Special Degree in Computer Science
4th Year - Project

VIRTUAL INTERNET COMPUTER

By

Tilan Niranjan Ukwatta

Index Number: S5290
Registration Number: 96/S/6405

December 2001

This dissertation is submitted in partial fulfillment of the requirements for the special degree in computer science.

Department of Computer Science
University of Colombo, Sri Lanka

DECLARATION

I hereby certify that this dissertation titled "Virtual Internet Computer" is entirely my own work. It has never been submitted nor is currently being submitted for any other degree.

……
Candidate
(T. N. Ukwatta)

……..
Supervisor
(Dr. Nalin Ranasinghe)

……..
Date

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to give my sincere gratitude to Dr. Nalin Ranasinghe for his invaluable guidance and help to achieve this success.

Also I would like to give my heartfelt thanks to Dr. N. Kodikara and Mr. Rukman Senanayka who initially read my proposal and gave guidance.

And lot of thanks goes to our network administrators Mr. Rasika Dayarathena and Mr. Kamal Periyapperuma, who sacrifice their invaluable time for us.

And I would like to thank all my friends who were always with me during the project specially Thusara and Kumudu.

Finally I would like to thank my parents and my sisters for their love, patience and understanding.

Tilan Niranjan Ukwatta

GOAL

To create a computer on the Internet, which can be accessed independent of client machine architecture, operating system and physical location.

ABSTRACT

Use of computer networks grew at phenomenal rate after the introduction of the World Wide Web. In this project we will introduce a concept, which might change the way people use computer networks in the future.

This new concept is called Virtual Internet Computer. Virtual Internet Computer is a virtual computer, which runs on a computing system connected to the Internet. Users can use it independent of the client device, location and operating system.

In this project a proof of concept implementation is done using Java. Virtual Internet Computer is implemented as an application running on a dedicated Java virtual machine. This application behaves like a simple operating system, which enables uses to do file management, execute various files and open documents. Operation of it is controlled by various interfaces. These interfaces include a Java client application, Java client applet, html based client and WAP client. The WAP client enable users to control their Virtual Internet Computer using a WAP enable mobile phone.

Chapter 01 *INTRODUCTION*

1.0 INTRODUCTION

1.1 Motivation

Most of the new discoveries, ideas, and concepts in the world came to existence because of some mistake made by a scientist taken in to good use as in the case with penicillin; Or because of clever observation of events in normal life as in the case with invention of steam engine or when someone trying to find a solution to some problem.

The idea of Virtual Internet Computer came to being as a result of an experience that the author had. One-day author has been working on a computer in the university computer lab he unconsciously tried to open a file, which was on the desktop of his home computer. After few seconds he realized that he was working on the university computer and the file that he looking for was in his home computer. At that time he thought that it would be nice to have one common computer, which can be accessed from anywhere, regardless of the location, type of the machined used and types of the operating system. This incident led to the concept of Virtual Internet Computer.

This project introduces a concept called Virtual Internet Computer, which is a centralized virtual computer on the Internet that can be accessed from anywhere in the world. In this report from this point onwards, to refer to the Virtual Internet Computer we will use VIC and Virtual Internet Computer interchangeably.

1.2 Problems to solve, Aims and Objectives

- To enable users to access their virtual computer from anywhere in the world.
- To enable users to access their computer using many forms of computing devices such as PCs, Laptops, PDAs, mobile phones … etc.
- To provide computing power to users that is not possible with single CPU systems.
- To make the Virtual Internet Computer independent of network bandwidth.
- To eliminate more than one desktop per user.
- To make it possible to access the computer using client devices which is small, cheap and portable.
- To provide different kind of interfaces to access the virtual computers such as Java interface, Web interface and Wap interface.

- To enable users of virtual computers to run their computers on the Internet on behalf of them and do various tasks when users are offline.
- To release users from constant need to upgrade and maintain their hardware.

1.3 Overview of VIC

VIC is a virtual computer, which runs on a computing system connected to the Internet. This computing system can be a single high-end computer, multi processor super computer or a computer network running some kind of distributed system. Users can use their VIC independent of the client device, location and operating system. Figure 1.1 illustrates this situation.

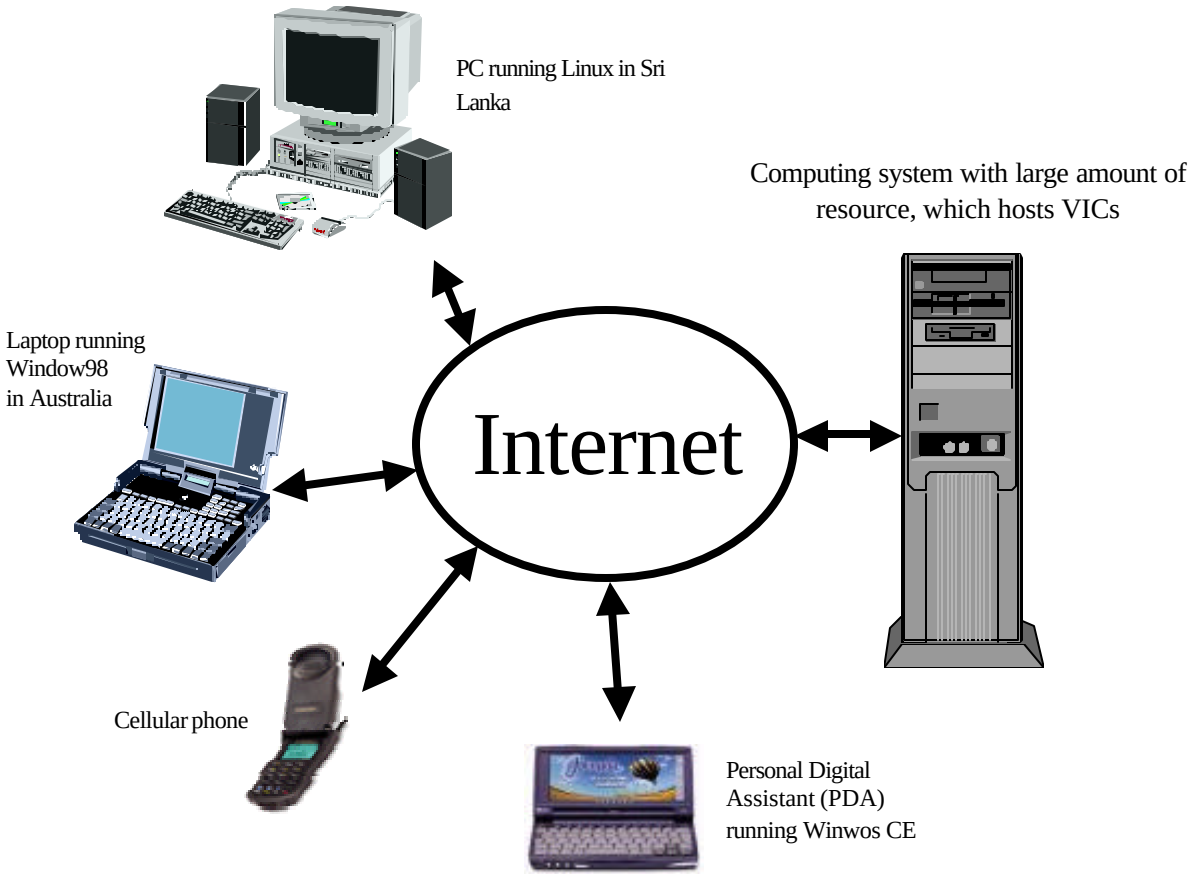


Figure 1.1

VIC provides a facility to store document files on the Internet and opens it using an application on the same location. Users can access it independent of the client machine architecture and operating system. Also to access VICs, users do not need lot of network bandwidth. Resource requirements of client devices that are used to access VICs are similar to thin client[I] devices.

[I] Thin client devices are described in chapter 2.

Using the Virtual Internet Computer concept attempt is made to achieve not only above capabilities but to get almost all the capabilities of a today's desktop computer, to the VIC and more.

1.4 Advantages and Disadvantages of VIC

1.4.1 Computing Resources

Processing speed of a computer doubles every 18 months. This is the case in the past. But it is not practical to believe this will continue forever. There are physical limits that control the growth of the computer industry such as speed of the light, the upper limit of how much transistors can be packed inside a silicon wafer of a given size and heat dissipation problems **[Tanen2000a]**. It is logical to conclude that in the future the amount of resource a single CPU system can provide will get to a maximum. In this context multiprocessor or multi computer systems will have to play a major role in providing computing resources in the future.

VIC utilizes both multiprocessor and multi computer systems by creating a virtual computer, which runs on top of these systems. Since network of computers can provided much larger amount of resources than a single computer, VIC running on a distributed system can provide computing resources that were not possible with single CPU systems. Also in the case where the VIC is situated in a super computer, users get more computing power than single processor systems.

1.4.2 Virtually Centralized Data and Applications

VIC is a virtually centralized computing system with all data and applications are in same virtual location. Users of the VIC sees that their data and applications are in the same place but physically data may be distributed. This centralized nature and the ability to access from anywhere in the world will eliminate the need for carrying storage media such as disk from place to place to carry data.

1.4.3 Location Independence

Internet is a global computer network, which is growing very rapidly. Since VICs reside on the Internet users can access it from anywhere in the world. This location independence is very important advantage of the VIC.

1.4.4 Client Devices

All the processing and storage of VICs is done in the computing system on the Internet. Because of that client devices do not have to do lot of processing and also those devices do not need to store user data. This small resource requirement of the client devices allows designing very small, lighting weight and cheap client devices to access VICs. We can also use mobile phones without lot of storage and processing power to access and operate VICs.

1.4.5 Mobile Working Environment

The today's trend of work environment is allowing more and more employees to work away from office. This means that a wide variety of network connections are being used to access corporate applications, documents and

databases. In this kind of environment network bandwidth and client device is an important issue. VIC facilitates this kind of work environment by allowing each employee to have a VIC, which can be accessed from anywhere, using wide variety of devices and network connections.

1.4.6 Computer Maintenance

Conventional PC users had to deal with the problem of constant needs to upgrade their hardware and software. Also the hardware failure rate of PCs is considerably high. Non-technical people find this situation very difficult to cope with. Advantage of VIC is that users do not need do much in computer maintenance, especially in hardware maintenance.

1.4.7 Off Line Work

We can keep our Virtual Internet Computer running even when, we are not logon. This gives us chance to accomplish lengthy downloads, complex animation rendering or scientific calculations off line. Some of these tasks may take day's to get completed.

1.4.8 Security

Another advantage of VIC is that it is safe from conventional thieves and environmental hazards such as power failures and lightning. Since client devices do not store anything, even when it is stolen there is no risk of data being seen by unwanted people.

But Virtual Internet Computers are vulnerable to Hacker attacks. As in most computer related situations security is a problem in Virtual Internet Computer also. We can use encryption methods and other standard computer security measures to prevent unauthorized use of Virtual Internet Computers.

1.5 Outcome of the project

Outcome of this project is the concept of Virtual Internet Computer and a proof of concept implementation. There are other possible ways to implement a VIC system but in this project Java based implementation is produced.

1.6 Structure of the report

Report is divided in to chapters each discuss particular aspect of the project. The remainder of this report is organized as follows: We will give a more detailed background information about computing models of computer networks in the second chapter. This chapter also gives an overview of technologies used in the project. Third chapter discusses the VIC concept giving the general architecture of the VIC system. In the fourth chapter design issuers of the VIC system implemented in this project are discussed. Chapter five relates to the implementation of the VIC system. Finally, we present in the final chapter further work that can be carried out in the future.

Chapter 02 *BACKGROUND*

2.0 BACKGROUND

2.1 Computing Models for Computer Networks

Computing models for computer networks describes or models how we use a computer network for computing. There is many kind of computing models for computer networks. For example if we consider Internet/intranet environment there is Download and Run model of Java applets or ActiveX components. In the next couple of sections we will discuss some existing network computing models and elaborate on how Virtual Internet Computer model differs.

2.1.1 Network Computing Model

In the Network Computing Model as defined by Sun, Oracle, Netscape, IBM and Apple, the client dynamically downloads components from the network into the client device for execution.

Here applications are downloaded to local machine from the network and get executed. Java applets and ActiveX components use this model. If we consider documents they also have to be downloaded and then open to obtain information in it.

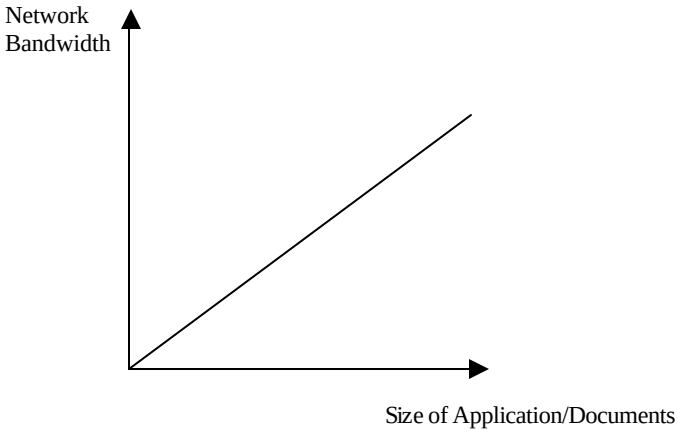


Figure 2.1

This model is not a bandwidth friendly. As shown in the figure 2.1 required network bandwidth increases with the size of applications and documents. Also the performance of applications depends on the hardware of the local machine.

Network Computing Model requires considerable amount of computing resources from the computing device, which is used to access the network. This limits the variety computing devices that can be used to access applications and documents.

2.1.2 X Windows Network Computing Model

In this model person in one machine uses other available machines in the network to do his jobs. Here one desktop shows interface screens of several different applications running on different machines on the network. This model is implemented in the X Windows System **[Tim96]**.

X windows is a hardware-independent network-based window system which was developed at MIT **[Rodger94]**. Any X window system consists of 2 distinct parts, the X server and 1 or more X clients. The server controls the display directly, and is responsible for all input/output via the keyboard, mouse or display. The clients, on the other hand, do not access the screen directly; they communicate with the server, which handles all input and output. It is the clients, which do the actual computing work of running applications. The clients communicate with the server, causing the server to open one or more windows to handle input and output for those clients.

The X server invariably runs on the machine to which the monitor is connected. The clients may also run on this machine, communicating directly with the server. On most workstations, this is the normal situation. However, X is a networked window system, and it is possible for the client to run on a remote machine, communicating with the server via some form of network. In most cases, this would be via TCP-IP over an Ethernet link.

The main advantage of utilizing a client on a remote machine is that the client may run on a much more powerful machine and the output is displayed on the local machine. Alternatively, a number of clients may be run on different machines, all sending the output to different windows on the same server.

2.1.3 Virtual Network Computing (VNC) Model

In this model one can control any machine on the network by special desktop viewer. Here the viewer shows the desktop of the remote machine, which runs the VNC server and we can control that machine as if we were at that location.

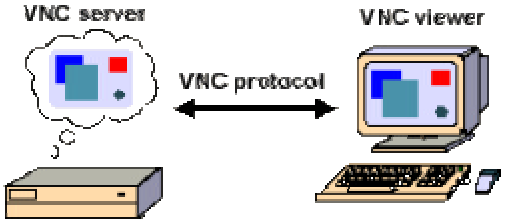


Figure 2.2 [II]

As shown in figure 2.2, VNC, in essence, is a remote display system, which allows user to view a computing 'desktop' environment not only on the machine where it is running, but also from anywhere on the Internet and from a wide variety of machine architectures **[Vnc2001]**.

VNC system is a product of research project conducted at AT&T Laboratories Cambridge. VNC system is implemented as two components. That is VNC

[II] Graphic courtesy of AT&T Laboratories.

server, which runs on the remote machine and VNC viewer, which runs on the local machine. Most people use a VNC viewer running on a PC to display Unix environments, which are running on large servers. People at AT&T Laboratories have been able to write VNC viewers for many platforms but VNC server is available only for UNIX and WindowsNT/95 platforms.

Features of VNC system can be listed as follows.

?? No state is stored at the viewer. This means user can leave his desk, go to another machine, whether next door or several hundred miles away, reconnect to his desktop from there and finish the sentence he were typing. Even the cursor will be in the same place.

?? It is truly platform-independent. A desktop running on a Linux machine may be displayed on a PC or on a Macintosh machine.

?? It is sharable. One desktop can be displayed and used by several viewers at once.

The VNC Protocol

The VNC protocol is a simple open protocol for remote access to graphical user interfaces. It is based on the concept of a remote framebuffer or RFB. The protocol simply allows a server to update the framebuffer displayed on a viewer. Because it works at the framebuffer level it is potentially applicable to all operating systems, windowing systems and applications. This includes X/Unix, Windows 3.1/95/NT and Macintosh, but might also include PDAs, and any device with some form of communications link. The protocol will operate over any reliable transport such as TCP/IP.

VNC protocol has been designed to make very few requirements of the viewer. In this way, clients can run on the widest range of hardware, and the task of implementing a client is made as simple as possible.

The input side of the protocol is based on a standard workstation model of a keyboard and multi-button pointing device. Input events are sent to the server by the client whenever the user presses a key or pointer button, or whenever the pointing device is moved.

VNC Clients

VNC viewer is a simple system. It requires only a reliable transport (usually TCP/IP), and a way of displaying pixels (either directly writing to the framebuffer, or going through a windowing system). VNC viewers available as X-based client (which runs on Solaris, Linux and Digital Unix workstations), a Win32 client which runs on Windows NT and 95, a Macintosh client, and a Java client which runs on any Java-capable browser.

VNC Servers

VNC server is slightly complex than the client because of number of reasons. The protocol is designed to make the client as simple as possible, so it is up to the server to perform any necessary translations. For example, the server must provide pixel data in the format the client wants.

2.1.4 Server based Computing Model

Server based computing is a model, in which applications are deployed, managed, supported and executed 100% on a server **[Citrix99a]**. With server based computing, client devices have instant access to applications via the server, without application rewrites or download. This means improved efficiency when deploying applications. In addition, server based computing works with in the current computing infrastructure and current computing standards.

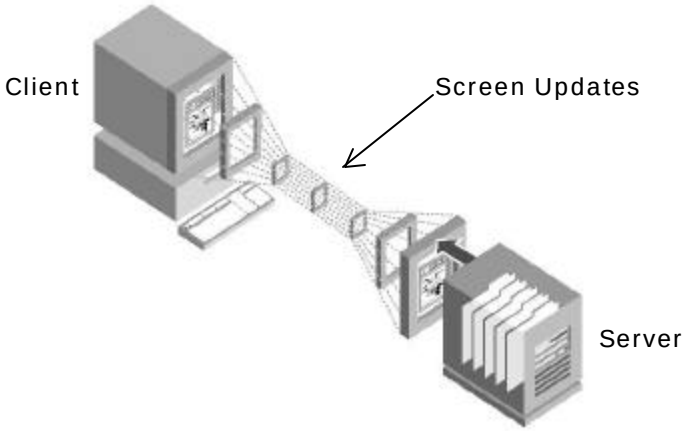


Figure 2.3 [III]

The server based computing model employs three critical components. The first is a multi-user operating system that enables multiple concurrent users to log on and run applications in separate, protected sessions on a single server. The second is a method that separates the application's logic from its user interface so only keystrokes, mouse clicks and screen updates travel the network as illustrated in figure 2.3. As a result application performance is bandwidth independent. The third key component is centralized application and client management.

Thin – Client Devices

A thin client is a desktop device connected over a network or serial connection to a central computer or server that is operating in multiuser mode **[Citrix99b]**. While the desktop device may look like a PC, it is actually much simpler, if equally sophisticated device. All application processing and storage is done on the server. The thin client is simply a device for input, output and display. Thin client is used only to transmit keystrokes and mouse clicks to the server, and to display on a desktop monitor what the server is doing. With thin-client/server computing, the desktop device is only a window in to the activity on the server, allowing it to be a simpler device. It does not require a hard drive, a floppy drive, or the latest CPU, and the thin-client chip only has to drive the display, and transfer input/output bit streams. Thin client device uses less power than a PC, and because it has no moving parts, less RAM, and a CPU that generate less heat, it does not require a fan. It has, therefore, no moving parts that can break.

[III] Graphic courtesy of Citrix.

Contrast this with a PC, which combines the computing and data storage functions of the server with the display, input and output capabilities of a thin client, in one desktop machine. While a PC may be used in thin -client/server computing, only true thin clients are optimized for use in this server based computing environment.

Windows Based Terminals

Windows based terminal is a thin client hardware device that connects to server based system software. Windows based terminals are the graphic version of the old text based terminals. Because the graphical applications that it accesses are installed on the server, a Windows based terminal is not the equivalent of a PC with its operating system and array of local applications. Nor it is interchangeable with a network computer, because these devices download and run applications off the network.

Companies like Citrix (www.citrix.com), NCD (www.ncd.com) and IBM are world leaders in providing Windows Based Terminals.

Figure 2.4 [IV]

Figure 2.4 shows an IMB NetVista Windows Based Terminal.

2.1.5 Virtual Internet Computer Model

In the X Windows Network Computing Model clients serve for the server. Here clients do processing and send screen shots to the server and accept keyboard inputs and mouse events from the server. X Windows Network Computing Model is somewhat opposite of Virtual Internet Computer model, which clients do not do application processing. But the method of communication between the server and client in both models are similar.

