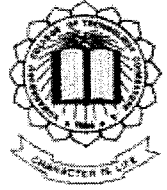




P-1605



COLLABORATED SERVER ARCHITECTURE IMPLEMENTATION USING RFB PROTOCOL

A PROJECT REPORT

Submitted by

LAVANYA MANICKAN (71202104019)

NISHA VENUGOPAL MENON (71202104024)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY:: CHENNAI 600 025

APRIL 2006

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **“COLLABORATED SERVER ARCHITECTURE IMPLEMENTATION USING RFB PROTOCOL”** is the bonafide work of **“Lavanya Manickan (71202104019)** and **Nisha Venugopal Menon (71202104024)”** who carried out the project work under my supervision.


SIGNATURE

Dr. S. Thangasamy

HEAD OF THE DEPARTMENT

Department of Computer Sci & Engg.

Kumaraguru College of Technology

Chinnavedampatti Post

Coimbatore-641 006


SIGNATURE

Mrs. V. Vanitha

SUPERVISOR

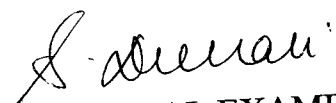
Department of Computer Sci & Engg

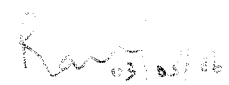
Kumaraguru College of Technology

Chinnavedampatti Post

Coimbatore-641 006

Submitted for Viva-Voce examination held on 03-05-2006


INTERNAL EXAMINER


EXTERNAL EXAMINER

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

It is our pleasure to express our thanks to Prof. K. Arumugam, B.E., (Hons), M.S. (U.S.A), M.I.E., Correspondent, Kumaraguru College of Technology for his kind contact, inspiration and constant encouragement.

We sincerely thank our beloved Principle **Dr. K.K.Padmanabhan**, B.Sc. (Engg), M.Tech., for his Invaluable support and appreciation he had shown towards our project.

We express our heartfelt gratitude to our Head of Department, Computer Science and Engineering Branch, **Dr. S. Thangasamy**, B.E.(Hons), Ph.D., for the encouragement and the technical knowledge he has bestowed towards the completion of the project successfully.

We are thankful to our Internal Guide, **Mrs. V. Vanitha**, M.E., Senior Lecturer, Department of Computer Science and Engineering, Kumaraguru College of Technology, who inspired us to do the project. Her knowledge in the subject helped us to circumvent the obstacles and helped us to complete the project successfully in a short period of time.

We would also like to thank our project co-ordinator **Mrs. S. Devaki** M.S., Assistant Professor, Kumaraguru College of Technology, who has been very helpful and has guided us throughout the project.

We express our sincere thanks to **Mr. C.Jeyaram**, System Administrator for his valuable suggestions and doing the needful for the completion of the project.

We express our debt of gratitude to our Almighty God, family and friends who have been the driving force for our accomplishments.

ABSTRACT

ABSTRACT

This project is in essence a collaborated server architecture that is used to enhance the operability of the existing Virtual Network Computing environment.

Virtual Network Computing is basically a remote display system which allows one to view a computing 'desktop' environment not only on the machine where it is stored, but anywhere from a wide variety of machine architectures. The VNC architecture has two main components:

Remote System, which generates a display and a

Viewer, which actually draws the display on its screen.

The existing system has the following disadvantages:

- 1) The remote system has to keep track of and update multiple viewer machines when it is simultaneously accessed.
- 2) Since graphical information needs to be transferred, the time required to do the same is increased.
- 3) Due to the above reasons, the speed of the remote system reduces.

In order to boost up the remote system's speed, in case of 1: N remote system-viewer scenario, we go for **collaborated server**. Viewers share remote system's desktop in read-only mode through collaborated server. Many remote systems can register themselves to the collaborated server thus providing the viewer an option to choose one amongst them. The collaborated server takes care of updation in all viewers thereby reducing the workload of the remote system and hence increasing its speed.

Thus the project allows multiple viewers to capture the desktop of the remote system from several places simultaneously.

***TABLE OF
CONTENTS***

CHAPTER NO	CONTENTS	PAGE NO
	ABSTRACT	iii
	LIST OF TABLES	iv
	LIST OF FIGURES	v
	LIST OF SYMBOLS	vi
	<u>CHAPTER OF REPORT</u>	
1.	INTRODUCTION	
	1.1 OVERVIEW OF THE PROJECT	1
	1.2 AIM OF THE PROJECT	3
	1.3 PURPOSE	4
2.	SYSTEM ANALYSIS	
	2.1 EXISTING SYSTEM	5
	2.2 PROPOSED SYSTEM	7
3.	SYSTEM SPECIFICATION	
	3.1 HARDWARE CONFIGURATION	11
	3.2 SOFTWARE CONFIGURATION	12
	3.3 DOMAIN REQUIREMENTS FOR THE PROJECT	13
	3.4 FLOWCHART	14
	3.5 DATA FLOW DIAGRAMS	17

4.	PROJECT IMPLEMENTATION	
	4.1 PROCESS MODULE	19
	4.2 MODULE DESCRIPTION	20
	4.3 MODULE IMPLEMENTATION	23
	4.4 MODULE FEATURES	36
	4.5 TESTING	42
	4.6 OUTPUT	44
5.	PROJECT FEATURES	
	5.2 ADVANTAGES	52
	5.3 LIMITATIONS	53
	5.4 APPLICATIONS	54
6.	CONCLUSION	55
7.	FUTURE ENHANCEMENTS	56
8.	APPENDICES	
	APPENDIX 1: SAMPLE CODE	57
	APPENDIX 2: USER MANUAL	72
9.	REFERENCES	76

LIST OF TABLES

(TABLE 1)

OPERATING CONDITION DURING TEST ANALYSIS

LIST OF FIGURES

(FIG 1)	EXCLUSIVE CONNECTION
(FIG 2)	EXCLUSIVE MODE DISCONNECTION
(FIG 3)	SHARED MODE CONNECTION
(FIG 4)	SHARED MODE CONNECTION IN COLLABORATED SERVER VIEWER
(FIG 5)	VNC VIEWER-REMOTE SYSTEM
(FIG 6)	COLLABORATED SERVER VIEWER CONNECTION ESTABLISHMENT AND UPDATION
(FIG 7)	CONNECTION ESTABLISHMENT
(FIG 8)	COLLABORATED VIEWER COLLABORATED SERVER
(FIG 9)	RFB PROTOCOL
(FIG 10)	RUNNING VNC REMOTE SYSTEM
(FIG 11)	SETTING VNC REMOTE SYSTEM PROPERTIES
(FIG 12)	RUNNING COLLABORATED SERVER
(FIG 13)	RUNNING COLLABORATED VIEWER
(FIG 14)	CONNECTING TO THE COLLABORATED SERVER
(FIG 15)	CHOOSING A VNC REMOTE SYSTEM
(FIG 16)	COLLABORATED VIEWER VIEWING VNC REMOTE SYSTEM DESKTOP:
(FIG 17)	UPDATION INFORMATION OF VNC REMOTE SYSTEM IN COLLABORATED SERVER

LIST OF SYMBOLS

iv) LIST OF SYMBOLS

CSA	COLLABORATED SERVER ARCHITECTURE
DES	DATA ENCRYPTION STANDARD
JDK	JAVA DEVELOPMENT KIT
RGB	RED, GREEN, BLUE
RFB	REMOTE FRAME BUFFER
SOCK	SOCKET
TCP	TRANSMISSION CONTROL PROTOCOL
VNC	VIRTUAL NETWORK COMPUTING

INTRODUCTION

INTRODUCTION

1.1 OVERVIEW OF PROJECT

In recent advancements of Information Technology areas, a new era called Virtual Network Computing has evolved. Virtual Network Computing enables PC-PC communication using a protocol called RFB (Remote Frame Buffer). From the olden days there are solutions used, which enables one to work with his/her PC from a remote place (ex. Telnet, Rlogin, MainFrame nodes, Unix/Linux Terminals). As PC cost was drastically reduced, there was less need for remote controlling solutions. But still these solutions are not out of the market.

Now days the Industry expects a better quality service/maintenance for any software solution, which a company would have deployed in the past. Software Installation is one of the major tasks for the system administrators in a Software Production department of an organization. Product Demonstration/ Broadcasting a video over network is a tedious task for the Tech-Executives. The software licensing is enabling people not to adapt to latest technologies unless otherwise they bought the software. Application Maintenance or Software Debugging is also a Major task in a Software Development/Training Industry for Coordinators. Accessing/controlling Hardware Resources, which are associated with a remote machine, is also becoming a routine job for the network users/administrators. To provide a cost effective solution with respect to above described problems, we are proposing this solution.

Thus Remote controlling solutions are widely picked up in the market from the last decade. There are various solutions provided for the same need but which are not effective (cost wise, flexibility wise, performance wise) as the proposed solution.

To name a few,

Telnet – Here from a dumb terminal a command can be executed remotely (emulation). But, when pc cost is reduced there is no need of dumb terminal. Thus telnet is not ultimate and no graphical pictures are possible to be viewed. Further the work environment is entirely different and not further customizable in Telnet.

Windows 98 - Netmeeting 3.0 – This will enable certain features but not to a full extent and further this solution is not customizable and maintainable by corporate companies unless otherwise Microsoft comes forward to do the same. Further working with Netmeeting & Remote Desktop Sharing both is not possible at the same time.

New Microsoft Windows- XP has come out with a better solution but which may not be feasible, for every consumer to upgrade from Windows – 98 to Windows XP. Considering all these factors and need of the system, the proposed system is an Intranet solution, which can be further customizable to corporate needs if required.

Thus, the proposed solution will enable to view a remote machine desktop over Intranet provided the access rights are enabled. This remote control solution will enable at the same time to serve many viewers who would like to view the desktop of a remote PC. Along with this feature it serves all the needs described above.

1.2 AIM OF THE PROJECT

The project is used to enhance the operability of the existing Virtual Network Computing architecture. The proposed collaborated architecture solution identifies the need for increased speed in transferring graphical information from one remote desktop to all viewers.

It thus enables to view a remote machine desktop over Internet/Intranet (depends on bandwidth) provided the access rights are enabled. The remote control solution will enable at the same time to serve many viewers who would like to view the desktop of a remote desktop.

1.3 PURPOSE

The project solves the speed lag, which is present in the existing VNC architecture and can be extended to support video broadcasting applications.

The enhanced operability provides easy mobile computing without requiring the user to carry any device whatsoever.

In addition, it allows a desktop to be accessed from several places simultaneously thus supporting application sharing in the style of Computer supported Co-operative work.

SYSTEM ANALYSIS

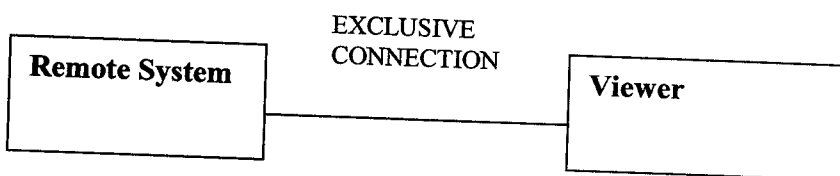
SYSTEM ANALYSIS:

2.1 EXISTING SYSTEM

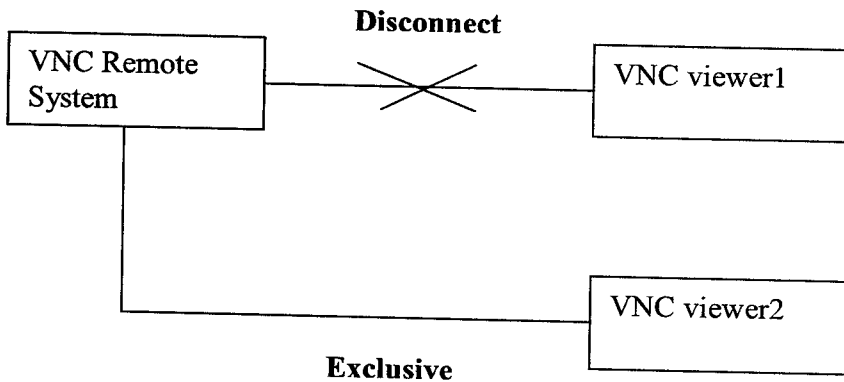
The existing system use VIRTUAL NETWORK COMPUTING which will enable PC-PC communication using a protocol called RFB(remote frame buffer).Any event triggered on the viewer side is captured and transmitted to the remote system along with the portion (x, y coordinates). Remote system updates its desktop according to the event received and reflects the same changes in the viewer. In this way, viewer and remote system shares the desktop. Two types of connection are possible,

- Exclusive connection
- Sharable connection

In the exclusive connection, at a time only one viewer and a remote system can be connected. If another viewer establishes a connection with the remote system in exclusive mode, the previous viewer will be disconnected from the remote system irrespective of its mode of connection.

