

**DESIGN AND IMPLEMENTATION OF A SOFTWARE
INTERCOM ON A LOCAL AREA NETWORK**

BY

**ALI-NOCK GYET
99/8087EE**

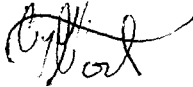
**DEPARTMENT OF ELECTRICAL/COMPUTER ENGINEERING
SCHOOL OF ENGINEERING AND ENGINEERING TECHNOLOGY
FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA
NIGER STATE, NIGERIA.**

**A PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT FOR
THE AWARD OF BACHELOR OF ENGINEERING DEGREE (B.ENG.)
IN ELECTRICAL/COMPUTER ENGINEERING**

NOVEMBER 2005

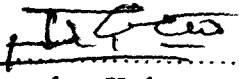
ATTESTATION/DECLARATION

This is to attest that this project "Design and Implementation of a Software Intercom on a Local Area Network (LAN)" was designed and implemented by **ALI-NOCK GYET** for the award of bachelor degree in Electrical/Computer Engineering of the Federal University of Technology, Minna, Niger State, Nigeria.



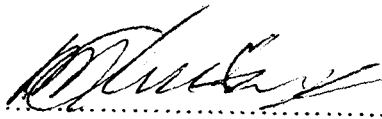
.....
STUDENT

9/12/2005
.....
DATE



.....
Mr. Jonathan Kolo
PROJECT SUPERVISOR

9/12/05
.....
DATE



.....
Engr. Musa Abdullahi
HEAD OF DEPARTMENT

27/02/06
.....
DATE

.....
EXTERNAL EXAMINER

.....
DATE

ACKNOWLEDGEMENT

I would want to first of all say thank you to my parents Mr. and Mrs. M. I. Ali-Nock, whom have loved and nurtured me unconditionally and have given me the room and support to grow and explore my true potential,

I would also love to acknowledge my departmental lecturers, through whom I have been groomed to face the challenges of work as an engineer. I want to also show appreciation to my supervisor Mr. Jonathan Kolo, whom through his patience and constructive criticism has enabled me to complete this project with ease.

Finally I want to thank my Lord God Almighty, through whom my existence was made possible, thank you for being my Strongest support.

DEDICATION

This project is dedicated to my closet friends. With their support and friendship I have come to appreciate the influence of positive thinking people on a person's life and realize my true potential. They inspire me to reach for greater heights.

ABSTRACT

This project represents a branch of Voice over IP (VoIP) technology which in itself incorporates many aspects of digital signal processing. Quick, effective, reliable, efficient, and cheap means of communication has necessitated the need to finding new solutions to improving telecommunication which is the backbone in the operation and day to day running of most businesses and institutions today. The need for information at the press of a button cannot be over-emphasized.

This project stands as the foundation of the revolution of multimedia communication today. The project enables the use of a Local Area Network (LAN, which are mostly dedicated to data traffic only) installed in an office area as a means of communication using voice and video media, presented in real-time between peers by the use of a software. It also has an extra functionality of voice conferencing and multicast of live events via video over the network. The project also hopes to provide a solution which eliminates the use of expensive hand-wired intercoms and Public Branch Exchange (PABX), allowing the use of only the LAN network and readily available computer hardware, in essence giving the user a truly Multimedia experience.

TABLE OF CONTENT

Certification	i
Dedication	ii
Acknowledgement	iii
Table of Content	iv
Abstract	v
Chapter One	
1.0 Introduction	1
1.1 Project overview	1
1.2 Statement of problem	2
1.3 Rationale for the project	2
1.4 Objectives of the project	3
Chapter Two	
2.1 An overview of digital signal processing	4
2.2 An overview of multimedia communication on digital networks	9
2.3 Audio Communication	15
2.4 Video communication	18
2.5 Summary of multimedia communication	19

Chapter Three

3.0 Program Analysis and Design	20
3.1 Why Java?	20
3.2 Program Design	24

Chapter Four

4.0 System Requirement and Implementation	27
4.1 software requirements	27
4.2 Hardware requirements	28
4.3 Program Implementation	29
4.4 Program segments	31

Chapter Five

5.0 Conclusion and Recommendation	34
5.1 Project conclusion	34
5.2 Recommendations	35
Reference	36
Appendixes	37

CHAPTER ONE

INTRODUCTION

1.1 PROJECT OVERVIEW

In this day and age, digital technology is the backbone of our entire information industry. As a part of this, the transformation of audio information into digital signals is now a routine process which is incorporated into our telephone, digital networks, televisions and music equipments.

The invention of the transistor in 1947 and later evolution of semiconductor microelectronics techniques have facilitated the digital revolution and ushered in the electronic age, giving us the capabilities we have today in digital communications. Voice communication has traditionally been carried over dedicated Telephone networks operated by Telecommunication service providers such as the NITEL and GLO, e.t.c in Nigeria. These telephone networks have progressively evolved from the initial analog circuits to the current digital networks with bandwidth in excess of 1 Gbps. For reasons of varying bandwidth and networking requirements, different services were provided on separate networks. For example, Telegraph networks, Telex networks, Telephone networks, Facsimile networks, Cable networks and Data networks support different services, as their names would suggest. These networks possessed characteristics that satisfied the peculiar requirements of the service they provided. For example, the voice network would support bandwidths of 64 Kbps for voice communication and would ensure telco-grade voice communication with little jitter and echo cancellation. Likewise, the cable networks would provide even higher bandwidth and improved quality of service (QoS) for video transmission. On the other hand, the data communication networks'

bandwidth and QoS requirements are highly flexible.. For most types of data communication applications, reliability is critical, which means that the delivery protocols would implement mechanisms for error checking, acknowledgment, re-transmissions and sequencing. On the other hand, for real-time applications such as voice communications, it would make little sense to retransmit a lost packet for play back at the receiving end, if it is out of sequence and is considerably delayed. Essentially, the main point to be noted is that these networks have been designed differently in terms of their underlying architecture and communication protocols.

1.2 STATEMENT OF THE PROBLEM

With the immense growth of digital networks, which is the marriage of two technologies i.e. telecommunication technology and computer technology. Networks are being explored to the fullest. Ways in which existing networks can be used optimally, with minimum additional cost. For example, facilities like video conferencing, mail serving, can be incorporated into the network, saving cost, granting easier access to remote database and remote programs.

Integrating these networks into a single integrated network, such that all services would use common facilities, presents a technological hurdle.

This project therefore is designed to provide a service which transmits voice and video over a Data network, as against having dedicated voice and cable networks meant for either voice or video communication only.

1.3 RATIONALE FOR THE PROJECT

Certain advantages could be gained from having an all purpose network as against having only dedicated networks. Below are some reasons why this evolution in digital

communication is deemed necessary.

1.3.1 Cost Reduction

With the implementation of a software based intercom, the need for expensive PABX, Telephone handsets would be eliminated since the only hardware requirement needed would be headset and a computer on the network. Moreover, with the prevalence of IP nodes and the abundant supply of IP based switches and routers, communication might not be limited to just local area networks(LAN), but may include Wide Area Networks(WAN), thereby reduced the cost of making long distance calls.

1.3.2 Simplification and Consolidation

An integrated infrastructure that supports all forms of communication could allow more standardization and reduce the total equipment compliment. The differences between the traffic patterns of voice and data offer further opportunities for significant improvement in efficiency. Simplified installation and maintenance are a major advantages the project will offer.

1.4 OBJECTIVES OF THE PROJECT

The major aim of this project is to develop a network based software intercom where anyone on the network can dial a peer that is logged on the network. This is an intercom without the PABX and expensive handsets. Another aim of the software is to provide video peer to peer conferencing, where two or more clients can see each in real time using web cams or any other video capture devices connected to their computers.

CHAPTER TWO LITERATURE REVIEW

2.1 AN OVERVIEW OF DIGITAL SIGNAL PROCESSING (DSP)

Digital Signal Processing is one of the most powerful technologies that has shaped Science and Engineering in this century. Revolutionary changes have been made in a broad range of fields: communication, medical imaging, radar and sonar, high fidelity music reproduction, and oil prospecting to name just a few. Each of these areas has developed a deep DSP technology, with its own algorithms, mathematics, and specialized techniques. DSP in essence is the technology making possible this project possible(Software Intercom).