VNC Model only allows users to use and administer remote computers. Here given computer can be used exclusively by only one person at a given time. But in the Virtual Internet Computer model one powerful computing system can host lot of virtual computers and serve many people at once. This powerful computing system can be a cluster of computers or a supercomputer running any operating system. The Virtual Internet Computer system can use the technique used in the VNC system to display desktop of the VIC on the clients' machine.

[IV] Graphic courtesy of IBM.

Server based computing model gives us good application performance but it is a modern version of the old mainframe based centralized computing concept. VIC model is somewhat similar to server based computing model, but in VIC model each user gets his own virtual computer, which is like a Virtual PC.

The Virtual Internet Computer model is more or less different from other computing models for computer networks. It is the concept of a virtual computer, which sits on the Internet, to be accessed by its owner from anywhere in the world using any computing device, running any operating system, makes it different from others. Users can run multiple programs in this virtual computer and do whatever tasks they do in normal computers. Physical location of the this virtual computer (Virtual Internet Computer) is not important at all as long as users have a fairly good connection to the Internet.

2.2 Technological Overview

2.2.1 Computer Network Challenges and Java

One challenge presented by the increasingly network centric hardware environment is the wide range of devices that networks interconnect. A typical network usually has many different kinds of attached devices, with diverse hardware architectures, operating systems, and purposes. Java addresses this challenge by enabling the creation of platform-independent programs. A single Java program can run unchanged on a wide range of computers and devices. Compared with programs compiled for a specific hardware and operating system, platform-independent programs written in Java can be easier and cheaper to develop, administer, and maintain **[Venners2001]**.

Another challenge the computer networks present is security. In addition to their potential for good, networks represent an avenue for malicious programmers to steal or destroy information or steal computing resources. Java addresses the security challenge by providing an environment in which programs downloaded across a network can be run with customizable degrees of security.

Features of Java architecture such as platform independence, security, and network-mobility work together to make Java suitable for the emerging networked computing environment. The same code can be sent to all the computers and devices the network interconnects. Objects can be exchanged between various components of a distributed system, which can be running on different kinds of hardware. Java's built-in security framework also helps make network-mobility of software more practical.

Java was designed for networks. Its suitability for networked environments is inherent in its architecture, which enables secure, robust, platform-independent programs to be delivered across networks and run on a great variety of computers and devices.

2.2.2 Java Servlets

Servlets are Java programs that perform the same function as CGI scripts. They receive a request that has been sent by a client to the web server, do some computations, and produce a response to be sent back to the client.

In order to run servlets, we need either a web server that understands servlets (such as Apache with Tomcat[V]), or a self-standing servlet engine that communicates with the web server.

Servlets can be embedded in many different servers because the servlet API, which we use to write servlets, assumes nothing about the server's environment or protocol. Servlets have become most widely used within HTTP servers and currently many web servers support Java Servlet technology.

Servlets can be used to handle HTTP client requests. For example, servlets can process data posted over HTTP using an HTML form. A servlet can handle multiple requests concurrently, and can synchronize requests. Servlets can forward requests to other servers and servlets. Thus servlets can be used to balance load among several servers that mirror the same content, and to partition a single logical service over several servers, according to task type.

A servlet's lifetime consists of three stages. That is initialization init(), the service stage doGet()/doPost(), and destruction destroy(). These are methods that are not called by the servlet itself but rather by the servlet engine, in response to certain events. The lifetime of a servlet starts when a request addressed to the servlet arrives at the server. Then the servlet engine calls its custom designed class loader. The class loader checks to see whether the servlet class is already loaded and whether the loaded version has the same time stamp as the .class file. If the servlet class is not loaded or is older than the disk file, the engine loads the class into the special Java Virtual Machine (JVM) provided for that purpose and creates an instance of it. At this point the servlet is ready for initialization. Initialization and destruction occur once per lifetime, while the service methods are repeated in response to every request.

Servlets are, in some ways, like applets. They are not self-standing applications and do not have a main() method. Also, they are not called by the user or programmer, but by another application usually a web server.

2.2.3 Wireless Application Protocol (WAP)

Most Internet technologies have been designed for desktop and large computers running on reliable networks (with relatively high bandwidth). Handheld wireless devices, however, have a more constrained computing environment compared to desktop computers. Handheld devices tend to have less memory, less powerful CPUs, different input devices, and smaller displays. Further, wireless networks have less bandwidth and more latency compared to wired computer networks.

WAP, the Wireless Application Protocol, was designed to take advantage of the several data-handling approaches already in use. WAP integrates the Handheld Device Markup Language (HDML) and the Handheld Device Transport Protocol (HDTP) developed by Unwired Planet, as well as Nokia's Smart Messaging Protocol (SMP), and Ericsson's Intelligent Terminal Transfer Protocol (ITTP). WAP services can be hosted on Web servers using technologies such as Java servlets and Java Server Pages (JSP). WAP and Java are complementary, not competing, technologies. WAP is meant for cellular phones, and Java aims at more sophisticated network terminals **[Mahmoud2001], [Chris2000]**.

[V] Apache with Tomcat is the web server that is used in this project.

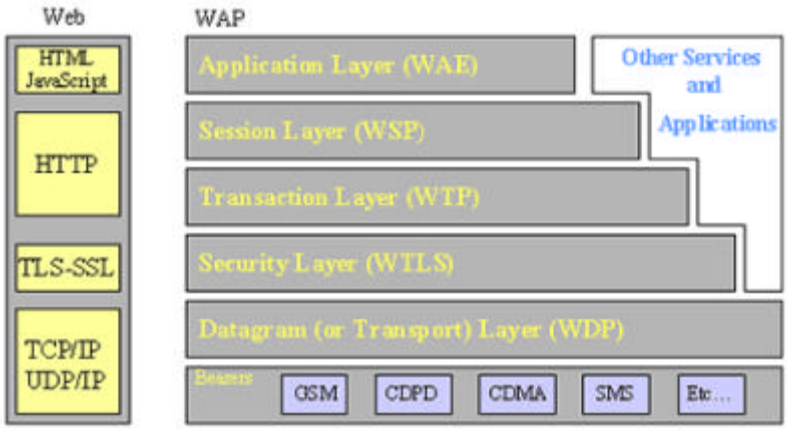


Figure 2.5 [VI]

Figure 2.5 illustrates WAP stack protocol and also shows a comparison to a Web-based protocol stack.

WML, the Wireless Markup Language, is the language adopted by members of the WAP Forum to deliver web content to wireless devices. The language is a descendent of the Handheld Device Markup Language, itself a subset of HTML, based on the World Wide Web Consortium's Guidelines for Mobile Access.

Like HTML, WML specifies the format of content and thus how it will display. It also specifies how documents are linked and how data is gathered in forms. Because of the very limited screen size available to display information on a mobile phone, WML uses the metaphor of a deck of cards to send data. A deck can be broken into several cards. Different phones display same page differently. Something that looks good on a large, new mobile phone screen may be a lot tougher on another phone with a smaller screen.

[VI] Graphic courtesy of JavaWorld.

Chapter 03 *ARCHITECTURE*

3.0 ARCHITECTURE

3.1 General Architecture of the VIC System

VIC network computing model describes a virtual computer, which is assigned to each user. Figure 3.1 illustrates the general architecture of the system.

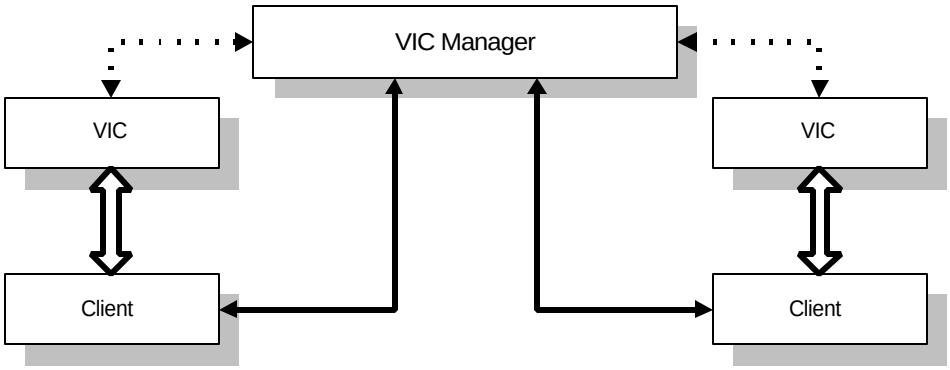


Figure 3.1

VIC System consists of number of different components. VIC Manager creates VIC for users when they request for them and send details of the created VIC to the client. Then using those details client starts a connection with the appropriate VIC. The exact method of connection establishment varies according to the type of client that is used to access the VIC[VII].

3.2 VIC Functioning Modes

Virtual Internet Computer has in two modes. These two modes are graphics mode and text mode. In the text mode user enters commands at the client side and those commands are executed in the appropriate Virtual Internet Computer and results are send to the client. In this mode we can only run text-based applications. To run graphics applications we have to switch to the graphics mode. In the graphics mode communication between the two sides are done by, Virtual Internet Computer sending only the screen shots to the client side and client software sending keyboard characters and mouse events back to the Virtual Internet Computer.

Here the role of the client is to paint the screen shots sent by the VIC on the user's screen. It also captures mouse events and keystrokes and send them to the VIC.

[VII] Connection methods used in this project are discussed in detail in the fourth chapter.

This method of communication is a bandwidth friendly. As shown in the figure 3.2, required network bandwidth gets stable with increasing resource requirements.

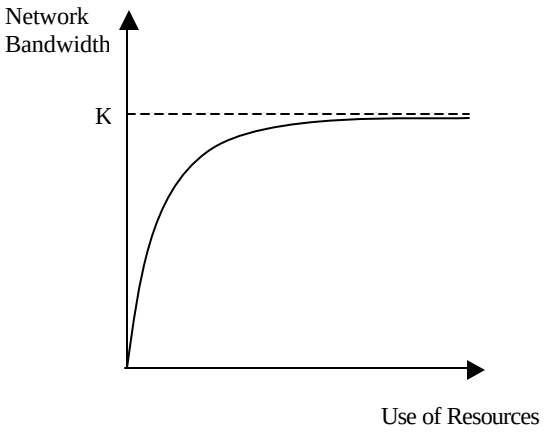


Figure 3.2

We can achieve the goal of independence of network bandwidth by reducing the value of K than the minimum bandwidth value of all available channels. This can be done by, when sending images send only the part of the image, which was changed from the last image. Lots of research has been done in this area by Citrix **[Citrix99a]** and at AT&T Laboratories **[Vnc2001]**. They have found number of techniques that can be used to decrease the required network bandwidth. Especially work at AT&T Laboratories is open source and can be used by any one. In this project only the text mode is implemented. But it is also possible to implement the graphic mode in the future.

3.3 Computing Systems that are used to host VICs

Computing systems that are used to host VICs should have considerable amount of resources. If we are going to use single processor system to host VICs then it should be high-end machine. But the objective of the VIC concept is to host VICs on multiple CPU systems, which can provide much more computing resources. These systems are usually called distributed systems.

According to the way hardware is organized multiple CPU systems can be roughly classified as multiprocessor systems and multi-computer systems. Multiprocessor systems have shared memory and multi computer systems do not **[Tanen2000b]**. Distributed systems have many advantages over centralized systems consisting of a single CPU. Distributed systems offer a better price/performance ratio than centralized systems. A distributed system may have more total computing power than centralized system and computing power can be added in small increments. Reliability of a distributed system is higher. That is if one machine crashes, the system as a whole can survive. Also in distributed systems the workload can be spread over the available machines in the most cost-effective way.

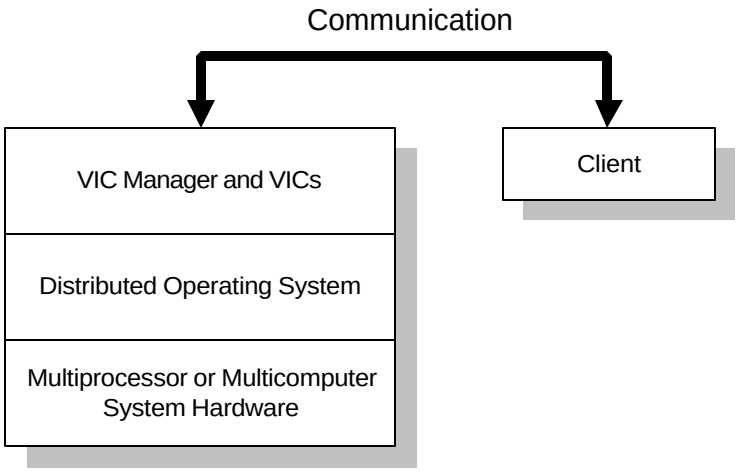


Figure 3.3

In this project VIC Manager and VICs are implemented using Java and it enable us to run it on different kinds on operating systems including distributed operating systems.

3.4 Computing Devices that are used to access VICs

One of the main objectives of the VIC concept is to allow users to access their virtual computers using variety of computing devices. There are few basic requirements that a client device need to have in order to access and use VICs effectively. First every client device needs to have some sort of a network connection. Second every client devices need to have at least text input and output capabilities. These enable users to use their VICs at least in the text mode when the capability of the client is very limited such as in the case with mobile phones. Figure 3.4 illustrates some client computing devices that can be used to access and use VICs.

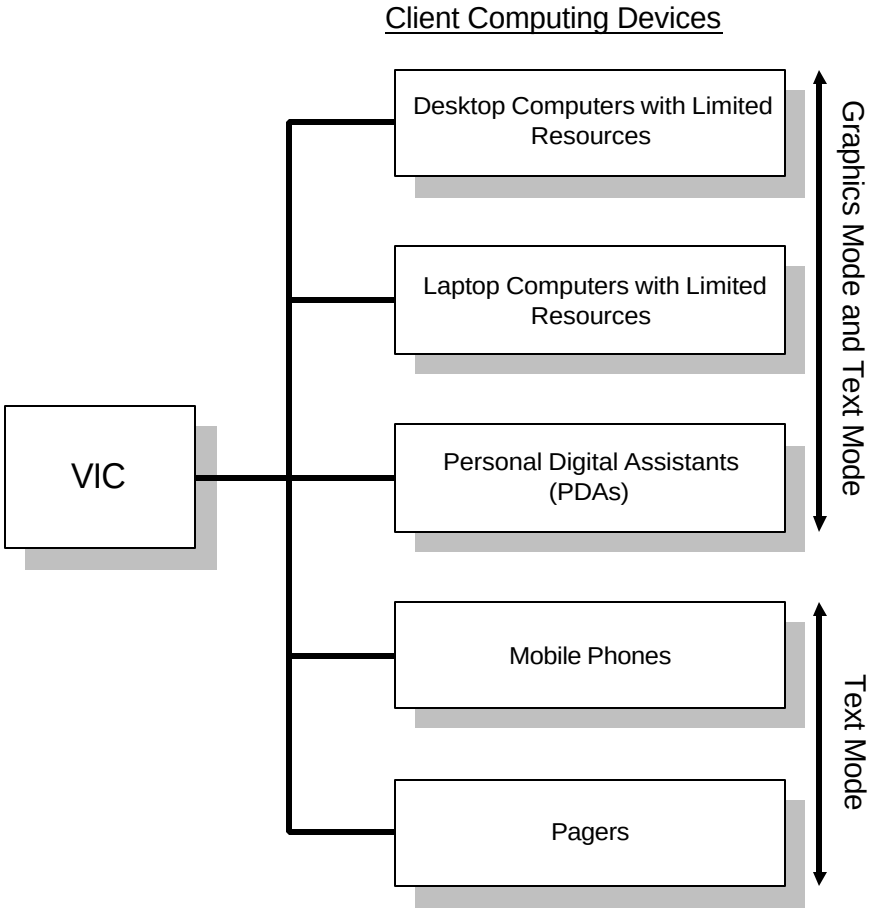


Figure 3.4

In this project we have implemented VIC system that support, some of the above client computing devices including mobile phones.

Chapter 04 *DESIGN*

4.0 DESIGN

4.1 Basic Design

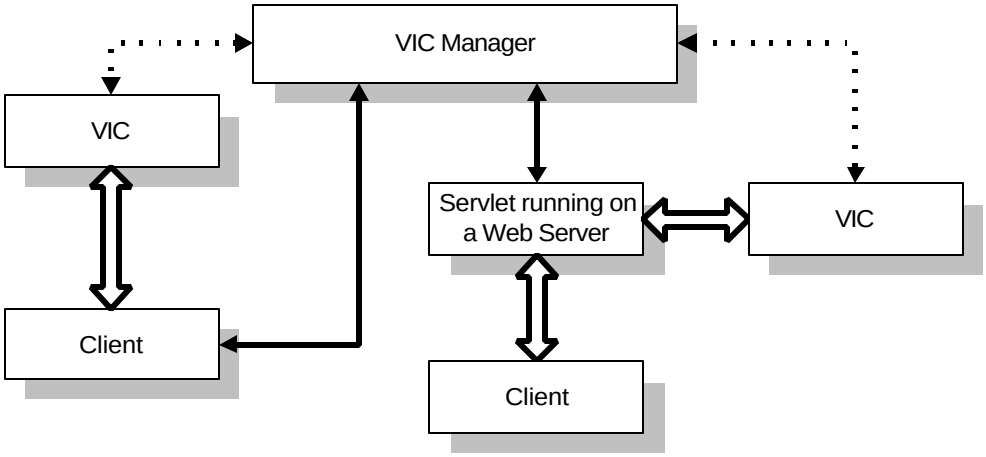


Figure 4.1

Figure 4.1 shows the VIC system design for this project. This system is based on client server architecture. According to this design there are two ways a client can initiate a connection with a VIC. One way is client directly communicates with the VIC manager and asks for a particular VIC. Then the VIC manager creates a VIC for the client and sends details about this newly created VIC to the client. Then using those details client makes a connection with the VIC. Also note that in the case where client is previously connected but did not shutdown the VIC, VIC manager does not create a new VIC. It sends the details of the existing VIC that belongs to this particular client and then the client uses this information to establish a connection.

When the connection between a client and a VIC is first established, the VIC begins by requesting authentication from the client using a challenge-response scheme, which results in the user being prompted for a password at the client end.

In the second way client do not directly communicate with VIC manager. Here the web browser or WAP browser in the client device makes a connection with particular web server. A servlet running on this web server dynamically create a login page and present it to the client. Then client sends login details to the servlet using this page. Now the servlet initiate a connection with the VIC manager and ask for a VIC. Then VIC manager sends details of the VIC to the servlet and servlet make connection with the VIC. Here the role of the servlet is to get requests from the client send it to the VIC and get the results from the VIC, create a html page or wml page containing those results dynamically and send it to the client.

4.2 Client - VIC Communication

4.2.1 Client Java Application and VIC Communication

Figure 4.2 illustrates the way client Java application participates in the VIC system.

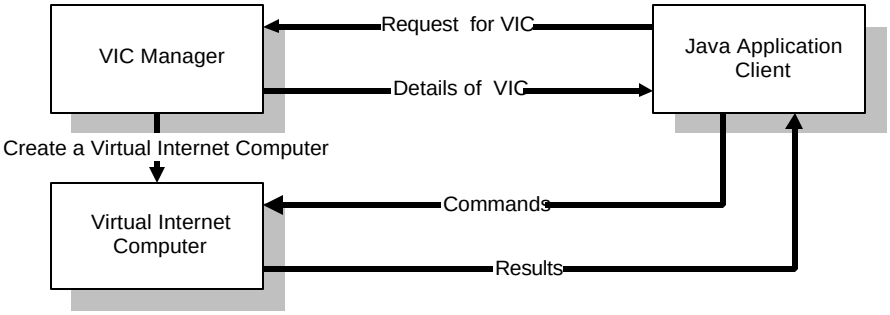


Figure 4.2

One of the basic requirements of this method of accessing the VIC is that we need the client application stored on the client device. Here client device does not need to be a computer. It can be any device, which is capable of running a Java application and also the device should have input output capability.

When the user enters login details, client application makes a connection with VIC manager and sends a request asking for a VIC. With this request client application sends the information of the user in order to identify his VIC. If it is already activated then VIC manager sends details of it to the client applet. If it is not activated then VIC manager starts that particular VIC and sends detail to the client. After this process client's connection with the VIC manager is terminated.

Now using the information send by the VIC manager, client application makes a connection with the activated VIC. Then VIC verifies the user id and password and proceeds with the communication.

When we finish working with Virtual Internet Computer, we have to log out. There are primarily two methods that we can use to logout. The first method shuts down the Virtual Internet Computer when we logoff. The second method leaves the VIC running after we have logout. The second method of log out is another important feature of the Virtual Internet Computer. Here VIC will carry out tasks, which was given by the user, after he had log out from the VIC. Here the VIC that corresponds to particular VIC user can exist with out the client. Some times it is important to leave the Virtual Internet Computer running in situations such as rendering of a complex animation are in progress.

Apart from the normal communication client application supports transfer of files from the client-computing device to Virtual Internet Computer and vise versa.

4.2.2 Client Java Applet and VIC Communication

Figure 4.3 illustrates the way client Java applet participates in the VIC system.