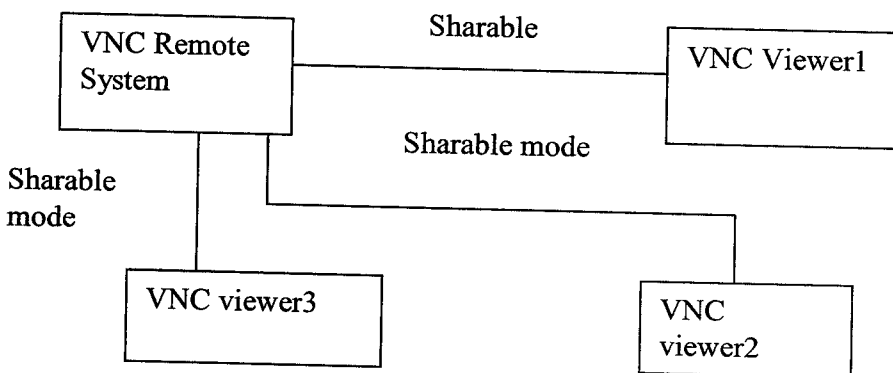


(FIG 1)- EXCLUSIVE CONNECTION



(FIG 2)-EXCLUSIVE MODE DISCONNECTION

In the sharable mode, more than one viewer can be connected to a RFB remote system so that many viewers can access and update a single RFB remote system. All the viewers which get connected to the remote system in this mode can initiate updation which is processed by the remote system and reflected to all the viewers. Thus, any viewer can update the remote system desktop with equal priority.



(FIG 3) SHARED MODE CONNECTION

2.2 PROPOSED SYSTEM

The proposed system CSA (Collaborative Server Architecture), It is in essence a remote display system, which allow you to view a computing desktop environment not only on the machine where it is running but from a wide variety of machine architectures. It is a networking project using JAVA as platform and the TCP/IP protocol.

The protocol used is a simple protocol for remote access to graphical user interface. It is based on the concept of remote frame buffer. The protocol simply allows the remote system to update the frame buffer displayed on a viewer. Because it works on the frame buffer level it is applicable to all operating systems.

Writing a viewer is a simple task. It requires only a reliable transport (usually TCP/IP) and a way of displaying pixels. Writing a remote system is slightly harder. The protocol is designed to make the viewer as simple as possible, so it is usually up to the remote system to perform any translation. For example, the remote system must provide the pixel data in the format the viewer wants.

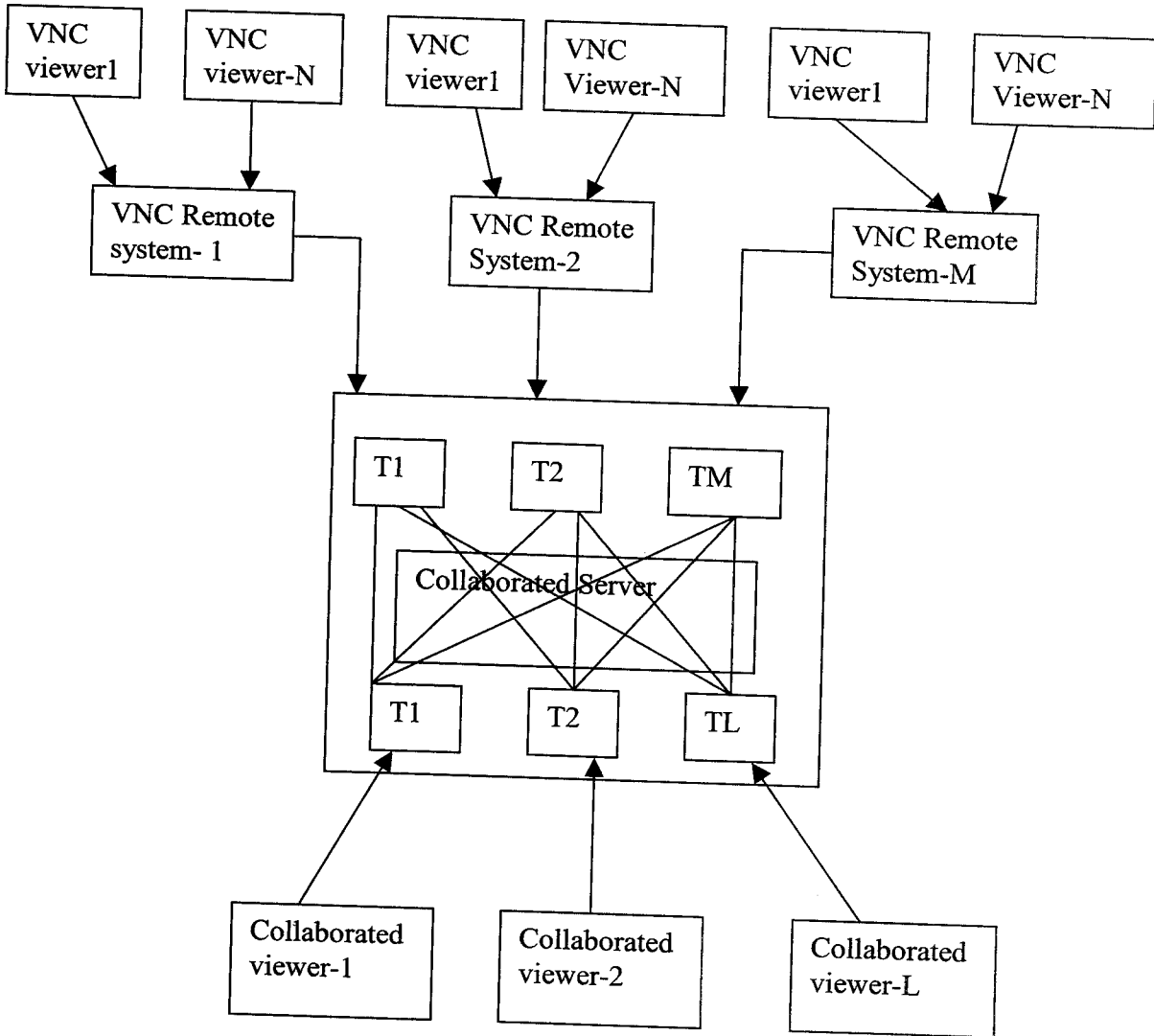
The VNC remote system makes the existing desktop on the PC available remotely rather than creating a separate desktop as it happened with the UNIX server. This will run on windows NT/95 and on any future Win32 based system without the need to replace any system files or run any OS specific versions of the program. VNC Remote System can also be run as a service which means that you can log in remotely and do some work and log

out again. Either the viewer or the remote system can initiate updation on remote system desktop and the same is reflected on both. When the number of viewers connected to the remote system increases, the speed of the remote system faces a setback. In order to boost up the remote system's speed, in case of 1: N scenario, we go for collaborated server. The collaborated server functions as an intermediate between the remote system and the viewer. It maintains the connection with each viewer and also with each remote system. This is taken care of by the collaborated server. The viewer is allowed to choose from different remote systems who are connected to collaborated server. Thus it is possible to have multiple user and multiple remote systems. The speed of the remote system here is not affected by number of viewers. Whenever the viewer wants to communicate with the remote system a connection has to be established. An important factor to be considered here is that there can be more than one viewer that can be connected to the remote system at a time.

There are two basic scenarios as far as the operation of this is concerned. On one side, the viewers will reflect any changes made in the remote system's side. In other words, the viewer is able to view the changes made by the remote system. Transferring the update rectangles to the viewers does this. Transfer of the entire frame buffer takes place pixel by pixel. In cases where there is only a small change in the frame buffer then a rectangle can be constructed around this updated area and only this portion can be broadcasted. A sequence of these rectangles makes a frame buffer update. An update represents a change from one valid frame buffer state to another. So in many ways it is similar to a frame of a video. But usually it is a small area of the frame buffer that will be affected by a given update.

The data from the frame buffer has to be compressed and transmitted. This is because transmission of bulk volume of data may lead to error and increase the cost of the cable. Since it is a networking environment, the data has to be encrypted or encoded for safety purpose to prevent eavesdropper from taping the information. Similarly decompression and decryption should be done on the receiver side.

(FIG 4) SHARED MODE CONNECTION IN COLLAB SERVER-CLIENT



***SYSTEM
SPECIFICATION***

SYSTEM SPECIFICATIONS:

3.1 HARDWARE CONFIGURATION:

- Monitors : 800 x 600 minimum resolution at 256 colors minimum
- Memory : Approximately 256MB of on board memory
- I/O : Mouse and a Standard 101- key board
- Processor Speed : Atleast 166 MHZ processor
- Processor Used : Pentium-4

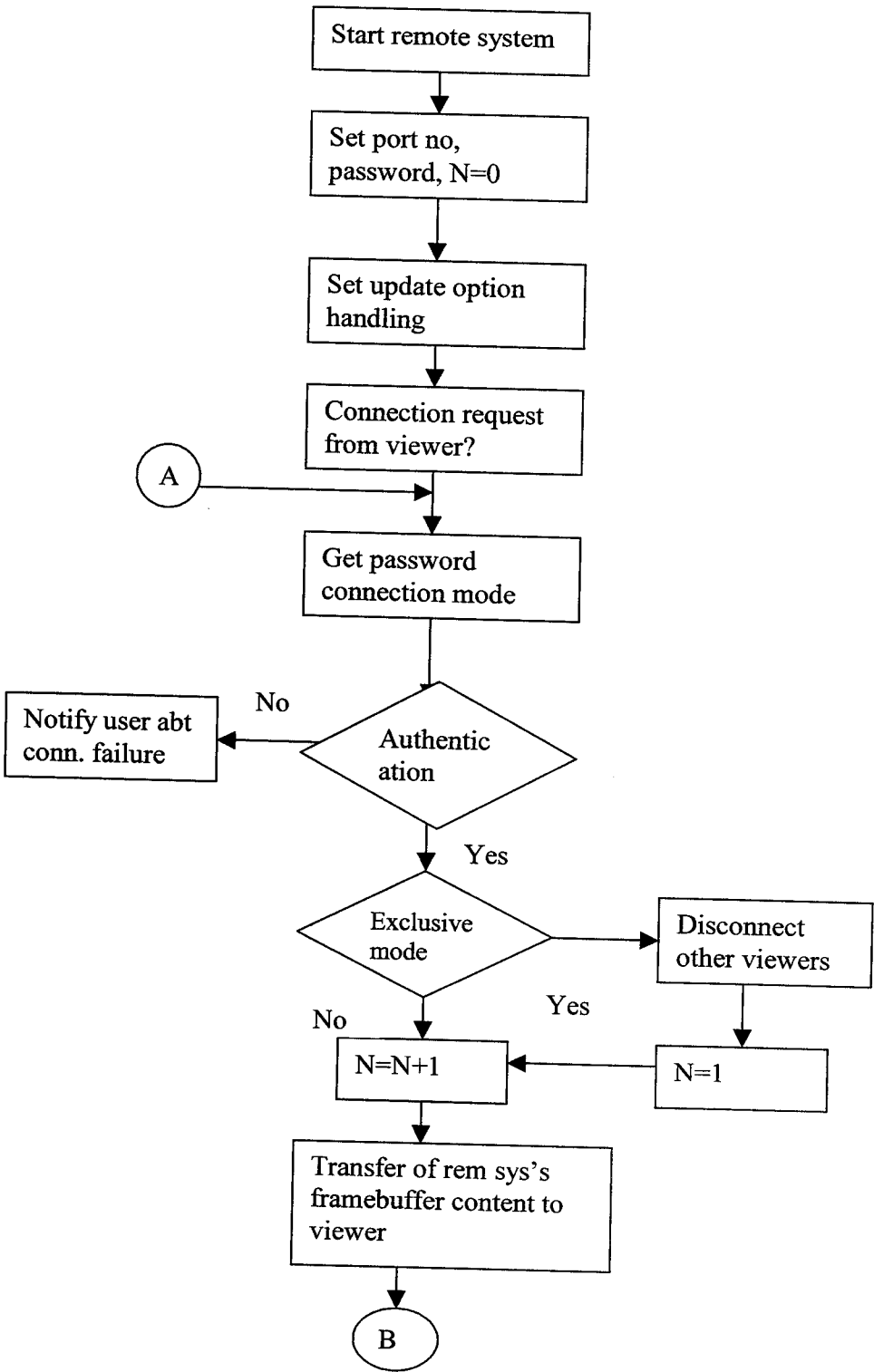
3.2 SOFTWARE CONFIGURATION:

