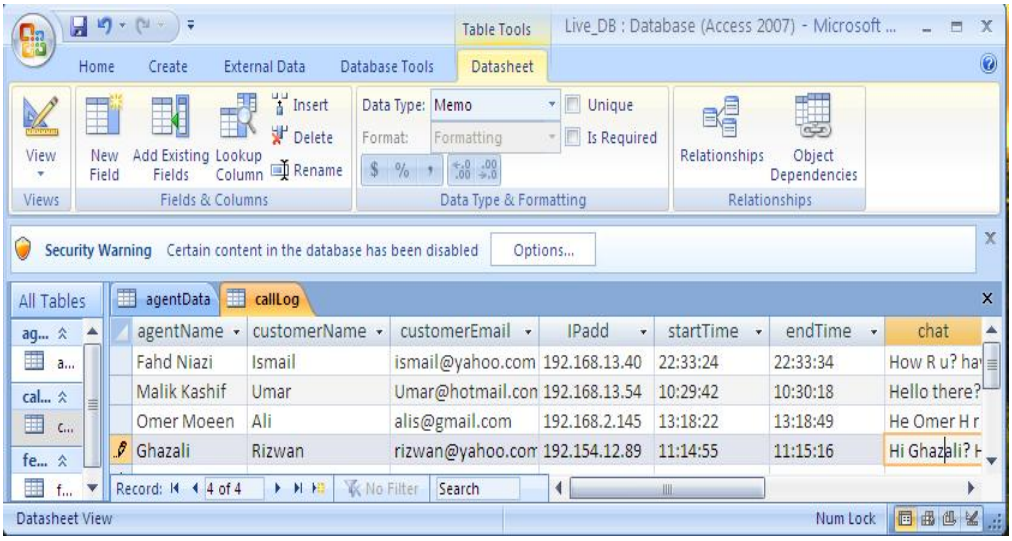


Figure 6-17 Call Log Database

6.7.3 Feed-Back

Similarly the client's feedback is stored in FeedBack table containing customerName, customerEmail, customerIP, rating and comments.

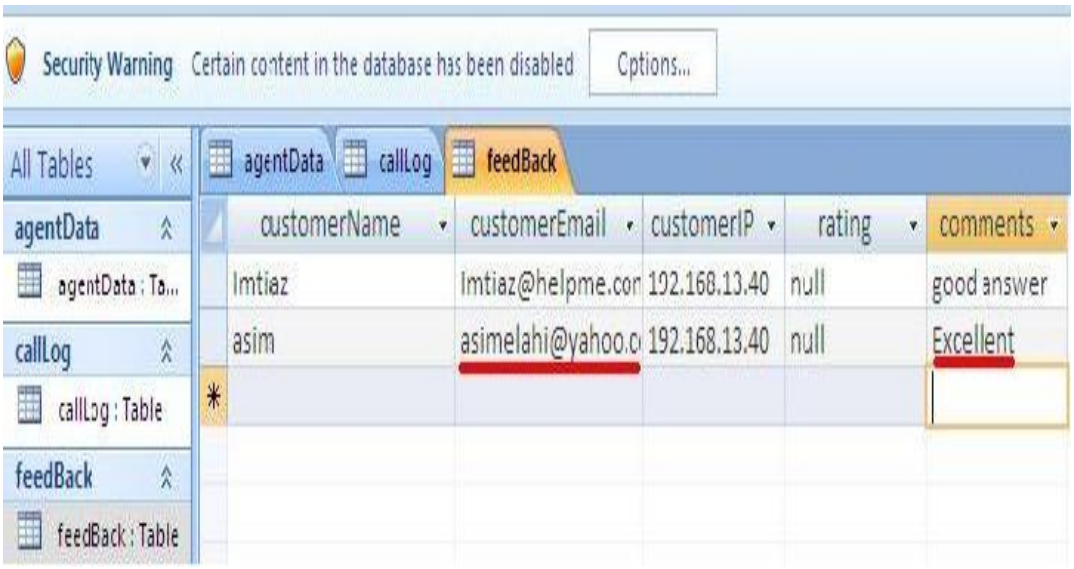


Figure 6-18 Feedback Database

6.8 Report Generation

At the end of session the user-agent can generate reports for service optimization and quality assurance.

For Example, if we have the callLog Table given, we can retrieve any of the values in database against a given parameter. In Figure 6-18 we have used the AGENT NAME as a parameter value.

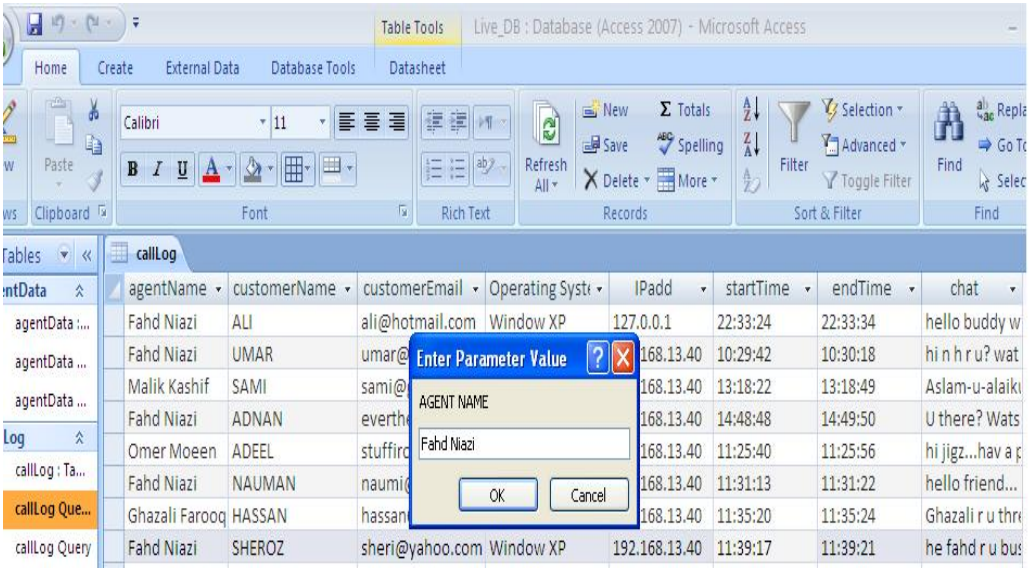


Figure 6-19 Entering Parameter Values for Retrieval

Entering the Ok Button will return another table containing records against the given parameter value. The number of records as well as whole information about the Agent Name will be shown in the table. Figure 6-19 illustrates this in a table.

