Remote Microscope for Inspection
of Integrated Circuits

by

James T. Kao

B.S. in Electrical Engineering, University of California at Berkeley (1993);
Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1995

© James T. Kao, 1995. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute copies of this thesis document in whole or in part.

Author

Department of Electrical Engineering and Computer Science
September 4, 1995

Certified by

Donald J. Troxel
Professor of Electrical Engineering
Thesis Supervisor

Certified by

Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

NOV 02 1995

ARCHIVES

LIBRARIES
Remote Microscope for Inspection
of Integrated Circuits

by

James T. Kao

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

The remote microscope was developed at MIT as part of the Computer Integrated Design and Manufacturing project to aid in the remote fabrication of integrated circuits, and will allow a user to operate and view in “real time” an actual microscope located at a distant facility. We envision a growing trend in the semiconductor processing field for more collaboration and sharing of resources, so it will become important for researchers to have access to a telemicroscopy system like the one developed at MIT in order to perform remote inspections of semiconductor wafers.

The MIT remote microscope is extremely versatile; it operates over the internet and allows a user to run the graphical microscope interface on any ordinary UNIX workstation, thereby providing easy access to the microscope for researchers located throughout the world. The actual remote microscope utilizes readily available hardware as well, making the entire system very straightforward and economical to implement. To the best of our knowledge, the MIT remote microscope is the first telemicroscopy system to operate on the internet. The remote microscope also is original in that it allows multiple users to view the microscope at the same time in a conference inspection, which should be very useful when geographically distant researchers wish to collaboratively inspect a wafer. The overwhelming response to the remote microscope has been very positive, as it was officially demonstrated in August 1995 between MIT and Stanford during the annual TCAD symposium.

Thesis Supervisor: Donald E. Troxel
Title: Professor of Electrical Engineering
Acknowledgments

I received my B.S. in Electrical Engineering and Computer Science at the University of California at Berkeley in the Spring of 1993, and during the Fall semester, I came to MIT eager to begin my graduate studies. During my first two semesters, I was a teaching assistant for 6.111, “Digital Systems Laboratory,” where I first met Professor Troxel. The following year I was offered a Research Assistantship with Professor Troxel’s group, and was very happy to work for a professor who I felt was not only a well respected expert in the field, but also a very considerate and kind person. I am very thankful that I have had the opportunity to work with Professor Troxel; none of this work would have been possible without him.

I would also like to thank my fellow research group for helping me this past year. Without their support and advice, I would have had a difficult time finishing this project, and without their companionship, I would not have had such a fun time working in the CIDM group. I would like to thank John Carney for helping me understand his CHAT messaging tools, which were used heavily in my project. William Moyne, with his encyclopedic knowledge of software, was also extremely helpful in answering my numerous questions throughout the year, and Jimmy Kwon was helpful in providing feedback and suggestions to my research. I’d also like to thank Mike McIlrath for his useful suggestions, my officemate Thomas Lohman for providing a relaxing (and interesting) environment in which to work, and Myron Freeman for helping me to solve hardware resource problems that I’ve encountered.

My family has also provided much emotional support during my graduate studies, and was instrumental in helping me keep my work in perspective. My mother, father, and my brother and sisters Susan, Nancy, Cindy, and John all deserve credit for my achievements.
# Table of Contents

List of Figures .................................................................................................................. 11
List of Tables .................................................................................................................... 13
1 Overview ......................................................................................................................... 15
2 Background ..................................................................................................................... 21
   2.1 Existing Video-Microscopy Systems ................................................................. 22
      2.1.1 Medical Imaging ......................................................................................... 22
      2.1.2 Microscopy for Integrated Circuits ............................................................ 24
   2.2 Televisualization Over the Internet ....................................................................... 25
3 Remote Microscope Function ...................................................................................... 27
   3.1 General Operation ................................................................................................. 27
      3.2 Client User Interface ......................................................................................... 30
         3.2.1 Client’s Console Display Window ............................................................ 32
         3.2.2 Client’s Global Display Window ................................................................. 34
         3.2.3 Client’s ZOOM Display Window ................................................................. 35
   3.3 Client for Conference Inspection ....................................................................... 36
   3.4 Server Interface ...................................................................................................... 37
4 Practical Uses for Remote Microscope ........................................................................ 39
   4.1 Use During Remote Fabrication of Integrated Circuits ...................................... 39
   4.2 Use During Local Fabrication of Integrated Circuits ......................................... 41
   4.3 Remote Microscope for Data Collection ............................................................. 41
   4.4 Remote Microscope Applied to Fields Outside of IC Fabrication ....................... 42
5 Hardware Implementation ............................................................................................ 43
   5.1 Targa+ Image Capture Board .............................................................................. 44
   5.2 Live Video Monitor ............................................................................................... 44
   5.3 Video Camera ........................................................................................................ 45
List of Figures

1. Remote microscope main console ................................................................. 17
2. Remote microscope GLOBAL window .......................................................... 18
3. Remote microscope ZOOM window .............................................................. 19
4. A possible remote microscope setup ............................................................ 29
5. Initial connection window .............................................................................. 30
6. “Console”, “Global”, “Zoom” windows that make up client interface ............ 31
7. Flow chart showing inspection steps during mask alignment ....................... 40
8. Component setup for remote microscope ...................................................... 46
9. Setup with PC acting as microscope controller and server .......................... 50
10. Setup with UNIX Server, UNIX Client, and OS/2 PC as microscope controller 51
11. Software components showing communication channels .......................... 55
12. Diagram showing messaging required after a “GRABFRAME” request ........ 61
13. Data structure used to record current users of the remote microscope .......... 64
14. Model for SERVER execution ...................................................................... 67
15. Model of SERVER execution after “GRABFRAME” message ....................... 70
16. Model of SERVER execution after “CURPOS” or “RECTANGLEPOS” .......... 71
17. Model of SERVER execution after “TAKECONTROL” message ................... 72
18. Model of SERVER execution after “CEDECONTROL” message ................... 73
19. Model of capture utility execution ................................................................. 74
20. Model of DAEMON event handling ............................................................. 76
21. Model of DAEMON handling incoming messages from Tcl/Tk Client .......... 77
22. Model of DAEMON handling incoming requests from Server .................... 78
23. Model of Tcl/Tk handling incoming requests from the daemon ................... 82
24. Model of Tcl/Tk event handling of user console button inputs .................... 85
25. Model of Tcl/Tk event handling of radiobutton or keyboard entries ............ 87
26. Model of Tcl/Tk event handling of mouse operations ................................. 88
27. Excerpt of code setting up mouse bindings over Global image..................89
28. Prototype for put_cursor and put_rectangle Tcl/Tk client routines...........90
29. Transformation between pixel and global coordinates..........................92
30. Model of Tcl/Tk Client handling incoming messages from daemon ..........94
31. Code excerpt of Tcl/Tk execution after “GRABFRAMEREPLY” message......96
32. State transition diagram showing control state of a client.....................99
33. State transition diagram showing the server state ................................100
List of Tables

1. Approximate cost of remote microscope hardware ................................................. 43
2. Message formats used between Tcl/Tk Client and Daemon ........................................ 58
3. Message formats between Daemon and PC Server .................................................... 59
4. Package of tools to manipulate data structure of current users ..................................... 65
5. Server global structures to maintain microscope state .................................................. 66
6. Messaging involved with events described in Fig. 21 .................................................... 77
7. Messaging involved with Server/Daemon events described in Fig. 22 ......................... 79
8. Messaging involved with making server connection events shown in Fig. 23 .......... 83
9. Tcl/Tk Client listing of Global variables........................................................................ 84
10. Messaging resulting from console button inputs shown in Fig 24 ............................... 86
11. Messaging resulting from mouse click over images shown in Fig 26 ....................... 88
12. Messaging resulting from incoming messages from daemon shown in Fig 30 ......... 95
13. Server package........................................................................................................ 110
14. Image capture package ............................................................................................ 111
15. Tcl/Tk client interface package ................................................................................ 111
16. Client daemon package ........................................................................................... 112
17. Remote microscope related equipment ...................................................................... 113
Chapter 1

Overview

The MIT remote microscope allows a user to remotely control and view a distant microscope over the internet, and was designed primarily to provide remote inspection capabilities for semiconductor researchers. Although the remote microscope was designed to be used in the semiconductor field as an inspection tool during remote fabrication, the remote microscope is actually a general purpose tool that will allow any user with access to an X-windows UNIX workstation to view any type of specimen under the microscope.

The remote microscope is extremely versatile; it operates over the internet and consists of a graphical client interface running on any UNIX based X terminal that communicates with a server process running on a personal computer connected to an inspection microscope. The PC contains a framegrabber board which is capable of capturing a video signal from a CCD camera that is mounted on the top of the microscope. To a user though, the remote microscope is simply the graphical client interface that runs on a workstation, because the exact hardware implementation details are hidden. Typically one operates the remote microscope by selecting a magnification setting and the X-Y pan positions, and then requesting a new image from the microscope. When the server process receives a new request, the PC will then grab a new image at 640 x 480 x 16 bit\(^1\) from the

\(^1\) Actually the colors are quantized to 200 before transmission, see 7.3.1
video camera after the microscope is positioned properly, and transmit the data back to the client to be displayed.

The software and hardware to implement a complete microscope system are readily available, making the remote microscope cost effective and accessible for most researchers. The basic hardware needed to implement the remote microscope include a PC running the OS/2 Warp operating system, a framegrabber board, and video camera mounted on an inspection microscope. The client/server software was also written at MIT, and is easily exportable to other systems. As a result, the remote microscope is a highly portable, low cost solution to providing remote inspection capability to researchers.

The remote microscope has the added capability of allowing multiple clients to view the microscope at the same time during a conference inspection mode. This allows any number of individuals anywhere on the internet to simultaneously view the microscope images collaboratively, which can be very productive when semiconductor researchers work together during processing steps.

To the best of our knowledge, the MIT remote microscope was the first instance of remotely viewing a microscope over the internet. The remote microscope was officially demonstrated between Stanford University and MIT, where users at Stanford could see and manipulate in real time a wafer located under a microscope at MIT. This was demonstrated at the August 1995 Stanford TCAD Symposium, “Hierarchical Technology CAD - Process, Device, and Circuits,” where the MIT-Stanford team described “A New Research Paradigm using Internet Collaboration”1 that would allow distributed researchers to work together in the semiconductor field. Remote fabrication and remote inspection of integrated circuits played an important role in this future vision of semiconductor manufacturing, and conference attendees in general were very excited about the remote microscope’s capabilities.

The user of the remote microscope is presented with a very natural interface to the microscope that provides two views of the microscope image: a global panoramic view of the wafer at a low magnification, and a zoomed view at a higher magnification of a partic-

1. Demo talk available online at http://aramis.stanford.edu/comproto/nre/tcad95.html
ular region of interest. The user is allowed to set the magnification setting of the microscope, to select a new pan position by either entering coordinates or clicking on a region within the global window, and finally to initiate a capture where an image from the microscope is displayed into a selected window. The following diagrams are screen dumps of the microscope images that were displayed during the MIT-Stanford demonstration in August 95.

**FIGURE 1.** Remote microscope main console.
FIGURE 2. Remote microscope GLOBAL window. This shows panoramic view at 50X.
FIGURE 3. Remote microscope ZOOM window. This shows area of interest at 200X.
Chapter 2

Background

The driving force behind developing the remote microscope here at MIT was to provide an inspection tool to aid in the remote fabrication of integrated circuits, which hopefully will improve the efficiency and flexibility of current semiconductor manufacturing technologies. By having the capability to perform different wafer processing steps at different facilities, manufacturing can be much more cost effective and robust. For example, expensive machines can be shared, rather than duplicated, among several different manufacturers and wafers can be rerouted to different facilities if a machine goes down in the production line. Currently, the process of remote fabrication is discontinuous and unorganized; there is no clean way to exchange data or provide feedback between two facilities. However, if the fabrication process is viewed at a higher level of abstraction, as in MIT’s CAFE (Computer Aided Fabrication Environment), remote fabrication can appear “hidden” from the user, since he/she will be using CAFE to write PFR’s that will automatically keep track of machine locations and record results from each fabrication step [McI92, Bon91]. In order to provide the capability for remote fabrication with CAFE, two closely related projects were pursued: one to modify the existing CAFE program to allow remote fabrication steps [Kwo95] and the second to develop a remote microscope for inspection of wafers at the back up facility.
Although ideally, the process designer could simply write a PFR and then mechanically perform machine operations, in actuality he/she must play an interactive role in maintaining quality by inspecting the wafer after each sensitive processing step.[FT93] He/she can then make decisions whether to continue with the next step, repeat previous ones, make alterations to the recipe, or make machine adjustments, etc. During remote fabrication, a similar process would be desirable where the primary process designer, who has the best understanding of a wafer's particular manufacturing tolerances, can inspect wafers at the back up facility and make process decisions to be carried out remotely. We believe the remote microscope may play a vital role in helping to provide this kind of feedback between process designer and remote facility during collaborative semiconductor fabrication in the future.

2.1 Existing Video-Microscopy Systems

Many systems today already involve using computers and/or video monitors to view and store microscope images, but none (of which we are aware) are geared towards allowing a user to remotely view and control a microscope effectively over the internet. In medical imaging, remote microscopy has been performed, but at the expense of extremely costly hardware and dedicated communication links like personal microwave channels. In the field of semiconductor processing, remote microscopy has generally been ignored until MIT's development of the remote microscope.

We believe the MIT remote microscope can play a very important role in future remote processing of semiconductors because it is versatile, low cost, and uses readily available hardware, especially since the client can be run on any UNIX X windows system connected to the internet. With the tremendous growth in interest over the internet, we feel that the remote microscope can be extremely accessible by many researchers.

2.1.1 Medical Imaging

The medical community has been very active in developing highly sophisticated remote imaging systems that allow doctors to receive video images and make remote diagnosis. In recent years, there has been a strong interest in telepathology (diagnosis of disease by viewing a specimen on a remote video monitor rather than through a microscope
directly) [WBRa87] and teleradiology (diagnosis of radiologic films remotely). In fact, several demonstration units have already been tested in hospitals, and have been shown to be almost as effective as traditional diagnosis.

Teleradiology deals with radiology at a distance, which involves viewing and diagnosing radiographs such as MR, CT, or X-ray etc. remotely. These static images need to be digitized and transmitted over a network (usually point to point ISDN leased lines). A teleradiation assessment published in October 1990 estimated the cost of a consultation site to be over $300,000 while a peripheral site was estimated to cost almost $200,000, not including maintenance costs on the order or $30,000/year for both. Today of course, the costs of a teleradiology system is much lower, but still on the order of $100,000. The assessment also suggested that teleradiation systems must have extremely high resolution upwards of 1000x1000 pixels x 12 bits, which require very expensive digitizing scanners.[Bat90]

Telepathology as described by Weinstein is more difficult and complex than teleradiology. Telepathology involves diagnosing diseases remotely and relies heavily on microscopy. As a result, telepathology requires higher resolutions, remote control of a microscope (involving bidirectional control), and real time video transmission. In order to handle high bandwidth of live video, typical transmission mediums that have been used include point to point microwave linkages, satellite transmission, and fiberoptic networks. For example, the Mayo Clinic in rochester Minn. in 1986 dedicated a broadband satellite communications network for telemedicine [WBRb89].

Recently, a “proof of concept” project involving transmitting high resolution static digital images over phone lines for teleradiology and telepathology was performed between Cambridge, MA and Saudi Arabia, where pathologists and radiologists demonstrated global telemedicine [Gol94]. The system actually involved using voice-grade telephone lines to transmit images in order to reduce cost. Teleradiology was demonstrated by transmitting and digitizing radiographs (1664x2020x12 bits), while “static” telepathology was performed by transmitting individual images (1024x1024x24 bits) captured from a high resolution CCD camera mounted on a conventional microscope (using a propriety system). The digitized images took 2-5 minutes to transmit and were then displayed on
special diagnostic workstations (RSTAR DWS-2000). The results of the experiment showed that remote diagnosis could be performed successfully with existing technologies.