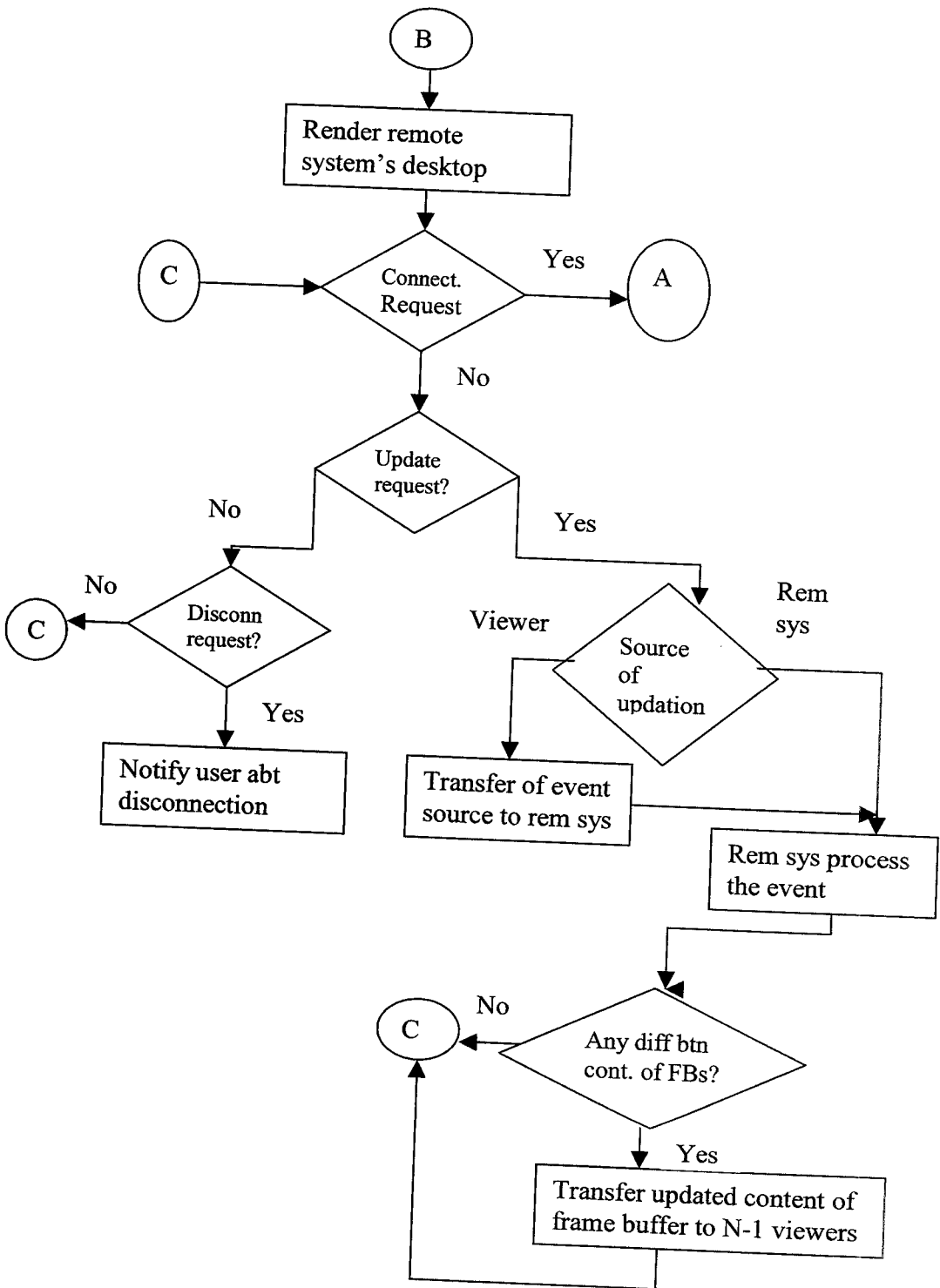
- Language : JDK 1.4 or Above
- Operating System : Windows 98, NT, XP, 2000
- Tools: NetBeans 3.6 or above
- TCP/IP architecture
- Protocol : Remote Frame Buffer Protocol
- Algorithm implemented by the protocol: DES
- Encoding technique: Raw encoding

3.3 DOMAIN REQUIREMENTS FOR THE PROJECT:

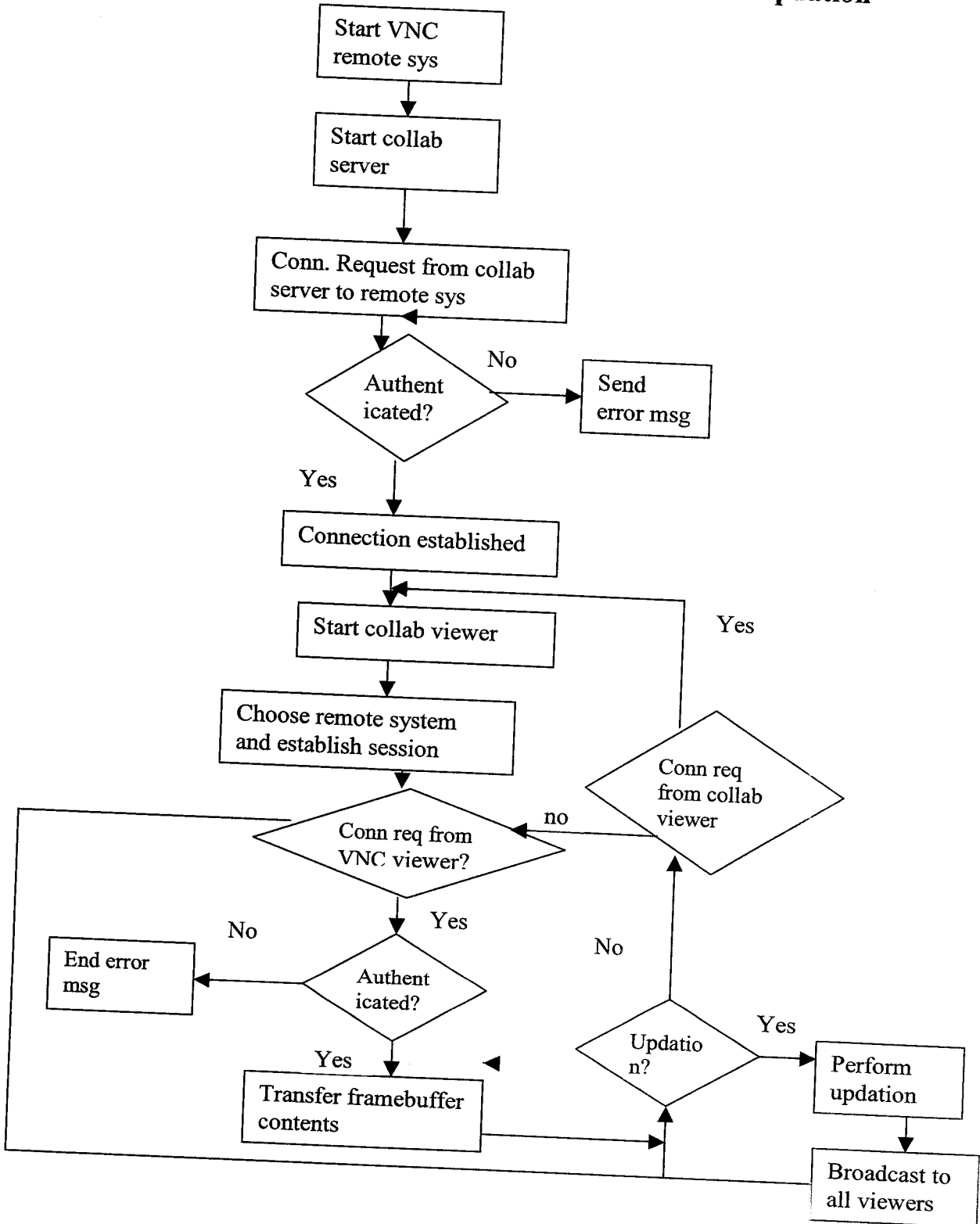
- Windows Operating System Fundamentals
- TCP/IP Architecture
- Telnet, Rlogin Protocol Services for comparative study
- Remote Frame Buffer Protocol
- Data Encryption Standard Algorithm for Cryptography
- Multi-Threading Concepts