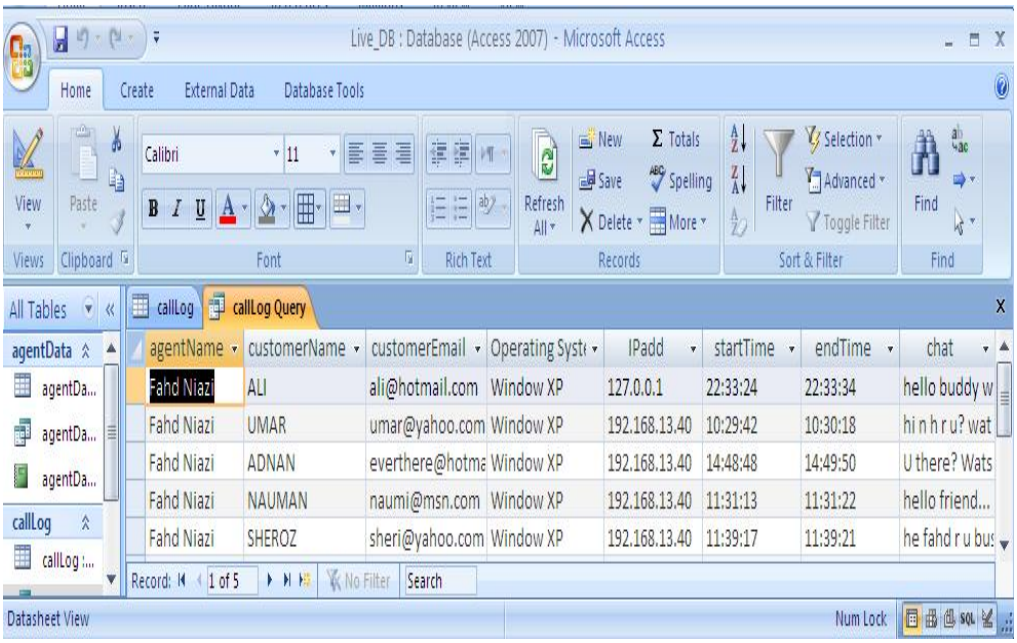
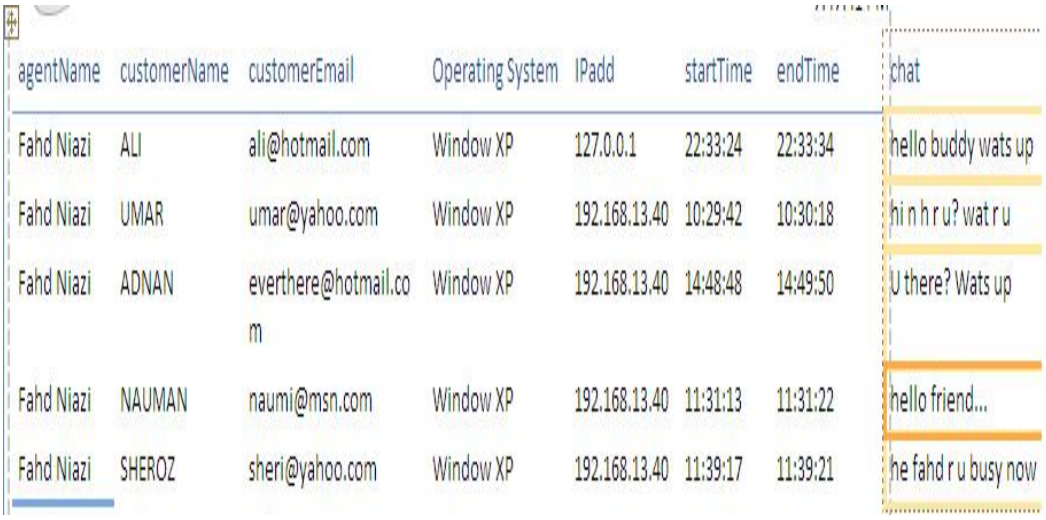


Figure 6-20 Records Retrieved for Agent Name

In the end an exclusive report is generated which can be used for over viewing of previous chat histories.



agentName	customerName	customerEmail	Operating System	IPadd	startTime	endTime	chat
Fahd Niazi	ALI	ali@hotmail.com	Window XP	127.0.0.1	22:33:24	22:33:34	hello buddy wats up
Fahd Niazi	UMAR	umar@yahoo.com	Window XP	192.168.13.40	10:29:42	10:30:18	hi n h r u? wat r u
Fahd Niazi	ADNAN	everthere@hotmail.com	Window XP	192.168.13.40	14:48:48	14:49:50	U there? Wats up
Fahd Niazi	NAUMAN	naumi@msn.com	Window XP	192.168.13.40	11:31:13	11:31:22	hello friend...
Fahd Niazi	SHEROZ	sheri@yahoo.com	Window XP	192.168.13.40	11:39:17	11:39:21	he fahd r u busy now

Figure 6-21 Report Generation

Analysis and Conclusion

7.1 Analysis

We studied the drivers of customer satisfaction for Online marketing, Sales and Customer Support application systems. Based on the existing literature on business value of information technology, we provided a framework to understand the drivers of overall customer satisfaction with Internet solutions. Our analysis is based on data collected from a survey of existing customer support systems that use Internet marketing applications. The overall goal is to provide customers such features that lower total cost of ownership and provide competitive benefits as critical drivers of overall customer satisfaction with the systems.

7.2 Conclusion

In this research we have reported an empirical study that identifies the drivers of customer satisfaction for internet marketing application systems. Our results provide insights for Internet solution vendors in their product development and design decisions. Our findings suggest that vendors should attempt to focus on the ‘whole product' by augmenting their core product with features and services that lower the total cost of ownership rather than just the initial purchase value of Internet systems .

A notable limitation in our analysis is that we have addressed only the perceived benefits of Internet solutions. A further challenge is to assess the business value from Internet and other electronic commerce solutions quantitatively. Senior management in organizations should be convinced that an Internet application is an empowering technology and may succeed only if all the users are allowed to access and share the appropriate information across corporate networks. However this openness to

information sharing should be supported with security controls and access permissions for confidential data. In addition, management may need to ensure that the Internets are being used by the employees for the right applications that will add value to any firm.

7.3 Future Work

The project can further be enhanced to address the quantifiable business value from Internet systems in future research. A successful integration of the Internet into the business processes and culture of an organization will depend on establishing a broad level of ownership and commitment. The evolution and direction of Internet applications should not remain a responsibility of the information technology group within organizations but involve a broader forum including senior executives from multiple disciplines. In addition, this evolution should also identify the quantitative metrics for assessing the business value from these systems at the appropriate level of process changes by these applications.

