**1. INTRODUCTION**

**1.1 PROJECT OVERVIEW**

This project deals with the functionalities of the terminal systems connected through a network. This enhances the work efficiency of the administrator and also reduces the physical work strain. The project deals with network Presentation/Training managing utility that will enable a presenter to share and control machines on his/her network and by that making sure that all the participants are attending the session. Sharing to Multiple Clients over Wi-Fi & Local Area Networks

The project is using LAN, by implementing such a software which is enable to carry out operations which are capable to monitor whole of the Connected network, remotely install software, transferring files, sitting on one chair by viewing remote desktop, passing Warning messages to remote system, Group Chatting and is also able to shut down the system by performing remote aborting operations. This software is purely developed in JAVA RMI (REMOTE METHOD INVOCATION). This project is to provide the maximum details about the network to the administrator on their screen without knowing them their users. The administrator can view the static image of client’s desktop and then he/she could sends warning message to the user to stop that operation immediately. Even than if client do not stops than administrator has the facility to abort the system remotely or restart the system whatever necessary he thinks

**1.2 ABOUT THE ORGANIZATION**

Faith Infosys is an ISO 9001: 2008 certified IT firm, committed to provide excellence in web and application based client services. We offer a wide variety of professional services, ranging from high quality application software to highly standard websites. We provide robust, scalable and reliable IT related services to all our clients. Faith Infosys, the leader in high end IT training segment offers an exciting training, specially designed for BCA, MCA, and M.Tech/ME and all other IT related graduates and postgraduates, which will expand their career horizons beyond geographical boundaries. Our curriculum has been structured and designed by our research department working closely with industry leaders and understanding the needs and requirements. It is this approach that has enabled Faith Infosys to emerge as a firm of pro-eminence. Our firm maintains an excellent standard of training which includes live projects, not only to unfold the cryptic world of computer technology, but also develop a culture of creative computing. Since our Establishment in 2008, our prime focus is customer satisfaction. From creative media designs to software applications, we work by understanding and adapting our skills to the client’s needs. Faith Infosys strives to deliver fast and cost effective services without compromising on quality.

**OUR STRENGTH**

Faith Infosys founders have extensive business and training acumen and experience within the IT field. Our greatest strength is a team of aspiring professionals who believe in quality, commitment and competency. The management team consists of IT industry veterans and they are highly trained in latest technologies.

## 

# 2. SOFTWARE REQUIREMENT ANALYSIS

**3.1 FEASIBILITY STUDY**

All projects are feasible, given unlimited resources and infinite time. But the development of software is plagued by the scarcity of resources and difficult delivery rates. It is prudent to evaluate the feasibility of the project at the earliest possible time. An estimate is made of whether the identified user may be satisfied using current hardware and software technologies. The study will decide if the proposed system will be cost effective from the business point of view and if it can be developed in the existing budgetary constraints. The result should inform the decision of whether to go ahead with a more detailed analysis. A feasibility study is not warranted for systems in which economic justification is obvious, technical risk is low, few legal problems are expected and no reasonable alternative exists.

Three key considerations are involved in feasibility analysis.

**3.1 Technical Feasibility**

The T-SHELL software is developed using the java environment. The reason for using java, as the development platform is that, java is an Object Oriented Language which handles most of the networking concepts. Since java is a platform independent language, the class files can be executed on any operating system easily.

The technical feasibility issues usually raised during the feasibility stage of investigation includes these

* This software can run in windows Operating system
* The hardware required is the minimum hardware for running Java swing.
* The software can be expanded easily to include more functionalities.

**3.2 Economic Feasibility**

This is the most frequently used method for evaluating the effectiveness of a system. It is also called as a cost analysis. The **T-SHELL** project, which is used to control all the remote systems in a network, requires resources such as the software and hardware components that support the Remote Control through java project effectively. Since all the clients are usually connected to the server in any organization, it reduces the cost factor. Hence there is no need of further physical connection to be established between the server and the client.

**3.3 Operational Feasibility**

The T-Shell project is a user-friendly tool developed in order to make the operations of the administrator much better. It will be easy for the administrator to handle all the systems in the network from the server itself which helps in increasing the operational efficiency of the administrator.

**3.2 System Specification**

This stage involves studying the existing system and interacting with the users, which determine user requirements and their expectations of the proposed system. Cost of incorporating changes required by the user is very less at this stage that steeply increases as development advance. System study is a general term that refers to an orderly, structured process for identifying and solving problems. The first phase of software development is system study. The importance of system study phase is the establishment of the requirements for the system to be acquired, developed and installed. Analyzing the project to understand the complexity forms the vital part of the system study. Problematic areas are identified and information is collected. Fact finding or gathering is essential to any analysis of requirements. It is also highly essential that the analyst familiarize himself with the objectives, activities and functions of organizations in which the system is to be implemented.

System study works with users to identify goals and build systems to achieve them. System study is an important phase of any system development process. The system is studied to the minute details and analyzed. The system analyst plays the role of an interrogator and dwells deep into the working of the present system. The system is viewed as a whole and the inputs to the system are identified. The outputs from the system are traced through various processing. During system study, data are collected on available sources, network data centers handled by the present system. Once system study is completed, the analyst has a firm understanding of what is to be done. If the information gathered in the system study is not enough to make a good software application, then we have to reschedules a new system study. Cost of incorporating changes required by the user is very less at this stage that steeply increases as development advance.

**3.4.2 Purpose and scope**

The SRS phase consists of two basic activities:

* **Problem/Requirement Analysis:** The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.
* **Requirement Specification:** Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity. The requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic goal of this phase.

**3.4.3 Software Requirement**

OPERATING PLATFORM : Windows XP or Above

FRONT END : JAVA, RMI, SWINGS

**3.4.4 Hardware Specifications**

PROCESSOR : Intel Pentium Dual Core or above

RAM : 1 GB

HARD DISK : 80 GB

DISPLAY : COLOR MONITOR

KEY BOARD : 104 STANDARDS

LAN : ENABLED

## 3. SOFTWARE DEVELOPMENT ENVIRONMENT

**3.1 Introduction to Java**

In the early days of the web, a server could dynamically construct a page by creating a separate process to handle each client request. The process would open connection to one or more databases in order to obtain the necessary information. It communicated with the web server via an interface known as the Common Gateway Interface (CGI) .CGI allowed the separate process to read data from HTTP request and write data to the HTTP response. A variety of different languages were used to build CGI programs including C, C++ and Perl.

**3.1.1 Java’s Magic: the Byte Code**

The key that allows java to solve both the security and the portability problems just described is that, the output of the java compiler is not an executable code. Rather, it is Byte Code. Byte Code is a highly optimized set of instructions designed to be executed by virtual machine that the java Run-time system emulates. However, the fact that a java program is interpreted helps solve the major problems associated with downloading the program over the Internet.

Here is why java was designed to be interpreted language. Because java programs are interpreted rather than compiled .It is easier to run them in wide variety of environments. Only the java runtime system needs to be implemented for each platform. Once the runtime package exists for a given system any java program can run on it. If java were a compiled language then different versions of the same program will have to exist for each type of CPU connected to the Internet. Thus interpretation is the easiest way to create truly portable programs. Although java was designed to be interpreted, there is technically nothing about java that prevents on the fly compilation of Byte Code into native code. However, even if dynamic compilation were applied to Byte Code, the portability and safety would still apply, because the run time system would still be in change of the execution environment.

**3.1.2 The Java Buzz Words**

No discussion of the genesis of java is complete without a look at the java buzzwords. Although the fundamentals that necessitated the invention of java are portability and security, there are other factors that played an important role on molding the final form of the language.

The java in the following list of buzzwords summed up the key considerations.

* Simple
* Portable
* Object-oriented
* Robust
* Multithreaded
* Architectural-neutral
* High performance
* Distributed
* Dynamic

**3.1.3 Object Oriented Programming and Java**

Object-oriented Programming was developed because of limitations found in earlier approaches of programming. In order appreciate what OOP does, we need to understand the limitations present in the traditional programming.

* **Procedural Languages**

The procedural languages include Pascal, C, Basic, FORTRAN, and some other similar languages. Each statement in these languages tells the computer to perform some operation. A program in a procedural language is a list of instructions.

For very small programs no other organizing principle (often called a paradigm) is needed. The programmer creates the list of instructions, and the computer carries them out.

* **Division into Functions**

When programs become larger, a single list of instructions becomes unwieldy. Few programmers can comprehend a program of more than a few hundred statements unless it is broken down into smaller units. For this reason the function was adopted as a way to make programs more comprehensible to their human creators. (The term function is used in C++ and C. In other languages the same concept may be referred to as a subroutine, a subprogram, or a procedure.) A program is divided into functions, and (ideally, at least) each function has a clearly defined purpose and a clearly defined interface to the other functions in the program.

The idea of breaking a program into functions can be further extended by grouping a number of functions together into a larger entity called a module, but the principle is similar: grouping a number of components that carry out specific tasks.

Dividing a program into functions and modules is one of the cornerstones of structured programming, the somewhat loosely defined discipline that has influenced programming organization for more than a decade.

* **Problems with Structured Programming**

As programs grow ever larger and more complex, even the structured programming approach begins to show signs of strain. You may have heard about the problems involved in program development. The project is too complex, the schedule slips, more programmers are added, complexity increases, costs skyrocket, the schedule slips further, and disaster ensues. Analyzing the reasons for these failures reveals that there are weaknesses in the procedural paradigm itself. No matter how well the structured programming approach is implemented, large programs become excessively complex. The reasons for this failure of procedural languages are mainly because of the role played by data.

* **Relationship to the Real World**

Procedural programs are often difficult to design. The problem is that their chief components--functions and data structures--don't model the real world very well. For example, suppose you are writing a program to create the elements of a graphics user interface: menus, windows, and so on. Quick now, what functions will you need? What data structures? The answers are not obvious, to say the least. It would be better if windows and menus corresponded more closely to actual program elements.

* **New Data Types**

There are other problems with traditional languages. One is the difficulty of creating new data types. Computer languages typically have several built-in data types: integers, floating-point numbers, characters, and so on. What if you want to invent your own data type? Perhaps you want to work with complex numbers, or two dimensional coordinates, or dates—quantities the built-in data types don’t handle easily. Being able to create your own types is called extensibility; you can extend the capabilities of the language. Traditional languages are not usually extensible. Without unnatural convolutions, you can’t bundle together both X and Y coordinates into a single variable called Point, and then add and subtract values of this type. The result is that traditional programs are more complex to write and maintain.

* **The object oriented approach**

The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object.

An object’s functions, called member methods in Java, typically provide the only way to access its data. If you want to read the item and return the value to you, you call a member function in the object. It will read the item and return the value to you. You can’t access the data directly. The data is hidden, so it is safe from accidental modification. Data and its functions are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the description of object oriented languages.

A Java program typically consists of a number of objects, which communicate with each other by calling one another’s members functions. We should mention that what are called member functions in C++ are called methods in Java. Also, data items are referred to as instance variables. Calling an object’s member function is referred to as sending a message to the object.

* **Classes**

In OOP we say that objects are members of classes. What does this mean? Let’s look at an analogy. Almost all computer languages have built-in data types. For instance, a data type int, meaning integer is pre-defined in Java. You can declare as many variables of type int as you need in your program:

Int day;

Int count;

Int divisor;

Int answer;

A class serves as a plan, or template. It specifies what data, and what functions will be included in objects of that class. Defining the class doesn’t create any objects, just as the mere existence of a type int doesn’t create any variables.

A class is thus a collection of similar objects. This fits our non-technical understanding of the word class, Prince, sting etc., are members of the class of rock musicians. There is no person called rock musician but specific people with specific names are members of this class if they possess certain characteristics.

* **Abstraction**

An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior. This abstraction allows people to use a car to drive to the grocery store without being overwhelmed by the complexity of the parts that form the car. They can ignore the details of how the engine, transmission, and braking systems work. Instead they are free to utilize the object as a whole.

A powerful way to manage abstraction is through the use of hierarchical classifications. This allows you to layer the semantics of complex systems, breaking them into more manageable pieces. From the outside, the car is a single object. Once inside, you see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone, and so on. In turn, each of these subsystems is made up of more specialized units. For instance, the sound system consists of a radio, a CD player, and/or a tape player. The point is that you manage the complexity of the car (or any other complex system) through the use of hierarchical abstractions.

Hierarchical abstractions of complex systems can also be applied to computer programs. The data from a traditional process-oriented program can be transformed by abstraction into its component objects. A sequence of process steps can become a collection of messages between these objects. Thus, each of each object describes its own unique behavior. You can treat these objects as concrete entities that respond to messages telling them to do something. This is the essence of object-oriented programming.

Object-oriented concepts form the heart of Java just as they form the basis for human understanding. It is important that you understand how these concepts translate into programs. As you will see, object-oriented programming is a powerful and natural paradigm for creating programs that survive the inevitable changes accompanying the life cycle of any major software project, including conception, growth, and aging. For example, once you have a well-defined objects and clean, reliable interfaces to those objects, you can gracefully decommission or replace parts of an older system without fear.