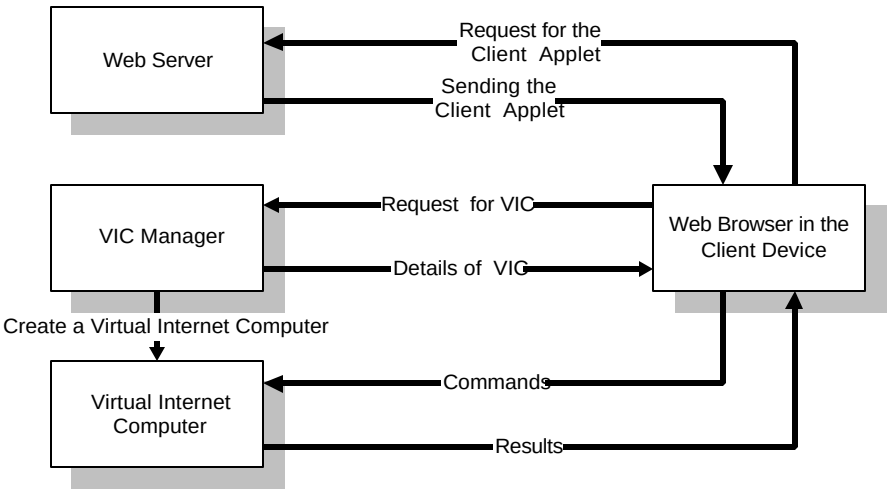


Figure 4.3

Here when the user enters the URL of the Virtual Internet Computer web site, browser will download the latest client Java Applet and execute it. Then the applet connects with the VIC Manager application. This application may be located on the same computer system as the web server or in a different computer system. The client applet needs to be digitally signed by a certificate authority to make this network connection. Otherwise users required reducing their security level to permit the applet to make network connection.

Client applet's communication with the VIC system is very much similar to the client Java application. But client Java applet does not support file transfers between VIC and client devices.

4.2.3 Web Client and VIC Communication

Figure 4.4 illustrates the way Web client participates in the VIC system.

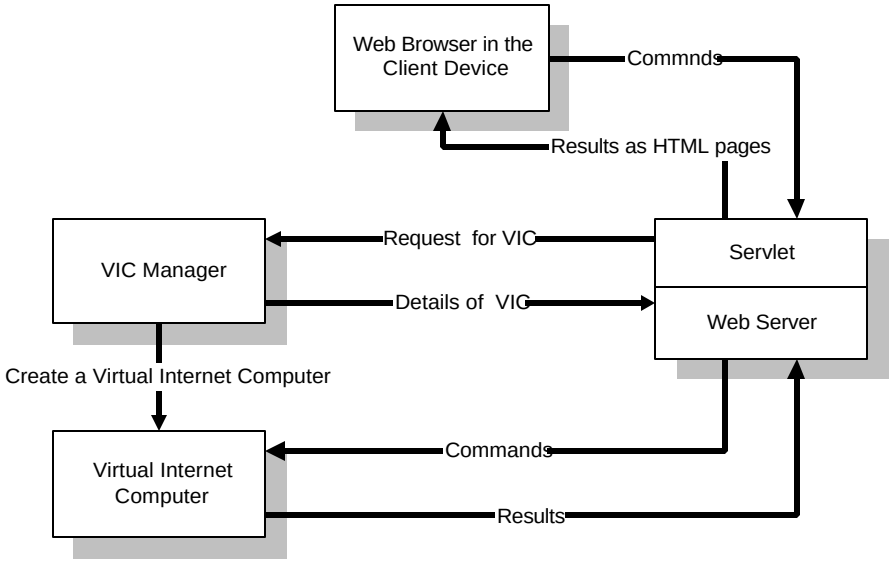


Figure 4.4

When we use client applet to connect to our Virtual Internet Computer we need a Java enabled web browser. Also if the applet is not signed by a certification authority then we have to disable some of our security features in order to get connected to our VIC. In these situations we can use the web client to access our Virtual Internet Computer. This HTML based web client does not need sophisticate web browsers. It can be used with very simple web browsers located in some mobile computing devices.

As in the case with client applet when the user enters the URL of the Virtual Internet Computer web site, the Web browser will initiate a servlet, which was situated in the web server. This servlet will dynamically create an HTML page, which allows user to enter login details and submit them. These login details are used by the servlet to communicate with the VIC manager and locate the appropriate VIC.

After a successful connection is established with the VIC, servlet will act as a converter. It will convert HTML requests from the browser to requests that are passed through a socket to the VIC and results that are coming from the VIC to HTML pages that are sent to the browser.

4.2.4 WAP Client and VIC Communication

Figure 4.5 illustrates the way WAP client participates in the VIC system.

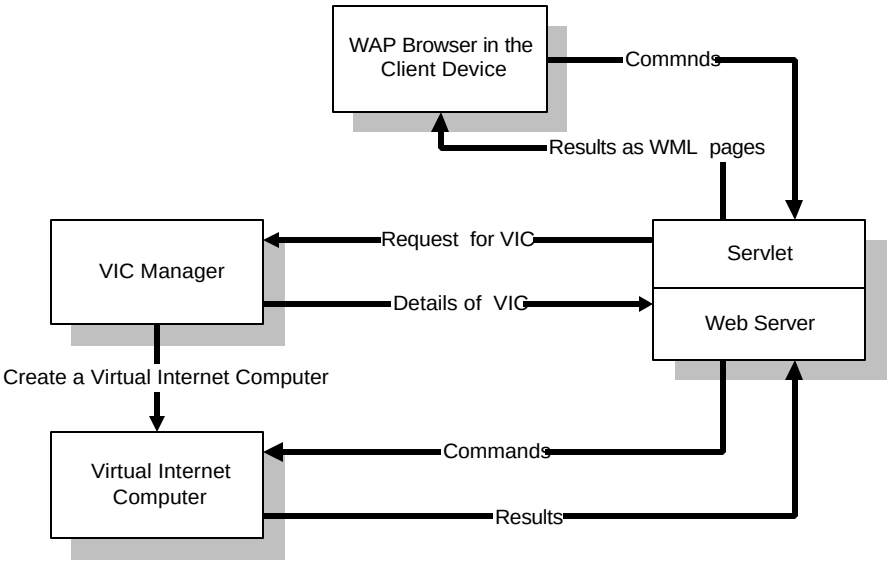


Figure 4.5

WAP client enables users with a WAP enabled mobile phone to access their Virtual Internet Computers. Users who are on the road can access their VICs and give some task to it by using the WAP client.

Here the user enters the URL of the Virtual Internet Computer web site, and selects the appropriate link to go to the WAP client page. Then the WAP browser in the mobile phone will initiate a servlet, which was situated in the web server. This servlet will dynamically create a WML page, which allows user to enter login details. These login details are then used by the servlet to communicate with the VIC manager and locate the appropriate VIC.

WAP client, works almost same as the Web client. The difference is that in the case with the Web client servlet convert from HTML request to socket request and vise versa. But in the case of the WAP client conversion take place from WML request to socket request and vise versa.

4.3 Why Java is used to design VIC System.

Java is chosen to design and implement the VIC System because of number of reasons. First every Java application is executed in it's own virtual machine. Using Java it is easy to implement a Virtual Internet Computer. Secondly implementing in Java enables us to use the VIC System on variety of computer system ranging from PC to supercomputers, without changing it. Use of Java to implement the VIC System also enable us to use it on distributed systems, which support Java platform.

4.4 Java Virtual Machine (JVM)

The heart of VIC System is creating a JVM for each VIC user and running VIC application on it. Virtual Internet Computer, in this implementation is the VIC application plus JVM. This VIC application works as a simple operating system, which runs on the JVM.

The Java virtual machine is an abstract computer. Its specification **[TimLin99]** defines certain features every Java virtual machine must have, but leaves many choices to the designers of each implementation. For example, although all Java virtual machines must be able to execute Java bytecodes, they may use any technique to execute them. Also, the specification is flexible enough to allow a Java virtual machine to be implemented either completely in software or to varying degrees in hardware. The flexible nature of the Java virtual machine's specification enables it to be implemented on a wide variety of computers and devices.

A Java virtual machine's main job is to load class files and execute the bytecodes they contain. Java virtual machine contains a class loader, which loads class files from both the program and the Java API. Only those class files from the Java API that are actually needed by a running program are loaded into the virtual machine. The bytecodes are executed in an execution engine **[Venners2001]**.

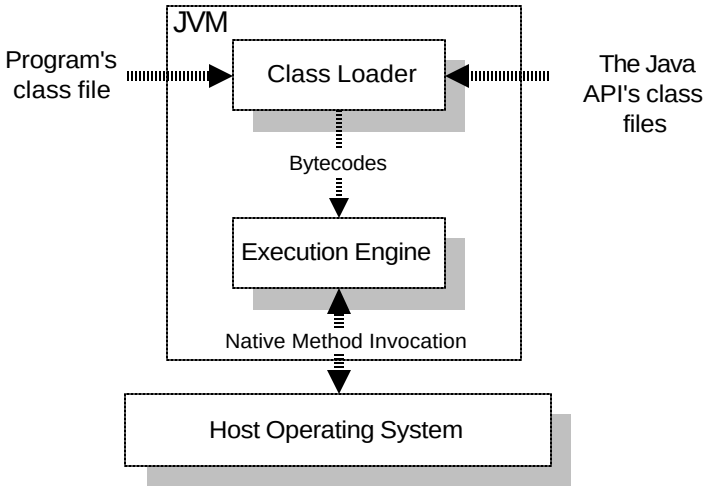


Figure 4.6

Figure 4.6 shows a Java virtual machine implemented in software on top of a host operating system.

4.5 VIC Design

4.5.1 VIC Architecture

VIC consists of two parts. That is Java Virtual Machine and VIC application. In VIC system each user gets dedicated JVM, which is an abstract computer. VIC application is the operating system that runs on that computer. Client represents the input output devices that are used to communicate with the abstract computer.

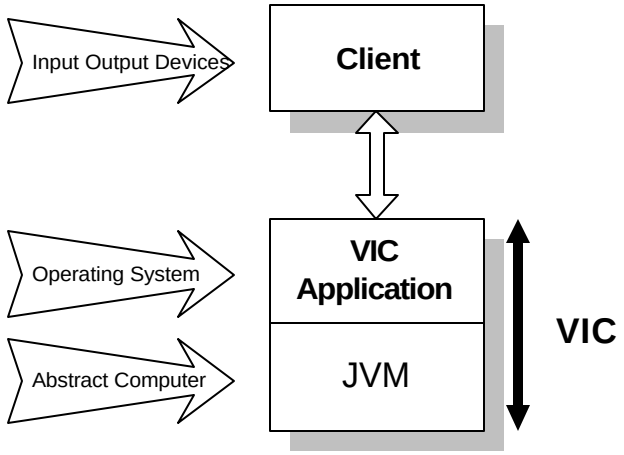


Figure 4.7

Figure 4.7 illustrates the architecture of the VIC.

4.5.2 VIC Commands

Using various clients, users can access their VICs. After a successful connection is established users need some way to do various tasks in the VIC such as file management, processing data, executing files etc. This is accomplished by using set of commands.

The VIC supplies some basic commands internally. These commands can be used to do basic operations. Since lots of people are familiar with DOS commands, the VIC commands are design with lot of similarity to DOS commands.

If additional functionality than that is provided by these basic commands are needed then it can be obtained by implementing that function as Java class file and coping it to a directory in the VIC and executing it. Class files can be executed by typing only the name of that class file with arguments. If you want to execute the command in background then add 'bg' at the end of the command. Note that any text based Java application class file can be executed in the VIC.

Table 4.1 lists the basic commands in a VIC and its description.

Command	Description
attrib	Display and change file attributes.
cd	Changes the current directory.
copy	Copies files between directories.
cls	Clears the user screen.
comp	Compares two files.
dir	Lists a directory of filenames on the current directory.
delete	Delete files or directories.
find	Finds and display files.
get	Copies file from the VIC to the client device .
help	Displays help information.
logoff	Log out from the VIC.
mem	Display the amount of used and available memory.
move	Moves files or directories between directories.
md	Creates a new directory.
pcd	Print's the current directory.
ren	Rename an existing file.
shutdown	Shut down the Virtual Internet Computer.
type	Display a designated file on the screen.
ver	Display the version of the Virtual Internet Computer being used.

Table 4.1

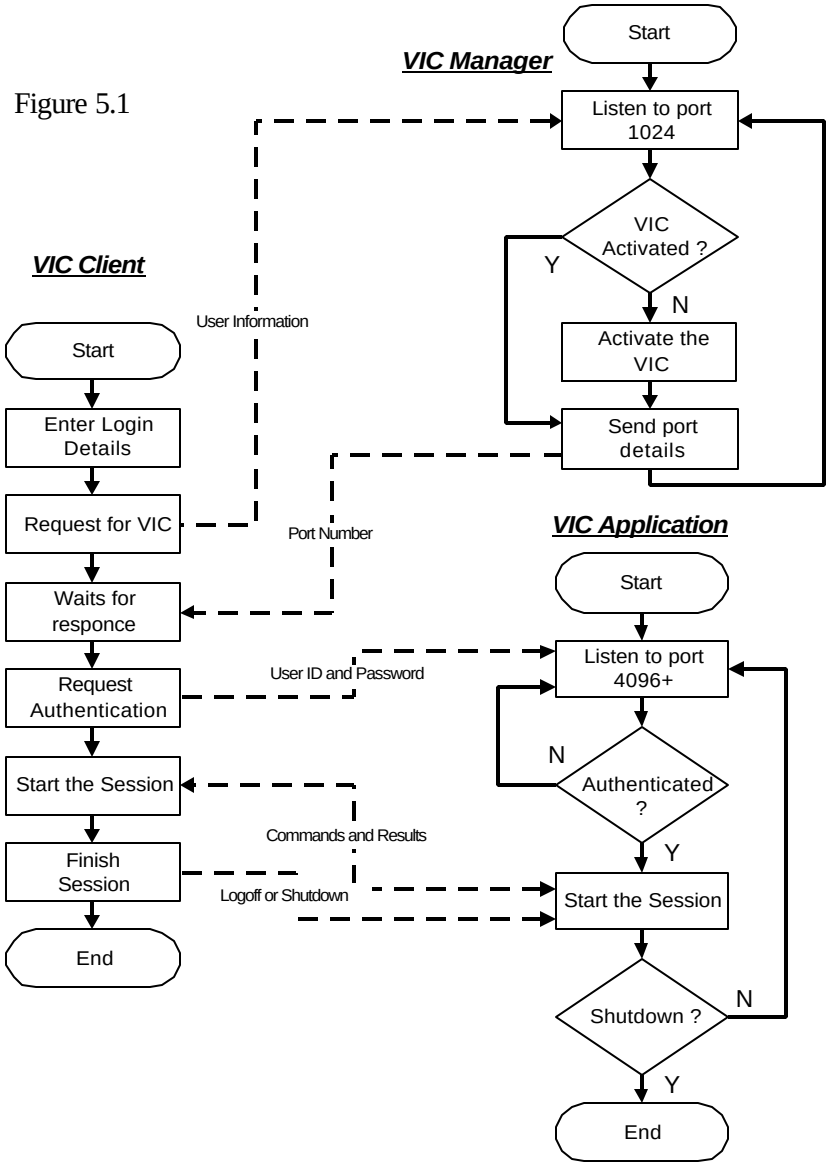
Chapter 05 *IMPLEMENTATION*

5.0 IMPLEMENTATION

5.1 VIC System

VIC System consists of VIC manager application and set of VICs that belongs to registered users. Functionality and communication of the VIC system and a typical client is shown in the figure 5.1. According to the figure each client makes two connections with the VIC Manager and VIC application in order to start a session. This kind of communication needed because VIC Manager and VIC application running in two separate Java virtual machines.

Figure 5.1



As shown in the figure 5.1 client first gets connected to the VIC application. Then the client provides it's user information and ask for the user's VIC. After this request VIC manager check to see whether the requested VIC is already activated. If it is activated then the VIC sends the client the information about the port in which the requested VIC waits for connection. If the requested VIC is not activated then VIC manager activates it by creating a new JVM and running VIC application on it. After activation VIC manager sends details about the activated VIC to the client as in the previous case. When all this is finished the VIC application waits for another client at port 1024.

When the client gets information about the port in which the user's VIC is listening to, client sends a request to it and the VIC responds and makes a connection with the client. Then VIC verifies the user name and password of the client and start to accept commands.

After the session client can get log out of the VIC or shut it down. If the client chooses to logout then the VIC continues to operate and perform tasks assigned to it by the user. If the user decides to shutdown then the VIC is terminated until reactivated by the appropriate user.

Now we will take a closer look at the VIC Manager application and VIC application (operating system of the VIC).

5.1.1 VIC Manager Application

The main purpose of the VIC Manager is the management of VICs. Functionality required of the VIC Manager from the VIC administrator's viewpoint is given in the Use Case Diagram in figure 5.2.

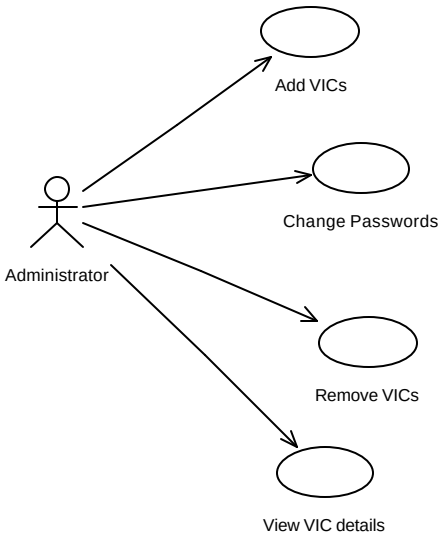


Figure 5.2

VIC administrator needs to add new VICs to the system using the VIC Manager. Also he needs to change password, remove VICs and view the status of a particulat VIC when needed. These functions are provided by the VIC Manager application.

VIC Manager application consists of only one class. VIC_Manager class have methods to handle VIC management and graphical user interface. In figure 5.3 some of the methods and attributes of the VIC_Manager class is shown.

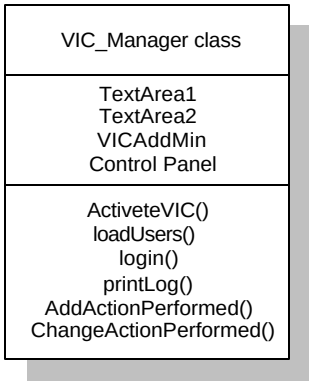


Figure 5.3

VIC Manager always listen to the port 1024. All the VIC clients send VIC request to this port. When a user tries to connect, then VIC Manager accept the connection, create a separate scocket to deal with that client, pass it to login() method and continues to listen to the port 1024 for other users.

```
try {
   serverSocket = new ServerSocket(1024);
} catch (IOException e) {
   System .err.println("Could not listen on port: 1024.");
   System.exit(-1);
}
  ...
  ...
while (true) {
 login(serverSocket.accept());
  }
 ...
```

In the login() method user is check to see whether he is authorized and invloke the ActiveVIC() method to activate the required VIC.

ActiveVIC() method is responsible of creation of VIC. Creation of the VIC is done using Java class called Runtime class. Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the getRuntime() method.

Using the exec() method of the Runtime class, it is possible to execute extern programs from a Java application. In order to do it a new Process object is instantiated and it is used to execute the specified command. In this case VIC application is executed.

The Java code below is used for the creation of the JVMs or in other words VIC.

```
public static void ActivateVIC (String ExternCommand) throws IOException        {

    try {
       Process p = Runtime.getRuntime().exec(ExternCommand);
    }
    catch (IOException e) {
       System.out.println("Can not activate the VIC.");
       e.printStackTrace();
       System.exit(-1);
    }
}
```

All the VICs activated by the VIC Manager listen to a port. This port is specified by the VIC Manager and taken from a pool of ports. In this case we have chosen port numbers between 4096-5006, which allows creation of 10 VICs. The number of VICs that can be created by the VIC Manager is not constant and can be change. Note that if the number of VICs is changed then the pool of port numbers also need to be expanded.

The following figure illustrate a screen shot of the VIC Manager application.

