Order Number 1351342

X-window based user interface and network communications for an image network

Patel, Saurabh Sureshbhai, M.S.

The University of Arizona, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

X-Window Based User Interface and Network Communications for an Image Network

by

Saurabh Sureshbhai Patel

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 2

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Saurabh S. Patel

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Ralph Martinez

Ralph Martinez
Associate Professor of
Electrical and Computer Engineering

12/11/92
Date

ACKNOWLEDGMENTS

Working on my thesis was a very interesting and satisfying experience that was made possible by the advice and support of a number of people. First of all, I would like to thank Dr. Ralph Martinez for his leadership, encouragement, and technical advice that made this work possible. I also wish to thank my thesis committee members, Dr. Max Liu and Dr. William Dallas.

Finally, I would like to thank my parents, friends and family for their moral support in bringing all my work together.

To my parents

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Recent advances in the field of high speed computer network have opened many new applications. One such application is the transfer of color image data between geographically distributed color image databases. ImageNet is a distributed color image database system with multiple database nodes and user workstation linked by a communications network. The work presented here is the design and implementation of user-interface and communication software for a workstation. The system was emulated on the Ethernet in Computer Engineering Research Laboratory. Dedicated servers running at a database node simulate database operations. Experimental comparison between RPC with UDP and RPC with TCP was made and the use of RPC/UDP for image-transfer is investigated in this document. The X-Window user interface is user friendly. It enables users to retrieve image lists and images from remote workstations. The use of *wscrawl* for the remote consultation enables more than two users to participate in remote consultation from different workstations.

CHAPTER 1

INTRODUCTION

1.1 Statement Of Problem

Recent developments in the field of high speed networking has opened many applications in the commercial and military sectors[5][6][7][8][9]. One such application is the color image transfer between geographically dispersed color image database[1][2][3]. Most previous approaches have considered the use of telephone lines and leased lines as the underlying communication system. But it resulted in limited bandwidth and poor response time. There is a need for image visualization in many fields. In order to provide this facility, integration of workstation, network, database and user-interface technology is required. This project addresses the same problem with distributed approach, with the assumption of existence of a high speed network, such as the 45 Mbps NSFNET backbone[4]. The color image database system is called ImageNet.

This project will have an application in teleradiology. The images are generated at different locations in hospital and are stored in different nodes. The radiologists are located at different places with their workstations. They need to retrieve the images, stored at different locations to their local workstation. The operation of remote

consultation is required among radiologists. The radiologists do not have knowledge of computer so user-interface should be as simple as possible. The X-windows user-interface of this project is user-friendly with on-line help. It enables users to retrieve the list of images as well as images from different remote nodes. Sophisticated remote consultation software, *wscrawl*, can be invoked from the main menu, so radiologists can carry out remote consultation.

1.2 Thesis Objective

The objectives of the thesis are multi-faceted. They are,

1. Evaluate use of Remote Procedure Call(RPC) with User Datagram Protocol(UDP) for image transfer and compare with RPC with Transmission Control Protocol(TCP) performance.

2. Design, implement and test X-window based user-interface which includes a remote consultation operation for radiology.

This project will provide a facility for retrieving image lists, images and viewing on the local workstation. The user-interface is easy to use and does not require any prior computer knowledge. The retrieved results and errors are displayed on separate windows. The remote consultation facility will enable the user to perform remote consultation with more than one remote workstation at a time.

1.3 Approach

The problem of transferring images between geographically dispersed databases is solved with a distributed approach. A distributed approach has several advantages[10][11]. It increases performance,reliability and scalability of the system. The system architecture becomes modular and resource sharing becomes possible. Thus, it increases the performance to cost ratio. ImageNet's distributed architecture will have multiple database nodes and user workstations linked by a communication network. Figure 1.1 shows the distributed architecture of ImageNet.

Figure 1.2 shows the protocol stack of ImageNet. The ImageNet will exist on national level and each region will have a database node and database nodes connected together by communication network.

Since we do not have access to regional nodes, the IamgeNet has been emulated using an Ethernet network, in the Computer Engineering Research Laboratory. It includes user-workstation and image database nodes. The network software design uses UDP/IP as the transport layer protocol. The system is designed in terms of a client-server model[27]. The dedicated servers are designed on the server side which simulate the operation. The servers are always waiting for a request from client, residing on user-workstation. Figure 1.3 shows a DBN in more detail.

To accomplish above mentioned objectives, I have designed the software in the Session ,Presentation and Application layer above TCP/IP protocol suite. Access to transport layer protocols, TCP or UDP is made through *socket* interface mechanism.

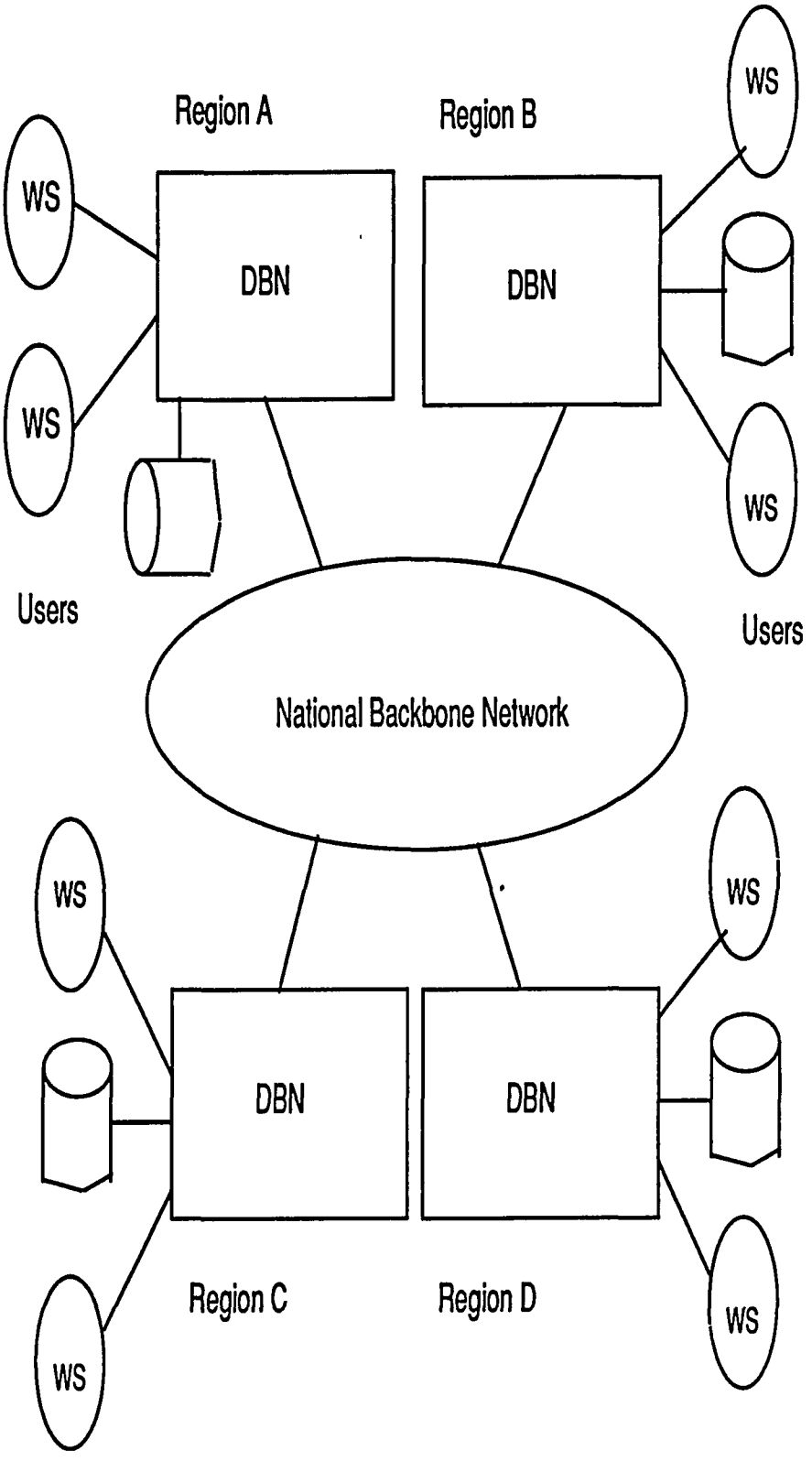

Figure 1.1: Overall ImageNet Architecture

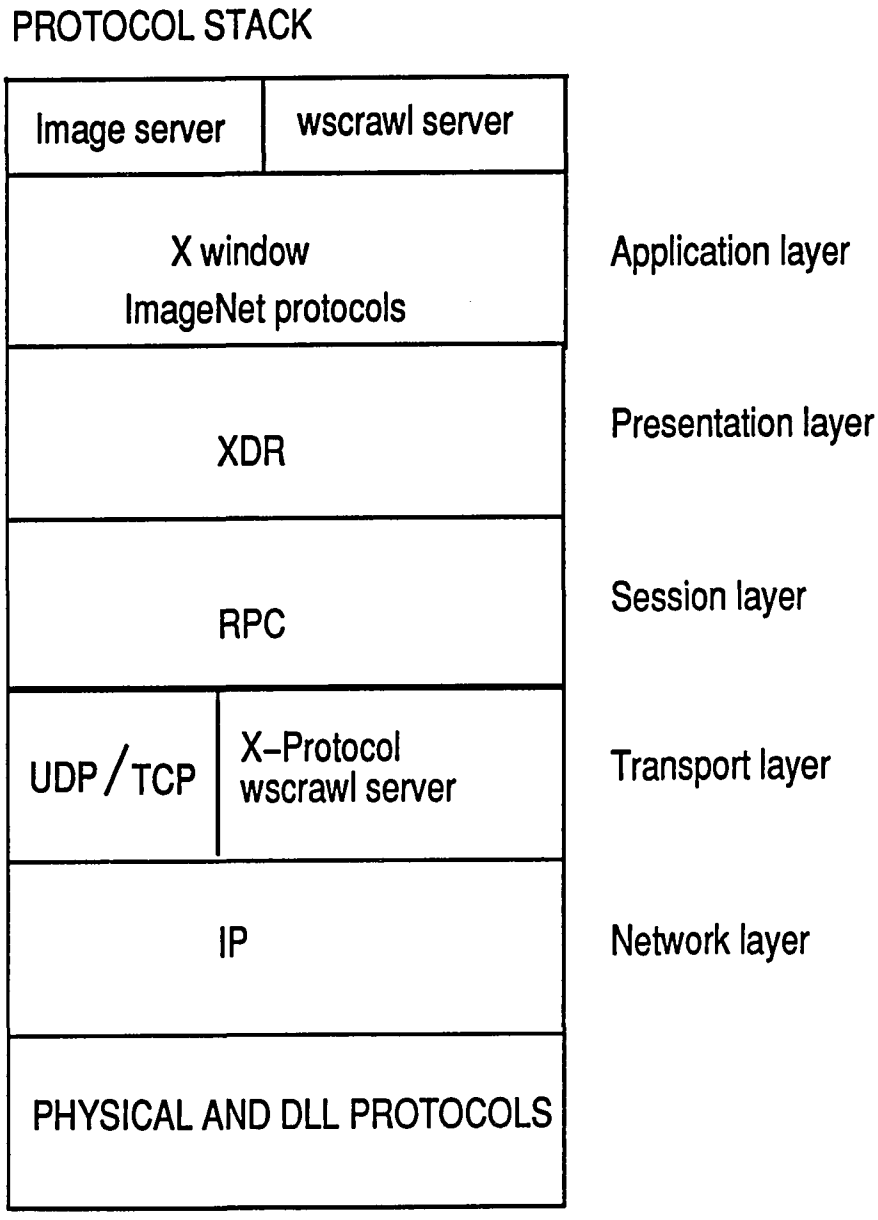

Figure 1.2: ImageNet Protocol Stack

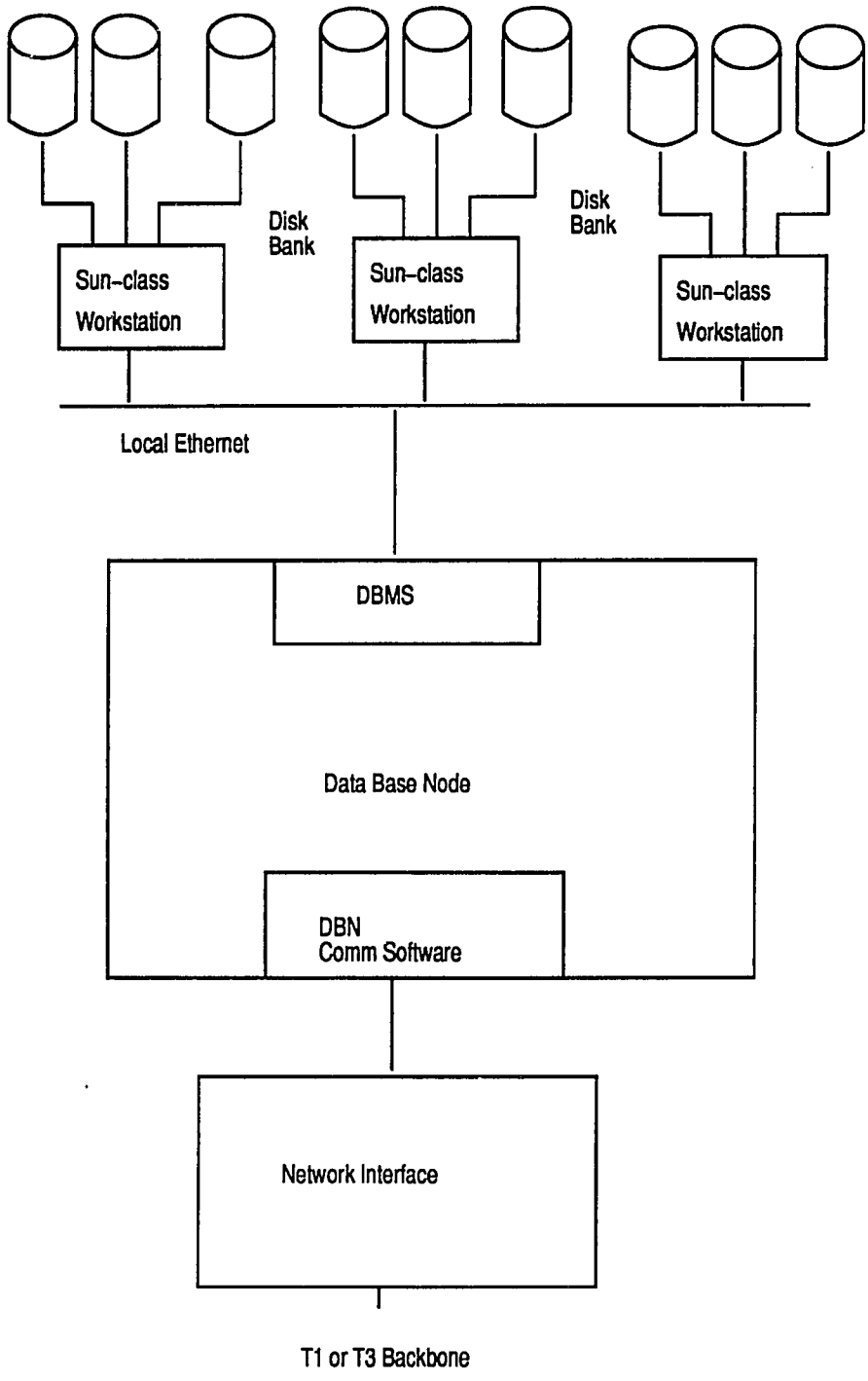

Figure 1.3: Details of a Regional DBN Implementation

In the session layer I designed RPC to retrieve the images or image lists from the remote site. I have developed RPC software above UDP, with lowest layer interface of Sun RPC, which enables the software designer to set all the network specific details according to the application requirement. I have developed RPC software with TCP with Sun RPC's highest layer interface because TCP is a reliable protocol and error handling is taken care by it. The application and RPC software designs take care of error handling in case of RPC/UDP implementation as UDP is not a reliable protocol. I have compared the performance of RPC/TCP with RPC/UDP for image transfer. To get experimental data, I have run experiments between Computer Science Department of University of Arizona and Wake Forest University in North Carolina across Internet and over the local Ethernet in Electrical and Computer Engineering Department . The experiments were conducted in week days and weekends and are grouped according to different times of the day. To provide the user friendly user-interface, I have developed the X-window software for user-interface, using X-toolkit and Athena widget set. For remote consultation operation, I have used the modules of *wscrawl* software, which is a public domain software, to interface with my main menu software design.

The rest of the thesis is organized as follows. Chapter 2 describes the overall ImageNet architecture and description of various user-interface capabilities. Chapter 3 illustrates the details of X-windows user-interface and networking software design.

In the beginning of the chapter some background of X-windows is described. Immediately after that, X-windows user-interface design is described. In network software design UNIX interface to UDP/IP and RPC mechanism is described in the beginning since they are the key points of the design. In the last subsection performance of RPC/TCP and RPC/UDP protocols are compared experimentally for image-transfer. Chapter 4 presents conclusion, system constraints and suggestion of some areas where future work can be done to improve the performance.

CHAPTER 2

FUNCTIONAL SPECIFICATIONS

This section contains scenarios, mapped into functional specifications and description of X-window widgets and RPC communication.

2.1 ImageNet Architecture

This section describes the ImageNet architecture and its major components. Figure 1.1 shows the overall architecture for ImageNet. ImageNet is distributed nationally over a large geographical area. ImageNet users are spread all over the country and are put into clusters or regions by the definition of a regional Database Nodes(DBNs). Figure 1.1 shows a small ImageNet network, consisting of four DBNs. The DBNs communicate with each other through a national communication network. ImageNet users in each region are connected point-to-point to a regional DBN using telephone and modem-communications. The user uses workstation for requesting image from the DBN. The DBN retrieves the image set from the database disks and transmits the images to the user workstation where they are stored in local disk. If the image set is not within regional DBN, then the DBN searches the other regional DBNs for that image set. Eventually, if the imageset is found the DBNs exchange

the image set and it is finally delivered to the user WS. Figure-1.3 shows a DBN in more detail. The ImageNet system components include the following:

Database Nodes(DBNs)

Regional database nodes which contain the region's images and textual information. Users retrieve images and text from these nodes. Figure-1.3 shows a DBN in more detail. The database nodes have a database management system(DBMS) and SQL software loaded in it. The previous ImageNet was implemented by team of two persons, one working on database side and other on workstation side. In this thesis, the database node operations are simulated by dedicated servers running on that machine. One server provides the list of available images by name and other server is the image server.

User Workstations(WSs)

The user workstations are UNIX based Sun class workstations under an X-windows environment with color graphics application and local disk storage. The Unix operating system provides many suitable facilities for developing X-windows user-interface and communications software. The WSs are connected to the communications network or to local networks.

Communication Network

The communication network consists of two parts, one to link the DBNs and the other provides direct communication between a user WS and a DBN node. The

DBN network uses optical fiber as its transmission medium and consists of a partially interconnected point-to-point backbone network. The topologies for distributed systems and other network considerations are described in [28][29].

2.1.1 User Scenarios

This section describes how the users retrieve, store and view images retrieved from the DBN.

1. **Image Acquisition and Storage**

   The color image database is generated at each DBN site. The color images will be accumulated at each DBN site, mostly in picture form. The pictures are digitized using a color frame grabber interface in the DBN computer system. The output of the color frame grabber interface will be a color formatted image, formatted according to the Graphics Interchange Format(GIF) standard. Each image contains textual information identifying the image and its characteristics. This textual information is used for the retrieval of the image using Query Language in the DBMS of the DBN. The DBN contains equipment to perform the digitizing of images on a production basis.

   The color images are compressed using Lempel-Ziv Welch(LZW) image compression algorithm provided by GIF and stored in the compression format. The DBMS is responsible for storing the images and text in the database disks. A

relationship between the textual information, the color images, and the storage on disks is kept by the DBMS.

Graphics Interchange Format(GIF)

Most of the images located in the database are stored in a format known as the Graphics Interchange Format(GIF)[23]. This format was created to provide a hardware independent protocol for the exchange of raster graphic data. The main entities in GIF are protocol blocks and sub-blocks which contain vital information for graphic reproductions. GIF does not provide error detection or recovery, thus a reliable transport layer protocol is required when transferring images across a network. GIF classifies blocks into three different groups:

(a) Control.

(b) Graphic-rendering.

(c) Special Purpose.

Figure 2.1 shows the general GIF file format. GIF uses a slight variation of the Lempel-Ziv Welch compression algorithm(LZW)[12][24]. The LZW algorithm uses string repetition in the data as the basis for its compression. It defines a rule for parsing strings from a stream of data into substrings, and a coding scheme to map these substrings into unique codewords. The LZW compression algorithm requires the initialization of a string table. We can initialize the string table by choosing the code size and number of values the character can take .

For example, if the code size is 10 and the number of values n=24, we can have 2**10 entries in the table and 24 different colors. Code #0 in the table must be initialized to character #0...to character #(n-1). Now, we define {p} as the current prefix, which initially is empty, and the current string denoted by the current prefix {p} and some character "c" as {p}c. The complete compression algorithm can be summarized as follows:

(a) Initialize the string table.

(b) Initialize the current prefix {p} to be empty.

(c) Read next character "c" from the character stream.

(d) Combine the current prefix and the next character "c"

(e) If {p}c is in the string table, make {p}c the current prefix and go back to step 3. Otherwise, add {p}c to the string table, output the code for {p} to the code stream (compressed output), make "c" the current prefix and go back to step 3.

2. **Image Retrieval**

Images are retrieved from a DBN upon the initiation of an image retrieval request at a User Workstation(WS). The WS user makes an image request through a menu set at the WS, which is then packaged into a Query Language request for the DBMS in the DBN. The WS communications software calls the nearest DBN site and automatically transfers the request to the DBMS.