* **Encapsulation**

Encapsulation is the process of binding the code and the data, thus providing security from the outside interface. The other way of defining encapsulation is by providing the code and data with a protective wrapper, thus preventing them from being accessed by the code present outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface. In Java the basis of encapsulation is the class. A class defines the structure and behavior (data and code) that will be shared by a set of objects. Each object of a given class contains the structure and behavior defined by the class. For this reason, objects are sometimes referred to as instances of a class. Thus, a class is a logical construct; an object has physical reality.

Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods. Since the purpose of a class is to encapsulate complexity, there are mechanisms for hiding the complexity of the implementation inside the class. Each method or variable in a class may be marked private or public. The public interface of a class represents everything that external users of the class need to know, or may know. The private methods and data can only be accessed by code that is a member of the class. Therefore, any other code that is not a member of the class cannot access a private method or variable. Since the private members of a class may only be accessed by other parts of your program through the class’ public methods, you can ensure that no improper actions take place. Of course, this means that the public interface should be carefully designed not to expose too much of the inner workings of a class.

* **Inheritance**

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. As mentioned earlier, most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal. Without the use of hierarchies, each object would need to define all of its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

Most people naturally view the world as made up of objects that are related to each other in a hierarchical way, such as animals, mammals, and dogs. If you wanted to describe animals in an abstract way, you would say they have some attributes, such as size, intelligence, and type of skeletal system. Animals also have certain behavioral aspects; they hear, breathe, and sleep. This description of attributes and behavior is the class definition for animals.

If you wanted to describe a more specific class of animals, such as mammals, they would have more specific attributes, such as type of teeth, and mammary glands. This is known as a subclass of animals, where animals are referred to as mammals’ super class.

Since mammals are simply more precisely specified animals, they inherit all of the attributes from animals. A deeply inherited subclass inherits all of the attributes from each of its ancestors in the class hierarchy.

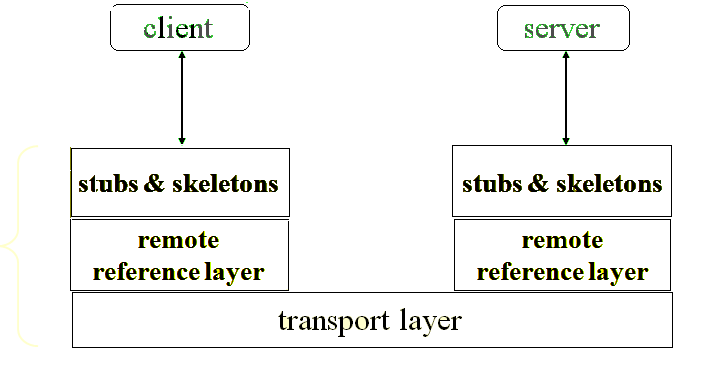
Inheritance interacts with encapsulation as well. If a given class encapsulates some attributes, then any subclass will have the same attributes plus any that it adds as part of its specialization. This is a key concept, which lets object-oriented programs grow in complexity linearly rather than geometrically. A new subclass inherits all of the attributes of all of its ancestors. It does not have unpredictable interactions with the majority of the rest of the code in the system.

* **Polymorphism**

Polymorphism (from the Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stack. One stack is used for integer values, one for floating-point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three difference sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names. More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods.” This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler’s job to select the specific action (that is, method) as it applies to each situation.

**3.2 Introduction to RMI**

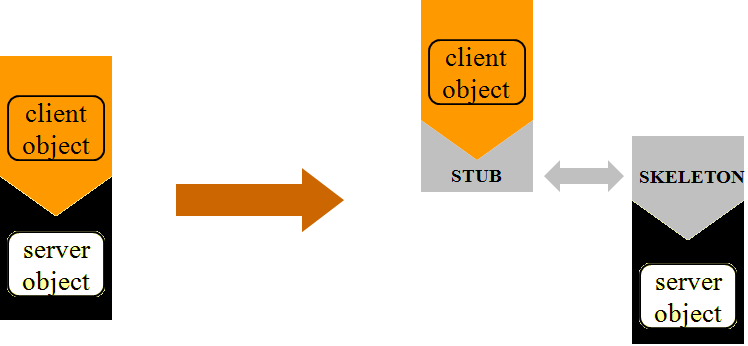
Remote Method Invocation (RMI) facilitates object function calls between Java Virtual Machines (JVMs). JVMs can be located on separate computers - yet one JVM can invoke methods belonging to an object stored in another JVM. Methods can even pass objects that a foreign virtual machine has never encountered before, allowing dynamic loading of new classes as required. Remote method invocation allows applications to call object methods located remotely, sharing resources and processing load across systems. Unlike other systems for remote execution which require that only simple data types or defined structures be passed to and from methods, RMI allows any Java object type to be used - even if the client or server has never encountered it before. Any object that can be invoked this way must implement the Remote interface. When such an object is invoked, its arguments are “marshaled” and sent from the local virtual machine to the remote one, where the arguments are “unmarshalled”. When the method terminates, the results are marshaled from the remote machine and sent to the caller's virtual machine. If the method invocation results in an exception being thrown, the exception is indicated to caller.



**Fig3.2.1: RMI architecture**

* **Locate Remote Objects**

Applications can use one of two mechanisms to obtain references to remote objects. An application can register its remote objects with RMI's simple naming facility, the rmiregistry, or the application can pass and return remote object references as part of its normal operation.

**CONCEPT**

**Fig3.2.2: Concept of RMI**

* **Communicate with remote objects**

Details of communication between remote objects are handled by RMI; to the programmer, remote communication looks like a standard Java method invocation.

* **Load class byte codes for objects that are passed**

Because RMI allows a caller to pass objects to remote objects, RMI provides the necessary mechanisms for loading an object's code, as well as for transmitting its data. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it.

**Advantages of RMI**

One of the important features of RMI is its ability to download the byte codes (or simply code) of an object's class even though the class is not defined in the receiver's virtual machine. The type and behavior of an object, previously available only in a single virtual machine, can be transmitted to another, virtual machine. RMI passes objects by their true type, and hence the behavior of those objects is not changed, when they are sent to another virtual machine. This allows new types to be introduced into a remote virtual machine, thus extending the behavior of an application dynamically.

RMI is made up of interfaces and classes. The methods used in RMI are defined using the interfaces which are implemented by the classes. In a distributed application some of the implementations are assumed to reside in different virtual machines.

Objects that have methods that can be called across virtual machines are remote objects. An object becomes remote by implementing a remote interface, which has the following characteristics. A remote interface extends the interface java.rmi.Remote. When the object is passed from one virtual machine to another, Instead of making a copy of the implementation object in the receiving virtual machine, RMI passes a remote stub for a remote object. The stub acts as the local representative, or proxy.

The caller invokes a method on the local stub, which is responsible for carrying out the method call on the remote object. A stub for a remote object implements the same set of remote interfaces that the remote object implements. This allows a stub to be cast to any of the interfaces that the remote object implements. However, this also means that only those methods defined in a remote interface are available to be called in the receiving virtual machine.

* **Defining the remote interfaces**

A remote interface specifies the methods that can be invoked remotely by a client. Clients program to remote interfaces, not to the implementation classes of those interfaces. Part of the design of such interfaces is the determination of any local objects that will be used as parameters and return values for these methods. In case of any of these interfaces or classes being not available, the user has to define them.

* **Implementing the remote objects**

Remote objects must implement one or more remote interfaces. The remote object class may include implementations of other interfaces (either local or remote) and other methods (which are available only locally). If any local classes are to be used as parameters or return values to any of these methods, they must be implemented as well.

* **Implementing the clients**

Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects has been deployed.

* **Compile Sources and Generate Stubs**

This is a two-step process. In the first step the user uses the javac compiler to compile the source files, which contain the implementation file, interfaces, the server program and the client program. In the second step you use the rmic compiler to create stubs for the remote objects. RMI uses a remote object's stub class as a proxy in clients so that clients can communicate with a particular remote object.

**Working of an RMI applications**

The working of RMI applications is based on accessing the remote object. A remote object can be defined as an object associated with methods that can be called from another java virtual machine. In this mechanism the application that access to the remote object is treated as client application and the application that implements that object is treated as a server application. Thus an RMI application can be satirized into two parts.

1. Server side application

2. Client side application

* **Server side application**

The first file is an interface that declares the methods, which are accessed remotely. The purpose of making an interface is to enable the client to access the methods of an interface with the help of its reference. The second file creates a class and the binds that object in RMI registry. An RMI registry is the naming server, which allows the remote request to be redirected to an object, which is bind inside it.

* **Client side application**

On a client side, a file is created that creates a class access that remote object through RMI registry and invokes the remote methods.

* **Creating RMI server**

The first side to create an RMI application is to create server side application. For server side application, the first thing to create is an interface that defines the methods, which are accessed by the remote clients. The code for defining an interface is given below which is the first step towards creating server application.

**Step 1 Defining a Remote Interface**

To call the methods of an object remotely, you have to create a remote interface that should extend the java.rmi.Remote interface. This is the first steps in creating an RMI application, which extends the Remote interface. All the method declares in this interface can be called remotely. Each and every methods declares in the remote interface must have the java.rmi.RemoteException in the throws clause as they throw this exception whenever there is an error in sending or receiving information.

**Remote Interface**

import java.rmi.\*;

import java.rmi.server.\*;

Public interface Intf extends Remote

{

public int add(int a,int b)throws RemoteException;

public int sub(int a1,int b1) throws RemoteException;

}

**Implementation of an interface**

In this step the interface created in step 1 is implemented. The class, which implements the interface, must extend java.rmi.server.unicastRemoteObject. The purpose of an extending the UnicasRemoteObject class is to provide the functionality to import an object from a remote machines. The code for implementation of an interface is given below

import java.rmi.\*;

import java.rmi.server.\*;

Public class serverIMPL extends UnicastRemoteObject implements Intf

{

public serverIMPL() throws RemoteException

{

}

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

{

return(a+b);

}

public int sub(int a1,int b1) throws RemoteException

{

return(a1-b1);

}

}

**Step 3 creating server classes:**

This step contains the main program for the server machine by registering the object of the class, which implements the interface. This is done through bind () and rebind () method found in java.rmi.Naming class bind () and rebind () are static methods found in the naming class in the code given code given below the method rebind () has been used

import java.rmi.\*;

import java.rmi.server.\*;

public class Server

{

public static void main (String s[]) THROWS Exception

{

serverimpl impl=new serverIMPL()

System.out.printl(“initializing server…………………………….”);

Registry registry = LocateRegistry.createRegistry(1099);

registry.rebind(“myMessage”, impl);

System.out.println(“registered………”);

}

}

**Creating client**

The next step is creating an RMI application is to create a client that will actually invoke the remoter methods. In this program, you will request a reference of that object which is present in the registry. This is done through the method lookup () which is present inside the class java.rmi.Naming. The method lookup () is a static method, which takes the RMI URL that points to the registry as a parameter and, if it finds the object, will return a reference to it. The code for creating the clients is given below

Import java.rmi.\*;

Import java.rmi.server.\*;

Public class Client

{

public static void main (String s[]) throws Exception

{

Registry myRegistry = LocateRegistry.getRegistry("127.0.0.1", 1099);

// search for myMessage service

Intf intf1= (Intf) myRegistry.lookup("myMessage");

Int a =intf1.add(10,20);

Int b=intf1.sub(30,10);

System.out.println(a);

System.out.println(b);

}

}

**Configuring and executing client and server**

To execute the codes written while creating server and client, you have to follow certain steps Note before following the steps given below you have to set the classpath to current working directory

**Step1:**

Compile all the four source file i.e intf.java, serversimple.java, sever.java and client.java

**Step 2:**

The next step is to create the stub and skeleton of the serveIMPL for the creation of the RMOI server. For this purpose you have to use the rmic compiler rmic serverIMPL

The rmic command will create the following 2 files

1. ServerIMPL\_Skel.class

2. SeverIMPL\_stub.class

**Step 3:**

Start the rmiregistry by typing command rmiregistry in the command prompt of the server machine

c:\rmiexample>rmiregistry

**3.3 Introduction to SWINGS**

**A Swing Architecture Overview**



**Fig 3.3.1 Overview of Swings**

Most Swing developers know by now that Swing components have a separable model-and-view design. And many Swing users have run across articles saying that Swing is based on something called a "modified MVC (model-view-controller) architecture."

But accurate explanations of how Swing components are designed, and how their parts all fit together, have been hard to come by until now.

The silence ends with the publication of this article, a major white paper on Swing component design. It provides a comprehensive technical overview of Swing's modified MVC structure and demystifies many other facets of Swing component architecture as well.This document presents a technical overview of the Swing component architecture. In particular, it covers the following areas in detail:

* [Design goals](http://java.sun.com/products/jfc/tsc/articles/architecture/#design_goals)
* [Roots in MVC](http://java.sun.com/products/jfc/tsc/articles/architecture/#roots)
* [Separable model architecture](http://java.sun.com/products/jfc/tsc/articles/architecture/#separable)
* [Pluggable look-and-feel architecture](http://java.sun.com/products/jfc/tsc/articles/architecture/#pluggable)

**Design Goals**

The overall goal for the Swing project was:

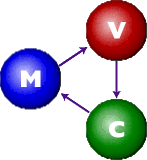
To build a set of extensible GUI components to enable developers to more rapidly develop powerful Java front ends for commercial applications. To this end, the Swing team established a set of design goals early in the project that drove the resulting architecture. These guidelines mandated that Swing would:

1. Be implemented entirely in Java to promote cross-platform consistency and easier maintenance.
2. Provide a single API capable of supporting multiple look-and-feels so that developers and end-users would not be locked into a single look-and-feel.
3. Enable the power of model-driven programming without requiring it in the highest-level API.
4. Adhere to JavaBeans design principles to ensure that components behave well in IDEs and builder tools.
5. Provide compatibility with AWT APIs where there is overlapping, to leverage the AWT knowledge base and ease porting.