**3.4 FLOW CHART
(FIG 5)VNC VIEWER-REMOTE SYSTEM**



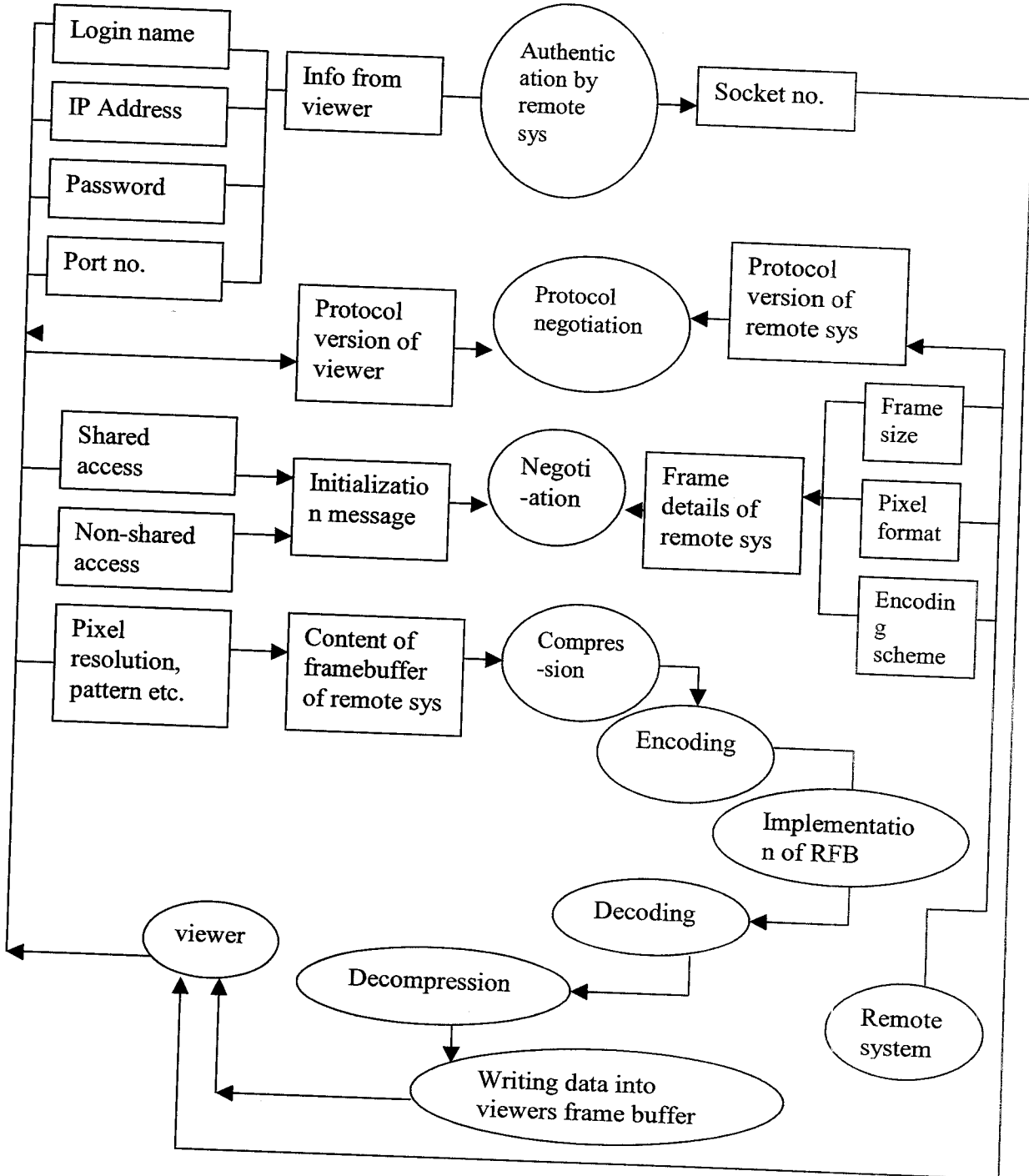


(FIG 6) Collab server-viewer connection establishment and update

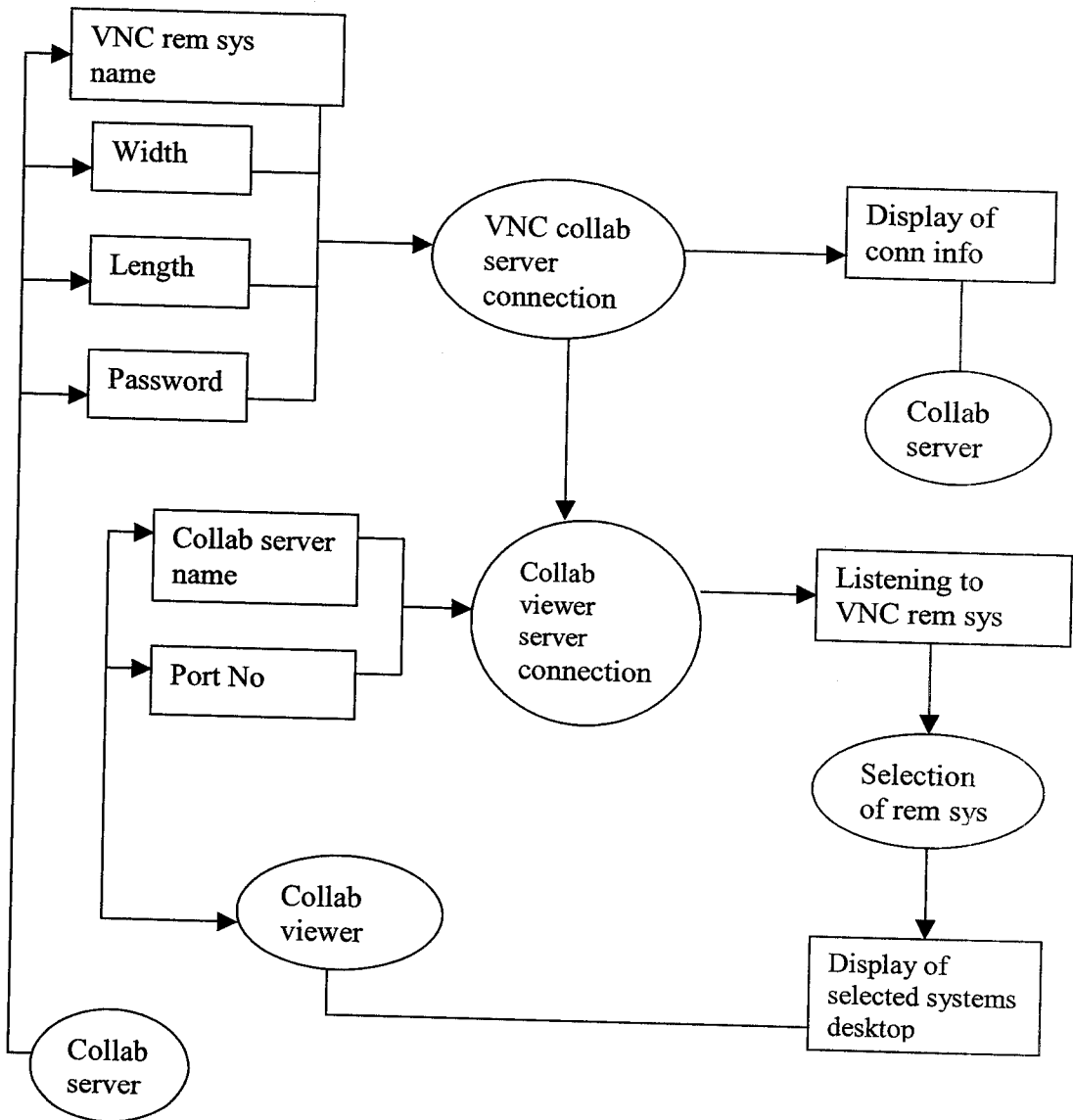


3.5 DATA FLOW DIAGRAMS:

(FIG 7) CONNECTION ESTABLISHMENT:



(FIG 8) COLLABORATED VIEWER AND COLLABORATED SERVER



***PROJECT
IMPLEMENTATION***

PROJECT IMPLEMENTATION:

4.1 PROCESS MODULES

Modularity is the key feature of any system for provision of effective testing and debugging. This helps in acquiring control over system development and modification. Our project comprises of totally 4 modules. They are:

- RFB Remote System
- Connection establishment
- Collaborated-server
- Collaborated-viewer

4.2 MODULE DESCRIPTION:

RFB Remote System:

RFB Remote system (remote frame buffer) uses a simple protocol for remote access to graphical user interface. Because it works at the frame buffer level it is applicable to all windowing systems and applications, including x11, windows 3.1/95/NT and Macintosh.

The remote end point where the user sits (that is the display plus the keyboard and/or pointer is called the RFB viewer. The end point where changes to the frame buffer originate (that is windowing system and the application) is known as RFB remote system. The remote system notifies the protocol version message to the user. Later the authentication scheme used is also notified. The remote system and the viewer initialization messages are written out using RFB protocol. The pixel format, encoding scheme, color map entries are also set. The requests for frame buffer updates and the remote system updates the frame buffer.

Connection Establishment:

This module is where the viewer connects with the VNC- remote system. More than one user can connect to the VNC- remote system. It takes care of the connection between VNC remote system and viewer. Here DES is applied for secure data transmission. This module takes care of writing various viewer side options handling, receiving frame buffer updates,

update of viewer side frame buffer with respect to new frame buffer updates and event handling. It also maintains a clipboard text.

Collaborated- Server:

The collaborated server reduces the overhead of the VNC- remote system by maintaining session for each viewer. The process of connecting to collaborated-server and the process of authentication are carried out in this module. This module is one where the connection with VNC- remote system is established. After the connection is established the remaining protocol initialization is performed, the encoding scheme is send from options frame to VNC- remote system. The details about the remote system such as identification, session password, mcport no, port no; width and height of the screen are stored. The connection to required remote system is established after reading the user name.

This module comprises of registering various collaboratedviewersessions with Collaborated-Server, writing all VNC-Server Desktop Names to the Collaborated-Viewers, handling CollaboratedViewerRequest for a particular VNC- remote system, attaching a ServiceHandle between a VNC remote system ServiceThread and CollaboratedViewerServiceThread (many-many (or)1-many (or) many-1 (or) 1-1), sending the VNC remote system FrameBufferUpdates to the respective CollaboratedViewerSessions and Handling ConnectionCloseRequests initiated from CollaboratedViewerSessions.

Collaborated- Viewer:

The collaborated viewer reads the information about the remote system. The process of initializing the components takes place in this module and the validity of remote system details are tested. After giving the update requests the viewer receives the update information from the remote system. Then the updation information is made known to user by printing the status of updation on the screen in this module. Any error in receiving the update information is indicated to the user. This module takes care of CollaboratedViewerRegistrationRequest to CollaboratedServer, initiating CollaboratedSessionRequest for a particular VNC- remote system which is handled indirectly through CollaboratedServer, adding an another CollaboratedSessionRequest for another VNC- remote system which will be again handled indirectly through CollaboratedServer while other initiated session is still in ServiceMode (1 CollaboratedViewer – many VNC remote system (only One CollabServer)), Handling VNC remote system FrameBufferUpdates sent by VNC remote system ServiceThread through CollaboratedServer, Updating InternalFrames for every CollaboratedServiceThread which has been served by CollaboratedServer, Handling ConnectionCloseRequest and terminating respective Internal Frames.

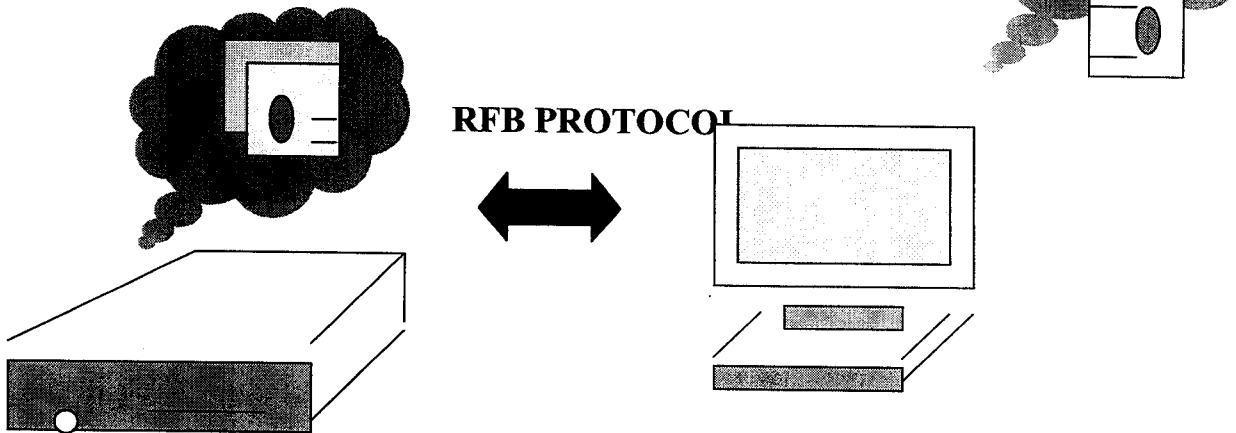
4.3 MODULE IMPLEMENTATION:

➤ RFB Protocol

RFB (“remote framebuffer”) is a simple protocol for remote access to graphical user interfaces. Because it works at the framebuffer level it is applicable to all windowing systems and applications, including X11, Windows 3.1/95/NT and Macintosh. The remote endpoint where the user sits (i.e. the display plus keyboard and/or pointer) is called the RFB viewer. The endpoint where changes to the framebuffer originate (i.e. the windowing system and applications) is known as the RFB remote system.

RFB Remote System

Viewer



(FIG 9) RFB PROTOCOL

The emphasis in the design of the RFB protocol is to make very few requirements of the viewer. In this way, viewers can run on the widest range of hardware, and the task of implementing a viewer is made as simple as possible. The protocol also makes the viewer stateless. If a viewer disconnects from a given remote system and subsequently reconnects to that same, the state of the user interface is preserved. Furthermore, a different viewer endpoint can be used to connect to the same RFB remote system. At the new endpoint, the user will see exactly the same graphical user interface as at the original endpoint. In effect, the interface to the user's applications becomes completely mobile. Wherever suitable network connectivity exists, the user can access their own personal applications, and the state of these applications is preserved between accesses from different locations. This provides the user with a familiar, uniform view of the computing infrastructure wherever they go.

Display Protocol

The display side of the protocol is based around a single graphics primitive: "put a rectangle of pixel data at a given x,y position". At first glance this might seem an inefficient way of drawing many user interface components. However, allowing various different encodings for the pixel data gives us a large degree of flexibility in how to trade off various parameters such as network bandwidth, viewer drawing speed and server processing speed. A sequence of these rectangles makes a framebuffer update (or simply update). An update represents a change from one valid framebuffer state to another, so in some ways is similar to a frame of video.

The rectangles in an update are usually disjoint but this is not necessarily the case.

The update protocol is demand-driven by the viewer. That is, an update is only sent from the remote system to the viewer in response to an explicit request from the later. This gives the protocol an adaptive quality. The slower the viewer and the network are, the lower the rate of updates becomes. With typical applications, changes to the same area of the framebuffer tend to happen soon after one another. With a slow viewer and/or network, transient states of the framebuffer can be ignored, resulting in less network traffic and less drawing for the viewer.

➤ **COLLABORATED SERVER:**

This module illustrates the actual function of the Collaborated Server. The reason behind implementing this Collaborated Server is to decrease the workload put on the remote system. Since all the viewers are connected only to this Collaborated server the actual RFB Remote System need not keep track of the status of the n number of viewers. As a result of this the speed of remote system is increased.

The CollaboratedProperties plays an important role in the connection process. Resource bundles contain locale-specific objects. When your program needs a locale-specific resource, a String for example, your program can load it from the resource bundle that is appropriate for the current user's locale. In this way, you can write program code that is largely independent of the user's locale isolating most, if not all, of the locale-

specific information in resource bundles. This allows you to write programs that can:

- Be easily localized, or translated, into different languages
- Handle multiple locales at once
- Be easily modified later to support even more locales

One resource bundle is, conceptually, a set of related classes that inherit from `ResourceBundle`. Each related subclass of `ResourceBundle` has the same base name plus an additional component that identifies its locale. Resource bundles contain key/value pairs. The keys uniquely identify a locale-specific object in the bundle. The Java 2 platform provides two subclasses of `ResourceBundle`, `ListResourceBundle` and `PropertyResourceBundle`, that provide a fairly simple way to create resources. `ListResourceBundle` manages its resource as a `List` of key/value pairs. `PropertyResourceBundle` uses a properties file to manage its resources. All the information about the various VNC remote systems such as name, password, framebuffer width and height, port number etc are entered into this file in a particular format. By viewing this file we can see the number of VNC remote systems that are actively running. The viewer is allowed to connect to any one of these remote systems. It does so by using the `CollaboratedProperties` file.

This module creates a `propertyresourcebundle` and reads the property file. It prints the remote system name and creates an instance of `collabconnector` for each serverinfo. A remote system socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to them. It creates remote system socket at port

number 2345. The VNC remote system details from the collaboratedproperties are stored in the vector.

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created. Each vector tries to optimize storage management by maintaining a capacity and a capacity Increment. The capacity is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of capacity Increment. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

The function of the Collaborated server is to connect the viewers to the VNC remote system without decreasing its performance and speed. On establishing connection, a thread for collaboratedviewer is started. The other modules are Remote system Info, CollaboratedConnector, CollaboratedViewerServiceThread .

Remote System Info:

This is a very simple module and its function is to supply the Collaborated server with the actual values from the CollaboratedProperties. RemoteSystemInfo is a small class that consists of three types of constructors. In the first one the framebuffer width and height, port and mcport are set as constants. In the second one only the port and mcport are set whereas in the third one all the values can be altered by the users. Hence

the function of this module is to return the current values to the Collaborated server from the CollaboratedProperties. The password length is restricted to have their maximum number of characters as 8.

Collabconnector:

The function of this module is to connect to the VNC Remote system and authenticate the user and do the protocol initialization. This module is called by the Collaborated Server for each of the existing VNC Remote system. The connection and authentication is done the way described below.

Firstly the protocol version of the VNC remote system is read and printed.

Then the version message of the client is written.

The authentication scheme is then found out and depending on that various actions are performed.

1. If there is no authentication scheme used then the appropriate message is printed.

2. If authentication is used then the encryption process takes place by using the password given by the user as a key. The length of this password is restricted to 8 characters.

The result of authentication is then read and one of the following takes place.

- a. If VncAuthOK is returned then the Authentication was successful.

- b. If VncAuthFailed is returned then the Authentication failed.

- c. If VncAuthTooMany is returned then it indicates that there were too many tries during the process of authentication.