2.1.1 WHAT IS DIGITAL SIGNAL PROCESSING (DSP).

Digital Signal Processing is distinguished from other areas of computing by the Type of data it uses: signals. In most cases, these signals originate as sensory data from the real world i.e. seismic vibration, visual images, sound waves etc. DSP is the mathematics, the algorithm and the technique used to manipulate these signals after they have been converted into digital form by transducers.

2.1.2 WHY PROCESS SIGNALS DIGITALLY

Certain advantages are gained from processing real world sensory signal into digital signals. The following are some the reasons why.

1. Enhancement of visual images.
2. Recognition and generation of speech.
3. Compression of data for storage and transmission.
4. Encrypting of signals for security purposes.

The application of DSP is limitless and has application in areas of commerce, military, research and the day to day running of most house holds. The remainder of this chapter illustrates areas in which has had an effect, especially with regards to the technologies behind the workings of this project.

2.1.3 TELECOMMUNICATION

Telecommunication is about transferring information from one location to another. This includes many forms of information: telephone conversations, television signals, computer files, and other data types. To transfer the information, a channel between the two locations is needed. This may be a wire pair, radio signals, optical fiber, etc. Telecommunication companies receive payment for transferring their customer's information, while they must pay to establish and maintain the channel. The financial bottom line is simple; the more the information they can pass through a single channel, the more money they make. DSP has revolutionized the telecommunications industry in many areas: signaling tone generation and detection, frequency band shifting, filtering to remove power line hum etc. Three specific technologies would be discussed here:

Multiplexing, Compression, and Echo control.

2.1.3.1 MULTIPLEXING

There are approximately one billion telephones in the world. At the press of a few Buttons, switching networks allow any one of these to be connected to any other in only a few seconds. The immensity of this task is mind boggling! Until the 1960s, a connection between two telephones required passing the analog voice signals through mechanical switches and amplifiers. One connection required one pair of wires. In comparison, DSP converts audio signals into a stream of serial

digital data. Since bits can be easily intertwined and later separated, many telephone conversations can be transmitted on a single channel. For example, a telephone standard known as the T-carrier system can simultaneously transmit 24 voice signals. Each voice signal is sampled 8000 times per second using 8 bit compounded (logarithmic compressed) analog-to-digital conversion. This results in each voice signal being represented as 64,000 bits/sec, and all 24 channels being contained in 1.544 megabits/sec. This signal can be transmitted about 6000 feet using ordinary telephone lines of 22 gauge copper wire, a typical interconnection distance. The financial advantage of digital transmission is enormous. Wire and analog switches are expensive; digital gates are cheap.

2.1.3.2 COMPRESSION

When a voice signal is digitized at 8000 samples/sec, most of the digital information is redundant. That is, the information carried by any one sample largely duplicated by neighboring samples. Dozens of DSP algorithms have been developed to convert digitized voice signals into streams that require fewer bits/sec. These are called data compression algorithms. Matching decompression algorithms are used to restore the signals to its original form. These algorithms vary in the amount of compression achieved and the resulting sound quality. In general, reducing the data rate from 64 kilobits/sec to 32 kilobits/sec results in no loss of sound quality. When compressed to a data rate of 8 kilobits/sec, the sound is noticeably affected, but still usable for long distance telephone networks. The highest achievable compression is about 2 kilobits/sec, resulting in sound that is highly distorted, but usable for some applications such as military and

undersea communications.

2.1.3.3 ECHO CONTROL

Echoes are a serious problem in long distance telephone connections. When you speak into a telephone, a signal representing your voice travels to the connecting receiver, where a portion of it returns as an echo. If the connection is within a hundred miles, the elapsed time for receiving the echo is only a few milliseconds. The human ear is accustomed to hearing echoes with these small time delays, and the connection sounds quite normal. As the distance becomes larger, the echo becomes increasingly noticeable and irritating. The delay can be several hundred milliseconds for international communications, and is particularly objectionable. Digital Signal Processing attacks this type of problem by measuring the returned signal and generating an appropriate anti-signal to cancel the offending echo. This same technique allows speakerphone users to hear and speak at the same time without fighting audio feedback (squealing). It can be used to reduce environmental noise by canceling it with digitally generated anti-noise.

2.1.4 AUDIO PROCESSING

The two principal human senses are vision and hearing. Correspondingly, much of DSP is related to image and audio processing. People listen to both music and speech. DSP has made revolutionary changes in both these areas.

2.1.4.1 Music

The path leading from the musician's microphone to the audiophile's speaker is remarkably long. Digital data representation is important to prevent the

degradation commonly associated with analog storage and manipulation. This is very familiar to anyone who has compared the musical quality of cassette tapes with compact disks. In a typical scenario, a musical piece is recorded in a sound studio on multiple channels or tracks. In some cases, this even involves recording individual instruments and singers separately. This is done to give the sound engineer greater flexibility in creating the final product. The complex process of combining the individual tracks into a final product is called mix down. DSP can provide several important functions during mix down, including: filtering, signal addition and subtraction, signal editing, etc.

One of the most interesting DSP applications in music preparation is artificial reverberation. If the individual channels are simply added together, the resulting piece sounds frail and diluted, much as if the musicians were playing outdoors. This is because listeners are greatly influenced by the echo of reverberation content of the music, which is usually minimized in the sound studio. DSP allows artificial echoes and reverberation to be added during mix down to simulate various ideal listening environments. Echoes with delays of a few hundred milliseconds give the impression of cathedral like locations.

Adding echoes with delays of 10-20 milliseconds provide the perception of more modest size listening rooms.

2.1.4.2 Speech Generation

Speech generation and recognition are used to communicate between humans and machines. Rather than using one's hands and eyes, the mouth and ears are used instead. Two approaches are used for computer generated speech: digital

recording and vocal tract Simulation. In digital recording, the voice of a human speaker is digitized and stored, usually in a compressed form. During playback, the stored data are uncompressed and converted back into an analog signal. Vocal tract simulators are more complicated trying to mimic the physical mechanisms by which humans create speech. The humans vocal tract is an acoustic cavity with resonant frequencies determined by size and shape of the chambers. Sound originates in the vocal tract in one of two ways, called voiced and fricative sounds. With voiced sounds, vocal cord vibration produces near periodic pulses of air into vocal cavities. In comparison fricative sounds originate from the noisy air turbulence at narrow constrictions, such as the teeth and lips. Vocal tract simulators operate by generating digital signals that resemble these two types of excitation. The characteristics of the resonate chamber are simulated resonances. The approach was used in one of the very early DSP success stories, the Speak & Spell, a widely sold electronic learning aid for children.

2.2 AN OVERVIEW OF MULTIMEDIA COMMUNICATION ON DIGITAL NETWORKS

In the past, separate infrastructures were required to support different types of communication. Most workstations are still equipped with both a telephone and a personal computer (PC) to support voice and data communication. The telephone is the user's front end the Private Branch Exchange (PBX) or other telephony switch, while the PC is an intelligent device that uses LANs and WANs to communicate with other devices.

Technological innovations have eroded the distinctions between previously distinct communication technologies. Advances in components technologies have enabled the development of increasingly sophisticated and specialized applications. Multimedia communication, for example, owes its existence to advances in microprocessors are continuing to increase in power and speed while steadily becoming more affordable. Similar advances have been made in the various types of magnetic storage devices.

These advantages encouraged creative attempts to exploit their potential. One creative attempt was the digitization of data that traditionally existed in other forms. Today, numerous data type converging into a common format: digital encoding. Integrating multiple application types that use digitally encoded data into a single host platform, that is, a computer has become euphemistically known as “multimedia computing.”

Some of the more common multimedia applications include

1. Computer-based telephony
2. Video conferencing
3. Video transmission
4. Audio transmission
5. High-density graphics

Multimedia communication can be equally difficult to define; *multimedia communication* means the integration of multiple data types into a common bit stream.

A subtle but important point is that multimedia computing and multimedia communication are not completely synonymous. For example, a LAN can support client

machines that are used exclusively for traditional forms of data communication and also provide networking for machines that are dedicated to provide video or audio service for external clients. Consequently the LAN is simultaneously transporting multiple media types with extremely different performance requirements, even though no true multimedia computers are directly connected to it.

Multimedia applications typically impose two basic types of performance requirements: latency and bandwidth.