**Roots in MVC**

Swing architecture is rooted in the model-view-controller (MVC) design that dates back to Small Talk. MVC architecture calls for a visual application to be broken up into three separate parts:

* A model that represents the data for the application.
* The view that is the visual representation of that data.
* A controller that takes use input on the view and translates that to changes in the model.



Early on, MVC was a logical choice for Swing because it provided a basis for meeting the first three of our design goals within the bounds of the latter two. The first Swing prototype followed a traditional MVC separation in which each component had a separate model object and delegated its look-and-feel implementation to separate view and controller objects.

**Separable model architecture**

It is generally considered good practice to center the architecture of an application around its data rather than around its user interface. To support this paradigm, Swing defines a separate model interface for each component that has a logical data or value abstraction. This separation provides programs with the option of plugging in their own model implementations for Swing components.

The following table shows the component-to-model mapping for Swing.

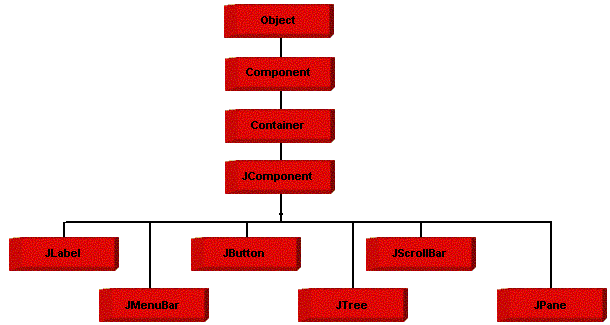
|  |  |  |
| --- | --- | --- |
| **Component** | **Model Interface** | **Model Type** |
| JButton | ButtonModel | GUI |
| JToggleButton | ButtonModel | GUI/data |
| JCheckBox | ButtonModel | GUI/data |
| JRadioButton | ButtonModel | GUI/data |
| JMenu | ButtonModel | GUI |
| JMenuItem | ButtonModel | GUI |
| JCheckBoxMenuItem | ButtonModel | GUI/data |
| JRadioButtonMenuItem | ButtonModel | GUI/data |
| JComboBox | ComboBoxModel | data |
| JProgressBar | BoundedRangeModel | GUI/data |
| JScrollBar | BoundedRangeModel | GUI/data |
| JSlider | BoundedRangeModel | GUI/data |
| JTabbedPane | SingleSelectionModel | GUI |
| JList | ListModel | data |
| JList | ListSelectionModel | GUI |
| JTable | TableModel | data |
| JTable | TableColumnModel | GUI |
| JTree | TreeModel | data |
| JTree | TreeSelectionModel | GUI |
| JEditorPane | Document | data |
| JTextPane | Document | data |
| JTextArea | Document | data |
| JTextField | Document | data |
| JPasswordField | Document | data |

**Fig 3.3 component-to-model mapping for Swing**

In Swing , classes that represent GUI components have names beginning with the letter J. Some examples are JButton, JLabel, and JSlider. Altogether there are more than 250 new classes and 75 interfaces in Swing — twice as many as in AWT.

**Java Swing class hierarchy:**

The class **JComponent**, descended directly from Container, is the root class for most of Swing’s user interface components.



**Fig 3.3.2 Java Swing class hierarchy**

Swing contains **components**  that you’ll use to build a GUI. I am listing you some of the commonly used Swing components. To learn and understand these swing programs, AWT Programming knowledge **is not**  required.

**Note:** Below are some links to java swing tutorials that forms a helping hand to get started with java programming swing.

* JPanel is Swing’s version of the AWT class Panel and uses the same default layout, FlowLayout. JPanel is descended directly from JComponent.
* [JFrame](http://www.javabeginner.com/java-swing/java-jframe-class-example) is Swing’s version of Frame and is descended directly from that class. The components added to the frame are referred to as its contents; these are managed by the contentPane. To add a component to a JFrame, we must use its contentPane instead.
* [JInternalFrame](http://www.javabeginner.com/java-swing/java-jinternalframe-class-example) is confined to a visible area of a container it is placed in. It can be iconified, maximized and layered.
* [JWindow](http://www.javabeginner.com/java-swing/java-jwindow-class-example) is Swing’s version of Window and is descended directly from that class. Like Window, it uses BorderLayout by default.
* JDialog is Swing’s version of Dialog and is descended directly from that class. Like Dialog, it uses Border Layout by default. Like JFrame and JWindow,  
  JDialog contains a rootPane hierarchy including a contentPane, and It allows layered and glass panes. All dialogs are modal, which means the current thread is blocked until user interaction with it has been completed. JDialog class is intended as the basis for creating custom dialogs; however , some of the most common dialogs are provided through static methods in the class JOptionPane.
* [JLabel](http://www.javabeginner.com/java-swing/java-jlabel-class-example), descended from JComponent, is used to create text labels.
* The abstract class AbstractButton extends class JComponent and provides a foundation for a family of button classes, including JButton.
* [JTextField](http://www.javabeginner.com/java-swing/java-jtextfield-class-example) allows editing of a single line of text. New features include the ability to justify the text left, right, or center, and to set the text’s font.
* [JPasswordField](http://www.javabeginner.com/java-swing/java-jpasswordfield-class-example) (a direct subclass of JTextField) you can suppress the display of input. Each character entered can be replace by an echo character.This allows confidential input for passwords, for example. By default, the echo character is the asterisk, \*.
* [JTextArea](http://www.javabeginner.com/java-swing/java-jtextarea-class-example) allows editing of multiple lines of text. JTextArea can be used in conjunction with class JScrollPane to achieve scrolling. The underlying JScrollPane can be forced to always or neverhave either the vertical or horizontal scrollbar;  
  JButton is a component the user clicks to trigger a specific action.
* [JRadioButton](http://www.javabeginner.com/java-swing/java-jbutton-class-example) is similar to JCheckbox, except for the default icon for each class. A set of radio buttons can be associated as a group in which only one button at a time can be selected.
* [JCheckBox](http://www.javabeginner.com/java-swing/java-jcheckbox-class-example) is not a member of a checkbox group. A checkbox can be selected and deselected, and it also displays its current state.

# 

# 4. SYSTEM ANALYSIS AND DESIGN

## 4.1 EXISTING SYSTEM

* In the present generation systems, there is a need for the administrator has to go all around the network in order to terminate any system that is left non-terminated.
* The administrator has to take all the trouble of going to a particular system to access a file that is needed by him.
* In order to get the system configuration details of any particular system, the administrator has to take the trouble of going to that system for obtaining the information.
* The processes that are running in a particular system can be viewed only in that system itself by using the present generation software’s.

**Problem Domain**

This project deals with the functionalities of the terminal systems connected through a network. This system implements Remote control at the application level.

**4.2 PROPOSED SYSTEM**

The disadvantages present in the existing systems can be overcome using the proposal systems.

* Using the **T-SHELL** software the administrator can control the operations of the remote system from his system itself.
* The administrator can install software in the remote system from the server system itself using this software.
* The system also assures security by blocking the usage of external devices during the Training/examination time and also notifies the administrator of such situations.
* In order to terminate the operations on the remote systems, the administrator can obtain the current process details of the remote systems from the server itself.
* The administrator can monitor remote system from his system itself.
* Both administrator and remote system can transfer files to each other.
* The system can also be used while teaching as the usage of projectors can be avoided as the Presenter screen will be shared to all the students.

## 4.3 MODULE DESCRIPTION

The **T-SHELL (Trainer’s Shell)** software is divided in to two modules. These modules are as follows.

* Presentation
* Training

**4.3.1 Presentation**

This program module provides a User Graphic Interface that will be available to all computers currently attached on the network and will not allow the users to select and view information other than what the admin is sharing. The users can only view the shared desktop of the admin. The main core function of the utility is to communicate the presentation with the client program in regular intervals and make sure that they are attending the presentation. The UGI of this module provided in the admin side provides information on all those are attending the session, and notifies if they try to quit the session.

**4.3.2 Training**

In the Training session Admin can also allow the clients to use only selected software in special cases like examination, and can monitor weather he is using any other software. The admin also get the notification whenever a client attempts to connect an external devices. Admin also monitor client’s activity by using this system and can also sent notification in case of any abnormal activity detection.

This module makes sure that the client could use only those facilities which are allowed by the admin or trainer. This section will get permission to run other software only if it allowed by the admin and if any other software is executed the process will be monitored and killed by admin. He will also get notifications from admin during the session.

Modules are described by the following categories

* + Remote Desktop
  + Chat
  + Retrieve
  + Control
  + Pen drive detection
  + File Transfer
  + **Remote desktop**

Using this module we can access the desktop of any remote system by giving the desired system’s IP address and port number we can get the desktop of that system. It is useful for the administrator to monitor the activities of the employees of that organization.

The administrator can be able to get the desktop of that particular system and can warn the employees if they are doing any illegal activities.

* + **Retrieve**

The purpose of the Retrieve using this module can determine the processes that are currently running in a particular client system, present in a network. The administrator can know the details of the processes by specifying the IP address of the remote system. By using the process information we can determine the processes running at the client system at that time. By knowing these information the server can know requirements for installing software remotely. This module is similar to that of a task manager present in the windows operating system.

* + **Control**

The Control module handles the terminal operations such as shutdown, restart and logoff, by specifying the IP address of the remote system. Additionally, admin have the ability to take control of a remote computer by controlling the mouse and keyboard. For this purpose we are using RMI (Remote Method Invocation) and Java Runtime class.

This module asks the administrator to choose the system address (IP) on which he wants to perform the terminal operations. It also asks the administrator to mention one of the above three terminal operations to be performed on the remote system. Once the administrator has confirmed the action, an echo message is sent to the remote system in order to ensure whether the user is using the system or not. In case of the system being accessed by the user a response is sent to the administrator. The administrator can either suspend or continue with his actions after receiving the response. If the administrator knows that no one is using the system or no processes running at that time then he can use these options like shutdown, restart or logoff.

* + **File Transfer**

Using this module we can transfer files between the client and the server based on the IP address. The administrator can transfer file with desired user or client can transfer file to admin.

* + **Pen Drive Detection**

This module defines the detection of the pen drive port. When the external device is added to the client systems, then the server will get a message from that particular client system with IP address. Then the server knows. This is used in any lab examinations or training time. If any student will try for copying by using pen drives then the server knows that client system and will take an action on him. Additionally server can format that pen drive remotely.

* + **Chat**

This module provides us the chat facility. Using this chat the server system and the client system can communicate with each other. The administrator can chat with the all user connected in the network.

## 4.4 SYSTEM DESIGN

This section defines the general description of the functionality, context and design of the project.

**4.4.1 Definition**

The most creative and challenging face of the system development is System Design. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Design goes through the logical and physical stages of development. In designing a new system, the system analyst must have a clear understanding of the objectives, which the design is aiming to fulfill. The first step is to determine how the output is to be produced and in what format. Second, input data and master files have to be designed to meet the requirements of the proposed output. The operational phases are handled through program construction and testing.

Thus system design is a solution to “how to” approach to the creation of a new system. Thus important phase provides the understanding and the procedural details necessary for implementing the system recommended in the feasibility study. The design step provides a data design, architectural design, and a procedural design.

**4.4.2 INPUT DESIGN**

Input is a process of converting user-oriented inputs into computer based formats, and the data entry screen is designed. Inaccurate input data is the most common cause of error in data processing. Well-designed input serves five purposes; control work flow, reduce redundancies, record data increases clerical accuracy and allow easier checking of data. Input design is the most important part of the overall system design which requires very care full attention

**4.4.3 Output Design**

Outputs are the very essence of our system. In the system, Output is obtained on the screen as well as on the papers, as reports. Efficient, intelligible output design should improve the system relationship with the user and the helps in decision making. A major form of output is hard copy from the printer.

**4.5 DATA FLOW DIAGRAM**

A data flow diagram is a graphical technique that depicts information flow and transforms that are applied as data move from input to output. The DFD is also known as Data Flow Graph or Bubble Chart. The DFD is used to represent increasing information flow and functional details. Also DFD can be stated as the starting point of the design phase that functionally decomposes the requirements specifications down to the lowest level of detail.

A Level 0 also called a fundamental system model or a context level DFD that represent the entire software elements as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively. Additional process and information flow parts are represented in the next level, i.e., level 1 DFD. Each of the processes represented at level 1 are sub functions of overall system depicted in the context model. Any processes that are complex in level 1 will be further represented into sub functions in the next level, i.e., level 2.