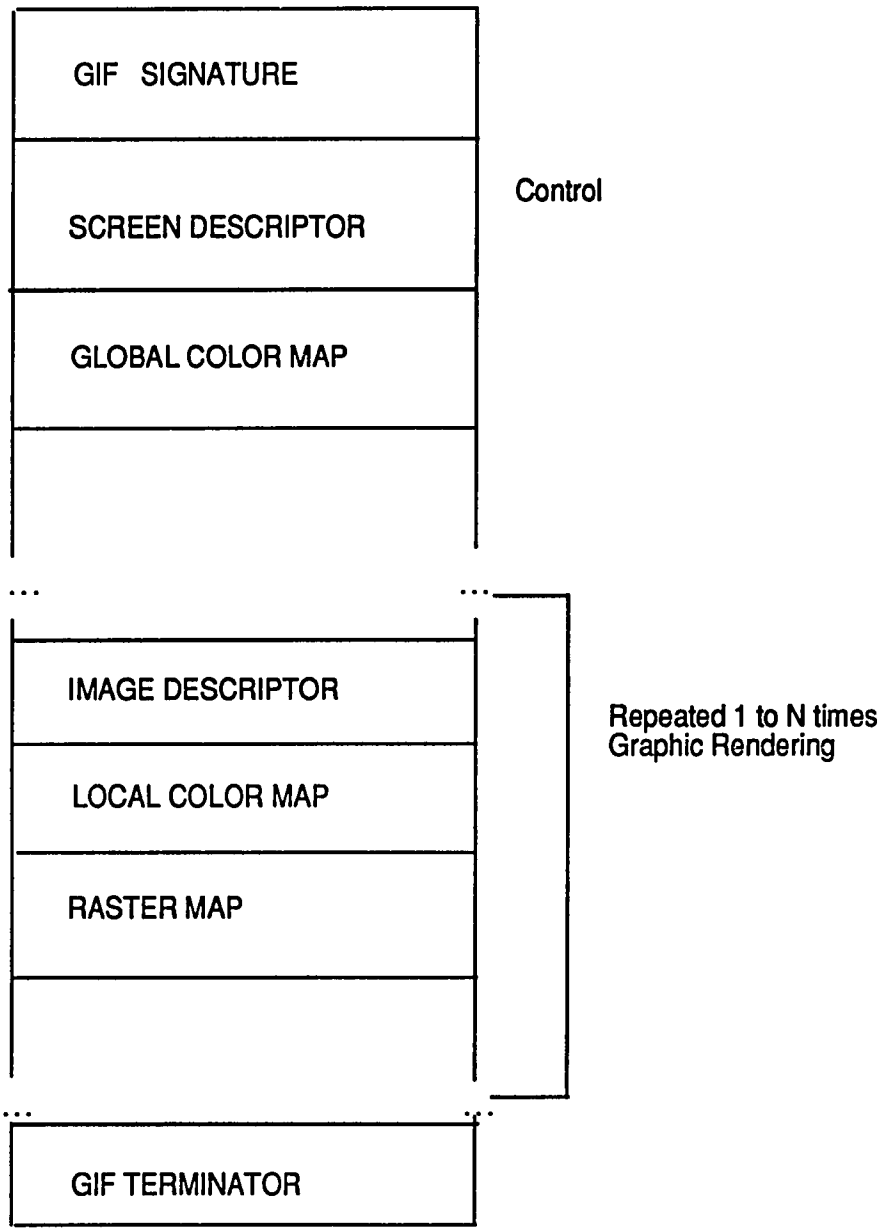

Figure 2.1: GIF File Format

The DBMS locates the image set, retrieves the images from the disks, and transfers them one by one to the WS site. Each image contains identifying textual information to be used and displayed at the WS.

3. **Image Viewing**

The users at each WS can view the images in uncompressed form once they are stored in the local disk at the WS. The images are uncompressed when they are called up by the WS user. The image set at a WS is viewed by using XV software for images in GIF formate. Figure 2.2 shows the image opened in XV. The image set can be kept at the WS for a period of time and later discarded. Images are never returned back to the DBN site.

4. **Remote Consultation**

Remote consultation operation is carried out between two or more users using remote consultation facility provided by the main menu. The user has to input the name of the node where it is required to open the image for remote consultation. The image can be opened at more than two places and the cursor movement can be seen simultaneously at all the places. This application will be very useful for the radiologist who are geographically dispersed and still want to take second opinion for diagnosis.

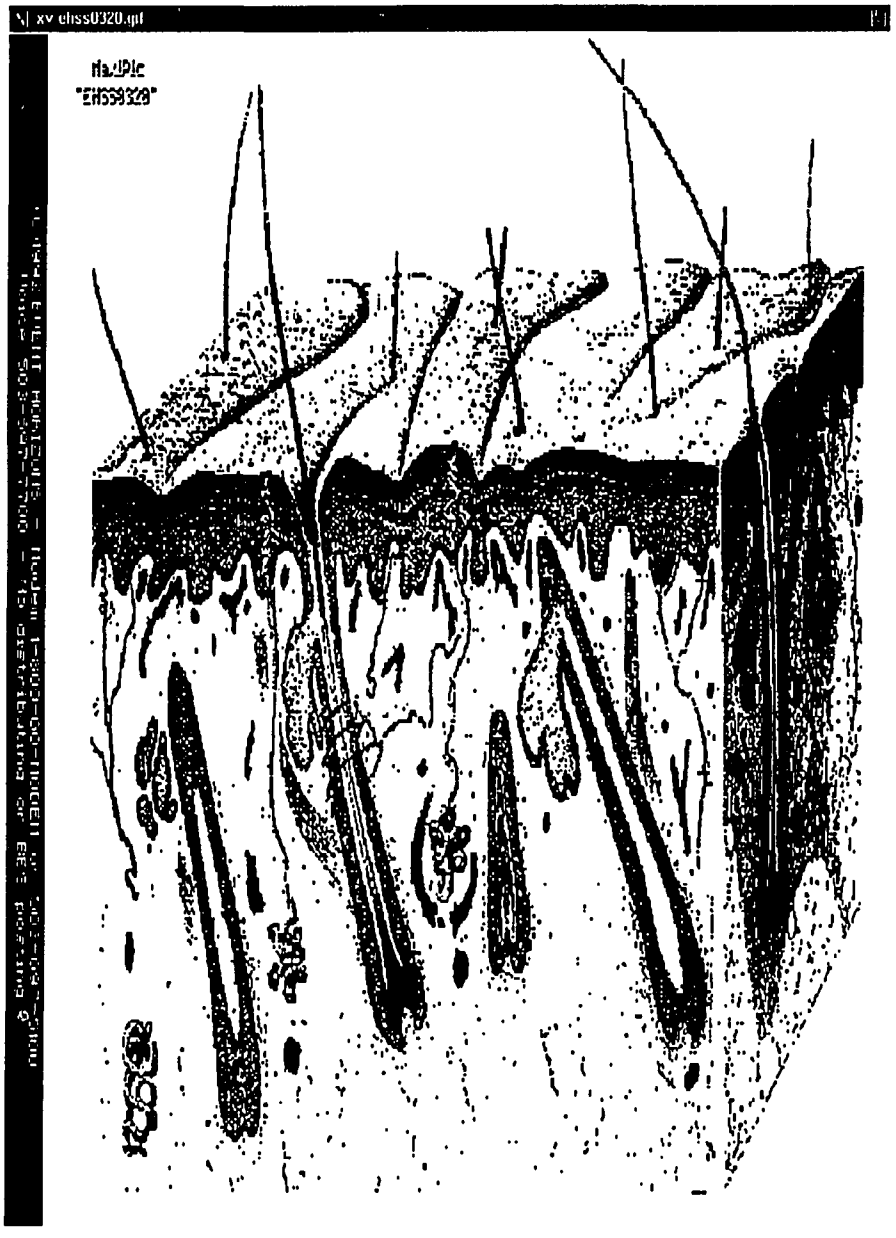

Figure 2.2: Imageviewing in XV

2.1.2 Performance Requirements

This section describes the preliminary performance requirements for ImageNet based on TCP/IP. The performance requirements are derived based on user scenarios and from results from simulations of Picture Archiving and Communications Systems(PACS)[3]. It is assumed ImageNet has the national network architecture, as shown in Figure 1.1.

1. DBN Connection Delay Time

   This time should not exceed 30 seconds from the start of the connection request at the WS. The communication software is developed on top of User Datagram Protocol where no connection set up takes place.

2. Image Query Response Time

   This time is the amount of time the user waits once the WS sends an image query and receives a response from the DBN, and should not exceed 30 seconds.

3. WS Image Retrieval Delay Time

   This time is the amount of time for transfer of the first image from the DBN to the WS, and depends on the image size and speed of the communications line. This time should be less than 30 seconds.

4. DBN Image Retrieval Delay Time

   This is the amount of time to retrieve an image from the disk system in the DBN and should be less than 30 seconds.

5. Image Viewing Time

   This is the time it takes to display a stored image on the WS monitor, including uncompression algorithm time, and should take less than 60 seconds.

6. DBN-to-DBN Image Set Transfer Time

   This is the amount of time required to transfer an image set between DBN nodes in the internet. An image set can be set up to 20 images. This time depends on the Internet communications system and protocol's overhead and speed.

2.2 User Workstations

This section describes the functions performed at a user workstation. The WS must have support for local image storage, X-windows environment for user-interface and network support for retrieving images from DBNs. Figure 2.3 shows the flow diagram of the workstation software. Whenever a request is made, first it checks whether it is for a remote DBN or not. If it is for DBN then request is sent by Remote Procedure Call. If any system or communication error occurs then it will be displayed in the separate dedicated error window. The received data will be displayed in the separate window. After completion of task it again waits for the user request. If the request is not for DBN, then it is executed locally.

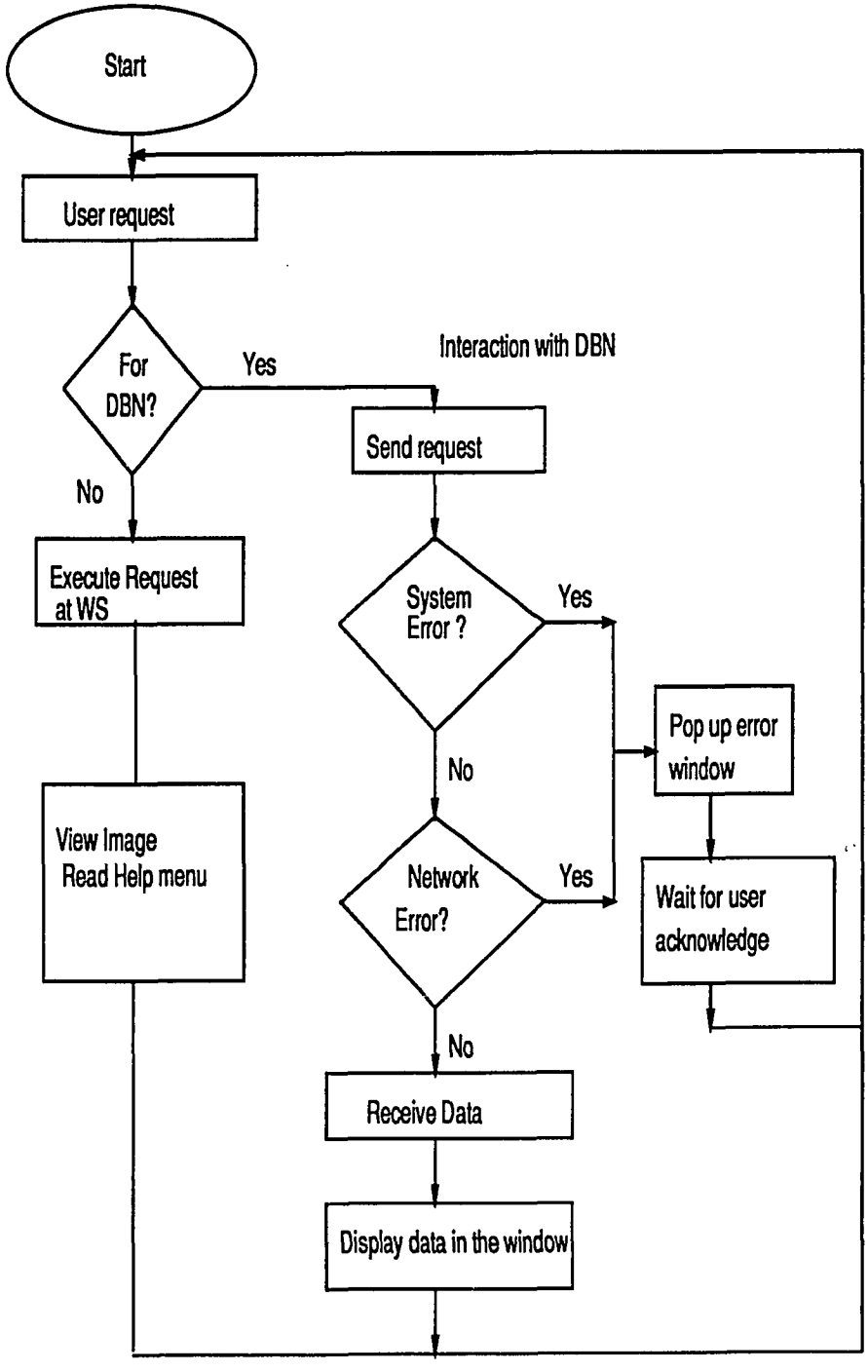

Figure 2.3: Workstation Software Flowchart

2.2.1 X-windows User-interface

The user interface for ImageNet has been designed taking into consideration that potential users have no prior knowledge of computers. So,cascaded pop-up menus are avoided. The toplevel menu consists of several buttons which are activated by leftmost mouse buttons. Figure 2.4 shows the main menu of the ImageNet user interface. When activated they pop-up menus or dialog boxes. All the retrieved data is displayed in the separate window which has the scrollbar to scroll the data. Any error message is displayed in the separate window.

The main menu command buttons are:

1. **FOLDER LIST**
2. **RETRIEVE SINGLE IMAGE**
3. **VIEW**
4. **REMOTE SESSION**
5. **HELP**
6. **QUIT**

These menu items are described in detail below.

FOLDER LIST: When the menu item is clicked , a dialog box is popped up. If the user does not enter anything and presses leftmost mouse button on dialog Done

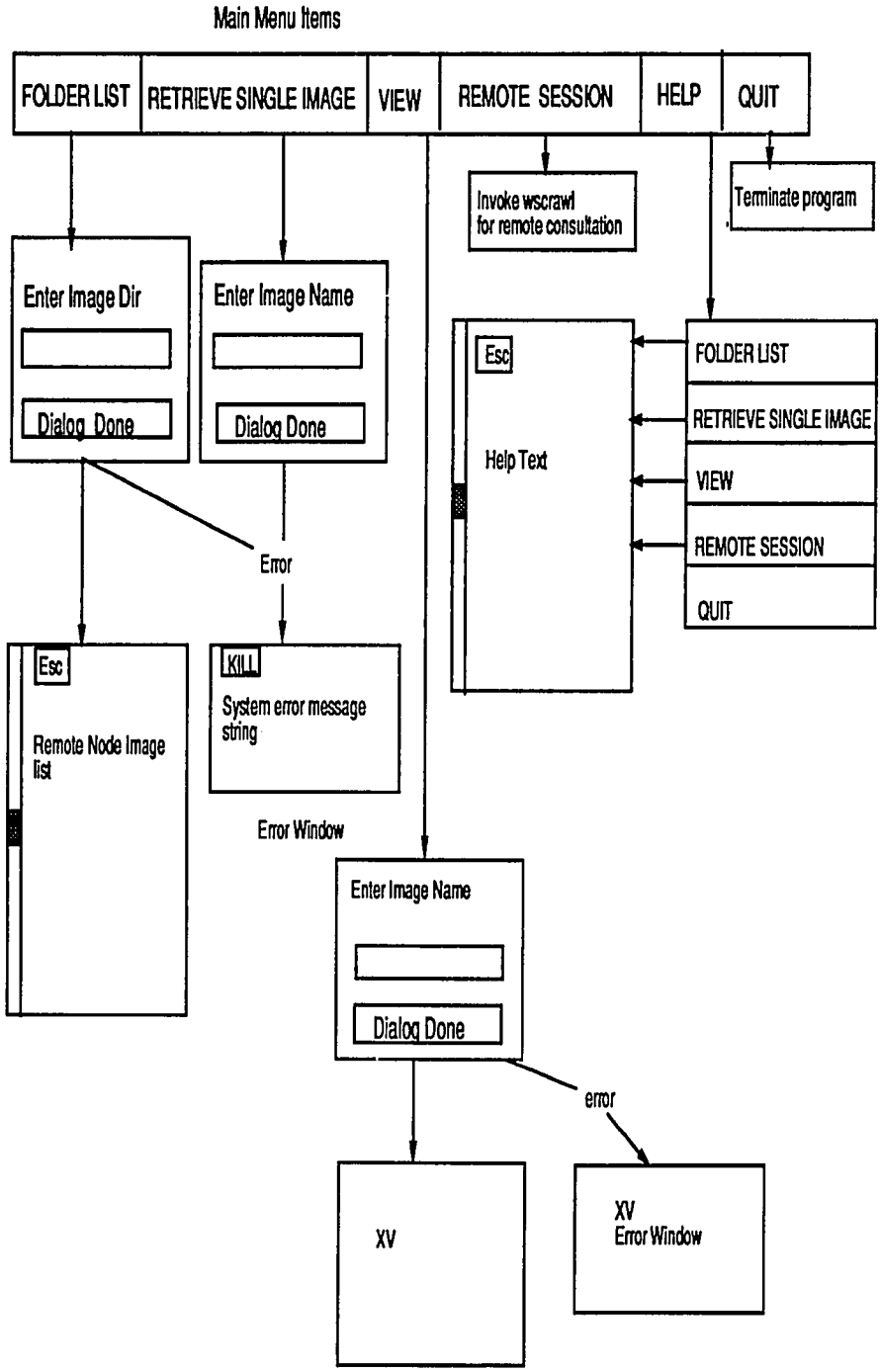

Figure 2.4: Toplevel Menu with Sub-menus

command button then the list of directories on a particular node will be retrieved. The retrieved list is displayed in the separate window. The window has a scrollbar, so the list can be scrolled up and down by the user. By pressing pointer on Esc command button, it pops down the window. If the user enters the image directory in the dialog box, the list of images will be retrieved and displayed in the separate window. In case of any system or communication error, a separate error window is popped up. It displays the system error message string. When the user clicks the pointer on KILL command button, it pops down the error window.

RETRIEVE SINGLE IMAGE: When the pointer is clicked on this command button, it will pop up a dialog box. The user enters the name of the image and clicks the pointer on Dialog Done button. The callback procedure invokes the Remote Procedure Call to retrieve the image. Any system or communication errors are displayed in the separate windows. When the pointer is clicked on KILL button, it will pop down the window.

VIEW: When the pointer is clicked on this command button, it pops up a dialog box. When the user enters the image name and clicks the pointer on Dialog Done button, the image is displayed by XV. XV is a public domain package used for GIF image display and manipulation. If the file is not in the GIF format or the permissions are denied, then error message will be displayed by XV in a separate window.

2.3 Remote Session With Wscrawl Program

When the pointer is clicked on REMOTE SESSION , it will invoke *wscrawl*, which is a sophisticated program for remote consultation. Figure 2.5 shows the *wscrawl* program menu. The modules of it are available from the Internet. It uses Xlib and based on X protocol. The main menu for *wscrawl* is shown in Figure 2.6. The image to be read in must be in local disk, so it must be retrieved using image retrieval facility. To read in image, *Read In Image* option must be selected in the pop-up menu of *File I/O* in the main menu. This will pop-up a dialog box. Enter the name of the image to be opened. Click the pointer on the window for positioning the image. The image will be displayed in such a way that the center will be the point where click is made with the pointer. For remote consultation, the image should be opened at the remote workstations. Figure 2.7 shows the workstation interaction using *wscrawl* for remote consulation. The user first retrieves the image from Data Base Nodes using Remote Procedure Calls. The communication software is developed using RPC/UDP protocol. Once the image is in local disk, *wscrawl* uses X-Protocol to send it to remote workstations for the remote consultation. The image window can be opened on any remote workstation in the Internet provided that it has X-windows support. To open the image on remote workstation, select *Control* on main menu and *Add Display* in the pop-up menu. It will pop up the dialog box, where complete address of the node followed by semicolon and zero must be entered. For example, if the image is to be

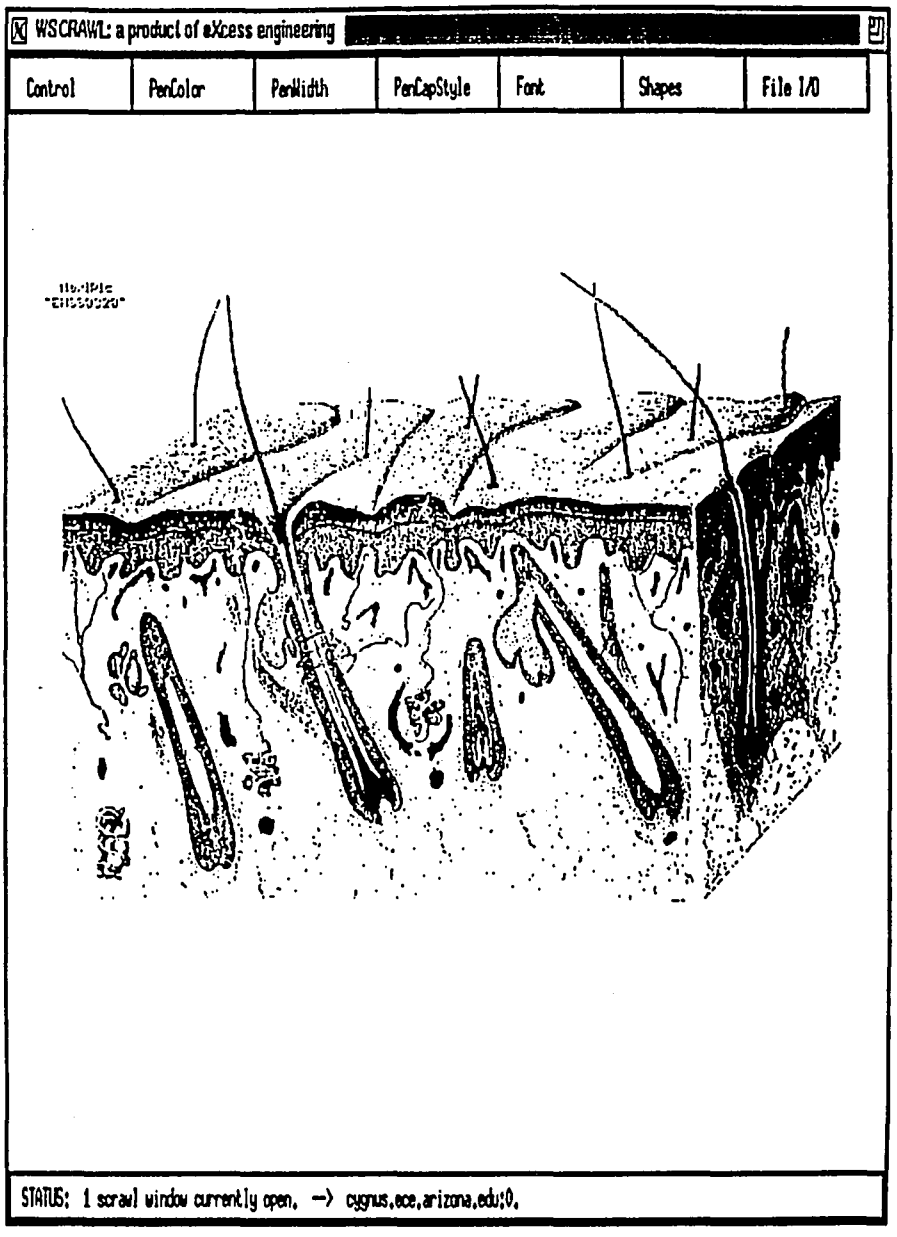

Figure 2.5: WSCRAWL Program

opened on lyra workstation in Sun Workstation lab in ECE, in the dialog box, the user should enter,

```
lyra.ece.arizona.edu:0
```

This will open the window on lyra workstation and the image is placed in the identical position. The images are transferred to the remote workstation by X protocol, which is a network protocol. There is no restriction on number of workstations on which image can be opened for remote consultation. Selecting *Rubber Pointer* option in control menu will tag node name with the pointer. This is very useful when more than two users are involved in remote consultation. If the name of the node is not displayed with the pointer then it is not possible to identify the user. For identifying certain portion of the image, the user can draw shapes by selecting *Draw Shapes* in pop up menu of *Control* in the main menu. The shapes can be selected from pop-up menu of *Shapes*. The user can select straight line,outline rectangular, filled rectangular, outline oval and filled oval from the pop-up menu of *Shapes* in the main panel. The user can select different pen colors by selecting the pen color from pop-up menu of *Pen Color* in the main menu. The pen width can be increased or decreased by selecting it from the pop up menu of *PenWidth* in the main menu. The pen cap style can also be changed by the options selected from *Font* in the main menu.