Although much research has been done with these effective (but prohibitively expensive) medical imaging systems, we feel that research into a less expensive internet based microscope system would still be very useful. The MIT remote microscope uses readily available commercial cameras, framegrabbers, and computers, which will make remote inspection much more affordable and practical to implement, and will also be interfaced to the internet, which is more accessible than are ISDN leased lines or dedicated satellite links. The limited resolution of the commercial products (whose capabilities are continually improving), while probably not good enough for certain telepathology needs, are acceptable for our goal of using the remote microscope to inspect integrated circuits. Finally, because our microscope will be controlled over the internet, it was straightforward to provide conference inspections and manipulate the images with ordinary workstation computers.

2.1.2 Microscopy for Integrated Circuits

Many video systems are also being used in integrated circuits fabrication environments where microscopic inspection is required. However, in the field of integrated circuits, unlike in medical imaging, little effort has been made in recent years to provide remote inspection of integrated circuits. For example, the MIT Integrated Circuits Laboratory currently has several inspection Nikon light microscopes with mounted video cameras (Hitachi VKC150) that are simply connected to viewing monitors and video printers. Integrating our remote microscope into this laboratory would make the inspection process much more versatile, providing remote access as well as easy access to images in electronic form. The Berkeley Microlab at the University of California at Berkeley currently has a setup where an SEM microscope (with a CCD camera mount) is connected to a RasterOps frame grabber board on a Sun Sparc1 workstation that is being used to store and view pictures in electronic form [Mic94]. The greatest use for this system has been to create electronic files that can be printed on an ordinary laser printer, which is more cost effective than using a video printer that requires special film. In another system located at Texas Instrument’s Semiconductor Process and Device Center, a video image from a
microscope in their lab is being sent through ordinary video cables to an external video monitor outside the clean room. The idea here was to allow designers to inspect the wafers without having to suit up and enter the clean room. However, this system is still limited because the user can only observe passively, and it is impractical for more distant types of inspection.

The remote microscope to be developed here at MIT's Computer Integrated Design and Manufacturing group is much more versatile than current inspection systems because it provides interactive tele-operation and inspection over the internet. Although the remote microscope is still a first generation system that still needs to be improved, the initial responses to this system have been very positive.

2.2 Televisualization Over the Internet

Recently there has been a growing number of tele-visualization systems that have been developed to control video cameras and pass video information over the internet, specifically the world wide web. However, current developments generally do not provide the level of interaction and feedback needed to implement a remote microscope system. For example, a wide variety of live video sources can be accessed using web browsers such as Mosaic or Netscape, but due to the nature of HTML files, there is limited interaction between client and server and a high overhead for communications. There are several HTML pages on the internet that provide "live" images such as the University of Cambridge's infamous "Coffee Pot" or University of Washington's "Seattle Skyline". Generally, these video images are frame grabbed every few minutes and stored to a file which is linked to an HTML page that the user is viewing on his browser. By reloading the page, or clicking on the required hyper link, the user can get "real time" images from the video camera. More sophisticated HTML implementations do exist such as NYU's Labcam and the University of Chicago Computer Science Department's Labcam, which accept

1. For more information, contact Harold H. Hosack, Texas Instruments, Director of Process Synthesis, Semiconductor Process and Device Center R&D/Technology.
4. http://found.cs.nyu.edu/cgi-bin/rsw/labcam1
“posts” from the client so that the user has limited control of the remote camera. New versions of Netscape are also beginning to provide limited “server push” capability so that the server may directly communicate with the client. Although using the world wide web to pass images is straightforward and simple, the overhead and limited interaction between client and server makes this an unsuitable basis for the remote microscope. The remote microscope needed to have a “session” control where the client and server could maintain a direct connection so that the server could actively update client members (especially during conference inspections). By developing our own software to implement the remote microscope, we also had the flexibility to design a custom interface to allow us to utilize various pointing visual aids as well as to provide easy point-and-click buttons to control the microscope.

5. http://vision.uchicago.edu/cgi-bin/labcam
Chapter 3

Remote Microscope Function

The remote microscope was built to address the need for visual feedback during remote processing of integrated circuits. As described in the previous chapter, the CIDM group here at MIT (as well as Stanford’s CIM research group) envision a more distributed and collaborative approach to fabrication of ICs so that expensive and specialized equipment and resources can be shared by many users. Ideally, this resource sharing should be as transparent as possible so that the actual process designer need not physically be present at the remote facility. As a result, the remote microscope will play a very important role in orchestrating the communication between the remote laboratory and designer.

3.1 General Operation

The remote microscope developed as part of the CIDM project is a first generation system that allows a user to view and “virtually control” a microscope over the internet. The user has the capability to pan and zoom the microscope, request single frames, store images to disk, and participate in conference inspections where multiple clients can observe the microscope at the same time. The remote microscope can be used effectively on any workstation or X-terminal on the internet, and can conceivably be located as close as outside a clean room (so that one can view a wafer without having to enter) or as far away as across the country (as was demonstrated between MIT and Stanford in August.
The remote microscope basically consists of two parts: a client interface that consists of three windows on an X-terminal (Control Panel, Global Image, and Zoom Image), and a server interface which consists of a PC equipped with a frame grabber board and video camera connected to an inspection microscope. The PC is connected to the internet and acts as the server for multiple clients that wish to view the microscope.

By using a remote microscope client, an individual ideally should have no need to be physically present at the actual microscope, for he/she should be able to control and view the remote microscope as easily as any traditional microscope. In actuality, there is an undesirable loss in resolution during image capture and latency during transmission that will make telemicroscopy less effective than manually operating the microscope, but hopefully these problems can be addressed with future design enhancements and improvements in communication bandwidth. Nonetheless, the remote microscope currently is a very cost effective and practical way to provide real time visual images of wafers to distant users.

Figure 4 shows a typical setup of how a remote microscope might be used to interface between different facilities. In this example, the primary facility, MIT's Integrated Circuit Laboratory has several non-operational machines, so certain processing steps are to be performed at MIT's Lincoln Labs, rather than delay the processing while the ICL machines are repaired. The wafers are physically transported to the Lincoln Labs facility, but can be viewed remotely, possibly in conference mode, by the microscope clients depicted at the MIT ICL and UC Berkeley sites. Both MIT's ICL and Lincoln Labs use the same CIM (Computer Integrated Manufacturing) systems, CAFE, which has been modified to incorporate remote processing steps [Kwo95]. This makes it very simple, and transparent to the user to have an intermediate remote processing step.
FIGURE 4. A possible remote microscope setup. The primary facility is MIT’s ICL, and a remote step is performed at Lincoln Labs.

In our initial implementation of the remote microscope, we actually did not have the system completely autonomous with the microscope controlled from the PC. Instead, a person at the remote microscope site must manually pan, zoom, or focus the microscope when prompted by the PC. Because the exact way in which the microscope moves is irrelevant to the remote user, this setup will still appear, from the users point of view, to be fully automated. This system will be easy to automate in the future because it will only involve writing new drivers for the PC to control the motors and gears for mechanically positioning the microscope, while the rest of the software is unaffected.
3.2 Client User Interface

Typically, at the remote microscope site, a server program will be running to which a client process will try to connect. At the local fabrication site, the user will run the client’s user interface program on his workstation and connect to the remote microscope server. Although only one client will have control over the microscope, our system will allow multiple clients to passively view the microscope. When a request for a frame by the client in control is sent to the server, the server grabs a new frame from the video camera, converts the image to a GIF format, and transmits the image back to all of the clients.

Before the microscope can be used, the server must be running on the PC and be ready to accept connections from clients. When the client is initially started up, the user is prompted with a dialog box to input the server name and the port number.

![Figure 5. Initial connection window. Process is requesting user to enter microscope location.](image)

After a connection is made, the user is then presented with three main windows that make up the microscope interface: the control panel, global view, and zoom view (see figure 6). The control panel basically allows the user to set the magnification and position of the microscope, and initiate a capture into either the global view or zoom view. The reason for having both a global view and a zoom view is for the user to be able to have a frame of reference in the global image while examining an individual feature more closely in the zoomed frame. Typically, a user would first slowly increase the magnification and repeatedly pan the microscope until an appropriate global view is seen, after which the user would select a region of interest in the global view to magnify into the zoom frame.
FIGURE 6. "Console", "Global", "Zoom" windows that make up client interface.
3.2.1 Client’s Console Display Window

The console for the remote microscope is divided into three sections, as seen in figures 1,6a. The left most column consists of radio buttons that allows the user to select a new magnification setting for the microscope. In a future development we would also provide the option to set the light intensity level. Below these buttons is an entry for the X-Y coordinates, CURX and CURY, that the user would like to pan towards. A global coordinate system is used in the remote microscope with spacing equal to one pixel for a microscope image viewed at the highest magnification (1000X in this implementation). In otherwords, when an image at 1000X is displayed in either the global or zoom window, the coordinates in the X and Y directions will increment by one for every pixel space. On the otherhand, for a magnification of 100X, each pixel of the image displayed should increment the coordinate by 10. Thus a well defined global coordinate can be unambiguously defined for the microscope. In actuality, such a clean coordinate system may not be possible. Depending on the minimum resolution of the stepper motor for panning the microscope platform, a unit change in the microscope stage most likely will not map to an integral number of pixels in the global or zoom image. Depending on the exact specifications of the automatic stage controls, the coordinate system used to describe the wafer may be much more complex.

Once the CURX and CURY positions are set, then the user can initiate a frame capture which will first cause the microscope stage to move to the correct coordinates and magnification, and secondly initiate a video capture by the PC to transmit the image back to the client. The middle column of the console window contains the action buttons that allow the user to send requests to the microscope. The two most important controls are the “GRAB” and “ZOOM” buttons, which are used to initiate frame captures into either window once the magnification and position are entered. It is efficient to have these separate submit buttons because the user can first prepare all the microscope settings and then send a single request to the microscope to minimize communication overhead.

The “CEDE” and “TAKE” control buttons are also extremely important for orchestrating the control of multiple clients for the remote microscope. In order to keep the microscope operation coherent, it is important for only one client to have control of the
microscope at any time, during which all other clients should be in a passive “view only” mode. In fact, a remote microscope can be in one of three states: “NONE”, “PASSIVE”, or “CONTROL”. When clients first begin to connect to the server, they enter the “NONE” state where no client is in control, and the server is simply waiting for someone to finally take control. In the “NONE” state, most of the control buttons are inactivated except for the “TAKE” button which is made valid. (in fact, the only time when it is valid). After a client issues a take control message, it must wait for a reply by the server before the state can change. This is extremely important because several clients can asynchronously request control of the microscope at the same time, so the server must decide which client’s request to honor, and must then notify each client whether or not their request was granted. In actuality, the server program maintains a data structure to keep track of the state of each client, and whenever the server decides to grant one of the clients control status, relays this fact back to every patron. The client that manages to take control will then have a fully functional control panel that will allow the user to set the microscope magnification, position, and to initiate a frame grab. Furthermore, the user can also cede control so that another client might be able to operate the microscope.

While one client is in control, all other clients must be in the “PASSIVE” mode where the user can view, but has no control of the microscope. The passive client is a mirror representation of the control client, showing each user the same images and settings for the microscope. In passive mode, the many controls on the access panel are made invalid to insure that the program will not interfere with the microscope operations. This interlocking mechanism of having a client send a take control or cede control message to the server, yet having to receive an acknowledgment before changing state, synchronizes the control of the microscope to ensure that the microscope is only operated by one person at a time.

In the control panel there are also button commands to save either the zoom or global image to a file in GIF format1. This functionality is very useful for keeping a record of images during the fabrication, and can be used to easily incorporate images into docu-

---

1. CompuServe Corporation’s Graphical Interchange Format
ments for papers or journals. In fact one of the most useful purposes for the remote microscope is in order to produce easily accessible electronic images of the microscope view.

Finally, the right hand column of the control panel lists all current users of the microscope. As different users dynamically connect and disconnect to the remote microscope, the server will keep track of all registered users and keep each user list updated. Users currently are identified by the hostname of the computer running the client program. Although not very user friendly, this was a practical way of keeping track of microscope users because the hostname information was readily available as part of the data exchanged during a socket connection.

3.2.2 Client’s Global Display Window

The global display window (figs. 2, 6b) is the primary window to view microscope images in the client interface. Any particular microscope image in the display has an associated magnification level and X-Y position corresponding to the location of the microscope stage. In the global display, these variables are listed at the top of the window. In addition, the global display also keeps track of the current cursor location whenever it is positioned within the microscope image (CX and CY values). Again, these coordinates reflect the global coordinate system of the wafer, and as a result scale appropriately with changing magnifications.

The cursor position in the global display plays a primary function during the operation of the microscope because one usually pans the microscope stage by positioning the cursor at an appropriate location and then clicking the first mouse button in order to set the CURX and CURY value (in the main control panel) that will be sent to the microscope during the next grab or zoom command. The CURX and CURY (which will become the XPOS and YPOS values in the display after the grab is complete) corresponds to the upper left hand corner of the new microscope image. When the first mouse button is clicked and the CURX and CURY values are set, a pointer will also appear on the image, indicating to where the microscope will pan (upper left hand corner). In fact, every client connected to the server will see the pointer in the image, so that in effect, the pointer can be used to communicate between users to bring everyone's attention a certain feature in the micro-
scope image. The new pan position for the next frame grab can also be explicitly set by manually entering a value into CURX and CURY, but this will not place a pointer in the image, which results in little visual feedback to the user. If however, the point to which one would finally zoom lies outside the borders of the global display, the easiest and quickest way to set the pan location is to manually enter the coordinates. Obviously, if one first views the microscope at the lowest magnification and makes incremental adjustments, then it will be easy to quickly locate the desired area of interest by a series button clicks on the mouse.

In addition to a pointer appearing in the image after a button click, a rectangle detailing the region that will be magnified will also appear. However, this rectangle is only created when the new magnification setting in the console window is higher than the current magnification of the reference image, because otherwise the effective rectangle would stretch beyond the boundaries of the display. Also, the dimensions of the rectangle are directly proportional to the ratio of reference magnification to new magnification. Effectively, the rectangular region will show exactly what fraction of the microscope image will be blown up during the next view. The user can also generate arbitrary size rectangles by dragging the mouse while clicking the middle mouse button. Although this rectangle will no longer have the meaning of showing what the next microscope image will be, it does serve a purpose (like the pointer) to direct people’s attention to various features or regions within the microscope. The third mouse button is used to clear the rectangle.

3.2.3 Client’s ZOOM Display Window

After a suitable panoramic view is positioned in the global window, and the user wishes to then zoom in further to investigate certain features more closely, one generally will choose a new magnification and pan position, and then grab the image into the zoom window. This has the benefit of allowing the user to examine a particular feature with high magnification, while still keeping a panoramic view of the region of interest. On subsequent captures, the user can choose to pan within the global view again, pan within the zoom view, or by manually entering coordinates, pan to an area outside the current view.

In actuality, the zoom window and global window have exactly the same behavior
and can be used interchangeably. Given a magnification and X-Y position, the user can
place the new microscope image into either window by asserting the "GRAB" or
"ZOOM" buttons, after first positioning the pointer in either window to set the X-Y pan
position (fig 1, 6a). The benefit in having identical functionality for the global and zoom
windows is flexibility for the user to choose which windows should contain the new
image. For example, it might be conceivable for the user to "ping pong" between the glo-
bal and zoom images since it may be desirable to have a one frame memory rather than to
maintain a panoramic view of the wafer.