Data flow diagram is a means of representing a system at any level of detail with a graphic network of symbols showing data flows, data stores, data processes and data sources. The purpose of data flow diagram is to provide a semantic bridge between users and system developers. The diagram is the basis of structured system analysis. A DFD describes what data flows rather than how they are processed, so it does not depend on hardware, software, data structure or file organization.

**Components of Data Flow Diagram**

There are four symbols that are used in the drawing of Data Flow Diagrams:

* **Entities**



External entities represent the sources of data that enter the system or the recipients of data that leave the system.

* **Process**

Processes represent activities in which data is manipulated by being stored or retrieved or transformed in some way. A circle represents it. The process will show the data transformation or change.

* **Databases**

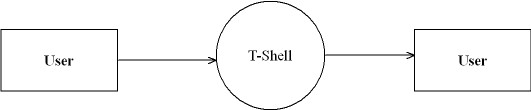
Databases represent storage of data within the system

* **Data flow**

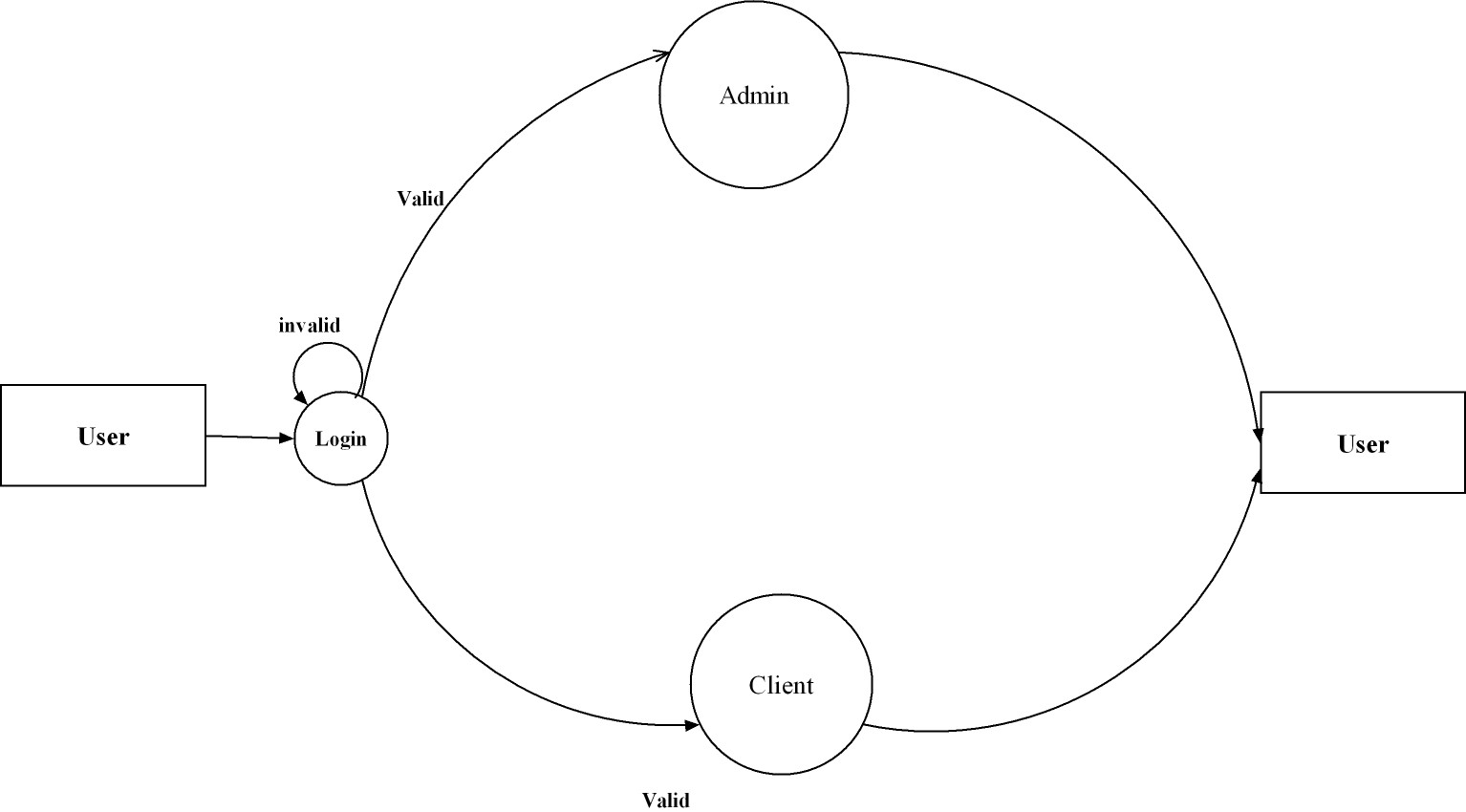
A data flow shows the flow of information from its source to its destination. A line represents a data flow, with arrowheads showing the direction of flow.