Hashtable class implements a hashtable, which maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

The protocol initialization is basically done in the RFB protocol. The rest of it such as printing corresponding messages, setting of encoding schemes is done here. The Desktop name and size are printed and the width and height of the frame buffer are set appropriately. The raw encoding is implemented here because it is the most effective encoding technique. Any error that takes place in the protocol initialization or connection and Authentication are displayed so that the user knows what exactly had caused the error.

Collabviewerservicethread:

This module is called for every viewer that has a valid connection with the remote system. This initializes the format string that contains information about the host frame buffer width & height etc. once the initialization is done a procedure called checkViewerList is called. Here it is checked that the remote system required by the viewer is a valid one and the viewer is not already connected to that same remote system. If this condition is satisfied then a message specifying that the viewer is added to the required remote system is displayed. This function is done in a procedure called addViewerToConnector. The concept of multithreading is implemented here to satisfy all the viewers simultaneously.

Information as to which viewer the processing is done is specified so as to avoid confusion in a multiuser environment. This module basically makes use of Input/Output streams to get the information regarding the user name

and the server name. After fetching then it checks if the user is requested for the right remote system and then services the viewer if it chooses the right remote system. This module makes use of the CollaboratedProperties to check if the viewer is connected to a valid user.

A sample of information contained in CollaboratedProperties is shown below:

1.remotesystem=tcms

1.port=5900

1.width=800

1.height=600

1.password=at

1.mcport=9999

The CollabServer plays an important role in this module and the program flow is appropriately transferred to the other sub-modules. The CollaboratedProperties serves as a source of information about the remote systems, if any remote systems have to be added, entries for that system are made in this CollaboratedProperties before executing.

➤ **COLLABORATED VIEWER:**

A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal

to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named `main` of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:

- The `exit` method of class `Runtime` has been called and the security manager has permitted the exit operation to take place.
- All threads that are not daemon threads have died, either by returning from the call to the `run` method or by throwing an exception that propagate beyond the `run` method.

The user name and remote system name is obtained first. A new socket connection is formed and the thread is started, thus multiple viewers are possible. A socket is an endpoint for communication between two machines. The VNC remote systems which are registered with `CollaboratedServer` through `CollaboratedProperties` are obtained in an array list. The user is provided with an option to choose the RFB remote system from this list and is allowed to establish a connection with the same.

If the remote system is not found then an error message is printed in the console. When the internal frame closing operation is performed the connection is disconnected and disposed. The viewer checks if current host is there in the remote system list. All the properties of the remote systems such as the host name, display, port number, `mcport`, width and height are read. The details are stored in the array. An option pane appears once the viewer is connected to collaborated server. This asks the user to choose the

remote system he requires. If the remote system is not present in the list then error message is given. Once all the required information about the remote system is obtained as stream of data, the X, Y coordinates as well as width and height are read. Then the whole pixel array is read. From the canvas the pixel are obtained and validated. Then the control is given to collaborated canvas where the painting operation is performed. Any error during the reading of input data is informed to the user. There are two sub modules that come under this. They are CollabDesktop and CollabInitDialog. Each one of this performs a specific function.

Collabdesktop:

This is the start-up screen as soon as the user runs the viewer. The GUI for this constitutes a frame titled Collaborated Server and a Menubar. The option present in the menubar is AddServer. The CollabDesktop gets the information regarding tge user name and the Collaborated server name from CollabInitDialog. Hence the control is passed on the next module CollabInitDialog. After getting the correct user name and remote system name it starts a thread for that particular viewer. The viewer then gets connected to the VNC remote system and can view the desktop. Before we run the CollabDesktop a proper connection should be set between the Collaborated server and the VNC remote system.

Collabinitdialog:

The function of this module is to get the user name and the Collaborated server's name from the user. A GUI is designed for this purpose. The user is

expected to know the Collaborated server. This GUI consists of a frame titled CollabInitDialog, three labels and two textfields namely userfield and serverfield. Further it contains two buttons OK and CANCEL. It is this module that authenticates the user. If the userfield or serverfield is left blank then appropriate messages are displayed asking the user to enter the names. Once it finds that valid username and remote system name is entered it delivers them to CollabDesktop. The CollabDesktop then displays these names for our information.

➤ **UPDATION:**

Updation comprises of four classes. This is connected with the process of redrawing or repainting, where the contents of frame buffer are changed. The viewer requests the remote system to send the updated data through the RFB protocol and the remote system sends the updated information through the same, then the updated information should be reflected on the desktop of the remote system. For this purpose we consider that repainting or redrawing the image can effect the updation. The four classes used for this purpose are:

- a) Animatedmemoryimagesource
- b) Collabcanvas
- c) CollabReader

The functions of each class are given in the next page.

Animatedmemoryimagesource:

This class first reads the width, height, color model used and the pixel array. There is one image producer which in our case is the remote system and one

image consumer, the user or the viewer. The interfaces image producer and image consumer are used.

Each image contains an ImageProducer which is used to reconstruct the image whenever it is needed, for example, when a new size of the Image is scaled, or when the width or height of the Image is being requested. When a consumer is added to an image producer, the producer delivers all of the data about the image using the method calls defined in this interface. Then the dimensions, color model, pixels are all set in this class so that the consumer and producer have compatible properties. The ColorModel abstract class encapsulates the methods for translating a pixel value to color components (for example, red, green, and blue) and an alpha component. In order to render an image to the screen, a printer, or another image, pixel values must be converted to color and alpha components. As arguments to or return values from methods of this class, pixels are represented as 32-bit ints or as arrays of primitive types. The number, order, and interpretation of color components for a ColorModel is specified by its ColorSpace. A ColorModel used with pixel data that does not include alpha information treats all pixels as opaque, which is an alpha value of 1.0. This ColorModel class supports two representations of pixel values. A pixel value can be a single 32-bit int or an array of primitive types.

There is provision to add and remove consumer which in this context is the viewer. Once the verification of the consumer is over the image is sent. Any error in sending the frame is checked using the SINGLERAMEDONE. While setting the pixels the x, y coordinates, color model, height and width are also sent. It is also indicated the order in which pixels are sent the random pixels order is used. This is set by using setclips method. Thus the

Animatedmemoryimagesource identifies the source and destination of image and also the order in which pixels are sent.

Collab Canvas:

A Canvas component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user. An application must subclass the Canvas class in order to get useful functionality such as creating a custom component. The paint method must be overridden in order to perform custom graphics on the canvas.

In this class the color model is set and the viewer is identified. The image with the specified width and height is obtained from graphics. The preferred size and minimum size are obtained in this class. Using drawImage method the picture is painted in the current viewer. The picture is obtained after referring to Animatedmemoryimagesource for pixel array. Then using the new pixels obtained the desktop is updated. The x, y coordinates, width and height are used to find the exact position where the update has to take place. The method setClip is used for this purpose. By using setClip the coordinates of update rectangle are set. Thus only these coordinates have to be repainted. If there is no such method then error message is provided. In case of an exception when clip cannot be found create is used to create the clip rectangle and then the clipRect method is used to set the dimension of the rectangle to the size of the updated portion. The drawImage method is used for the purpose of drawing the new pixel in the viewer.

4.4 MODULE FEATURES

VNC – viewers:

Writing an VNC viewer is a simple task since 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 – remote system:

Writing a VNC remote system is slightly harder than writing a viewer for a number of reasons. The protocol is designed to make the viewer as simple as possible. So it is usually up to the remote system to perform any necessary translation.

Remote System Features:

Properties: This will cause the properties dialog to be displayed, allowing the user to change various remote system parameters.

Add new viewer: This allows outgoing connections to be made from the remote system to any “listening” viewer. The name of the target viewer machine can be entered in the dialog. Connections created this way are rated as shared.

Kill all viewers: This will disconnect all currently connected viewers from the remote system.

About remote system: this will display information about remote system.

Close: Shutdown the remote system.

Moving the mouse over the icon should cause the IP addresses of the local machine to be displayed, if they can be discovered at that time. You can connect to the remote system from another machine using a viewer.

Remote system Properties:

The following options are expected from the properties dialog.

Incoming connections:

Accept socket connections: The remote system normally accepts direct socket-based connections from the viewer program. Clearing this tick-box disables direct connection to remote system, so that only the CORBA interface used by our internal version may be used to start a connection.

Display number: This allows the user to specify the display number which the remote system will use. There is normally no need to change this from the default of zero.

Auto: This tick box indicates to remote system whether it should use the display number specified in the Display Number box, or whether it should use the first display number not already in use on the remote system machine.

Password: Incoming connections must be authenticated to verify that the person connecting is allowed to connect to this machine. This text box allows your password to be specified for authentication.

Disable remote keyboard & pointer: Any new incoming connections will be able to view the screen but not send any input.

Disable local keyboard & pointer: This is experimental. If selected, then the local keyboard and mouse will be disabled during a connection. Useful if you want to log in to a machine from elsewhere and don't want passers-by to be able to use your sessions.

Update Handling:

Note that clicking in a window will generally cause it to be updated. So if you have certain applications which don't update very well, you can use this! The default update handling setting should be the right ones for most people, and in general you will slow things down by changing them, so don't do this unless you have applications which cause problems.

Poll full screen:

Some applications are incompatible with the methods currently used in remote system to trap screen updates. For this reason, it is sometimes useful to be able to poll the entire screen in order to check for changes, sacrificing performance for accuracy.

Poll foreground window:

Polling only the currently selected window for changes is less CPU intensive than full-screen polling and often gives similar results.

Poll console windows only:

When this option is set, the only windows which will be ever be polled are Command Prompts. This works well in conjunction with Poll Window Under Cursor, to use polling only when the cursor is over a console window.

Poll on event received only:

When this option is set, the screen will only be polled for updates when a mouse or keyboard event is received from the remote viewer. This is provided for low bandwidth networks, where it may be useful to control how often the screen is polled and changes sent. The user's settings are saved into the user-specific section of the registry when remote system quits, meaning that they will be used next time you run server.

VIEWER FEATURES:

Disconnect: This option when set, causes the viewer to be disconnected from the server.

Option: In this, there will be option for selecting the type of encoding, to reverse the mouse button, to indicate sharable desktop mode.

Clipboard: When remote system selects a portion of text and copies that text will be displayed the clipboard to make the client aware of what server has copied.

Send CTR+ALT+DEL: This is used to unlock the workstation when Windows NT or 2000 is used.

Running remote system as a Service:

VNC- remote system can now be made to run as a service process under both Windows NT and Windows 95/98, by following the instructions outlined below. This allows you to unlock a locked workstation.

Collaborated Server and Viewer:

In the sharable desktop mode, if many viewers are connected to a remote system which seldom initiates updation, then it has to keep track of all the viewers that are connected to it. For every viewer it has to compare the status for update and has to send the desktop in case of remote system initiated updation. This will reduce the speed of remote system and it will be able to respond properly. If a viewer wants to connect to more than one remote system, it has to run many instance of viewer program. To avoid the above complexity, we are going for collaborated server and viewer concept.

The collaborated-server functions as an intermediate between the VNC-remote system and collaborated-viewer. It maintains the connection with each collaborated-viewer and also with the VNC- remote system.

Collaborated-viewer can only view the desktop of VNC- remote system and it cannot perform any updation in the VNC- remote system desktop, since it does not maintain any information about the collaborated-viewer.

As collaborated-server gets connected to the VNC- remote system in an exclusive mode, the VNC viewers that have been connected to the VNC- remote system will be disconnected. But a VNC-viewer can connect in a sharable mode to the VNC- remote system after the connection establishment of collaborated-server with the VNC- remote system. Now the VNC-viewer can control the Remote Desktop and the collaborated-viewers connected to the VNC- remote system through collaborated server can view them. Thus the workload of the VNC- remote system is taken up partially by the collaborated-server.

4.5 TESTING

Software Testing is the process of testing the software in a controlled manner to ensure it behaves the way it is expected to behave. Software testing is thus a critical element of software quality assurance. Testing requires that the developer discards preconceived notions of the correctness of the software thus developed and overcome the conflict of interest that occurs when errors are revealed.

Functional Test:

These specify operating condition, input value and expected results.

S.N	Test case	Object	Major Input	Output	Remarks
1	Checking for remote system's name in Collaborated Server's Remote system list	Remote System name	Sys24	"added to sys24" "Sending full desktop"	sys24 is a remote system name registered with the collaborated system.
			Sys25	"Error sending to sys25"	Sys25 is not a registered remote system.
2	Checking for Authentication schemes	Authentication Scheme number	1	"No authentication needed."	1 denotes that authentication is not required.
			2	Verification of password followed by return of authenticatin	2 insist that authentication is required.

				result.	
3	Password Verification.	Authentication result	0	Authentication succeeded	0 denotes correct password entry
			1	Authentication failed	1 denotes wrong password.
			2	Authentication failed-Too many Tries.	2 denotes too many trials of wrong password.
			3	Unknown VNC authentication result.	Invalid value entry.
4	Validation of Username Textbox entry.	Value entered in username textbox.	Null	Enter the user name	Prompts the user to enter the name.
			Sam	Textbox value entered.	