To load another image, the existing window should be cleared by selecting *Clear Window* option from *Control* in main menu. To close the window, *Close Window*

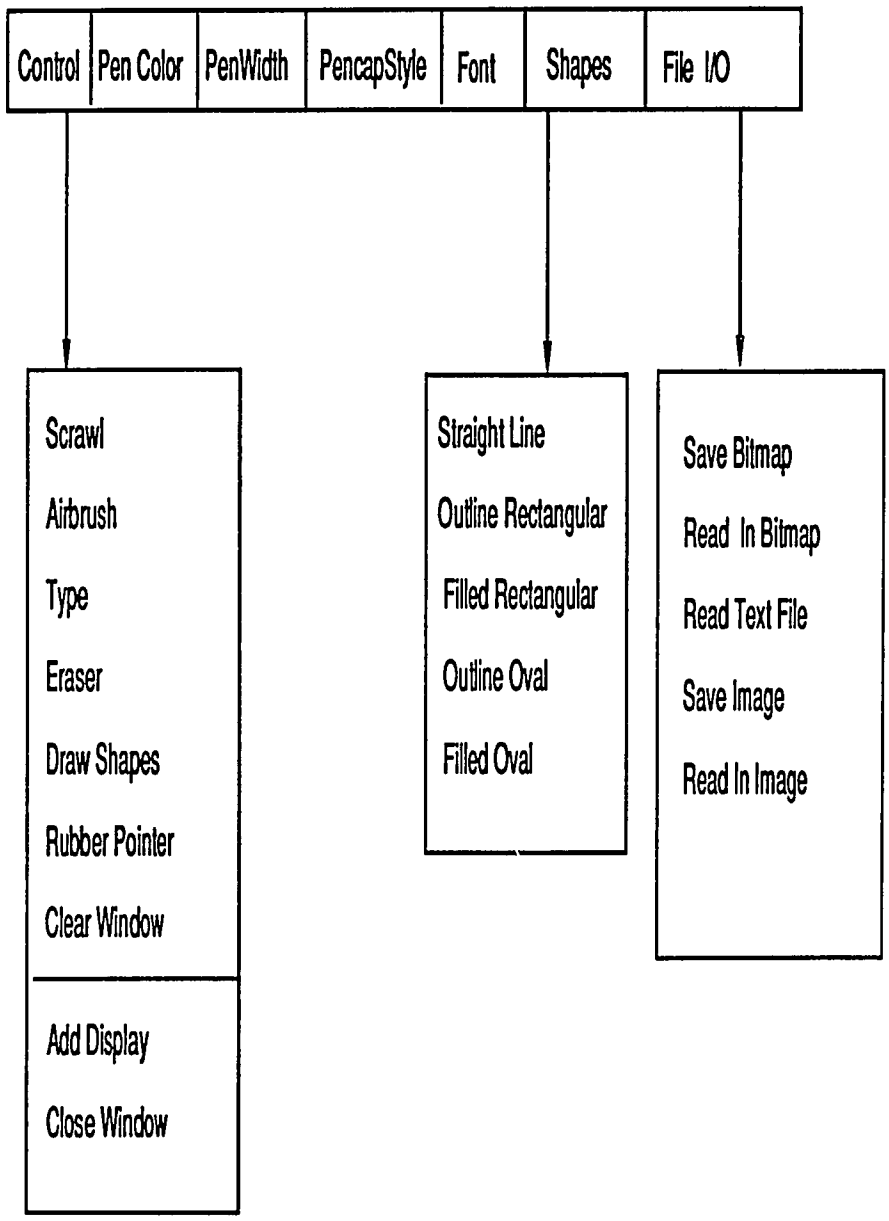

Figure 2.6: Main Menu of WSCRAWL

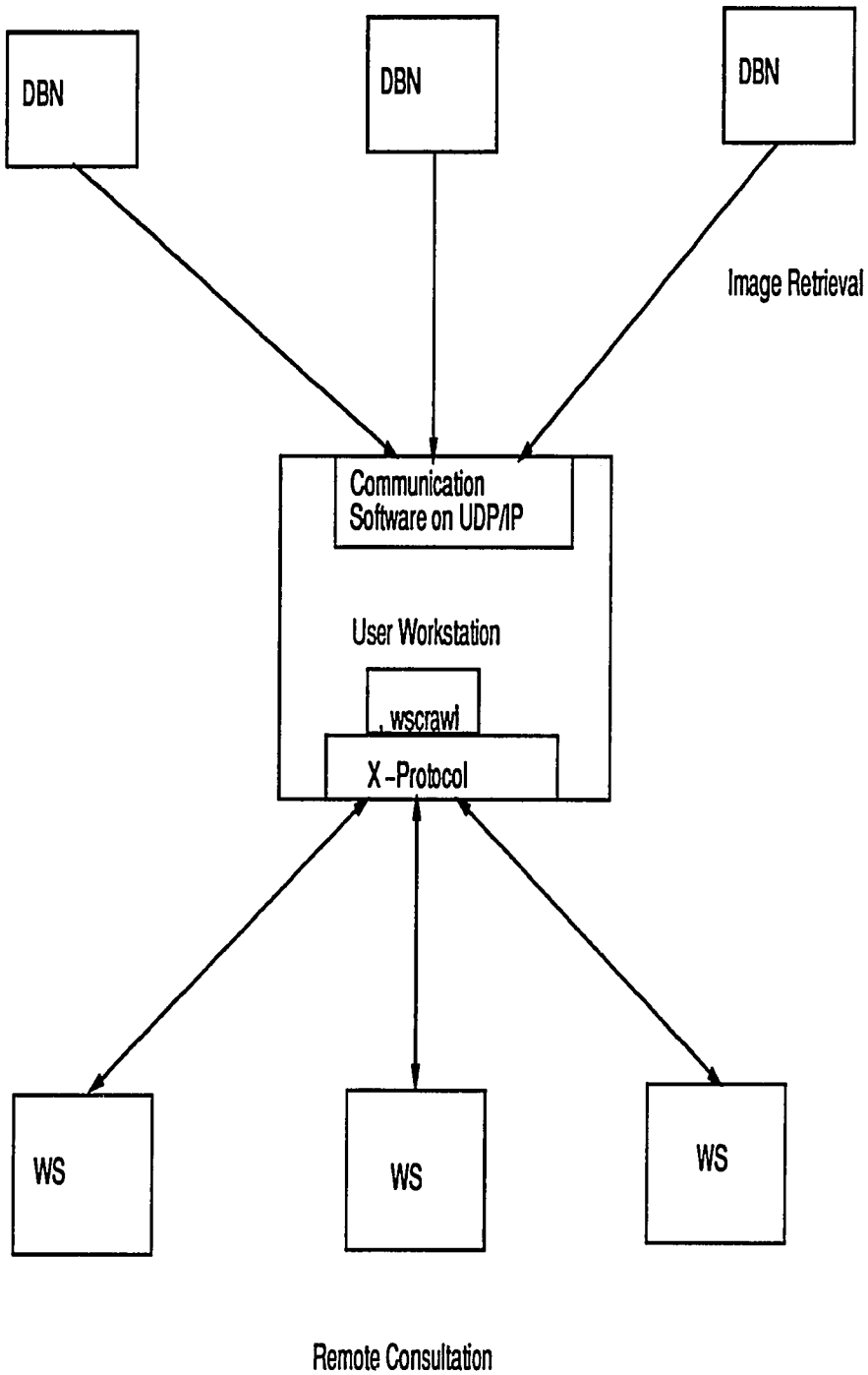

Figure 2.7: Workstations Interact With Each Other Using WSCRAWL

should be selected from *Control* in the main menu. Closing window in one workstation, will not close image window in remote workstations. Each user at different workstation must close it individually.

HELP: When the pointer is clicked on HELP command button, menu will be popped up at the point where click is made. The pop up menu has help text for FOLDER LIST, RETRIEVE SINGLE IMAGE,VIEW and REMOTE SESSION. Selecting the help topic by clicking, displays help text in a separate window. User can scroll the text up and down by using the scroll bar and middle mouse button. By clicking on Esc command button, the help window will pop down. By clicking on QUIT, the menu for help topic pops down.

QUIT: The ImageNet program can be terminated by clicking on the QUIT button in the main menu.

2.4 Communication Software

Figure 1.2 shows the protocol stack of ImageNet. XDR is used in the presentation layer to convert the data into hardware independent standard format[18]. XDR must be used when the network is made up of heterogeneous machine architectures. For retrieving image and image directory listing of remote node Remote Procedure Call(RPC) is used. RPC protocol specifications are written in Sun's RPC language

which when compiled by rpcgen compiler produces stubs which take care of communication software details. In the transport layer User Datagram Protocol(UDP) is used. The justification of its used is given in Chapter-3.

2.4.1 Remote Procedure Calls

Remote Procedure Calls are a distributed processing method designated to allow programmers to write client/server based distributed programs without having to be aware of details of underlying network[16][20][21]. Figure 2.8 illustrates the RPC model. Following are the steps of RPC:

1. The client routine calls the client stub. This call is like a local procedure call.
2. The client stub takes care of all network specific details. It marshals the arguments to send over the network. It makes a local system call to local kernel.
3. The network messages are transferred to remote system. Connection-less User Datagram Protocol is used for this application.
4. The server stub procedure is awaiting on remote system. It unmarshalls the arguments.
5. The server stub executes a local procedure call to the actual server function, passing it the arguments that it received over the network message.
6. The server procedure returns the value to the server stub. In the image file transfer example it returns the bytes read from the image file.

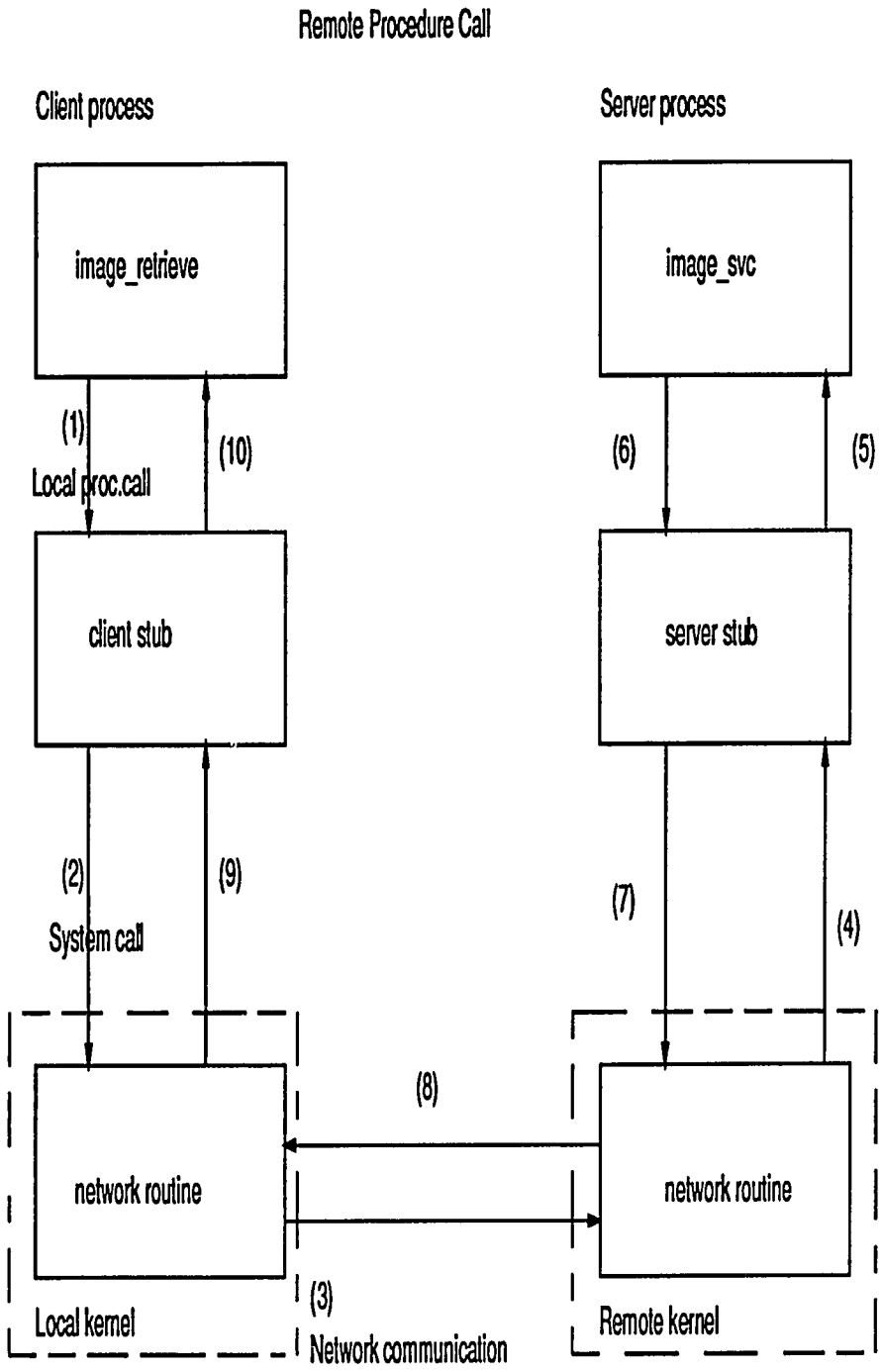

Figure 2.8: Remote Procedure Call(RPC) Steps

7. The server stub marshalls the return value and makes the system call.

8. The data gets transferred over the network.

9. The client stub reads the return value from the network.

10. The client stub returns the the value to the client process.

While the goal of Remote Procedure Calls is to hide the details of networking by providing a procedural interface, there are many important differences between Remote Procedure Calls and normal procedure calls.

1. Error handling: Remote Procedure Calls are vulnerable to perils as client/server machine crashes and lost packets.

2. Global Variables and Side Effects: Since there is no common address space between the client and the server, remote procedure can not manipulate global variables or cause other side effects.

3. Peformance: Remote Procedure Calls are much slower than local procedure calls.

4. Authentication: Programs using RPCs may need an authentication scheme when handling sensitive data over an unsecure network.

5. Parameter Passing: While passing variables by value is valid in the RPC paradigm, passing values by reference does not make sense, since the client

and server do not share the same address space. Many RPC implementations avoid this difficulty by not allowing remote procedure calls with pass by reference parameters.

6. Binding: Binding involves mapping a remote procedure call in a client to the server's host and process address. In Sun's implementation of RPC, clients must know the host the service resides on, and the server's port is determined using the portmapper service. The portmapper provides a lookup service for server ports. When a service is started, it registers the port it is listening to and the services it provides with the portmapper. A client then queries the portmapper to obtain the port of the service it desires.

7. Network Transport: There are no reliability guarantees implicit in RPCs . The programmer must be aware of the characteristics of the underlying transport protocol before making any assumptions about the reliability of the procedure calls.

8. Call Semantics: With local procedure calls, there is no question of how many times the called procedure were executed. With RPCs, however, this is not always the case. The possibility of server crashes alter the semantics of RPCs from those of conventional procedure calls.

There are three ways for clients to handle server crashes:

1. Do Nothing: In this case, the client waits forever for the server to respond.

2. Time Out and Raise Exception: The client sets a timer before calling the RPC. If the timer expires before a reply is received, the call 'returns' indicating error and invokes an exception handler. In this scenario, the procedure will have executed at most one time.

3. Timeout and Retransmit: If no reply is received after a certain length of time, the RPC is repeated. This leads to the possibility of the RPC having been executed more than once.

There are three types of RPC semantics depending upon dealing with server crashes:

1. Exactly Once: Every procedure call is executed once. This can not be guaranteed in a network environment.

2. At Most Once: In this semantics the RPC is executed at most once or not. If the normal value is returned, it is executed once. If the error is returned then it is executed once or not at all. Operations that are non-idempotent fall into this category. A non-idempotent operation is the one that can not be repeated over and over without harm.

3. At Least Once: It is guaranteed that RPC is executed at least once. It involves timeouts and retransmissions. This is acceptable for idempotent operations.

At least once semantic is implemented for the image retrieval and image directory listing remote procedure call implementation.

2.4.2 Sun's Remote Procedure Call Facility

Sun's RPC facility is used for this project for RPC implementation. It has several advantages:

1. It has 'C' like *rpcgen* language for protocol specification. After compiling the protocol specification file in *rpcgen* language it generates client stub, server stub and XDR encoding-decoding routines.

2. The user need not know the details of socket and UNIX network programming. But at the same time it provides the flexibility to change lower level network details.

Figure-2.11 shows how to generate stubs. XDR encoding-decoding files and executable client and server procedures. The protocol specification for remote image file transfer is in umage.x file. When it is compiled by rpcgen compiler it generates umage_xdr.c(XDR encoding-decoding file), header file umage.h, server stub umage_svc.c and client stub umage_clnt.c. The executable client and server processes can be generated by compiling as follows:

cc -o umage_svc umage_proc.c umage_xdr.c umage_svc.c umage.h

cc -o umage_retrieve umage_clnt.c umage_xdr.c

The umage.h and umage_xdr.c are the common header and XDR encoding-decoding files for client and server processes. XDR is a Sun's data representation scheme which converts the data into hardware independent form. It also takes care of network byte

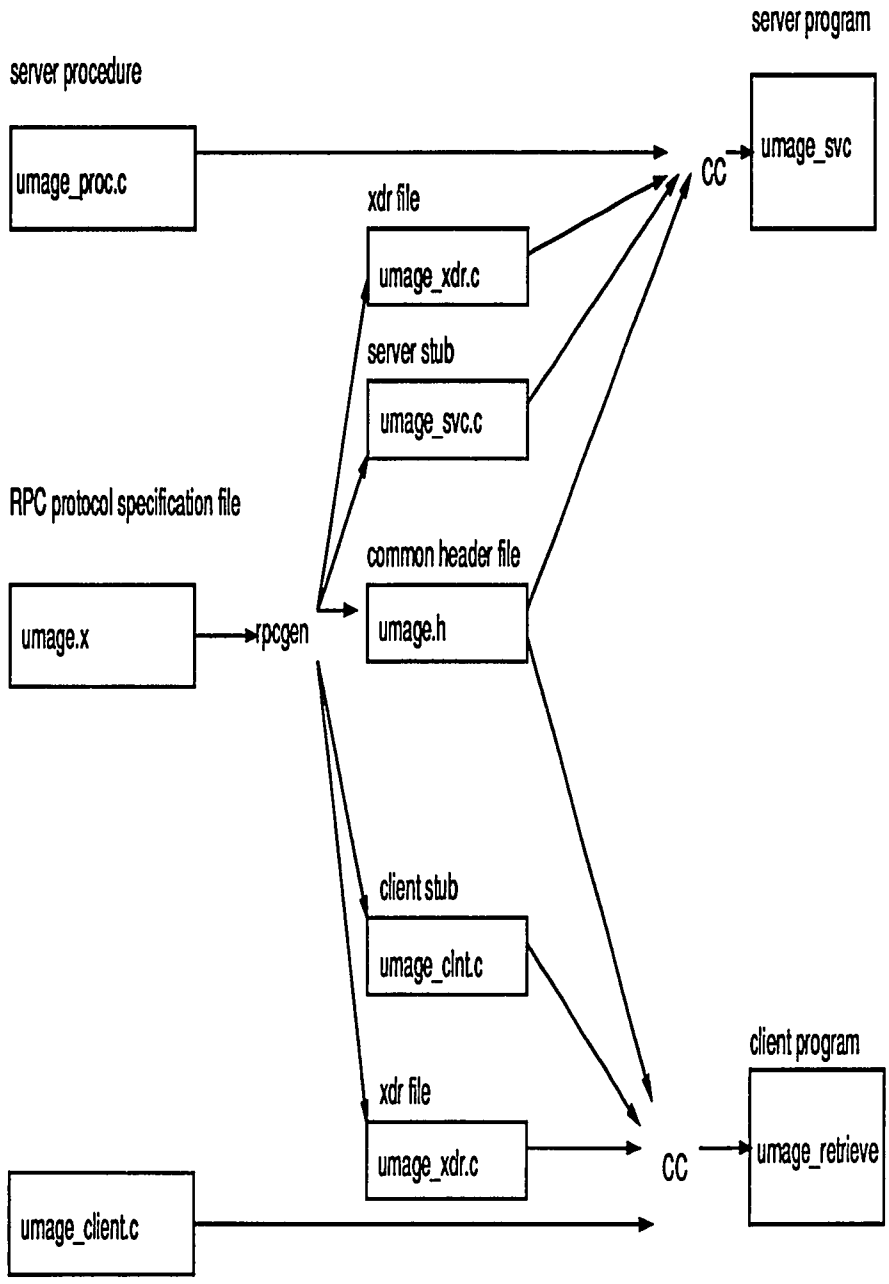

Figure 2.9: Stub Generation

ordering. Since the architecture of the existing user workstation and DBN may vary, it is necessary to use XDR to make the software portable. XDR fits in the presentation layer of the ISO model. The RPC and NFS protocol make use of XDR to present their data. All data items encoded with XDR are represented as 4 bytes, that is, the number of bytes must always be a multiple of four . To ensure 4-byte boundaries, zero padding is employed. This 4-byte encoding is used to prevent alignment problems with machines that use different word lengths and to keep the encoded data from growing out of proportion. Two byte encoding would have no alignment problems but considerably increase the amount of the encoded data. Bytes in a data stream are numbered from 0 to n-1 and they are always read and written in that order. The purpose of XDR is to provide representation for the most commonly used data types. A special data type provided by XDR is called opaque data, which is just a stream of uninterpreted bytes. This data type is used by the communication software to encode image data and transmit it across the network. Explicit data typing is not used in XDR, thus saving space and decreasing the overhead associated with reading the data type fields and performing the required decoding.

Sun's Remote Procedure Calls differ from local procedure call in the following aspects:

1. It always takes pointers to their arguments rather than arguments themselves.

2. It returns the pointer to the result rather than result.

3. The program name in the program definition is converted to all lower-case letters, an underscore is appended and version number is appended. e.g.In the protocol specification file for image file transfer, the program name is IMAGETRAN in umage.x file, hence , the procedure name is imagetran_1.

2.4.3 Execution Steps Of RPC

1. When umage_svc server program is started on the remote system, it creates a UDP socket and binds any local port to the socket. It then calls the function *svc_register* in the RPC library, to register its program number and version . This function contacts the port-mapper process to register itself. The portmapper is usually started as a daemon when the system is started. The portmapper keeps track of the program number, version number and port number. The server process umage_svc waits for a client request.

2. Client program umage_retrieve is started on local system. This call specifies the name of the remote system, the program number, version number and the protocol. This function contacts the portmapper on the remote system to find out the UDP port for the server.