**DATA FLOW DIAGRAM**

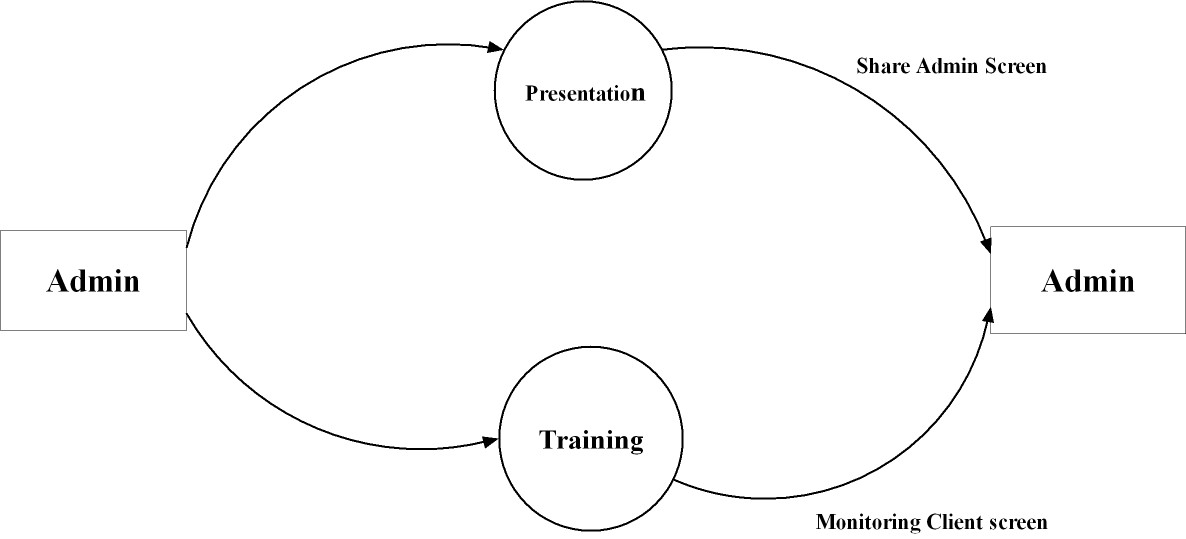
**Level – 0**

****

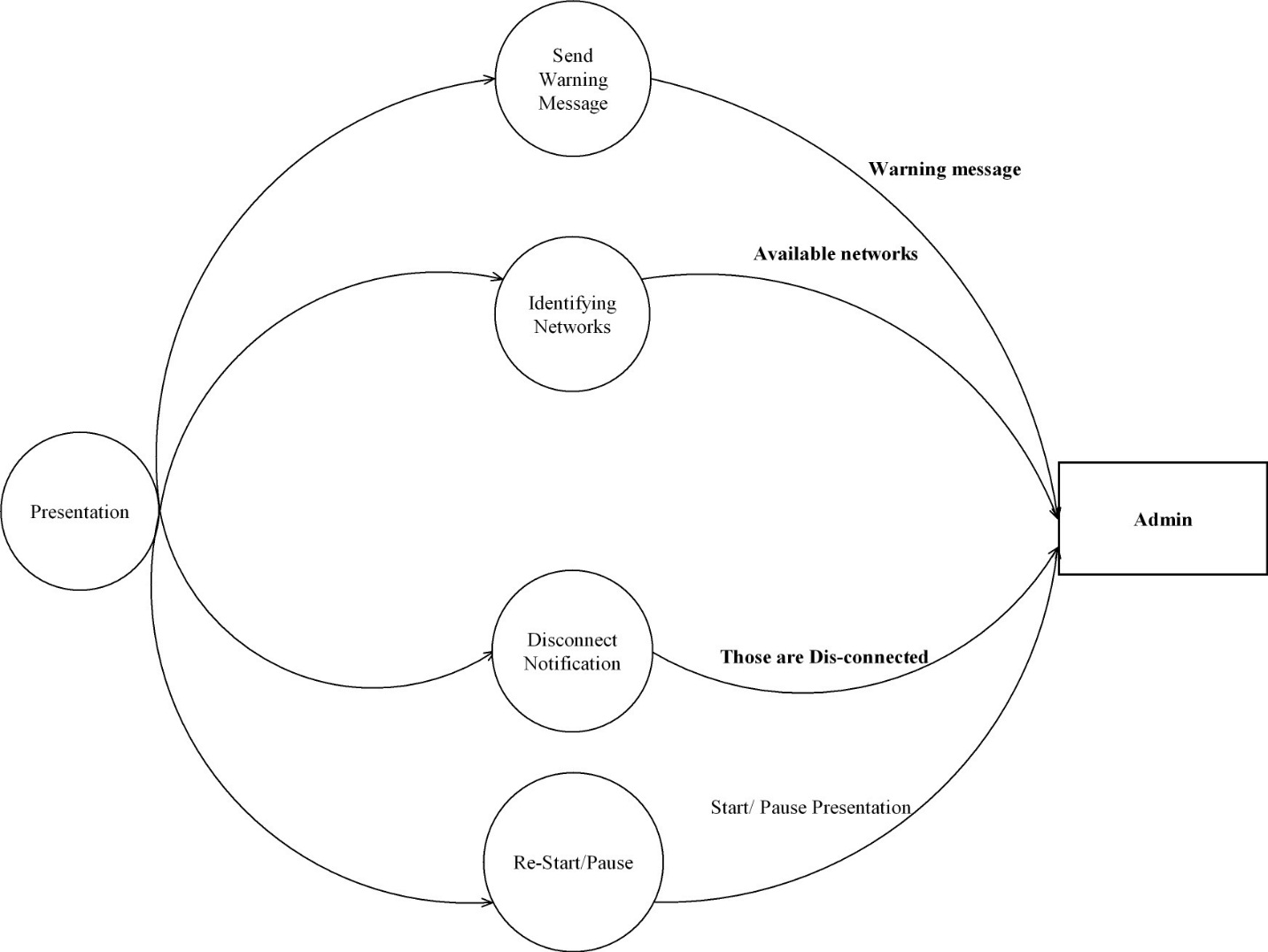
**Level - 1**

****

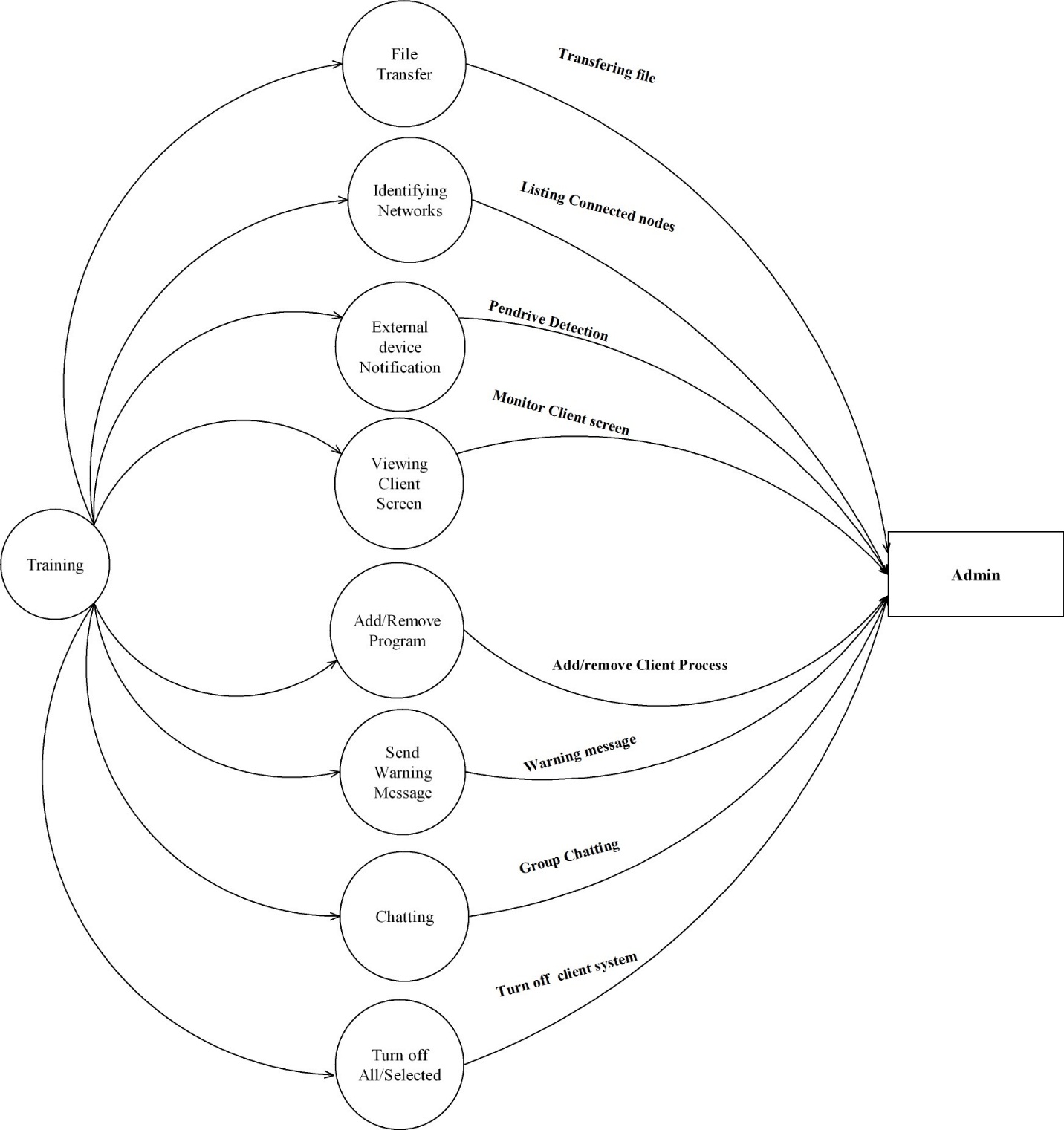
**Level – 2 (Admin)**

****

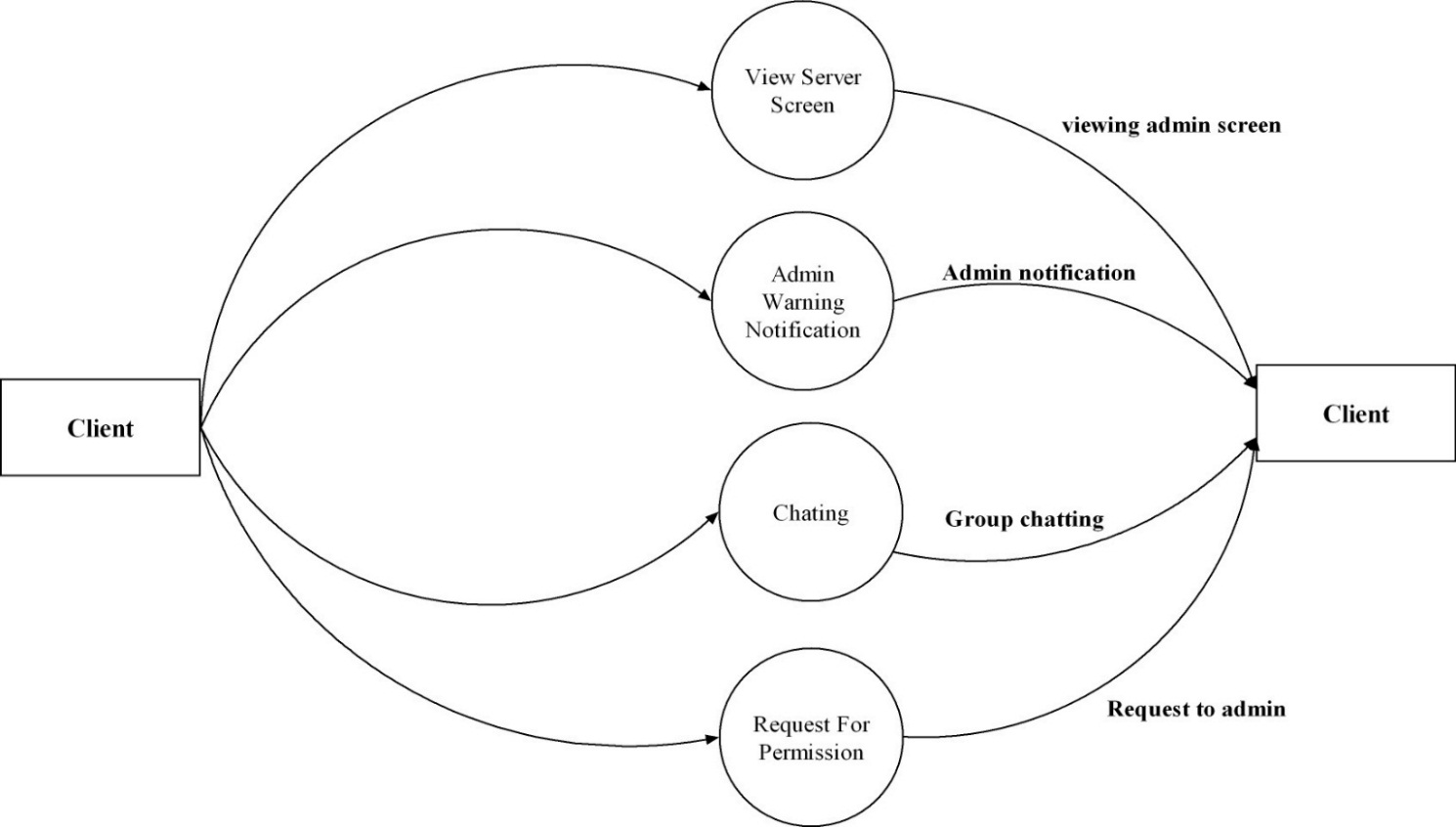
**Level – 2.1**

****

**Level – 2.2**

****

**Level – 3 (Client)**

****

**4.6 USE CASE DIAGRAM**

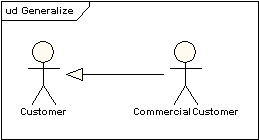
**Use Case Model**

The use case model captures the requirements of a system. Use cases are a means of communicating with users and other stakeholders what the system is intended to do.

**Actors**

A use case diagram shows the interaction between the system and entities external to the system. These external entities are referred to as actors. Actors represent roles which may include human users, external hardware or other systems. An actor is usually drawn as a named stick figure, or alternatively as a class rectangle with the «actor» keyword.

Actors can generalize other actors as detailed in the following diagram:



**Use Cases**

A use case is a single unit of meaningful work. It provides a high-level view of behavior observable to someone or something outside the system. The notation for a use case is an ellipse. The notation for using a use case is a connecting line with an optional arrowhead showing the direction of control.

**Use Case Definition**

A use case typically includes:

* Name and description
* Requirements
* Constraints
* Scenarios
* Scenario diagrams

**Name and Description**

A use case is normally named as a verb-phrase and given a brief informal textual description.

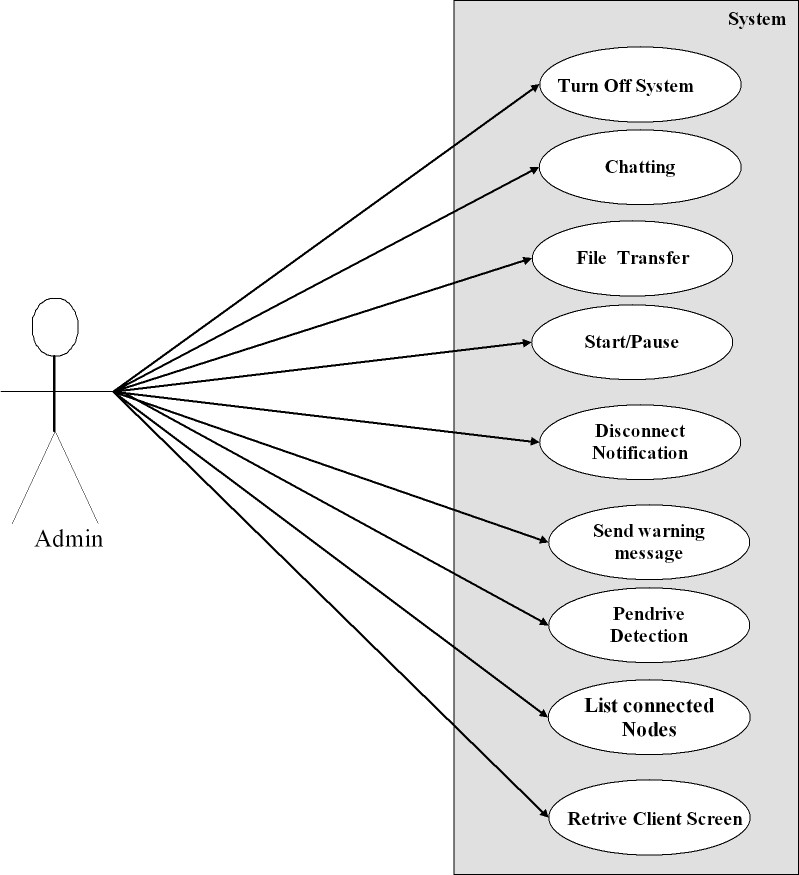
**Requirements**  
 The requirements define the formal functional requirements that a use case must supply to the end user. They correspond to the functional specifications found in structured methodologies. A requirement is a contract or promise that the use case will perform an action or provide some value to the system.

**Constraints**  
 A constraint is a condition or restriction that a use case operates under and includes pre-, post- and invariant conditions. A precondition specifies the conditions that need to be met before the use case can proceed. A post-condition is used to document the change in conditions that must be true after the execution of the use case. An invariant condition specifies the conditions that are true throughout the execution of the use case.

**Scenarios**  
 A Scenario is a formal description of the flow of events that occur during the execution of a use case instance. It defines the specific sequence of events between the system and the external actors. It is normally described in text and corresponds to the textual representation of the sequence diagram.

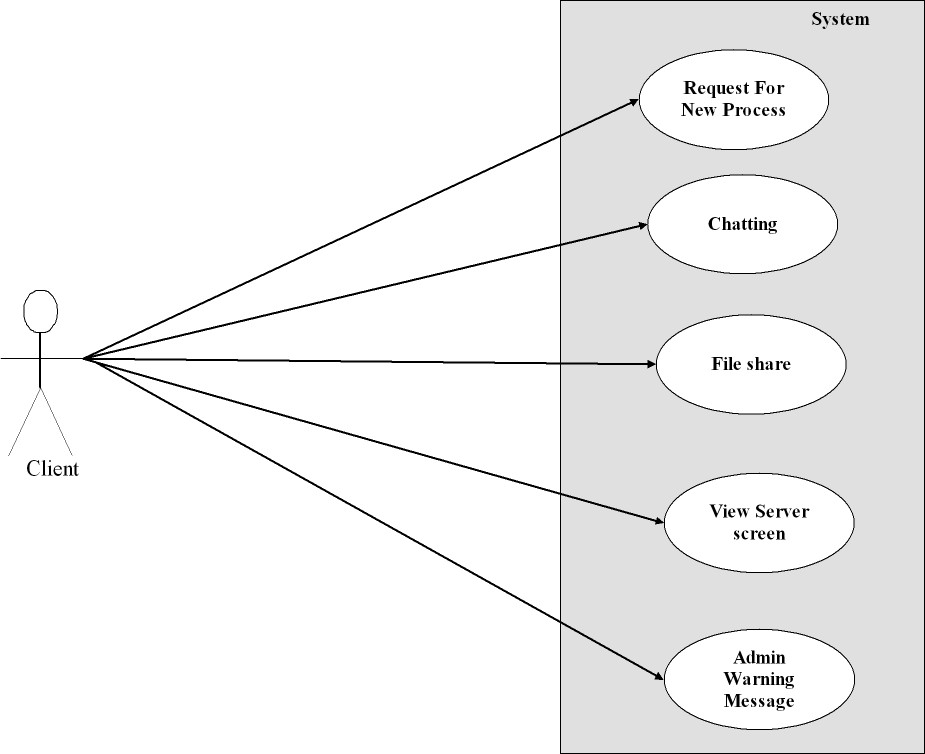
**Use Case Diagram 1:**

**Admin**



**Use Case Diagram 2:**

**Client**



**4.7 CLASS DIAGRAM**

The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.



A class with three sections.

In the diagram, classes are represented with boxes which contain three parts:

* The top part contains the name of the class. It is printed in Bold, centered and the first letter capitalized.
* The middle part contains the attributes of the class. They are left aligned and the first letter is lower case.
* The bottom part gives the methods or operations the class can take or undertake. They are also left aligned and the first letter is lower case.

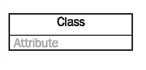
In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

So the purpose of the class diagram can be summarized as:

* Analysis and design of the static view of an application.
* Describe responsibilities of a system.
* Base for component and deployment diagrams.
* Forward and reverse engineering.

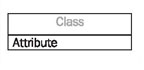
### **Class**

A class represents a relevant concept from the domain, a set of persons, objects, or ideas that are depicted in the IT system:



**Attribute**

An attribute of a class represents a characteristic of a class that is of interest for the user of the IT system:



Characteristics of interest of a passenger, for example, are name and age.