In summary, answers to questions such as how to keep track of information or what kind of management rules to impose for use of Customer Support systems are important and should be addressed in future research.

```
import java.awt.*;

import java.awt.Dimension;

import java.awt.event.*;

import javax.swing.*;

import javax.swing.event.*;

import java.util.*;

import java.text.SimpleDateFormat;

import java.rmi.*;

//jAVA mail API

import javax.mail.*;

import javax.mail.internet.*;
```

//The Server Interface

```
public interface ServerInterface extends Remote{

 public boolean login(String u_name, String Passwrd) throws RemoteException;

 public void regUnreg(String IP, String status, String user) throws RemoteException;

 public void timeSetting(String IP, String time) throws RemoteException;

        public String[] onlineContact(String user) throws RemoteException;

 public boolean agentSearch(String dept)throws RemoteException;

 public void callLog(String agent,String customer, String email, String ip,String
st,String et, String chat)throws RemoteException;

 public boolean searchLog()throws RemoteException;

 //Local User Set methods

 public void setUserName(String name)throws RemoteException;

 public void setagentName(String a_name)throws RemoteException;

 public void setDept(String a_dept)throws RemoteException;
```

```
 public void setLoc(String a_loc)throws RemoteException;

 public void setCont(String a_con)throws RemoteException;

 public void setIP(String ip)throws RemoteException;

 public void setStatus(String user,String state)throws RemoteException;

 public void feedBack(String cust, String em, String ip,String r,String c) throws
RemoteException;

 //Customer setMethods

 public void setCustomer(String cn)throws RemoteException;

 public void setCustomerEmail(String ce)throws RemoteException;

 public void setCustomerIP(String cip)throws RemoteException;

 //customer getMethods

 public String getCustomer()throws RemoteException;

 public String getCustomerEmail()throws RemoteException;

 public String getCustomerIP()throws RemoteException;

//Local User get methods

 public String getUserName()throws RemoteException;

 public String getAgentName()throws RemoteException;

 public String getDept()throws RemoteException;

 public String getLoc()throws RemoteException;

 public String getCont()throws RemoteException;

 public String getIP()throws RemoteException;

 public String getStatus()throws RemoteException;

 //Remote User set methods

 public void setRemoteUserName(String name)throws RemoteException;

 public void setRemoteAgentName(String name)throws RemoteException;

 public void setRemoteDept(String name)throws RemoteException;
```

```
public void setRemoteLoc(String name)throws RemoteException;

public void setRemoteCon(String name)throws RemoteException;

public boolean RemoteUser(String IP)throws RemoteException;

public String getRemoteUserName()throws RemoteException;

public String getRemoteAgentName()throws RemoteException;

public String getRemoteDept()throws RemoteException;

public String getRemoteLoc()throws RemoteException;

public String getRemoteCon()throws RemoteException;}//end interface
```