3. Client program calls imagetran_1 function. The function is defined in the client stub and it sends a datagram to the server, using UDP port number from the previous step. It then waits for response. XDR encoding and decoding is done by umage_xdr.c routine. When the response is received, client stub takes

the value and returns it to image_retrieve program. The execution steps are summarized in Figure 2.10.

2.5 Background of X-windows

The X-window System is a hardware independent and operating system independent windowing system. X-window controls a "bit-mapped" display in which every pixel is individually controllable. This allows applications to draw pictures as well as text.

Figure 2.11 shows the layering of software in an application that uses Xt Intrinsics and a widget set. Intrinsics are based upon Xlib, the lowest level C-language interface to X-window. Xlib provides full access to the capabilities of the X protocol, but at the cost of simplicity of programming. Since all the user-interface specifications could be implemented by X-toolkit, the software development was done using it due to it simplicity compare to Xlib. Xt is built upon Xlib. Xt provides an object-oriented layer that supports the user-interface abstraction called a *widget.* A *widget* is a reusable, configurable piece of code that operates independently of the application except through prearranged interactions. A *widget set* is a collection of widgets that provide commonly used user-interface components tied together with a consistent appearance and user-interface. There are several different *widget sets* from various vendors that are designed to work with Xt. Athena widget set is the only available widget set in ECE department computing facilities, so it was chosen

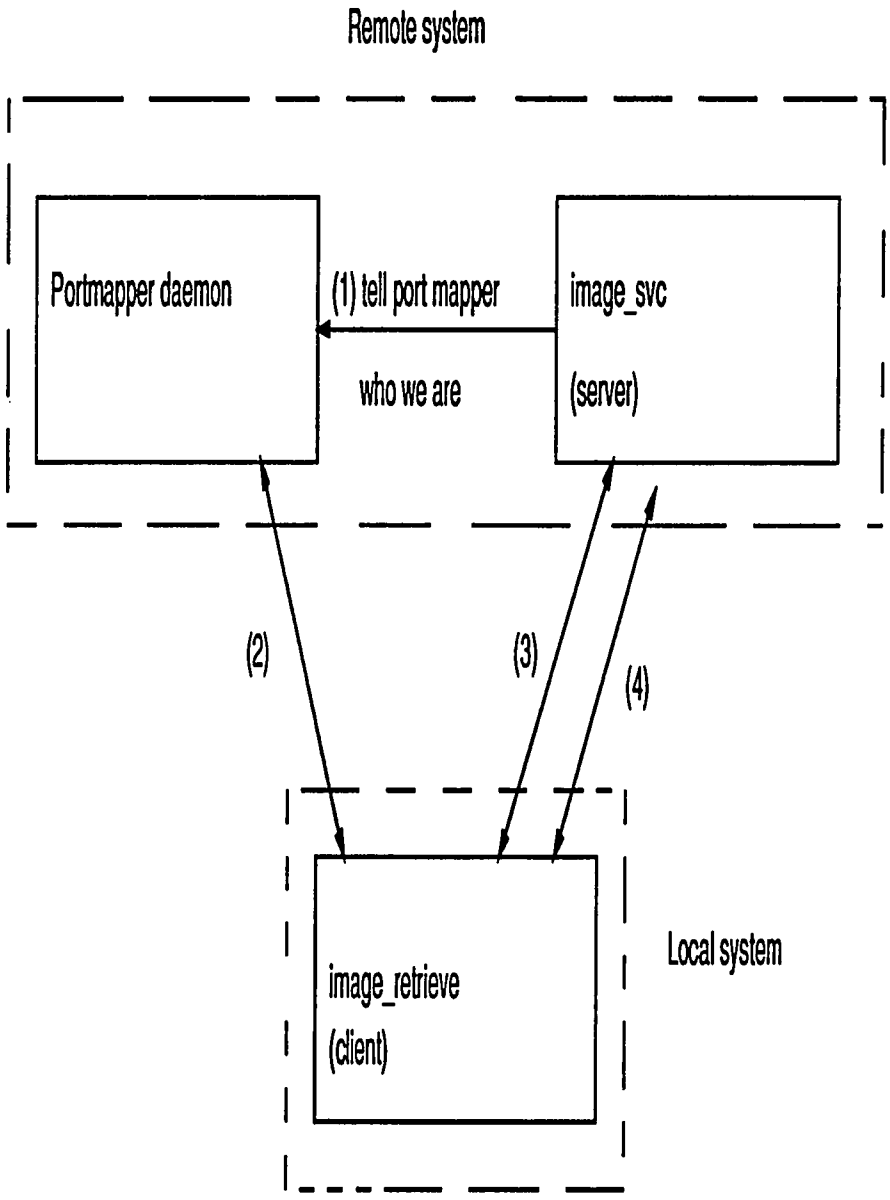

Figure 2.10: Steps In Sun RPC

for implementation of all the functions. Applications can call Xlib directly even if Xt is used. So, the features like access to X protocol and graphics call which are not provided by Xt, if required in future application can be accomplished by calling Xlib directly.

2.5.1 Widget Classes And Instances

A *widget set* defines classes of widgets. Each widget class has certain procedures and data associated with it, where the data is a structure that defines the characteristics for that specific class. From the programmer's point of view it is a pointer to a structure of its class. A *widget class* has certain fixed features which are common to all instances of that class and certain characteristics which can be changed from instance to instance. The configurable feature is called a *resource.* Figure 2.12 shows Athena widget hierarchy. There are three fundamental widget classes:

1. Core widgets.

2. Composite widgets.

3. Constraint widget.

Widget features and characteristics are inherited from basic classes of widgets. Class inheritance is an important concept for a widget user because resources of a widget class are defined not only be that class but also by the classes that class inherits features from, called its *superclasses.*

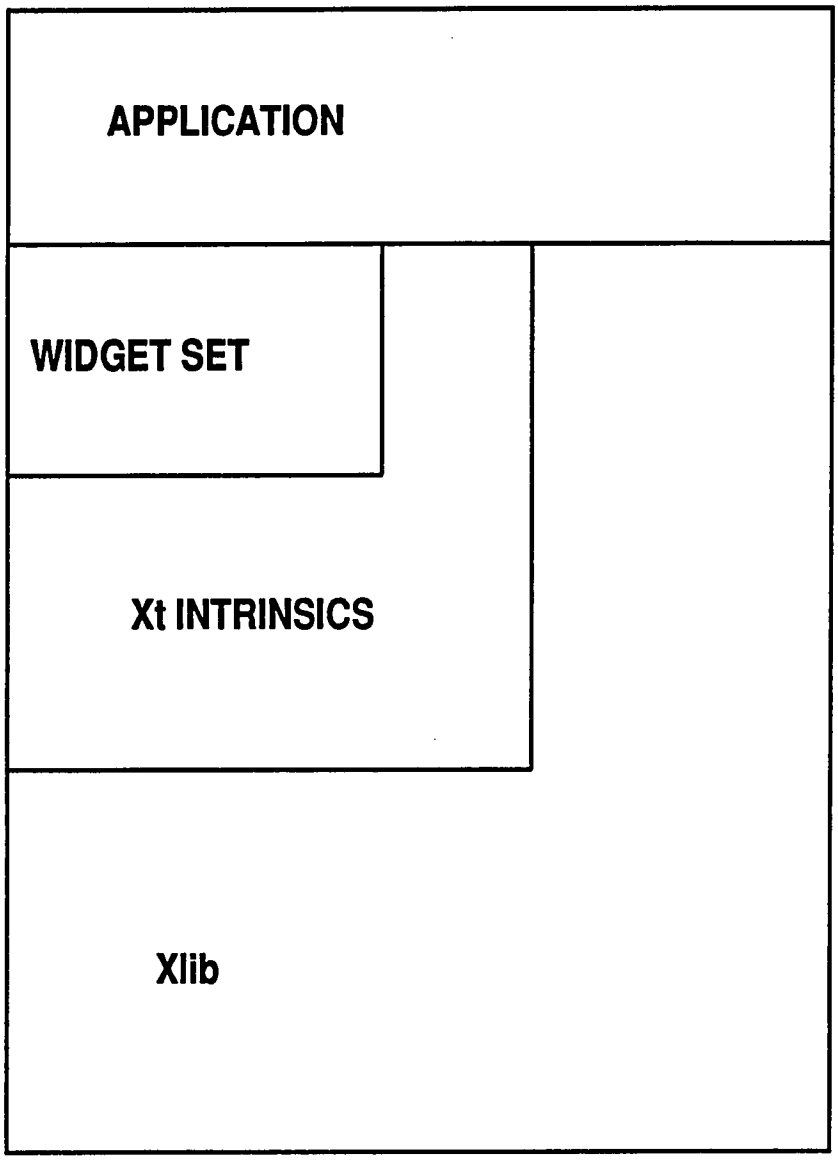

Figure 2.11: X-windows Software Layer

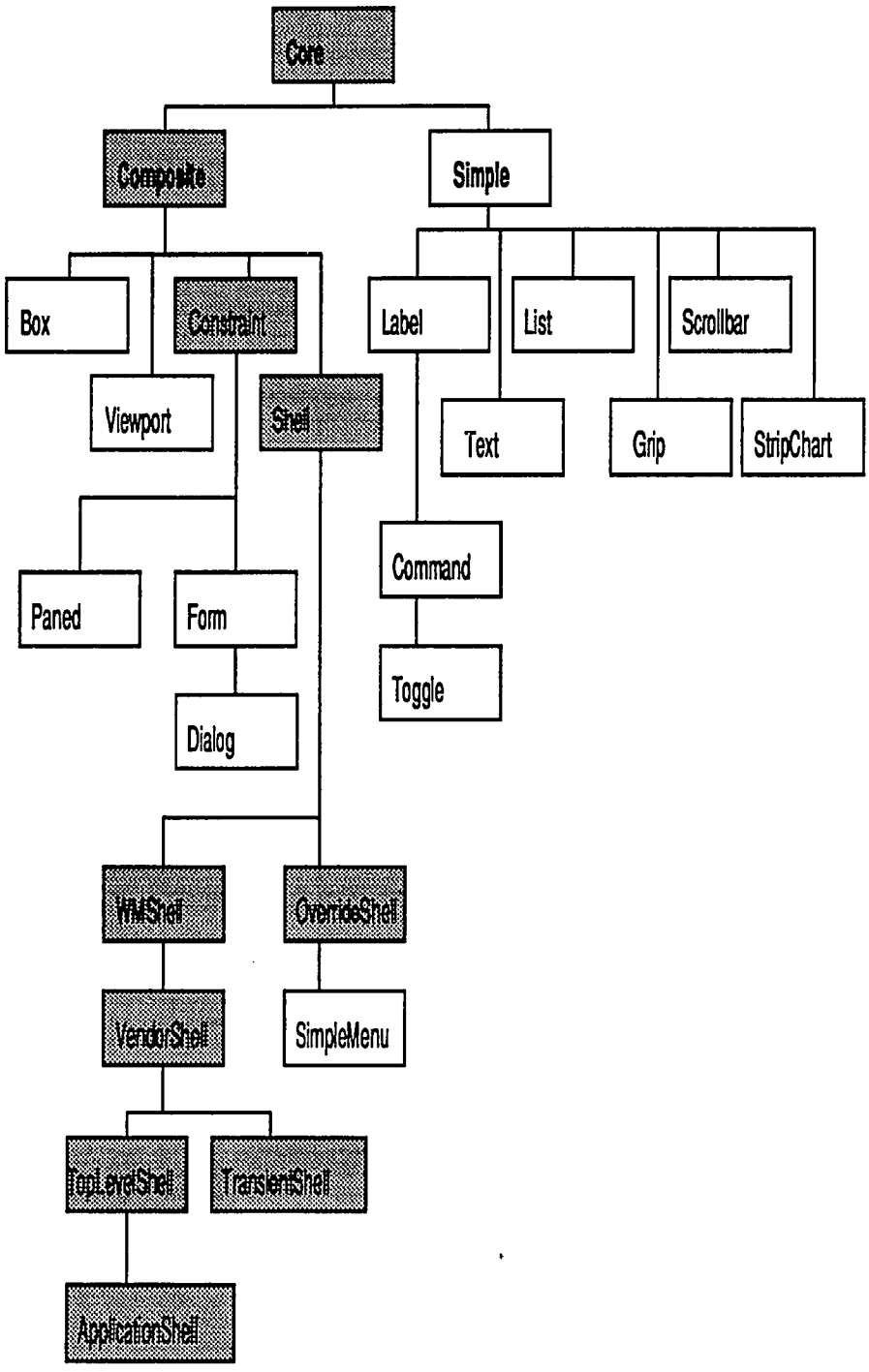

Figure 2.12: Athena Widget Hierarchy

The core widget class, defined by Intrinsics, is the root of the hierarchy , from which all other classes are descended. The core class defines characteristics common to all widgets, such as size and position. The Athena Simple widget class inherits basic features from Core and adds a few minor features of its own. The Label widget in turn adds the ability to print a string, and mechanism for changing the font and placement of the string. Command then inherits features from Label and adds more features. Command is known as a *subclass* of Label and Label is the *superclas s* of Command. So,lower classes in the hierarchy have more features.

2.5.2 Widget Application Interaction

The user can set configurable features by XtSetvalues function calls. There are three separate mechanisms that can be used to link widgets and a application functions: callbacks, actions and event handlers. In ImageNet user-interface design callbacks and actions are used. A widget expecting to interact with an application will declare one or more callback lists as resources, the application adds functions to these callback lists, which will be invoked whenever the predefined callback conditions are met. Callback lists are resources, so that the application can set or change the function that will be invoked. Figure 2.13 and Figure 2.14 illustrates the mechanisms of interactions. The prototype for a callback procedure and the XtAddcallback intrinsics are given below:

```
void callbackproc(w,client_data,call_data)
```

```
Widget w;
XtPointer client_data;
XtPointer call_data;

voide XtAddcallback(w,callbackname,callback,client_data)
Widget w;
String callbackname;
XtCallbackProc callback;
XtPointer client_data;
```

While using callback procedures user can pass data to it via client_data, call_data is passed to the callback procedure by the widget itself.

A widget's callbacks are often designed to support the intended use of the widget while adding actions to a widget can make it behave in ways the designer did not foresee and the user might not expect. Thus,Actions are most appropriate for adding minor features to an existing widget when the widget does not provide a callback that is required for building a specified application window by adding to a core widget. Action procedures must be registered with the application by calling XtAppAddActions and declared in a transition table in the application-defaults file. The function prototype for an action procedure is given below:

```
static void Action_procedure(w,event,params,num_params)
```

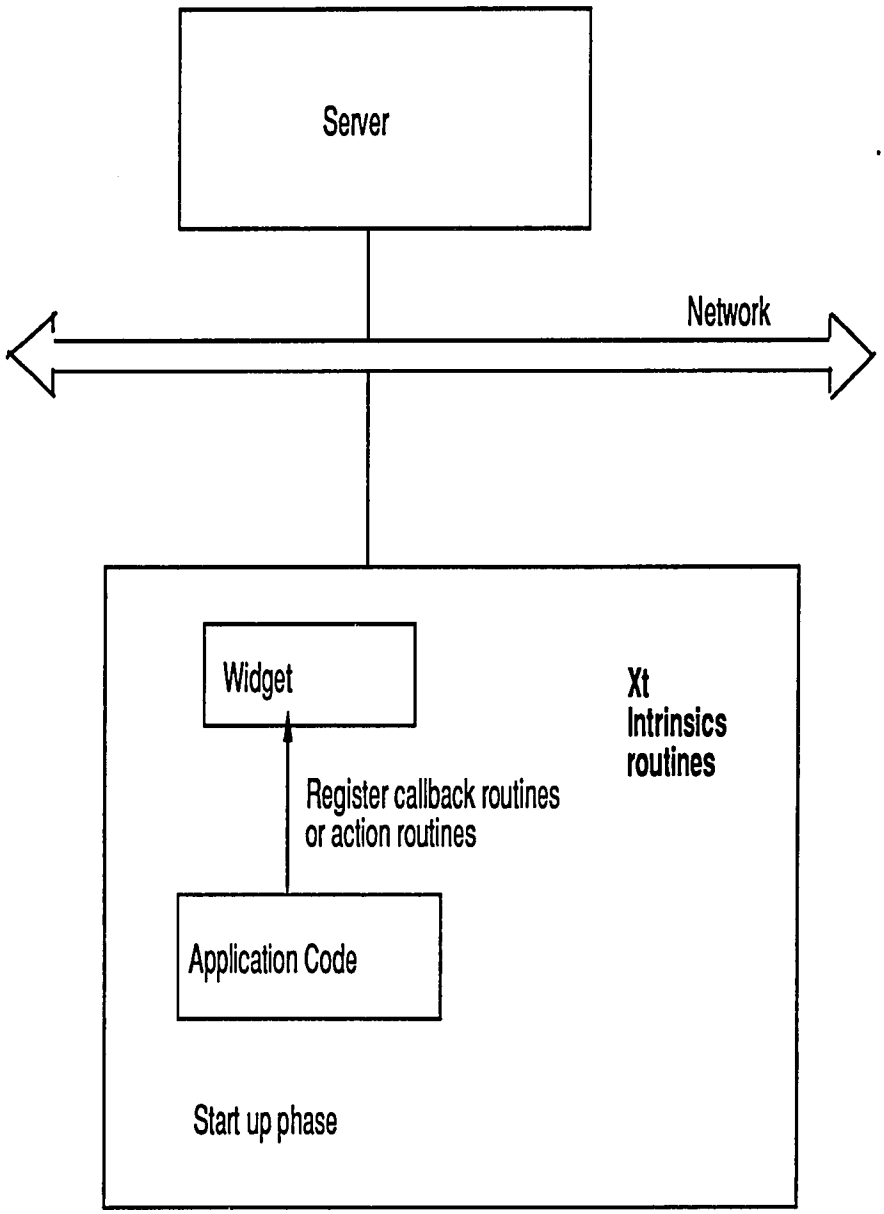

Figure 2.13: Widget Application Interaction:Registration of Callbacks and Actions

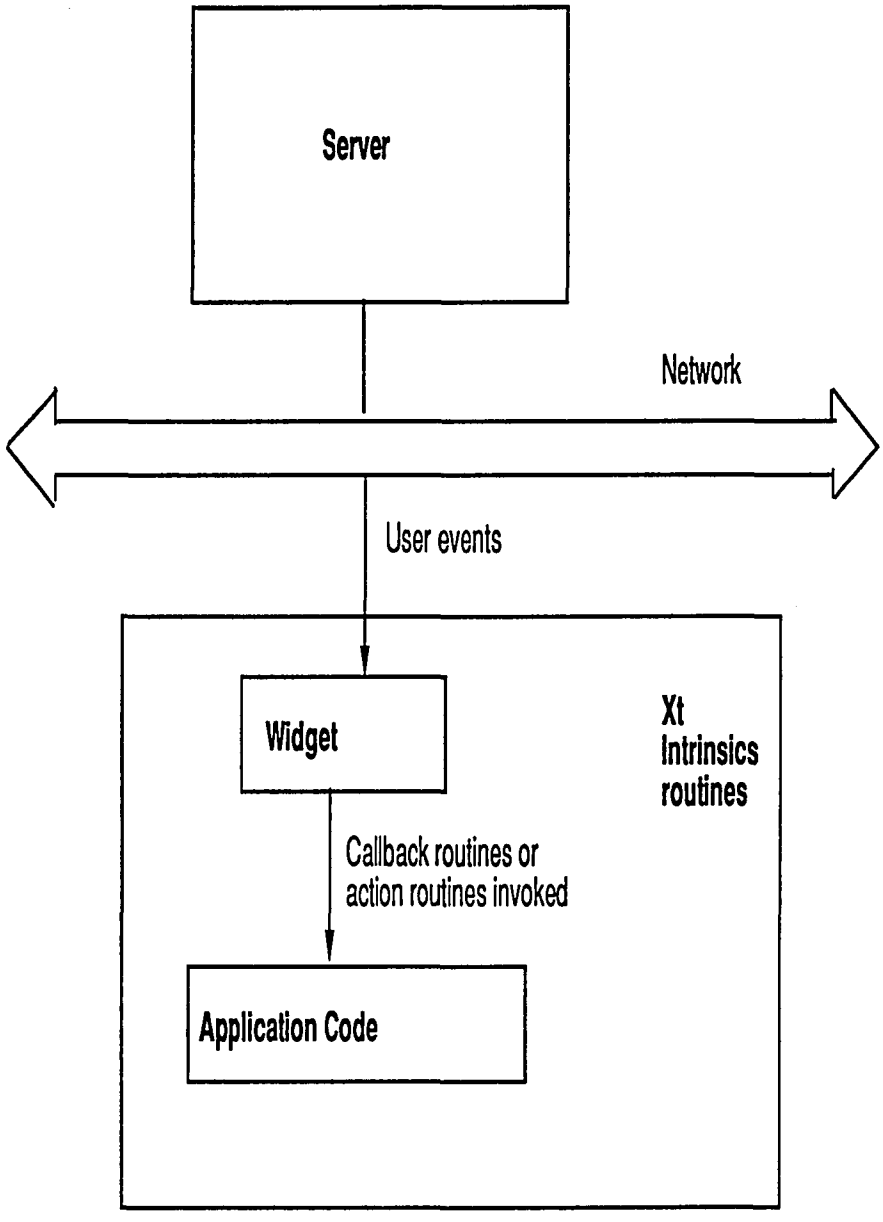

Figure 2.14: Widget Application Interaction

```
Widget w;
XButtonEvent *event;
String *params;
Cardinal *num_params;
```

2.5.3 Application Structure

All X-toolkit applications have the same basic structure as follows:

1. Include <X11/Intrinsics.h> and <X11/StringDefs.h>, the standard files for Xt.

2. Include the public header file for each widget class used in the application.

3. Initialize the Toolkit with XtVaAppInitialize function call.

4. Create widgets and inform their composite parent widget about them.

5. Register callbacks, actions and event handlers ,if any, with Xt.

6. Realize the widgets by calling XtRealizeWidget. Call should be made only once in the entire application, passing it the shell widget returned by XtVaAppInitialize.

7. Call XtAppMainLoop, where Xt takes control of the application.

CHAPTER 3

USER INTERFACE AND NETWORK SOFTWARE DESIGN

In this section, the functions are mapped into procedures in X-windows and communications software. The overall software design is broken into two parts:

1. X-windows user-interface.

2. Network software design.

Section 3.1 describes the X-windows user-interface design and interface between X-window and network software. Figure 3.3 and figure 3.4 describes the interface between X-window and network software to implement directory listing and image retrieval functions. The details of image retrieval function with error handling in application layer is described in section 3.2 and section 3.3. Section 3.2 describes network software design. Section 3.3 deals with experimental results for transport protocol selection. Use of Remote Procedure Calls with User Datagram Protocol(UDP) and application design to deal with lost packets in UDP are also dealt in detail. Figure 3.1 shows top-down overall software design.