**Generalization**

Generalization is a relationship between two classes: a general class and a special class:

http://sourcemaking.com/files/sm/images/uml/img_127.jpg

**Association**

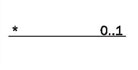
An association represents a relationship between two classes:

http://sourcemaking.com/files/sm/images/uml/img_128.jpg

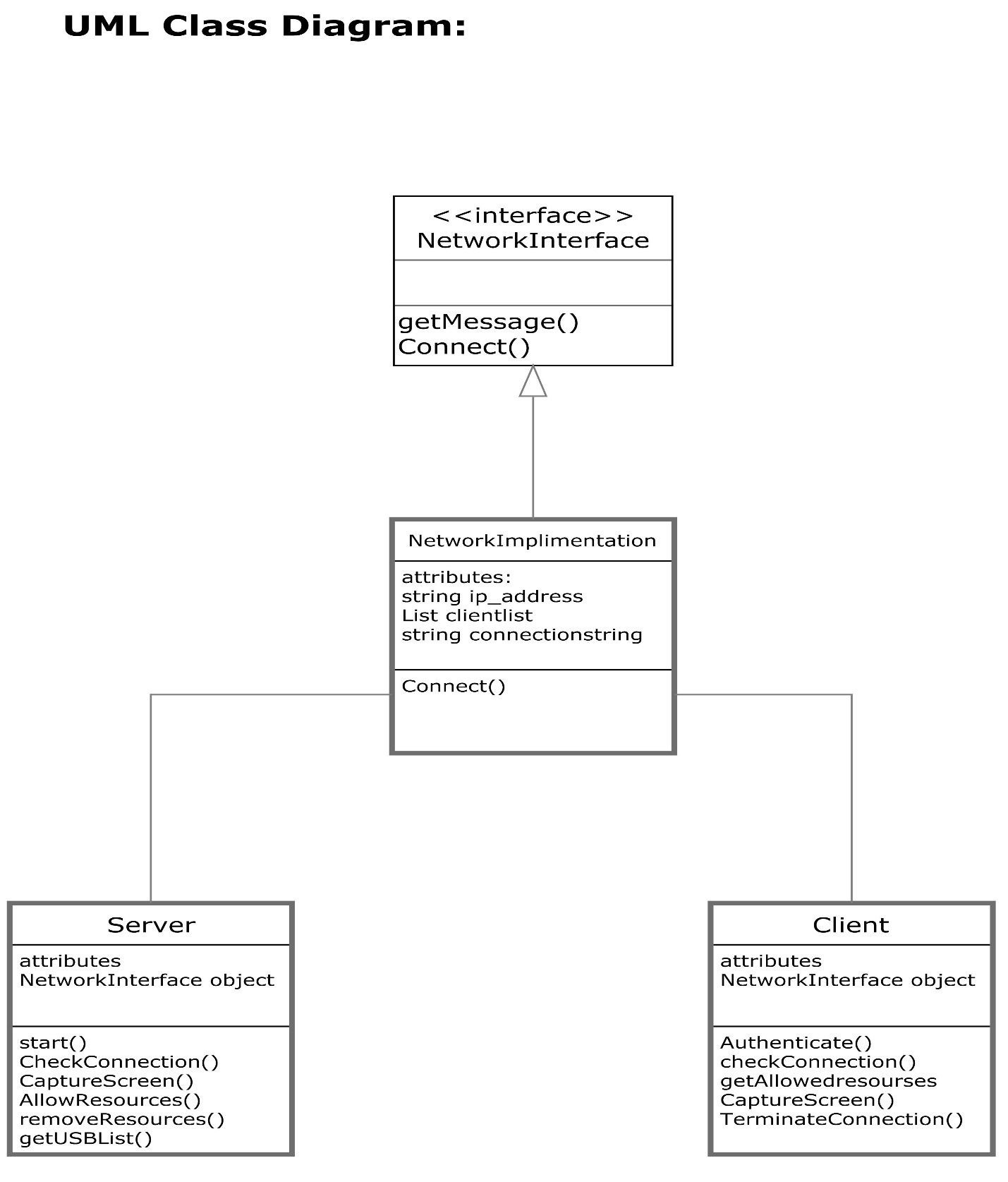
An association indicates that objects of one class have a relationship with objects of another class, in which this connection has a specifically defined meaning (for example, "is flown with").

**Multiplicity**

A multiplicity allows for statements about the number of objects that are involved in an association:



**Class Diagram**

****

**5. SYSTEM TESTING**

**AND**

**IMPLEMENTATION**

**5. SYSTEM TESTING**

**5.1 FORMAL TECHNICAL REVIEW**

Testing represents an interesting anomaly for the software. During earlier definition and development phases, it was attempted to build software from abstract concepts to tangible implementation. The testing is responsible to ensure that the product that has built performs the way that the detailed design documentation specifies.

**5.2 TEST PLAN**

Software testing is a critical element of quality assurance and represents the ultimate previews of specifications, design and coding. The testing phase involves the testing of the developed system using various test data. After preparing the test data, the system under study is tested using those test data. While testing the system by using test data, errors were found and corrected. Thus a series of tests were performed for the proposed system.

After a program has been coded, it is tested. The testing process is a part of the design and implementation phase. The primary goal of testing process is to make sure that the application performs as described in the requirements and specification in the documents. Before the system was ready for implementation various types of testing are done on the system. They are.

* Unit testing
* Integration testing
* Validation testing
* User acceptance testing
* Black box testing
* White box testing.

**5.2.1 UNIT TESTING**

A unit testing focuses verification effort on the smallest limit of software design. Using the unit test plan prepares in the design phase of the system, important control paths are tested to uncover the errors within the module. This testing was carried out during the coding itself. In the testing step, each module is going to be working satisfactory as the expected output from module.

**5.2.2 INTEGRATION TESTING**

Integration testing is the systematic technique for constructing the program structure while at the same time conducting test to uncover errors associated with. The objective is to take unit tested modules and build the program structure that has been dictated by design. All modules are combined in this testing step. Then the entire program is tested as a whole. If a set of errors is encountered, correction is difficult because the isolation of causes is complicated of vastness of entire program. Using integrated test plans prepared in the design phase of the system developed as guide, the integration was carried out. All errors found in the system were corrected for the next validation testing.

**5.2.3 VALIDATION TESTING**

At the end of program testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected and final series of software validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in a manner that can be reasonably accepted by the user/customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirements.

**5.2.4 USER ACCEPTANCE TESTING**

User acceptance of a system is a key factor to success of any system. The system under consideration was tested for user acceptance by constantly keeping in touch with the prospective system user at the time of developing and making changes wherever required. This is done with regard to the following points.

* Input screen design.
* Output screen design.
* Format of the reports and other output.

**5.2.5 BLACK BOX TESTING**

Knowing the specified function that a product has been designed to perform, test can be conducted that each function is fully operational. Black box test are carried out to test that input to a function is properly accepted and output is correctly produced. Specific knowledge of the application code structure and programming knowledge in general is not required. The tester is only aware of what the software is supposed to do, but not how i.e., what the application is supposed to do. It uses external description of the software including specification, requirements, and design to derive test cases.

**5.2.6 WHITE BOX TESTING**

White box testing of software is predicted on a close examination of procedural details. The status of the program may test at various points to determine whether the expected or asserted status is corresponding to the actual status. In White-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses input to exercise paths through the code and determine the appropriate outputs.

**5.3 SYSTEM IMPLEMENTATION**

System Implementation is used to bring a developed system or sub system into operational use and turning it over to the user. It involves programmer, users and operational management. It also needs to introduce and train the people to work with a new system. In the implementation of any new system, it is necessary to ensure that the consumer community is best positioned to utilize the system once deployment efforts have been validated. Therefore, all necessary training activities must be scheduled and coordinated. As this training often the first exposure to the system for many individuals, it should be conducted as professionally and competently as possible. A positive training experience is a great first step towards customer acceptance of the system. During system implementation it is essential that everyone involved be absolutely synchronized with the deployment plan and with each other. Often the performance of deployment efforts impact many of the performing organization normal business operations. Examples of these impacts include:

* Consumer may experience a period of time in which the system that they depend on to perform their jobs are temporarily unavailable to them. They may be asked to maintain detailed manual records or logs of business functions that they perform to be entered into the new system once it is operational.
* Technical Services personal may be required to assume significant implementation responsibilities while at the time having to continue current levels of service on other critical business systems.
* Technical Support personnel may experience unusually high volumes of requests due to the possible disruption of day-to-day processing.

Because of these and other impacts, the communication of planned deployment activities to all parties involved in the project is critical. A smooth deployment requires strong leadership, planning, and communications. By this point in the project lifecycle, the team will have spent countless hours devising and refining the steps to be followed. During this preparation process the Project Manager must verify that all conditions that must be met prior to initiating deployment activities have been met, and the final ‘green light’ is on for the team to proceed.

When the Manager’s system is linked to terminals on remote sites, the telecommunication network and tests of the network along with the system are also included under implementation. Depending upon the nature of the system, extensive user training may be required. Programming itself is a design work. The initial parameters of the management information system should be modified as a result of programming efforts; programming provides a Reality test for the assumptions made by the analyst. System testing check the readiness and accuracy of the system access update and retrieve data from new files. Once the program becomes available, the test data are read into the computer and processed. In this system, conventional Parallel Run was conducted to establish the efficiency of the system. Implementation is used here to mean the process of converting a new or a revised system design into an operational one. Conversion is one aspect of Implementation. Conversion means changing from one system to another. The objective is to put the tested system into operation while holding costs, risks and personal irritation to a minimum.

Changeover is the process of adopting the new system. The new system has to be introduced however. This is done after the system has been developed and tested completely. There is a set of methods like Direct Changeover, Parallel Changeover, Pilot running etc. Pilot running is intended here. Data from one or more previous periods for the whole or part of the system is run on the new system after results have been obtained from the old system and both are compared. It is performed till the completion of one system life cycle. When the changeover has taken place there will be a need for amendment to correct or improve the new system. When the user wants to add any new records, some fields will automatically get their default values. If the user desires to change these default values he can do it.

**6. SYSTEM MAINTENANCE**

**6. SYSTEM MAINTENANCE**

Software maintenance denotes any changes made to the software product after it has been delivered to the customer. Maintenance is inevitable for almost any kind of products. However, most products need maintenance due to the wear and tear caused by use. On the other hand software products do not need maintenance on this account, but need maintenance to correct errors, enhance features, port to new platform etc.

The requirements of Software maintenance arises on account of three main reasons:

**1. Corrective:** Corrective Maintenance of a software product becomes necessary to rectify the bugs observed while the system is in use.

**2. Adaptive:** A software product might need maintenance when the customers need the product to run on new platforms, new operating systems, or when they need the product to be interfaced with new hardware or software.

**3. Perfective:** A software product need maintenance to support the new features that uses want to support, to change different functionalities of the system according to customer demands , or to enhance the performance of the system.

**7. CONCLUSION**

**7. CONCLUSION**

The project “**T-Shell**” has been successfully completed. In the present generation systems, there is a need for the administrator has to go all around the network in order to terminate any unauthorized user activity. In order to get the system configuration details of any particular system, the administrator has to take the trouble of going to that system for obtaining the information. So, by using this project we can avoid all the above problems. This project is very useful in the class presentation & Training session or corporate field for monitoring the Students/Employee’s activities. This software has been developed as per the requirements, specifications and conditions analyzed and so the system can definitely be implemented in the System. It is easy to access since it uses the GUI with user friendly screens and dialogues. The usage of the software increases the efficiency and decreases the manual effort. It has been thoroughly tested and implemented.

**8. FUTURE ENHANCEMENT**

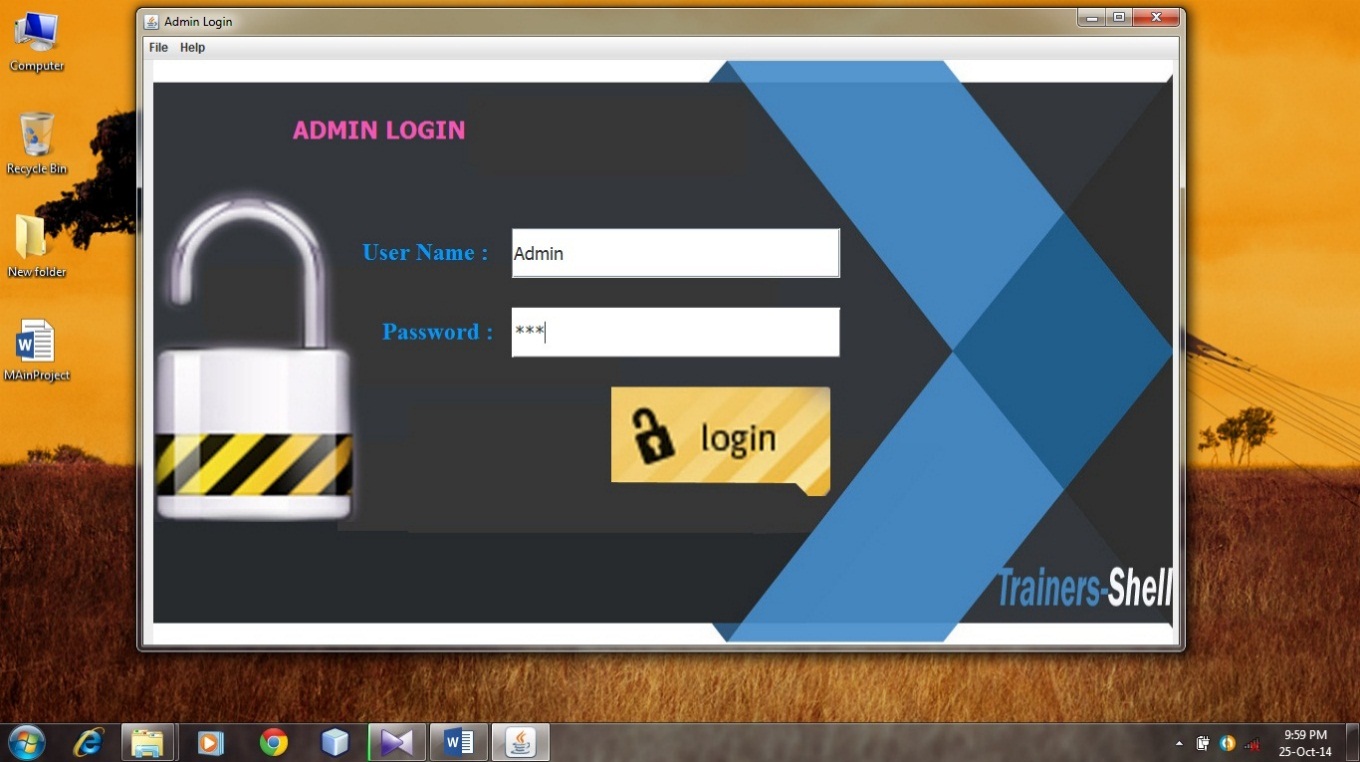
**8. FUTURE ENHANCEMENT**

In this project it is unable to scan any malware and virus in client system and notify it to the server system. Further we want to implement this concept that if any malware or virus found in client systems then a message will be sent to the server system as a notification along with brief details and provide the option to live voice and video Conferencing to communicate with each other during the training session.