//The Server Implementation

public class ServerImpl extends UnicastRemoteObject implements ServerInterface{

```
private Connection con = null;

private Statement st = null;

final String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";

public String username, password, agentname,location,dept,contact,status,IP,
remoteUser, remoteAgentName,remoteDept, remoteLoc, remoteCon;

public String customer,customerEmail, customerIP;

public ServerImpl() throws RemoteException{

                    try{

Class.forName(DRIVER);

con = DriverManager.getConnection("jdbc:odbc:Live_DB","","");

st = con.createStatement();}

                    catch(SQLException e)

                      {e.printStackTrace();}

                    catch(ClassNotFoundException ce)

                      {ce.printStackTrace();}
```

```
                    try {  }//end constructor

public boolean login(String u_name, String pswd) throws RemoteException{

    {     boolean found=false;

          username=u_name;

          password=pswd;

                try {

 st = con.createStatement();

 String namequery = "SELECT * FROM agentData";

 namequery += " WHERE username = '" + username + "'AND password=
'"+password+"'";

 ResultSet resultset = st.executeQuery(namequery);

 if(resultset.next()) {


if((username.equals(resultset.getString("username")))&&(password.equals(resultset.g
etString("password"))))

                    {

                      setUserName(username);

                      setagentName(resultset.getString("agentName"));

                      setLoc(resultset.getString("agentLocation"));

                      setDept(resultset.getString("agentDept"));

                      setCont(resultset.getString("agentContact"));

                      found = true;

                      }

                     else{}

                    }//end outer if

else   {
```

```
  JOptionPane.showMessageDialog(null, "Invalid username or Password", "Invalid
Password",JOptionPane.ERROR_MESSAGE);

                found=false;} }//end try

                catch(SQLException e){ e.printStackTrace();}

                 catch(RemoteException e){ e.printStackTrace();}

                 return found; }//end login  }

public void regUnreg(String IPadd, String state, String user) throws
RemoteException{

        IP=IPadd;

        status=state;

        username=user;

         try{

                st=con.createStatement();

                String query="UPDATE agentData SET IPaddress='"+IP+"',
status='"+status+"' WHERE username='"+username+"';";

                st.executeUpdate(query); }//end try

        catch(SQLException e) {     }

        catch(Exception e)  {   } }//end reg

 public String[] onlineContact(String user) throws RemoteException{

           String username=user;

           String[] online=new String[4];

           String[] papu={"No Contact Online"};

          try{  st=con.createStatement();

String query="SELECT IPaddress FROM agentData WHERE status='online'  AND
NOT (username ='"+username+"');";

ResultSet rs=st.executeQuery(query);

        int j = 0;
```

```
        while (rs.next()) {
        online[j] = rs.getString("IPaddress");
        System.out.println(online[j]);
        j++;   }    }//end try
        catch(SQLException e){   e.printStackTrace();}//end catch
        return  online;  }//end onlineContact
 public boolean agentSearch(String department)throws RemoteException{
        boolean available=false;
        String dept=department;
    try
   {       st = con.createStatement();
String query = "SELECT * FROM agentData WHERE agentDept = "'+dept+'" AND
status= 'online' ORDER BY timeStmp";
ResultSet resultset = st.executeQuery(query);
                if(resultset.next()){
                    setUserName(resultset.getString("username"));
                    setagentName(resultset.getString("agentName"));
                    setLoc(resultset.getString("agentLocation"));
                    setDept(resultset.getString("agentDept"));
                    setCont(resultset.getString("agentContact"));
                    setIP(resultset.getString("IPaddress"));
                    available = true;
                    }
                else{available=false;}
            }//end try
             catch(SQLException e){ e.printStackTrace();}
```

```
            catch(RemoteException e){e.printStackTrace();

            }

          return available;}

 public boolean RemoteUser(String ip)throws RemoteException{

boolean exits=false;

String IP=ip;

          try{

             st = con.createStatement();

String query ="SELECT * FROM agentData WHERE status= 'online' AND
IPaddress='"+IP+ "';";

ResultSet resultset = st.executeQuery(query);

        if(resultset.next()){

           setRemoteUserName(resultset.getString("username"));

           setRemoteAgentName(resultset.getString("agentName"));

           setRemoteLoc(resultset.getString("agentLocation"));

           setRemoteDept(resultset.getString("agentDept"));

           setRemoteCon(resultset.getString("agentContact"));

           exits=true; }

       else{

       JOptionPane.showMessageDialog(null,"No User Online");

       exits=false;      }  }

  catch(Exception e){  }

return exits; }//end remoteUser


public void setStatus(String ipadd,String state)throws RemoteException{

status=state;
```

```
String IPad=ipadd;

try{ st=con.createStatement();

String query="UPDATE agentData SET status="'+state+"' WHERE
IPaddress="'+IPad+"';";

        st.executeUpdate(query);}//end try

        catch(SQLException e){ e.printStackTrace();}

        catch(Exception e){ e.printStackTrace();}

 }

public void timeSetting(String ip, String t) throws RemoteException{

String IPadd=ip;

String time=t;

            try{

                st=con.createStatement();

String query="UPDATE agentData SET timeStmp="'+time+"' WHERE
IPaddress="'+IPadd+"';";

           st.executeUpdate(query);}//end try

     catch(SQLException e)

     {

            System.out.print("\nSQL exception  fail to set timeStamp\n");

            e.printStackTrace();

     }

     catch(Exception e) {System.out.print("\n fail to set timeStamp");}}

public void callLog(String agent,String cust, String em, String ip,String stime,String
et, String chat)throws RemoteException{

    String Agent=agent;

    String customer=cust;

    String email=em;
```

```
    String IPadd=ip;
    String startTime=stime;
    String endTime=et;
    String chatLog=chat;
   try{
                st=con.createStatement();
String query="INSERT INTO callLog VALUES
'"+Agent+"','"+customer+"','"+email+"','"+IPadd+"','"+startTime+"','"+endTime+"','"+
chatLog+"');";
                st.executeUpdate(query); }//end try
          catch(SQLException e) { e.printStackTrace(); }
          catch(Exception e) {e.printStackTrace();}}
public boolean searchLog()throws RemoteException{
boolean found=false;
    try{
                st = con.createStatement();
                String query = "SELECT * FROM callLog ORDER BY 'endTime'
DESC";
                ResultSet resultset = st.executeQuery(query);
                if(resultset.next()) {
                     setCustomer(resultset.getString("customerName"));
                     setCustomerEmail(resultset.getString("customerEmail"));
                     setCustomerIP(resultset.getString("IPadd"));
                     found = true; }
                else{found=false;}}//end try
            catch(SQLException e) e.printStackTrace();
            catch(RemoteException e) { e.printStackTrace();
```

```
    return found;
  }
 public void feedBack(String cust, String em, String ip,String r,String c)throws
RemoteException
  {
    String customer=cust;
    String email=em;
    String IPadd=ip;
    String rating=r;
    String comments=c;
   try{
                st=con.createStatement();
                String query="INSERT INTO feedBack VALUES
('"+customer+"','"+email+"','"+IPadd+"','"+rating+"','"+comments+"');";
                st.executeUpdate(query); }//end try
          catch(SQLException e){e.printStackTrace();}
          catch(Exception e) { e.printStackTrace();}//set methods for Local User
 public void setUserName(String name)throws RemoteException{
    username=name;}
public void setagentName(String a_name)throws RemoteException{
    agentname=a_name;}
public void setDept(String a_dept)throws RemoteException{
     dept=a_dept;}
public void setLoc(String a_loc)throws RemoteException{
    location=a_loc;}
public void setCont(String a_con)throws RemoteException{
```

```
    contact=a_con;
public void setIP(String ip)throws RemoteException{
    IP=ip; //get methods for LocalUser
public String getUserName()throws RemoteException{
    return username;}
public String getAgentName()throws RemoteException{
    return agentname;}
 public String getDept()throws RemoteException{
    return dept; }
public String getLoc()throws RemoteException{
   return location;}
public String getCont()throws RemoteException{
  return contact;}
public String getIP()throws RemoteException{
   return IP;}
public String getStatus()throws RemoteException{
    return status;}
 //set Methods for customer
 public void setCustomer(String cn)throws RemoteException{
   customer=cn;  }
public void setCustomerEmail(String ce)throws RemoteException{
   customerEmail=ce;
public void setCustomerIP(String cip)throws RemoteException{
   customerIP=cip;}
  //get Methods for Customer
```

```
public String getCustomer()throws RemoteException{
          return customer;}
public String getCustomerEmail()throws RemoteException{
     return customerEmail;}
public String getCustomerIP()throws RemoteException{
     return customerIP;}
  //set method for remote user
public void setRemoteUserName(String u)throws RemoteException{
   remoteUser=u;}
public void setRemoteAgentName(String name)throws RemoteException{
   remoteAgentName=name;}
public void setRemoteDept(String d)throws RemoteException{
      remoteDept=d; }
public void setRemoteLoc(String l)throws RemoteException{
         remoteLoc=l;}
 public void setRemoteCon(String c)throws RemoteException{
           remoteCon=c;}
  //get methods for remoteUser
public String getRemoteUserName()throws RemoteException{
          return remoteUser;}
 public String getRemoteAgentName()throws RemoteException{
          return remoteAgentName;}
public String getRemoteDept()throws RemoteException{
         return remoteDept;}
public String getRemoteLoc()throws RemoteException{return remoteLoc;}
```

```
public String getRemoteCon()throws RemoteException{return remoteCon;}

}//end class
```

//The Server

```
public class Server {

public Server() {

public static void main(String args[]) throws Exception{

ServerImpl liveObject = new ServerImpl();

int hostPort = Integer.parseInt(javax.swing.JOptionPane.showInputDialog(null,"Enter
the Port Where you wish to start the server"));

String hostName = InetAddress.getLocalHost().getHostName();

String registryURL = "rmi://" + hostName + ":" + hostPort + "/server";

LocateRegistry.createRegistry(hostPort);

 Naming.bind(registryURL,liveObject);

 System.out.println("Server is Online at " + hostName + ":" + hostPort);}

 private void jbInit() throws Exception {

  }}
```

//TCP Channel

```
public class TCPChannel {

private Socket outSocket;

private ObjectOutputStream outgoingChannel;

  int port;

  int tooPort=6060;

  private String myAddress;

  private InetAddress peerAddress;

  private int peerPort;

  public static String expMessage = null;
```

```
 public TCPChannel() {

public void sendRequest(String message, String IP) {

System.out.print("###############Start of Message###############");

String toAddress = IP;

System.out.println("\n\nRemote Address : " + toAddress+ " port is : " +tooPort);

System.out.println(message.toString());

   try {

  SocketAddress sockAddr = new
InetSocketAddress(InetAddress.getByName(toAddress),

        tooPort); //new InetSocketAddress(Address, port);

 System.out.print("Sending Message at\t" + sockAddr.toString());

 outSocket = new Socket();

 outSocket.connect(sockAddr);

 outgoingChannel = new ObjectOutputStream(outSocket.getOutputStream());

 //String strMsg = null;

  //strMsg = new String(message.toString());

  outgoingChannel.writeObject(message);

  outgoingChannel.flush();

 outgoingChannel.close();

 outSocket.close();

System.out.println("\n\tMessage Sending Successful");

   }

     catch (Exception exp) {

     System.out.println(exp.getMessage());

     expMessage = exp.getMessage();

    // return false;}//return true;}
```

```
 public void sendWindow(String message, String IP, String MyIP) {

String fromAddress=MyIP;

System.out.print("##############Start of Message##############");

String toAddress = IP;

//listener.showWin(fromAddress);

System.out.println("\n\nFrom Address : " + fromAddress+ " port is : " +tooPort);

System.out.println(message.toString());

try {

SocketAddress sockAddr = new
InetSocketAddress(InetAddress.getByName(toAddress),tooPort); //new
InetSocketAddress(Address, port);

System.out.print("Sending Message at\t" + sockAddr.toString());

outSocket = new Socket();

outSocket.connect(sockAddr);

outgoingChannel = new ObjectOutputStream(outSocket.getOutputStream());

//String strMsg = null;

 /strMsg = new String(message.toString());

outgoingChannel.writeObject(message);

outgoingChannel.flush();

outgoingChannel.close();

outSocket.close();

System.out.println("\n\tMessage Sending Successful");

  }

  catch (Exception exp) {

    System.out.println(exp.getMessage());

    expMessage = exp.getMessage();// return false; }
```

```
  //return true; }
```