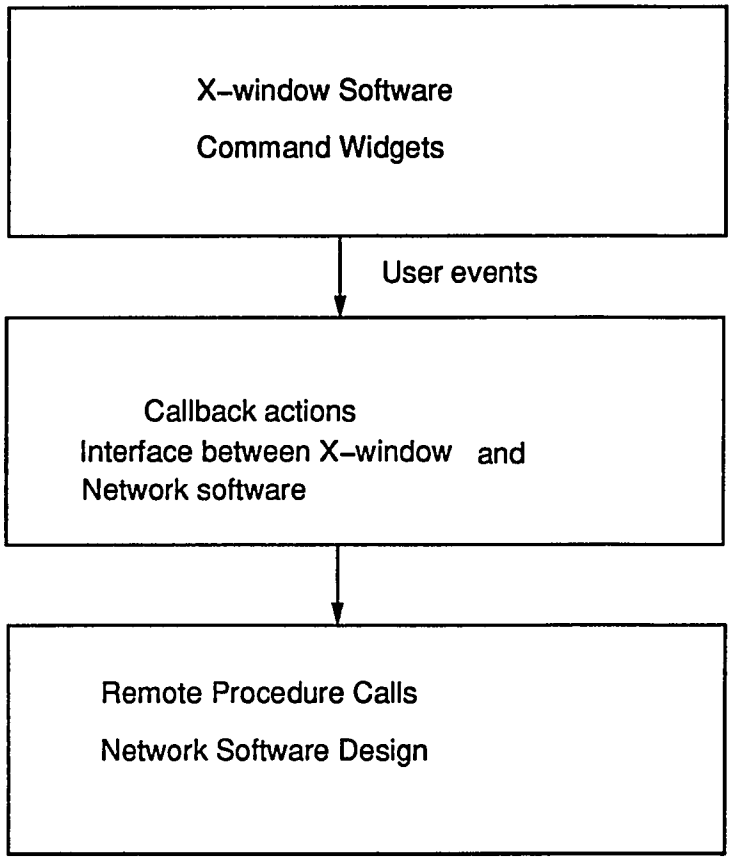

Figure 3.1: Overall Topdown Software Design

3.1 X-windows User-Interface Design

This section describes X-window user-interface design in detail. Figure 3.2 shows the main menu when the main program is executed. Implementation of all pop-up menus and dialog-boxes are carried out using X-toolkit and Athena Widget Set. Design description of each menu item, callbacks and actions are described in this section. The section starts with some fundamental design concepts of X-windows. For detailed description please refer to [13][14][21] and [22].

3.1.1 Toplevel Menu Design

Figure 3.2 shows the widget hierarchy and interactions with application for main menu. Widget instances with their classname is specified in the rectangular box and interactions and function calls are specified in the filled boxes. This notation will be followed for all the modules.

To create main menu, boxwidget is chosen as the composite widget. Composite widgets insulate the application programmer from having to place each widget individually or from having to reposition or resize various widgets when the application is resized by the user. Composite widgets automatically adjust the layout of their children when child widgets are added or removed. The box widget has six command widgets as its children. The orientation of box widget is set to horizontal in the application default file. When width of command widget is set, box will automatically change its size in the horizontal direction. The default label of the command widget

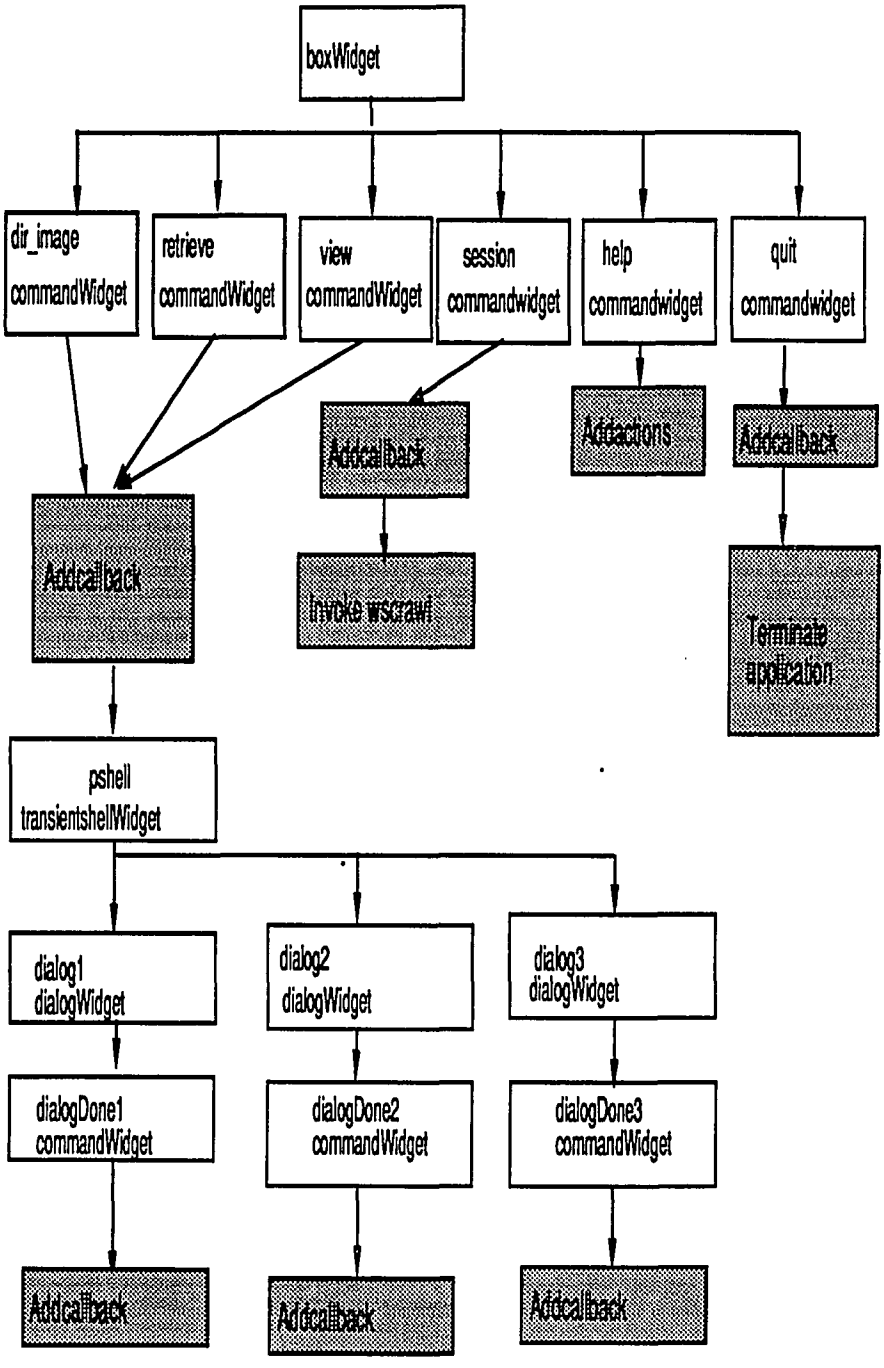

Figure 3.2: Main Menu Design

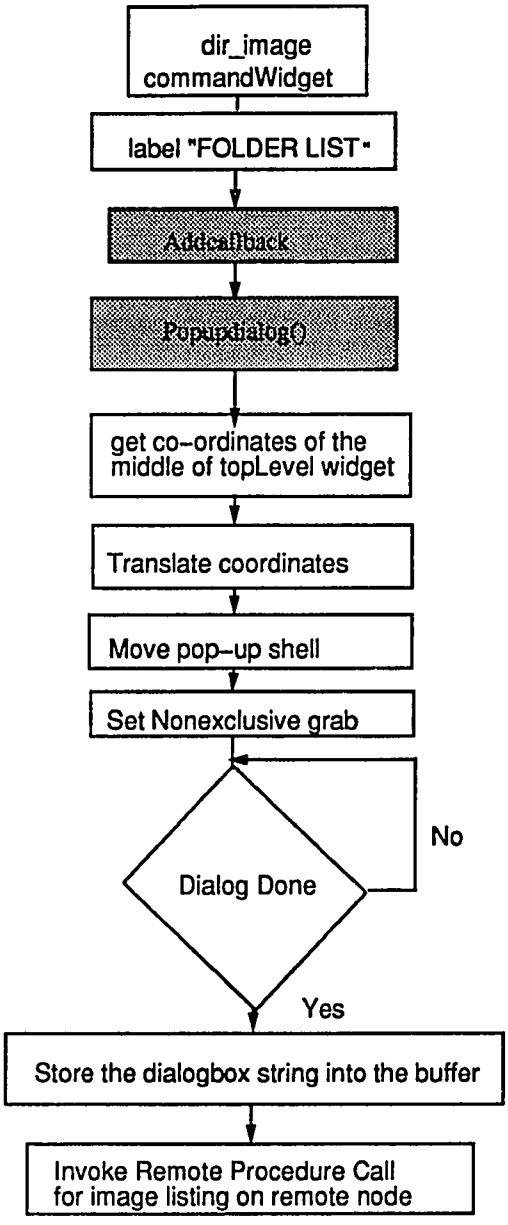

Figure 3.3: Remote Node Image List

is its name. The label of the command widgets are set in the application default file. Callback functions are registered for all the command widgets. Whenever pointer is clicked on the dir_image, retrieve or view widget, due to callback action it will pop-up the transient shell. Transient shell has two children, dialogwidget and command widget. The text message displayed in the dialog is set as label resource for the dialog widget in the application default file. When the pointer is clicked on the Dialog Done, the registered callback function will popdown the transient shell and passes the entered string in the callback procedure argument,client-data.

In the case of dir_image the callback function of Dialog Done, will invoke Remote Procedure Call(RPC) to retrieve remtote image directory listings while in case of retrieval, it will invoke RPC to retrieve image. Figure 3.3 and 3.4 illustrates logic flow of it. The label resource of dir_image widget is set to FOLDER LIST, so when the widget is actually mapped, it will display the string FOLDER LIST. The label resource is set in application default file. To grab the event of pointer click, callback function is registered, so when the click is made, it will pop-up transient shell. The transient shell is mapped by setting its co-ordinates. The dialog widget is a child of transientshell widget. The user inputs string in dialogbox of dialog widget. The Remote Procedure Call is invoked depending on input string. There are two types of pointer grabs: passive grabs and active grabs. An *active grabs* is invoked directly with the Xt function XtGrabPointer. This function tells the server that grab should begin right away and to continue until specifically released by UngrabPointer. Active

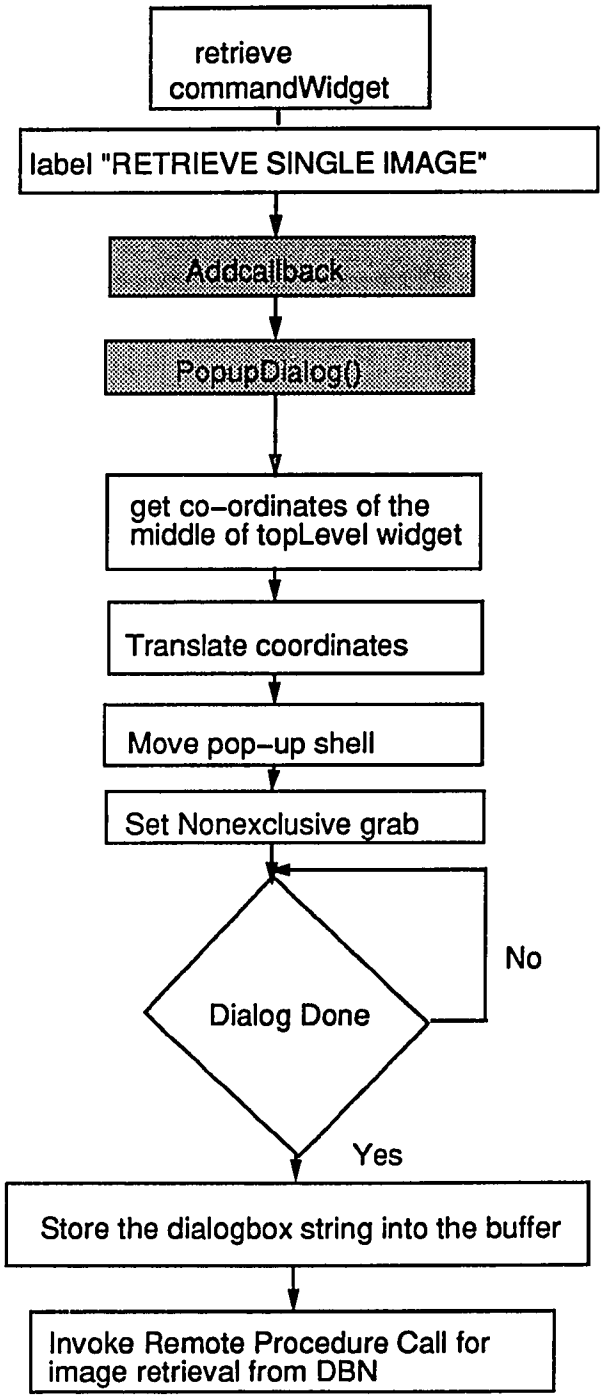

Figure 3.4: Image Retrieval

grabs are not useful for this type of application, hence are not used. The *passive grabs* is perfect for menus because it tells server that grab should begin when a certain combination is pressed in a certain window. The grab continues until the button in the combination is released. There are two local grabs, namely exclusive and nonexclusive, which affect only the distribution of events within the application. An *exclusive* Xt grab redirects all user events that occur within the application to the latest pop-up in the cascade. A *nonexclusive* Xt grab redirects events to whichever pop-up the pointer is in. In this application *nonexclusive* grab is used. The label resource of *retrieve* widget is set to RETRIEVE SINGLE IMAGE in application default file. The logic flow of it is illustrated in Figure 3.4 and it is identical to Figure 3.3, which is explained above.

When pointer is clicked on view, the callback will pop-up dialog for image viewing. The callback of Dialog Done will invoke XView software with image name as the argument. The callback function of session will invoke *wscrawl* software for remote consultation. The on-line help menu design is described in detail in the next section. The quit widget has the callback to terminate the ImageNet program.

3.1.2 On Line Help Menu Design

The reason for choosing action in place of callback for the help menu is the simplicity of setting the co-ordinates of pop-up menu. The menu is popped up at the

location where pointer is clicked. Figure 3.5 shows the steps to create pop-up help menu.

Figure 3.6 shows the widget hierarchy and interactions with the application. The orientation of the box widget is vertical, which is default. It has five command widgets, each of them has one callback function registered to it. The callback function of dir_list1,retrievel,view1 and session1 will pop-up one transient shell which has box widget as a child. The box widget has two children text widget and command widget. The resources can be set either through application default file or through hard coding in the program. The XtNstring resource is set by reading the buffer. The content of help text is stored in the file and everytime text widget is created, this file is read in the buffer. The resource XtNscrollVertical is set to XawtextScrollAlways, so that text widget will provide the scrollbar and the content can be scrolled on the screen. When click is made on escape command widget, the callback action is to popdown the transient shell.

3.1.3 Error Reporting

The errors in the Remote Procedure Calls are reported in separate error windows displaying system error message string. When user clicks on KILL button, it will terminate RPC. The system error message is retrieved by using errno as an index,

```
extern int errno;
char *sys_errlist[errno];
```

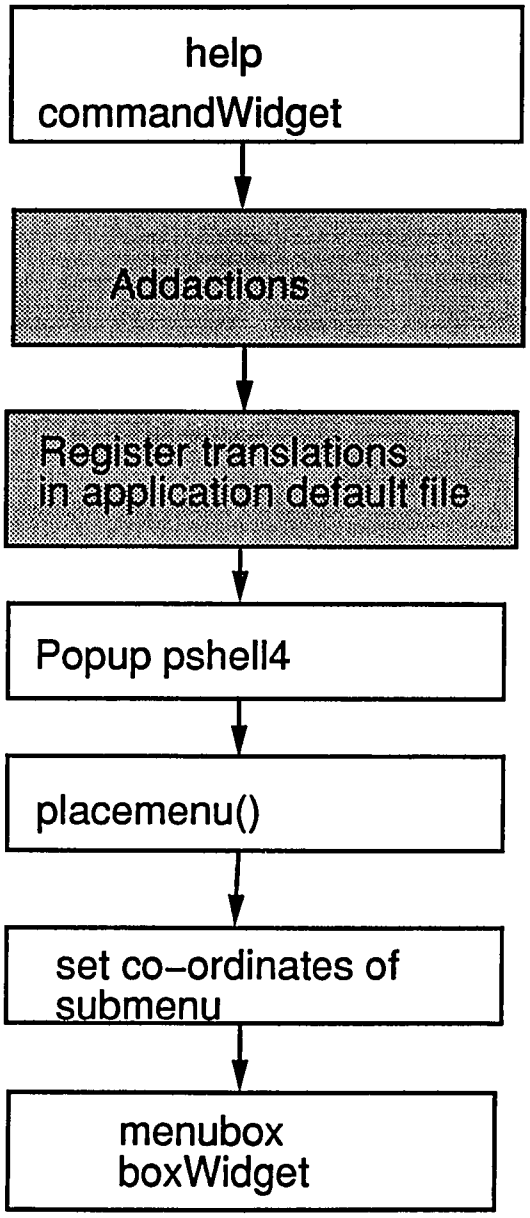

Figure 3.5: Widget Hierarchy and Interaction With Application of Online Help

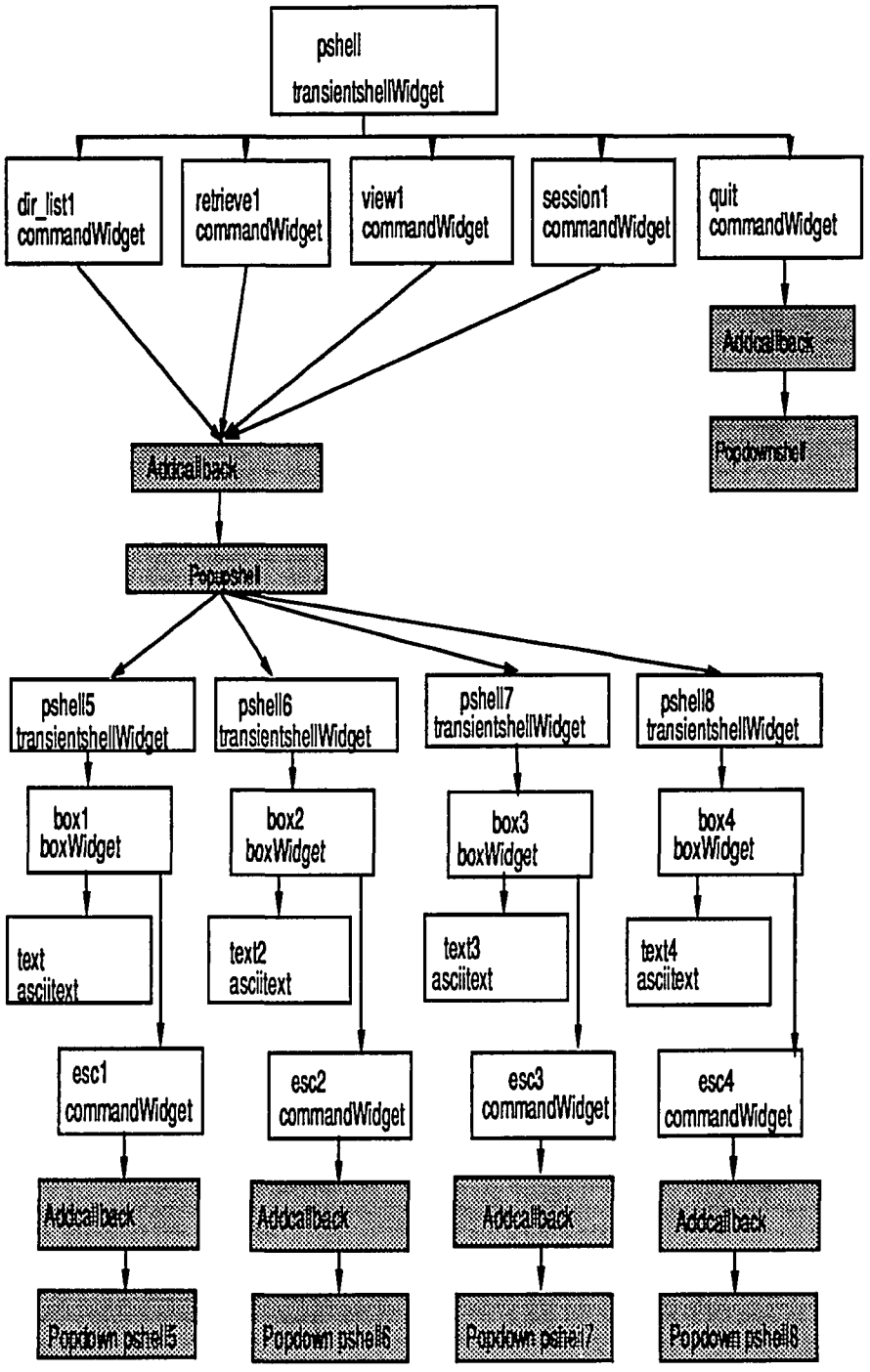

Figure 3.6: Help Menu:Widget Hierarchy and Interaction With Application

The errno is set according to most recent error and it is used as the index in sys_errlist. The error message is stored in the buffer. To generate error window, topLevel shell is created with command widget and label as its children. The label resource of widget is hardcoded by using XtVaSetValues. The label resource of command widget is set to KILL in application default file. The callback function registered with command widget terminates RPC and popdown the transient shell when pointer is clicked on command widget. Figure 3.7 illustrates the steps of error window generations.

3.1.4 Display of Retrieved Data

Figure 3.8 describes the steps of creating the window displaying the remote directory listing. Any error generated in the RPC will pop-up errorwindow. If the transfer takes place without error, results are stored in the buffer. A topLevel shell is created and ascii text widget and command widget are created as its children. The XtNstring resource of the ascii text widget is hard coded.

```
XtVaSetValues(textWidgetname,
XtNscrollVertical, XawtextScrollAlways,
NULL);
```

Setting the XtNscrollVertical resource will create the scrollbar and the displayed content in the text windows can be scrolled vertically with the middle mouse button. When click is made on escape command widget the callback action is to popdown the shell.