Also we want to introduce a concept that, this software working in LAN network & wiFi network a Further we want to implement this in Global network

**9. SCREEN SHOTS**

**Admin Login**

****

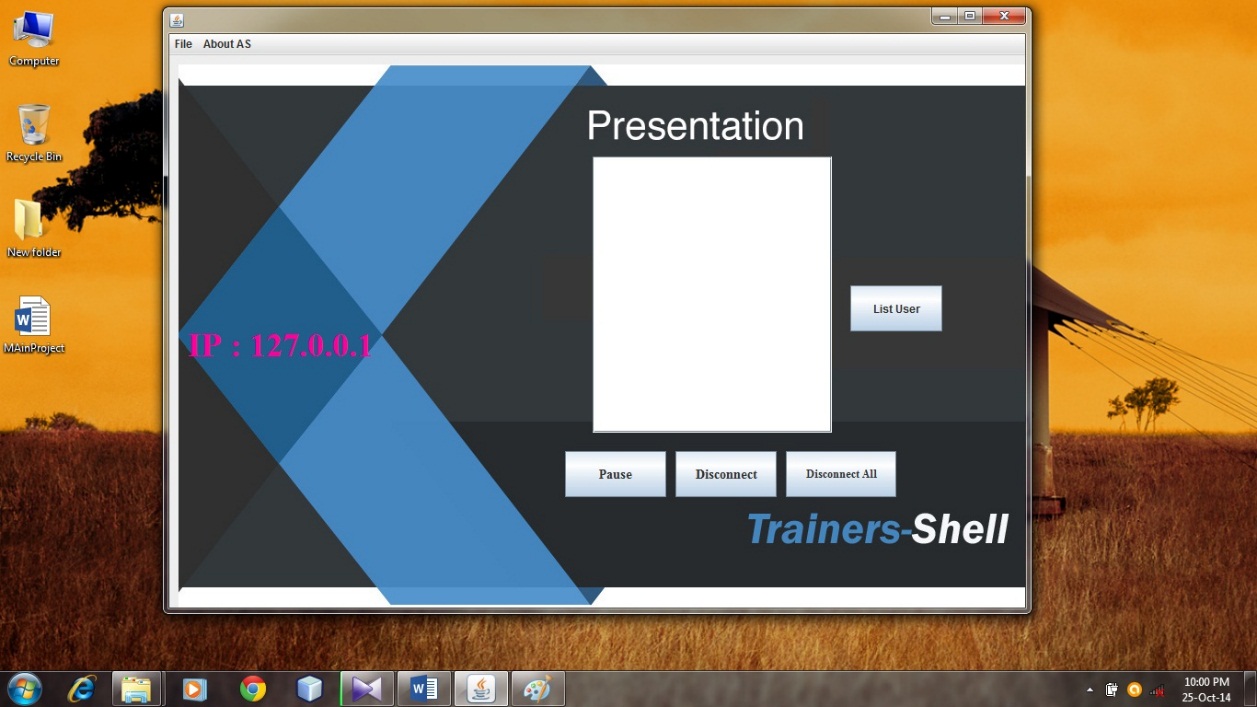
**Description**: This is the start page for the server .The server IP generated using server IP which the clients can connect to the server.

**Admin Home**

****

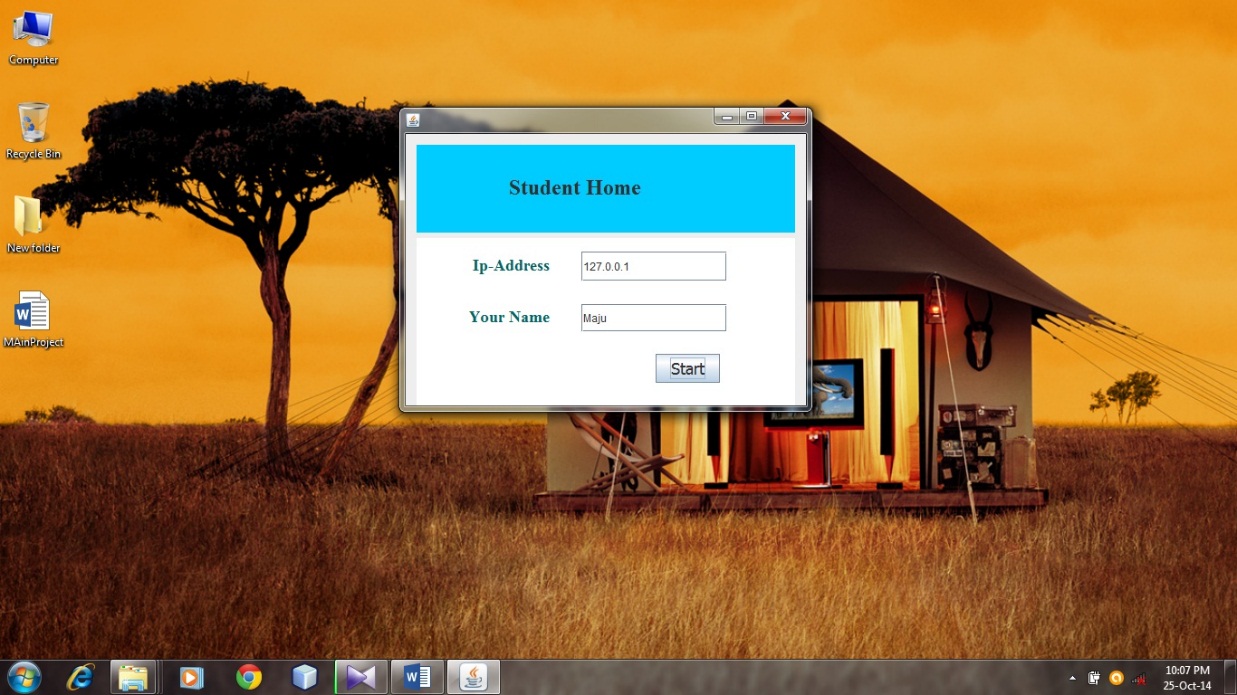
**Description**: The server can choose the mode of operation. This control button allows the server to control a client machine

**Admin (Presentation Session)**

****

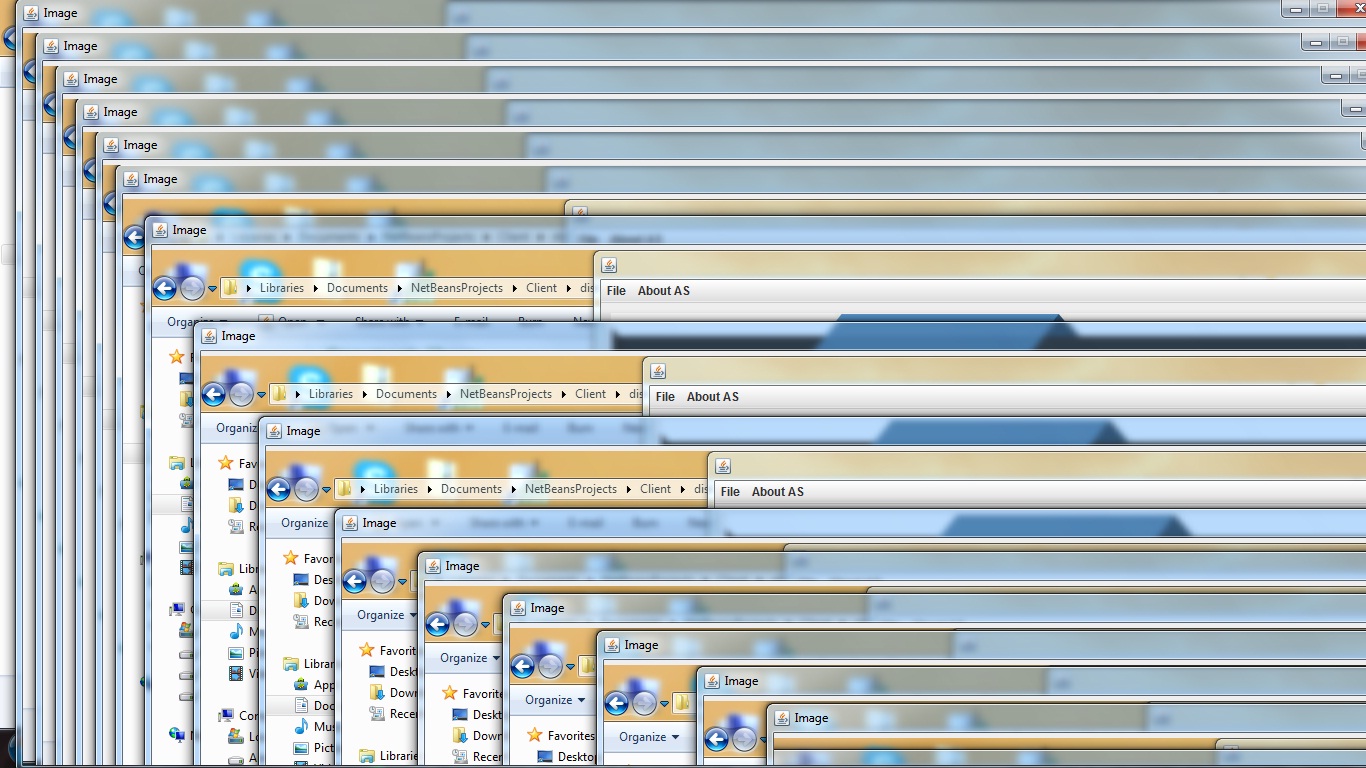
**Description**: This page allows the server to start or stop a presentation. When the start button is pressed the servers screen is shared to all the clients. The server can close the presentation by choosing the Window also server can disconnect particular client by choosing IP from list then click Disconnect button.

**Client Login**

****

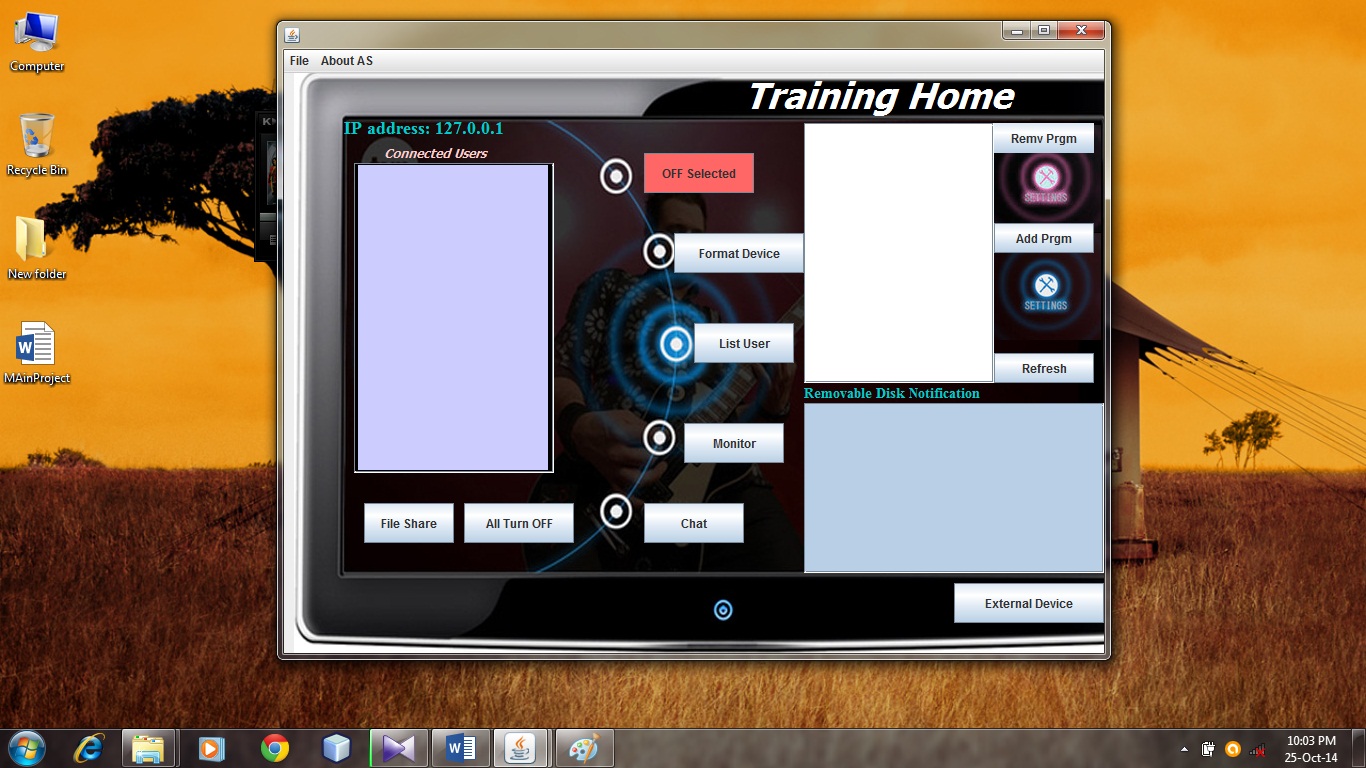
**Description**: This page shows client’s start page. The client can enter the IP address and name to get connected to the server.