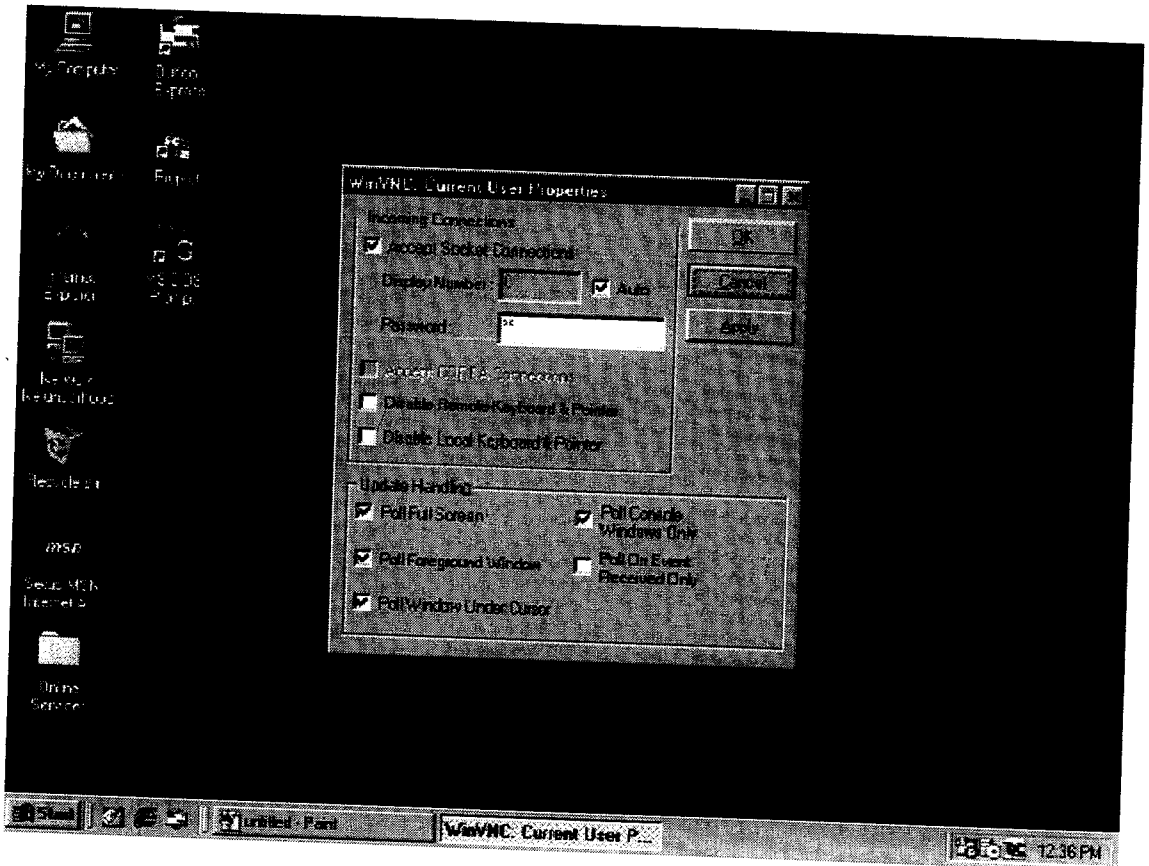
(TABLE 1) OPERATING CONDITION DURING TEST ANALYSIS

4.6 OUTPUT:

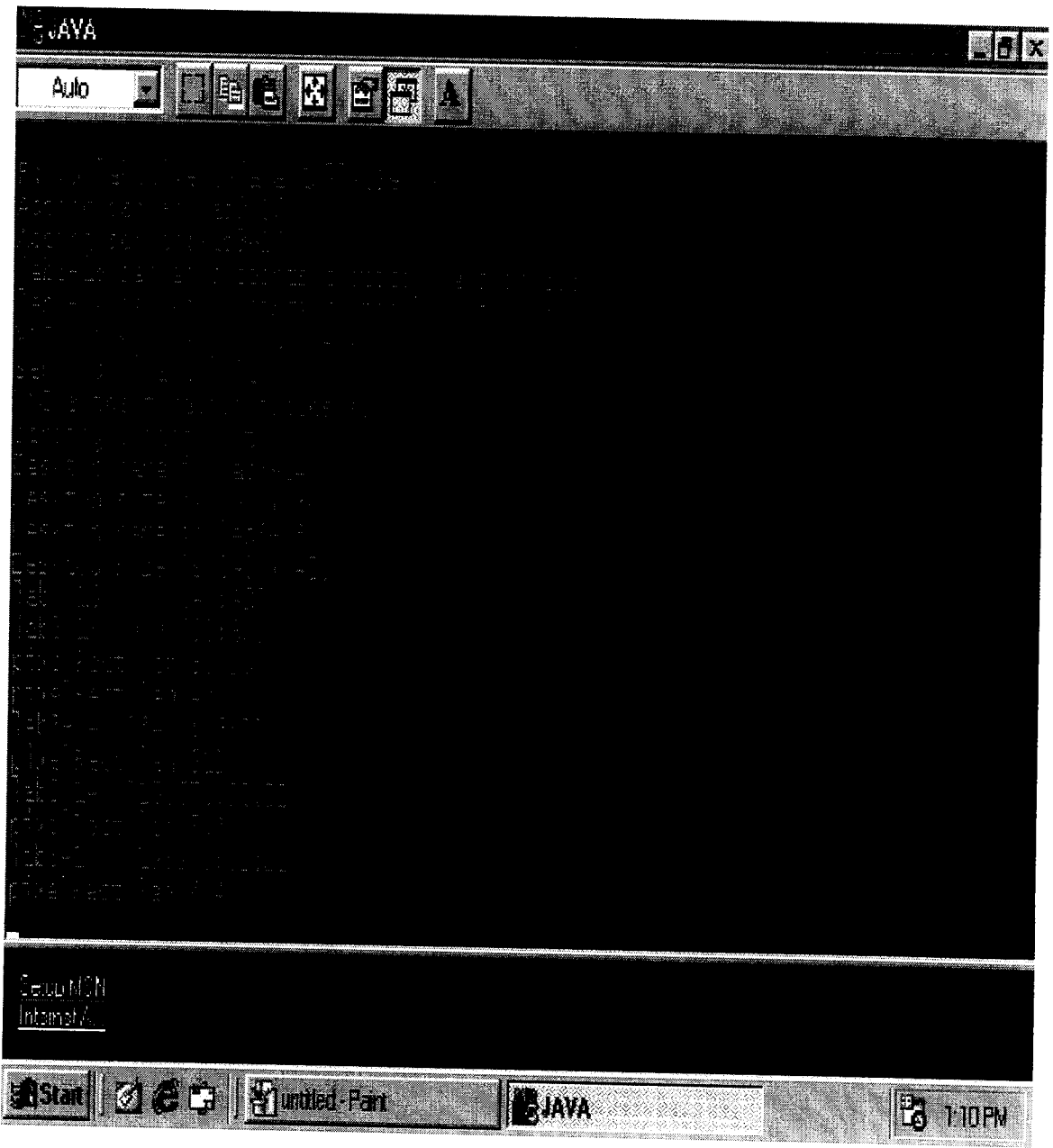
(FIG 10) RUNNING VNC REMOTE SYSTEM



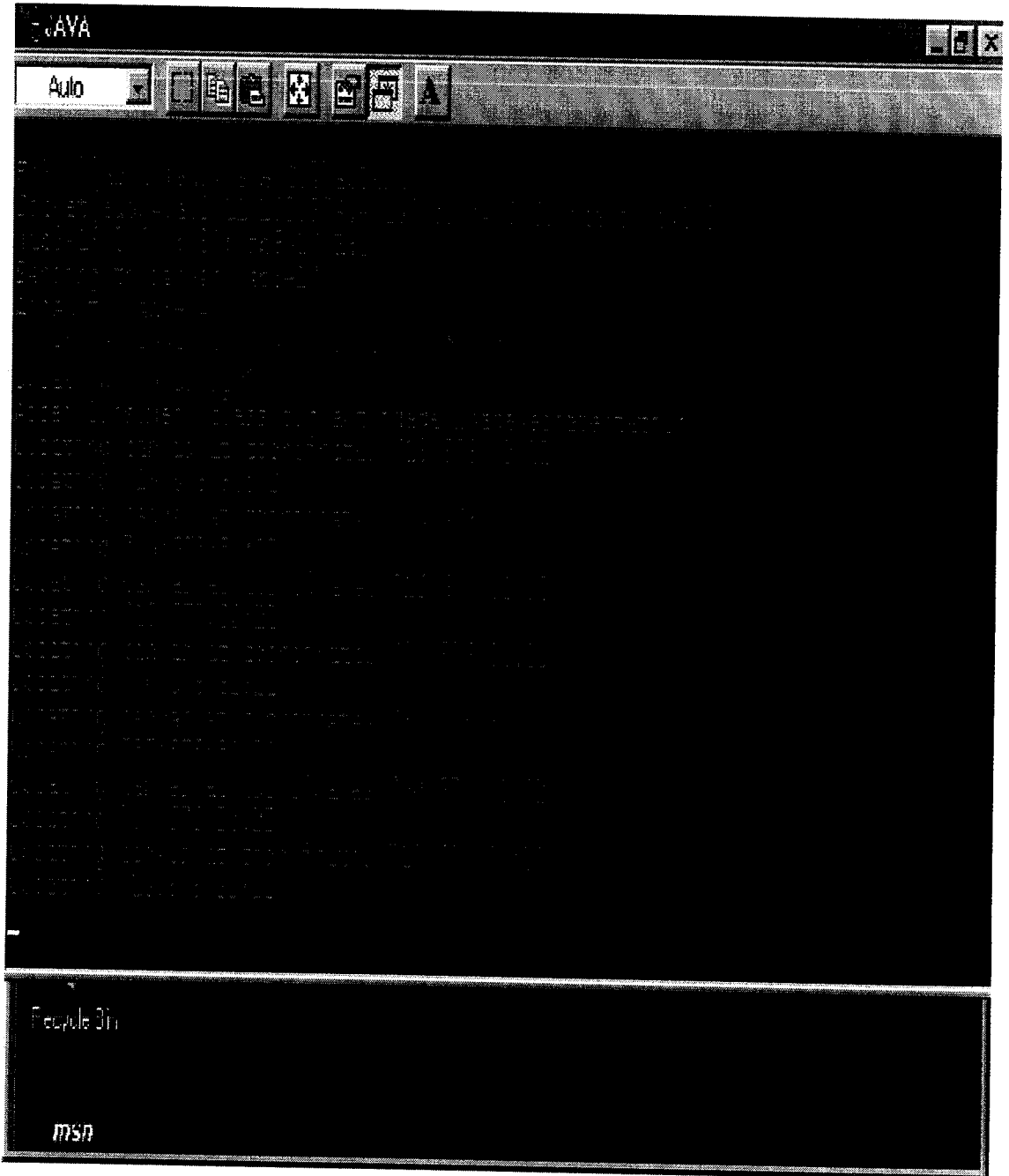
(FIG 11)SETTING VNC REMOTE SYSTEM PROPERTIES