2.2.1 LATENCY

Latency is defined as the minimum amount of time required for a packet to clear any given network device, such as a router, a hub, a switch and so on. Time sensitive applications like live video or video conferencing require low-latency connections throughout the entire network. A low latency connection, ideally, provides a and consistent delay between the transmission and receipt of a packet. Low but inconsistent delays can result in jittery images, choppy sounds, or otherwise degraded performance of networked multimedia applications.

2.2.1.1 LAN Access Methods

The cumulative latency of a network is affected in many ways. The first, and most obvious, is the LAN's access method. LAN's that feature a contention-based access method (for example, all "flavors" of Ethernet) are likely to have a higher, and less consistent, latency due to the vicissitudes of competing for empty packets. LANs that use a deterministic access method (for example, Token Ring and FDDI) have a lower latency, but remain inconsistent and unpredictable.

A good way to improve the latency of both contention and token based networks is to implement them using switching hubs. Switches are very fast forwarding devices that function completely at Layers 1 and 2 of the OSI reference model. A LAN based on switching hubs enjoys a lower innate latency than one based on repeating hubs simply because the switches forward packets faster than conventional repeating hubs.

2.2.1.2 Routing

Another aspect of a network's cumulative latency is whether or not routing is used. Routers, an integral part of virtually all WANs, operate at layer 3 of the OSI Model and require software-driven table lookups to forward packets. This means that they cannot forward packets as quickly as a switch. Thus, they directly increase the cumulative latency of a network.

2.2.1.3 Frame and Packet Structures

Latency is also directly affected by a network's frame structure. Many of today's more common and mature network protocols also use flexible frame and packet sizes. Flexible data fields excel at transporting traditional data types by minimizing the packet-to-payload ratio. Unfortunately, this has the exact opposite impact on time-sensitive applications. Forcing such applications to intermingle in the bit stream with indeterminate packet sizes adversely affects time-sensitive applications by introducing inconsistency to the network's cumulative latency.

2.2.1.4 Packet Discrimination

Given that conventional networks are designed explicitly to support the transport of data, the time-sensitive packets of multimedia applications require non-standard handling. Networks must be able to

- Identify packets that require special handling.
- Be capable of accommodating those special requirements.

A mechanism that can provide networks with the ability to discriminate between packets, based on their performance requirements, is known as Quality of Service (QoS). QoS has two distinct facets: network and application. Valid application QoS parameters include image size and quality, frame rate (if the application is video), start-up delays, reliability, and so on. The network however has a very different set of QoS parameters. These include: bandwidth, loss rate, delay. Users are not allowed to specify network QoS parameters. A QoS-capable protocol, such as RSVP (Resource Reservation Protocol), provides the translation between application and network parameters.

Obviously, the relationship between these sets of parameters is very complex. In theory, QoS will enable the different application types to receive the special handling that they require. Applications that require guaranteed integrity of packet contents can receive that, while others that need low delay and/or response times can tell the network about requirement through QoS tags too.

2.2.2 BANDWIDTH

Timely delivery of multimedia data packets is not the only challenge in multimedia communication. The other challenge to be overcome is transporting the

volume of data generated by multimedia applications. Applications that are not time-sensitive like high-density graphics or non-streaming audio and/or video transmissions can be transported better by conventional networks. Their performance requirements are guaranteed integrity of packet contents and re-sequencing upon arrival. However they can be extremely bandwidth intensive.

Ways have been devised to improve on bandwidth utilization and some of the are discussed briefly:

2.2.2.1 Bandwidth Conservation:

One of the best ways to increase the amount of usable is to conserve the consumption of existing bandwidth. There are numerous techniques for conserving bandwidth, most of which are automatically built into multimedia application software. Compression is an invaluable tool for conserving bandwidth.

2.2.2.2 Increasing available bandwidth

The other way to increase the bandwidth available for multimedia applications is to increase the speed of the network. Fast Ethernet, gigabit Ethernet, ATM, FDDI, Fiber Channel, and so on can also be used to substantially increase the available bandwidth on a LAN. Segmentation increases usable bandwidth without increasing a network's velocity by creating multiple collision domains or logical rings within a common broadcast domain. This type of upgrade allows the retention of the station wiring and network interface cards.

2.2.2.3 High Bandwidth with low Latency

Certain applications, like video conferencing, simultaneously require low latency and high levels of throughput to operate successfully. Unless a network is intentionally and severely over-engineered, adding such applications to an existing network may tax the network's capabilities and reduce overall performance for all the applications that rely upon the network for transport. Given the various contributors to a network's cumulative latency, the surest way to provide high bandwidth and low latency is to select network technologies that are specifically designed for this tandem purpose. Some specific examples include ATM and isochronous Ethernet.

2.2.2.4 Bandwidth Reservation

One way to maximize bandwidth utilization is to use protocols that reserve bandwidth. RSVP (Resource Reservation Protocol), for example is an emerging network protocol that can reserve the amount of bandwidth that will be needed by establishing a temporary but dedicated virtual circuit between the source and destination machines. The obvious danger inherent in bandwidth reservation schemes is that once bandwidth is reserved by an application, it is unavailable to other machines and their application.

2.3 AUDIO COMMUNICATION

Audio communication can take three distinct forms, each with slightly different set of network performance requirements. Specific categories include: computer-based telephony, audio conferencing and audio transmission.

2.3.1 COMPUTER-BASED TELEPHONY

Computer-based telephony uses PCs and LANs/WANs to integrate voice telephony into a data network. The client PC buffers inbound transmission and plays them using its sound card and speaker. This buffering can “smooth out” the sound quality of the transmission somewhat, despite its having transverse contention-based LANs using error-correcting protocols.

Audio communication is not bandwidth intensive. Audio can be delivered over dialup facilities as low as 14.4Kbps using Point-to-Point Protocol (PPP). This form of communication however is extremely susceptible to corruption from packets delivered late or out of sequence. Any such packets are discarded because, by the time a successful retransmission can be made, the stream being played back will likely have progressed beyond the point at which that packet was needed. Thus, re-inserting it late creates a second disturbance that is readily detectable by the user.

Computer-based telephony suffers from two limitations. Transmissions are to date, half-duplex only. Half-duplex transmission means that only one party can “talk” at a time, much like “push-to-talk” walkie-talkies. Telephones are full-duplex mechanisms. Both parties on a telephone call can talk and listen simultaneously.

The second limitation is that computer-based telephony capable of providing sound quality on par with an AM radio. The combination of half-duplex transmission and relatively low sound quality renders this technology more of a curiosity or techno-toy than a business tool.

2.3.2 COMPUTER-BASED AUDIO CONFERENCING

Audio conferencing differs from computer-based telephony only in that it is used in other than point-to-point sessions. Conferences tend to be multipoint-to-multipoint in the case of a collaborative conference of peers, or point-to-multipoint for broadcast of major events.

Given the half-duplex nature of this technology set, point-to-multipoint unidirectional broadcast may be the use of this technology. The network must have some mechanism for this form of multicasting. Multicasting is the transmission of a single stream of packet data with an address that is recognized by more than one workstation. This is far more bandwidth efficient than transmitting multiple simultaneous streams, each destined for a single, specific end-point. End-point that belong to a multicast group listens for both their unique Internet address and address of their group.

2.3.3 STREAMING AUDIO

Streaming audio transmission are unidirectional transmissions of a stream of audio data. It uses a host that either records audio in real-time or uses prerecorded audio media. In either case packets stream out onto the network as soon as they are generated. Recipients listen to them as they arrive, generally without buffering them. Dropped or damaged packets are usually left out of the playback session.

Streaming audio, like most audio only multimedia application, is relatively easy to support in a LAN/WAN environment. It is low bandwidth and benefits from but does not require low network latency. Streaming audio can be used to distribute, on demand, either a feed from a live speech or copies of recorded speeches, question and answer sessions or even the latest disk from your favorite group.

2.4 VIDEO COMMUNICATION

Video communication requires a fairly high powered computer and can also be extremely bandwidth intensive. It also benefits greatly from low latency network components.

Video communication can occur at surprisingly low levels of throughput given the right compromise of picture size, quality and refresh rate. The ideal rate of refresh is 30 frames per second. At this rate known as "full motion" the picture appears smooth and movement is smooth not jerky. Unfortunately, even using a small picture size, like 288 pixels by 352 pixels, the uncompressed stream is approximately 500Kbps. This represents a generous portion of a t-1's available bandwidth. It is also a sizeable portion of the useable bandwidth on most LANs.