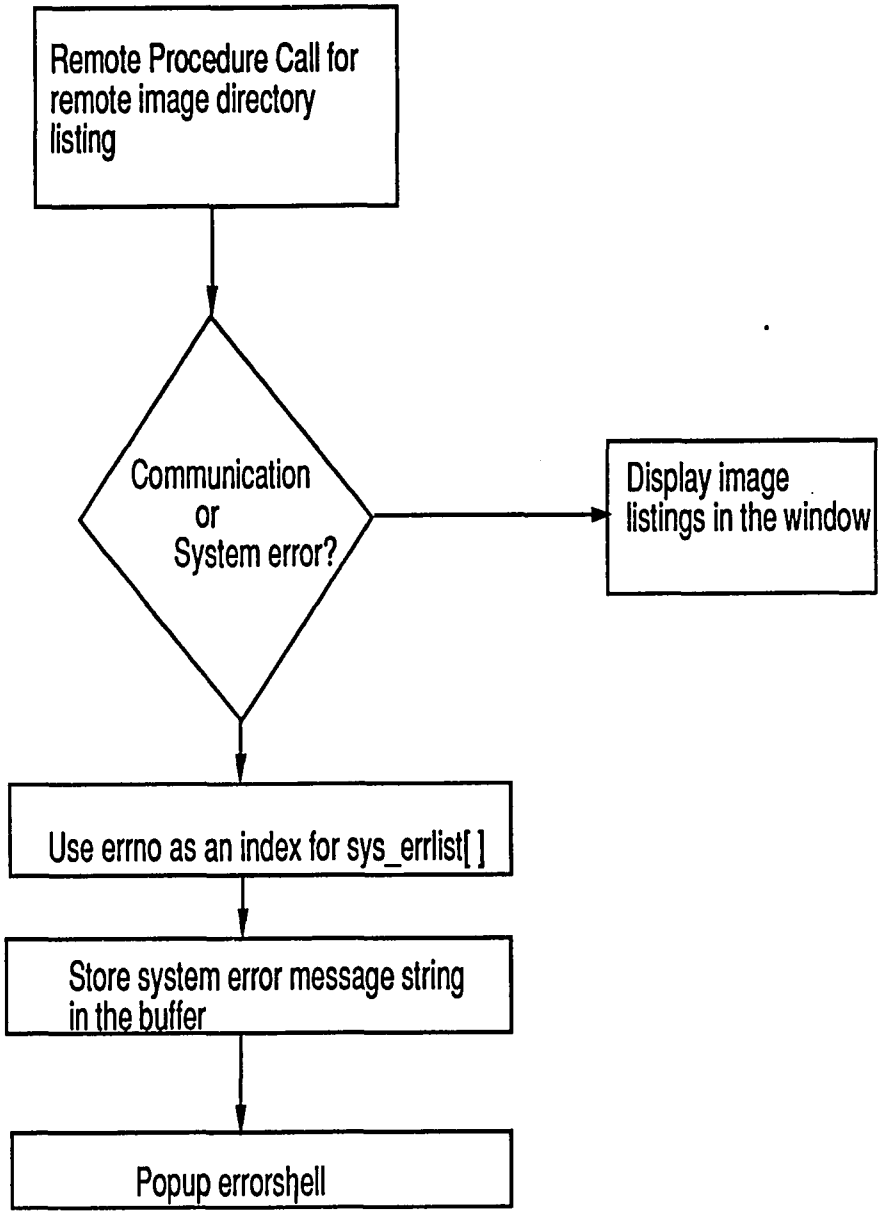

Figure 3.7: Error Window Generation

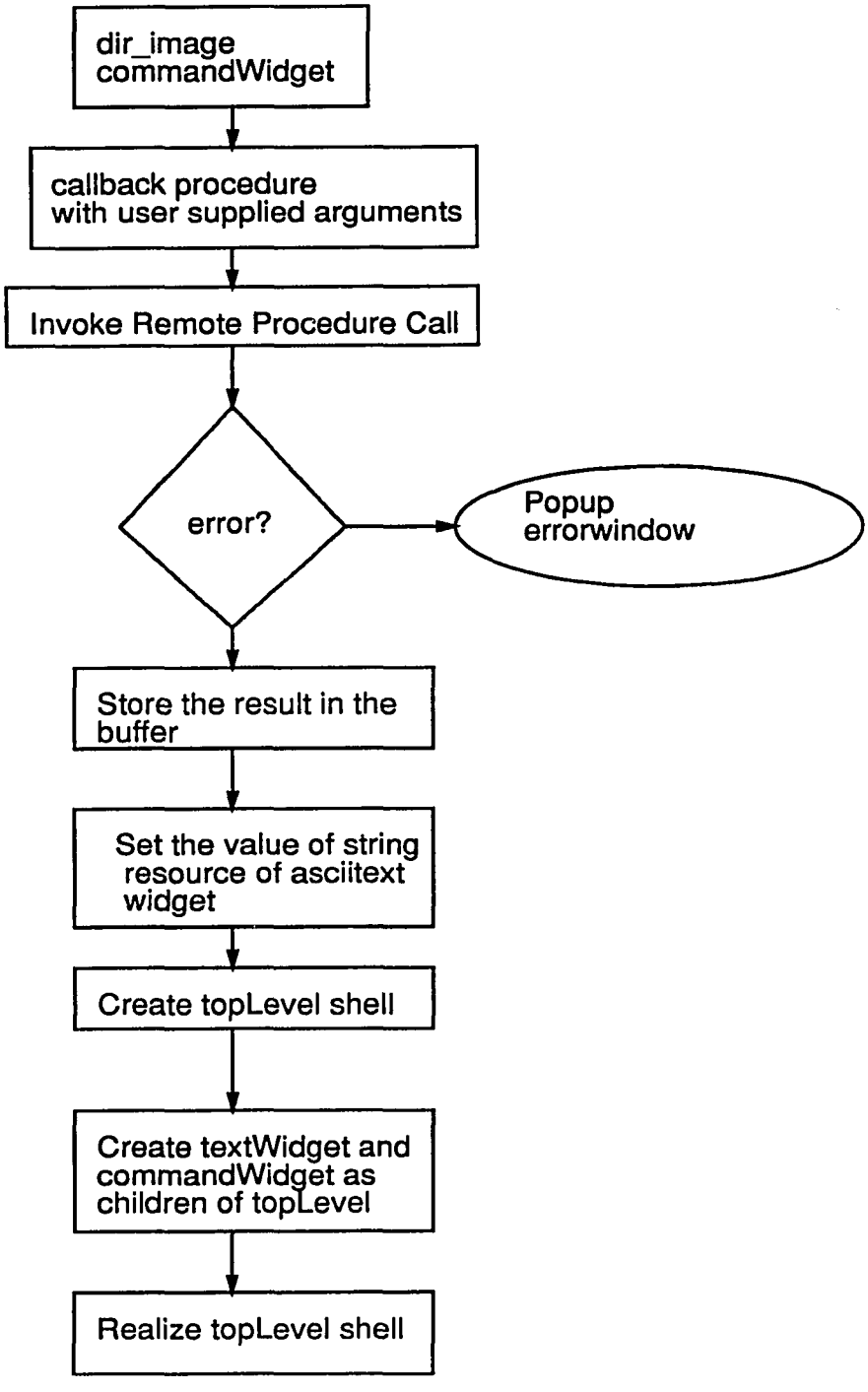

Figure 3.8: Display of Retrieved Remote Image List

3.2 Networking Software Design

There are two requirements implemented in network software:

1. Directory listing.

2. Retrieve Single Image.

Directory Listing:

The interface between command widget, dir_image, to the networking software is explained in section 3.1. The remote procedure call for directory listing is invoked as a callback action of the command widget. Figure 3.9 shows the logic of the remote procedure at the remote machine. The client at the local machine passes the directory name. The directory is opened at the remote server and link list is made of the names of images. In case of failure for opening the directory, the errno is returned to the server stub. The server stub will marshal the arguments and sent them back to the client machine. In this case the entire result will be retrieved in a single call because only the names of the images are required. So, there is no need to implement a error control in this case above RPC/UDP.

Retrieve Single Image:

The interface between the command widget, image_retrieve, to the networking software is explained in section 3.1 with figure 3.4. The remote procedure calls will be invoked to retrieve the image file from the remote machine. Since, UDP is not a reliable protocol, the application layer software design has to take care of lost

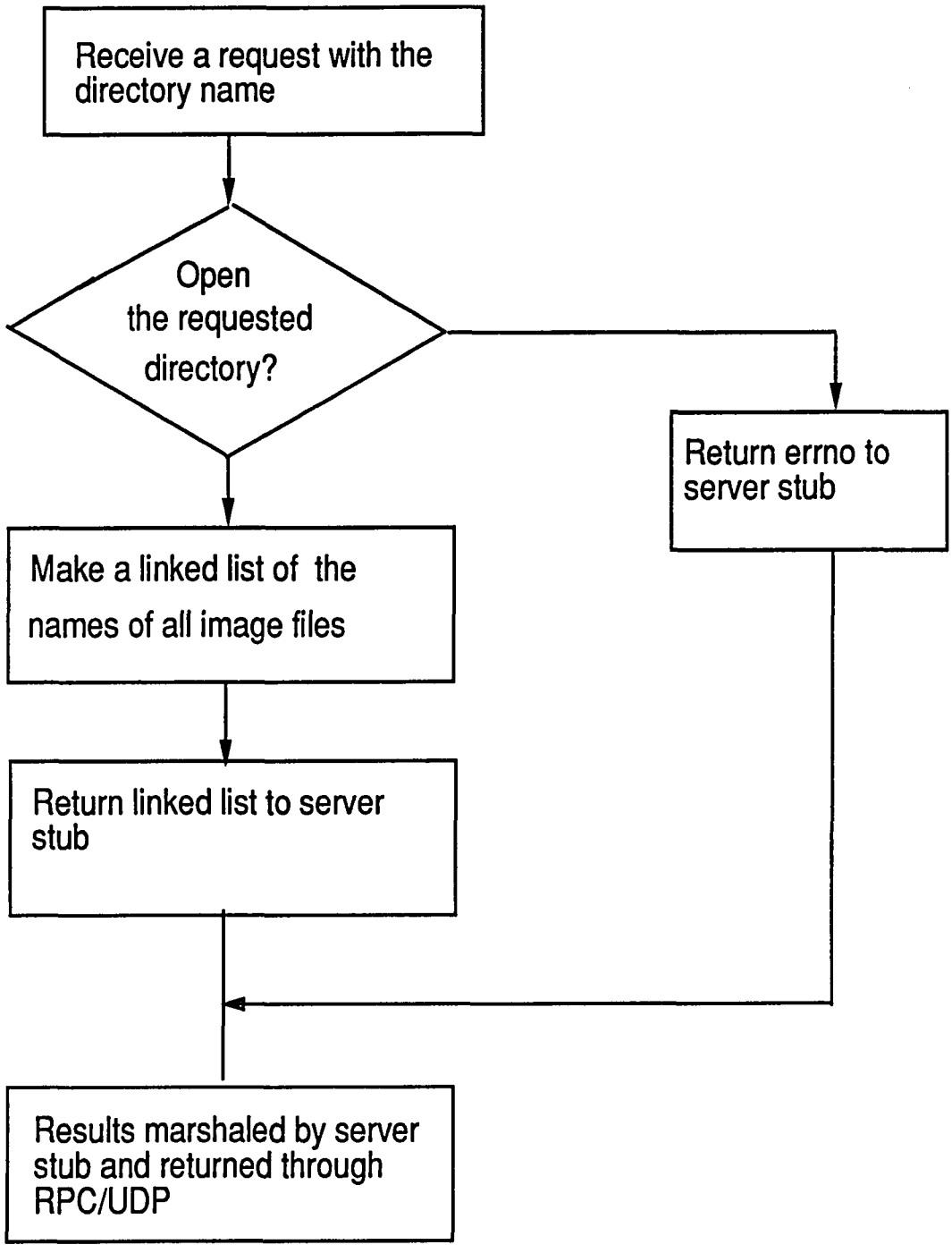

Figure 3.9: Directory List: Remote Server

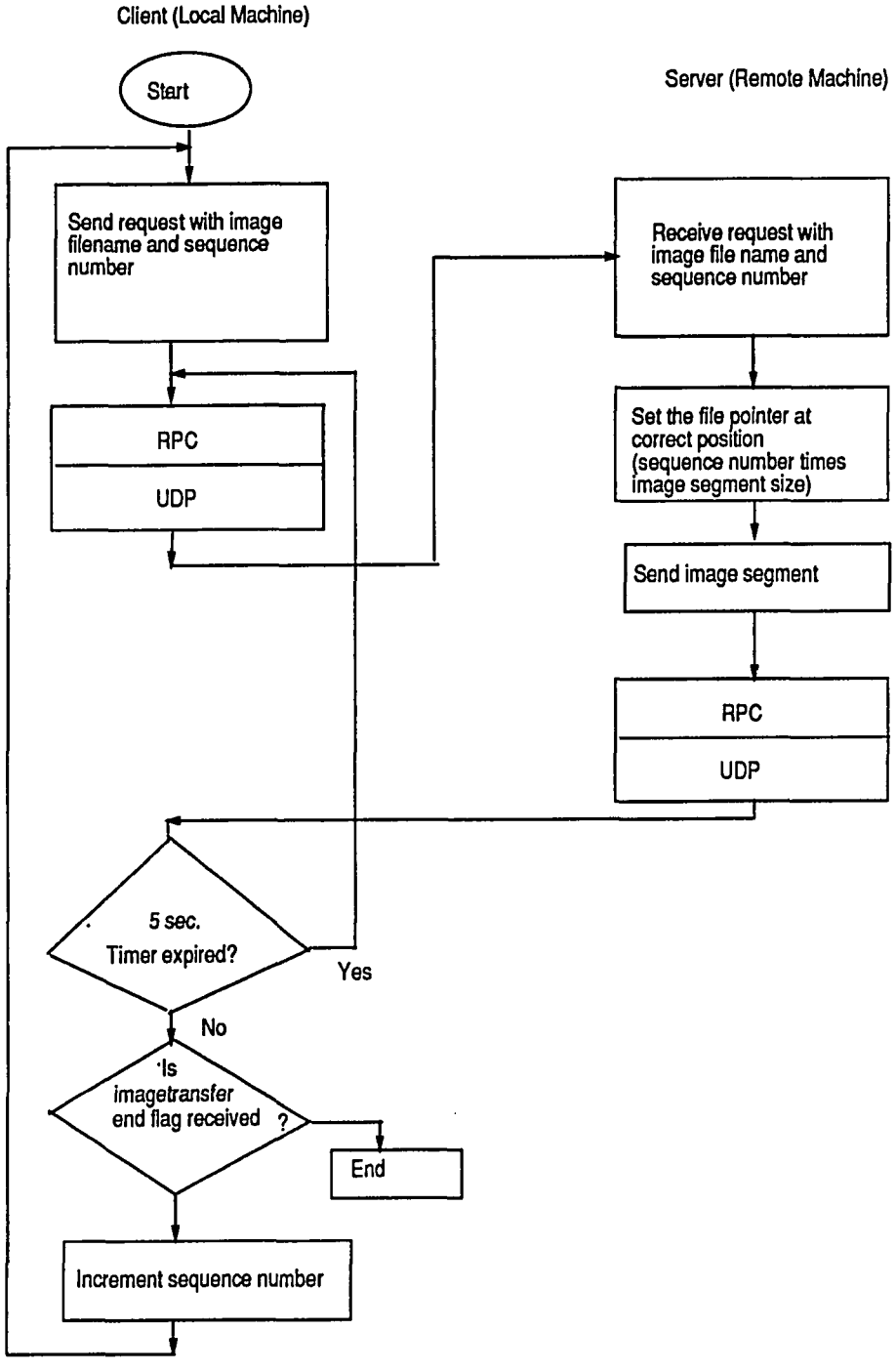

Figure 3.10: Retrieve Single Image: Error handling in Application layer

packets in RPC/UDP. To implement the error control at the application layer, the client keeps track of the file pointer from which image segments are retrieved. The scheme is shown in Figure 3.10. The client will send a request with image file name and sequence number as parameters in a remote procedure call. The server will open the image file at the remote machine. The file pointer will be set at the position, offset by sequence number times image segment size, from the beginning of the file. If the client does not receive the image segment in less than retransmission time, it will again request the server with same sequence number. So, the server reads the same segment of the image file. In the case of correct transfer of the segment, the sequence number is increased by one at the client side and sent to the server. So, the server will read the next segment of the image. The retransmission counter is kept equal to 5 seconds and if the client does not receive the correct data in five consecutive requests, RPC is timed out. When the client receives the imagefile transfer end flag set, it will exit. The imagefile transfer end flag is set at the remote server when the last segment of imagefile is read and returned.

In an Internet, the path between a pair of machine may have a single high speed network, or it may have multiple intermediate networks through multiple gateways. So, theoretically it is not possible to know a priori how quickly acknowledgements will return to the source. The delay at each gateway depends on traffic, so the time to transmit the data and receive the an acknowledgement varies considerably from one instant to another. So, the experiments were conducted to find out the

distribution of round trip time for IP datagrams across Internet. The experimental results show that for 90 percent of the packets the round trip time was less than equal to 5 seconds[15]. So, keeping the retransmission timer greater than 5 seconds will deteriorate the performance without any significant improvement in the reliability. At the same time keeping the timer value less than 5 seconds will not provide the correct results. So, the retransmission timer value is kept equal to 5 seconds based on the experimental test results.

The client and server stub take care of network interfacing and hide underlying network details from the application. Client and server stubs bind a socket and enable asynchronous I/O. In UNIX asynchronous events are notified to a process by means of signal. Signals are called "software interrupts". Signals usually take place asynchronously. Signals can be sent by one process to another process or by the kernel to a process.

A process specifies how it wants a signal handled by calling the signal system call.

```
#include <signal.h>
int (*signal (int sig, void (*func)(int))(int);
```

Signal is a function that returns a pointer to a function that returns an integer. The *func* argument specifies the address of a function that does not return anything. There are two special values for the *func* argument: SIG_DFL to specify that the signal is to be handled in the default way, and SIG_IGN to specify that the signal is

to be ignored. The *signal* system call always returns the previous value of *func* for the specified signal.

A process can deal with the signal in three ways.

1. A process can provide a function that is called whenever a specific type of signal occurs. This function is called a *signal handler*, can do whatever the process wants to handle the condition. This is called the *catching* the signal.

2. A process can choose to ignore a signal. All signals, other than SIGKILL, can be ignored. The SIGKILL is a guaranteed way of terminating any process.

3. A process can allow the default to happen. Normally a process is terminated on receipt of a signal. But default action of certain signals like SIGURG,SIGCONT,SIGIO, and SIGWINCH is to be ignored while SIGSTOP, SIGTSTP, SIGTTIN and SIGTTOU signals is to stop the process.

Whenever data is received by local kernel from network SIGIO signal is generated. The stub routines enables processid to receive the the SIGIO signal and enable the asynchronous I/O. The retransmission is taken care by using *alarm* system call with argument equal to retransmission timer value.

```
unsigned int alarm(sec);
unsigned int sec;
```

The *sec* argument specifies the number of seconds to elapse before the kernel is to send the process a SIGALRM signal. The argument specifies the "wall clock time",

not the CPU time. If the argument is zero, any previous alarm clock for the process is cancelled.

The properties of the already open socket is changed by using *fcntl* system call.

```
#include <fcntl.h>
int fcntl(filedes,cmd,arg);
int filedes;
int cmd;
int arg;
```

The *cmd* argument specifies the action to be taken by *fcntl.* In the stub software design F_SETOWN argument is used to set the processid to receive the SIGIO signal and F_SETFL is used with FASYNC and FNDELAY for enabling asynchronous and non-blocking I/O.

The signal handling in stub software is illustrated below.

```
/* prototype of signal handler */

void receive_data();
void send_data();

/*set the processid to receive the SIGIO signal */
```

```
if(fcntl(sockfd,F_SETOWN,getpid())<0)
{
perror("F_SETOWN error");
exit(-1);
}

/* Enable the asynchronous I/O */

if(fcntl(sockfd,F_SETFL,FASYNC|FNDELAY)<0)
{
perror("FASYNC and FNDELAY F_SETFL error");
exit(-1)1;
}

/* sleep until SIGIO or SIGALRM occur.
   Catch SIGIO and SIGALRM and invoke signal handler */
while(1)
{
if(signal(SIGALRM,send_data)==SIG_ERR)
{
perror("error in SIGALRM");
```

```
exit(-1);
}
if(signal(SIGIO,receive_data)==SIG_ERR)
{
perror("error in SIGIO");
exit(-1);
}
if(alarm(25)<0)
{
perror("error in setting alarm");
exit(-1);
}
sigpause(0);
}
```

3.2.1 UNIX Interface To UDP/IP

The networking software was designed using User Datagram Protocol at transport layer and Sun's RPC facility. To establish communication link between two processes on different host, the association 5-tuple must be established. The term association is used for the 5-tuple that completly specifies the two processes that make up a connection:

```
{protocol,local-address,local-process,foreign-address,foreign-process}
```

The half-association as either {protocol,local-address,local-process} or {protocol,foreign-address,foreign-process} is called a socket. In Berkeley UNIX socket is the programmer's interface to communication protocol. Sockets differ from file-descriptors in the way that, they are created and bound later if desired while the file descriptor is bound when created.

Sockets are created using socket system call.

```
#include<sys/types.h>
#include<sys/socket.h>
int socket(int family,int type,int protocol)
```

Internet protocols are specified by AF_INET. The socket type is one of the following:

1. Stream socket.
2. Datagram socket.
3. Raw socket.
4. Sequenced packet socket.

For UDP/IP datagram socket type is used. Datagram sockets offer bidirectional communication with no guarantee of reliable delivery. Application level program must take care of lost,duplicated and out-of-sequence packets. Before binding the socket following structures in <netinet/in.h> must be filled.

```
short sin_family; /*AF_INET */
short sin_port; /*16 bit port number */
struct in_addr sin_addr; /*32 bit netid/hostid */
char sin_zero[8]; /*unused */
```

Socket system calls for connectionless UDP protocol are shown in Figure 3.11. The *bind* system call assigns a name to an unnamed socket.

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(sockfd,myaddr,addrlen);
int sockfd,addrlen;
struct sockaddr *myaddr;
```

The third argument of *bind* system call is the size of the address structure and the second argument is a pointer to a protocol-specific address. There are three uses of *bind.*

1. Servers register their well-known addresses with the system. Both connection-oriented and connectionless servers have to bind their address before serving any client request. After making this call *local-address* and *local-process* elements of the association 5-tuple are filled.

2. A client can register a specific address for itself.

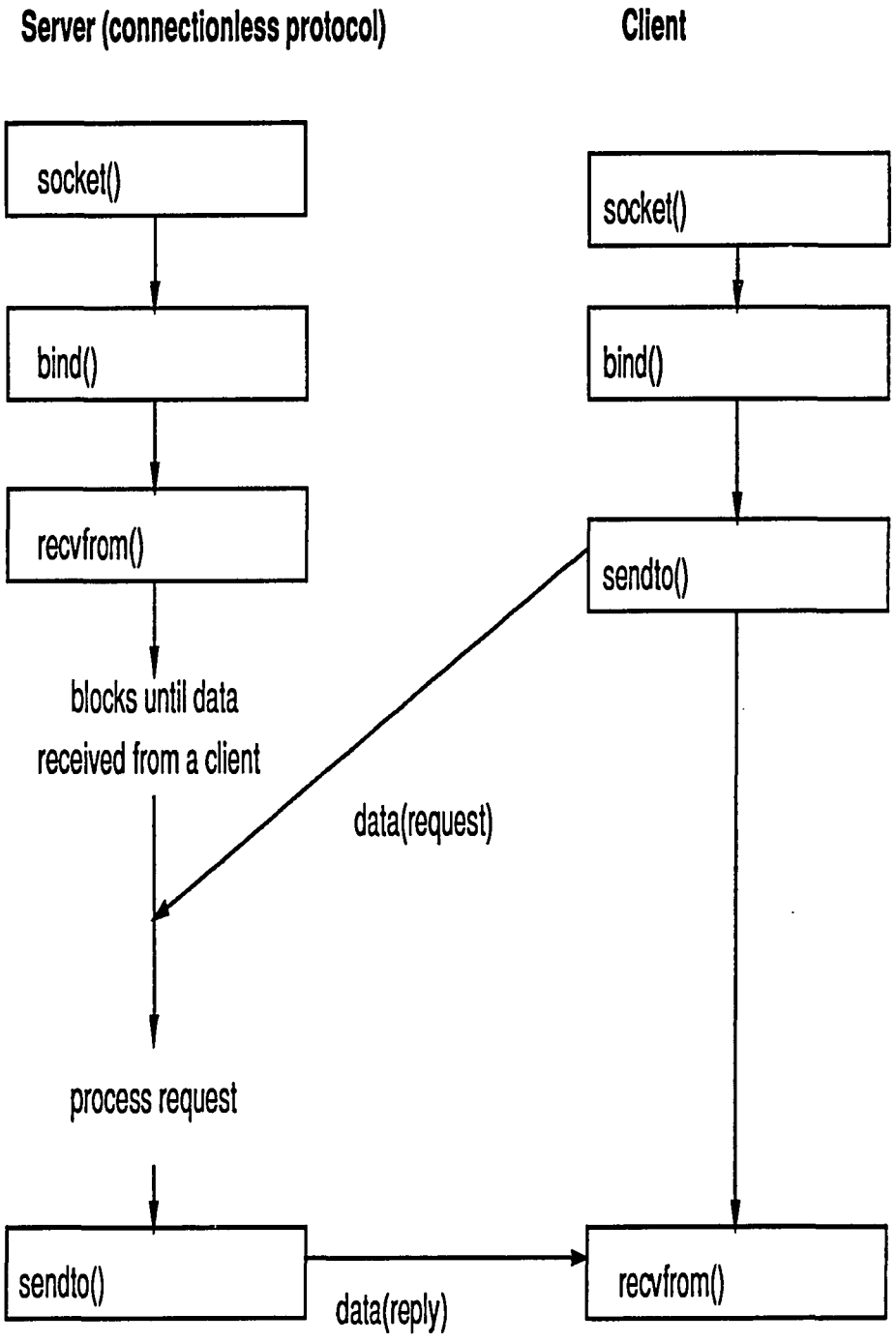

Figure 3.11: Socket System Calls For Connectionless Protocol

3. A connectionless client needs to assure that the system assigns it some unique address, so that the server has a valid return address to send its responses to.

The *sendto* and *recvfrom* system calls are used for sending and receiving the data without actually establishing the connection.