3.3 Client for Conference Inspection

By implementing the remote microscope over the internet, it is possible to perform
conference inspections among several clients. In conference mode, one client will take
control of the microscope while all other clients observe passively. Each time the primary
client requests another image, or pans within the global or active display, the result will be
relayed to all of the clients. Each client will have the same basic interface, so any of the
clients can be in control. As described above however only one client is allowed to be in
control of the microscope at a time and users have the capability to take or relinquish con-
trol at appropriate times. The capability for conference inspections is a very important fea-
ture for our remote microscope system that will be extremely useful when several
individuals, each located anywhere on the internet, are needed to view the microscope. For
example, in the context of integrated circuits, it might be very useful to have the remote
fabrication engineer, the local fabrication designer, and the circuit designer to all partake
in a conference inspection to examine an especially critical feature in a wafer.

The pointer and rectangle visual aids described above are invariably used during
conference inspections to direct the conference members attentions to certain areas of the
microscope images. The rectangle aid is also important for bounding a region of interest
while the pointer is useful for bringing attention to individual features, so these tools,
when used in conjunction with a conference telephone call or other audio link, can be used
very effectively to communicate between users. While only the client in control is allowed
to manipulate the visual aids, all members will see the effects, and each will have the
opportunity take control of the microscope when another relinquishes power.
3.4 Server Interface

Ideally, the server should be fully automated so that the client user has full control over the microscope system, but as described above, our initial implementation involves an attendant responsible for controlling the microscope when prompted by the PC. As a result, it would be extremely useful for the attendant at the physical microscope to have a graphical display that shows the same data as the client’s display in order to understand a particular request. One option was to implement a graphical user interface for the server, however the simpler alternative that we used was to have the attendant just start up another client and initiate a conference inspection. As a passive client, the user manning the microscope could see exactly what regions of the microscope image that the remote user would like to examine.

In future developments, we would like to automate the microscope completely so that the panning, magnification setting, and focus can all be controlled by the PC. This improvement will have little impact on the overall remote microscope system since we simply need to create subroutines that hook into the existing server code to drive the mechanical motors to position the microscope controls. Since the communications and interface backbone to the microscope are currently in place, even now the microscope “appears” fully automated because the human manipulation of the equipment is transparent to the user.
Chapter 4

Practical Uses for Remote Microscope

The remote microscope is a general purpose tool that will provide remote microscopy for users on the internet. Since it is used just as one would use an ordinary microscope, it is applicable to any field involving microscopy such as semiconductor processing, medicine, or biology for example. Also because the remote microscope allows multiple clients to simultaneously view the microscope in a conference mode, it naturally lends itself to more collaborative research.

4.1 Use During Remote Fabrication of Integrated Circuits

Our proposed remote microscope can be used in a variety of situations that involve viewing a microscope image. Of foremost importance to us is to apply this system to the remote fabrication of integrated circuits, which would allow the local researcher to inspect his/her wafer being processed at a remote facility. By having full control over the facility’s microscope, the actual designer is given practically the same access to visual data after a processing step as any technician physically located at the remote facility. It is possible then that the primary designer can inspect the wafer after each step and uniquely determine the evolution of the wafer operations. As a result, the remote fabrication engineer can be completely ignorant about the details of the processing results, and simply confer with the primary designer before proceeding to the next machine operation. In a futuristic
fully automated lab, the actual lab technicians could be replaced with automated machines that query the primary designer before proceeding. Figure 7 shows a typical flow chart describing how the remote microscope could be integrated into the remote manufacturing process to provide inspection/decision making capabilities during a mask alignment/photoresist exposure step.

**FIGURE 7.** Flow chart showing inspection steps during mask alignment/lithography process.
4.2 Use During Local Fabrication of Integrated Circuits

Another way the remote microscope can be used is simply as a way to inspect wafers at the local facility without having to suit up and enter the clean room. This would allow the researcher to be outside the clean room yet still be able to manage and oversee the production process. Furthermore, many more people involved with the design of the overall circuit, including process and circuit designers, can have much easier access to the wafers during the semiconductor processing. As described above, conference inspections will allow multiple experts including technology, circuits, and systems engineers to cooperate during the actual fabrication design stage. The remote microscope definitely is not limited to long distance inspection, and in fact can be extremely useful on a local scale.

4.3 Remote Microscope for Data Collection

Finally, the remote microscope system can be used to record visual images during the process because it is simple to store images to disk once they are frame grabbed and displayed on a computer. For example, a picture of the wafer can be recorded after each relevant machine operation to keep a diary of the process history, or images of interesting features can be stored and filed away for future viewing. In the CAFE system, an additional field can even be added to the opinions in the task tree that point to a location where a corresponding microscope image is located. When an individual invokes a traveler report that contains pictures as well as raw data from the CAFE system, he will be able to analyze the process run much more effectively, and have a permanent record of the process performance. For example, if one is comparing and contrasting new recipes and processes, having a history of microscope images can be an invaluable resource.

The ability for a computer to capture microscope images is in itself useful for integrating pictures into papers or memos. A variety of microscope peripherals that allow one to make printed hardcopies are available, but usually the equipment is expensive and the hardcopies are difficult to incorporate into other documents. The standard GIF images that can be created with the remote microscope on the otherhand can be readily imported into documents or graphics programs that can edit and annotate images. Images in electronic form also lend themselves nicely to web based publications, which are becoming more
commonplace as the popularity of the web increases.

4.4 Remote Microscope Applied to Fields Outside of IC Fabrication

Although our primary emphasis for the remote microscope is for applications with respect to the CAFE environment for manufacturing integrated circuits (and especially remote fabrication), the remote microscope can be applied to any field that involves inspection through microscopes such as medicine or biology. For example, in medical research, the remote microscope can be used for remote diagnosis, conference inspection, or as a digital camera to capture images to be incorporated into other computer programs. Although expensive medical imaging systems already exist, the remote microscope might be a readily available alternative for performing inspections that have lower resolution requirements.
Chapter 5

Hardware Implementation

One of the most attractive qualities of implementing the remote microscope is its low cost, which is largely due to the fact that the microscope software was written here at MIT, and the hardware needed is readily available commercial equipment. The hardware required to implement the remote inspection setup include a framegrabber board, video camera, viewing monitor, and an ordinary PC. To interface with the video camera, we decided to use a Pentium PC because personal computers are relatively cheap, readily available, and are compatible with a variety of framegrabber boards. The cost of the entire system (not including the microscope) was only on the order of $5800. For a more detailed listing of equipment costs, please refer to the appendices.

<table>
<thead>
<tr>
<th>Component</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>75 Mhz Pentium PC</td>
<td>~$2100</td>
</tr>
<tr>
<td>Targa+ 16/32 Image Capture Equipment</td>
<td>~$1800</td>
</tr>
<tr>
<td>Video Camera Equipment</td>
<td>~$1900</td>
</tr>
<tr>
<td>Total</td>
<td>~$5800</td>
</tr>
</tbody>
</table>

**TABLE 1. Approximate cost of remote microscope hardware.**

The current hardware implementation of the remote microscope however does not include any capability for the PC to automatically position and focus the microscope. Because our goal is to have the microscope completely automated, future versions of the
MIT remote microscope will definitely be more expensive. Nonetheless, the overall cost of the system should still be reasonable, especially in comparison to expensive medical telemicroscopy systems.

5.1 Targa+ Image Capture Board

For our purposes, we decided to purchase the hi-end Truevision Targa+ 16/32 frame grabber board over other models because we wanted to be sure to capture very high quality images for our demonstration system, since feature resolution is extremely important for inspecting integrated circuits. Depending on one’s particular application, other framegrabber boards may be more cost effective, and it should be straightforward to develop new device drivers for our system that would work with a variety of different video hardware. The Targa+ is capable of capturing frames at a variety of resolutions, but because the stored images will be displayed on a computer screen, the 640X480 square pixel mode with 16 bits/pixel is best suited in order to have the correct aspect ratio on the computer monitor. In addition to composite and S video, the Targa+ is also capable of receiving RGB video inputs, which are less susceptible to noise and interference than composite video, resulting in cleaner grabbed frames[Tru94]. However before being transmitted to the client, the Targa+ image is converted into a GIF format, which introduces color quantization that limits the number of colors to 200.

In actuality, the Truevision Targa+ 16/32 may have not been the best frame grabber board to choose from because the toolkit routines were very archaic and difficult to port to OS/2. Truevision “officially” supports only DOS/Windows operating systems, and in fact, from my experiences, works correctly only with the Microsoft C Compiler. In our final implementation, we used a different OS/2 based toolkit that was supplied by an independent vendor to control the video capture board, but that too could only be compiled with the IBM CSet++, and could not be easily ported to GNU GCC.

5.2 Live Video Monitor

The Targa+ board cannot overlay live video on an ordinary computer monitor. It requires a special (and expensive) multisync monitor that has a horizontal frequency scan rate as low as 15.75 Khz. It was more cost efficient for us to purchase a separate video
monitor (Panasonic CT1384Y color monitor) that independently displays the live video from the camera. Having a separate video monitor has a practical advantage as well because it can be used as a persistent viewing display even when the computer is used for other purposes. The Panasonic monitor has a resolution of 420 horizontal TV lines and is also capable of displaying S (or Y/C) video which is outputted from the Targa board [Pan94].

5.3 Video Camera

The Sony DVC-151A video camera was chosen largely to match the performance capabilities of the Targa+ board. This camera has a horizontal resolution of 440 TV lines, is capable of RGB video (which takes advantage of the Targa board's capabilities), and has a C-mount that will allow us to easily attach it to a microscope eye piece. The camera also requires the CMAD2 power source, which supplies DC power and additional video outputs. By using the DVC-151A camera in RGB mode in conjunction with the Targa+ framegrabber, very high quality images can be captured. In fact, the captured frame displayed on a computer monitor was at an even higher resolution than the live video on the external monitor, since the camera has a horizontal resolution of 440 TV lines while the monitor has a resolution of 420 TV lines. On the other hand, the Targa+ framegrabber can capture images at 640x480 square pixels, which is greater than the resolution capabilities of the DVC-151A camera [Son94]. Furthermore, the computer monitor inherently displays images better than on a video monitor because the computer monitor does not flicker noticeably like the interlaced image on a TV screen.

5.4 Interconnection

There are several components involved in connecting the system together. At the centerpiece of the setup is a PC with an ethernet card and the Targa+ framegrabber board, which has 9 pin D-Sub connectors for both the video in and video out. The CCD camera also has a D-Sub connector, although with a slightly different pinout. However, by using a straight 9pin to 9 pin video cable, the camera can output RGB, Composite (but not S video) to the Targa+ board, which is quite tolerable since RGB video typically generates the highest quality video captures. The CCD camera also needs a special power supply
unit which uses a 12 pin connector to supply power to the camera and to also provide the option for routing video signals through the power supply itself. Finally, the TV monitor is connected to the Targa+ board directly so that the user can view the video image exactly as how the board does. The Targa board outputs RGB, Composite, and S video, while the TV monitor can view only Composite or S video. Usually the S video input on the monitor will be used since the image is cleaner.

FIGURE 8. Component setup for remote microscope.
Chapter 6
Software System Overview

This chapter describes the overall software system design of the remote microscope, while later sections of this document will describe in more detail the exact implementation and functional behavior of each software component. The remote microscope involves several pieces of software which include a Tcl/Tk user interface, a client daemon for processing binary data, a server process, and a capture routine that grabs video frames to disk. The client interface and client daemon both run on a UNIX workstation, while the server process and capture routine run on the PC under OS/2 Warp. The communication between the client interface, daemon, and server process is based on the CHAT messaging tools also developed here in MIT’s CIDM group.

6.1 CHAT Message Passing Tools

The CHAT message passing tools played a very important role in implementing the remote microscope because it served as the basis for the communications link over the internet using TCP/IP Berkeley Sockets between the client and server [Ste90]. These tools were useful because they provided a suite of routines that abstracted away low level details concerning making socket connections, packaging new messages into a suitable format, and polling incoming sockets for new data. Perhaps the most useful capability of these tools was to implement event based message handling, which would automatically
call a handler routine whenever a complete new message is received on a socket [Car95]. The CHAT tools consists mainly of three libraries: the connection routines were responsible for setting up the initial socket connection, the communications handler routines allowed for event based processing of messages, and finally the object routines allowed one to easily package record-type messages into a suitable formatted buffer for transmitting. The tools were written in both C and Tcl, and was also ported to OS/2, making it an ideal medium in which to base the messaging between the client and server for the remote microscope. With the use of these tools, it was simple to integrate the different processes together by implementing message passing communications between each program.

The server and client for the remote microscope use a message passing scheme to communicate with one another. The server program usually sits idle until a message is received and processed, while the client interface is idle until a return message is received or a command request from the user is issued. The message formats used are based on the “object” structures implemented in the CHAT communications routines.

6.1.1 CHAT object messaging units

A CHAT “object” is a data structure, similar to a C “struct” or Pascal “record” that is treated as a single unit but consists of any number of slots that each contain data as an arbitrary dimensional vector of primitive CHAT data types that include binary byte, ASCII, 32-bit Long, or Short. Other data types described in the CHAT documentation [Car95]. The objects are created and manipulated with a set of constructors, selectors, and destructors provided by the CHAT tools to easily create messages objects that are understandable by all CHAT based applications. Ideally, these objects should be compatible with any Tcl or C program running the CHAT tools on different types of machines. Objects themselves cannot be sent across the network through socket connections, because the objects must first be packaged into a CHAT specific buffer format that can be received by another CHAT program. Once this buffer is received, then the CHAT routines will recompose the message object, taking care of any details such as accounting for the Endian type of the host machine when non ASCII data is processed.

When using the CHAT C routines, the binary data types are all well defined. How-
ever, because TCL cannot process binary data, the CHAT Tcl routines in actuality implement the various data binary data types in objects as simply ASCII data that is being interpreted as one of these data types. This distinction is actually transparent to the user, because the different data types in the objects will be treated correctly regardless of whether the buffer containing the data is received by a Tcl client or a C client.

The C based CHAT tools actually provide two types of buffers to package these object types, a binary and an ASCII buffer. The binary buffer is based on the TIFF file format and is not compatible with the Tcl CHAT tools because Tcl has no way of receiving binary data on a socket descriptor, so it is only useful for communicating between two C programs. The ASCII buffer on the otherhand is understood by both the C and Tcl CHAT based tools. Packaging an object into an ASCII buffer involves converting any non ASCII data types slots by replacing every 4 bits with an 8 bit hexadecimal ASCII value (‘0’ - ‘F’). Once the ASCII buffer is received by a Tcl or C CHAT application, it would then be expanded into an object message.

6.2 Remote Microscope Server

The remote microscope server is responsible for receiving and processing user requests, and must be able to communicate with the hardware needed to capture an image from a microscope. Because the video hardware was setup to be controlled by an ordinary personal computer, we felt that it would be very desirable to have the PC itself run the server process so that the entire remote microscope server could be implemented on a single personal computer. We also chose to run the PC under IBM OS/2 Warp rather than traditional MS-DOS largely because of the multitasking capabilities available to OS/2 systems. For example, with OS/2, the microscope server can be running in the background, while the PC is running other useful tools. Multitasking capabilities would also allow us to upgrade the single PC in the future to simultaneously control and support several microscope or camera systems that are installed on the PC. For example, the frame grabber input can be switched from several different video sources, each of which corresponds to a different remote microscope, and each system having it’s own controlling process. With a single-task operating system like MS-DOS, it would be very difficult to simulate several sessions concurrently with several different servers. The OS/2 operating
system is also desirable because it is widespread, reliable, easily installable, and supports TCP/IP protocols [OH93].

An important reason why we were able to run the PC under OS/2 was because we managed to port the CHAT message passing tools to OS/2. This allowed the server process on the PC to communicate very easily with the client process. There were some drawbacks to using the OS/2 operating system though. The Truevision Targa+ capture board does not officially support any OS/2 device drivers, although we did find an independent developer that did market them. We purchased the OS/2 drivers, but discovered that without further modifications, it could only be compiled using IBM CSet++ or Microsoft C 6.0 (now obsolete), whereas the CHAT tools could only be compiled using GCC. As a result, the server process and the image capture routine had to be two distinct executables. An obvious future improvement to this remote microscope system is to try to make either the Targa+ drivers or the CHAT tools more portable so that they can be compiled using either GCC or CSet++.