Dropping the refresh rate to 15 frames per second and decreasing the number of colors recognized can dramatically reduce bandwidth consumption and reduce the size of the transmission stream.

Video communication includes video conferencing and streaming video transmissions. Although very similar, they have distinct functionality sets and consequently, different network performance requirements.

2.4.1 VIDEO CONFERENCING

Real-time, bi-directional transmission between two or more points are known as video conferencing. The accompanying audio can be handled "in-band" or "out-band". In-band audio transmissions bundle the audio signals with the video signals in the same bit stream. This requires the video conferencing system to have its own speaker and microphone or to interface with those already installed in the computer. Out-of-band

audio relieves software from having to capture and play back the audio signals. It also means the video conferencing system doesn't synchronize the audio and video. Rather the video system ignores the audio signals and requires conferences to establish a second communication link over conventional telephony.

2.5 SUMMARY ON MULTIMEDIA COMMUNICATION

It is inevitable that the currently separate voice, data, and video communication infrastructure will eventually integrate into a single broadband multimedia communication infrastructure with common user interface and vehicle. This integration is only just beginning and will take years to complete. It is already clear that LAN is capable of progressively growing into the role of multimedia communication network. In the interim it is used to support fledgling attempts at this degree of integration: today's multimedia application like the one this project is based on i.e. Software Intercom.

Some of these applications require levels of performance that are difficult to achieve in a conventional LAN/WAN environment. Today's LANs/WANs and their protocols are not well suited to transporting time sensitive data of many multimedia communication technologies. However the good news is that today's networks have the potential to evolve incrementally into a true multimedia communication infrastructure..

CHAPTER THREE

3.0 PROGRAM ANALYSIS AND DESIGN

The Software Intercom has a graphical user interface (GUI) much like the various internet phoning software found on the world wide web. The exception about this software is that it uses a network card to establish communication instead of a modem (as with most phoning software). Although there is no phone line involved in as with PABX, there definitely won't be a dialing tone. Instead the recording of a sound emulating the conventional phone ring tone is used. When a caller dials the number of a peer, the sound file is activated which lets the call recipient know he is being called, and also a pop-up dialog box appears on the screen interrupting whatever program is being run. Implementation of this project is achieved with the JAVA Programming.

3.1 WHY JAVA?

Java is a high level programming language that was developed solely with the Internet in mind. Most web programs used today are designed using the JAVA programming language. It has unique attributes embedded into its design features which make it the program of choice for most web application developers. It's embedded design feature include; Object Oriented Programming (OOP), Platform Independence, High Performance, Multi-Threading, and Dynamic linking. All these feature have made programming complex applications rather simple and straight forward. The design features would be explained in detail so as to aid an understanding into way JAVA is very important in the implementation of this project.

3.1.1 OBJECT ORIENTED PROGRAMMING (AN ANALOGY).

You can walk into a computer store and, with a little background and assemble an entire PC computer system from various components: a motherboard, a CPU chip, a video card, a hard disk, a keyboard, and so on. Ideally, when you finish assembling all the various self-contained units, you have a system in which all the units work together to create a larger system with which you can solve the problems you bought the computer for in the first place. Internally, each of those components may be vastly complicated and engineered by different companies with different methods of design. But you don't need to know how the component works, what every chip on the board does. As the assembler of the overall system, each component you use is a self contained unit, and all you are interested in is how the units interact with each other. Will this video card fit into the slots on the motherboard and will this monitor work with this video card? Will each particular component speak the right commands to the other components it interacts with so that each part of the computer is understood by every other part? Once you know what the interactions are between the components and can match the interactions, putting together the overall system is easy.

What does this have to do with programming? Everything. Object-oriented programming works in exactly this same way. Using object-oriented programming, your overall program is made up of lots of different self-contained components (objects), each of which has a specific role in the program and all of which can talk to each other in predefined ways.

3.1.2 PLATFORM INDEPENDENCE

Java was designed to not only be cross-platform in source form like C, but in compiled binary form. Since this is impossible across processor architecture JAVA is compiled to an intermediate form called byte-code. A java program never really executes native on the host machine, rather a special native program called the JAVA interpreter reads the byte code and executes the corresponding native machine instructions. Thus to port JAVA programs to a new platform all that is needed is to port the interpreter and some of the library routines. The byte-code is precisely defined and remain same on all platforms.

The second important part of making JAVA cross-platform is the elimination of undefined or architecture dependent constructs. Integers are always four bytes long and floating point variables follow the IEEE 754 standard for computer arithmetic exactly. One doesn't need to worry about rewriting a program to fit to the design and architecture of various computer systems and operating system. JAVA in essence can be said to be "write once and run anywhere".

3.1.3 HIGH PERFORMANCE

Java byte codes can be compiled on the fly to code that rivals C++ in speed using a "just in time compiler". Several companies are also working on native machine architecture compilers for JAVA. These will produce executable code that does not require a separate interpreter and that is indistinguishable in speed to C++.

While you will never get the same speed out a JAVA program that you might be able to, writing from C or FORTRAN, the results will be suitable for all but the most demanding applications.

3.1.4 MULTI-THREADING

Java is inherently multi-threaded. A single JAVA program can have many different threads executing independently and continuously. Three JAVA applets on the same page can run together with each getting equal time from the CPU with very little extra effort on the part of the programmer.

This makes JAVA very responsive to user input. It also helps to contribute to JAVA's robustness and provides a mechanism whereby the JAVA environment can ensure that a malicious applet doesn't steal the entire host's CPU cycles.

There is a downside to multi-threading, multi-threading is to JAVA what pointer arithmetic is to C, that is a source of hard to find bugs (an error in a computer program).

3.1.5 DYNAMIC LINKING

Java does not have a dynamic link phase. JAVA source code is divided in java files, roughly one per class in your program. The compiler compiles these into .class files containing byte code. The compiler searches the current directory and directories specified in the CLASSPATH environment variable to find other classes explicitly referenced by name in each source code file. If the file being compiled depends on the other non-compiled files, the compiler will try to find them and compile them as well. The compiler is quite smart and can handle circular dependencies as well as methods that are used before they are declared.

More importantly, classes that were unknown to a program when it was compiled can still be loaded into it at runtime. For example, a web browser can load applets of differing classes that it's never seen before without compilation.

3.2 PROGRAM DESIGN

This program is designed to ease communication over a network. The major challenge in the running of this program is capturing the analog sound or video images from the microphone or web cam, buffering it and transmitting it seamlessly to the specified destination in real-time. On the other side the recipient should be able to hear and reply in a two way communication (full duplex mode). The figure below shows the software a startup.

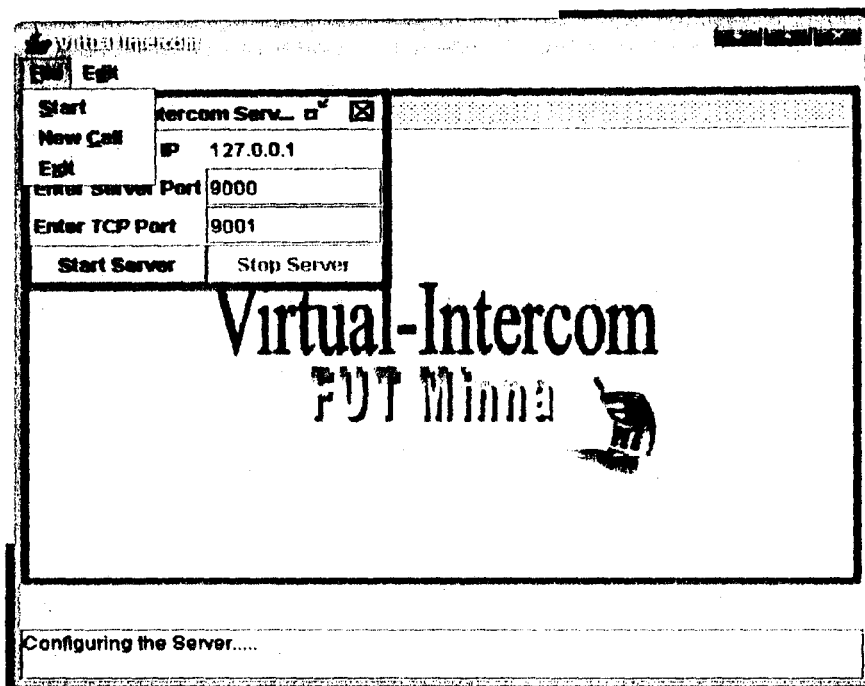


Figure 1.0 Showing the program a startup with its Server page.