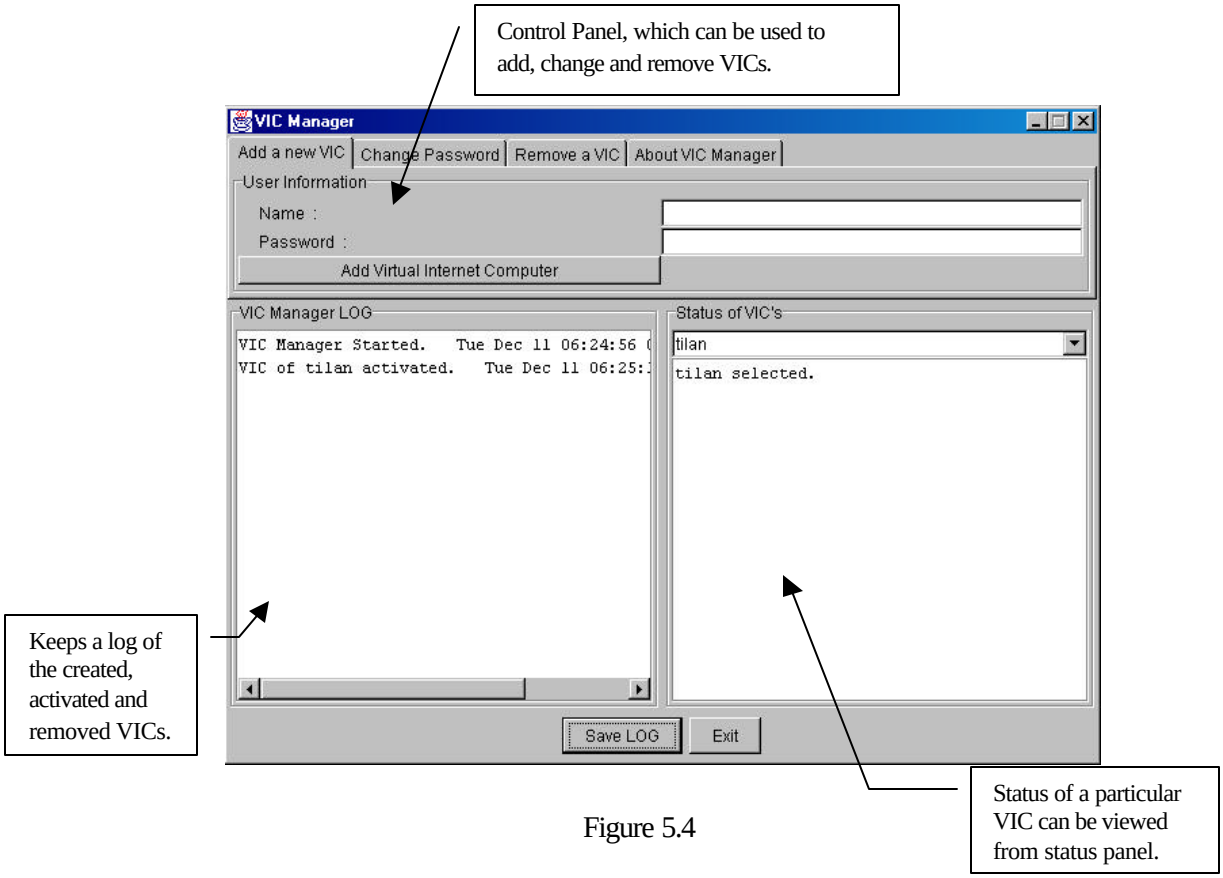


Figure 5.4

Control Panel is used to add, remove VICs and change passwords. Each tab page in the Control Panel allows VIC administrator to do these tasks. VIC Manager LOG section shows all the activities carried out by the VIC Manager. It can be also saved as a log file for later analysis. Status panel section allows VIC administrator to view the status of a particular VIC.

5.1.2 VIC Application

VIC Application is the operating system that runs on the Java Virtual Machine. VIC application combined with the JVM forms the Virtual Internet Computer. VIC facilitates us to run Java applications, as we execute EXE files on a DOS machine.

VIC application consists of five classes as shown in the class diagram in figure 5.5. VIC class is the main class of the application. Most of the internal commands are implemented in this class. Also this class is responsible for authorization and communication with the client.

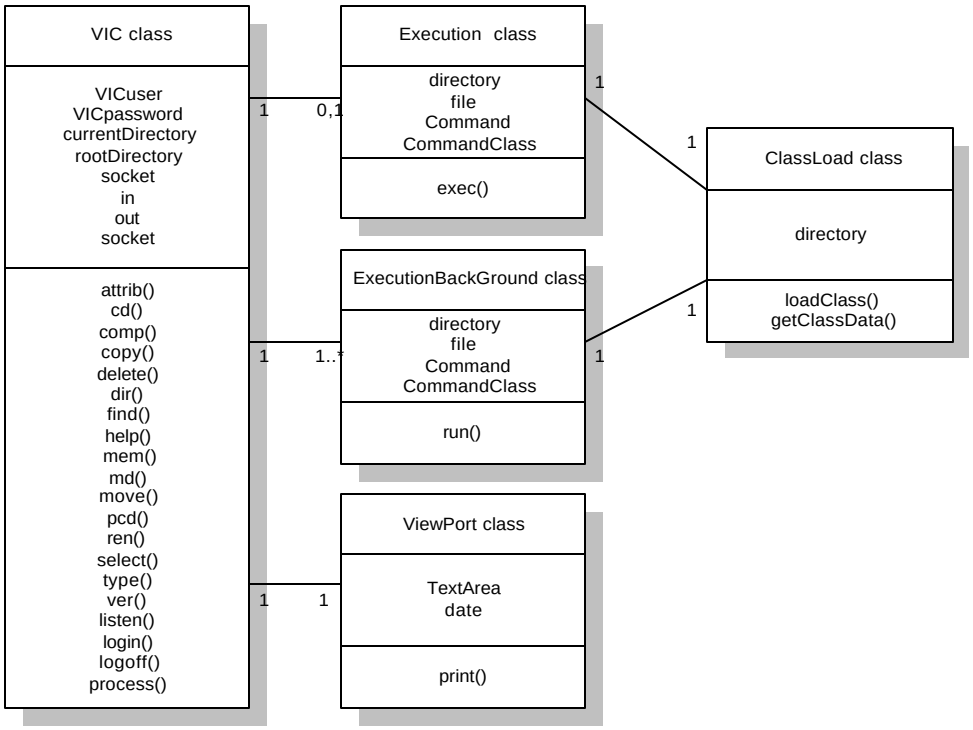


Figure 5.5

Execution class is used when VIC application executes an external class file. If we use this class then execution will take place in the thread of the VIC. If we want to execute the command as a backgroud process the we need to use the ExecutionBackGround class. Both Execution and ExecutionBackGround classes use ClassLoad class to load classes from the file system.

ViewPort class is used to see what command are beeing executed by the user and when they are executed.

VIC application uses a dedicated directory of the hosting system as the hard disk of the Virtual Internet Computer. Users store their applications and documents on this directory but they sees it as their storage disk.

VIC application redirects the standard input and output of the Java Virtual Machine to the input and output streams coming from the client using the following code.

```
try {
...
PrintStream StreamOut = new PrintStream(new BufferedOutputStream
                                                (socket.getOutputStream()));
BufferedInputStream StreamIn = new BufferedInputStream(socket.getInputStream());
System.setIn(StreamIn);
System.setOut(StreamOut);
System.setErr(StreamOut);
...
catch catch (IOException e) {
...
}
```

The process() method of the VIC class is responsible for calling appropriate method when the user gives a command.

Figure 5.6 illustrate a screen shot of the view port of VIC. View port shows the commands that the user executes with the date and time of the execution.

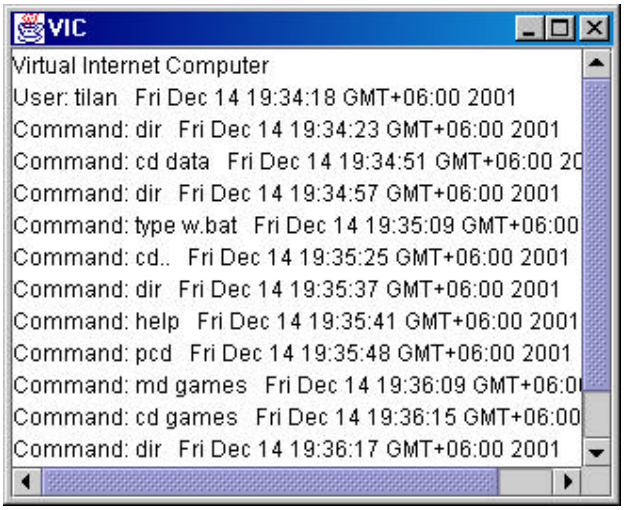


Figure 5.6

5.2 VIC Clients

5.2.1 VIC Client Application

VIC client application is one of the many ways a user can access his Virtual Internet Computer. In order to use VIC Client application user need a client device, which has a Java Platform and graphics capability. So VIC Client application requires relatively more resources from the client-computing device.

The main purpose of the VIC application is sending user commands typed at command area to the VIC and displaying the results send back by the VIC, on the user view area.

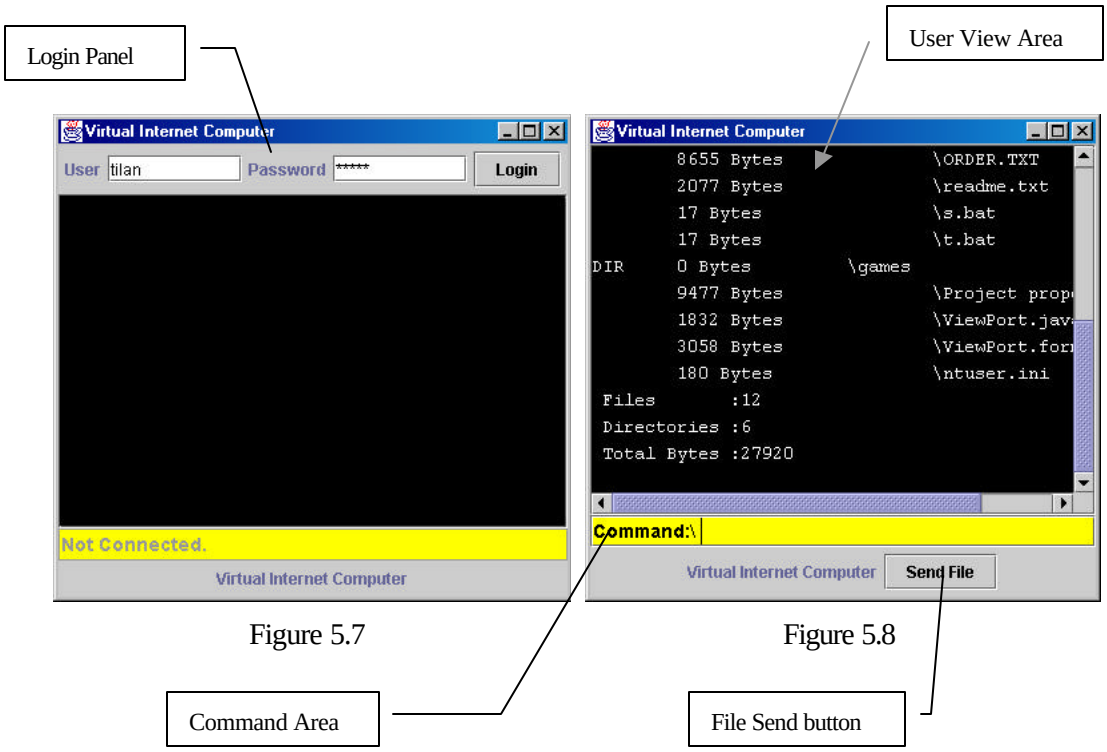


Figure 5.7

Figure 5.8

Login Panel is used to enter user name and password. VIC application just before login is shown in the figure 5.7. When the login button is pressed login information send to the VIC Manager and after successful login, Login Panel disappears as shown in the figure 5.8. Now the user can enter commands in the Command Area and view the results in the User View Area.

User View Area of the VIC Client application is scrollable. Users can scroll and view results of the previous commands if they want.

Some times users may need to send files to the Virtual Internet Computer. This can be achieved by pressing "Send File" button. When the user presses the "Send File" button, a dialog box will open and ask to choose the file that want to be sent. When the file is chosen it will be copied to the current directory in the Virtual Internet Computer.

VIC Client application is implemented as VIC_Client class. Figure 5.9 shows some of the attributes of VIC_Client class.

The method initComponents() creates the graphical elements of the VIC Client application. When user press the login button after inserting user details, loginActionPerformed() method is executed. This method invokes the login() method and after successful connection print the welcome message on the User View Area.

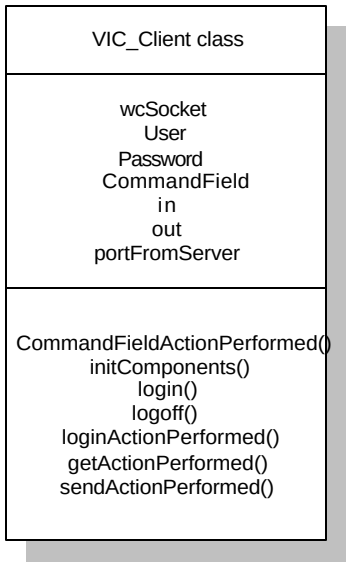


Figure 5.9

CommandFieldActionPerformed()[VIII] is the most important method in the VIC Client application. It is invoked every time a user enters a command. First it checks whether a command is actually typed. Then if the command is to clear the User View Area, perform that operation. If the command is something else then it sends it to the VIC across the network to perform that operation. Finally, when the results come, this method prints them on the User View Area.

getActionPerformedMethod() and sendActionPerformed() methods are used to transfer files between the client device and the Virtual Internet Computer. sendActionPerformed() method is responsible for sending files from the client device to VIC and getActionPerformed() method is used for transferring files from the Virtual Internet Computer to the client device.

VIC Client Application is the only client implemented in this project, which has the capability of transferring files between the Virtual Internet Computer and client device.

[VIII] Commented Source Code of this method is given in the Appendix.

5.2.2 VIC Client Applet

VIC Client applet is another way we can access our Virtual Internet Computer. VIC Client applet enable us to use it on any Java capable web browser. It has scrollable User View Area and good look and feel.

VIC Client application also consist of a single class which send request to server and print results on the User View Area. Functionality of this applet is similar to the VIC Client application. But VIC Client applet does not support file transfering between VIC and the client device.

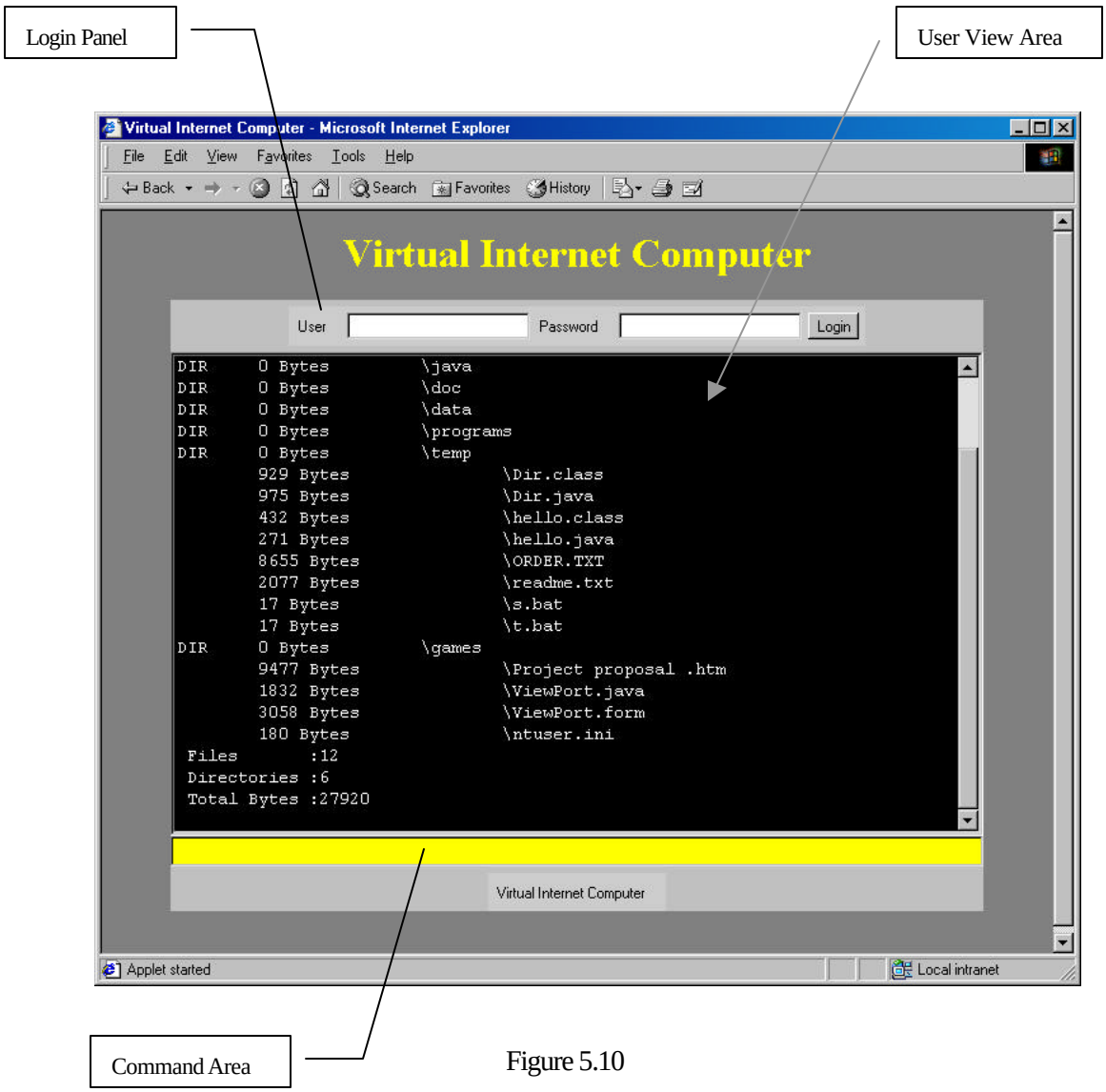


Figure 5.10

Figure 5.10 shown a screen shot taken, while a VIC session is progressing with the VIC Client applet.

5.2.3 Web Client

Web Client, generates html pages dynamically to handle a VIC session. This method of access is very useful for users with web browsers without Java capability.

As explained in section 4.2.3, Web client is a Java servlet that run on top of a web server, which creates html pages and present it to the user. This servlet take care of communication with the VIC System. Servlet and the user communicate using standard html request and responds.

Web Client is implemented as VIC_Html class. Figure 5.11 shows some of the attributes of VIC_Html class.

The first four methods are responsible for generation html pages for the user. login() and logoff() methods generate html page for login and logout. invalidLogin() method generates the html page for invalid login. outputScreen() method generate dynamically changing html page according to user commands.

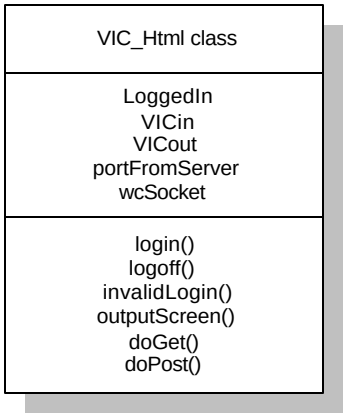


Figure 5.11

Figure 5.12 shows the login page generated by the login() method.

Figure 5.12

To handle HTTP requests in a servlet, we have extended the HttpServlet class and override the servlet methods that handle the HTTP requests that the servlet supports. In the Web Client we have done only the handling of GET and POST requests. The methods that handle these requests are doGet() and doPost(). doGet() method is invoked when the web browser request the first html page from the servlet. This method set the MIME[IX] type and calls the login() method to enable users to enter login information.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException, ServletException
        {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          this.login(out);
        }
```

The line "response.setContentType("text/html");" ensures that the correct MIME type is set for the Html documents.

After displaying the login page, for every other browser request doPost() method is invoked. It handles the connection with the VIC System and invokes html page generating methods. Following figure 5.13 shows an on going VIC session using the Web Client.

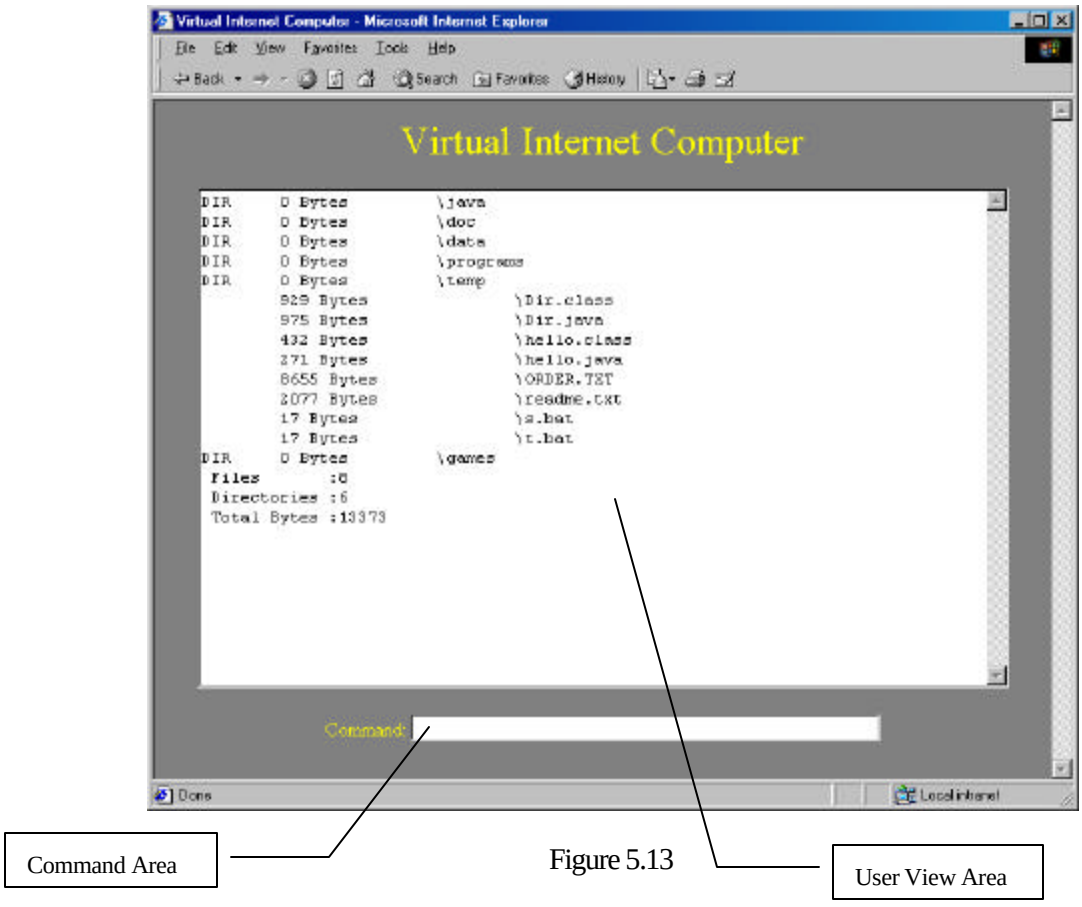


Figure 5.13

[IX] When a regular Web browser receives a page, it has to distinguish between HTML, image data, audio, and video. To enable that, with every response from the Web server, a piece of header information with every file comes down to the browser. This piece of information is known as MIME, which stands for Multipurpose Internet Mail Extension.

5.2.4 WAP Client

WAP Clients enables users to access their VICs using WAP enabled mobile phones. This is achieved using another Java servlet, which dynamically generate WML pages and sent them to the mobile phone.

VIC_Wap class, which implements WAP Client, has similar methods as the Web Client except methods in VIC_Wap class generate WML pages. Also in the doGet() method "Response.setContentType("text/vnd.wap.wml");" line is used to ensures that the correct MIME type is set for WML pages.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException, ServletException
  {
    response.setContentType("text/vnd.wap.wml");
    PrintWriter out = response.getWriter();
      this.login(out);
  }
```

Figure 5.14 shows how the VIC login page is displayed on mobile phone. In this project we have used a WAP emulator to develop and test the WAP client. Figure 5.15 shows a VIC session that is in progress using the WAP client.