//TCP Listener

```
public class TCPListener implements Runnable {

public TCPListener() {

 private ServerSocket listenSocket = null;

 private Socket clientSocket = null;

 private ObjectInputStream incommingChannel = null;

TCPChannel tcpChannel = null;

AgentConsole agentConsole=null;

 static boolean isActive = false;

String LocalAgent,localIP,remoteIP,remoteRegURL, remoteUser, remoteAgent,
remoteDept, remoteLoc;

public TCPListener(String registry,String ip) {

setRegistry("rmi://niazi:1111/server");

setLocalIP(ip);

tcpChannel = new TCPChannel();}

public TCPListener(String ip) {


  setRemoteIP(ip);}

 public void listen() {

   try {

     InetAddress addr = InetAddress.getLocalHost();

     listenSocket = new ServerSocket(6060, -1, addr);

     System.out.println("Starting TCP Listener at: " +

    listenSocket.getLocalSocketAddress().toString());
```

```
//Read the Messages
      while (true) {
        //Accept Connection
        clientSocket = listenSocket.accept();
        //Get Inputstream
        incommingChannel = new ObjectInputStream(clientSocket.getInputStream());
        //Loop for reading the contents
        //int count = incommingChannel.available();
        //System.out.println(count);
        // byte[] buffer = new byte[count + 1];
        //incommingChannel.read(buffer);
        //String msg = new String(buffer, "utf-8");
        // buffer = null;
        Object msg = incommingChannel.readObject();
        String comp=msg.toString().substring(0,6);
        System.out.print(comp);
        System.out.println("\nIP : "+msg.toString().substring(7,19));
       if (msg.toString().substring(0,6).matches("INVITE")) {
         System.out.println(msg.toString().substring(7,19));
         setRemoteIP(msg.toString().substring(7,19).trim());
             System.out.println("\nReceiced Invite Request form "+getRemoteIP());
             System.out.print("\n Registry : "+remoteRegURL);
             showWin();
             agentConsole.chatWindow = new
ChatWindow(getRemoteUser(),getRemoteAgent(),getRemoteDept(),
getRemoteLoc(), getRemoteIP(),getRemoteReg());
```

```
            agentConsole.chatWindow.init();
            System.out.print("\n OKKK");
            agentConsole.chatWindow.setParent(agentConsole);
            agentConsole.chatWindow.setSendWindow(false);
          }
          processRequest(msg.toString());

      //end outerif
     } //end while
   }
   catch (Exception exp) {
    exp.getMessage();}}
public void showWin(){
   try {
   ServerInterface ref = (ServerInterface) Naming.lookup(getRemoteReg());
   ref.RemoteUser(getRemoteIP());
   setRemoteUser(ref.getRemoteUserName());
   setRemoteAgent(ref.getRemoteAgentName());
   setRemoteDept(ref.getRemoteDept());
   setRemoteLoc(ref.getRemoteLoc()); }
   catch(Exception e){   } }
  public void processRequest(String request) {
 if (AgentConsole.chatWindow != null &&
AgentConsole.chatWindow.isSendWindow()) {
    AgentConsole.chatWindow.messageArea.append("Local(remote/Kashif) ; " +
request.toString()+"\n");}
```

```
   else if (AgentConsole.chatWindow != null &&

           !AgentConsole.chatWindow.isSendWindow()) {

           AgentConsole.chatWindow.messageArea.append(getRemoteUser()+" says : "
+request.toString()+"\n");    }  }

 public void setRegistry(String reg){

   remoteRegURL=reg; }

 public void setRemoteUser(String u){

   remoteUser=u; }

  public void setRemoteAgent(String a){

   remoteAgent=a;}

  public void setRemoteDept(String d){

     remoteAgent=d;}

   public void setRemoteLoc(String l){

       remoteLoc=l;  }

   public void setLocalIP(String ip){

    localIP=ip;}

  public void setRemoteIP(String ip){

       remoteIP=ip;  }

 public String getRemoteReg(){

   return remoteRegURL; }

 public String getLocalIP(){

       return localIP; }

     public String getRemoteIP(){

      return remoteIP;}

 public String getRemoteUser(){

   Font f=new Font("Serif", Font.BOLD, 14);
```

```
  return remoteUser;}
 public String getRemoteAgent(){
   return remoteAgent;}
 public String getRemoteDept(){
    return remoteDept; }
  public String getRemoteLoc(){
     return remoteLoc; }
 public void run() { listen(); }
public class ClientApplet extends JApplet implements WindowListener, Runnable{
public ClientApplet() {
  JFrame loginWindow;
 JTextField nameField, emailField;
 JRadioButton support, technical;
 JLabel namelbl, emaillbl, picture;
 JPanel headerPanel,inputPanel,radioPanel,questionPanel, picturePanel, buttonPanel;
 JTextArea questionArea;
 ButtonGroup radioGroup;
 JButton chat;
 private static String picPath, name, email, question;
 String RemoteRegistryURL;
 boolean sup=true, tech=false;
 public static ChatWindow chatWindow = null;
 TCPListener tcpListener;
 public void init() {
 loginWindow=new JFrame("Live Help Desk");
 loginWindow.getContentPane().setLayout(new FlowLayout());
```

```
 headerPanel=new JPanel();
 headerPanel.setLayout(new FlowLayout());
 headerPanel.setBorder(BorderFactory.createMatteBorder(
                              2, 2, 2, 2, Color.DARK_GRAY));
picPath = getCurrentDirectory() + "/images/header.gif";
 LoadPicture(picPath);
 headerPanel.add(picturePanel);
 nameField=new JTextField(20);
 emailField=new JTextField(20);
 inputPanel = new JPanel();
 inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y_AXIS));
 inputPanel.add(new JLabel("Name"));
 inputPanel.add(nameField);
inputPanel.add(new JLabel("E-mail"));
 inputPanel.add(emailField);
 radioPanel=new JPanel();
 radioPanel.setLayout(new BoxLayout(radioPanel, BoxLayout.Y_AXIS));
 support=new JRadioButton("Customer Support", true);
 technical=new JRadioButton("Technical Support", false);
 radioGroup=new ButtonGroup();
 radioGroup.add(support);
 radioGroup.add(technical);
 support.addItemListener(new ItemListener(){
   public void itemStateChanged(ItemEvent e){
     if(e.getSource()==support)
```

```
        sup=true;   }  }  );
 technical.addItemListener(new ItemListener(){
   public void itemStateChanged(ItemEvent e) {
     if (e.getSource() == technical)
       tech = true; }  }  );
radioPanel.add(support);
 radioPanel.add(technical);
questionPanel=new JPanel();
 questionPanel.setLayout(new GridLayout(2,1));
JLabel ques=new JLabel(" What is your question");
 questionArea=new JTextArea(5,35);
 questionArea.setBorder(BorderFactory.createMatteBorder(
                          2, 2, 2, 2, Color.DARK_GRAY));
questionPanel.add(ques);
 questionPanel.add(questionArea);
 buttonPanel=new JPanel();
 buttonPanel.setBackground(Color.WHITE);
 buttonPanel.setLayout(new BorderLayout());
 chat=new JButton("Start Chat");
 chat.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent e) {
  setname(nameField.getText());
  setemail(emailField.getText());
  setquestion(questionArea.getText());
  loginWindow.setVisible(false);
```

```
   run(); } } );
  buttonPanel.add(chat, BorderLayout.CENTER);
   loginWindow.getContentPane().add(headerPanel);
   loginWindow.getContentPane().add(inputPanel);
   loginWindow.getContentPane().add(radioPanel);
   loginWindow.getContentPane().add(questionPanel);
   loginWindow.getContentPane().add(buttonPanel);
  loginWindow.pack();
   loginWindow.setSize(275,495);
   loginWindow.setVisible(true);
 }//end init
 public void start(){  initTCP();  }
  public void LoadPicture(String path) {
      picture = new JLabel(new ImageIcon(path));
      picture.setPreferredSize(new Dimension(250, 100));
      picture.setBorder(BorderFactory.createMatteBorder(
                              2, 1, 2, 2, Color.black));
       picturePanel = new JPanel(new GridBagLayout());
       picturePanel.setBackground(Color.LIGHT_GRAY);
       picturePanel.add(picture);  }
 public void showNewChatWindow(String user, String agent, String department,
String location, String IPadd) {
   String username=user;
   String agentname=agent;
   String dept=department;
   String loc=location;
```

```
   String ip=IPadd;

  chatWindow = new ChatWindow(username,
agentname,dept,loc,ip,getRemoteRegistryURL(),getname(),getemail(),getquestion());

   chatWindow.init();

   chatWindow.setSendWindow(true);}

    public String getLocalSystemIP() {

        String ip = "";

        try {

          ip = Inet4Address.getLocalHost().getHostAddress(); }

        catch (Exception exp) {

          System.out.println(exp.toString()); }

        return ip;   }

  public void initTCP() {

      //Set Master/Slave Here.

      tcpListener = new TCPListener(getRemoteRegistryURL(),getLocalSystemIP());

      Thread tcpListenerThread = new Thread(tcpListener);

      tcpListenerThread.start();     }

  public String getCurrentDirectory() {

    return System.getProperty("user.dir");  }

  public void run(){

    if(sup)   {

     try {

       setRemoteRegistryURL("rmi://mslab20:1111/server");

ServerInterface ref = (ServerInterface) Naming.lookup(getRemoteRegistryURL());

       try {

         while(!ref.agentSearch("Support")){
```

```
            Thread.sleep(6000);

JOptionPane.showMessageDialog(null,"Please Wait while your call is connected to
agent available ");  }

            System.out.println("SUPOORT DEPT");

             String IP=ref.getIP();

              String user=ref.getUserName();

              String name=ref.getAgentName();

              String dept=ref.getDept();

              String loc=ref.getLoc();

              ref.setStatus(IP,"Busy");

              System.out.print("\nSetting Status "+IP+" Busy");

              System.out.println(name);

              showNewChatWindow(user,name,dept,loc,IP);

          }//end iiner try

        catch (Exception oe) { oe.printStackTrace(); }    } //end outer try

      catch (Exception le) {    le.printStackTrace();   }

 }//end if

    else if(tech)

    {

      try {

        setRemoteRegistryURL("rmi://mslab20:1111/server");

        ServerInterface ref = (ServerInterface)
Naming.lookup(getRemoteRegistryURL());

       try {

         while(!ref.agentSearch("Technical")){

JOptionPane.showMessageDialog(null,"Please Wait while your call is connected to
agent available ");
```

```
        }
            System.out.println("TECHNICAL DEPT");
            String IP=ref.getIP();
            String user=ref.getUserName();
            String name=ref.getAgentName();
            String dept=ref.getDept();
            String loc=ref.getLoc();
            ref.setStatus(name,"Busy");
            showNewChatWindow(user,name,dept,loc,IP);  } //end inner try
       catch (Exception oe) {
         oe.printStackTrace();
       }
     } //end outer try
     catch (Exception le) {
       le.printStackTrace(); }/end elseif
}//end run
private void setname(String n){
  name=n;  }
private void setemail(String e){
  email=e;}
private void setquestion(String q){
  question=q;}
public void setRemoteRegistryURL(String r){
  RemoteRegistryURL=r;}
public String getRemoteRegistryURL(){
```

```
  return RemoteRegistryURL;}
 public String getname(){
   return name;}
  public String getemail(){   return email;}
  public String getquestion(){  return question;}
 public void windowClosing(WindowEvent e) {
   System.out.println("Closed"); }
  public void windowClosed(WindowEvent e) {
   System.out.println("Closed");
  }
  public void windowOpened(WindowEvent e) {   }
  public void windowIconified(WindowEvent e) {    }
  public void windowDeiconified(WindowEvent e) {    }
  public void windowActivated(WindowEvent e) {    }
public void windowDeactivated(WindowEvent e) {  System.out.println("Main
Window Deactivated"); }
```