When a caller wants to make a call to another on the network, he runs the program by clicking an icon on his desktop, then from the program's edit menu he chooses "configure server" option which configures his local server to listen for any incoming calls from other callers on the network. He then starts his Virtual intercom by choosing the "Start" option from the file menu. Finally the caller loads a window called "New Call" from the file menu.

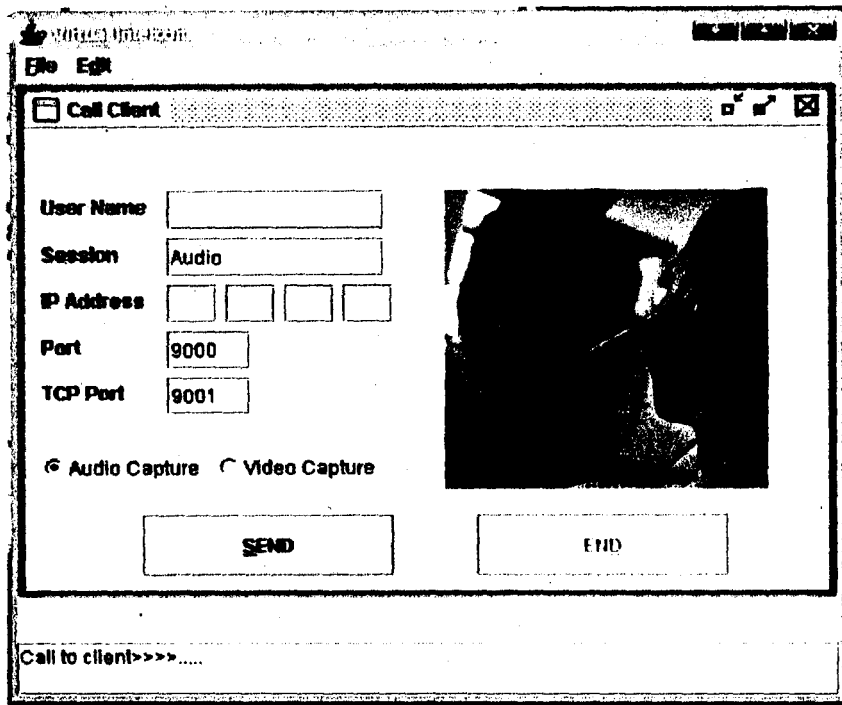


Figure 2. Showing the Call Client.

The window or internal frame contains several parameters; the caller needs to use to establish a connection with another on the network i.e. a text-field displaying the caller identity and other fields to input the recipient systems IP addresses, and also a choice of what type of media is to be captured, that is audio or video. He then makes the call by clicking on the send button and the specified recipient is prompted about an incoming call. As long as the call parameter are correct and the other person's software is running on his system then secure link between the two peers is established.

3.2.1 INSTALLATION

The program will have to be installed on each system the application will run on. The program automatically acquires the IP address of the system on which it is installed, so that its local server can be easily configured without much problem. The program uses a JAVA Runtime Environment (JRE 1.5.0) to execute and also JAVA Media Framework (JMF) which is an API (Application Package Interface) that does not come with the standard JAVA development toolkit, has to be installed then the program is ready to run. All other IP addresses of the other systems on the network would have to be obtained manually. They are important because these addresses are used in place of phone numbers to establish the connection between peers.

CHAPTER FOUR

4.0 SYSTEMS REQUIREMENT AND IMPLEMENTATION.

This chapter highlights the requirements needed for the smooth running of the program (both hardware and software). The Implementation of the program is also explained in this Chapter.

4.1 SOFTWARE REQUIREMENTS

The aim of this project is to enable the software to run on any computer platform (any operating system). To do so would require some software, application packages and API's (application package interface) to be installed basically on every computer system to enable the software intercom to run effectively.

4.1.1 OPERATING SYSTEM.

The software most definitely requires an operating system to run on the computer. An operating system manages the resources of the computer system i.e. hardware and software resources. The software runs on any of the major operating systems available in the market or the I.T. world i.e. **WINDOWS 98, 2000, NT, AND XP**; others would be **LINUX, UNIX, and MAC OS**.

4.1.2 RUNTIME ENVIROMENTS AND API'S

Since this program was written in **JAVA**, it needs a runtime environment to interface it with the operating system of choice. It uses a **JRE 1.5.0** (Java Runtime Environment version 1.5.0). Another very important Application Package Interface (API) needed to run this program is **JMF 2.0** (Java Media Frameworks 2.0). Its importance and purpose would be explained below.

Java Media Frameworks (JMF 2.0), is an application programming interface for in cooperating time based media into JAVA application and applets. This API supports media capture and addresses the needs of application developers who want additional control over media processing and rendering. It also provides a plug-in architecture that provides direct access to media data and enables JMF to be more customized and extended.

4.2 HARWARE REQUIREMENTS

A computer with full multimedia capabilities is the basic requirement for running the program. There should be installed a good full duplex sound card, with working speakers and a microphone. Optionally because the software is also capable of transmitting videos, a TV card could be installed too. This forms the interface needed for connecting a video camera to the computer. A webcam can be used instead of a video camera which requires a TV card, primarily because the webcam can be connected to the computers USB port with no need for extra expensive hardware. Only draw back to using a webcam is that the quality of pictures is greatly diminished.

4.2.1 A QUALITY LOCAL AREA NETWORK (LAN)

Today local area networking is a **shared access technology**. This means that all of the devices attached to the LAN share a single communication medium, usually a coaxial, twisted pair or fiber optic cable. Several computers are connected to a single cable that serves as the communications medium for all of them. The physical connection to the work is made by putting a network interface card (NIC) inside the computer and connecting it to the network cable. Once the physical connection is in place, it is up to the

network software to manage communication between stations on the network. This LAN should of quality speed.

4.3 PROGRAM IMPLEMENTATION

The system works with both point to point and point to multipoint configuration with full duplex transmission for voice and video conferencing. This means that the software works as both a **client and server software**. The purpose of the server is to listen in on any communication in the network that is addressed to the IP address of the computer it is resident on. It then relays the message to the client addressed which in this case is the computer it is running on.

When the program is launched the software is configured to set the machine to receive any incoming calls from another on the network. By default the server is initialized through TCP/IP socket-socket communication on a selected port (9001). When a call is being setup the transmitting station is required to provide user identification which is transmitted along with its IP address to the receiving end. This enables the receiving end to know what machine on the network is trying to establish a connection.

The transmitting system then sets up a socket communication with the receiving station. During this call setup phase, the transmitting and the receiving station agree on transmission parameters like the RTP session port, the audio quality signal (this particular implementation uses the base sampling quality on the system, specifically using the following audio format; LINEAR, 8000.0Hz, 16-bit, Mono, LittleEndian, signed) and the capture device used is Java Sound Audio Capture locator = java sound;//44100 which represents the system microphone. Other capturing system include the system dependent Direct Sound Capture (locator = dsound ://).

4.4 PROGRAM SEGMENTS

Basically, there are four modules or segments;

1. Capture the media data from the input device.
2. Encode the captured audio data.
3. Transmit the captured data.
4. Decoding and rendering the data stream at the destination.

4.4.1 CAPTURING MEDIA DATA

To capture media data:

Locate the capture device to be used by querying the CaptureDeviceManager.

- Get a CaptureDeviceInfo object for the device
- Get a MediaLocator from the CaptureDeviceInfo object and use it to create a DataSource.
- Create a Player or Processor using the DataSource.
- Start the player or processor to begin the capture process.

4.4.2 ENCODING CAPTURED AUDIO DATA

The processor can be configured to transcode captured media data before presenting, transmitting, or storing the data. To encode captured audio data in the IMA4 format before saving it to a file:

1. Get the MediaLocator for the capture device and construct a processor.
2. Call configure on the processor.
3. Once the processor is in the Configured state, call getTrackControls.
4. Call setFormat on each track until one that can be converted to IMA4.