![Diagram of system setup](image)

**FIGURE 9.** Setup with PC acting as microscope controller and server. PC runs OS/2 and remote client UNIX based, communications using TCP/IP socket interface.

### 6.2.1 Alternative server implementations

Before we decided to implement the server directly on the PC under OS/2, we did experiment with some alternative approaches to organizing the software. One of our first approaches was to consider running the PC under Linux because the CHAT tools were specifically written for a UNIX machine. Although running Linux was very desirable
since it had built in TCP/IP protocols, UNIX system calls, and multitasking capabilities, the operating system was not compatible with the Targa+ board, and would require writing a completely new set of Linux device drivers.

In another approach to designing the remote microscope (before we were able to port the CHAT tools to OS/2), we initially felt that it might be easiest to implement the overall server in two parts consisting of a server running on a UNIX workstation that shared a file system with the PC running either OS/2 or MS-DOS, and a simple image capture program on the PC that writes new images to disk. Our main reason for wanting to separate the server and the PC is that the server is more straightforward to implement in a UNIX environment (because it involves the CHAT tools) while the framegrabber board drivers supports only MS-DOS and to a lesser degree OS/2. By installing the PC with PC-NFS to mount the server’s filesystem, whenever the PC grabs a new microscope image and saves it to disk, the server will be able to access the image and transmit the data to the remote client over the internet. This alternative was not very clean though, and was abandoned once we were able to run the CHAT tools under OS/2.

**FIGURE 10. Setup with UNIX Server, UNIX Client, and OS/2 PC as microscope controller.**

### 6.3 Image Capture Software

The Targa+ 16/32 is actually a very complex video board which involves setting dozens of control registers to put the board in its various states [Tru94]. Although we were mainly interested in using the board to simply capture video frames, it also has the capability for a variety of video special effects that can manipulate live and stored images
together such as overlaying or chromakeying. In the future, it may be possible to take advantage of these capabilities to enhance the remote microscope by allowing the microscope attendant to overlay test images or markers onto the live video that might be useful for comparing the microscope image to a CAD drawing, for example.

Our main interest with the Targa+ board though was to capture video frames from a camera mounted on a microscope, so we developed a program that would capture a video frame to memory, and then output the image to disk in a TGA format\(^1\). Although the desired function of the board is very straightforward, interfacing with the Targa+ board under OS/2 turned out to be a major obstacle for implementing the remote microscope.

### 6.3.1 Capture utility under OS/2

Truevision does supply a high level toolkit that allows user to program the Targa+ board using a set of C libraries, but we discovered that the official toolkit was only compatible with DOS and Windows, while we hoped to run the PC under OS/2. However, with some effort, we did manage to find an OS/2 toolkit supplied by an independent contractor for Truevision, which allowed us to program the Targa+ board in native OS/2. Although the capture program could run under OS/2 it could not be incorporated directly into the server code because the CHAT library routines compiled only under GCC, while the OS/2 toolkit compiled only under IBM CSet++. As a result, the capture routine was implemented as a completely different program, which when called from the server process, would grab a video frame and store the image to file. This intermediate step can be improved upon in the future by either piping the output of the capture program to the server process, or by porting the server code to the CSet++ compiler (or capture program to GCC) so that both pieces of code can be compiled into a single executable.

Because the server has to fork another process in order to execute an image capture, the performance of the remote microscope is degraded, but in actuality the real bottleneck in the entire remote microscope operation was due to the conversion between the TGA format file created by the capture program and GIF format image that is transmitted

---

\(^1\) Truevision truecolor file format
to and displayed by the Tcl/Tk user interface. After the server initiates a frame capture, it then uses the PPMplus shareware graphics conversion routines to perform the TGA to GIF conversion, which involves a color quantization step since GIF images are limited to 256 colors. This quantization and conversion time takes up the majority of the time to process a new microscope image, and should probably be improved in the future.

6.3.2 Earlier capture utility implementation

Before we were able to find the OS/2 programmer’s toolkit for the Targa+ 16/32 image capture board, we were still able to run the PC under OS/2 yet interface with the hardware using the Targa+ DOS toolkit, since it is possible to shell out to DOS to execute a DOS program from within an OS/2 process. An important reason why we could interface with the board under OS/2 using a DOS shell was because the toolkit actually consisted of two sets of tools: the Targa+ Toolkit and the Targa Compatible Toolkit. The first set of tools required a device driver to be loaded when the computer boots DOS, which would prevent us from shelling out of OS/2 to DOS to run the software. However, the second set of tools, the Targa Compatible Toolkit, does not require a device driver to be loaded because the software directly communicates with the hardware. By using this set of library routines, we were able to write a capture program that initializes the board, sets the correct magnification, grabs a video frame, and writes the image to a TGA graphics file, and most importantly, can be run from within an OS/2 application by having the operating system shell out to DOS. An unavoidable consequence of implementing the frame grabber routine like this though is that the frame grabbing software must be completely separate from the server program running under OS/2.

This approach though was abandoned once we were able to purchase the new OS/2 Targa+ Toolkit so that the capture utility could be implemented under native OS/2. Because the new capture utility executable was an OS/2 executable, there was no need for the operating system to shell out to DOS like before, which sped up the capture process significantly. This approach of shelling out to DOS from under OS/2 is still useful to keep in mind though if for some reason one tries to use a different capture board that can only be programmed in DOS for example.
6.4 Client Interface

The client interface was implemented using Tcl/Tk and presents a purely graphical interface of the microscope to the user. The Tcl/Tk interface runs on any X terminal UNIX workstation and consists of three components including a console window, global window, and zoom window (see figure 6). The user of the remote microscope can only view and interact with the client interface, so it is very important that the interface emulate all the controls necessary to implement a remote microscope. As described earlier, the user has the capability to set the magnification, pan position, and grab a new microscope image into either the global or zoom windows. In later sections, the detailed implementation of the client are discussed more thoroughly.

Because the client interface was implemented using Tcl/Tk and uses the Tcl CHAT tools for communications, it can only transmit and receive ASCII buffer types. However, it is much more efficient for the server to transmit the GIF image message to the client through a binary buffer rather than converting the message into an ASCII buffer before transmission. As a result, we also needed to implement a client daemon that would sit between the Tcl/Tk client interface and the C program, and to process binary data for the Tcl/Tk client.

6.5 Client Daemon

A client daemon is needed to run in conjunction with the Tcl/Tk client process because it is responsible for receiving a binary buffer from the server process and then forwarding a new message in ASCII format back to the Tcl/Tk client. In order to simplify the communication channels involved between the interface, daemon, and server, we also decided to force all messages between the Tcl/Tk client and server to first pass through the daemon. The following diagram shows the role that the daemon plays in the overall software system.
A remote microscope Client  

Remote microscope Server

**FIGURE 11. Software components showing communication channels.**

For most messages, the client daemon simply forwards the buffer between the Tcl/Tk client and the server process. For example, when the Tcl/Tk client generates a message that needs to be sent to the server, the client will first package the message into an ASCII buffer, transmit the buffer across a socket to the daemon process that will then forward the exact same buffer to the server through another socket connection. Because both the Tcl/Tk client-to-daemon connection and the daemon-to-server connection use ASCII buffers based on the CHAT messaging tools, no conversion in format needs to be performed. Only two messages produced by the Tcl/Tk client are not treated in this manner: these are the "CONNECT" and "DISCONNECT" messages, which are used by the Tcl/Tk client to communicate directly to the daemon process. By having all Tcl/Tk client messages travel through the daemon first, we reduce the number of communication channels necessary to keep track of, and as a result simplify the software implementation significantly.

When a message is produced at the server end, it will be packaged into a binary buffer which can be received by the client daemon. For most messages, the daemon will simply recover the imbedded message in the buffer, and then convert it to an ASCII buffer that will be transmitted to the Tcl/Tk client. Again, we could have bypassed the daemon process and had the server directly communicate with the Tcl/Tk client using an ASCII buffer for non-image based messages, but this again would introduce a more complicated
communication network to keep track of. The main reason the daemon process was necessary was to receive binary buffers that contained the "GRABFRAMEREPLY" and "ZOOMGRABREPLY" messages from the server. These messages contain the new updated microscope state including the magnification setting, pan position, and most importantly the actual binary image in GIF format. We discovered that it was not efficient to try to package a large image file into an ASCII buffer, because the conversion time between the binary message to ASCII buffer itself could take on the order of several minutes (using the CHAT conversion tools). Instead, we were forced to transmit a binary buffer to the client, which drove us to develop the client daemon that was responsible for processing the new microscope image, and then notifying the Tcl/Tk (with another message) when the image is available.

Because all messages passed between the client and server are specified to also pass through the daemon, there is a slight performance hit. However, this extra level of processing only adds an insignificant delay between non-image message transit times because messages are produced as a result of human actions, and the socket delay is much less than human reaction time. The added benefit of using this rigorous communication scheme is a much simpler interface and easier software implementation. The daemon however is important to speed up the process of transferring a large image across the network.

6.5.1 Alternate to using daemon

We actually considered using alternative software systems that would not require an extra daemon process, but ultimately did not find these alternatives to be very efficient. For example, one easy alternative we looked at would be to simply have the server transmit an ASCII buffer instead of transmitting a binary buffer so that the Tcl/Tk client can directly receive data on a socket connection with the C server. Even though the image message object itself has a binary byte data format, an ASCII based buffer can still be created and transferred as described earlier (converting 4 bit binary to 8 bit hexadecimal ASCII) using the CHAT tools. If the object to be packaged is small, then the CHAT routines work acceptably, but if the message slot contains an extremely large non ASCII vector like an image, the resulting conversion step and packaging into an ASCII buffer can
take an unacceptable amount of time of up to several minutes to execute. Not only will the server have a difficult time converting the message into an ASCII buffer, but the Tcl client will also have a difficult time deciphering the received data. Since Tcl is purely an ASCII based scripting language, it will interpret any byte-type data as a number (represented by ASCII digits) rather than a byte of image data. A separate filter program (that is capable of dealing with binary data) would need to be written in order to post process the data and output the actual binary information.

A better solution though is to manually convert the binary image into an ASCII representation, and use a pure ASCII representation in the object message. (In the previous scenario, the object still had binary data vectors, and only the buffer was converted to ASCII). For example, using uuencode/uudecode or some other binary-to-ASCII conversion, we could convert our image so that the data in the object message will be a vector of ASCII characters instead of a vector of binary bytes. Consequently, creating the ASCII buffer will not involve any slow transformations, and the Tcl client will deal with pure ASCII data. Of course to recover the binary data, the Tcl client would have to initiate another program like uudecode on the message data. This alternative is feasible, but still time consuming because a compact GIF image would still need to be converted to a large ASCII file, pumped across the network, and then converted back to a GIF to be viewed with a special Tcl/Tk graphics utility.

6.6 Remote Microscope Messaging

The client and server programs that implement the remote microscope communicate with one another using the message passing functions built into the CHAT communication tools. Thus the application “messages” are simply CHAT objects that must be imbedded into either ASCII or binary buffers before being transmitted across TCP/IP Berkeley sockets, which connect the Tcl/Tk client to the daemon, and the daemon to the server [Ste90].

All messages used in the remote microscope have the same general format. They must consist of at least a “COMMAND” slot field with an ASCII string entry representing the message type, and depending on the particular command, the message can have addi-
tional slots and fields. For the remote microscope, only ASCII and binary byte data types were used within the message objects, and the binary data was used only to represent the graphic image that will be sent from the server to the client daemon.

6.6.1 Messaging between Tcl/Tk client and C daemon

The following is a list of all message types that are passed between the Tcl client and C daemon (that acts as an interpreter to the remote server). Even though it is possible to use the various integer CHAT data types such as byte, long, or short, that could be used to represent numbers such as in the XPOS, or YPOS or MAG message slots shown below, we decided to represent all the Tcl - daemon messages as ASCII strings. Pure ASCII notation was chosen in order to minimize the possibility of encountering any problems with the CHAT tools, which actually turned out to be slightly buggy in certain situations. However, by using the CHAT tools with Tcl/Tk ASCII message slots, the software was very rigorous.

<table>
<thead>
<tr>
<th>TCL CLIENT - TO - DAEMON</th>
<th>DAEMON - TO - TCL CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMAND</strong>: “CONNECT”</td>
<td><strong>COMMAND</strong>: “CONNECTREPLY”</td>
</tr>
<tr>
<td><strong>ADDRESS</strong>: server address</td>
<td></td>
</tr>
<tr>
<td><strong>PORT</strong>: port number</td>
<td></td>
</tr>
<tr>
<td><strong>COMMAND</strong>: “DISCONNECT”</td>
<td><strong>COMMAND</strong>: “DISCONNECTREPLY”</td>
</tr>
<tr>
<td><strong>COMMAND</strong>: “GRABFRAME”</td>
<td><strong>COMMAND</strong>: “GRABFRAMEREPLY”</td>
</tr>
<tr>
<td><strong>XPOS</strong>: X pan position</td>
<td><strong>XPOS</strong>: X pan position</td>
</tr>
<tr>
<td><strong>YPOS</strong>: Y pan position</td>
<td><strong>YPOS</strong>: Y pan position</td>
</tr>
<tr>
<td><strong>MAG</strong>: microscope magnification setting</td>
<td><strong>MAG</strong>: microscope magnification setting</td>
</tr>
<tr>
<td><strong>COMMAND</strong>: “ZOOMGRAB”</td>
<td><strong>COMMAND</strong>: “ZOOMGRABREPLY”</td>
</tr>
<tr>
<td><strong>XPOS</strong>: X pan position</td>
<td><strong>XPOS</strong>: X pan position</td>
</tr>
<tr>
<td><strong>YPOS</strong>: Y pan position</td>
<td><strong>YPOS</strong>: Y pan position</td>
</tr>
<tr>
<td><strong>MAG</strong>: microscope magnification setting</td>
<td><strong>MAG</strong>: microscope magnification setting</td>
</tr>
<tr>
<td><strong>COMMAND</strong>: “TAKECONTROL”</td>
<td><strong>COMMAND</strong>: “USERLISTDELETE”</td>
</tr>
<tr>
<td><strong>COMMAND</strong>: “CEDECONTROL”</td>
<td><strong>COMMAND</strong>: “USERLISTADD”</td>
</tr>
<tr>
<td><strong>USER</strong>: user name</td>
<td><strong>MODE</strong>: control state</td>
</tr>
<tr>
<td><strong>COMMAND</strong>: “CURPOS”</td>
<td><strong>COMMAND</strong>: “CURPOSREPLY”</td>
</tr>
<tr>
<td><strong>CURX</strong>: cursor X position</td>
<td><strong>CURX</strong>: cursor X position</td>
</tr>
<tr>
<td><strong>CURY</strong>: cursor Y position</td>
<td><strong>CURY</strong>: cursor Y position</td>
</tr>
<tr>
<td><strong>WHICH</strong>: “GLOBAL” or “ZOOM” window</td>
<td><strong>WHICH</strong>: “GLOBAL” or “ZOOM” window</td>
</tr>
</tbody>
</table>

<p>| TABLE 2. Message formats used between Tcl/Tk Client and Daemon. |</p>
<table>
<thead>
<tr>
<th>TCL CLIENT - TO - DAEMON</th>
<th>DAEMON - TO - TCL CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMAND</strong>: “RECTANGLEPOS”</td>
<td><strong>COMMAND</strong>: “RECTANGLEREPLY”</td>
</tr>
<tr>
<td>RX1: rectangle upper left X position</td>
<td>RX1: rectangle upper left X position</td>
</tr>
<tr>
<td>RY1: rectangle upper left Y position</td>
<td>RY1: rectangle upper left Y position</td>
</tr>
<tr>
<td>RX2: rectangle lower right X position</td>
<td>RX2: rectangle lower right X position</td>
</tr>
<tr>
<td>RY2: rectangle lower right Y position</td>
<td>RY2: rectangle lower right Y position</td>
</tr>
<tr>
<td><strong>WHICH</strong>: “GLOBAL” or “ZOOM” window</td>
<td><strong>WHICH</strong>: “GLOBAL” or “ZOOM” window</td>
</tr>
</tbody>
</table>

**TABLE 2. Message formats used between Tcl/Tk Client and Daemon.**

6.6.2 Messaging between server and C daemon

Because the server communicates only with the C client daemon (which is run on the same machine as the Tcl/Tk client), we needed to implement another set of messaging objects. In general, the messages are very similar to the messages passed between the Tcl/Tk client and daemon described in the previous section because usually the daemon just relays messages between the Tcl/Tk client and the server.