```
int sendto(sockfd,buff,nbytes,flags,to,addrlen);
int sockfd,nbytes,flags,addrlen;
char *buff;
struct sockaddr *to;

int recvfrom(sockfd,buff,nbytes,flags,from,addrlen);
int sockfd,nbytes,flags;
int *addrlen;
char *buff;
struct sockaddr *from;
```

The *flags* argument is either zero,or is formed by or'ing one of the following constants:

```
MSG_OOB send or receive out-of-band data
MSG_PEEK peek at incoming message(recv or recvfrom)
MSG_DONTROUTE   bypass routing(send or sendto)
```

The MSG_PEEK flag lets the caller look at the data that's available to be read, without having the system discard the data after the *recvfrom* returns. The *to* argument

for sendto specifies the protocol-specific address of where the data is to be sent. The address is protocol specific, its length must be specified by *addrlen.* The *recvfrom* system call fills in the protocol-specific address of who sent the data into *from.* The length of this address is also returned to the caller in *addrlen.* The final argumentof *sendto* is an integer value, while the final argument to *recvfrom* is a pointer to an integer value, which is a value-result argument. For details refer [17][19]. The image file is opened using *open* system call.

```
#include <fcntl.h>
int open(pathname,oflag,mode)
char *pathname;
int oflag;
int mode;
```

It returns a file descriptor if successful,otherwise -1 is returned.

The data is written into the file using *write* system call.

```
int write(filedes,buff,nbytes)
int filedes;
char *buff;
unsigned int nbytes;
```

The actual number of bytes written is returned by the system call.

Data is read from the open file using *read* system call.

```
int read(fildes,buff,nbytes)
int fildes;
char *buff;
unsigned int nbytes;
```

If read is successful, the number of bytes read is returned. If an error occurs in *write* or *read*, -1 is returned.

3.3 Transport Protocol Selection

The comparison of RPC/TCP and RPC/UDP is presented in this section from the view point of image transfer. The comparison was performed to determine any performance benefits of RPC/UDP, if any, over RPC/TCP for image transfer service. Figure 3.12 describes the image transfer with RPC/UDP, with error handling in Application layer. The details of it is described in section 3.2. Figure 3.13 describes the image transfer with RPC/TCP. TCP is a reliable packet delivery protocol and takes care of error handling. The software designer can assume the reliable transport when TCP is used as the transport protocol.

3.3.1 Comparison of RPC/TCP and RPC/UDP

The previous implementation of ImageNet was based on RPC/TCP. In this document RPC/UDP is evaluated against RPC/TCP. The comparison between them is as follows:

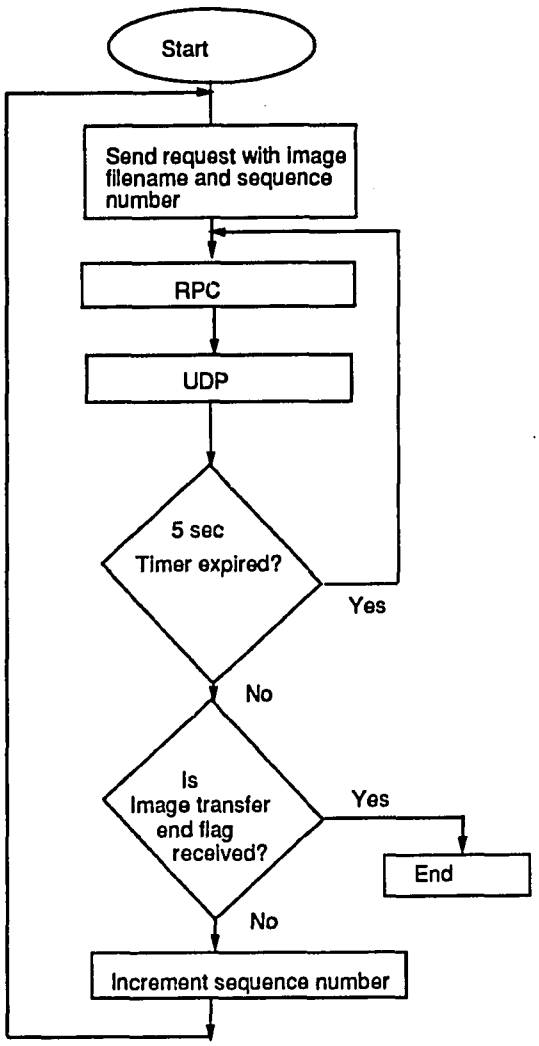

Figure 3.12: Image transfer with RPC/UDP

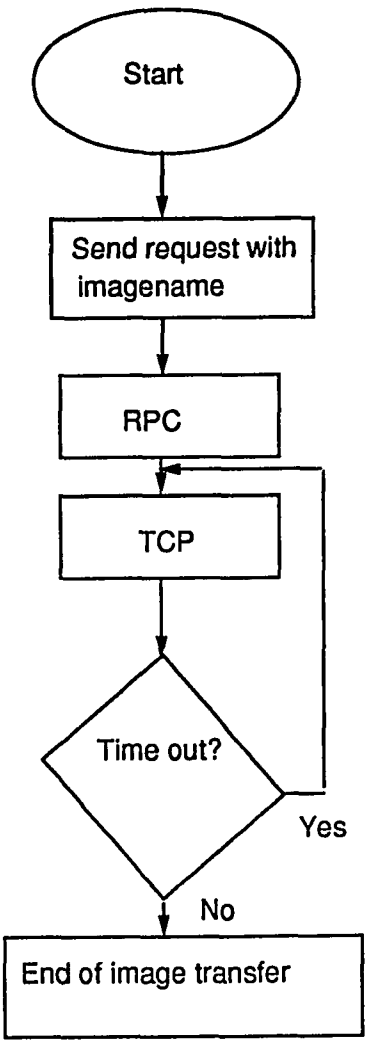

Figure 3.13: Image transfer with RPC/TCP

1. Only a maximum 1024 bytes can be passed by TCP when socket interface is used, while UDP has no maximum limit with socket interface but if Sun RPC is used on the top of it then it will restrict the argument size to 8 Kbytes.

2. While using TCP, a connection must be established before image transfer, where in the case of UDP there is no overhead for connection establishment.

3. The header size of UDP is 8 bytes while TCP header size is at least 20 bytes, so TCP has more protocol processing overhead.

4. TCP has positive acknowledgement, for each packet transmitted. While in UDP, no positive acknowledgements are provided.

5. TCP is a reliable packet delivery protocol while UDP is not. So, the software design in upper layers must take care of reliability issues when using UDP.

To evaluate RPC/UDP against RPC/TCP, experiments were conducted to retrieve images across Internet and local Ethernet at Electrical and Computer Engineering. The details of the experiments are as follows. The experiments were conducted in weekdays and weekends. There were three sets of the experiments,

1. Midnight to 8 a.m.: Test 1

2. 8 a.m. to 4 p.m.: Test 2

3. 4 p.m. to midnight.: Test 3

In each set of experiments the images were retrieved one hundred times to find out the average transfer time and percentage packet lost in RPC/UDP. The images were

retrieved from Wake Forest University, North Carolina to University of Arizona. The experiments were also conducted over Ethernet in the Electrical and Computer Engineering department.

Figure 3.14, 3.15, 3.16, 3.17, 3.18, 3.19 show the results of experiments. In the figures 3.14 to 3.18, time period 1 denotes time of Test 1, time period 2 denotes time of Test 2 and time period 3 denotes that of Test 3. There is no significant speed difference between RPC/UDP and RPC/TCP for image transfer across Internet. In case of RPC/UDP the packet loss was between 1 to 7 percent, thus requiring retransmission for lost packets. In the case of local Ethernet, RPC/UDP showed speed up but the packet loss was negligible.

The percent packet lost is calculated by dividing total packet lost by total packets transferred. The retransmission timer was set equal to 5 seconds, the justification for the value of retransmission counter is explained in section 3.2.

Percent Packet Lost = 100 * (Total Packet Lost)/(Total Packets Transferred)

Total Packets Transferred= Request Packets + Response Packets

Taking the example of earth.gif image, of 404 Kbytes size, the total response packets will be 404 divided by 8 because in case of UDP with Sun RPC, the argument size is restricted to 8 Kbytes due to Sun RPC. Each image was retrieved one hundred times in each experiment and there were 51*100*2=10,200 packets transferred. Since the experiment was repeated at six different times the total packets transferred will be 10200*6=61200. The total packets lost during these experiments were 1005, so

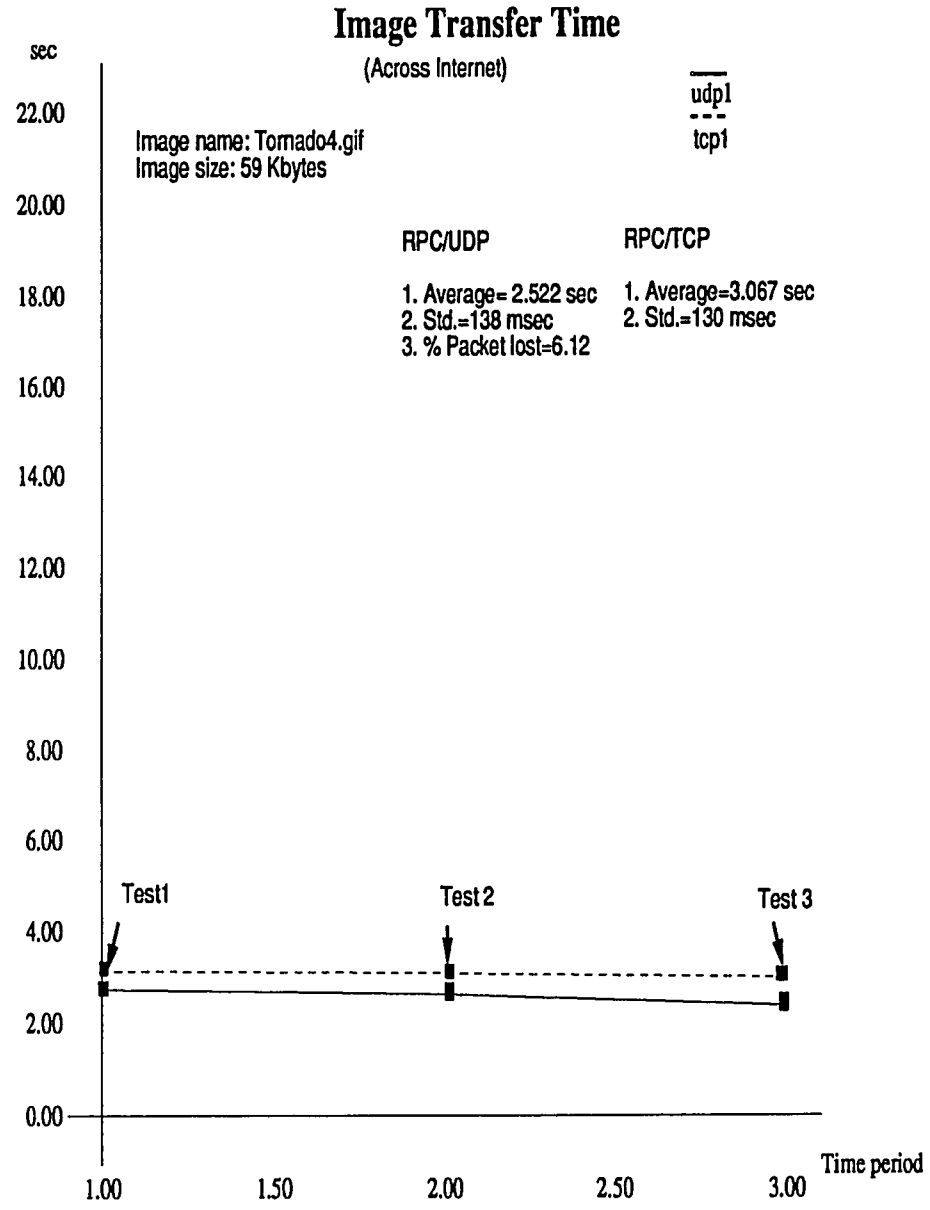

Figure 3.14: Transfer of image, Tornado4.gif: Experimental test results

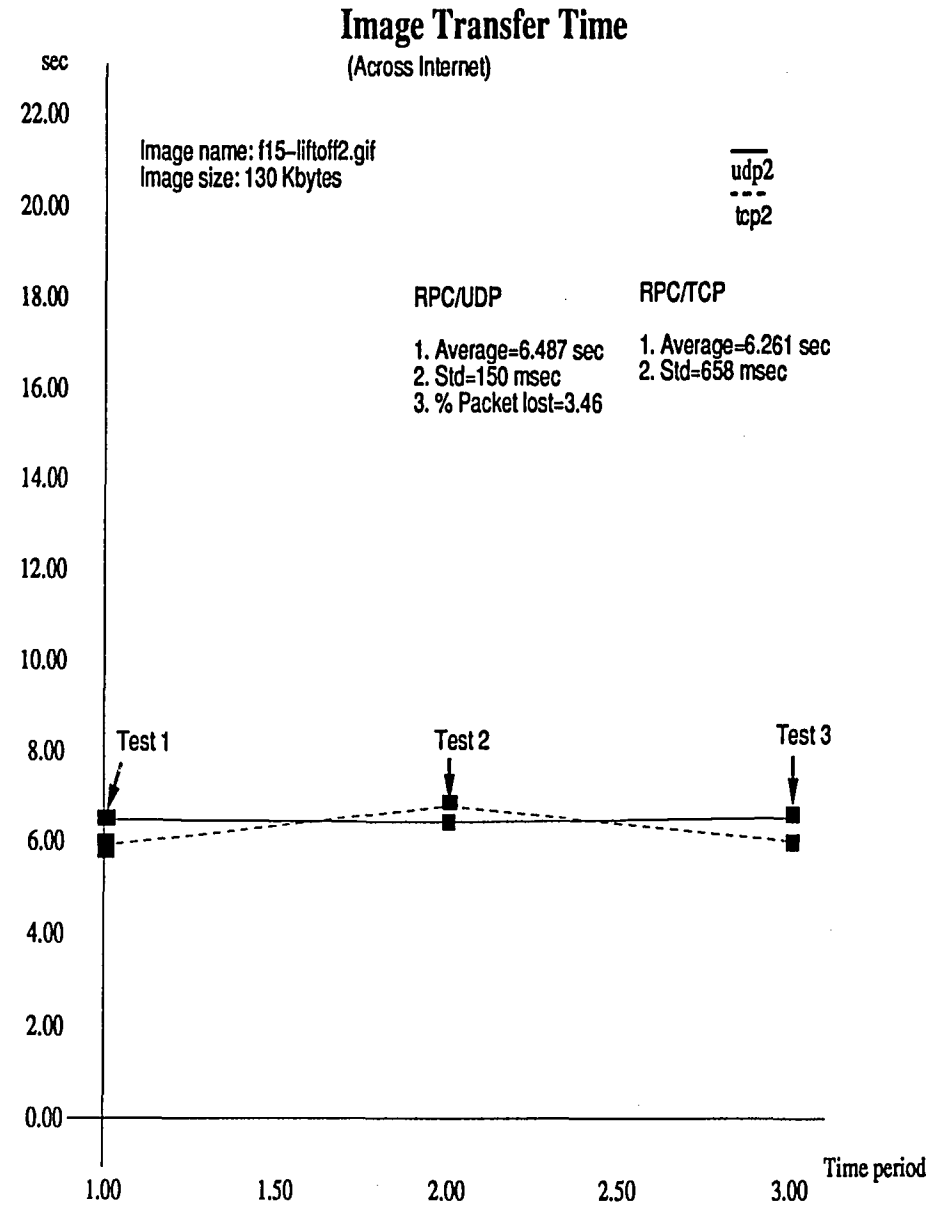

Figure 3.15: Transfer of image, f15-liftoff2.gif: Experimental test results

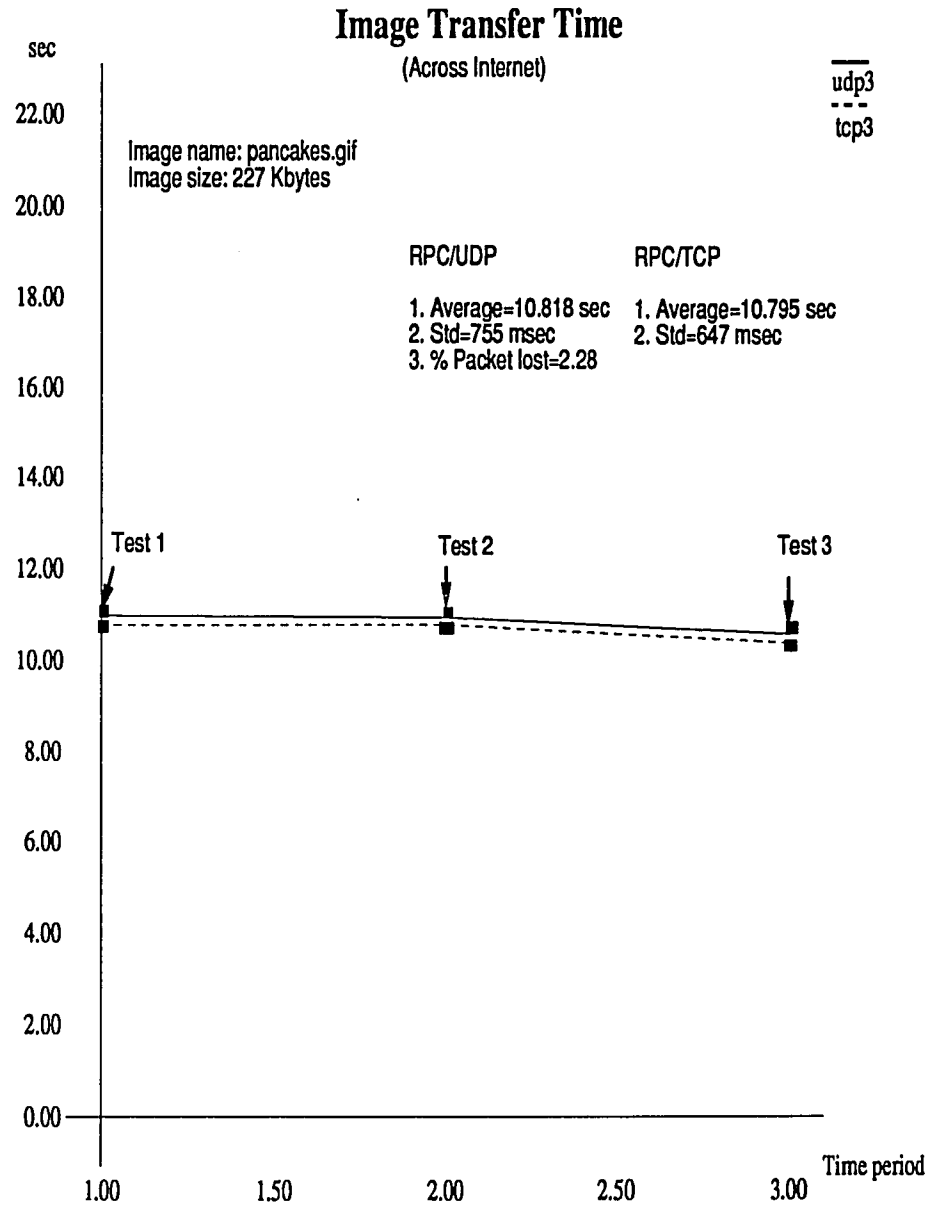

Figure 3.16: Transfer of image, Pancakes.gif: Experimental test results

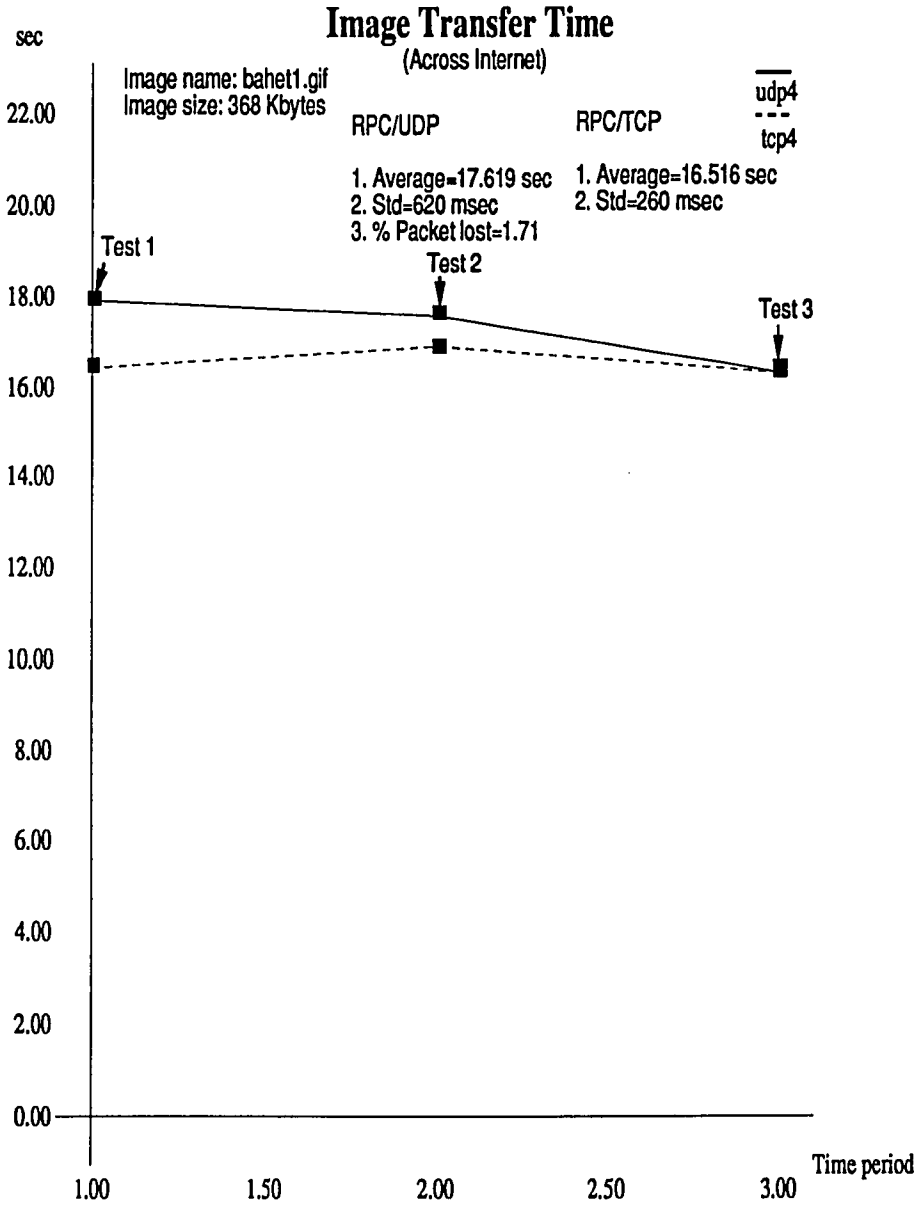

Figure 3.17: Transfer of image, Bahet1.gif: Experimental test results

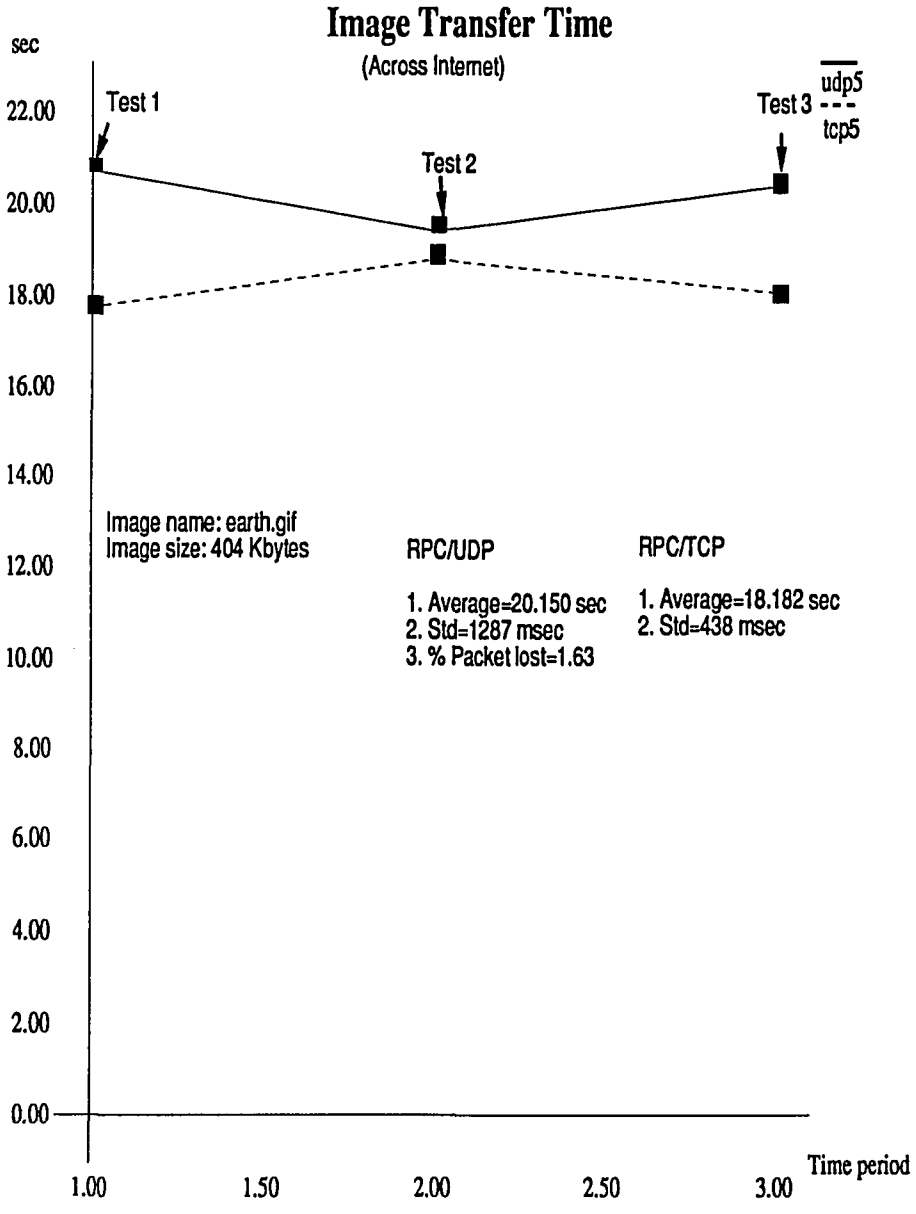

Figure 3.18: Transfer of image, Earth.gif: Experimental test results

dividing this number by total packets transferred 61200 will give percent packet lost equal to 1.63 percent.

Sun's RPC facility is used for this project which is inherently stop-and-wait protocol so speed up gain is not so high as using UDP with sliding-window protocol. Sun's RPC also restricts the argument size to 8Kbyte while using UDP. The time of image transfer is measured by system timer facility of UNIX. The timer is started when RPC is made to retrieve the image and it is stopped when the image is retrieved. Each image is retrieved hundred times and the average of retrieval time is taken. This experiment was carried out for TCP and UDP while using Sun's RPC facility. The experiment was conducted in the Computer Engineering Research Lab where the Sun Workstations are connected by Ethernet.

UDP does not take care of lost packets. So,while designing this application with RPC on top of UDP it has to be taken care of. The RPC is inherently a "stop and wait" protocol with unique transaction id in case of Sun's RPC facility. It greatly reduces the problem of lost or delayed packets. But the real problem is to read data from correct position from server side in case of lost packets because the file pointer will be advanced by the amount of packet size when the request is made again. In RPC, if the data is not retrieved within a time equal to retransmission counter time, a request is made again. So,at the time of repeated request the file pointer will be pointing to a location offset by number of bytes read from file last time. In order to take care of this, two arguments are passed from client side, image filename and

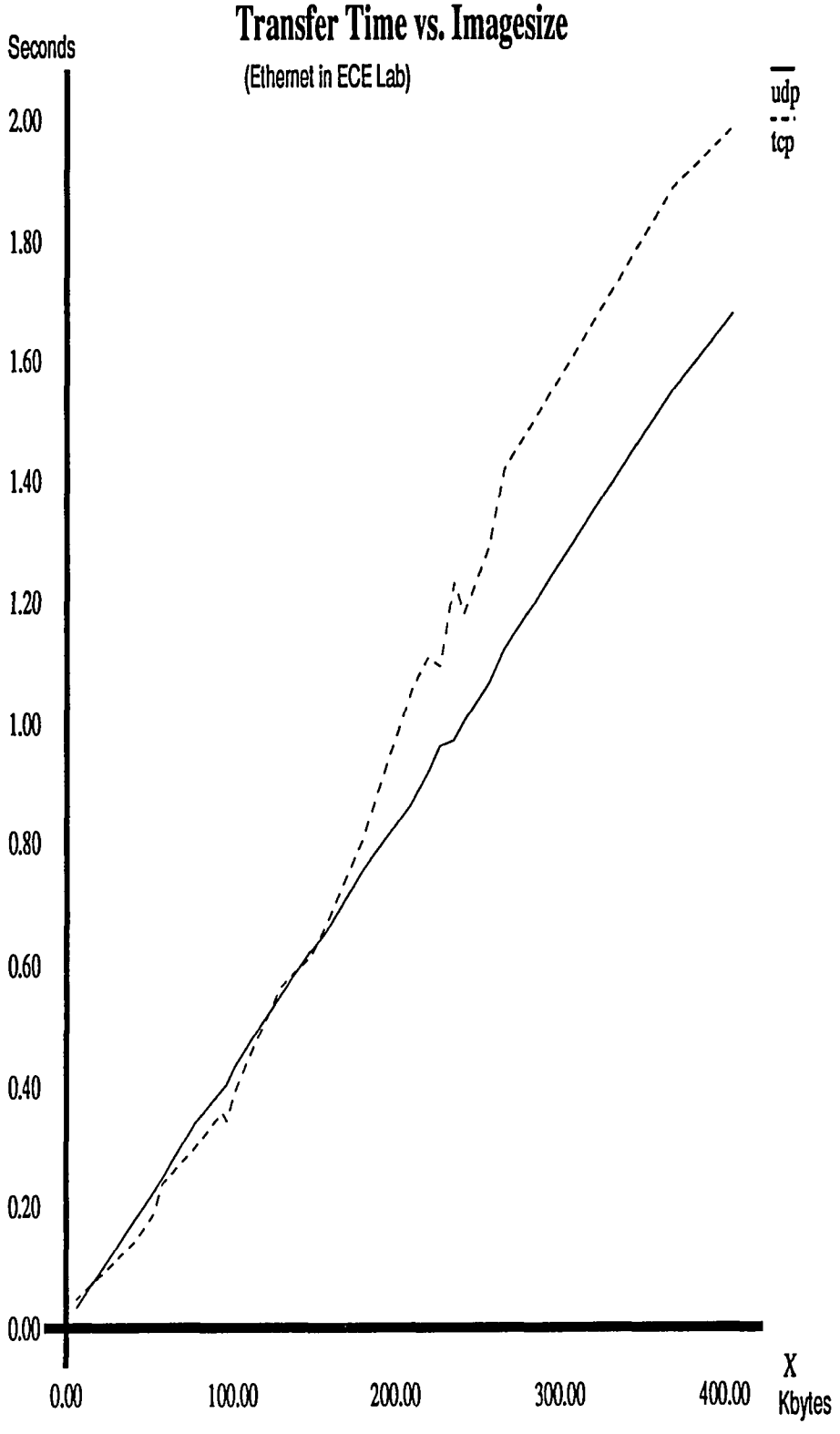

Figure 3.19: RPC/UDP vs. RPC/TCP:Local Ethernet

sequence number. So,the client at the local workstation keeps track of the file pointer position rather than server procedure. The sequence number is incremented each time a RPC is made, the RPC to get a new data is made from application only when data is retrieved from previous RPC. At the server side before reading data from file, the file pointer is set to the position offset by sequence number times the number of bytes read. So, in case of lost packets, timeout will take place and RPC is made with the same sequence number which correctly puts file pointer position. Thus, the problem of lost packets from remtote node to local workstation is taken care by the application design on top of UDP.

3.4 RPC Protocol Specification and Data Structures

Following is the protocol specification for remote image directory listing in the rpcgen language. It is compiled by the rpcgen compiler which produces the header and files in C language.