Figure 5.14

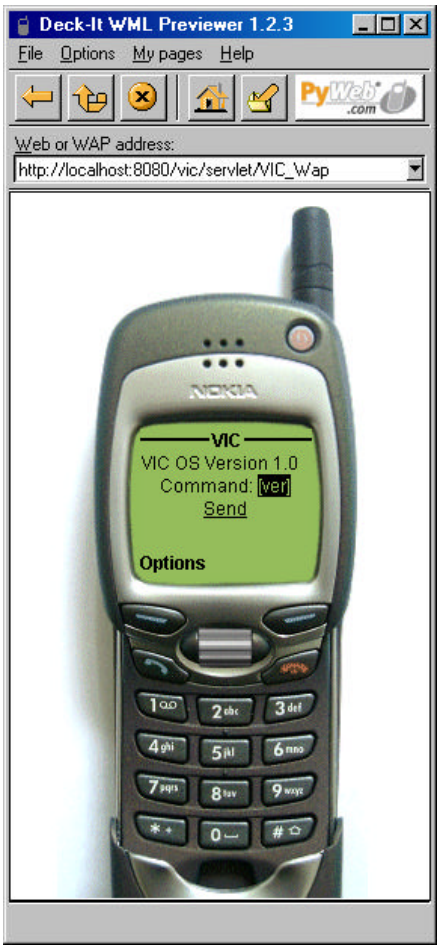


Figure 5.15

Chapter 06 *CONCLUSION AND FURTHER WORK*

6.0 CONCLUSION AND FURTHER WORK

6.1 Aims and Objectives Achieved

Proof of concept implementation of the VIC concept achieved most of its aims and objectives. VIC System implemented in this project can be accessed from anywhere in the world. It enables users to access their VICs using many forms of clients. If the VIC System is on top of a distributed system it can achieve the goal of providing computing power that was not possible with single CPU systems. Also VIC clients develped in this project requires very little network bandwidth to access VICs.

What we have achieved paved the way for more improvements that can be made to the VIC system. In final sections of this report we will discuss some improvements that can be made to the VIC system and possible future application of the VIC concept.

6.2 Improvements for text mode implementation of VIC System

6.2.1 Natural Language Processors

In order to use VIC in text mode users need to know the command language used by the VIC. We can improve this situation by using a natural language processor to interpret natural language commands given by the users. This allows any one without the knowledge about the commands used by VIC System to operate the VIC using natural language. This gives the VIC system the independence from command language.

6.2.2 Controlling VIC using Email

To control a VIC, a user needs to be directly connected to the VIC using some sort of computing device. Here the direct connection is essential. But we can add an interface to the VIC to process commands or natural language commands sent by an email to the VIC. Then email back the results automatically to the user. This extends the reach of users to their Virtual Internet Computer bit further.

6.2.3 Controlling VIC using SMS

Another offline way to controlling VIC is using SMS[X] in mobile phones. This is very much similar to the offline email controlling method above. Here a VIC user sends an SMS asking to do a particular job from his VIC and results of that job are sent back by the VIC as another SMS message to the user. This method of controlling is useful for the users who do not have WAP enabled mobile phone or direct Internet connection.

[X] SMS stands for Short Message Service.

6.3 Implementation of the Graphic mode

In this project we have done only the text mode implementation of the VIC concept. Text mode operation can be very useful especially for client devices, which does not have graphics capability. But the complete implementation of the VIC concept requires the implementation of the graphics mode. Since today's computer users are addicted to easy and user friendly Graphical User Interface, graphic mode, which supports GUI, is very important.

6.4 Support for Windows Application

Proof of concept implementation of this project is done using Java and it can be used to run Java applications. But today most of the people in the world use Windows based software. We can implement VIC System that supports Windows applications. This can be done in two ways. The first way is to implement Win32 API on top of the Java implementation. This will provide us the platform independence for Windows applications and the ability to run Windows applications on distributed systems. The other way is to implement VIC System using conventional programming language such as C++. But in the latter case only limited number of computing platforms will be able to support this VIC System.

6.5 Companies that host VICs.

Concept of Virtual Internet Computer gives rise to a question who is going to host these VICs. Hosting of VICs is something similar to hosting web sites. Owner of the web site pays some amount of money to the hosting party. In the case with VIC System some one should host and maintain VICs on behalf of VIC users. In return VIC users can pay them monthly or yearly.

Hosting a VIC System is not simple as hosting a web site. It involves lot of issuers that should be considered carefully. These include,

- *Resource Management Issues.*

  Amount of resources a VIC user can utilize should be controlled. Each VIC user should be given some mount of processing resources and storage resources according their needs. If a particular user needs more resources then he has to pay more. Hosting party must be able to prevent VIC users from using more resources than that are allocated to them.

- *Security Issues.*

  Hosting a VIC System involves lot of security issues. Since VIC users can execute commands in the hosting computer system, measures must be taken to prevent them from damaging other part of the system. Also lot of unauthorized people will try to access the VIC system. This is also should be prevented.

- *Maintenance Issues.*

  VIC hosting party should provide services such as data backup to the VIC users in case something disastrous happens. Also it is the responsibility of the hosting party to upgrading their hosting computer system with improvement in the technology.

6.6 Intelligent VICs

Virtual Internet Computers that are implemented in this project are simple virtual computers that can run text-based Java applications. But we can improve these VICs to behave intelligently on the Internet behalf of its users. These intelligent VICs can do lot of things. Some of the tasks these intelligent VIC can do are as follows.

- Communicate with other VICs on the Internet to achieve a common goal.
- Represent the user when he is not online in day-to-day work.
- Do intelligent searches on the Internet.
- Be alert about unauthorized access of hackers.
- Function as an email answering machine.
- VICs can monitor stock market and inform the user if some thing important happed.

6.7 Extinction of Personal Computers

Introduction of personal computers changed the way people use computers. But despite of all the advantages of PCs there are few disadvantages that might prevent it from going in to the future.

One of the main problems with PC is the constant need to upgrade the hardware. Once in every six months we need to upgrade hardware in our PCs to stay with the current technology. This duration is getting shorter consistently. Also today's PCs consist of lot of complex equipment and failure rate of these is fairly high.

PCs need considerable amount of power to run. Usually desktop PCs get power from main AC power supply and it makes those PCs less portable. Desktop PCs are also vulnerable to power failures and lightning. Laptop computers, which have batteries, are portable, but require constant need for recharge. This is very inconvenient. So PC maintenance is big problem especially for non-technical people. Also PCs are fairly large and heavy devices. Even laptops carry considerable amount of weight.

VIC System makes it possible to construct low cost, lightweight, wirelessly networked, sophisticate digital terminals that can be used to access Virtual Internet Computers. These digital terminals are capable of displaying graphics. Unlike PCs they will not have part mechanical devices such as hard disks and floppy drives. Because of that they will consume very little power.

These devices will be very similar to thin client devices that are used in server based computing today. In corporate Intranet environments these thin client devices are already replacing PCs **[Citrix99a]**. It is possible to believe this trend will spread throughout the world, with advances in networking technologies, which gives fast, cheap access to the Internet.

6.8 Internet becoming a huge distributed system

The future is difficult to predict especially in the computer industry. But we can make some guesses about the future by analyzing today's trends in the computing world. Today there is an emerging network-centric computing environment. In this environment the Internet will play an important role.

With the current fiber technology, the achievable bandwidth is exceeding 50,000 Gbps mark **[Tanen2000a]**. In the near future almost all network cabling will change from copper to optical. This makes the communication between computers on the Internet much faster. This new network environment gives a possibility of making a large distributed system using the computers in the Internet.

In this kind of global computing environment idea of physical computer to physical computer communication will diminish from the user view of the computing environment. So the idea of virtual computer (Virtual PC) to every person may override today's personal computer notion.

In the future every person might have his own Virtual Internet Computer on the Internet in which he can access and use, in various ways to do unlimited number of tasks.

Appendix *Source Code*

VIC_Manager.java

```
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.net.*;
import java.io.*;

/**
 * Title:      Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

public class VIC_Manager extends javax.swing.JFrame {


  public VIC_Manager() {
    initComponents ();
    pack ();

        //Center the window
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = this.getSize();
        if (frameSize.height > screenSize.height) {
          frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
          frameSize.width = screenSize.width;
        }
        this.setLocation((screenSize.width - frameSize.width) / 2,
                                          (screenSize.height - frameSize.height) / 2);
        this.setVisible (true);
        this.setResizable(false);
  }

  // Create Visual Components
  private void initComponents() {

    ControlPanel = new javax.swing.JPanel();
    Save = new javax.swing.JButton();
    Exit = new javax.swing.JButton();
    LogPanel = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    TextArea1 = new javax.swing.JTextArea();
    StatusPanel = new javax.swing.JPanel();
    jScrollPane2 = new javax.swing.JScrollPane();
    TextArea2 = new javax.swing.JTextArea();
    VIClist = new javax.swing.JComboBox();
    VICAddMin = new javax.swing.JPanel();
    setName("VIC_Manager");
    setTitle("VIC Manager");
    addWindowListener(new java.awt.event.WindowAdapter() {
      public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
      }
    }
    );
```

```
Save.setText("Save LOG");
 Save.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
     SaveActionPerformed(evt);
   }
 }
 );
ControlPanel.add(Save);


Exit.setLabel("Exit");
 Exit.setName("Exit");
 Exit.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
     ExitActionPerformed(evt);
   }
 }
 );
ControlPanel.add(Exit);


getContentPane().add(ControlPanel, java.awt.BorderLayout.SOUTH);


LogPanel.setLayout(new java.awt.BorderLayout());
LogPanel.setPreferredSize(new java.awt.Dimension(320, 300));
LogPanel.setBorder(new javax.swing.border.TitledBorder("VIC Manager LOG"));


TextArea1.setEditable(false);
  jScrollPane1.setViewportView(TextArea1);
  LogPanel.add(jScrollPane1, java.awt.BorderLayout.CENTER);


getContentPane().add(LogPanel, java.awt.BorderLayout.WEST);


StatusPanel.setLayout(new java.awt.BorderLayout());
StatusPanel.setPreferredSize(new java.awt.Dimension(320, 300));
StatusPanel.setBorder(new javax.swing.border.TitledBorder("Status of VIC's"));


TextArea2.setEditable(false);
  jScrollPane2.setViewportView(TextArea2);
  StatusPanel.add(jScrollPane2, java.awt.BorderLayout.CENTER);


VIClist.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(java.awt.event.ActionEvent evt) {
            VIClistActionPerformed(evt);
       }
  }
  );

StatusPanel.add(VIClist, java.awt.BorderLayout.NORTH);
getContentPane().add(StatusPanel, java.awt.BorderLayout.EAST);


JTabbedPane tabbedPane = new JTabbedPane();

Component panel1 = addPanel();
tabbedPane.addTab("Add a new VIC", panel1);
tabbedPane.setSelectedIndex(0);
```

```
    Component panel2 = changePanel();
    tabbedPane.addTab("Change Password", panel2);

    Component panel3 = removePanel();
    tabbedPane.addTab("Remove a VIC", panel3);

     Component panel4 = aboutPanel("Tilan Niranjan Ukwatta - University of Colombo
     2001");

     tabbedPane.addTab("About VIC Manager", panel4);

    getContentPane().add(tabbedPane, BorderLayout.NORTH);
}


protected Component addPanel() {

   JPanel jPanel1;
   JLabel jLabel1;
   JLabel jLabel2;
   JButton Add;

    jPanel1 = new javax.swing.JPanel();
    jLabel1 = new javax.swing.JLabel();
    addName = new javax.swing.JTextField();
    jLabel2 = new javax.swing.JLabel();
    addPassword = new javax.swing.JPasswordField();
    Add = new javax.swing.JButton();

    jPanel1.setLayout(new java.awt.GridLayout(3, 2));
    jPanel1.setPreferredSize(new java.awt.Dimension(200, 75));
    jPanel1.setBorder(new javax.swing.border.TitledBorder("User Information"));

    jLabel1.setText("    Name :");
    jPanel1.add(jLabel1);

    jPanel1.add(addName);

    jLabel2.setText("    Password :");
    jPanel1.add(jLabel2);

    jPanel1.add(addPassword);


    Add.setText("Add Virtual Internet Computer");
     Add.addActionListener(new java.awt.event.ActionListener() {
       public void actionPerformed(java.awt.event.ActionEvent evt) {
          AddActionPerformed(evt);
       }
     }
     );
    jPanel1.add(Add);
    return jPanel1;
}

protected Component changePanel() {

         JPanel jPanel1;
         JLabel jLabel1;
         JLabel jLabel2;
         JButton Change;

    jPanel1 = new javax.swing.JPanel();
    jLabel1 = new javax.swing.JLabel();
    previousPassword = new javax.swing.JPasswordField();
    jLabel2 = new javax.swing.JLabel();
    newPassword = new javax.swing.JPasswordField();
    Change = new javax.swing.JButton();
    changeVIClist = new javax.swing.JComboBox();
```

```
    jPanel1.setLayout(new java.awt.GridLayout(3, 2));
    jPanel1.setPreferredSize(new java.awt.Dimension(200, 75));
    jPanel1.setBorder(new javax.swing.border.TitledBorder("Change User Password"));

    jPanel1.add(changeVIClist);

    Change.setText("Change Password");
     Change.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
           ChangeActionPerformed(evt);
        }
     }
     );
    jPanel1.add(Change);


    jLabel1.setText("    Previous Password :");
    jPanel1.add(jLabel1);

    jPanel1.add(previousPassword);

    jLabel2.setText("    New Password :");
    jPanel1.add(jLabel2);

    jPanel1.add(newPassword);

    return jPanel1;
  }

  protected Component removePanel() {

    JPanel jPanel1;
    JButton Remove;
    jPanel1 = new javax.swing.JPanel();
    Remove = new javax.swing.JButton();
    removeVIClist = new javax.swing.JComboBox();

    jPanel1.setLayout(new java.awt.GridLayout(3, 2));
    jPanel1.setPreferredSize(new java.awt.Dimension(200, 75));
    jPanel1.setBorder(new javax.swing.border.TitledBorder("Remove a VIC"));

    jPanel1.add(removeVIClist);

    Remove.setText("Remove Virtual Internet Computer");
     Remove.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
           RemoveActionPerformed(evt);
        }
     }
     );
    jPanel1.add(Remove);

    return jPanel1;
  }

  protected Component aboutPanel(String text) {
    JPanel panel = new JPanel(false);
    JLabel filler = new JLabel(text);
    filler.setHorizontalAlignment(JLabel.CENTER);
    panel.setLayout(new GridLayout(1, 1));
    panel.add(filler);
    return panel;
  }
```

```
private void VIClistActionPerformed(java.awt.event.ActionEvent evt) {

        String selectedUser;
        selectedUser = (String)VIClist.getSelectedItem();
        printStatus(selectedUser + " selected.");

}

private void AddActionPerformed(java.awt.event.ActionEvent evt) {

        String newUser, newPassword;
        newUser = addName.getText();
        newPassword = addPassword.getText();
        boolean userExist=false;
        String[] regBuffer = new String[MAX_VIC];
        int index =0;

          for (int i=0; i<MAX_VIC; i++) {
                  regBuffer[i]=" ";
          }
                int x=newUser.indexOf(" ");
                if ((x==-1) && newUser.length()>0) {

                  try {
                  File userFile = new File("users.txt");
                  BufferedReader regin = new BufferedReader(new FileReader(userFile));
                  String reginputLine;
                  while ((reginputLine = regin.readLine()) != null) {
                          regBuffer[index]=reginputLine;
                          index++;
                          // separate the user name
                          int y=reginputLine.indexOf(" ");
                          reginputLine=reginputLine.substring(0, y);
                          // check whether a user exists with the same name
                          if (reginputLine.equals(newUser)) {userExist=true;};
                  }
                  regin.close();
                  if (userExist) {
                  JOptionPane.showMessageDialog(VICAddMin,
                  "Another user exist with the same name!",
                                  "VIC Server", JOptionPane.ERROR_MESSAGE);
                  }
                  }
                  catch (IOException e) {
                  System.out.println("Can not open the user registration file!");
                  }
                  if (!userExist) {
                          regBuffer[index]= newUser + " " + newPassword;
                          File userFile = null;
                  File rootDir =null;

                  try {
                  userFile = new File("users.txt");
                  rootDir = new File(userFile.getAbsolutePath().substring(0,
                (userFile.getAbsolutePath().length()-4)) + rootDir.separator+newUser );
                  printLog(rootDir.getAbsolutePath());

                  // Create user root directory
                  if (!rootDir.mkdir()) {
                          System.out.println("User root directory can not be created!");
                  };

                  // Update the user.txt file
                  PrintWriter logout = new PrintWriter(new FileWriter(userFile));

                  for (int i=0; i<MAX_VIC; i++) {
                          int length=regBuffer[i].length();
                    if (length > 1) {
                          logout.println(regBuffer[i]);
                          }
                  }
```

```
                    logout.flush();
                    logout.close();
                    }
                    catch (IOException e) {
                    System.out.println("Can not write to log file!");
                    }
                            printLog(newUser + " added to the VIC system.");
                    }
                  } else {
                     JOptionPane.showMessageDialog(VICAddMin,"User name should not be
                           blank and should not contain spaces!",
                                  "VIC Server", JOptionPane.ERROR_MESSAGE);
                  };
          // clear text fields
          addName.selectAll();
          addName.replaceSelection("");
          addPassword.selectAll();
          addPassword.replaceSelection("");
          loadUsers();
  }

  private void ChangeActionPerformed(java.awt.event.ActionEvent evt) {

          String changeUser, oldPassword, nPassword;
          changeUser = (String)changeVIClist.getSelectedItem();
          oldPassword = previousPassword.getText();
          nPassword = newPassword.getText();
          int index=0;
          String[] regBuffer = new String[MAX_VIC];
            for (int i=0; i<MAX_VIC; i++) {
                    regBuffer[i]=" ";
            }
                            try {
                    File userFile = new File("users.txt");
                    BufferedReader regin = new BufferedReader(new FileReader(userFile));
                            String reginputLine;
                            while ((reginputLine = regin.readLine()) != null) {
                                    regBuffer[index]=reginputLine;
                                    index++;
                            }
                    regin.close();
                    for (int i=0; i<MAX_VIC; i++) {

                    // separate the user name
                    String uName;
                    int y=regBuffer[i].indexOf(" ");
                    uName=regBuffer[i].substring(0, y);
                    // select the user
                     if (uName.equals(changeUser)) {
                     String PrePassword=regBuffer[i].substring(y+1, regBuffer[i].length());
                     if (PrePassword.equals(oldPassword)) {
                            regBuffer[i]=changeUser + " " + nPassword;
                            printLog(changeUser + "'s password changed.");
                     } else {
                            JOptionPane.showMessageDialog(VICAddMin,
                                    "Previous password does not match!", "VIC Manager",
                                            JOptionPane.ERROR_MESSAGE);
                     };
                     };
                     };

                    // Update the user.txt file
                    PrintWriter logout = new PrintWriter(new FileWriter(userFile));

                    for (int i=0; i<MAX_VIC; i++) {
                            int length=regBuffer[i].length();
                      if (length > 1) {
                            logout.println(regBuffer[i]);
                            }
                    }
                            logout.flush();
```

```
                    logout.close();

                }
                catch (IOException e) {
                System.out.println("Can not open the user registration file!");
                }

        // clear text fields
        previousPassword.selectAll();
        previousPassword.replaceSelection("");
        newPassword.selectAll();
        newPassword.replaceSelection("");
        loadUsers();
}

private void RemoveActionPerformed(java.awt.event.ActionEvent evt) {

        String removeUser;
        removeUser = (String)removeVIClist.getSelectedItem();
        int index=0;
        String[] regBuffer = new String[MAX_VIC];
          for (int i=0; i<MAX_VIC; i++) {
                regBuffer[i]=" ";
          }
                try {
                File userFile = new File("users.txt");
                BufferedReader regin = new BufferedReader(new FileReader(userFile));
                        String reginputLine;
                                while ((reginputLine = regin.readLine()) != null) {
                                        regBuffer[index]=reginputLine;
                                        index++;
                                }
                regin.close();
                for (int i=0; i<MAX_VIC; i++) {

                // separate the user name
                String uName;
                int y=regBuffer[i].indexOf(" ");
                uName=regBuffer[i].substring(0, y);
                // select the user
                if (uName.equals(removeUser)) {int n =
                        JOptionPane.showConfirmDialog(VICAddMin,
                        "Do you want to remove " + removeUser + "'s VIC?",
                        "Confirm VIC Removal",
                        JOptionPane.YES_NO_OPTION);
                if (n==0) {     regBuffer[i]="";
                // implement code for removal of the user root directory
                printLog(removeUser + " removed.");
                };
                };
                };

                // Update the user.txt file
                PrintWriter logout = new PrintWriter(new FileWriter(userFile));

                for (int i=0; i<MAX_VIC; i++) {
                        int length=regBuffer[i].length();
                  if (length > 1) {
                        logout.println(regBuffer[i]);
                        }
                }
                        logout.flush();
                        logout.close();

                }
                catch (IOException e) {
                System.out.println("Can not open the user registration file!");
                }

        loadUsers();
}
```

```
private void ExitActionPerformed(java.awt.event.ActionEvent evt) {
   System.exit (0);
}

private void SaveActionPerformed(java.awt.event.ActionEvent evt) {

                    File logFile = null;
                    try {
                    logFile = new File("log.txt");
                  PrintWriter logout = new PrintWriter(new FileWriter(logFile));
                    String logText;
                    logout.println("VIC Manager LOG");
                    logout.println("==============");
                    logText = TextArea1.getText();
                    logout.println(logText);
                    logout.close();
                    }
                    catch (IOException e) {
                    System.out.println("Can not write to log file!");
                    }
}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) {
   System.exit (0);
}

private static void loadUsers () {
          try {
          File userFile = new File("users.txt");
          BufferedReader regin = new BufferedReader(new FileReader(userFile));
          String reginputLine;

          VIClist.removeAllItems();
           changeVIClist.removeAllItems();
           removeVIClist.removeAllItems();

          while ((reginputLine = regin.readLine()) != null) {
                  // separate the user name
                  int y=reginputLine.indexOf(" ");
                  reginputLine=reginputLine.substring(0, y);
                  VIClist.addItem(reginputLine);
                  changeVIClist.addItem(reginputLine);
                  removeVIClist.addItem(reginputLine);
          }

          regin.close();
          }
          catch (IOException e) {
          System.out.println("Can not open the user registration file!");
          }
        }

private static void login(Socket socket) {

        String username=null, userpassword=null;
        File rootDirectory;

        try {
        PrintStream StreamOut = new PrintStream(new
                                BufferedOutputStream(socket.getOutputStream()));
        BufferedInputStream StreamIn = new
                                BufferedInputStream(socket.getInputStream());
        BufferedReader in = new BufferedReader(new InputStreamReader(StreamIn));
        PrintWriter out = new PrintWriter(StreamOut, true);

        String inputLine, outputLine, user, password, port, UserPort=null;
```

```
                    user = in.readLine();
                      password = in.readLine();
                      rootDirectory  = new File("./vics/"+user);


                      File userFile = null;
                      boolean OK = false;
                      try {
                      userFile = new File("users.txt");
                      BufferedReader regin = new BufferedReader(new FileReader(userFile));
                      String reginputLine;
                      while ((reginputLine = regin.readLine()) != null) {
                              // separate the user name and password
                              int y=reginputLine.indexOf(" ");
                              username=reginputLine.substring(0, y);
                              reginputLine=reginputLine.substring(y+1, reginputLine.length());
                              y=reginputLine.indexOf(" ");
                              userpassword=reginputLine.substring(0, y);
                              port=reginputLine.substring(y+1, reginputLine.length());
                              // check the validity of user name and password
                            if (username.equals(user) && userpassword.equals(password) ) {
                                      OK =true;
                                      UserPort=port;
                              };
                       }
                       regin.close();
                       }
                        catch (IOException e) {
                        System.out.println("Can not open the user registration file!");
                        }

                    if (OK) {
                    ActivateVIC("java -cp ./vics/"+user+" VIC "+user+" "+password
                    +" "+UserPort+" "+ rootDirectory.getAbsolutePath());

                    printLog("VIC of " + user + " activated." );
                    takeTime(5);
                    out.println(UserPort);

                    }else{
                         out.println("InvalidLogin");
                         out.close();
                         in.close();
                         socket.close();
                    }
                    } catch (IOException e) {
                      e.printStackTrace();
                    }
            }

// this method waits for given number of seconds
public static void TakeTime(int sec) {
            Date date1=null, date2=null;
            int x=0,y=0;
            date1 = new Date();
            x=date1.getSeconds();
            while ((y-x) < sec) {
            date2 = new Date();
            y=date2.getSeconds();
            };
}

public static void printLog (String text) {

            date = new Date();
            text = text + "  " + date.toString();
            TextArea1.append(text + "\n");

}
```

```
public static void printStatus (String text) {

        TextArea2.selectAll();
        TextArea2.replaceSelection("");
        TextArea2.append(text + "\n");

}

public static void ActivateVIC (String ExternCommand) throws IOException        {

    try {
      Process p = Runtime.getRuntime().exec(ExternCommand);
    }
    catch (IOException e) {
      System.out.println("Can not activate the VIC.");
      e.printStackTrace();
      System.exit(-1);
    }
}


public static void main (String args[])  throws IOException {
    try {
       UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
       e.printStackTrace();
    }
    new VIC_Manager();
    ServerSocket serverSocket = null;
    boolean listening = true;

    try {
      serverSocket = new ServerSocket(1024);
    } catch (IOException e) {
      System.err.println("Could not listen on port: 1024.");
      System.exit(-1);
    }

    printLog("VIC Manager Started.");
    loadUsers();

    while (listening)
    {
      login(serverSocket.accept());
    }
    serverSocket.close();
 }
  // Variables declaration
  private final int MAX_VIC = 10;
  private javax.swing.JPanel ControlPanel;
  private javax.swing.JButton Save;
  private javax.swing.JButton Exit;
  private javax.swing.JPanel LogPanel;
  private javax.swing.JScrollPane jScrollPane1;
  private static javax.swing.JTextArea TextArea1;
  private javax.swing.JPanel StatusPanel;
  private javax.swing.JScrollPane jScrollPane2;
  private static javax.swing.JTextArea TextArea2;
  private static javax.swing.JComboBox VIClist;
  private javax.swing.JPanel VICAddMin;
  private javax.swing.JTextField addName;
  private javax.swing.JPasswordField addPassword;
  private javax.swing.JPasswordField previousPassword;
  private javax.swing.JPasswordField newPassword;
  private static javax.swing.JComboBox changeVIClist;
  private static javax.swing.JComboBox removeVIClist;
  private static java.util.Date date;
  // End of variables declaration
}
```