//Feed Back Servlet

```
public class FeedBackServlet extends HttpServlet{
  String customer,email,IP;
public void doPost(HttpServletRequest req, HttpServletResponse res) throws
IOException, ServletException
        {
              res.setContentType("text/html");
            PrintWriter out = res.getWriter();
```

```
        String rating = req.getParameter("rating");
              String comments = req.getParameter("comments");
      try
      {
     String RemoteRegistryURL = "rmi://mslab20:1111/server";
     ServerInterface ref = (ServerInterface)Naming.lookup(RemoteRegistryURL);
       try        {
         if(ref.searchLog()){
           customer=ref.getCustomer();
           email=ref.getCustomerEmail();
           IP=ref.getCustomerIP();     }
         ref.feedBack(customer,email,IP,rating,comments);
         out.println("<html><head><title>Thank You</title></head>");
out.println("<body><center><br><br><br><br><br><font color=blue
size=6>Thank <font color=red size=6><b>"+customer +"</b></font> You For Your
Response</font></center> ");
         out.println("</body></html>");
              }//end inner try
            catch(Exception oe){
                  PrintWriter p = new PrintWriter(res.getWriter());
                    oe.printStackTrace(p);  }/end outer try
                    catch(Exception le){
                    PrintWriter p = new PrintWriter(res.getWriter());
                          le.printStackTrace(p); }
      out.close();}//end method
}//end class
```