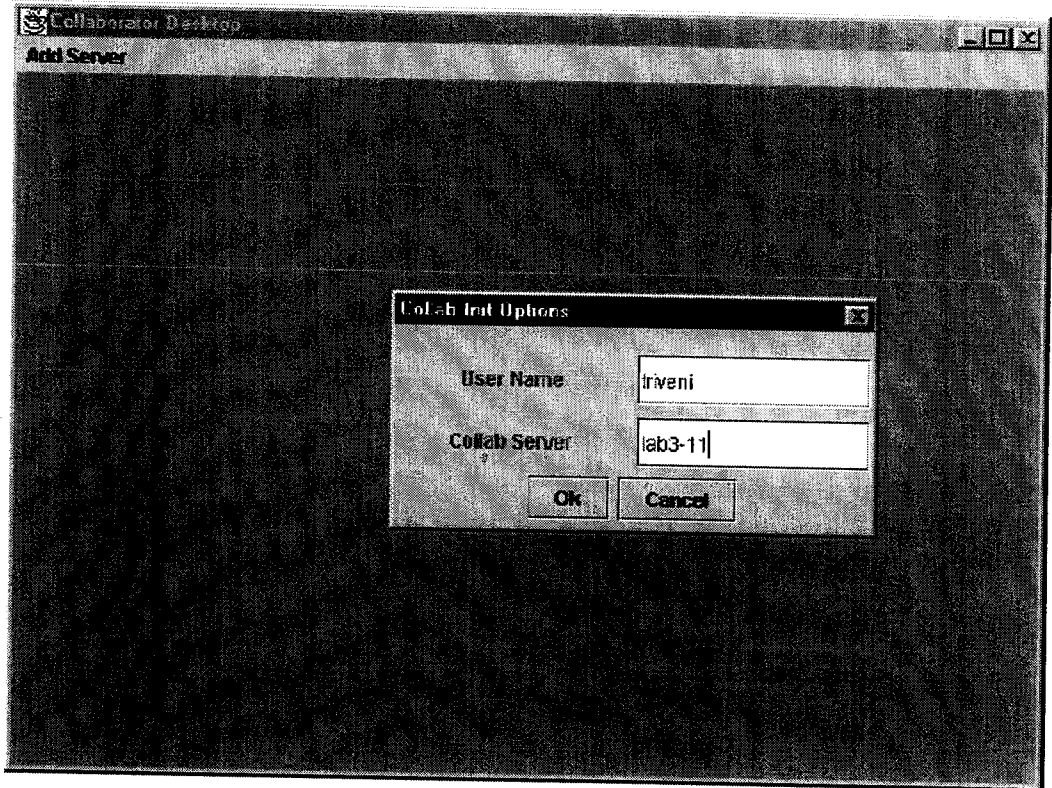
(FIG 12)RUNNING COLLABORATED SERVER



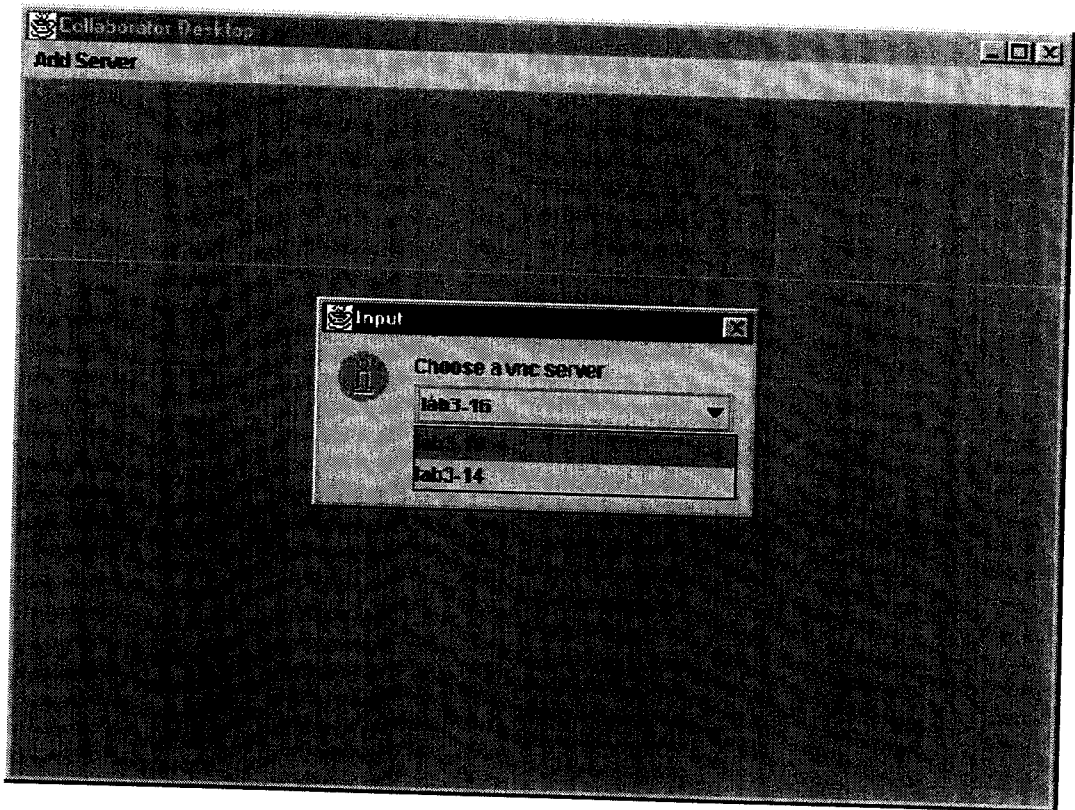
(FIG 13)RUNNING COLLOBORATED VIEWER:



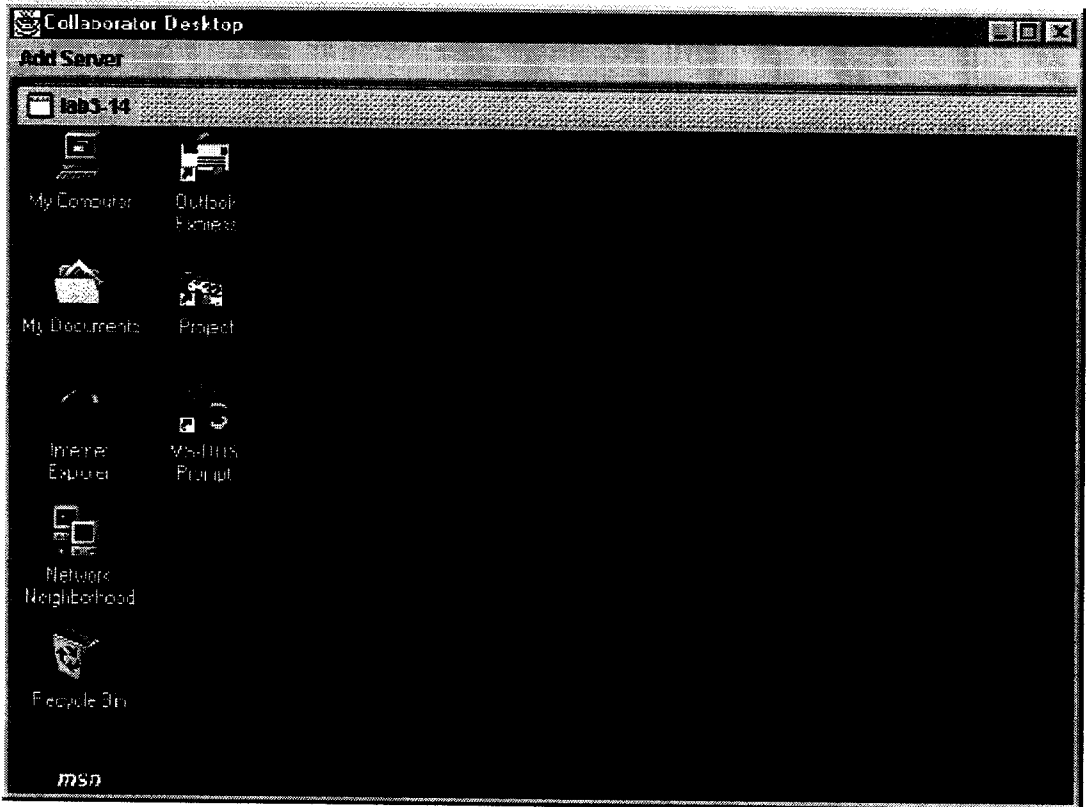
(FIG 14)CONNECTING TO THE COLLOBORATED SERVER:



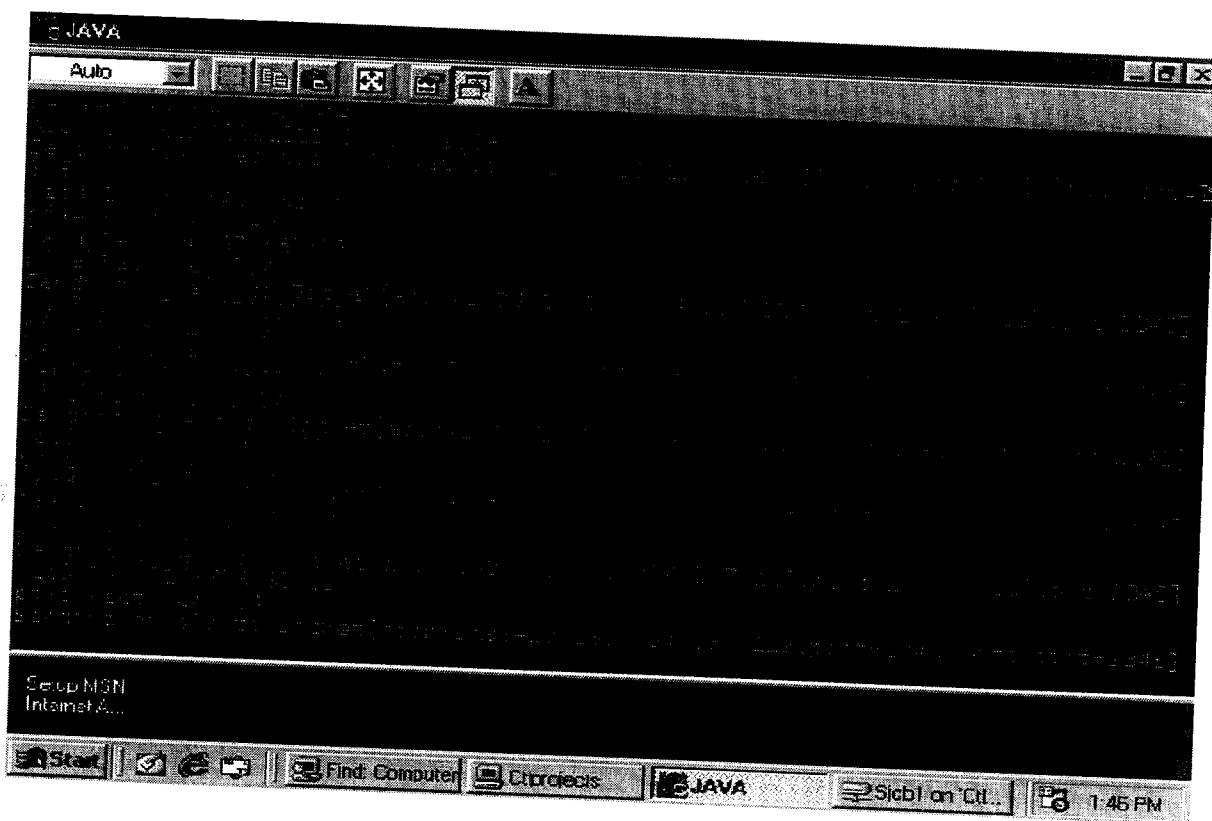
(FIG 15) CHOOSING A VNC REMOTE SYSTEM:



(FIG.16) COLLABORATED VIEWER VIEWING VNC REMOTE SYSTEM DESKTOP:



**(FIG.17)UPDATION INFORMATION OF VNC REMOTE SYSTEM
IN COLLOBORATED SERVER:**



2 1605

***PROJECT
FEATURES***

PROJECT FEATURES

5.1 ADVANTAGES:

- No state is stored at the viewer. The proposed solution is simple.
- It is truly platform-independent. The simplicity of the protocol makes it easy to port to new platforms. This is truly a "thin-viewer" protocol: it has been designed to make very few requirements of the viewer. In this way, viewers can run on the widest range of hardware, and the task of implementing a viewer is made as simple as possible.
- Collaborated Server will reduce the overhead of VNC remote system by handling updates and sending updates to various viewers connected to VNC remote systems.
- CollaboratedServer is handling All CollaboratedViewerSessionsHandling, Sending Updated Rectangles to all, so a single process is distributed across two machines, thus there is an increase in speed.
- No Separate Authentication for every Collaborated Viewer Session initiated against RFBRemote system other than Registering with Collaborated Server (since only one session is maintained with RFB remote system from Collaborated Server irrespective of number of Collaborated Viewers).

5.3 LIMITATIONS:

- Video conferencing on a slow bandwidth network is not achieved, since normal data transmission rate is 58/128 KBPS but as per our solution initial screen update itself required (as per raw encoding) approx 480 KB(800 X 600 resolution).
- Enabling RFBViewer Session also through the Collaborated Server thus no direct connections are required even from RFBViewer.

5.4 APPLICATIONS:

- Supports video broadcasting applications at suitable network bandwidth.
- Location and ManPower Transparency is achieved through easy mobile computing without requiring the user to carry any device whatsoever.
- Allows a desktop to be viewed from several places simultaneously, thus supporting application sharing in the style of computer supported co-operative work.
- If network bandwidth is high, video conferencing is also possible.

CONCLUSION

CONCLUSION:

The system was found to perform within the expected parameters satisfactorily. It managed to successfully overcome some of the drawbacks of the existing systems, while still remaining wholly self-contained.

The system was found to be suitably robust under test as well as in working conditions. All instances of technical glitches were debugged and rectified to a large extent. The system, thus, managed to meet all the requirements stated and even add a few extra features, while still maintaining its integrity.

FUTURE

ENHANCEMENTS

FUTURE ENHANCEMENTS:

- Enabling Video conferencing on a slow bandwidth network.
- Enabling RFBViewer Session through Collaborated Server.