```
const MAXNAMELEN=255;
typedef string nametype<MAXNAMELEN>;
typedef struct namenode *namelist;
struct namenode{
nametype name;
namelist next;
};
```

```
union readdir_res switch(int errno) {
case 0:
namelist list;
default:
void;
};
program DIRPROG{
version DIRVERS{
readdir_res READDIR(nametype)=1;
}=1;
}=77;
```

Following is the protocol specification for image retrieval,

```
const MAXNAMELEN=30;
typedef opaque buffer<512>;
typedef string nametype<MAXNAMELEN>;
typedef struct namenode *namelist;
/*
A node of image
*/
struct imagedata{
buffer nbyte;
```

```
int end;
namelist next;
};
/* The result of imagetran operation */
union readdir_res switch(int errno) {
case 0:
namelist list; /* no error:return data */
default:
void;
};
/* Image transfer program definition */
program IMGPROG{
version IMGVERS{
readimg_res IMAGETRAN(nametype)=1;
}=1;
}=76;
```

Following is the structure which contains the image data or error number.

```
struct imagedata{
buffer nbyte;
int end;
namelist next;
```

```
};
struct readdir_res{
int errno;
union{
namelist list;
}readdir_res_u;
};
```

Host computers are normally known by human readable names. But the packets that are communicated between hosts use numeric addresses of some form to get to their destination,but human deal much better with names instead of numbers. To accomplish this, the function *gethostbyname* is used in the ImageNet networking software.

```
#include <netdb.h>
struct hostent *gethostbyname(char *hostname);
```

The *gethostbyname* function returns a pointer to a *hostent* structure that is defined in the <netdb.h> header file.

```
struct hostent{
char *h_name; /*official name of host */
char **h_aliases; /* alias list */
int h_addrtype; /* host address type */
```

```
int h_length; /* length of address */
char **h_addr_list; /* list of addresses from name server */
/* a null terminates the list */
};
#define h_addr h_addr_list[0]  /* first address in list */
```

This function is available only for Internet services. So,currently the *h_addrtype* field always contain AF_INET, and similarly the *h_length*field always contains 4(the length of an Internet address).

```
/* get the internet address of the host in dotted decimal string format by
   gethostbyname function */
struct in_addr *ptr;
struct hostent *hostptr;
if((hostptr=gethostbyname(server))==NULL)
{
perror("error in gethostbyname");
exit(-1);
}
if((ptr=(struct in_addr*)*hostptr->h_addr_list++)==NULL)
{
perror("error in getting pointer");
exit(-1);
```

```
}
```

Now,*inet_ntoa(*ptr)* gives the dotted-decimal internet address of the host.

The time measurements of image transfer using UDP and TCP were made by using *gettimeofday* system call. It returns the number of seconds since 00:00:00 GMT, January 1,1970. It also returns timezone information, which are not required for this application.

```
#include <sys/time.h>
int gettimeofday(tvalptr,tzoneptr)
struct timeval *tvalptr;
struct timezone *tzoneptr;

struct timeval{
long tv_sec; /* seconds since 00:00:00 GMT, Jan.1, 1970 */
long tv_usec; /* and microseconds */
};
```

It returns zero if all is OK, with the structure pointed to by the tvalptr filled in. Since zone information is not needed, tzoneptr is specified as NULL. The time when RPC is made to retrieve image is subtracted by time when image is retrieved in user-workstation. So, the difference between the two returned value of this call gives the image transfer time.

CHAPTER 4

SUMMARY AND CONCLUSION

4.1 Conclusion

The user-interface is designed in X-windows using Athena widget set and X-toolkit. It is primarily designed for users with no prior computer background. Cascaded pop-up menus are avoided from simplicity point of view. Figure 4.1 shows the main menu of user-interface. Dialog boxes are implemented to take input from user. The results retrieved over the network are displayed in separate window. The user can change the thumb by using the middle mouse button to view retrieved data which is not visible in the window. Help menu can be selected from the main menu. In case of any network errors or system errors, a separate error window is popped up and error message is displayed. The appearance of Athena widget is not so good as commercially popular OPENLOOK or MOTIF widget set but this software was not available during the time of the project. The literature and software for Athena widget is easily available from public domain in Internet sites which makes it popular in University environment. X-toolkit is used because it is simple, flexible yet a powerful tool for building application. It allows direct access to X-lib for the tasks

Figure 4.1: Main menu

which can not be implemented by it. The implementation is carried out in 'C' programming language which does not provide object-oriented programming features, but the X-toolkit is built on object-oriented philosophy. The design has a flavor of object-oriented programming, where widgets are considered objects.

Experimental comparison between RPC with User Datagram Protocol(UDP) and Transmission Control Protocol(TCP) were made for retrieving image. The experiments were conducted between University of Arizona and Wake Forest University, North Carolina across Internet and in Computer Engineering lab over Ethernet using RPC with UDP and TCP in transport layer. Each image was retrieved one hundred times to get an average at three different time periods in weekdays and week ends. There was no significant performance difference between RPC/UDP and RPC/TCP across Internet.

In case of RPC/UDP packet loss was between 1 to 7 percent and it must be taken care by upper layer software design. The reason is the most fundamental Internet service consists of an unreliable, best-effort, connectionless, packet delivery system. The service is called unreliable because delivery is not guaranteed. The packet may

be lost, duplicated, or delivered out of order, but the network layer(IP) will not do any error correction or reporting to the sender or receiver. A sequence of packets sent from one machine to another may travel over different paths, or some may be lost while others are delivered, the unreliability arises when resources are exhausted or underlying networks fail[15]. The reliability measures are taken care by TCP in the case of RPC/TCP at the transport layer. While in the case of RPC/UDP, UDP does not provide any reliability above transport layer. So, over the Internet where the incidence of packet loss is frequent, RPC/TCP will perform better in this respect compared to RPC/UDP where error handling is done at the session and application layer software design. This is one of the reason which makes no performance difference between RPC/UDP and RPC/TCP even there is no overhead of connection establishment, positive acknowledgements and less time for protocol processing in UDP. However, in the local Ethernet environment RPC/UDP showed speed up over RPC/TCP but the packet loss was negligible.

A sophisticated remote consultation program *wscrawl* is interfaced with the main menu. It allows more than two workstations to participate in remote consultation. Each user can be identified by the rubber-pointer having the node name with it. This program allows user to draw different shapes on image to select different portions. It also enables user to draw with different pen-thickness and colors. The user can also type on the image and erase it.

This system will have an application in teleradiology, telepathology and real-estate business. It enables the user to list and retrieve images stored in remote nodes. Once the image is in local disk, the user can use *wscrawl* for remote consultation.

4.2 System Constraint and Future Work

The software will run only on UNIX based machines. Sun's Remote Procedure Call facility restricts the size of argument to 8 Kbytes when UDP is used. This will slow down speed. Moreover, RPC is inherently a "stop and wait" protocol. If sliding window protocol is implemented on top of UDP, it will speed up image transfer considerably.

Currently, keyboard input events are not caught by the X-windows interface, for some tasks user may prefer keyboard. A list of possible DBNs to which a workstation can connect should be added to workstation software and in case of link failure or DBN failure, the workstation software should automatically search through the list until a working DBN is found or the end of list is reached.

REFERENCES

[1] Steven Ferner, Sandor Nagy, and Andries Van Dam, "An Experimental System for Creating and Presenting Interactive Graphical Documents" ACM Transaction. Graphics, vol. 2, no. 1, pp. 59-77, Jan. 1982.

[2] H.M. Gladney and P.E.Mantey, *"Essential Issues in the Design of Shared Document/Image Libraries"* SPIE vol. 1258 Image Communications and Workstations, 1990.

[3] Ralph Martinez and Mohammed Nematbakhsh,*Design and Performance Evaluation of a High Speed Fiber Optic Integrated Computer Network for Picture Archiving and Communications Systems"*SPIE Vol. 1093 Medical Imaging III: PACS System Design and Evaluation, 1989.

[4] Andrew S. Tanenbaum, *Computer Networks.* Englewood Cliffs, New Jersey: Prentice Hall, second ed., 1988.

[5] S.K.Chang,"Image Information Systems," *Proceedings of the IEEE, vol.73,pp.754-764, April 1985.*

[6] R.Popescu-Zeletin et al.,"A Global Architecture for Broadband Communications Systems: The BERKOM Approach," in *Proceedings of the workshop on the Future Trends of Distributed Computing Systems in the 1990s,*pp.366-373, IEEE Computer Society Press, 1988.

[7] A.Ghafoor et al.,"A Distributed Multimedia Database System," in *Proceedings of the Workshop on the Future Trends of Distributed Computing Systems in the 1990s,* pp.461-466, IEEE Computer Society Press, 1988.

[8] M.Hatzopoulos, "Distributed Aspects of Information Systems," in *Research into Networks and Distributed Applications(R.Speeth, ed.),pp. 1029-1049,* Elsevier Science Publishers B.V.,1988.

[9] J.Q.McQuillan, "Broadband Networks: The End of Distance?" *Data Communications,vol.19,pp.76-86, June 1990.*

[10] J.Lin and M.T.Liu, "DDLDN:A Distributed Double-Loop Data Network for Very Large On-line Distributed Databases," in *Proceedings of a Symposium on Reliability in Distributed Software and Database Systems*, pp. 83-88,1981.

[11] S.Ceri,B.Pernici, and G.Wiederhold, "Distributed Database Design Methodologies," *Proceedings of the IEEE,* vol.75, no. 5,pp.563-546,1987.

[12] Jacob Ziv and Abrahame Lempel, *"A Universal Algorithm for Sequential Data Compression"* IEEE Transactions on Information Theory, vol.IT-23, no.3, May 1977.

[13] Adrien Nye and Tim O'Reilly, *X Toolkit Intrinsics Programming Manual,* second edition, O'Reilly and Associates,Inc.

[14] Adrien Nye and Tim O'Reilly, *X Toolkit Intrinsics Reference Manual. second edition. O'Reilly and Associates,Inc.*

[15] Douglas Comer,*Internetworking with TCP/IP Principles, Protocols and Architecture.* Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[16] Sun Microsystems, Inc.,*Network Programming Guide, 1990.*

[17] P.Wang, *An Introduction to Berkeley UNIX.*Belmont, California: Wadsworth Publishing Company, 1989.

[18] Sun Microsystems,Inc., "XDR: External Data Representation Standard." Network Information Center Request for comments 1014, 1987.

[19] W.R.Stevens, *UNIX Network Programming.* Englewood Cliffs, New Jersey: Prentice Hall, 1990.

[20] Sun Microsystems, Inc., "RPC: Remote Procedure Call Protocol specification." Network Information Center Request for Comments 1057, 1988.

[21] C.D.Peterson, *Athena Widget Set- C Language Interface.*

[22] P.J.Asente and R.R.Swick, *X Window System Toolkit-The complete Programmer's Guide and Specification, Bedford,* MA:Digital press,1990.

[23] CompuServe Incorporated, Graphics Interchange Format, Programming Reference, July 1990.

[24] Steve Blackstock, LZW and GIF explained, June 1987.

[25] Paul Fisher, Brent Grover, Gerhard Braver, and Gordon Ritchie, *Digital Image Display Station Performance Requirements Based on Physician Experience with a Prototype System"* SPIE Vol.1091 Medical Imaging III: Image Capture and Display, 1989.

[26] R.Castanet and X. Vavarro, "Networks for Image Transmission," in *Research into Networks and Distributed Applications*(R.Speeth, ed.),pp.649-652, Elsevier Science Publishers B.V., 1988.

[27] C.Malamud, "Share the Wealth: RPCs Help Programs Go Places,"*Data Communications*, vol.19, pp. 61-70, June 21 1990.

[28] Thomas D.C.Little and Arif Ghaffor, *"Network Considerations for Distributed Multimedia Object Composition and Communication"*, IEEE Network Magazine,November 1990.

[29] G.F.Yanbykh, *Choice of Hardware Configuration and Communications Link Throughputs in Computer Networks with Distributed Databases"*, Automatic Control and Computer Sciences, Vol. 23, No.2, 1989.