//Chat Window

public class ChatWindow

extends JFrame implements KeyListener, WindowListener {

public ChatWindow() {

public JTextArea inputArea, messageArea = null;

JButton sendButton,endChat,fontButton,emotionsButton,emailButton;

TCPChannel tcpChannel = null;

TCPListener listen;

String body;

boolean isSendWindow = true;

boolean isRecvWindow=true;

boolean isClientWindow=false;

String title, IP,user, agent, dept,location, customer,email, question, picPath,remoteRegURL;

String startTime,endTime;

AgentConsole parent = null;

String recipients[];

JPanel picturePanel;

//Over Loaded Constructor

public ChatWindow(String username, String agent, String dept, String loc, String ip,String reg) {

tcpChannel = new TCPChannel();

setUsername(username);

setAgent(agent);

setDept(dept);

setLoc(loc);

```
setIP(ip);

setRemoteReg(reg);

Date dateTime = new Date();

SimpleDateFormat format = new SimpleDateFormat("H:mm:ss", Locale.ENGLISH);

setStartTime(format.format(dateTime).toString());

recipients=new String[2];

recipients[0]="omermoeen@mcs.edu.pk"; }

 //Over Loaded Constructor for client

public ChatWindow(String username, String agent, String dept, String loc, String
ip,String reg, String cust,String email, String q) {

tcpChannel = new TCPChannel();

setUsername(username);

setAgent(agent);

setDept(dept);

setLoc(loc);

setIP(ip);

setRemoteReg(reg);

setCustomer(cust);

setEmail(email);

setQuestion(q);

setClientWindow(true);

Date dateTime = new Date();

SimpleDateFormat format = new SimpleDateFormat("H:mm:ss", Locale.ENGLISH);

setStartTime(format.format(dateTime).toString());

recipients=new String[2];

recipients[0]="omermoeen@mcs.edu.pk";}
```

```
public boolean isSendWindow() {
return isSendWindow; }
public void setSendWindow(boolean flag) {
this.isSendWindow = flag;}
public boolean isRecvWindow() {
return isRecvWindow; }
public void setRecvWindow(boolean flag) {
this.isRecvWindow = flag;}
public boolean isClientWindow() {
return isClientWindow; }
public void setClientWindow(boolean flag) {
this.isClientWindow = flag; }
public void init() {
this.getContentPane().setLayout(new FlowLayout());
this.setLocation(new Point(200, 80));
JPanel labelPanel=new JPanel();
labelPanel.setBackground(Color.WHITE);
labelPanel.setLayout(new GridLayout(4,1));
Label agentlbl=new JLabel(getAgent()+"            ");
agentlbl.setForeground(Color.BLUE);
agentlbl.setFont(new Font("Serif", Font.BOLD, 18));
 JLabel deptlbl=new JLabel(getDept());
deptlbl.setForeground(Color.BLACK);
deptlbl.setFont(new Font("Serif", Font.PLAIN, 15));
JLabel loclbl=new JLabel(getLoc());
```

```
loclbl.setForeground(Color.RED);

loclbl.setFont(new Font("Serif", Font.PLAIN, 13));

JLabel conlbl=new JLabel("                ");

conlbl.setForeground(Color.DARK_GRAY);

conlbl.setFont(new Font("Serif", Font.PLAIN, 12));

labelPanel.add(agentlbl);

labelPanel.add(deptlbl);

labelPanel.add(loclbl);

labelPanel.add(conlbl);

labelPanel.setBorder(BorderFactory.createMatteBorder(2, 2, 2, 2, Color.BLACK));

picPath = getCurrentDirectory() + "/images/"+getUser()+".gif";

LoadPicture(picPath);

JPanel headPanel=new JPanel();

headPanel.setBackground(Color.WHITE);

headPanel.setLayout(new FlowLayout());

headPanel.setBorder(BorderFactory.createMatteBorder(2, 2, 2, 2,
Color.DARK_GRAY));

headPanel.add(labelPanel);

headPanel.add(picturePanel);

if(isClientWindow()){

messageArea = new JTextArea(15, 36);}

else

messageArea = new JTextArea(15, 23);

messageArea.setEditable(false);

messageArea.setWrapStyleWord(true);

messageArea.setLineWrap(true);
```

```
messageArea.setBorder(BorderFactory.createMatteBorder(2, 2, 2, 2,
Color.DARK_GRAY));

if(isClientWindow()){

messageArea.append("Hi "+getCustomer()+"\n You Asked  "+getQuestion()+"\n");}

JPanel messagePanel = new JPanel();

messagePanel.setLayout(new BorderLayout());

messagePanel.add(new JScrollPane(messageArea));

inputArea = new JTextArea(4,18);

inputArea.setWrapStyleWord(true);

inputArea.setLineWrap(true);

inputArea.addKeyListener(this);

inputArea.setBorder(BorderFactory.createMatteBorder(2, 2, 2, 2,
Color.DARK_GRAY));

sendButton=new JButton("    Send");

endChat=new JButton("     End Chat");

endChat.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent e){

Date dateTime = new Date();

SimpleDateFormat format = new SimpleDateFormat("H:mm:ss", Locale.ENGLISH);

setEndTime(format.format(dateTime).toString());

StringBuffer chat=new StringBuffer();

 chat.append(messageArea.getText());

try {

ServerInterface ref = (ServerInterface) Naming.lookup(getRemoteReg());

ref.callLog(getAgent(),getCustomer(),getemail(),getIP(),getStartTime(),getEndTime()
,chat.toString());

ref.setStatus(getIP(),"Online");
```

```
System.out.print("Setting status online : "+getIP());
 ref.timeSetting(getIP(),getEndTime());
System.out.print("\nSetting TimeStamp : "+getIP()+" and time is : "+getEndTime());
System.exit(0);}
catch (Exception oe) { oe.printStackTrace();}}});
JPanel inputPanel=new JPanel(new GridLayout(1,2));
JPanel buttonPane=new JPanel(new GridLayout(2,1));
buttonPane.add(sendButton);
buttonPane.add(endChat);
inputPanel.add(new JScrollPane(inputArea));
inputPanel.add(inputArea);
inputPanel.add(buttonPane);
//Add Buttons
JPanel buttonsPanel = new JPanel(new GridLayout(1,3));
emailButton = new JButton("Email");
emailButton.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent event){
try{
recipients[1]=getemail();
postMail(recipients, getQuestion(), messageArea.getText(),
"omermoeen@mcs.edu.pk");}
catch(Exception exp){}} });
fontButton = new JButton("Font");
emotionsButton = new JButton("Emotions");
emailButton.setEnabled(true);
fontButton.setEnabled(true);
```

```
emotionsButton.setEnabled(true);
buttonsPanel.add(emailButton);
buttonsPanel.add(fontButton);
buttonsPanel.add(emotionsButton);
this.getContentPane().add(headPanel);