VIC.java

```
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.net.*;
import java.io.*;
import ViewPort;

/**
 * Title:       Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

public class VIC {

   private static Socket socket = null;
   private static File rootDirectory = null;
   private static File currentDirectory = null;
   private static String VICuser=null;
   private static String VICpassword=null;
   private static int port;
   private static BufferedReader in=null;
   private static PrintWriter out=null;
   private static boolean SHUTDOWN = true;

// --------- Start of Basic Command Method of the VIC --------------------------

public static void dir() {
   try {
   File  names[] =  new File[0];          // list of files in a directory
   String  filename;
   int ndir=0;
   int nfile=0;
   long nBytes=0;
   names = currentDirectory.listFiles();   // create list of files in this dir

   for (int i=0; i < names.length; i++) {
          String FileType;
          if (names[i].isDirectory()) {
          FileType = "DIR   \t";
          ndir++;
          } else {
          FileType = "    \t";
          nfile++;
          };
          nBytes = nBytes + names[i].length();
     filename = FileType + names[i].length() + " B ytes\t\t" + File.separator +
                                                              names[i].getName();
     System.out.println(filename);
   }

   System.out.println(" Files      :" + nfile);
   System.out.println(" Directories :" + ndir);
   System.out.println(" Total Bytes :" + nBytes);

   } catch (Exception e) {
   System.out.println("Command can not be executed!");
   }
}

public static void ver() {
   System.out.println("VIC OS Version 1.0");
}
```

```
public static void pcd() {
   int x;
   String currDir;
   currDir = currentDirectory.getAbsolutePath();
   x=currDir.indexOf(" \\" + VICuser);
   currDir = currDir.substring(x + VICuser.length() + 1);
   System.out.println("Current Directory: " + currDir);
}

public static void mem() {
   System.out.println("Total Memory: " + Runtime.getRuntime().totalMemory()/1024 +
                                                                          "kBytes" );
   System.out.println("Free  Memory: " + Runtime.getRuntime().freeMemory()/1024 +
                                                                          "kBytes" );
}

public static void help() {
   System.out.println("Command            Description");
   System.out.println("========  ============");
   System.out.println("attrib        Display and set file attributes.");
   System.out.println("cd      Changes the current directory.");
   System.out.println("copy          Copies files");
   System.out.println("cls     Clears the screen of the Virtual Internet Computer.");
   System.out.println("comp          Compares designated files.");
   System.out.println("dir     Lists a directory of filenames on the current directory.");
   System.out.println("delete    Delete files or directories.");
   System.out.println("find          Finds and displays files.");
   System.out.println("help          Displays information about the named commands.");
   System.out.println("get           Copies file from the VIC to the client device.");
   System.out.println("logoff        Log out from the VIC.");
   System.out.println("mem     Display the amount of used and available memory.");
   System.out.println("move          Moves files or directories to another directory.");
   System.out.println("md      Creates a new directory.");
   System.out.println("pcd     Print the current directory.");
   System.out.println("ren     Rename an existing file.");
   System.out.println("shutdown  Shutdown the Virtual Internet Computer.");
   System.out.println("type          Display a disignated file on the screen.");
   System.out.println("ver     Display the version of the Virtual Internet Computer being
                                                                          used.");
}

public static void cd(String file) {
   File changedDirectory = null;
   file=file.substring(2).trim();
            if (file.length()!=0) {
            if (file.equals("..")) {
            // Prevent going to unauthorized directories
            if (!rootDirectory.getName().equals(currentDirectory.getName())) {
                                    currentDirectory = currentDirectory.getParentFile();}
            } else {
                     changedDirectory = new File(currentDirectory, file);
                     if (changedDirectory.isDirectory()) {
                     currentDirectory = changedDirectory;
            } else {
                     System.out.println(changedDirectory.getName() + " is not a directory.");
            }
            }
            } else {
                  System.out.println("Command can not be executed!");
            }
         pcd();
}
```

```
public static void attrib(String file) {

  File   path;
  file=file.substring(6).trim();
  if (file.length()!=0) {
  path  = new File(currentDirectory, file);     // grab command line argument

  String exists   = path.exists()   ? "Yes" : "No";
  String canRead  = path.canRead()  ? "Yes" : "No";
  String canWrite = path.canWrite() ? "Yes" : "No";
  String isFile   = path.isFile()   ? "Yes" : "No";
  String isDir    = path.isDirectory()? "Yes" : "No";

  System.out.println("File attributes for '" + file + "'");

  System.out.println("Exists      : " + exists);
  if (path.exists()) {
    System.out.println("Readable    : " + canRead);
    System.out.println("Writable    : " + canWrite);
    System.out.println("Is directory  : " + isDir);
    System.out.println("Is file     : " + isFile);
  }
  } else {
        System.out.println("Command can not be executed!");
  }
}

public static void md(String file) {
  File newDirectory = null;
  file=file.substring(2).trim();
  if (file.length()!=0) {
  newDirectory = new File(currentDirectory, file);
  if (!newDirectory.mkdir()) {
  System.out.println("Directory can not be created!");
  }
  } else {
  System.out.println("Command can not be executed!");
  }
}

public static void delete(String file) {
  File newFile = null;
  file=file.substring(6).trim();
  if (file.length()!=0) {
  newFile = new File(currentDirectory, file);
  if (!newFile.delete()) {
        System.out.println("File or Directory does not exist or can not be deleted!");
  }
  } else {
  System.out.println("Command can not be executed!");
  }
}

public static boolean find(String fileName, String searchPath) {
   File  names[] = new File[0];    // list of files in a directory
  File  path;
  boolean found=false;
  String file;
  file=fileName.substring(4).trim();
  path  = new File(searchPath);
  names = path.listFiles();      // create list of files in this dir
  for (int i=0; i < names.length; i++) {
        if (names[i].getName().equals(file)) {
              found = true;
              System.out.println(names[i].getAbsolutePath());
        }
  }
  if (found) {return(true);} else {
      return(false);
  };
}
```

```
public static void type(String file) {

  File newFile = null;
  file=file.substring(4).trim();
  if (file.length()!=0) {
  try {
  newFile = new File(currentDirectory, file);
  if (newFile.isFile()) {
            BufferedReader in = new BufferedReader(new FileReader(newFile));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                  System.out.println(inputLine);
            }
  } else {
        System.out.println("Can not display!");
  }
  }
  catch (IOException e) {
  System.out.println("Can not display the file!");
  }
  } else {
        System.out.println("Command can not be executed!");
  }
}

public static void ren(String file) {
        file = file.trim() + "    ";
        String oldName, newName;
        int x;
        x=file.indexOf(' ');
        file=file.substring(x+1).trim();
        if (file.length()!=0) {
          file = file.trim() + "    ";
          x=file.indexOf(' ');
          oldName=file.substring(0, x);
          newName=file.substring(x+1).trim();
          if (newName.length()!=0) {
          File oldFile = new File(currentDirectory, oldName);
          File newFile = new File(currentDirectory, newName);
        if (!oldFile.renameTo(newFile)) {
          System.out.println("Can not rename the file or directory!");
        }
        } else {
          System.out.println("Command can not be executed!");
        }
        } else {
          System.out.println("Command can not be executed!");
        }
}

public static void move(String file) {
        file = file.trim() + "    ";
        String oldName, newName;
        int x;
        x=file.indexOf(' ');
        file=file.substring(x+1).trim();
        if (file.length()!=0) {
        file = file.trim() + "    ";
        x=file.indexOf(' ');
        oldName=file.substring(0, x);
        newName=file.substring(x+1).trim();
        if (newName.length()!=0) {
        File oldFile = new File(currentDirectory, oldName);
        File newFile = new File(currentDirectory, newName);
        try {
        BufferedInputStream source =new BufferedInputStream(new
                                                    FileInputStream(oldFile));
        PrintStream sink =new PrintStream(new BufferedOutputStream(new
                                                    FileOutputStream(newFile)));
        int ch;
```

```
            while ((ch=source.read())!= -1){
                    sink.write(ch);
            }
            source.close();
            sink.close();
            } catch (IOException e) {
                    System.out.println("File can not be copied!");
            }
            if (!oldFile.delete()) {
                    System.out.println("File or Directory does not exist or can not be moved!");
            }
            } else {
                    System.out.println("Command can not be executed!");
            }
            } else {
                    System.out.println("Command can not be executed!");
            }
}

public static void comp(String file) {
            file = file.trim() + "    ";
            String oldName, newName;
            int x;
            x=file.indexOf(' ');
            file=file.substring(x+1).trim();
              if (file.length()!=0) {
                  file = file.trim() + "    ";
                    x=file.indexOf(' ');
                    oldName=file.substring(0, x);
                    newName=file.substring(x+1).trim();
                    if (newName.length()!=0) {
                    File oldFile = new File(currentDirectory, oldName);
                    File newFile = new File(currentDirectory, newName);
                    if (oldFile.equals(newFile)) {
                            System.out.println("Files are identical. ");
                    }
                    else {
                            System.out.println("Files are different.");
                    }
                    } else {
                            System.out.println("Command can not be executed!");
                    }
                    } else {
                            System.out.println("Command can not be executed!");
                    }
}

public static void copy(String file) {
            file = file.trim() + "    ";
            String oldName, newName;
            int x;
            x=file.indexOf(' ');
            file=file.substring(x+1).trim();
            if (file.length()!=0) {
            file = file.trim() + "    ";
                    x=file.indexOf(' ');
                    oldName=file.substring(0, x);
                    newName=file.substring(x+1).trim();
            if (newName.length()!=0) {
                    File oldFile = new File(currentDirectory, oldName);
                    File newFile = new File(currentDirectory, newName);
           try {
            BufferedInputStream source =new BufferedInputStream(new
                                                            FileInputStream(oldFile));
            PrintStream sink =new PrintStream(new BufferedOutputStream(new
                                                        FileOutputStream(newFile)));
```

```
            int ch;
            while ((ch=source.read())!= -1){
                    sink.write(ch);
            }
            source.close();
            sink.close();
            } catch (IOException e) {
                    System.out.println("File can not be copied!");
            }
            } else {
                    System.out.println("Command can not be executed!");
            }
            } else {
                    System.out.println("Command can not be executed!");
            }
}

// --------- End of Basic Command Method of the VIC -------------------------

// Transfer file from VIC to the client device

private static void getFile(String newFileName) {

            ServerSocket gSocket = null;
            Socket getSocket = null;
            newFileName=newFileName.substring(3).trim();

            try {
                gSocket = new ServerSocket(port);
              getSocket = gSocket.accept();
              gSocket.close();
           } catch (IOException e) {
           System.err.println("Could not listen on port: " + port);
           System.exit(-1);
           };

              try {
              File newFile = new File(currentDirectory, newFileName);
              BufferedInputStream FileIn =new BufferedInputStream(new
                                                          FileInputStream(newFile));
              PrintStream sink =new PrintStream(new
                                    BufferedOutputStream(getSocket.getOutputStream()));
              int ch;
              while ((ch=FileIn.read())!= -1){
                      sink.write(ch);
              }
                      sink.close();
                      FileIn.close();
              } catch (IOException e) {
                      System.out.println("File can not be copied!");
                e.printStackTrace();
              }
              System.out.println("File transfer is complete.");
}

// Transfer file from the client device to VIC
private static void sendFile(String newFileName) {

     ServerSocket gSocket = null;
     Socket getSocket = null;
     newFileName=newFileName.substring(4).trim();

     try {
        gSocket = new ServerSocket(port);
        getSocket = gSocket.accept();
        gSocket.close();
     } catch (IOException e) {
        System.err.println("Could not listen on port: " + port);
        System.exit(-1);
     };
```

```
    try {
    BufferedInputStream FileIn = new
    BufferedInputStream(getSocket.getInputStream());
    File newFile = new File(currentDirectory, newFileName);
    PrintStream sink =new PrintStream(new BufferedOutputStream(new
                                                    FileOutputStream(newFile)));
    int ch;
        while ((ch=FileIn.read())!= -1){
        sink.write(ch);
        }
        sink.close();
        FileIn.close();
        } catch (IOException e) {
        System.out.println("File can not be copied!");
        e.printStackTrace();
        }
    System.out.println("File transfer is complete.");
}

// Take commands and return a unique number
public static int select(String command) {

        command = command + "      ";
        if (command.substring(0,3).equals("dir")) {
        return(1);
        };
        if (command.substring(0,2).equals("cd")){
        return(2);
        };
        if (command.substring(0,2).equals("md")){
        return(3);
        };
        if (command.substring(0,3).equals("pcd")){
        return(4);
        };
        if (command.substring(0,6).equals("attrib")){
        return(5);
        };
        if (command.substring(0,4).equals("copy")){
        return(6);
        };
        if (command.substring(0,6).equals("delete")){
        return(7);
        };
        if (command.substring(0,4).equals("find")){
        return(8);
        };
        if (command.substring(0,4).equals("comp")){
        return(9);
        };
        if (command.substring(0,4).equals("help")){
        return(10);
        };
        if (command.substring(0,3).equals(" mem")){
        return(11);
        };
        if (command.substring(0,4).equals("move")){
        return(12);
        };
        if (command.substring(0,3).equals("ren")){
        return(13);
        };
        if (command.substring(0,4).equals("type")){
        return(14);
        };
        if (command.substring(0,3).equals("ver")){
        return(15);
        };
        if (command.substring(0,3).equals("get")){
        return(16);
        };
```

```
        if (command.substring(0,4).equals("send")){
        return(17);
        } else {

                command = command.trim() + "   ";
                int x;
                x=command.indexOf(' ');
                command=command.substring(0, x).trim();
                if (command.length()!=0) {
                        // check whether the .class file exist or not
                        command=command + ".class";
                File  names[] = new File[0];
                boolean found=false;
                names = currentDirectory.listFiles();
                        for (int i=0; i < names.length; i++) {
                                if (names[i].getName().equals(command)) {
                                found = true;
                                }
                          }
                        if (found) {
                        return(18);
                        }
                        else {
                        return(0);
                        };
                } else {
          return(0);
        }
  }
}


private static void login(Socket socket) {

    try {
    PrintStream StreamOut = new PrintStream(new
                                BufferedOutputStream(socket.getOutputStream()));
    BufferedInputStream StreamIn = new
                                BufferedInputStream(socket.getInputStream());
    System.setIn(StreamIn);
    System.setOut(StreamOut);
    System.setErr(StreamOut);
    in = new BufferedReader(new InputStreamReader(System.in));
    out = new PrintWriter(System.out, true);

    String inputLine, outputLine, user, password;

    user = in.readLine();
    password = in.readLine();

    boolean OK = false;
    if (user.equals(VICuser) && password.equals(VICpassword) ) { OK =true; };

                if (OK) {
                  out.println("ValidLogin");
                  outputLine = "Welcome to your Virtual Internet Computer";
                  out.println(outputLine);
                  ViewPort.print("User: " + user);
                  out.println("User: " + user);
                  out.println("end");
                }else{
                  out.println("InvalidLogin");
                  out.close();
                  in.close();
                  socket.close();
                }
                } catch (IOException e) {
                  e.printStackTrace();
                }
    }
```

```
private static void logoff() {
        try {
        socket.close();
        } catch (IOException e) {
                System.err.println("Could not close the connection.");
        };
}

private static void listen() {

ServerSocket serverSocket = null;
    try {
      serverSocket = new ServerSocket(port);
    } catch (IOException e) {
      System.err.println("Could not listen on port: " + port);
      System.exit(-1);
    };

    try {
        login(serverSocket.accept());
        serverSocket.close();
    } catch (IOException e) {
        System.err.println("Could not accept the connection.");
    };
}

private static void process() {

          try {
          String inputLine, outputLine;

          // NatuIral language protocaol is natural language processor
          // that enable users to enter commands in natural langage.
          // This class was not fully implemented in this project.
          NaturalLanguageProtocol nlp = new NaturalLanguageProtocol();

          while ((inputLine = in.readLine()) != null) {
                outputLine = nlp.processInput(inputLine);
                ViewPort.print("Command: " + outputLine);
                if (outputLine.equals("shutdown")){
                  SHUTDOWN=true;
                  break;
                };

                if (outputLine.equals(" logoff")){
                          logoff();
                } else {;
               switch( select(outputLine)) {
                  case 0 :
                  {
                        System.out.println("Bad command or file name.");
                   };
                  ; break;
                  case 1 : dir(); break;
                  case 2 : cd(outputLine); break;
                  case 3 : md(outputLine); break;
                  case 4 : pcd(); break;
                  case 5 : attrib(outputLine); break;
                  case 6 : copy(outputLine); break;
                  case 7 : delete(outputLine); break;
                  case 8 : {
                          if (!find(outputLine, currentDirectory.getAbsolutePath())) {
                          System.out.println("File or directory can not be found.");}
                          }; break;
                  case 9 : comp(outputLine); break;
                  case 10 : help(); break;
                  case 11 : mem(); break;
                  case 12 : move(outputLine); break;
                  case 13 : ren(outputLine); break;
                  case 14 : type(outputLine); break;
                  case 15 : ver(); break;
```

```
                    case 16 : getFile(outputLine); break;
                    case 17 : sendFile(outputLine); break;
                    case 18 : {

                            outputLine = outputLine.trim() + "   ";
                            int x;
                            x=outputLine.lastIndexOf(' ');
                            String check=outputLine.substring(x).trim();

                            // check to see whether the command is to execute in
                            // background.
                            if (check.equals("bg")) {
                            new ExecutionBackGround(currentDirectory, outputLine).start();
                            } else {
                            new Execution(currentDirectory, outputLine).exec();
                            }
                    }; break;
                    default:{};break;
                  };
                  out.println("end");
                  };
                  };
                  } catch (IOException e) {
                            System.err.println("Could not accept commands.");
                  }
}


public static void main(String args[]) {

                  String rootPath=null;
                  VICuser = args[0];
                  VICpassword = args[1];
                  port = Integer.parseInt(args[2]);
                  rootPath = args[3];
                  rootPath = rootPath + "\\" + VICuser;

                  rootDirectory = new File(rootPath);
                  currentDirectory = rootDirectory;
                  SHUTDOWN=false;

                  ViewPort vp;
                 vp = new ViewPort ();
                  vp.show();

                  while (!SHUTDOWN) {
                  listen();
                  process();
                  };

                  if (SHUTDOWN) {
                  System.exit(0);
                  }
        }
}
```