APPENDICES

APPENDICES

APPENDIX 1:

SAMPLE CODE:

Collab client:

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

public class CollabClient extends JFrame implements
Runnable,InternalFrameListener{
public CollabCanvas canvas;
public byte[] pixels;
Socket s;
Thread thread;
BufferedReader br;
PrintWriter pw;
ServerDetails sd;
String userName,server;
public CollabClient(String server,String userName){
super(server,false,true,false,false);
```

```
this.server = server;
this.userName = userName;
}
public void start(){
try{
s = new Socket(server,2345);
System.out.println(s);
thread = new Thread(this);
thread.start();
}catch(IOException ioe){
System.out.println("Error: "+ioe);
}
}
public void stop() throws IOException{
thread = null;
s.close();
}
public void initComponents(){
try{
show();
canvas = new CollabCanvas(this);
getContentPane().add(canvas);
canvas.setGraphicsRefs();
canvas.setPixels(pixels);
validate();
}catch(IOException ioe){
System.out.println("Error: "+ioe);
```

```

}
}
public void update(){
}
public void internalFrameActivated(InternalFrameEvent e){}
public void internalFrameClosed(InternalFrameEvent e){}
public void internalFrameClosing(InternalFrameEvent e) {
try{
this.stop();
this.dispose();
}catch(IOException ioe){
system.out.println("Error closing socket "+ioe);
}}

public void internalFrameDeactivated(InternalFrameEvent e) {}
public void internalFrameDeiconified(InternalFrameEvent e) {}
public void internalFrameIconified(InternalFrameEvent e) {}
public void internalFrameOpened(InternalFrameEvent e) {}
private class ServerDetails{
String host;
int display,width,height;
public boolean equals(Object o){
if( !(sd instanceof ServerDetails) )return false;
ServerDetails sd= (ServerDetails) o;
if( !this.host.equals(sd.host) ) return false;
return true;
}
public String toString(){

```

```

return host;
}
}
public void run() {
try{
br = new BufferedReader(new InputStreamReader(s.getInputStream()));
pw          =          new          PrintWriter(new
OutputStreamWriter(s.getOutputStream()),true);
pw.println(userName);
String srvsAvail = br.readLine();
System.out.println(srvsAvail);
Vector servers =new Vector();
StringTokenizer stTok = new StringTokenizer(srvsAvail,":");
while(stTok.hasMoreElements()){
ServerDetails sd = new ServerDetails();
sd.host = (String) stTok.nextElement();
sd.display = Integer.parseInt((String) stTok.nextElement());
sd.width = Integer.parseInt((String) stTok.nextElement());
sd.height = Integer.parseInt((String) stTok.nextElement());
servers.addElement(sd);
}
Object[] objArr = new Object[servers.size()];
for(int i = 0;i<servers.size();i++){
objArr[i] = ((ServerDetails)servers.elementAt(i)).host;
}
}

```

```

String serverhost = (String)JOptionPane.showInputDialog(null,"Choose a
vnc server", "Input", JOptionPane.INFORMATION_MESSAGE,
null,objArr, null);
Enumeration enum = servers.elements();
while(enum.hasMoreElements()){
ServerDetails sdtemp = (ServerDetails)enum.nextElement();
String actualhost = sdtemp.host;
if( actualhost.equals(serverhost) ){
sd = sdtemp;
break;
}
}
if( sd == null ){
System.out.println("Server not found:"+serverhost);
return;
}
setTitle(sd.host);
setSize(sd.width+10,sd.height+30);
pixels = new byte[sd.width*sd.height];
System.out.println("Sending to server "+sd.host);
pw.println(sd.host);
initComponents();
DataInputStream dis = new DataInputStream(s.getInputStream());
System.out.println(dis.readLine());
int x=0,y=0,w=0,h=0;
while(thread != null){
String rectInfo = dis.readLine();

```

```

//System.out.println("updating canvas "+rectInfo+".");
stTok = new StringTokenizer(rectInfo,":");
x = Integer.parseInt((String)stTok.nextElement());
y = Integer.parseInt((String)stTok.nextElement());
w = Integer.parseInt((String)stTok.nextElement());
h = Integer.parseInt((String)stTok.nextElement());
//dis.readFully(pixels,0,pixels.length);
for (int j = y; j < (y + h); j++) {
dis.readFully(pixels, j * sd.width + x, w);
}
System.out.println("updating          canvas          at          coordinates
"+x+": "+y+": "+w+": "+h);
canvas.updateCanvas(x,y,w,h);
}
} catch(IOException ioe){
System.out.println("Error opening / reading I/O streams to "+s);
return;
} } }

```

collab desktop:

```

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.util.*;

public class CollabDesktop extends JFrame{

```

```
JDesktopPane jdp;
CollabClient cc;
JMenuBar mainMenu;
JMenu fileMenu,addMenu;
String user,server;
public CollabDesktop(){
jdp = new JDesktopPane();
getContentPane().add(jdp);
this.setTitle("Collaborator Desktop");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
mainMenu = new JMenuBar();
addMenu = new JMenu("Add Server");
mainMenu.add(addMenu);
this.setJMenuBar(mainMenu);
addMenu.addMenuListener(new MenuListener(){
public void menuCanceled(MenuEvent e) {}
public void menuDeselected(MenuEvent e) {
}
public void menuSelected(MenuEvent e) {
if( e.getSource() != addMenu ) return;
getParams();
}
});
Dimension deskdim = Toolkit.getDefaultToolkit().getScreenSize();
this.setSize(deskdim.width,deskdim.height);
//this.setExtendedState(JFrame.MAXIMIZED_BOTH);
this.setVisible(true);
```

```

getParams();
}
public void getParams(){
    CollabInitDialog cid =new CollabInitDialog(this,true);
    this.server = cid.getCollabServer();
    this.user = cid.getUserName() + new Date().getTime();
    cid.dispose();

    if( user != null && server != null){
        startNewCanvas();
    }
}

public void startNewCanvas(){
    cc = new CollabClient(server,user);

    cc.start();
    jdp.add(cc);
}

public static void main(String args[]){
    new CollabDesktop();}}

```

collab server:

```

import java.util.*;
import java.net.*;
import java.io.*;
public class CollabServer{

```



```

public Vector servers;
public ServerSocket ss;
public static void main(String args[]){
new CollabServer();
}
public CollabServer(){
servers = new Vector();
readParameters();
try{
ss = new ServerSocket(2345);
Enumeration enum = servers.elements();
while(enum.hasMoreElements()){
new CollabConnector(((ServerInfo)enum.nextElement()));
}
Socket s = null;
while( (s = ss.accept()) !=null )
{
new CollabClientServiceThread(this,s).start();
}
}catch(IOException ioe){
System.out.println("Error opening server socket");
}
}
public void readParameters() {
ResourceBundle properties = null;
try
{

```

```

properties = ResourceBundle.getBundle( "collab" );
}
catch( MissingResourceException x )
{
System.err.println( "Can't load 'collab.properties'." );
return;
}
String key=null, host=null, password=null ;
int display, width, height, port, mcport;
// Create RFB hosts and web servers
for( Enumeration e = properties.getKeys(); e.hasMoreElements(); )
{
key = (String) e.nextElement();
if( key.endsWith( ".server" ) )
{

display = Integer.parseInt( key.substring( 0, key.indexOf( '.' ) ) );
host = properties.getString( key );
width = Integer.parseInt( properties.getString( display + ".width" ) );
height = Integer.parseInt( properties.getString( display + ".height" ) );
port = Integer.parseInt( properties.getString( display + ".port" ) );
password = properties.getString( display + ".password" );
mcport = Integer.parseInt( properties.getString( display + ".mcport" ) );
servers.addElement(new
ServerInfo(display,host,password,width,height,port,mcport));
System.out.println("Adding server:"+host);
} } } }

```

Remote Systeminfo:

```
public class ServerInfo {
    int display,width,height;
    int port,mcport;
    String host,password;
    byte[] pixels;
    CollabConnector cc;
    public ServerInfo(int display,String host,String password){
        this(display,host,password,800,600,5900,9999);
    }
    public ServerInfo(int display,String host,String password,int width,int
height){
        this(display,host,password,width,height,5900,9999);
    }

    public ServerInfo(int display,String host,String password,int width,int
height,int
port ,int mcport){
        this.display = display;
        this.host = host;
        this.password = password;
        this.width = width;
        this.height = height;
        this.port = port;
```

```
this.mcport = mcport;
}
}
```

collab client server connection:

```
import java.net.*;
import java.io.*;
import java.util.*;
public class CollabClientServiceThread extends Thread {
CollabServer cs;
Socket s;
BufferedReader br;
PrintWriter pw;
ServerInfo si;
public CollabClientServiceThread(CollabServer cs,Socket s){
this.cs =cs;
this.s = s;
}
public void run(){
System.out.println("Processing :"+s);
InetAddress addr= s.getInetAddress();
String name,reqServer;
try{
br = new BufferedReader(new InputStreamReader(s.getInputStream()));
pw = new PrintWriter(new OutputStreamWriter(s.getOutputStream()),true);
name = br.readLine();
```

```

} catch(IOException ioe){
System.out.println("Error opening / reading I/O streams to "+addr);
return;
}
try{
pw.println(formatServerInfo());
reqServer = br.readLine();
System.out.println("request server "+reqServer);
if( checkClientList(name,reqServer) ){
pw.println("Added to "+reqServer);
pw.println("0:0:"+si.width+"."+si.height);
pw.flush();
BufferedOutputStream
bos=newBufferedOutputStream(s.getOutputStream());
//s.getOutputStream().write(si.cc.cr.pixels);
bos.write(si.cc.cr.pixels);
addClientToCollabConnector(name,reqServer);
System.out.println("Sending full desktop: "+s);
}else{
pw.println("Error adding to "+reqServer);
}
} catch(IOException ioe){
System.out.println("Error:"+ioe);
return;
} }

public String formatServerInfo(){
Enumeration enum = cs.servers.elements();

```

```

String formatString="";
while(enum.hasMoreElements()){
ServerInfo si = (ServerInfo)enum.nextElement();
formatString = formatString+ ((formatString.length()>0)?":":"" ) +si.host;
formatString = formatString + ":"+ si.display;
formatString = formatString + ":"+ si.width;
formatString = formatString + ":"+ si.height;
}
return formatString;
}

public boolean checkClientList(String userName,String ccServer){
Enumeration enum = cs.servers.elements();
while(enum.hasMoreElements()){
ServerInfo sitemp = (ServerInfo) enum.nextElement();
if( sitemp.host.equals(ccServer) ){
si = sitemp;
break;
}
}
if( si == null){
System.out.println("Server list does not contain "+ccServer);
return false;
}
if( si.cc.users.containsKey(userName) ){
System.out.println(si.host+ "already has user named"+userName);
return false;
}
}

```

```
return true;
}
public boolean addClientToCollabConnector(String userName,String
ccServer){
    if(si == null) return false;
    si.cc.users.put(userName,s);
    System.out.println("Adding "+userName+" to "+si.host);
    return true;
}
}
```

APPENDIX 2:

USER MANUAL

RUNNING THE VNC Remote System

The following steps are included in establishing a connection between a VNC-remote system and a VNC-viewer. The VNC-remote system is started by setting the port number, password, connection type and updation handling option which listens for connection request in the specified socket.

- The VNC-viewer sends the IP address, Port Number and Password to the remote system.
- The VNC-remote system checks the Password sent by viewer.
- Then there will be a negotiation between the VNC-viewer and VNC-remote system regarding Protocol version, Frame buffer size, Pixel format and encoding schemes used.
- Then VNC-viewer sends the initialization message to remote system regarding the connection mode either shared or exclusive.

Finally the remote system sends the desktop to VNC-viewer and now viewer views the remote desktop. Either the remote system or the viewer is also allowed to update the server's desktop. When the remote system initiates the updation, it will read the frame buffer content of its display

and sends that to viewer. The viewer paints the desktop according to the information received.

RUNNING THE COLLABORATED SERVER

- Install the Collaborated Server software in the local directory
- Right click the properties and specify the number of remote systems to be connected ,Port no (uses a default number of 5900),width and height of the screen to be captured and the password specified in WinVNC Remote system.Save the changes made in the property screen.
- Right click the RunCode option and edit the path.Set the path in accordance with the location where the software is installed in the local directory.
- Thus the Collaborated Server starts its execution by displaying the protocol version, updating the pixel rectangle, the x,y co-ordinate and also the length and width of the updated screen.

The CollabProperties plays an important role in the connection process. Resource bundles contain locale-specific objects. When your program needs a locale-specific resource, a String for example, your program can load it from the resource bundle that is appropriate for the current user's locale. In this way, you can write program code that is largely independent of the

user's locale isolating most, if not all, of the locale-specific information in resource bundles. This allows you to write programs that can:

- Be easily localized, or translated, into different languages
- Handle multiple locales at once
- Be easily modified later to support even more locales

A sample of information contained in CollabProperties is shown below:

1. Server=tcms
1. Port=5900
1. width=800
1. height=600
1. password=at
1. mcport=9999

REMOTE SYSTEM SIDE UPDATION

- The remote system triggers an event.
- Then the remote system processes the event and updates its desktop.
- Remote system compares the updated desktop with the current desktop of viewer, which is stored by the remote system for each viewer.

- Then remote system generates updated rectangle, by specifying its x, y portion, height and width.
- The remote system perform compression, encryption and transmit it to viewer.
- Finally the viewer decompresses and decrypts the information and renders it on the screen

REFERENCES

REFERENCES:

BIBLIOGRAPHY:

- 1) Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper, "Virtual Network Computing", Vol.2, Jan-Feb1998
- 2) Patrick Naughton, Herbert Schildt, "Java 2 The Complete Reference", Third edition, Tata McGraw- Hill publication Company Limited.
- 3) Glenn L. Vanderburg et al., "Tricks of Java Programming Gurus", First edition, Sam's .NET Publishing Company Limited.
- 4) Matthew T. Nelson, "Java Foundation Classes", Third edition, Sun Microsystems, 1980.

WEBSITES REFERRED

1. WWW.Uk.Research.Co.In
2. WWW.Google.Com
3. WWW.howstuffworks.com