this.getContentPane().add(messagePanel);
this.getContentPane().add(buttonsPanel);
this.getContentPane().add(inputPanel);
this.addWindowListener(this);
if(isClientWindow)
this.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
else
this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
this.setTitle(title);
this.setResizable(true);
this.pack();
if(isClientWindow()){
 this.setSize(280, 522);}
else
this.setSize(280, 510);
this.setVisible(true);}
public void sendMessage() {
body = inputArea.getText();
if (body == null) {
body = "Null Message";
```

```
System.out.println("Null Body Messsage");}

else{}

tcpChannel.sendRequest(body,getIP());

inputArea.setText("");}

//mail sending method

public void postMail( String recipients[ ], String subject, String message , String
from) throws MessagingException{

 boolean debug = false;

//Set the host smtp address

Properties props = new Properties();

props.put("mail.smtp.host", "mail.mcs.edu.pk");

props.put("mail.smtp.auth", "true");

Authenticator auth = new SMTPAuthenticator();

Session session = Session.getDefaultInstance(props, auth);

session.setDebug(debug);

 // create a message

 Message msg = new MimeMessage(session);

 // set the from and to address

InternetAddress addressFrom = new InternetAddress(from);

msg.setFrom(addressFrom);

InternetAddress[] addressTo = new InternetAddress[recipients.length];

for (int i = 0; i < recipients.length; i++ {

addressTo[i] = new InternetAddress(recipients[i]);}

msg.setRecipients(Message.RecipientType.TO, addressTo);

 // Setting the Subject and Content Type

msg.setSubject(subject);
```

```
msg.setContent(message, "text/plain");
Transport.send(msg);}
public void sendChatWindow(String IP){
String myIP=IP;
tcpChannel.sendWindow("INVITE:"+myIP.toString(),getIP(),myIP);
System.out.print("chat window req called for :"+getIP());}
public void LoadPicture(String path) {
 JLabel picture = new JLabel(new ImageIcon(path));
picture.setPreferredSize(new Dimension(70, 100));
 picture.setBorder(BorderFactory.createMatteBorder( 2, 1, 2, 2, Color.black));
picturePanel = new JPanel(new GridBagLayout());
picturePanel.setBackground(Color.LIGHT_GRAY);
picturePanel.add(picture); }
public String getCurrentDirectory() {
return System.getProperty("user.dir"); }
public void run() {
init(); }
public int getRemotePort() {
return 6060;}
public void keyPressed(KeyEvent e) {
if (e.getKeyCode() == 10) {
messageArea.append(getUser()+ " says : " +inputArea.getText());
sendMessage();
inputArea.setText(""); } }
public void keyReleased(KeyEvent e) { }
```

```
public void keyTyped(KeyEvent e) {}
public static void main(String args[]) {
ChatWindow newChat = new ChatWindow("title","IP","","","","");
newChat.init();}
public void windowOpened(WindowEvent e) {}
public void setUsername(String u){
user=u;
public void setAgent(String a){
agent=a;  }
 public void setDept(String d){
dept=d;}
public void setLoc(String l){
location=l;}
public void setIP(String ip){
IP=ip; }
public void setRemoteReg(String r){
remoteRegURL=r;}
public void setCustomer(String c){
customer=c;}
public void setEmail(String e){
email=e; }
public void setQuestion(String q){
question=q;}
public void setStartTime(String st){
startTime=st;}
```

```
public void setEndTime(String et){
endTime=et; }
public String getUser(){
return user; }
public String getAgent(){
return agent; }
public String getDept(){
 return dept; }
public String getLoc(){
 return location;  }
public String getIP(){
 return IP;}
public String getRemoteReg(){
return remoteRegURL; }
public String getCustomer(){
return customer;}
public String getemail(){
return email; }
public String getStartTime(){
 return startTime  }
 public String getEndTime(){
 return endTime;}
public String getQuestion(){
return question;}
public void windowClosing(WindowEvent e) {
```

```
System.out.println("Closing Session & Sending BYE Request");
  //  if (parent != null && parent.isSessionColsed()) {
    //  parent.closeSession();   // }}
public void windowClosed(WindowEvent e) {}
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}
public void setParent(AgentConsole parentFrame) {
parent = parentFrame;}
 private class SMTPAuthenticator extends javax.mail.Authenticator {
public PasswordAuthentication getPasswordAuthentication(){
String username = "omermoeen";
String password = "mme-980";
return new PasswordAuthentication(username, password);}}
private void jbInit() throws Exception {}}
```

Bibiography

[1] *"Wikipedia, the free encyclopedia"*.

[2] "http://en.wikipedia.org/wiki/Java_remote_method_invocation", Mach 23, 2008.

[3] Jay Friedman and Jenna Woodul. "Online Customer Support Communities", *LiveWorld. Inc. Trevor Griffith, 2002, 2004*.

[4] Couloris, G., "Distributed Systems, Concepts and Design", *Addison-Wesley Publishing, 1994.*

[5] Farley, J., "Java Distributed Computing",*O'Reilly Publishing, Jan 1998*.

[6] S. Kekre, M.S. Krishnan and K. Srinivasan. "Drivers of customer satisfaction for software products: implications for design and service support". *Management Science* 41 9 (1995), pp. 1456–1470.

[7] B.J. Pine II, D. Peppers, M. Rogers. "Do You Want to Keep Your Customers Forever". *Harvard Business Review (1996).*

[8] "http://java.sun.com/docs/books/tutorial/rmi/index.html", March 30, 2008.

[9] A. Borg and A. Wellings. "A real-time RMI framework for the RTSJ". *In Proceedings of the 15th Euromicro Conference on Real Time Systems. Euromicro, IEEE, July 2003.*

[10] A. de Miguel. "Solutions to Make Java-RMI Time Predictable". *In Proceedings of the 4th IEEE International Symposium on Object-Oriented Real- Time Distributed Computing, pages 379–386, 2001*.

[11] A. Wellings, R. Clark, D. Jensen, and D. Wells. "A framework for integrating the real-time specification for java and java's remote method invocation". *In Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 13–22, 2002.