Execution.java

```
import java.net.*;
import java.io.*;
import java.lang.*;
import java.lang.reflect.*;

/**
 * Title:       Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

// This class is used to execute class files.

public class Execution {
        File directory=null;
        String file=null;
        Class CommandClass=null;
        Object Command=null;

  public Execution(File CDir, String Comm) {
                this.directory = CDir;
                this.file=Comm;
  }

public void exec() {

          try {

                ClassLoad cl =  new ClassLoad(directory);
                CommandClass = cl.loadClass(file);
                Command = CommandClass.newInstance();

          Class c = Command.getClass();
          Class[] args=null;
          // invoke the main() method of the loaded class
          Method m = c.getMethod("main", args);
          m.invoke(Command, args);

          }
          catch (NoSuchMethodException e) {
               System.out.println("NoSuchMethodException occured.");
          }
          catch (InvocationTargetException e) {
                System.out.println("InvocationTargetException occured.");
          }
          catch (InstantiationException e) {
                System.out.println("Instantiation Exception occured.");
          }
           catch (IllegalAccessException e) {
                System.out.println("Illegal Access Exception occured.");
           }
           catch (Exception e) {
                System.out.println("File can not be executed.");
           }

  }
}
```

ExecutionBackGround.java

```
import java.net.*;
import java.io.*;
import java.lang.*;
import java.lang.reflect.*;

/**
 * Title:      Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

// This class is used to execute class files
// in the background.

public class ExecutionBackGround extends Thread {
        File directory=null;
        String file=null;
        Class CommandClass=null;
        Object Command=null;

  public ExecutionBackGround(File CDir, String Comm) {
                super("ExecutionBackGround");
                this.directory = CDir;
                this.file=Comm;
  }

  public void run() {

          try {

                ClassLoad cl = new ClassLoad(directory);
                CommandClass = cl.loadClass(file);
                Command = CommandClass.newInstance();

          Class c = Command.getClass();
          Class[] args=null;
          // invoke the main() method of the loaded class
          Method m = c.getMethod("main", args);
          m.invoke(Command, args);

          }
          catch (NoSuchMethodException e) {
          System.out.println("NoSuchMethodException occured.");
          }
          catch (InvocationTargetException e) {
          System.out.println("InvocationTargetException occured.");
          }
          catch (InstantiationException e) {
          System.out.println("Instantiation Exception occured.");
          }
          catch (IllegalAccessException e) {
          System.out.println("Illegal Access Exception occured.");
          }
          catch (Exception e) {
          System.out.println("File can not be executed.");
          }

        }
}
```

ViewPort.java

```
import javax.swing.*;
import java.util.*;

/**
 * Title:      Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

// This is the View Port, which displays what the user is
// doing with time and date.

public class ViewPort extends javax.swing.JFrame {

// Creates new form ViewPort
public ViewPort() {
    initComponents ();
    pack ();
}

// Creates Visual Components
private void initComponents() {
    jScrollPane1 = new javax.swing.JScrollPane();
    TextArea = new javax.swing.JTextArea();
    setName("VIC");
    setTitle("VIC");
    addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
      }
    }
    );

    jScrollPane1.setPreferredSize(new java.awt.Dimension(320, 240));
     TextArea.setEditable(false);
     TextArea.setText("Virtual Internet Computer");
     jScrollPane1.setViewportView(TextArea);

    getContentPane().add(jScrollPane1, java.awt.BorderLayout.CENTER);
}

public static void print (String text) {
                    date = new Date();
                    text = text + "   " + date.toString();
                    TextArea.append("\n" + text);
}

private void exitForm(java.awt.event.WindowEvent evt) {
    System.exit (0);
}
  // Variables declaration
  private javax.swing.JScrollPane jScrollPane1;
  private static javax.swing.JTextArea TextArea;
  private static java.util.Date date;
  // End of variables declaration
}
```

ClassLoad.java

```
import java.net.*;
import java.io.*;
import java.lang.*;

/**
 * Title:       Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

// This a user defined class loader

public class ClassLoad extends ClassLoader{

        File directory=null;
        Class execute=null;

public ClassLoad(File CDir) {
        this.directory = CDir;
}

public synchronized Class loadClass(String Name) {
                Class c = findLoadedClass(Name);
                if (c != null) return c;
                try {
                        c = findSystemClass(Name);
                        return c;
                } catch (ClassNotFoundException e) {
                }
                try {
                        byte[] data = getClassData(directory, Name);
                        return defineClass(Name, data, 0, data.length);
                } catch (IOException e) {
                }
                return(null);
}

private byte[] getClassData(File dir, String file) throws IOException {

        file = file  + ".class";
        File classFile = new File(dir, file);
        int ln = (int)classFile.length();
        byte[] fileArray = new byte[ln];
        BufferedInputStream source = new BufferedInputStream(new
        FileInputStream(classFile));
        for (int i=0; i<ln; i++) {
        int ch;
        ch=source.read();
        fileArray[i]=(byte)ch;
        };
        source.close();
        return(fileArray);
        };

}
```

Web_Client.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

/**
 * Title:      Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

public class VIC_Client extends javax.swing.JFrame {

   Socket wcSocket = null;
   PrintWriter out = null;
   BufferedReader in = null;
   String portFromServer=null;

// Creates new form VIC_Client
public VIC_Client() {
      initComponents ();
      pack ();

     //Center the window
     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
     Dimension frameSize = this.getSize();
     if (frameSize.height > screenSize.height) {
                    frameSize.height = screenSize.height;
     }
     if (frameSize.width > screenSize.width) {
                    frameSize.width = screenSize.width;
     }

     this.setLocation((screenSize.width - frameSize.width) / 2,
                                                  (screenSize.height - frameSize.height) / 2);
     this.setVisible(true);

     LoginPanel.setVisible(true);
     CommandField.setText("Not Connected.");
     CommandField.disable();
     send.setVisible(false);
}

// Perform the dual connection login operation.
private void login() {

      String UserName=null;
      String Password=null;
      int port=0;
      UserName = user.getText();
      Password = password.getText();

      // Send user information to the VIC Manager
      out.println(UserName);
      out.println(Password);

      try {
       // Get the port, which VIC listens.
       portFromServer = in.readLine();
       if (!portFromServer.equals("ValidLogin")) {
       this.logoff();
       };
       out.close();
                    in.close();
      } catch (IOException e) {
```

```
            System.err.println("Couldn't get I/O for the connection to Server.");
            System.exit(1);
        }

        // Make a connection with the VIC
        try {
            wcSocket = new Socket("192.248.16.204", Integer.parseInt(portFromServer));
            out = new PrintWriter(wcSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: 192.248.16.204.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to: 192.248.16.204.");
            System.exit(1);
        }

        // Send user information to the VIC
        out.println(UserName);
        out.println(Password);
        LoginPanel.setVisible(false);
        CommandField.enable();
        send.setVisible(true);
        CommandField.setText("Command:\\ ");
        CommandField.requestFocus();
    }

    // Clear the User View Area
    private void cls() {
        TextMode.selectAll();
        TextMode.replaceSelection("");
    }

    private void logoff() {
        LoginPanel.setVisible(true);
        user.setText(null);
        password.setText(null);
        CommandField.setText("Not Connected.");
        CommandField.disable();
        send.setVisible(false);
        this.cls();
        user.requestFocus();
    }

    // Create graphical components
    private void initComponents() {
        LoginPanel = new java.awt.Panel();
        User = new javax.swing.JLabel();
        user = new javax.swing.JTextField();
        Password = new javax.swing.JLabel();
        password = new javax.swing.JPasswordField();
        login = new javax.swing.JButton();
        UtilityPanel = new java.awt.Panel();
        status = new javax.swing.JLabel();
        send = new javax.swing.JButton();
        VICpanel = new java.awt.Panel();
        CommandField = new javax.swing.JTextField();
        WorkArea = new javax.swing.JScrollPane();
        TextMode = new javax.swing.JTextArea();
        setName("VIC_Client");
        setTitle("Virtual Internet Computer");

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        }
        );
```

```
LoginPanel.setFont(new java.awt.Font ("Dialog", 0, 11));
LoginPanel.setBackground(new java.awt.Color (204, 204, 204));
LoginPanel.setForeground(java.awt.Color.black);

User.setName("User");
 User.setText("User");
 LoginPanel.add(User);


user.setPreferredSize(new java.awt.Dimension(100, 20));
 user.setName("user");
 LoginPanel.add(user);


Password.setName("Password");
 Password.setText("Password");
 LoginPanel.add(Password);


password.setPreferredSize(new java.awt.Dimension(100, 20));
 LoginPanel.add(password);


login.setName("login");
 login.setText("Login");
 login.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       loginActionPerformed(evt);
    }
 }
 );
LoginPanel.add(login);


getContentPane().add(LoginPanel, java.awt.BorderLayout.NORTH);


UtilityPanel.setFont(new java.awt.Font ("Dialog", 0, 11));
UtilityPanel.setBackground(new java.awt.Color (204, 204, 204));
UtilityPanel.setForeground(java.awt.Color.black);

status.setName("status");
 status.setText("Virtual Internet Computer");
 UtilityPanel.add(status);


send.setText("Send File");
 send.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       sendActionPerformed(evt);
    }
 }
 );
UtilityPanel.add(send);
getContentPane().add(UtilityPanel, java.awt.BorderLayout.SOUTH);

VICpanel.setLayout(new java.awt.BorderLayout());
VICpanel.setFont(new java.awt.Font  ("Dialog", 0, 11));
VICpanel.setName("panel5");
VICpanel.setBackground(new java.awt.Color (204, 204, 204));
VICpanel.setForeground(java.awt.Color.black);

CommandField.setFont(new java.awt.Font ("SansSerif", 1, 14));
 CommandField.setText("Not Connected.");
 CommandField.setBackground(java.awt.Color.yellow);
 CommandField.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       CommandFieldActionPerformed(evt);
    }
 }
 );
```

```
        VICpanel.add(CommandField, java.awt.BorderLayout.SOUTH);


        WorkArea.setPreferredSize(new java.awt.Dimension(320, 240));

         TextMode.setEditable(false);
          TextMode.setSelectedTextColor(java.awt.Color.yellow);
          TextMode.setForeground(java.awt.Color.white);
          TextMode.setFont(new java.awt.Font ("Monospaced", 0, 14));
          TextMode.setBackground(java.awt.Color.black);
          WorkArea.setViewportView(TextMode);

          VICpanel.add(WorkArea, java.awt.BorderLayout.CENTER);


        getContentPane().add(VICpanel, java.awt.BorderLayout.CENTER);

}

// Transfer files from VIC to the client device
private void getActionPerformed() {
    String source=null;
    JFileChooser fc = new JFileChooser();
    File getFile=null;
    Socket fgSocket = null;

    int returnVal = fc.showDialog(UtilityPanel, "Save");

    if (returnVal == JFileChooser.APPROVE_OPTION) {
        getFile = fc.getSelectedFile();
        source=getFile.getAbsolutePath();
    }
    if (returnVal == JFileChooser.CANCEL_OPTION) {
        source=null;
    }
    if (source!=null) {

                // Make separate connection with the VIC to transfer files.
                try {
                  fgSocket = new Socket("192.248.16.204", Integer.parseInt(portFromServer));
                } catch (UnknownHostException e) {
                  System.err.println("Don't know about host: 192.248.16.204.");
                  System.exit(1);
                } catch (IOException e) {
                  System.err.println("Couldn't get I/O for the connection to: 192.248.16.204.");
                  System.exit(1);
                }

                try {
                BufferedInputStream sendStream =new
                                        BufferedInputStream(fgSocket.getInputStream());
                PrintStream sink =new PrintStream(new BufferedOutputStream(new
                                        FileOutputStream(getFile)));
                // file transfering take place
                int ch;
                while ((ch=sendStream.read())!= -1){
                        sink.write(ch);
                        }
                        sink.close();
                        sendStream.close();
                } catch (IOException e) {
                  System.err.println("I/O Exception.");
                  System.exit(1);
                };
    };
  }
```

```
// This method waits for given number of seconds
public static void TakeTime(int sec) {
                    Date date1=null, date2=null;
                    int x=0,y=0;
                    date1 = new Date();
                    x=date1.getSeconds();
                    while ((y- x) < sec) {
                              date2 = new Date();
                              y=date2.getSeconds();
                    };
}

// Transfer files from client device to the VIC
private void sendActionPerformed(java.awt.event.ActionEvent evt) {
   String source=null;
   String fileName=null;
   JFileChooser fc = new JFileChooser();
          File sendFile=null;
          Socket fsSocket = null;

   int returnVal = fc.showDialog(UtilityPanel, "Send");

   if (returnVal == JFileChooser.APPROVE_OPTION) {
     sendFile = fc.getSelectedFile();
     source=sendFile.getAbsolutePath();
   }
   if (returnVal == JFileChooser.CANCEL_OPTION) {
     source=null;
   }

   int x;
   x = source.lastIndexOf('\\');
          fileName=source.substring(x+1).trim();

   if (source!=null) {
     TextMode.append("\n" + source);
     TextMode.append("\n" + fileName);

     out.println("send " + fileName);

     // wait for the VIC to be ready
     TakeTime(3);

     // Make separate connection with the VIC to transfer files.
          try {
            fsSocket = new Socket("192.248.16.204", Integer.parseInt(portFromServer));
          } catch (UnknownHostException e) {
            System.err.println("Don't know about host: 192.248.16.204.");
            System.exit(1);
          } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to: 192.248.16.204.");
            System.exit(1);
          }
          try {
          BufferedInputStream sendStream =new BufferedInputStream(new
                                                        FileInputStream(sendFile));
          PrintStream sink =new PrintStream(new
                              BufferedOutputStream(fsSocket.getOutputStream()));
          // transfer files
          int ch;
          while ((ch=sendStream.read())!= -1){
          sink.write(ch);
          }
          sink.close();
          sendStream.close();
          } catch (IOException e) {
            System.err.println("I/O Exception.");
            System.exit(1);
          };
   };
 }
```

```
 / / Executed when the user press the login button.
 private void loginActionPerformed(java.awt.event.ActionEvent evt) {

     / / Make connection with the VIC Manager application
     / / which is located in 192.248.16.204
     try {
        wcSocket = new Socket("192.248.16.204", 1024);
        out = new PrintWriter(wcSocket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));
     } catch (UnknownHostException e) {
        System.err.println("Don't know about host: 192.248.16.204.");
        System.exit(1);
     } catch (IOException e) {
        System.err.println("Couldn't get I/O for the connection to: 192.248.16.204.");
        System.exit(1);
     }
     this.login();
     String fromServer;
     try {
      / / Read out from the server and display it on the text area.
      fromServer = in.readLine();
      if (!fromServer.equals("ValidLogin")) {
      this.logoff();
      };
      while ((fromServer = in.readLine()) != null) {
        if (fromServer.equals("end"))
                    break;
        TextMode.append(" \n" + fromServer);
      }
     } catch (IOException e) {
        System.err.println("Couldn't get I/O for the connection to Server.");
        System.exit(1);
     }

}

/ / Executed when the user enters a coomand in the command area.
private void CommandFieldActionPerformed(java.awt.event.ActionEvent evt) {

    String command;
    String fromServer;
    / / get the command when press enter key
    command=CommandField.getText();

    / / check whether length of the command is greater than zero.
    if (command.length()>10) {
    TextMode.append(" \n" + command);
    command=command.substring(10);
    if (command.equals("cls")){
    this.cls();
    } else if (command.length()!=0) {

    / / Send the user comand to the VIC
    out.println(command);

    / / if the command is get file then start
    / / file transfering process
    if (command.substring(0,3).equals("get")){
                this.getActionPerformed();
    };


    if (command.equals("logoff") || command.equals("shutdown")){
    / / close the connection if the command is logoff or shutdown
     try {
        this.logoff();
        out.close();
        in.close();
        wcSocket.close();
     } catch (IOException e) {
```

```
            System.err.println("Couldn't get I/O for the connection to Server.");
            System.exit(1);
          }
        } else {

                    try {
                     // Read out from the server and display it on the user view area.
                     while ((fromServer = in.readLine()) != null) {
                            if (fromServer.equals("end"))
                            break;
                      TextMode.append("\n" + fromServer);
                     }
                    } catch (IOException e) {
                      System.err.println("Couldn't get I/O for the connection to Server.");
                      System.exit(1);
                    }
                    TextMode.append("\n");
                   }
                 };
        };
        // reset the command area
        CommandField.setText("Command: \ ");

}

// Exit the Application
private void exitForm(java.awt.event.WindowEvent evt) {
     System.exit (0);
}

public static void main (String args[]) {
     new VIC_Client ().show ();
}

  private java.awt.Panel LoginPanel;
  private javax.swing.JLabel User;
  private javax.swing.JTextField user;
  private javax.swing.JLabel Password;
  private javax.swing.JPasswordField password;
  private javax.swing.JButton login;
  private java.awt.Panel UtilityPanel;
  private javax.swing.JLabel status;
  private javax.swing.JButton send;
  private java.awt.Panel VICpanel;
  private javax.swing.JTextField CommandField;
  private javax.swing.JScrollPane WorkArea;
  private javax.swing.JTextArea TextMode;