<table>
<thead>
<tr>
<th>DAEMON - TO - PC SERVER</th>
<th>PC SERVER - TO - DAEMON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMAND</strong>: “GRABFRAME”</td>
<td><strong>COMMAND</strong>: “GRABFRAMEREPLY”</td>
</tr>
<tr>
<td>XPOS: X pan position</td>
<td>XPOS: X pan position</td>
</tr>
<tr>
<td>YPOS: Y pan position</td>
<td>YPOS: Y pan position</td>
</tr>
<tr>
<td><strong>MAG</strong>: microscope magnification setting</td>
<td><strong>MAG</strong>: microscope magnification setting</td>
</tr>
<tr>
<td>SIZE: size of image</td>
<td>SIZE: size of image</td>
</tr>
<tr>
<td>IMAGE: vector of image bytes</td>
<td>IMAGE: vector of image bytes</td>
</tr>
</tbody>
</table>

| **COMMAND**: “ZOOMGRAB” | **COMMAND**: “ZOOMGRABREPLY” |
| XPOS: X pan position | XPOS: X pan position |
| YPOS: Y pan position | YPOS: Y pan position |
| **MAG**: microscope magnification setting | **MAG**: microscope magnification setting |
| SIZE: size of image | SIZE: size of image |
| IMAGE: vector of image bytes | IMAGE: vector of image bytes |

| **COMMAND**: “TAKECONTROL” | **COMMAND**: "USERLISTDELETE" |
| **COMMAND**: “CEDECONTROL” | **COMMAND**: “USERLISTADD” |
| USER: user name | MODE: control state |

| **COMMAND**: “CURPOS” | **COMMAND**: “CURPOSREPLY” |
| CURX: cursor X position | CURX: cursor X position |
| CURY: cursor Y position | CURY: cursor Y position |
| **WHICH**: “GLOBAL” or “ZOOM” window | **WHICH**: “GLOBAL” or “ZOOM” window |

**TABLE 3. Message formats between Daemon and PC Server.**
<table>
<thead>
<tr>
<th>DAEMON - TO - PC SERVER</th>
<th>PC SERVER - TO - DAEMON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMAND</strong>: &quot;RECTANGLEPOS&quot;</td>
<td><strong>COMMAND</strong>: &quot;RECTANGLEREPLY&quot;</td>
</tr>
<tr>
<td>RX1: rectangle upper left X position</td>
<td>RX1: rectangle upper left X position</td>
</tr>
<tr>
<td>RY1: rectangle upper left Y position</td>
<td>RY1: rectangle upper left Y position</td>
</tr>
<tr>
<td>RX2: rectangle lower right X position</td>
<td>RX2: rectangle lower right X position</td>
</tr>
<tr>
<td>RY2: rectangle lower right Y position</td>
<td>RY2: rectangle lower right Y position</td>
</tr>
<tr>
<td>WHICH: &quot;GLOBAL&quot; or &quot;ZOOM&quot; window</td>
<td>WHICH: &quot;GLOBAL&quot; or &quot;ZOOM&quot; window</td>
</tr>
<tr>
<td><strong>COMMAND</strong>: &quot;SETSTATE&quot;</td>
<td><strong>COMMAND</strong>: &quot;NONE&quot; or &quot;PASSIVE&quot; or &quot;CONTROL&quot;</td>
</tr>
<tr>
<td>STATE: &quot;NONE&quot; or &quot;PASSIVE&quot; or &quot;CONTROL&quot;</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 3. Message formats between Daemon and PC Server.**

### 6.6.3 Differences between two types of messages

There are some important differences between the messages used to communicate between the daemon and server, and messages passed between the daemon and Tcl/Tk client. In particular, the PC server-to-daemon messages are packaged into a binary buffer that is received by the C daemon, whereas the daemon-to-client messages are packaged into ASCII buffers. The server process also does not need to receive any "Connect" or "Disconnect" messages from the Tcl client, because only the daemon process needs to initiate a connection or disconnection to the remote server over the network.

Finally, the "GRABFRAMEREPLY" and "ZOOMGRABREPLY" messages supplied by the server contain two extra fields specifying the image size and image data for a microscope picture that were not included in messages from the daemon to Tcl/Tk client. These two extra fields contain the large binary data describing the new microscope image, and are directly interpreted by the daemon program. When the client daemon receives a "GRABFRAMEREPLY" or "ZOOMGRABREPLY" message from the server, it will receive the binary image data over the network, save the graphics file (GIF) to disk, and then proceed to forward this message, with the "SIZE" and "IMAGE" slots removed, to the Tcl/Tk client. However, all the other message formats such as "GRABFRAME", "ZOOMGRAB", "CEDECONTROL", "TAKECONTROL", "CURPOS", "RECTANGLEPOS", "CURPOSREPLY", "RECTANGLEREPLY", "USERLISTDELETE", "USERLISTADD", and "SETSTATE" are consistent in both cases. The following diagram illustrates the messages involved in sending a "GRABFRAMEREPLY" more carefully.
FIGURE 12. Diagram showing messaging required after a "GRABFRAME" request.

Have received "Grab-frame" request, now send image message back to Client.

Receive image message, and save data to file. Forward result to Tcl Client.

Receive acknowledgment message and ready to load saved image file.
Chapter 7

Server Software Implementation Details

This chapter will describe the implementation details of the server software needed for the remote microscope. Subsequent chapters will discuss the client software implementation. Overall, the software needed to implement the remote microscope system include the server process and image capture utility running under OS/2 on the PC, and the Tcl/Tk client interface and C based client daemon running on a UNIX workstation. There may be multiple copies of the client software running during a conference inspection, although there is only one server process running on the PC for each physical microscope.

The server code was written in C, and is responsible for maintaining the overall state of the microscope and servicing user requests from clients. In general, the server process will be sitting idle until a client connects or disconnects, or until it receives an incoming message from one of the clients that needs to be handled. For example the server would be responsible for processing client requests for grabbing new microscope images, or registering requests to place the cursor or rectangle visual aids on the canvas. In addition to handling client requests to interface with the microscope, the server is also responsible for important book keeping functions. Because clients can dynamically connect and disconnect from the remote microscope, the server also needs to dynamically keep track of
the members using the microscope at all times, so that each client can be serviced. Furthermore, since clients may connect at anytime during a conference inspection, the server must also keep track of the most recent microscope view so that new clients that connect will have an updated view when they join the conference.

7.1 Maintaining Multiple Conference Users

The remote microscope was designed to be used by multiple users that can simultaneously view the microscope, so it was very important for the server to maintain a data structure that keeps track of all users. The data structure we maintain is simply an array of pointers to member structures that consist of the name, control-mode, and socket descriptor associated with each client. The socket descriptor for each client-server connection is uniquely determined when a new client connects to the server. The mode specifies a binary control state associated with the client either being in control or not being in control (the later does not differentiate between a passive viewer, or a waiting client when no one yet has taken control of the microscope).

![DATA STRUCTURE SHOWING 3 USERS OF MICROSCOPE](image)

**FIGURE 13.** Data structure used to record current users of the remote microscope.

The details of the data structure that records the dynamic connection are abstracted away so that high level procedures are used to query/modify the listing, which allows us to have the server code be independent of the exact data structure used. The following table
shows the package used by the server code to manipulate the list.

<table>
<thead>
<tr>
<th>PROCEDURE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int add_member (int sd, char *name)</td>
<td>Enter new member's data into structure. Initially not in control</td>
</tr>
<tr>
<td>int delete_member (int sd)</td>
<td>Delete member specified with the sd socket descriptor</td>
</tr>
<tr>
<td>int set_control (int sd)</td>
<td>Give control to member with sd socket descriptor. Fails if someone else already in control</td>
</tr>
<tr>
<td>int off_control (int sd)</td>
<td>User gives up control. Fails if not the one currently in control</td>
</tr>
<tr>
<td>int someone_in_control()</td>
<td>Returns true if someone is currently in control of microscope</td>
</tr>
<tr>
<td>int get_mode (int i)</td>
<td>Returns mode status of ith member</td>
</tr>
<tr>
<td>int get_sd (int i)</td>
<td>Return socket descriptor of ith member</td>
</tr>
<tr>
<td>char *get_name (int i)</td>
<td>Return name of ith member</td>
</tr>
</tbody>
</table>

**TABLE 4. Package of tools to manipulate data structure of current users.**

By adhering to the philosophy of only using the above procedures to interface with the underlying data structure, we will also have an advantage of ensuring the integrity of the data structure at the lowest level. For example, we can always be assured that at most one client will ever be in control, because users can only call the ‘set_control’ procedure, which will only take effect if no one is currently in control of the microscope.

### 7.2 Maintaining Current State of Image

The server still needs to keep track of more than just the state of current users of the microscope. Because the server must also be able to bring up to speed any new clients that connect to the microscope during the middle of a conference inspection, the server maintains four additional global structures that contain important image information. The relevant information that must be remembered include the state of the global window, the state of the zoom window, and the cursor and rectangle visual aid positions. For both the global and zoom windows, we must keep track of the image buffer, the X pan position, and the Y pan position. For the cursor pointer, we must record which window (global or zoom or none) is used, and the X-Y positions. Similarly, for the rectangle visual aid, we must keep track of which window is used, and two pairs of coordinates to specify two cor-
ners of the rectangles.

<table>
<thead>
<tr>
<th>GLOBAL STRUCT</th>
<th>ZOOM STRUCT</th>
<th>CURSORPOSITION STRUCT</th>
<th>REPOSITION STRUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *command</td>
<td>char *command</td>
<td>char which[MAX]</td>
<td>char which[MAX]</td>
</tr>
<tr>
<td>char *image</td>
<td>char *image</td>
<td>char curx[MAX]</td>
<td>char rx1 [MAX]</td>
</tr>
<tr>
<td>int imagesize;</td>
<td>int imagesize;</td>
<td>char cury[MAX]</td>
<td>char ry1 [MAX]</td>
</tr>
<tr>
<td>char xpos[MAX]</td>
<td>char xpos[MAX]</td>
<td>char ry2 [MAX]</td>
<td>char ry2 [MAX]</td>
</tr>
<tr>
<td>char ypos[MAX]</td>
<td>char ypos[MAX]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char mag[MAX]</td>
<td>char mag[MAX]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 5. Server global structures to maintain microscope state.**

During normal use of the microscope, there isn’t any need for the server to remember the image state because clients will keep track of this information directly. However, remembering the state of the microscope is extremely important in order for the server to maintain clients that connect at arbitrary times. It is important for new clients that join a conference inspection to have a consistent view of the microscope when they first connect.

### 7.3 Server Execution

The following state transition diagram shows how the server process runs during normal operations. Usually the program will sit idle until an event occurs, and after processing the request, will again wait in the idle state. Since the server execution is event based, we felt the most natural way to model the code was through state transition diagrams that showed the effects of incoming messages received from the daemon (originating from the Tcl/Tk client). Before the server can receive any new messages shown below though, a new client must first connect to the server.
FIGURE 14. Model for SERVER execution showing general events that need to be handled.

When the server process is first run, it sets up a “listening” socket on a specified port to which all new clients will try to connect, and registers the “listening” socket with a “new_client” connect routine. This routine is called whenever a new client tries to connect to the server program and will be responsible for setting up a data socket and then registering the data socket descriptor with a “new_message” routine that processes incoming mes-
sages. Please refer to the CHAT manuals for a more thorough description of the connection process [Car95].

The server program then enters the main loop which consists of simply a while loop encompassing a “pause” statement, and waits until an event occurs. The “new_message” procedure is called whenever a complete message is received on a valid socket, the “new_client” procedure is called whenever a new client connects to the port, and the “dead_client” procedure is called whenever a client is terminated. Furthermore, other event signals are disabled when one of these messages is being handled to ensure that the handling routines are not affected by incoming messages.

By calling the “new_client” routine whenever a new client connects, the server will be able to accommodate any number of clients, and be able to accept a new connection at any time during which the client is running. This makes the server very versatile and efficient at asynchronously accepting new client connections. After a valid socket connection is created for the new client, the server then must update the new client so that it represents the most current state of the microscope, which is important when individuals enter conference inspections at arbitrary times. Whenever a client is terminated and a socket connection is broken, the server must also be notified, especially if the client in control dies, since the server must then change each remaining client’s state from “PAS-SIVE” to “NONE”. The “dead_client” routine is automatically called whenever a socket connection is broken. The process of connecting and disconnecting to the server is a dynamic process, allowing multiple users to connect to the microscope arbitrarily.

The usual operation for the server however is to receive data on one of the socket connections and to process it with the “new_message” routine. Once a complete message buffer is received on a socket, the event handler will initiate execution of the “new_message” routine that will first convert the buffer to an object message, which can be read and processed. As described earlier in the message format section, all messages passed between server and client (daemon process actually) have the same format with a “COMMAND” entry to specify the message type, followed by an arbitrary number of slots depending on the particular message. As listed in Figure 14, the server is capable of recognizing the following message commands: “GRABFRAME,” “ZOOMGRAB,”
"RECTANGLEPOS," "CURPOS," "CEDECONTROL," "TAKECONTROL". These messages are basically the result of requests or actions taken by the microscope user that is currently in control of the microscope (except for "TAKECONTROL," which a request from a user who isn’t currently in control). Once the server decodes the command type, it can then proceed to satisfy the request.

7.3.1 "GRABFRAME" message

The "GRABFRAME" message basically is a request for the server to position the microscope, and to return the image given at the specified X-Y coordinates and magnification. After the microscope is manually positioned and set to the correct magnification, the server will then execute the capture routine which will save the microscope image to a file on disk (global.tga). Next, the microscope will then convert the TGA graphics file to a GIF format by passing the Targa image through three filters: "tgatoppm" converts the targa file to a standard ppm notation, "ppmquant" quantizes the colors to choose only 200, and "ppmtogif" converts the quantized image to a GIF file (global.gif). This conversion time actually constitutes most of the time lag between when a client requests and finally receives a new image, so we will definitely be looking into writing our own or finding an improved version of a TGA to GIF conversion routine. After the conversion is completed, the GIF file is then read into memory, stored into the GLOBAL state data structure, and packaged into a reply message that will then be sent back to the client that initiated the request. Furthermore, this return message (consisting of the image, size, x position, y position, and magnification) will be sent to all other clients currently connected to the server so that everyone in the conference inspection will have an updated view of the microscope.
7.3.2 "ZOOMGRAB" message

The ZOOMGRAB message is almost identical to the GRABFRAME message, except that the captured image is saved into the "zoom.tga" and "zoom.gif" files, and that the resultant microscope image will be packaged into a "ZOOMGRABREPLY" message that is sent to all the clients connected to the server. Also the ZOOM state that is maintained by the server is likewise updated. Although we could have effectively collapsed the "GRABFRAME" and "ZOOMGRAB" messages into a single type of message (with simply a field that differentiates between the two), we actually kept the two as separate distinct objects. The drawback is that the code is a little bit longer with replicated segments, but the advantage is that the code is easier to understand.