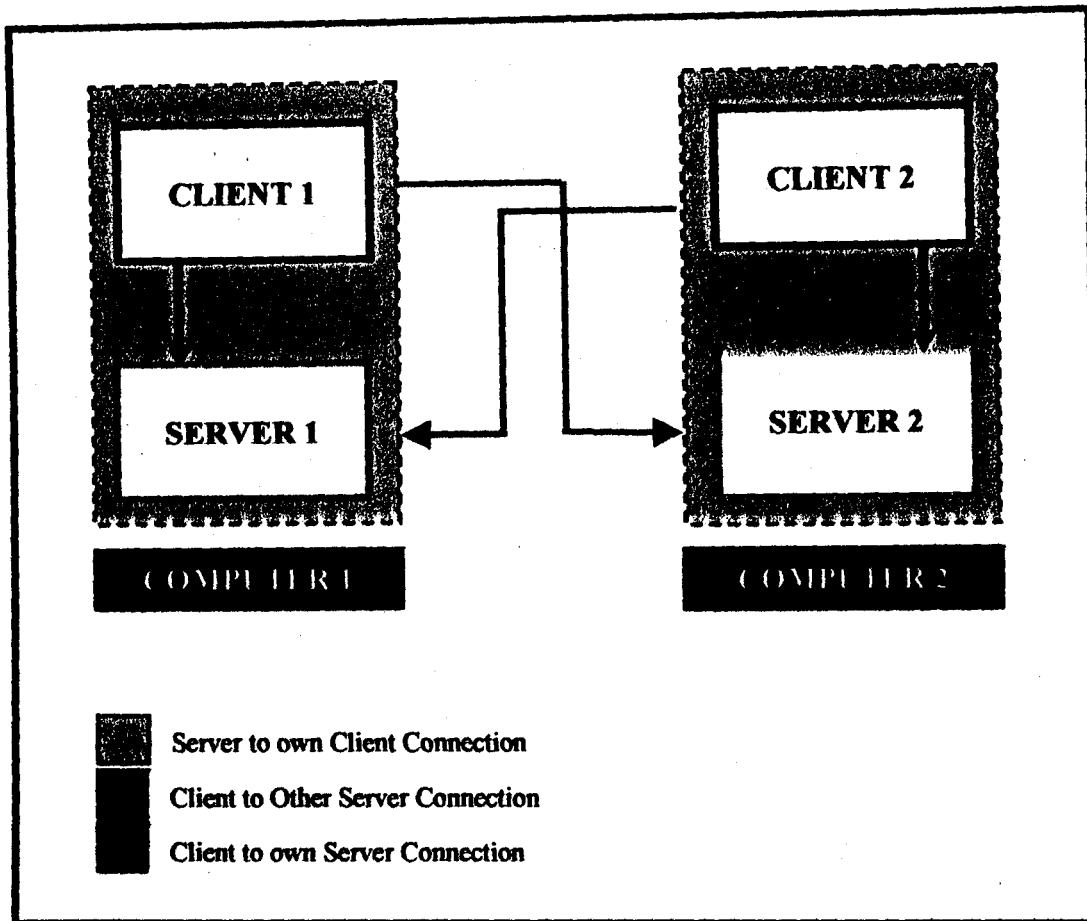


Figure 4.0 A Pictorial view of Peer-Peer communication of two Computers running the software.

After the socket communication has agreed on the parameters to be used for the communication, an RTP session is started based on the parameters to be used for the communication, an RTP session is started based on the parameters and real-time audio could be captured and transmitted between the two systems in full duplex point to point configuration. The system could be extended to handle full session, full duplex point to point configuration.

5. Realize the processor and use its output DataSource to construct a DataSink to write the data to a file.

4.4.3 TRANSMITTING THE MEDIA STREAM

The basic process for transmitting RTP data with the session manager is:

1. Create a JMF processor and set each track format to an RTP-specific format.
2. Retrieve the output DataSource from the processor.
3. Call createSendStream on a previously created and initialized SessionManager, passing in the DataSource and a stream index. The session manager creates a SendStream for the specified SourceStream.
4. Start the session manager by calling SessionManager start Session.
5. Control the transmission through the SendStream methods. A Send Stream Listener can be registered to listen to events on the Send Stream.

4.4.4 DECODING AND RENDING THE MEDIA STREAM

1. Set up the RTP session

- a. Create a SessionManager. For example, construct an instance of `com.Sun.media.rtp.RTPSessionMgr`. (RTPSessionMgr is an implementation of SessionManager provided with the JMF reference implementation.)
- b. Call `RTPSessionMgr.addAVStreamListener` to register as a listener
- c. Initialize the RTP session by calling the `RTPSessionMgrinitSession`.
- d. Start the RTP session by calling the `RTPSessionMgrstartSession`.

2. In your **AVStreamListener** update method, watch for **NewRecieveStreamEvent**, which indicates that a new data stream has been detected.

3. When a **NewRecieveStreamEvent** is detected retrieve and **RecieveStream** from the **NewRecieveStreamEvent** by calling **getRecieveStream**.

4. Receive the RTP **DataSource** from the **ReceiveStream** by calling **getDataSource**.

This is a **PushBufferDataSource** with an RTP-specific Format. For example, the encoding for a DVI audio player will be **DVI_RTP**.

5. Pass the Data Source to **manager.createplayer** to construct a player. For the player to be successfully constructed, the necessary plug-in for decoding and depacketizing the RTP-formatted data must be available. The basic files used are:

1. **AVRequestListener.java**

2. **AVStreamListener.java**

3. **AVStreamTransmitter.java**

4. **Vlserver.java**

5. **Vintercom.class**

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 PROJECT CONCLUSION

In conclusion, the implementation of the project was successful. Companies with good network installation can enhance the functionality of their network by using this software intercom. This will bypass the need for expensive Private Branch Exchange (PABX). The installation is also easy. The protocols involved in the program do not reserve bandwidth so this will not slow the network down. More so, one example of how today's open standard component can be assembled to support a multimedia application and can be used with audio conferencing over networks. The trend toward telecommuting and virtual office has created both near-ideal condition and a legitimate business need for audio and video conferencing.

The architecture of this software has been designed and tested to provide network users with cost effective, easy, reliable multimedia communication over a network.

5.2 RECOMMENDATIONS

The program is designed strictly for systems on a network; further study could be done to interface the program with a Public Switch Telephone Network (PSTN). In this mode a regular telephone call is received by the secretary and redirected using the software intercom to the target recipient of the call. This will be of immense use to companies and industrial firms.

Another recommendation is the installation design, work stations could be made to install from the server and used ID numbers so for every fresh installation the existing number are automatically loaded into the address book resident on the work station. Furthermore, the issue of security could also be integrated into the program, where user names and passwords will have to be given to access the program; this will eliminate impersonation over the network.

Voice messaging could also be incorporated, so if any work station isn't logged on, whenever the work station gets logged on the message could be relayed in a store and forward method (just like voice mail).

REFERENCES

1. **Elliote Rusty Harold, Java Network Programming, 3rd Edition, Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. (<http://safari.oreilly.com>)**
2. **Deitel and Deitel, Java How to Program, 4th Edition, Published by Prentice Hall, 2002.**
3. **Debashish Mitra, Network Convergence and Voice over IP, TATA Consultancy Services, 2001**
4. **Quitum Technologies Inc. Risk and Rewards. 2000 Strategies for Migrating Corporate Voice Traffic to data Network, Quitum Technologies Inc. 14 Christopher way Eatontown, NJ 07724. (www.quitum.com)**
5. **www.java.sun.com**
6. **www.javaworld.com/index.html**
7. **www.ibiblio.org/javafaq/**

APPENDIXES

Ten Files are involved in the writing of this program; all forming separate modules with specific functions, taking advantage of java's object oriented programming properties. For the purpose of documentation I have excluded all files that were used to create the GUI (graphical user interface) and included only those that form the core of the program itself. Below are a list of all the files used in the program.