}
```

WebApplet.java

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

/**
 * Title:       Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

// Client Java Applet

public class WebApplet extends Applet  {

   Socket wcSocket = null;
   PrintWriter out = null;
   BufferedReader in = null;
   String portFromServer=null;


// Creates new form WebApplet
public WebApplet() {
      initComponents ();
      LoginPanel.setVisible(true);
      password.setEchoChar('*');
      CommandField.setText("Not Connected.");
      CommandField.disable();
}

// This method waits for given number of seconds
public static void TakeTime(int sec) {
                    Date date1=null, date2=null;
                    int x=0,y=0;
                    date1 = new Date();
                    x=date1.getSeconds();
                    while ((y - x) < sec) {
                    date2 = new Date();
                    y=date2.getSeconds();
                    };
}

// Perform the dual connection login operation.
private void login() {
      String UserName=null;
      String Password=null;
      int port=0;
      UserName = user.getText();
      Password = password.getText();
      out.println(UserName);
      out.println(Password);
      try {
       // get the port, which the VIC listen from
       // the VIC Manager
       portFromServer = in.readLine();
       if (!portFromServer.equals("ValidLogin")) {
       this.logoff();
       };
       out.close();
                    in.close();
      } catch (IOException e) {
        System.err.println("Couldn't get I/O for the connection to Server.");
        System.exit(1);
      }
```

```
        // Waits for the required VIC to be activated.
        TakeTime(3);

        // Make connection with the VIC
        // which is located in 192.248.16.204
        try {
           wcSocket = new Socket("192.248.16.204", Integer.parseInt(portFromServer));
           out = new PrintWriter(wcSocket.getOutputStream(), true);
           in = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));
        } catch (UnknownHostException e) {
           System.err.println("Don't know about host.");
           System.exit(1);
        } catch (IOException e) {
           System.err.println("Couldn't get I/O for the connection.");
           System.exit(1);
        }

        out.println(UserName);
        out.println(Password);
        LoginPanel.disable();
        CommandField.enable();
        CommandField.setText("");
        CommandField.requestFocus();
    }

    private void logoff() {
        LoginPanel.enable();
        user.setText(null);
        password.setText(null);
        CommandField.setText("Not Connected.");
        CommandField.disable();
        this.cls();
        user.requestFocus();
    }

    // clear the User View Area
    private void cls() {
        TextMode.setText("");
    }

    // Construct the graphical components of the
    // applet.
    private void initComponents() {
        LoginPanel = new java.awt.Panel();
        User = new java.awt.Label();
        user = new java.awt.TextField();
        Password = new java.awt.Label();
        password = new java.awt.TextField();
        login = new java.awt.Button();
        UtilityPanel = new java.awt.Panel();
        status = new java.awt.Label();
        VICpanel = new java.awt.Panel();
        WorkArea = new java.awt.ScrollPane();
        TextMode = new java.awt.TextArea();
        CommandField = new java.awt.TextField();

        LoginPanel.setFont(new java.awt.Font ("Dialog", 0, 11));
        LoginPanel.setBackground(new java.awt.Color (204, 204, 204));
        LoginPanel.setForeground(java.awt.Color.black);

        User.setName("User");
         User.setText("User");
         LoginPanel.add(User);


        user.setColumns(20);
         user.setName("user");
         LoginPanel.add(user);
```

```
        Password.setName("Password");
         Password.setText("Password");
         LoginPanel.add(Password);


        password.setColumns(20);
         LoginPanel.add(password);


        login.setName("login");
        login.setLabel("Login");
        login.addActionListener(new java.awt.event.ActionListener() {
           public void actionPerformed(java.awt.event.ActionEvent evt) {
              loginActionPerformed(evt);
           }
        }
        );
        LoginPanel.add(login);


        this.add(LoginPanel, java.awt.BorderLayout.NORTH);


        VICpanel.setLayout(new java.awt.BorderLayout());
        VICpanel.setFont(new java.awt.Font ("Dialog", 0, 11));
        VICpanel.setBackground(new java.awt.Color (204, 204, 204));
        VICpanel.setForeground(java.awt.Color.black);


        TextMode.setEditable(false);
        TextMode.setForeground(java.awt.Color.white);
        TextMode.setFont(new java.awt.Font ("Monospaced", 0, 14));
        TextMode.setBackground(java.awt.Color.black);
        WorkArea.setSize(640, 380);
        WorkArea.add(TextMode);

        VICpanel.add(WorkArea, java.awt.BorderLayout.CENTER);


        CommandField.setFont(new java.awt.Font ("SansSerif", 1, 14));
        CommandField.setText("Not Connected.");
        CommandField.setBackground(java.awt.Color.yellow);
        CommandField.addActionListener(new java.awt.event.ActionListener() {
           public void actionPerformed(java.awt.event.ActionEvent evt) {
              CommandFieldActionPerformed(evt);
           }
        }
        );
        VICpanel.add(CommandField, java.awt.BorderLayout.SOUTH);


        this.add(VICpanel, java.awt.BorderLayout.CENTER);

        UtilityPanel.setFont(new java.awt.Font ("Dialog", 0, 11));
        UtilityPanel.setBackground(new java.awt.Color (204, 204, 204));
        UtilityPanel.setForeground(java.awt.Color.black);

        status.setName("status");
        status.setText("Virtual Internet Computer");
        UtilityPanel.add(status);

        this.add(UtilityPanel, java.awt.BorderLayout.SOUTH);

    }
```

```
// Executed when the user press the login button.
private void loginActionPerformed(java.awt.event.ActionEvent evt) {

    // Make connection with the VIC Manager application
    // which is located in 192.248.16.204
    try {
       wcSocket = new Socket("192.248.16.204", 1024);
       out = new PrintWriter(wcSocket.getOutputStream(), true);
       in = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));
    } catch (UnknownHostException e) {
       System.err.println("Don't know about host.");
       System.exit(1);
    } catch (IOException e) {
       System.err.println("Couldn't get I/O for the connection.");
       System.exit(1);
    }

    this.login();

    String fromServer;
    try {
     // Read out from the server and display it on the text area.
     fromServer = in.readLine();
     if (!fromServer.equals("ValidLogin")) {
     this.logoff();
     };
     while ((fromServer = in.readLine()) != null) {
       if (fromServer.equals("end"))
                   break;
       TextMode.append("\n" + fromServer);
     }

    } catch (IOException e) {
       System.err.println("Couldn't get I/O for the connection.");
       System.exit(1);
    }

  }

// Executed when the user enters a command in the command area.
private void CommandFieldActionPerformed(java.awt.event.ActionEvent evt) {

    String command;
    String fromServer;
    // get the command when press enter key
    command=CommandField.getText();

    // check whether length of the command is greater than zero.
    if (command.length()>0) {
    TextMode.append("\n" + "Command:\ \ " + command);
    if (command.equals("cls")){
    this.cls();
    } else if (command.length()!=0) {

    // Send the command to the VIC for processing
    out.println(command);

    if (command.equals("logoff")){
     // close the connection if the command is logoff
     try {
        this.logoff();
        out.close();
        in.close();
        wcSocket.close();
     } catch (IOException e) {
        System.err.println("Couldn't get I/O for the connection to Server.");
        System.exit(1);
     }
```

```
    } else {
      try {
        // Read out from the server and display it on the text area.
      while ((fromServer = in.readLine()) != null) {
              if (fromServer.equals("end"))
              break;
        TextMode.append("\n" + fromServer);
       }
      } catch (IOException e) {
         System.err.println("Couldn't get I/Ofor the connection to Server.");
         System.exit(1);
      }
      TextMode.append("\n");
     }
            }
            };
    // reset the command area
    CommandField.setText("");
}

//destroy the applet when the user exit from the web page
public void destroy() {
      try {
         wcSocket.close();
         out.close();
      } catch (IOException e) {
         System.err.println("Couldn't close I/O for the connection.");
         System.exit(1);
      }
}
   private java.awt.Panel LoginPanel;
   private java.awt.Label User;
   private java.awt.TextField user;
   private java.awt.Label Password;
   private java.awt.TextField password;
   private java.awt.Button login;
   private java.awt.Panel UtilityPanel;
   private java.awt.Label status;
   private java.awt.Panel VICpanel;
   private java.awt.ScrollPane WorkArea;
   private java.awt.TextArea TextMode;
   private java.awt.TextField CommandField;
}
```

VIC_HTML.java

```
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Title:       Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

// Client Java Servlet that Dynamically Generate HTML pages.

public class VIC_Html extends HttpServlet {

          Socket wcSocket=null;
          PrintWriter VICout=null;
          BufferedReader VICin=null;
          boolean LoggedIn = false;
        String portFromServer=null;

private void login(PrintWriter out) {

     // genarate HTML login page
     out.println("<html>");
     out.println("<head>");
                  String title = new String("Virtual Internet Computer");
     out.println("<title>" + title + "</title>");
     out.println("</head>");
     out.println("<body bgcolor=\"#000000\" text=\"#FFFF00\">");
     out.println("<p align= \"center\"><font size= \"6\">Virtual Internet
                                                          Computer</font></p>");
     out.println("<p align= \"center\"><font size= \"6\">Login</font></p>");
     out.println("<p align= \"center\">");
     out.println("<table border=\"2\" bordercolor=\"#FFFF00\"
                                                          bgcolor= \"#FF9933\">");
     out.println("<tr>");
     out.println("<form action= \"VIC_Html\" method=POST>");
     out.println("<td width=\"30%\">");
     out.println("    User ID :");
     out.println("</td>");
     out.println("<td width=\"70%\">");
     out.println("<input type=text size=20 name=name>");
     out.println("</td>");
     out.println("</tr>");
     out.println("<tr>");
     out.println("<td width=\"30%\">");
     out.println("    Password : ");
     out.println("</td>");
     out.println("<td width=\"70%\">");
     out.println("<input type=password size=20 name=password>");
     out.println("</td>");
     out.println("</tr>");
     out.println("<tr>");
     out.println("<p align= \"center\">");
     out.println("<td width=\"50%\">");
     out.println("<p align= \"center\">");
     out.println("<input type=\"submit\" value=\"    Login    \" name=\"login\">");
     out.println("</td>");
     out.println("</tr>");
     out.println("</table>");
     out.println("</form>");
     out.println("</html>");
     out.close();
}
```

```
private void logoff(PrintWriter out) {

    out.println("<html>");
    out.println("<head>");
    out.println("<title>Virtual Internet Computer</title>");
    out.println("</head>");
    out.println("<body bgcolor=\"#008000\" text=\"#FFFF00\">");
    out.println("<p align= \"center\"><font color=\"#FFFFFF\" size=\"6\">Virtual
                                                    Internet Computer</font></p>");
    out.println("<p align= \"center\"> </p>");
    out.println("<p align= \"center\"><font size= \"4\">You are logged out from your
                                                    VIC successfully.</font></p>");
    out.println("<p align= \"center\"> </p>");
    out.println("<p align= \"center\"><a href= \"http://localhost:8080/vic/\"><font
                        color= \"#FFFF00\" size= \"4\"><i>Go to the VIC Client
                                                    Page</i></font></a></p>");
    out.println("</html>");
    out.close();
}

private void invalidLogin(PrintWriter out) {
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Virtual Internet Computer</title>");
    out.println("</head>");
    out.println("<body bgcolor=\"#008000\" text=\"#FFFF00\">");
    out.println("<p align= \"center\"><font color=\"#FFFFFF\" size=\"6\">Virtual
                                                    Internet Computer</font></p>");
    out.println("<p align= \"center\"> </p>");
    out.println("<p align= \"center\"><font size= \"4\">User name and password is not
                                                        Correct!</font></p>");
    out.println("<p align= \"center\"> </p>");
    out.println("<p align= \"center\"><a href= \"http://localhost:8080/vic/\"><font
                        color= \"#FFFF00\" size= \"4\"><i>Go to the VIC Client Page
                                                        </i></font></a></p>");
    out.println("</html>");
    out.close();
}

private void outputScreen(PrintWriter out) {
    out.println("<html>");
    out.println("<head>");
    String title = new String("Virtual Internet Computer");
    out.println("<title>" + title + "</title>");
    out.println("</head>");
    out.println("<body bgcolor=\"#808080\" text=\"#FFFF00\">");
    out.println("<p align= \"center\"><font size= \"6\">Virtual Internet
                                                        Computer</font></p>");
    out.println("<p align= \"center\"><textarea rows= \"25\" name=\"Screen\"
                                                        cols=\"80\">");
    String fromServer;
    try {
     // Read results from the VIC
     while ((fromServer = VICin.readLine()) != null) {
       if (fromServer.equals("InvalidLogin")) {
       this.login(out);
       break;
       };
       if (fromServer.equals("end"))
                    break;

       out.println(fromServer);
     }
    } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to Server.");
      System.exit(1);
    }
```

```
        out.println("</textarea>");
        out.println("<p align= \"center\">");
        out.println("<form action= \"VIC_Html\" method=POST>");
        out.println("Command: ");
        out.println("<input type=text size=60 name=command>");
        out.println("</form>");
        out.println("</html>");
        out.close();
    }

    // This method waits for given number of seconds
    public static void TakeTime(int sec) {
            Date date1=null, date2=null;
            int x=0,y=0;
            date1 = new Date();
            x=date1.getSeconds();
            while ((y-x) < sec) {
            date2 = new Date();
            y=date2.getSeconds();
            };
    }

    // Invoked when the browser initialize the servlet
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        this.login(out);
    }

    // Invoked every time user enter a command
    public void doPost(HttpServletRequest request, HttpServletResponse res)
                                                throws IOException, ServletException {
        Enumeration e = request.getParameterNames();
        PrintWriter out = res.getWriter ();
        String UserId=null, Password=null;

                    while (e.hasMoreElements()) {
                        if (!LoggedIn) {
                        String name = (String)e.nextElement();
                        String value = request.getParameter(name);
                        if (name.equals("name")) {UserId=value;};
                        if (name.equals("password")) {Password=value;};
                        } else {
                        String name = (String)e.nextElement();
                        String value = request.getParameter(name);
                        VICout.println(value);
                            if (value.equals("logoff") || value.equals("shutdown")) {
                                    LoggedIn = false;
                                    this.logoff(out);
                            };
                        this.outputScreen(out);
                        }
                    };

        if (!LoggedIn) {

        // Make connection with the VIC Manager and
        // request for VIC
        wcSocket = new Socket("192.248.16.204", 1024);
        VICout = new PrintWriter(wcSocket.getOutputStream(), true);
        VICin = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));
```

```
    // Send user information to the VIC Manager
    VICout.println(UserId);
    VICout.println(Password);

    // get the port, which the VIC listening
    portFromServer = VICin.readLine();

    // Terminate the connection with the VIC Manager
    VICout.close();
    VICin.close();

    // wait until the VIC is activated.
    TakeTime(8);

    // Make a connection with the VIC
    wcSocket = new Socket("192.248.16.204", Integer.parseInt(portFromServer));
    VICout = new PrintWriter(wcSocket.getOutputStream(), true);
    VICin = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));

    // Send user information to the VIC
    VICout.println(UserId);
    VICout.println(Password);

    // Read the responce from the VIC.
    if (VICin.readLine().equals("ValidLogin")) {
                LoggedIn = true;
    this.outputScreen(out);
                } else {
                VICin.close();
                VICout.close();
                this.invalidLogin(out);
                };

    };
}
}
```

VIC_WAP.java

```
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Title:       Virtual Internet Computer Project
 * Author T. N. Ukwatta
 * University of Colombo
 */

// Client Java Servlet that Dynamically Generate WML pages.

public class VIC_Wap extends HttpServlet implements SingleThreadModel {

          Socket wcSocket=null;
          PrintWriter VICout=null;
          BufferedReader VICin=null;
          boolean LoggedIn = false;
         String portFromServer=null;

          String host = "http://localhost:8080/vic/servlet/VIC_Wap";

private void login(PrintWriter out) {

    // genarate WML login page
    out.println("<?xml version= \"1.0\"?>");
    out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
    out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
    out.println("<wml>");
    out.println("<card title=\"VIC Login\">");
    out.println(" <p align= \"center\">");

    out.println(" User: <input name= \"name\" size=\"15\"/>");
    out.println(" Password:  <input name= \"password\" size= \"15\" />");

    out.println("<anchor>");
    out.println("Login");
    out.println("<go method=\"post\" href= \"" + host + "\">");
    out.println("<postfield name= \"name\" value=\"$(name)\"/>");
    out.println("<postfield name= \"password\" value=\"$(password)\"/>");
    out.println("</go>");
    out.println("</anchor>");

    out.println("</p>");
    out.println("</card>");
    out.println("</wml>");
    out.close();
}

private void logoff(PrintWriter out) {
    out.println("<?xml version= \"1.0\"?>");
    out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
    out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
    out.println("<wml>");
    out.println("<card title=\"VIC\">");
    out.println(" <p align= \"center\">");
    out.println(" Logged Out");
    out.println("</p>");
    out.println("</card>");
    out.println("</wml>");
    out.close();
}
```

```
private void invalidLogin(PrintWriter out) {
    out.println("<?xml version= \"1.0\"?>");
    out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
    out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
    out.println("<wml>");
    out.println("<card title=\"VIC\">");
    out.println(" <p align= \"center\">");
    out.println(" Invalid Login");
    out.println("</p>");
    out.println("</card>");
    out.println("</wml>");
    out.close();
}

private void outputScreen(PrintWriter out) {
    out.println("<?xml version= \"1.0\"?>");
    out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
    out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
    out.println("<wml>");
    out.println("<card title=\"VIC\">");
    out.println("<p>");
    String fromServer;
    try {
      // Read results from the VIC
      while ((fromServer = VICin.readLine()) != null) {
        if (fromServer.equals("end"))
                    break;
       out.println(fromServer);
      }
    } catch (IOException e) {
       System.err.println("Couldn't get I/O for the connection to Server.");
       System.exit(1);
    }
    out.println("</p>");
    out.println(" <p align= \"center\">");
    out.println(" Command: <input name= \"command\" size=\"30\"/>");
    out.println("<anchor>");
    out.println("Send");
    out.println("<go method=\"post\" href= \"" + host + "\">");
    out.println("<postfield name= \"command\" value= \"$(command)\"/>");
    out.println("</go>");
    out.println("</anchor>");
    out.println("</p>");
    out.println("</card>");
    out.println("</wml>");
    out.close();
}

// This method waits for given number of seconds
public static void TakeTime(int sec) {
          Date date1=null, date2=null;
          int x=0,y=0;
          date1 = new Date();
          x=date1.getSeconds();
          while ((y - x) < sec) {
                    date2 = new Date();
                    y=date2.getSeconds();
          };
}

// Invoked when the browser initialize the servlet
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws IOException, ServletException {
     response.setContentType("text/vnd.wap.wml");
     PrintWriter out = response.getWriter();
     this.login(out);
}
```

```
// Invoked every time user enter a command
public void doPost(HttpServletRequest request, HttpServletResponse res)
                                        throws IOException, ServletException {
    Enumeration e = request.getParameterNames();
    res.setContentType("text/vnd.wap.wml");
    PrintWriter out = res.getWriter ();

    String UserId=null, Password=null;
          while (e.hasMoreElements()) {
              if (!LoggedIn) {
                    String name = (String)e.nextElement();
                    String value = request.getParameter(name);
                    if (name.equals("name")) {UserId=value;};
                    if (name.equals("password")) {Password=value;};
                    } else {
                      String name = (String)e.nextElement();
                      String value = request.getParameter(name);
                      VICout.println(value);
                      if (value.equals("shutdown") || value.equals("logoff")) {
                              LoggedIn = false;
                              this.logoff(out);
                              break;
                      } else {
                      this.outputScreen(out);
                      }
                    }
              }
    if (!LoggedIn) {

    // Make connection with the VIC Manager and
    // request for VIC
    wcSocket = new Socket("192.248.16.204", 1024);
    VICout = new PrintWriter(wcSocket.getOutputStream(), true);
    VICin = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));

    // Send user information to the VIC Manager
    VICout.println(UserId);
    VICout.println(Password);

    // get the port, which the VIC listening
    portFromServer = VICin.readLine();

    // Terminate the connection with the VIC Manager
    VICout.close();
    VICin.close();

    // wait until the VIC is activated.
    TakeTime(8);

    // Make a connection with the VIC
    wcSocket = new Socket("192.248.16.204", Integer.parseInt(portFromServer));
    VICout = new PrintWriter(wcSocket.getOutputStream(), true);
    VICin = new BufferedReader(new InputStreamReader(wcSocket.getInputStream()));

    // Send user information to the VIC
    VICout.println(UserId);
    VICout.println(Password);

    // Read the responce from the VIC.
    if (VICin.readLine().equals("ValidLogin")) {
        LoggedIn = true;
       this.outputScreen(out);
    } else {
        VICin.close();
        VICout.close();
        this.invalidLogin(out);
    };
   };
  }
}
```

Reference

[Citrix99a] Server Based Computing (CITRIX White Paper), 1999. http://www.citrix.com

[Citrix99b] Thin-Client Computing for the Government Sector (CITRIX White Paper), 1999. http://www.citrix.com

[Citrix2000a] The Thin Guide to Windows 2000 (CITRIX White Paper), 2000. http://www.citrix.com

[Citrix2000b] Wyse Winterm Thin-Client Terminals (CITRIX White Paper), 2000. http://www.citrix.com

[Vnc2001] AT&T Laboratories Cambridge, Virtual Network Computing, 2001 http://www.uk.research.att.com/vnc

[Tim96] Tim Love, Overview of X Windows, 1996. http://www-h.eng.cam.ac.uk/help/tpl/graphics/X/X11R5/node4.html

[Tanen2000a] Andrew S. Tanenbaum, "Computer Networks", pp-87-101, Third edition 2000.

[Tanen2000b] Andrew S. Tanenbaum, "Modern Operating Systems" pp-362-373, 2000.

[Rodger94] R.B.Rodger, Online X Windows course, 1994. http://www.strath.ac.uk/CC/Courses/oldXC

[Venners2001] Bill Venners, "Inside the Java 2 Virtual Machine" Chapter 01 2001

[Mahmoud2001] Qusay H. Mahmoud, "WAP for Java developers" 2000 http://www.javaworld.com

[Chris2000] Chris Pedley, "Professional WAP" , pp-109-163, pp-310-310, 2000

[TimLin99] Tim Lindholm, Frank Yellin, "The Java Virtual Machine Specification", Second Edition, HTML Version. 1999 http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html