7.3.3 "CURPOS" and "RECTANGLEPOS" messages

The "CURPOS" and "RECTANGLEPOS" messages are sent by the client to the server whenever a cursor or rectangle visual aid is placed on the canvas. The server is responsible for keeping track of any change in state, and for updating all connected clients
so that all users will have a coherent view of the visual aids. After receiving a "CURPOS" or "RECTANGLEPOS" message, the server will simply forward a "CURPOSREPLY" or "RECTANGLEREPLY" to each microscope user. Furthermore, the server will update its own CURSORPOSITION and REPOSITION data structures to reflect the current state of the microscope.

**FIGURE 16. Model of SERVER execution after receiving "CURPOS" or "RECTANGLEPOS" message from daemon (originating from Tcl/Tk Client).**

### 7.3.4 "TAKECONTROL" message

The "TAKECONTROL" message can be sent by any client who tries to take control of the microscope, but only at a time when the microscope is not currently occupied. After a server receives a "TAKECONTROL" message, it will grant control to the client, and then update all the other clients to change their states from "NONE" to "PASSIVE". If by chance more than one client attempts to take control of the microscope (for example
before the server has time to broadcast the fact that the first client has taken control), the
server will still operate correctly because the "set_control" procedure used to manipulate
the user list data structure will not be able to yield control to a particular client if the
microscope is already in use. Thus, the first one that manages to take control of the micro-
scope will keep control until he/she decides to cede control. None of the clients in the con-
ference inspection will be affected by a particular client sending a "TAKECONTROL"
message until the server finishes processing the request and responds by broadcasting
"SETSTATE" commands back to all the clients.

![Diagram of server execution after receiving "TAKECONTROL" message from
d daemon (originating from Tcl/Tk Client).]

FIGURE 17. Model of SERVER execution after receiving "TAKECONTROL" message from
daemon (originating from Tcl/Tk Client).

7.3.5 "CEDECONTROL" message

Naturally, the CEDECONTROL message is sent by a client when one wishes to
relinquish control over the microscope. The client user can only CEDECONTROL when
it has control of the microscope, and the result must be forwarded to all clients to force
users into the "NONE" state, where no user has taken control of the microscope.
7.4 Image Capture Software

The image capture program used to control the Truevision Targa+ frame grabber board was rather simple to write. As described earlier in the system implementation section of this document, the greatest difficulty in developing the capture software was to port the Targa+ toolkit to OS/2 so that it could execute under the same operating system that the server was running (OS/2). Although we were able to find an OS/2 based toolkit, we still could not completely integrate the server and the capture routines into a single process because the server could compile only with GCC (a limitation of the CHAT utilities) while the Targa+ capture program could be compiled only under CSet++.

The capture routine was thus written as a stand alone application that captures a video frame and stores the image (TGA format) to disk, and is executed from within the server process. After execution, the server would then read the TGA file, convert it to a
GIF format, and send the resultant image to clients connected to the microscope.

**FIGURE 19.** Model of capture utility execution.
Chapter 8

Client Daemon Software Implementation

The client software basically consisted of two parts: the client interface written under Tcl/Tk (described in the next section) and the client daemon written under C. The client daemon was necessary so that a compact binary buffer could be transmitted over the network from the server. In order to run the client software, the Tcl/Tk client must first be run because it was designed to wait for a connection request from a daemon that will expect the interface to already be set up. Once the interface-daemon connection is made, the user can then connect to the remote microscope and become an active user.

8.1 Client Daemon Event Handling

The daemon process simply acts like an interpreter between the Tcl/Tk client and the server, usually doing nothing else besides passing messages between the two. However, when a message containing a microscope image is received from the server, the daemon process needs to retrieve the image and store it to disk. Also, when the user attempts to connect or disconnect to the microscope, the daemon must also respond.
The client daemon can naturally be divided into two regions that define incoming events arriving from either the Tcl/Tk client or the server. However, the server events (dotted lines) actually are not possible until after a server connection is made. When the client software is first run, the Tcl/Tk interface and daemon immediately establish a connection, and the user is then prompted to enter the server address and port number. Only after a connection with the server is established will the microscope interface be fully functional. Figure 21 shows how the daemon can handle incoming events from the Tcl/Tk client, where the first event to occur is always the "CONNECT" to server message.
8.2 Client Daemon Events From Tcl/Tk Client

ASCII buffer received from Tcl/Tk client

Convert ASCII buffer to object
Decode message

"CONNECT" msg

MAKE CONNECTION TO SERVER

Send back to Tcl/Tk client
"CONNECTREPLY"

Send back to Tcl/Tk client
"CONNECTREPLY"

To IDLE State

"DISCONNECT" msg

CLOSE AND UNREGISTER SERVER SOCKET

Exit daemon program

Forward ASCII buffer to server socket

To IDLE State

FIGURE 21. Model of DAEMON event handling showing incoming messages from Tcl/Tk Client.

<table>
<thead>
<tr>
<th>TCL/TK CLIENT - TO - DAEMON</th>
<th>DAEMON - TO - TCL/TKCLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND: “CONNECT” ADDRESS: server address PORT: port to connect</td>
<td>COMMAND: “CONNECTREPLY”</td>
</tr>
<tr>
<td>COMMAND: “DISCONNECT”</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 6. Messaging involved with events described in Fig. 21

The only messages from the Tcl/Tk client that the daemon need to be concerned with are the “CONNECT” and “DISCONNECT” messages, which are issued when the client wants to make or break a connection with the remote microscope server. All other messages from the Tcl client are simply forwarded to the server. In fact the very same buffer that is received from the client can be forwarded directly to the server because ASCII buffers are used in both cases. It is important to establish a server connection
before any non-connection related messages are sent to the daemon, otherwise the daemon will try to write data to a closed server socket, which will result in an error. Although there is nothing in the daemon code that specifically catches this type of behavior, the client interface to the user was implemented so that it is impossible for a user to send a microscope message before making a connection to the server. After the connection is established, the daemon will also be able to respond to incoming binary messages from the server as shown below.

8.3 Client Daemon Events From Server

![Diagram of event handling process]

**FIGURE 22.** Model of DAEMON event handling showing incoming requests from Server on PC.
<table>
<thead>
<tr>
<th>SERVER - TO - DAEMON</th>
<th>DAEMON - TO - SERVER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMAND:</strong> “GRABFRAMEREPLY”</td>
<td><strong>COMMAND:</strong> “GRABFRAMEREPLY”</td>
</tr>
<tr>
<td><strong>XPOS:</strong> X position of image</td>
<td><strong>XPOS:</strong> X position of image</td>
</tr>
<tr>
<td><strong>YPOS:</strong> Y position of image</td>
<td><strong>YPOS:</strong> Y position of image</td>
</tr>
<tr>
<td><strong>MAG:</strong> magnification level</td>
<td><strong>MAG:</strong> magnification level</td>
</tr>
<tr>
<td><strong>SIZE:</strong> size in byte</td>
<td><strong>SIZE:</strong> size in byte</td>
</tr>
<tr>
<td><strong>IMAGE:</strong> GIF image</td>
<td><strong>IMAGE:</strong> GIF image</td>
</tr>
<tr>
<td><strong>COMMAND:</strong> “ZOOMGRABREPLY”</td>
<td><strong>COMMAND:</strong> “ZOOMGRABREPLY”</td>
</tr>
<tr>
<td><strong>XPOS:</strong> X position of image</td>
<td><strong>XPOS:</strong> X position of image</td>
</tr>
<tr>
<td><strong>YPOS:</strong> Y position of image</td>
<td><strong>YPOS:</strong> Y position of image</td>
</tr>
<tr>
<td><strong>MAG:</strong> magnification level</td>
<td><strong>MAG:</strong> magnification level</td>
</tr>
<tr>
<td><strong>SIZE:</strong> size in byte</td>
<td><strong>SIZE:</strong> size in byte</td>
</tr>
<tr>
<td><strong>IMAGE:</strong> GIF image</td>
<td><strong>IMAGE:</strong> GIF image</td>
</tr>
</tbody>
</table>

**TABLE 7. Messaging involved with Server/Daemon events described in Fig. 22**

The only messages from the server process that the daemon need to be concerned with are the “GRABFRAMEREPLY” and “ZOOMGRABREPLY”. All the other messages are simply forwarded to the Tcl/Tk client. Unlike messages sent by the client though, the messages sent by the server are through a binary buffer, so the message must be packaged into a different ASCII buffer before being sent to the client. The “GRABFRAMEREPLY” and “ZOOMGRABREPLY” messages are treated almost identically: the message includes a microscope image which needs to be saved to disk, and then the server must notify the Tcl/Tk client when the file is available to be loaded into the interface canvas.
Chapter 9

Tcl/Tk Client Interface Implementation

The Tcl/Tk client interface is the main program to which a user of the remote microscope will directly interact. The Tcl/Tk client presents a graphical display that completely represents the remote microscope, displaying images of the microscope contents and allowing the user to control the microscope settings. The client software has three main responsibilities: it must display the contents of the remote microscope, accept input from the user to control the microscope, and process return messages from the server that contain new information about the microscope state.

Although the visual interface is the first thing one will notice about the remote microscope, the most interesting behavior of the client interface is it's dynamic interaction with the local user and the remote server. The Tcl/Tk interface is event driven so that the program simply waits until either the user mouse-clicks on a widget in the interface or a microscope message is received from the daemon (though usually originating from the server). The messaging between the Tcl/Tk client and daemon are again based on the CHAT utilities, so the communication link was straightforward to implement. The only difficulty was that Tcl based CHAT tools required us to use ASCII buffers for communication, necessitating the need for the daemon process. The user communicated with the Tcl/Tk client by a series of mouse-clicks and keyboard entries into the graphical user inter-
face, which was also straightforward due to the nature of the Tcl/Tk scripting language. The following diagram illustrates the general operation of the Tcl/Tk client interface. Note that the client must first establish a connection with the server (by sending the “CONNECT” request, and receiving a “CONNECTREPLY” before the actual microscope interface is displayed. The dotted lines reflect the fact that these states are not valid until after a connection to the server has been made.

![Diagram](image)

**FIGURE 23.** Model of Tcl/Tk event handling showing incoming requests from the daemon (most originating from server) and input from user.
<table>
<thead>
<tr>
<th>TCL/TK CLIENT - TO - DAEMON</th>
<th>DAEMON - TO - TCL/TK CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMAND:</strong> “CONNECT”</td>
<td><strong>COMMAND:</strong> “CONNECTREPLY”</td>
</tr>
<tr>
<td><strong>ADDRESS:</strong> server address</td>
<td></td>
</tr>
<tr>
<td><strong>PORT:</strong> port number</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 8.** Messaging involved with making server connection events shown in Fig. 23

### 9.1 Graphical Implementation

Much of the client program involves initialization code that simply sets up the graphics that are needed to implement the remote microscope interface. The entire interface is made up of Tk widgets (window objects) such as top-level windows, frames, buttons, radio buttons, labels, entry forms, and canvases, etc., and each one must be explicitly coded within the Tcl Client script. The widgets must then be “packed”, which simply means that the Tcl/Tk geometry manager will position the widgets on the screen. A large part of programming the client interface involved positioning these widgets on the screen to make the remote microscope visually appealing. The remote microscope interface consists of three top-level windows: the console, global, and zoom windows, each of which is entirely made up of Tk widgets. For a detailed description of Tk widgets please refer to Ousterhaut’s definitive text “Tcl and the Tk toolkit” [Ous94].

### 9.2 Global Variables

Because the Tcl/Tk client’s dynamic execution is event based (either a message from the daemon or an input from the user), it is important to maintain a set of global variables that can be accessed by all parts of the Tcl code, which will represent the global state of the microscope. When a user mouse clicks on a button for example, a procedure is not called with the necessary arguments; instead a script associated with the button will simply be executed, and the variables that it manipulates must be global. Also, when a message is received from the daemon using the CHAT tools, relevant arguments are not passed to the message handling routines; instead the event handler again must have access
to global variables.

<table>
<thead>
<tr>
<th>Global Variable</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM, ZM</td>
<td>Current magnification setting in Global and Zoom windows</td>
</tr>
<tr>
<td>GX, ZX</td>
<td>Current X coordinate of the upper left hand corner of image in Global/Zoom window</td>
</tr>
<tr>
<td>GY, ZY</td>
<td>Current Y coordinate of the upper left hand corner of image in Global/Zoom window</td>
</tr>
<tr>
<td>GXlocal, ZXlocal</td>
<td>Current X coordinate of mouse position in Global/Zoom window (mouse tracking)</td>
</tr>
<tr>
<td>GYlocal, ZYlocal</td>
<td>Current Y coordinate of mouse position in Global/Zoom window (mouse tracking)</td>
</tr>
<tr>
<td>CURXPOS</td>
<td>New X coordinate to position microscope in subsequent grab</td>
</tr>
<tr>
<td>CURYPOS</td>
<td>New X coordinate to position microscope in subsequent grab</td>
</tr>
<tr>
<td>CURSWITCH</td>
<td>Equal to &quot;GLOBAL&quot; or &quot;ZOOM&quot; depending on which window the cursor is in.</td>
</tr>
<tr>
<td>RECSWITCH</td>
<td>Equal to &quot;GLOBAL&quot; or &quot;ZOOM&quot; depending on which window the rectangle is in</td>
</tr>
<tr>
<td>CONTROLSTATUS</td>
<td>Equal to &quot;NONE&quot;, &quot;PASSIVE&quot;, or &quot;CONTROL&quot; specifying the control status of the client.</td>
</tr>
<tr>
<td>SETM</td>
<td>New magnification setting for subsequent grab</td>
</tr>
<tr>
<td>BUTTGDOWNFLAG</td>
<td>Flag for rectangle drag mode</td>
</tr>
<tr>
<td>gcanvasimage, zcanvasimage</td>
<td>Variables identifying image canvas objects</td>
</tr>
<tr>
<td>gcanvascursor, zcanvascursor</td>
<td>Variables identifying cursor canvas objects</td>
</tr>
<tr>
<td>gcanvasrectangle, zcanvasrectangle</td>
<td>Variable identifying rectangle canvas objects</td>
</tr>
<tr>
<td>temp x</td>
<td>X anchor point for stretching rectangle</td>
</tr>
<tr>
<td>temp y</td>
<td>Y anchor point for stretching rectangle</td>
</tr>
</tbody>
</table>

Table 9. Tcl/Tk Client listing of Global variables.

9.3 Dynamically Handling Console Inputs

After the Tcl/Tk client finishes the initialization of the remote microscope, which involves having the user connect to the server and then setting up the graphical interface, it enters a wait state where the application waits for new events to occur. The events can either result from a message being received from the server (passed in by the daemon) or from input from the user.

The user interacts with the Tcl/Tk client through a series of mouse clicks or key-
strokes within widgets in the graphical display, which cause a sequence of Tcl commands to be executed. The most widespread method for the user to interface with the client is through “button” widgets, which when activated automatically executes a command sequence. Most of the control actions issued by the user are thus a result of clicking on the “button” widgets labeled “GRAB”, “ZOOM”, “CEDE”, “TAKE”, “QUIT”, “GSAVE”, or “ZSAVE” (see fig 1, 6a).

![Diagram of event handling process](image)

**FIGURE 24.** Model of Tcl/Tk event handling of user console button inputs. Depending on client state, not all these button choices are allowed.
The “GRAB” button is clicked whenever the user wishes to grab a new microscope image (at the specified magnification and cursor position) into the global workplace window. When the button is clicked, the client sends a “GRABFRAME” message to the daemon that will then forward the message to the server. The “ZOOM” button behaves similarly; it is activated when the user wants to grab the new image into the zoom workplace window, resulting in a “ZOOMGRAB” message being sent to the daemon. If the microscope is currently free, the “TAKE” button can be issued by the user to take control of the system, resulting in a “TAKECONTROL” message being sent to the daemon. On the other hand, if the user already has control of the microscope, he/she can select the “CEDE” button, which will issue a “CEDECONTROL” message to the daemon. The “GSAVE” and “ZSAVE” buttons are used when the user wishes to save the global or zoom image respectively to a file. When either button is asserted, the client will bring up a separate dialog window for the user to enter the filename to save. Finally the “QUIT” button breaks the connection with the daemon and exits the program.