- AVRequestListener.java
- AVStreamListener.java
- AVStreamTransmitter.java
- AVServer.java
- RemoveUserEntry.java
- callPanel.java
- startPanel.java
- execPanel.java
- SoundPlayer.java
- Vintercom.java

1 AVRequestListener.Java

```
import java.io.*;
import java.util.*;
import java.net.*;
import javax.media.*;
import javax.media.rtp.*;
import javax.swing.*;

class AVRequestListener extends Thread
{
    JTextArea B;
```

```

ServerSocket serverSock = null;
Socket userSock = null;
String sessionID = null, userName = null,
userIP = null;
int userPort;
StringTokenizer tokens = null;
DataOutputStream output = null;
DataInputStream input = null;
PrintStream print = null;
String data = null;
AVRequestListener(int tcpPort, JTextArea A)
{
    try
    {
        B=A;
        serverSock = new ServerSocket(tcpPort);
    } catch(Exception e) {}
    start();
}
public void run()
{
    while(true)
    {
        try
        {
            userSock = serverSock.accept();
            input = new
            DataInputStream(userSock.getInputStream());
            data = input.readLine();
            tokens = new StringTokenizer(data,"|");
            sessionID = tokens.nextToken();
            userName = tokens.nextToken();
            userIP = tokens.nextToken();
            userPort =
Integer.parseInt(tokens.nextToken());
            System.out.println("New client joined : Info
: "+sessionID+ " "+userName+ " "+userIP+ " "+userPort);
            JOptionPane.showMessageDialog(null, "New
client joined : Info : "+sessionID+ " "+userName+
"+userIP+ " "+userPort, JOptionPane.PLAIN_MESSAGE);
            B.setText("New client joined : Info :
"+sessionID+ " "+userName+ " "+userIP+ " "+userPort);

            try
            {

```

```

        RemoveUserEntry removeUserEntry = new
RemoveUserEntry(sessionID, userName);
    }
    catch(Exception ex) {}
    UserInfo userInfo = new UserInfo(sessionID,
userName, userIP, userPort, userSock);
    if (!VIservers.userInfoTable.isEmpty() &&
VIservers.userInfoTable.containsKey(sessionID))
    {
        ((Vector)VIservers.userInfoTable.get
(sessionID)).addElement(userInfo);
    }
    else
    {
        VIservers.userInfoTable.put(sessionID, new
Vector());

        ((Vector)VIservers.userInfoTable.get(sessionID)).addElement(
userInfo);
        /*check for the audio-video stream*/
        if (!VIservers.managerMapTable.isEmpty() &&
VIservers.managerMapTable.containsKey (sessionID))
        {
            output = new DataOutputStream
(userSock.getOutputStream());
            print = new PrintStream(output);
            print.println("Y");
            AVStreamTransmitter transmitter = new
AVStreamTransmitter(sessionID,
            userName, userIP, userPort);
        }
    } catch(Exception e) {}
}
}
}

```

```

class UserInfo
{
    String sessionID = null, userName = null, userIP = null;
    int userPort;
    Socket sock;
    UserInfo(String sessionID, String userName, String
userIP, int userPort, Socket sock)
    {
        this.sessionID = sessionID;
        this.userName = userName;
    }
}

```

```

        this.userIP = userIP;
        this.userPort = userPort;
        this.sock = sock;
    }
}

class RTPManagerInfo
{
    String sessionID = null, userName = null;
    RTPManager rtpMgrs;
    Processor proc;
    RTPManagerInfo(String sessionID, String userName,
RTPManager rtpMgrs, Processor proc)
    {
        this.sessionID = sessionID;
        this.userName = userName;
        this.rtpMgrs = rtpMgrs;
        this.proc = proc;
    }
}

```

2. AVStreamListener.Java

```

import java.io.*;
import java.net.*;
import java.util.*;
import javax.media.*;
import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;
import javax.media.protocol.*;
import javax.media.format.*;
import javax.media.control.*;
import javax.swing.*;

public class AVStreamListener implements
ReceiveStreamListener, SessionListener
{
    String sessionID = null, userName = null;
    RTPManager mgrs = null, rtpMgrs = null;
    DataSource dataOutput = null;
    Processor processor = null;
    boolean dataReceived = false;
    Object dataSync = new Object();
    boolean waitFlag = false;
    String address = null;
    int port;
}

```

```

JTextArea C;

public AVStreamListener(String address, int port,
JTextArea A)
{
    C=A;
    this.address = address;
    this.port = port;
    initialize();
}
/* This method is used for initializing the
sessions and add the session listener and
receivestream listener to the RTPManager class,
set the buffer length and detect the new stream &
participant detected whenever event generated
*/
boolean initialize()
{
    try
    {
        /*create the new instance of RTPManager*/
        mgrs = (RTPManager) RTPManager.newInstance();
        /*add the sessionListener*/
        mgrs.addSessionListener(this);
        /*add the receiveStreamListener */
        mgrs.addReceiveStreamListener(this);
        SessionAddress local = new
SessionAddress(InetAddress.getLocalHost(), port);
        SessionAddress destination = new
SessionAddress(InetAddress.getByName(address), port);
        mgrs.initialize(local);
        mgrs.addTarget(destination);
        BufferControl bc = (BufferControl)
mgrs.getControl
("javax.media.control.BufferControl");
        if (bc != null)
            bc.setBufferLength(500);
    }
    catch(Exception e)
    {
        System.err.println("Cannot create the RTP Session:
" + e);
        C.setText("Cannot create the RTP Session: " + e);
        return false;
    }
    return true;
}

```



```

void close()
{
    /* close the RTP session.*/
    if (mgrs != null)
    {
        mgrs.removeTargets( "Closing session");
        mgrs.dispose();
        mgrs = null;
    }
}
/*Event generate when new participant joins */
public synchronized void update(SessionEvent evt)
{
    if (evt instanceof NewParticipantEvent)
    {
        Participant p =
((NewParticipantEvent)evt).getParticipant();
        if(processor != null && waitFlag == false)
        {
            waitFlag = true;
            saveSessionAndTransmit(p);
        }
        else if(waitFlag)
        {
            waitFlag = false;
        }
    }
}
/*Event generate when new stream is detected*/
public synchronized void update(
ReceiveStreamEvent evt)
{
    RTPManager mgr = (RTPManager)evt.getSource();
    Participant participant = evt.getParticipant();
    ReceiveStream stream = evt.getReceiveStream();
    if (evt instanceof RemotePayloadChangeEvent)
    {
        System.err.println(" - Received an RTP
PayloadChangeEvent.");
        System.err.println("Sorry, cannot handle payload
change.");
        C.setText(" - Received an RTP
PayloadChangeEvent.\n" +
                "Sorry, cannot handle payload
change.\n");
    }
    /*event generated for the new stream*/
}

```

```

else if (evt instanceof NewReceiveStreamEvent)
{
    try
    {
        /*get the stream*/
        stream =
((NewReceiveStreamEvent) evt).getReceiveStream();
        /*create the datasource from the stream*/
        DataSource ds = stream.getDataSource();
        /* Find out the formats.*/
        RTPControl ctl = (RTPControl)ds.getControl
("javax.media.rtp.RTPControl");
        System.err.println(" - Received new RTP stream:
" + ctl.getFormat());
        C.setText(" - Received new RTP stream: " +
ctl.getFormat());
        /*create a processor to handle the
input media locator*/
        try
        {
            processor =
javax.media.Manager.createProcessor(ds);
        } catch(Exception npe) {}
        waitForState(processor,
Processor.Configured);
        ContentDescriptor cd = new
ContentDescriptor(ContentDescriptor.RAW_RTP);
        processor.setContentDescriptor(cd);
        waitForState(processor, Controller.Realized);
        /* Get the output data source of the
processor*/
        dataOutput = processor.getDataOutput();
        PushBufferDataSource pbds =
(PushBufferDataSource) dataOutput;
        PushBufferStream pbss[] = pbds.getStreams();
        SendStream sendStream;
        try
        {
            rtpMgrs = RTPManager.newInstance();
            SessionAddress local = new SessionAddress();
            SessionAddress destination = new
SessionAddress(InetAddress.getLocalHost(), 9999);
            rtpMgrs.initialize(local);
            rtpMgrs.addTarget(destination);
            sendStream =
rtpMgrs.createSendStream(dataOutput, 0);
            sendStream.start();