**Client Getting Server screen (Presentation Session)**

****

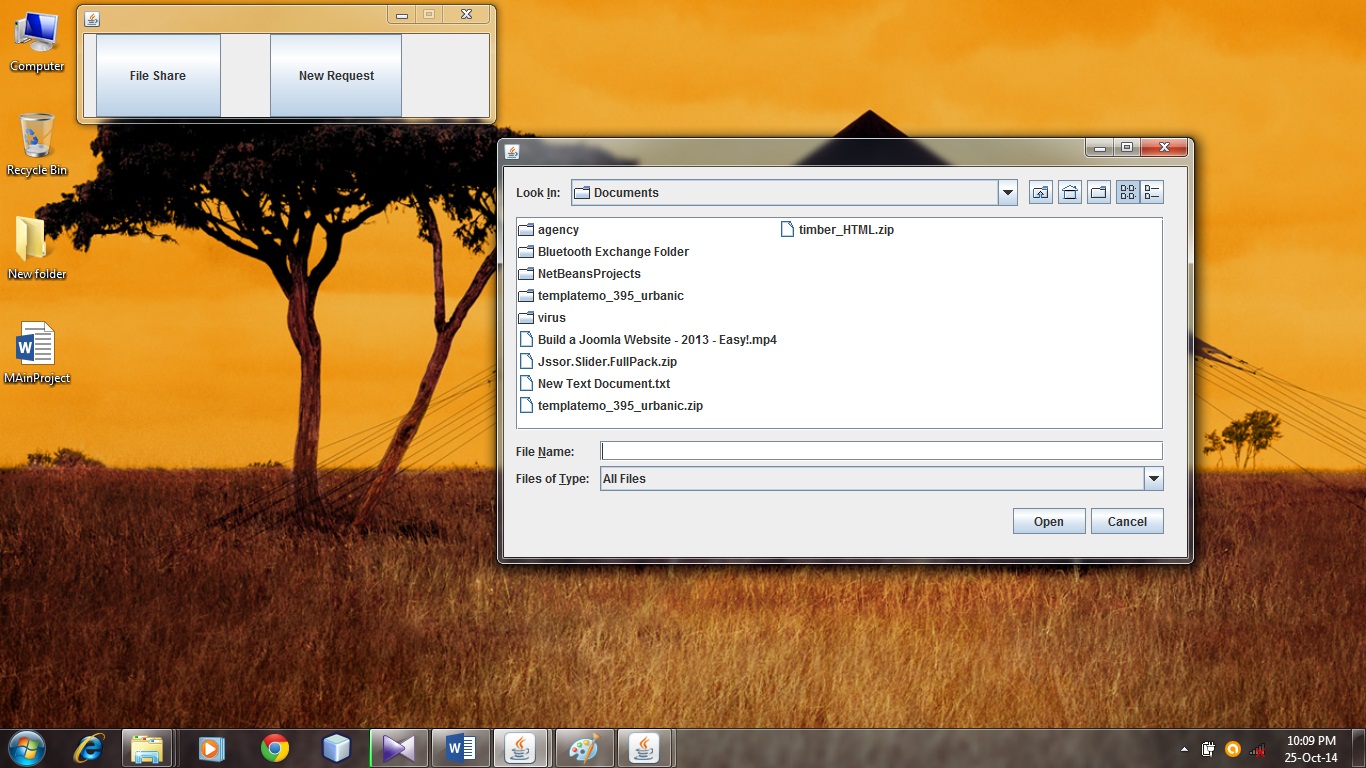
**Description**: This page shows client’s page. The client can only see server screen during the presentation section client cannot do any other activity during this section.

**Admin (Training Session)**

****

**Description**: This page shows that the allowed resource has been removed by the server. Thus, the clients will no longer be able to use the resource, server can transfer file to particular client.

**Client Share File (Training Session)**

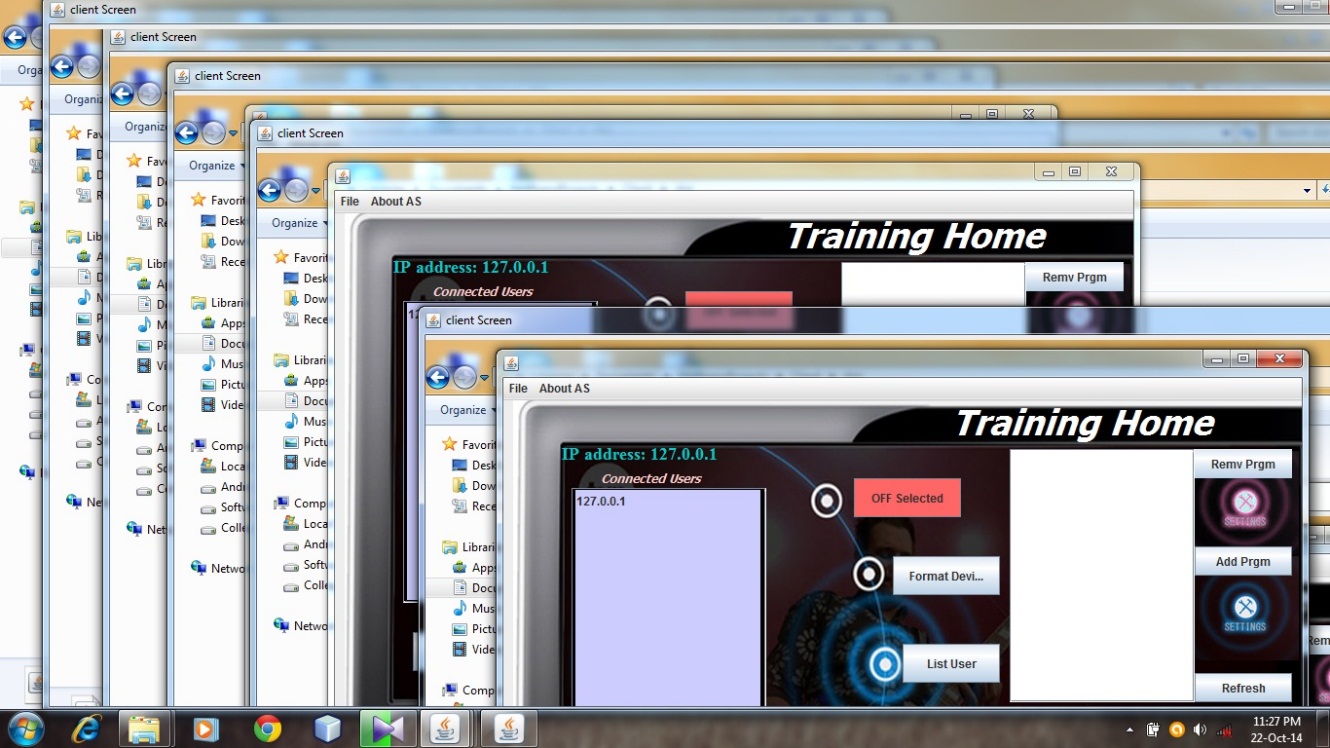
****

**Description**: This page shows that the client can transfer files to server.

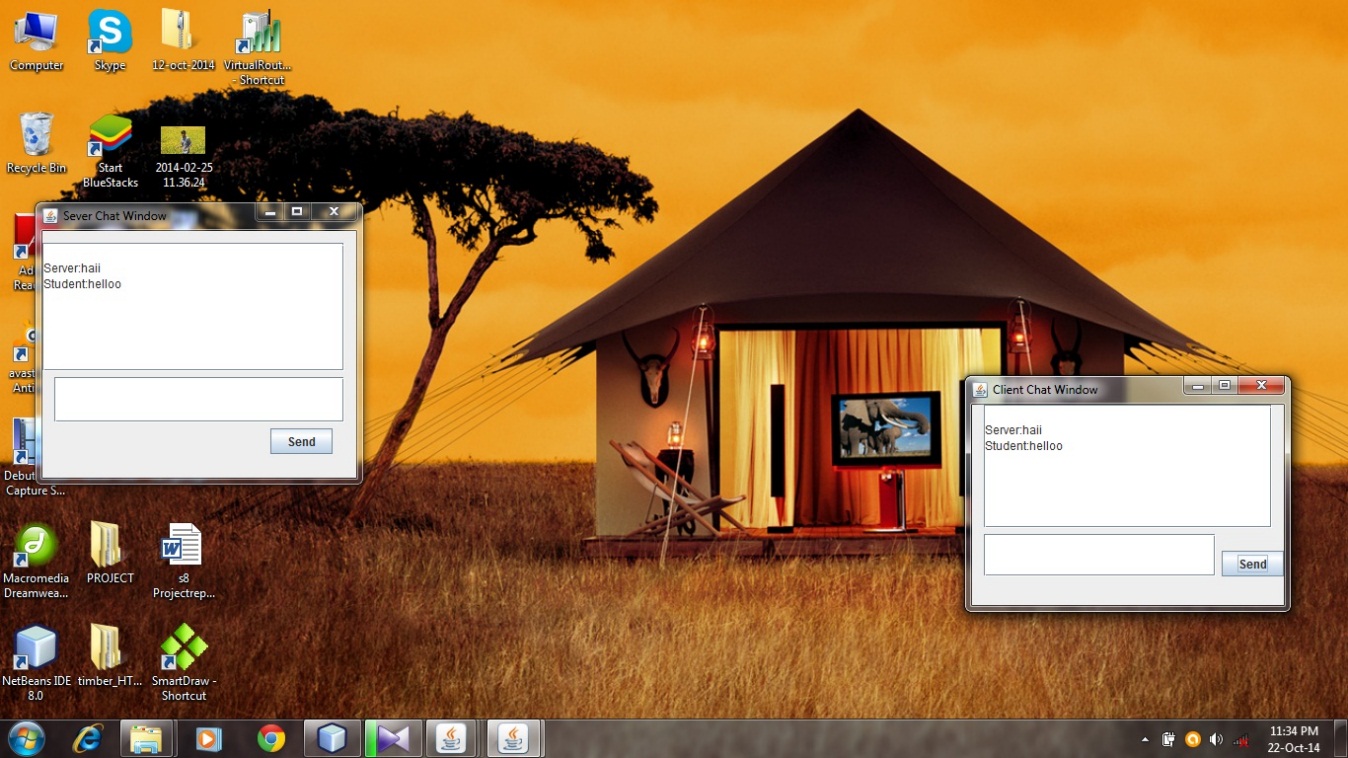
**Client requesting new Process (Training Session)**

**Description**: This page shows that the client has requested for service and waiting for server permission for use that resources.

**Client system Monitoring**

****

**Description**: This page shows that the client’s desktop has been captured on the servers system. The server can control the client’s system.

**Group Chatting**

**Description**: This page shows that the client’s and server can chat each other

**10. BIBLIOGRAPHY**

**10. BIBLIOGRAPHY**

* Advanced Programming in Java 2: Updated to J2SE(Dr.K.Somasundaram)
* [www.javatpoint.com/**RMI**](http://www.javatpoint.com/RMI)
* Java The Complete Reference 8th Edition(Herbert Schildt)
* <http://en.wikipedia.org/wiki/Java_remote_method_invocation>
* Nieh, J., S. J. Yang, and N. Novik, “A comparison of thin client computing architectures,” tech. rep., Network Computing Laboratory, Columbia University, November 2000.
* Yang, S. J. , J. Nieh, and N. Novik, “Measuring Thin-Client performance using Slow-Motion benchmarking,” Transactions on Computer Systems, vol. 21, no. 1, pp. 87–115, 2003.