Another type of Tcl/Tk button is the radiobutton, which allows the user to select a single button among several possible ones, is used to select between the various microscope magnification levels. When the user selects one of these button magnifications, the client process sets the SETM global variable to the corresponding magnification. When a user wishes to grab a new microscope image, he/she must specify the magnification (by selecting one of the radiobuttons) and the X-Y pan position. The X-Y pan location corresponds to the CURXPOS and CURYPOS global variables, and can be set directly by
entering coordinates into an entry widget which binds the entered values to the global variables (the next section describes the clicking method for setting pan position).

![Diagram of event handling process]

**FIGURE 25.** Model of Tcl/Tk event handling of user radio button entries or keyboard entries into console.

When the client is in control of the microscope, all of the buttons except the "TAKE" button are active. On the other hand if the client is in a passive viewer mode, then only the "GSAVE" and "ZSAVE" buttons are active, since the client should not be able to influence the microscope state during the conference inspection. Finally, when no one is in control of the microscope yet, all the clients should have only the "TAKE" button active.

### 9.4 Dynamically Handling Mouse Inputs Over Image

The final way that the client accepts input from the user is through manipulating the mouse over the image that is loaded into the canvas widget. Whenever a new image is displayed in either the Zoom or Global windows, the Tcl/Tk client sets up several bindings associated with moving the mouse or clicking the left, middle, and right mouse buttons over the image. The following diagram shows how the Tcl/Tk client responds to mouse requests, assuming that the bindings are setup (see below for details).
FIGURE 26. Model of Tcl/Tk event handling of user input using mouse operations over Global or Zoom images.

<table>
<thead>
<tr>
<th>TCL/TK CLIENT - TO - DAEMON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMMAND</strong>: “CURPOS”</td>
</tr>
<tr>
<td>CURX: cursor X position</td>
</tr>
<tr>
<td>CURY: cursor Y position</td>
</tr>
<tr>
<td>WHICH: “GLOBAL” or “ZOOM” window</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMMAND: “RECTANGLEPOS”</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX1: rectangle upper left x position</td>
</tr>
<tr>
<td>RY1: rectangle upper left y position</td>
</tr>
<tr>
<td>RX2: rectangle lower right x position</td>
</tr>
<tr>
<td>RY2: rectangle upper left y position</td>
</tr>
<tr>
<td>WHICH: “GLOBAL” or “ZOOM” window</td>
</tr>
</tbody>
</table>

TABLE 11. Messaging resulting from mouse click over images shown in Fig 26.
The following code segment shows the mouse bindings that are established whenever a new microscope image is loaded into the global window. A similar code segment is executed whenever an image is loaded into the Zoom window.

```c
.global.c bind $gcanvasimage <Motion> {
    set GXlocal [xprime $GX %x $GM 1000]
    set GYlocal [xprime $GY %y $GM 1000]
    if {($BUTTONDOWNFLAG == "true") && ($CONTROLSTATUS == "ON")} {
        put_rectangle $tempx1 $tempy1 [expr 10+%x] \ 
         [expr 10+%y] GLOBAL LOCAL
    }
}

if {$CONTROLSTATUS == "ON"} {
    .global.c bind $gcanvasimage <Button-1> {
        put_cursor %x %y GLOBAL NETWORK
    }
    .global.c bind $gcanvasimage <ButtonPress-2> {
        if {($BUTTONDOWNFLAG == "false")} {
            set tempx1 %x
            set tempy1 %y
            set BUTTONDOWNFLAG true
        } else {
            put_rectangle $tempx1 $tempy1 [expr 10+%x] \ 
             [expr 10+%y] GLOBAL NETWORK
            set BUTTONDOWNFLAG false
        }
    }
    .global.c bind $gcanvasimage <Button-3> {
        set BUTTONDOWNFLAG false
        #this was in case run off screen, and then hit 3
        put_rectangle 0 0 0 0 NONE NETWORK
    }
}
```

**FIGURE 27. Excerpt of code setting up mouse bindings over Global image. (Similar code exists for bindings over Zoom image).**

### 9.4.1 Mouse motion binding

Whenever the mouse is in motion over an image, the Tcl/Tk program will read the coordinates of the cursor and store the value into the GXlocal, GYlocal (or ZXlocal, ZYlocal if the cursor is in the zoom window) global variables, which will then be displayed in the upper right hand corner of the global (or zoom) window. Thus the location of the cursor will be constantly tracked so that the user can immediately see the cursor coor-
ordinates of the microscope image. This binding is setup regardless of whether or not the client is in control of the microscope, for every member of the conference inspection should have the ability to track his/her own cursor movement on the microscope image.

### 9.4.2 Mouse button bindings

The mouse buttons bindings for the global and zoom images however are only activated when the client is in control because only the controller of the microscope should have the ability to place the pointer and rectangle visual aids on to the canvas. The rectangle and pointer objects are placed onto the canvas by calling either the put_cursor subroutine or the put_rectangle subroutine. These procedures are prototyped below:

```tcl
proc put_cursor {x y which mode} proc put_rectangle {x1 y1 x2 y2 which mode}
```

- `x,y`: specifies the coordinates on the canvas to place pointer
- `x1,y1,x2,y2`: specifies two corners of rectangle to be placed on canvas
- `which`: for both cases specified which window to place object. Can be GLOBAL, ZOOM, or NONE.
- `mode`: for both cases specifies whether the object should be forwarded to the server. NETWORK mode forwards result to server so that all clients get result. LOCAL mode does not report result to server

**FIGURE 28. Prototype and description for put_cursor and put_rectangle Tcl/Tk client routines.**

Clicking on the image with the left mouse button will initiate the "put_cursor" subroutine, which will place a visual aid pointer at the cursor location and copy the coordinates into the CURXPOS and CURYPOS global variables. By calling the "put_cursor" procedure with a "NETWORK" argument, (which is the usual case) the subroutine will also send the latest pointer position back to the server with a "CURPOS" message. If the pointer is placed in a window whose current magnification is less than the new magnification setting of the microscope, the "put_cursor" routine will also initiate the "put_rectangle" subroutine in order to place a rectangular box in the image outlining the region which will be
zoomed into if the "GRAB" or "ZOOM" button is activated. When the rectangular box is placed in the canvas, the Tcl/Tk client will also send a "RECTANGLEPOS" message to the server.

Clicking the middle mouse button allows a user to place any arbitrary size rectangle on the canvas. The user first clicks the middle button once to set up an upper left hand corner anchor point for the rectangle by setting the tempx1 and tempy1 global variables and also setting a "BUTTONDOWNFLAG" to indicate that the middle mouse button has been pressed once. Any subsequent motion of the mouse will then cause a new rectangle (after deleting the old rectangle) to be drawn between the anchor point and the current cursor location, thus allowing the user to effectively stretch and shrink the rectangle to any size. When the middle mouse button is clicked again, the "BUTTONDOWNFLAG" is reset, and a final rectangle is placed on the canvas. However, "the put_rectangle" subroutine will then be called with a "NETWORK" argument, which would cause it to forward a "RECTANGLEPOS" to the server to send the updated rectangle information to each client. Finally, clicking the right mouse button erases the rectangle visual aid, and causes a "RECTANGLEPOS" message to be sent to the server with rectangle located at (0,0) (0,0) in the "NONE" window.

9.4.3 Bindings criteria

The mouse bindings generally occur each time a new image is loaded into either the Global or Zoom window of the interface. The mouse motion is always bound to the GXlocal and GYlocal variables, but the mouse click buttons are only bound if the client currently has control of the microscope. The mouse bindings are also only valid when the cursor is on top of a bitmap image. As a result, the binding must be reestablished whenever a new microscope image is loaded because the old bitmap image must be deleted, thus destroying the bindings, before the new image is loaded. Also the mouse click bindings should be broken whenever the user cedes control of the microscope, and conversely any client that takes control of the microscope should have the bindings reinstated.

9.4.4 Visual aid coordinate system

As described earlier, the remote microscope client interface utilizes a global coor-
ordinate system that maps each location on the wafer (more specifically, each pixel on the X term display when the magnification is at 1000X) to a specific coordinate, so that messages such as "GRABFRAME" or "GRABFRAMEREPLY" containing coordinate information that must be transmitted between client and server can describe any wafer location uniquely. Specifically, the CURXPOS, CURYPOS, GX, GY, ZX, ZY, GXlocal, GYlocal, ZXlocal, an ZYlocal variables all contain values based on this coordinate system. For example, when the user decides to place a pointer visual aid on the canvas, the CURXPOS and CURYPOS variables will be assigned the global coordinate corresponding to that location of the wafer, which will be sent as the pan coordinates for the next microscope capture.

However, a local coordinate system (640x480 grid) is used to actually position the pointer or rectangular visual aids within the canvases that hold the global or zoom microscope images. For example, when a user clicks the left mouse button over an image displayed in the global or zoom window (at any pan position or magnification setting), the Tcl/Tk software will have access to the x-y pixel coordinate, where the upper left hand corner is (0,0) and the lower right hand corner is (639,479). Because this local coordinate is the most straightforward one to use to manipulate objects in the canvas, the "CURPOS", "RECTANGLEPOS", "CURPOSREPLY", and "RECTANGLEREPLY" messages all reference the visual aid positions with respect to this 640x480 grid. However, once the visual aids locations need to be interpreted as a wafer position, then the software must perform a transformation between the coordinate systems. (Fig 29) For example, after a pointer is placed in the global window for example, the software must transform the pixel coordinate into the global coordinate system before setting the CURXPOS and CURYPOS values, which reflects the position of the pointer on the wafer.

\[
x_{\text{wafer}} = GX + x \times \left(\frac{1000}{GM}\right)
\]
\[
y_{\text{wafer}} = GY + y \times \left(\frac{1000}{GM}\right)
\]

FIGURE 29. Transformation between pixel coordinates of image and global coordinates of wafer. (case shown for point in Global window, similar for Zoom window).
9.5 Handling Dynamic Messages From Server

As described earlier, the client interface is responsible for processing user requests, but in addition it must also be able to respond to messages sent by the daemon process (but usually originating from the server process). In fact, immediately after the client software is first run and the user requests a connection to the remote microscope, the client must wait for a “CONNECTREPLY” message before continuing. After receiving the connection confirmation, the client then performs a set of initialization steps to setup the three windows that make up the microscope interface, and then enters an event loop where the process waits for a user or message event to occur. The possible user events were described in the previous section, so the following discussion will be based solely on message events arriving from the server. The following state transition diagram models the Tcl/Tk client response to incoming message events from the server.
FIGURE 30. Model of Tcl/Tk Client handling incoming messages from daemon (originating from Server running on PC).
**TABLE 12.** Messaging resulting from incoming messages from daemon (originating from server) shown in Fig 30.

9.5.1 "GRABFRAMEREPLY" and "ZOOMGRABREPLY" messages

The "GRABFRAMEREPLY" and "ZOOMGRABREPLY" commands are the most important messages that the client receives from the server because they indicate when a new microscope image has arrived. As described earlier, the server first passes a similar message that includes the actual image data to the daemon, which then saves the image to file and notifies the Tcl/Tk client that a new picture is available. When a "GRABFRAMEREPLY" message is finally received by the Tcl/Tk client for example, the following script is executed:
switch $message { ......

GRABFRAMEREPLY {
    puts "Now handling GRABFRAMEREPLY"
    set GX [object_slot_get $name XPOS]
    set GY [object_slot_get $name YPOS]
    set GM [object_slot_get $name MAG]
    .global.c delete $gcanvasimage
    put_cursor 0 0 NONE LOCAL
    set gcanvasimage [global.c create bitmap 0 0 -anchor nw -bitmap 
        @grabframe.gif]
    .global.c bind $gcanvasimage <Motion> {
        set GXlocal [xyprime $GX %x $GM 1000]
        set GYlocal [xyprime $GY %y $GM 1000]
        if {($BUTTONDOWNFLAG == "true") && ($CONTROLSTATUS == "ON")} {
            put_rectangle $tempx1 $tempy1 [expr 10+%x]
            [expr 10+%y] GLOBAL LOCAL
        }
    }
    if {$CONTROLSTATUS == "ON"} {
        .global.c bind $gcanvasimage <Button-1> {
            put_cursor %x %y GLOBAL NETWORK
        }
        .global.c bind $gcanvasimage <ButtonPress-2> {
            if {$BUTTONDOWNFLAG == "false"} {
                set tempx1 %x
                set tempy1 %y
                set BUTTONDOWNFLAG true
            } else {
                put_rectangle $tempx1 $tempy1 [expr 10+%x]\n                [expr 10+%y] GLOBAL NETWORK
                set BUTTONDOWNFLAG false
            }
        }
        .global.c bind $gcanvasimage <Button-3> {
            set BUTTONDOWNFLAG false
            #this was in case run off screen, and then hit 3
            put_rectangle 0 0 0 0 NONE NETWORK
        }
    }
    catch {destroy .busy}
}

...... }

FIGURE 31. Code excerpt showing Tcl/Tk client execution after "GRABFRAMEREPLY" message is received from Daemon.
The client must retrieve the new GX, GY, GM values (which correspond to the X pan position, Y pan position, and magnification of the new image), and then to create a new bitmap graphic object that is loaded from the file "grabframe.gif" into the global canvas. As described in the previous section, a set of bindings for the mouse buttons that allow the user to place visual aids onto the canvas must also be setup whenever a new image is received. The "ZOOMGRABREPLY" message is again very similar to the "GRABFRAMEREPLY" except that the new image will then be placed in the Zoom window.

9.5.2 "ZOOMGRABREPLY" and "GRABFRAMEREPLY" messages

The client can also receive "CURPOSREPLY" or "RECTANGLEREPLY" messages that are originally sent from the server, which cause the client to place a pointer or rectangle, respectively, at the location specified within the message. These messages are important for the server to make sure that all clients in a conference inspection will have an updated view of the pointer and rectangle visual aids.

9.5.3 "USERLISTDELETE" and "USERLISTADD" messages

The "USERLISTDELETE" message causes the Tcl/Tk client to remove all the names of the current users of the microscope from the console list. This message is sent whenever one of the clients in a conference inspection dies, and is usually followed by a sequence of "USERLISTADD" messages that adds the names of the current users back to the list one at a time. The reason for removing the entire list of users when one client dies is because we would like to centralize all the record keeping to be at the server end, so that there would be no need for each client to explicitly keep track of all users of the microscope, and the client would never need to search the user list to find a specific user to delete. By erasing the whole user list and replacing it with an entirely new list whenever a client disconnects, the record keeping involved is very straightforward. The only drawback in this method is that there is a network delay by retransmitting this data, but considering the fact that clients disconnect so infrequently and the delay times for transmitting the data is negligible compared to human time scales, this approach worked very well.
9.5.4 “SETSTATE” message

The final message that the Tcl/Tk client can receive is the “SETSTATE” message, which is important for placing clients into the “CONTROL”, “PASSIVE”, or “NONE” states. When new clients first start to connect to the remote microscope, the server initially must set the state of each new clients to be “NONE”, where no one is in control. When a particular user finally decides to take control of the microscope, the server must then set the state for the initiating client to be “CONTROL” while placing all other clients in the “PASSIVE” mode. Any new clients connecting to the server are then automatically placed into a “PASSIVE” mode. If for example the controlling client decides to give up control or disconnect from the system, then the server must set every client’s state to be “NONE” again, since no one will have control of the microscope.

The server must broadcast a “SETSTATE” message to all the clients connected to the system whenever a user disconnects or decides to take or cede control from the microscope. When one client is able to cede or take control, or if the client in control is terminated, then the state of every user of the microscope must change, so the server must broadcast a “SETSTATE” message to everyone. If a client not in control is terminated though, there isn’t a real need to re-broadcast the “SETSTATE” message to everyone, but we still update the state of every client simply because this is easier than checking to see if the terminated client was the client in control. Whenever a new client tries to connect to the microscope, the server simply needs to reply with a single “SETSTATE” message to orientate the new user, since a new connection has no impact on the state of other users.