```

```

processor.start();
if(participant != null && waitFlag==false)
{
    waitFlag = true;
    saveSessionAndTransmit(participant);
}
else if(waitFlag)
{
    waitFlag = false;
}
} catch(Exception e) {}
} catch(Exception e) {}
}
/*event generated when user disconnected*/
else if (evt instanceof ByeEvent)
{
    try
    {
        removeSessionAndClose(participant);
    }
    catch(Exception e) {}
}
}
/* This method is used for transmission of stream
from the server to the user */
void saveSessionAndTransmit(Participant p)
{
    try
    {
        String Info = p.getCNAME();
        StringTokenizer st = new StringTokenizer(Info,"|");
        sessionID = st.nextToken();
        userName = st.nextToken();
        System.out.println("User info... "+sessionID+"
"+userName);
        C.setText("User info... "+sessionID+" "+userName);
        RTPManagerInfo mgrInfo = new RTPManagerInfo
(sessionID,userName,rtpMgrs,processor);
        if (!VIservers.managerMapTable.isEmpty() &&
VIservers.managerMapTable.containsKey(sessionID))
        {
            /*already transmitting the stream in this
session as server only allows one stream per
session*/
            System.out.println("Sorry only one user is allowed to
transmit in this session : "+sessionID);

```

```

    C.setText("Sorry only one user is allowed to transmit
in this session : "+sessionID);
        if (VIsrver.userInfoTable.isEmpty() &&
VIsrver.userInfoTable.containsKey(sessionID))
        {
            /*sending the restriction info to the
transmitting end*/
            Vector userVec =
((Vector)VIsrver.userInfoTable.get(sessionID));
            for(int k = 0; k<userVec.size();k++)
            {
                UserInfo userInfo =
(UserInfo) ((Vector)VIsrver.userInfoTable.get(sessionID)).e
lementAt(k);
                    if((userInfo.userName).equals(userName))
                    {
                        DataOutputStream outputStream = new
DataOutputStream
                        ((userInfo.sock).getOutputStream());
                        PrintStream stream = new
PrintStream(outputStream);
                            stream.println("DENY");
                            System.out.println("Sending the
information back to the transmitting end");
                            C.setText("Sending the information back to
the transmitting end");
                                }
                            }
                    }
                return;
            }
        else
        {
            VIsrver.managerMapTable.put(sessionID, new Vector());
            ((Vector)VIsrver.managerMapTable.get(sessionID)).addE
lement(mgrInfo);
        }
        /*send the info (stream starts) to the user by
TCP*/
        if (!VIsrver.userInfoTable.isEmpty() &&
VIsrver.userInfoTable.containsKey(sessionID))
        {
            int socsize = ((Vector)VIsrver.userInfoTable.get
(sessionID)).size();
            DataOutputStream dos = null;
            PrintStream ps = null;
            UserInfo info = null;

```

```

for(int i = 0; i<socsize; i++)
{
    info =
(UserInfo) ((Vector) VIsServer.userInfoTable.get(sessionID)).e
lementAt(i);
    if((info.userName).equals(userName))
    {
        /*this is the same user who is
transmitting*/
        System.out.println("Sending the information
back to the transmitting end");
        C.setText("Sending the information back to the
transmitting end");
    }
    else
    {
        try
        {
            dos = new
DataOutputStream((info.sock).getOutputStream());
            ps = new PrintStream(dos);
            ps.println("Y");
        }
        catch(Exception ee)
        {
            (info.sock).close();
        }
        AVStreamTransmitter transmitter = new
AVStreamTransmitter(info.sessionID, info.userName,
info.userIP, info.userPort, C);
    }
}
}
}
catch(Exception me)
{
    System.out.println("Exception in the
saveSessionAndTransmit method : "+me);
    C.setText("Exception in the saveSessionAndTransmit
method : "+me);
}
}

void removeSessionAndClose(Participant participant)
{
    try
    {

```

```

        String byeInfo = participant.getCNAME();
        StringTokenizer st1 = new
StringTokenizer(byeInfo, "|");
        sessionID = st1.nextToken();
        userName = st1.nextToken();
        try
        {
            VIsERVER.userRecTable.remove
(sessionID+"_"+userName);
        } catch(Exception urt) {}
        int size =
((Vector)VIsERVER.managerMapTable.get(sessionID)).size();
        for(int i = 0; i<size; i++)
        {
            RTPManagerInfo mgrsInfo =
(RTPManagerInfo)((Vector)VIsERVER.managerMapTable.get(sessi
onID)).elementAt(i);
            if(((mgrsInfo.sessionID).equals(sessionID))
&&((mgrsInfo.userName).equals(userName)))
            {
                (mgrsInfo.rtpMgrs).removeTargets("Session
ended.");
                (mgrsInfo.rtpMgrs).dispose();
                Processor proc = (Processor)mgrsInfo.proc;
                if(proc != null)
                {
                    proc.stop();
                    proc.close();
                    proc = null;
                }

                ((Vector)VIsERVER.managerMapTable.get(sessionID)).removeEle
mentAt(i);
                i=size;
            }
        }

if((((Vector)VIsERVER.managerMapTable.get(sessionID))).size
() == 0)
{
    VIsERVER.managerMapTable.remove(sessionID);
}
/*send the info (stream stops) to the user by
TCP*/
if (!VIsERVER.userInfoTable.isEmpty() &&
VIsERVER.userInfoTable.containsKey(sessionID))
{

```

```

        int socsize =
((Vector)VIserver.userInfoTable.get(sessionID)).size();
        DataOutputStream dos = null;
        PrintStream ps = null;
        UserInfo info = null;
        for(int i = 0; i<socsize; i++)
        {
            info =
(UserInfo) ((Vector)VIserver.userInfoTable.get
(sessionID)).elementAt(i);
            dos = new DataOutputStream
((info.sock).getOutputStream());
            ps = new PrintStream(dos);
            try
            {
                ps.println("N");
            }
            catch(Exception ee)
            {
                (info.sock).close();
            }
        }
    } catch(Exception bee)
    {
        System.out.println("Stop Exception " +bee);
        C.setText("Stop Exception " +bee);
    }
}
/* These methods are used for handling processor's
state changes.*/
private Integer stateLock = new Integer(0);
private boolean failed = false;
Integer getStateLock()
{
    return stateLock;
}
void setFailed() {
    failed = true;
}
/*This method waits for the states of the
processor*/
private synchronized boolean waitForState(Processor p,
int state)
{
    p.addControllerListener(new StateListener());
    failed = false;
}

```

```

/*Call the required method on the processor*/
if (state == Processor.Configured)
{
    p.configure();
}
else if (state == Processor.Realized)
{
    p.realize();
}
/*Wait until an event is fired that confirms
the success of the method, or a failure
event */
while (p.getState() < state && !failed)
{
    synchronized (getStateLock())
    {
        try
        {
            getStateLock().wait();
        } catch (InterruptedException ie)
        {
            return false;
        }
    }
}

if (failed)
    return false;
else
    return true;
}
/* Inner class used to control the state of
the processor*/
class StateListener implements ControllerListener
{
    public void controllerUpdate(ControllerEvent ce)
    {
        if (ce instanceof ControllerClosedEvent)
            setFailed();
        /* All controller events, send a
notification to the waiting thread in
waitForState method.*/
        if (ce instanceof ControllerEvent)
        {
            synchronized (getStateLock())
            {
                getStateLock().notifyAll();
            }
        }
    }
}

```



```
    }  
  }  
}  
}  
}  
} // end of AVStreamListener
```

3. AVStreamTransmitter.Java

```
import java.io.*;  
import java.net.*;  
import java.util.*;  
import javax.media.*;  
import javax.media.rtp.*;  
import javax.swing.*;  
public class AVStreamTransmitter  
{  
    int port;  
    String address = null;  
    String userName = null, sessionID = null;  
    RTPManager rtpMgrs = null;  
    JTextArea D;  
    public AVStreamTransmitter(String sessionID, String  
userName, String address, int port, JTextArea C)  
    {  
        D=C;  
        this.sessionID = sessionID;  
        this.userName = userName;  
        this.address = address;  
        this.port = port;  
        addRxTarget();  
    }  
    String addRxTarget()  
    {  
        try  
        {  
            int size =  
((Vector)Viserver.managerMapTable.get(sessionID)).size();  
            for(int i = 0; i<size; i++)  
            {  
                RTPManagerInfo mgrsInfo = (RTPManagerInfo)  
((Vector)Viserver.managerMapTable.get  
(sessionID)).elementAt(i);  
                RTPManager rtpMgrs = mgrsInfo.rtpMgrs;  
                /*first check whether the Server is  
already transmitting to this particular  
user or not*/
```