The “SETSTATE” message thus is very important to establish the modes for the microscope users to ensure that only one person is ever in control of the microscope at a time. When a client receives a message to enter the “NONE” state, it must disable all of the console buttons except the “TAKE”, “GSAVE” and “ZSAVE” control buttons. It must also break any bindings that may have been set up for the mouse buttons on the canvas. For example, this would be important when a client in control (who currently is able to use the mouse buttons within the canvas) decides to cede control. The client will remain in control until the server explicitly sets the client state to be “NONE” in which case the bindings should then be broken.
When a client is instructed to enter the "PASSIVE" state, all of the console buttons should be disabled (except for the "GSAVE" and "ZSAVE" obviously) since a passive viewer ought not to be able to control the microscope. In the client code, we specifically destroy any current bindings, although this really was unnecessary because the "PASSIVE" state can only be entered from being in the "NONE" state, which has already removed any mouse bindings.

Finally, when a client is instructed to enter the "CONTROL" state, all the console buttons must be reactivated (except the "take" control button), so that the user can have full operation of the microscope. Also, the mouse bindings need to be reestablished as well in order to permit the user to place the pointer and rectangular visual aids onto the canvas. The following state transition diagram (figure 32) shows how the state of a particular client can change based on server instructions. Note that a newly connecting client will initially be placed in either the "NONE" state or "PASSIVE" state depending on whether or not anyone already has control of the microscope.

![State Transition Diagram](image)

**FIGURE 32.** State transition diagram showing the control state of a client. Transitions only occur due to a "SETSTATE" message sent by Server.
Figure 33 shows the state transition diagram of the microscope from the server’s point of view, where either someone is in control or no one is in control of the microscope. The remote microscope system can toggle between these two states when the server receives a “CEDECONTROL” or “TAKECONTROL” message, or if the client in control is terminated. During these transitions, the server must then issue “SETSTATE” messages back to each client, which can then affect the client state as specified in figure 32. The server also produces a “SETSTATE” messages whenever a new client connects, and depending on whether or not someone has control of the microscope, will set any newly connecting client to be placed initially in the “NONE” or “PASSIVE” state.

FIGURE 33. State transition diagram showing the server state. Changes in state occurs only when a client takes or cedes control, or if client in control terminates.
Chapter 10

Possible Improvements and Future Work

The remote microscope developed here at MIT was only a first generation system to demonstrate the feasibility of a remote microscope. Although the system was functional and can already be used effectively to help provide remote inspection of semiconductors, there are still many improvements that we will try to make in the future. Work on the remote microscope is continuing to progress, and we feel that we will be able to improve the effectiveness of this very useful tool.

10.1 Automation

First of all, we will be developing a version of the remote microscope where the microscope is actually controlled by the PC, since currently a person has to manually set the magnification, pan and focus of the microscope whenever the server requests a new image. We in fact have already looked into purchasing an automatic microscope system which can be fully controlled by a personal computer, thus automating the microscope operations on the server side. This is a major requirement in developing a true remotely controlled microscope and will definitely be incorporated into the next generation remote microscope.
10.2 Speedup

Another future goal is to improve the performance/delay time during new image captures. Currently, there is generally a delay greater than 30 seconds between when a user requests a new microscope image and when he/she finally receives the updated picture. Although tolerable, this delay should be reduced considerably in order to make the remote microscope more effective. The delay consists of several factors including the network delay, latency due to physically positioning the microscope, and delay due to converting the image file from a TGA format to a GIF format. The network delay is only on the order of two to five seconds (depending on network load) to transfer a 100K GIF file between Stanford and MIT and the time to position the microscope should also be rather short. The real cause for the delay time is due to the very time consuming conversion process between the TGA format provided by the framegrabber board and the GIF format utilized by the Tcl/Tk client. There are several solutions to this image conversion process that we will be looking into. First of all, it might be possible to find another set of image conversion utilities that run under OS/2 that will speed up the conversion process, or we might write our own conversion routine to optimize the execution time. It might also be possible to choose a different frame grabber board that can produce different image formats or possibly perform hardware compression of the data. We are confident however that significant improvements can be made to speed up this currently time consuming conversion process.

Although Amdahl’s law would advise us to put our greatest efforts into speeding up the slowest part of the system (conversion routine) in order to improve overall performance, we have been able to isolate a few other sections of the program that can be optimized to improve the overall delay between requesting and receiving a new microscope image. For example, it might be possible to improve the network delay performance by using more image compression to reduce the image size. For example, we could use a graduated lossy compression scheme that allows the user to specify the amount of compression to perform before each new image capture request. This can be very useful when the user does not need to view the microscope with a high resolution, but just needs to see enough of the general details to be able to position the wafer to a region that he/she is
more interested in. If we decide to use this approach, the compression algorithm might even be applied directly to the TGA file produced by the Targa+ board in order to bypass the previous conversion step to a GIF format for example.

Finally another way to speed up the time to capture and transmit a new microscope image would be to incorporate the image capture program and the server program into a single executable file. As described earlier, the capture utility had to be a completely different program because the Targa+ tools for OS/2 could compile only under CSet++, while the server’s CHAT routines could only be compiled using GCC. The capture utility saves a TGA file to disk, which is then read by the server process in order to transfer the image data into the computer’s memory. If we combine the capture and server processes into a single process, then it might possible to directly load the TGA image into memory that the conversion or compression routines can access immediately. Currently, having to fork off the capture process from within the server program, and storing and retrieving the image from disk adds only on the order of one second to the total delay time. Although the delay is significant, it is still small compared to the conversion delay that limits the performance of the remote microscope.

10.3 User Interface

After experimenting with the remote microscope and demonstrating and receiving feedback from users at the Stanford-MIT collaboration demo, we decided that there were improvements that could also be made to the microscope interface. For example, it was suggested that it would be extremely helpful to provide a “ruler” for the remote microscope so that a user could measure semiconductor features in microns. This could be incorporated into the next generation microscope without much difficulty, and would greatly improve the effectiveness of this tool. Another suggestion was to use two different types of rectangular visual aids in the interface: one programmable rectangle that the user can place anywhere, and a another rectangle that always displays the region within the panoramic wafer view that is magnified. Currently a single rectangle plays both roles, so that the user can either place the rectangle anywhere on the canvas or have the rectangle highlight a magnified portion of the wafer, but cannot do both. Finally, passive clients viewing the microscope during a conference inspection currently do not know when the
client in control has performed an image capture and is waiting for a response. It would be useful for the passive viewer to be notified whenever the remote microscope is busy.
Chapter 11

Conclusion

The remote microscope developed at MIT was a first generation system to test the feasibility of remote inspection of integrated circuits. At the 1995 TCAD symposium and other demonstrations of the remote microscope, the overwhelming response to this telemicroscopy system was very positive, so we feel that it will become a valuable tool in the future to assist during remote fabrication of semiconductor circuits. In actuality, the remote microscope is a general purpose tool that can easily be utilized in fields outside semiconductor processing such as the telemedicine branch of the medical field.

To the best of our knowledge, the MIT remote microscope was the first system ever to provide telemicroscopy over the internet. Although it is common place to transmit images over the internet, and the medical community has already made great strides in developing expensive remote microscopes utilizing custom communication links, the MIT remote microscope was the first system to combine the flexibility of the internet with the usefulness of telemicroscopy. The result is that the MIT remote microscope is extremely portable, cost effective, and versatile, providing a very practical solution for providing remote microscopy to the scientific community. By implementing the remote microscope with readily available hardware including an ordinary PC running OS/2, the remote microscope can also be reproduced at a very low cost. More importantly, the microscope clients
are virtually cost free because the interface is simply a software package written at MIT to be run on an ordinary UNIX workstation.

Although improvements to the system are still in progress, the current remote microscope already can be used effectively to inspect wafers during remote fabrication, which we feel to be a growing trend in the semiconductor processing field. The remote microscope was also implemented with collaboration in mind, since multiple users can simultaneously examine the microscope in a conference mode. As a result the remote microscope can be used effectively as a collaboration tool to help researchers work together during semiconductor processing. Because we feel the trend in future semiconductor processing will involve more sharing of resources and collaboration of expertise, we envision the remote microscope to be a valuable tool for researchers in the future.
References


Appendix A

Using the Remote Microscope Software Package

In order to set up the remote microscope, the user should initially run the “reset.cmd” batch file on the PC. This batch file initializes the Targa+ board, placing it in “live” mode so that the Targa+ video out shows the live microscope image, and performs an initial frame capture. This initial image will be used to choose 200 colors to quantize all further microscope images, so it is important to choose an initial mapping to reflect the type of images that will be displayed. The reason 200 constant colors were chosen is because most X terminals that display the microscope images would run into color map problems if too many different colors were being loaded during each new microscope frame. In a future enhancement of this system, we will look into a method of refreshing the color map used by the Tcl/Tk graphical interface in order to avoid this problem.

The server program is called by typing “server [ADDRESS] [PORT] QUERY/AUTO,” where the last argument determines whether or not the server process should pause before performing the next frame capture. For example, using the “QUERY” option will give an attendant at the microscope a chance to position the wafer before the microscope image is captured, whereas “AUTO” will cause the server to perform a frame capture whenever a client sends the appropriate message.

Setting up each microscope client involves running the Tcl/Tk client interface and
the associated daemon process. Currently each of these programs needs to be run in its own xterm window, mainly in order to display debugging output, although in future releases of the software we may spawn the daemon process directly from the Tcl/Tk client program. The Tcl/Tk client interface process needs to be run first because it was designed to wait for an incoming daemon connection. The Tcl/Tk client is executed by first running “wish” (a windowing shell that includes Tcl/Tk commands), and then typing “source package.tcl” at the prompt. The “package.tcl” file is a short script that loads several files that are needed to run the client interface, and also has a line that sets up the port number to be used by the daemon. After the Tcl/Tk package is loaded, then the client daemon should be run in a different window by typing “client [HOSTNAME] [PORT].” The hostname is simply the name of the computer that the Tcl/Tk client and daemon are running, and the port number is simply the one specified in the “package.tcl” script. Immediately after the daemon is run, the Tcl/Tk client interface should display a window for the user to input the server address and port number.

The following files are needed to compile the server software on the PC using GCC, which should produce a “server.exe” executable after running the makefile. The CHAT tools used to implement the server program were versions of the UNIX CHAT tools ported to OS/2. This following table briefly describes each component.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>Makefile to produce “server.exe”</td>
</tr>
<tr>
<td>basic_sockets.h</td>
<td>OS/2 CHAT basic sockets header file</td>
</tr>
<tr>
<td>basic_sockets.c</td>
<td>OS/2 CHAT basic sockets routines to provide socket connections.</td>
</tr>
<tr>
<td>com_handler.h</td>
<td>OS/2 CHAT communication handler header file</td>
</tr>
<tr>
<td>com_handler.c</td>
<td>OS/2 CHAT communication handler routines to provide event handling etc.</td>
</tr>
<tr>
<td>object.h</td>
<td>OS/2 CHAT object header file</td>
</tr>
<tr>
<td>object.c</td>
<td>OS/2 CHAT object routines to provide object message data abstraction</td>
</tr>
<tr>
<td>record.h</td>
<td>Remote microscope record-keeping package header file</td>
</tr>
<tr>
<td>record.c</td>
<td>Remote microscope record-keeping package routines.</td>
</tr>
</tbody>
</table>

**TABLE 13. Server package**
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server.h</td>
<td>Remote microscope server header file</td>
</tr>
<tr>
<td>server.c</td>
<td>Main remote microscope server code</td>
</tr>
</tbody>
</table>

**TABLE 13. Server package**

The image capture software is a separate program which is compiled using CSET/2, and utilizes the Targa+ compatible toolkit for OS/2. The following table describes the code elements more carefully.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>Makefile to produce “caprgb.exe”</td>
</tr>
<tr>
<td>caprgb.c</td>
<td>Main code for image capture program</td>
</tr>
<tr>
<td>tardev.h</td>
<td>Targa+ Compatible toolkit header</td>
</tr>
<tr>
<td>targaos2.lib</td>
<td>Targa+ Compatible toolkit library routines</td>
</tr>
</tbody>
</table>

**TABLE 14. Image capture package**

The Tcl/Tk code consists of several scripts that need to be loaded into the wish shell, but the user only needs to source the “package.tcl” file. All of this code (including the CHAT routines) were written using Tcl. Although currently the client software consists of files that need to be sourced into a wish shell, later versions of the code most likely will be launched directly from the UNIX prompt.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package.tcl</td>
<td>Short script that sources all necessary files into wish shell</td>
</tr>
<tr>
<td>basic.Sockets.tcl</td>
<td>Tcl CHAT basic sockets routines to provide socket connections.</td>
</tr>
<tr>
<td>com_handler.tcl</td>
<td>Tcl CHAT communication handler routines to provide event handling etc.</td>
</tr>
<tr>
<td>object.tcl</td>
<td>Tcl CHAT object routines to provide object message data abstraction</td>
</tr>
<tr>
<td>winwindowinit.tcl</td>
<td>Remote microscope client interface widget initialization</td>
</tr>
<tr>
<td>clientele.tcl</td>
<td>Remote microscope client interface main program (dynamic)</td>
</tr>
</tbody>
</table>

**TABLE 15. Tcl/Tk client interface package**
Finally, the client daemon was implementing using C and the UNIX version of the CHAT utilities with GCC. The client daemon is always run in conjunction with the Tcl/Tk client interface.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>Makefile to produce “client.exe”</td>
</tr>
<tr>
<td>basic_sockets.h</td>
<td>CHAT basic sockets header file</td>
</tr>
<tr>
<td>basic_sockets.c</td>
<td>CHAT basic sockets routines to provide socket connections.</td>
</tr>
<tr>
<td>com_handler.h</td>
<td>CHAT communication handler header file</td>
</tr>
<tr>
<td>com_handler.c</td>
<td>CHAT communication handler routines to provide event handling etc.</td>
</tr>
<tr>
<td>object.h</td>
<td>CHAT object header file</td>
</tr>
<tr>
<td>object.c</td>
<td>CHAT object routines to provide object message data abstraction</td>
</tr>
<tr>
<td>client.c</td>
<td>Remote microscope daemon main program</td>
</tr>
</tbody>
</table>

**TABLE 16. Client daemon package**
Appendix B

Hardware Costs

The following is a detailed list of the hardware equipment we needed to purchase to implement the remote microscope system. Prices are from February, 1995 and are compared between two video suppliers. (Note that Crimson Tech prices are lower only because we asked them to meet and/or beat the competitor prices. In fact, our initial quote from Crimson Tech was more expensive.)

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Costa</th>
<th>Comparisonb Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC Related</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dell XPS-P75 Personal Computer</td>
<td>1965</td>
<td>NA</td>
</tr>
<tr>
<td>3Com Etherlink III Card</td>
<td>116</td>
<td>NA</td>
</tr>
<tr>
<td>Subtotal</td>
<td>2081</td>
<td>2081</td>
</tr>
<tr>
<td>Frame Grabber Related</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Targa+ 16/32 Image Capture Board</td>
<td>1250</td>
<td>1399</td>
</tr>
<tr>
<td>Truevision Developer’s Tool Kit</td>
<td>134</td>
<td>139</td>
</tr>
<tr>
<td>CCXC9DD 9 pin to 9 pin camera cable</td>
<td>70</td>
<td>98</td>
</tr>
<tr>
<td>Panasonic CT1384Y 13&quot; color monitor</td>
<td>324</td>
<td>315</td>
</tr>
<tr>
<td>Subtotal</td>
<td>1778</td>
<td>1951</td>
</tr>
</tbody>
</table>

TABLE 17. Remote microscope related equipment