On-Line Laboratory for Remote Polymer Crystallization Experiments Using Optical Microscopy

by

Daniel J. Talavera

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 21, 2003

Author_____
Department of Electrical Engineering and Computer Science
May 21, 2003

Certified by_____
Gregory C. Rutledge
Associate Professor of Chemical Engineering
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

On-Line Laboratory for Remote Polymer Crystallization Experiments Using Optical Microscopy

by

Daniel J. Talavera

Submitted to the Department of Electrical Engineering and Computer Science

May 21, 2003

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis discusses the architecture of an on-line optical microscopy laboratory, or iLab, in which students remotely conduct and analyze polymer crystallization experiments using a polarized light microscope under controlled temperature conditions. The Polymer Crystallization iLab involves melting a polymer sample and subsequently cooling it down to a temperature below its melting point in order to study isothermal crystallization phenomena. By analyzing the rates of nucleation and crystallite growth, students can characterize the kinetics of crystallization. As melting the polymer erases any prior history of the sample, the experiment can be repeated numerous times without requiring intervention in the laboratory. The architecture was designed with the goal of replicating the real laboratory experience to the maximum extent possible. Streaming temperature data and images from the microscope are sent to a Java applet, allowing the student to view and interact with the experimental apparatus in real time. The Java applet client runs on any conventional web browser and provides considerable latitude to students conducting the experiment, while ensuring proper safeguards. Students can record and save images and related data to a server to perform analysis at a later date. The analysis can either be conducted remotely on the server or the images can be downloaded to the user's computer for local analysis

Thesis Supervisor: Gregory C. Rutledge

Title: Associate Professor of Chemical Engineering

Acknowledgments

This project would not have been possible without the help and guidance of Professor Gregory C. Rutledge. He offered me the opportunity to work on the project and allowed considerable latitude form me to explore the various technologies used to implement the iLab. He was always supportive of and engaged in my work. This project would not have been possible without his direction and encouragement.

I will also like to thank my parents: Alicia M. Talavera and Miguel A. Talavera and my brothers, Jason A. Talavera and Jonathan M. Talavera, who have supported and encouraged me throughout all of my educational endeavors. My extended family has also provided me with guidance and backing throughout my years at MIT. I would not have made it through five years without their support and their love. The non-exhaustive list includes Alicia Beteta, Maria & Jack Stone; Anna, Robert, & Stephanie Sampera; Leticia, Alison, & Joseph Campos; Adela & Erik Delong; Aye Hnin, Vilma & Luis Talavera.

I would also like to thank all of the groups and persons at MIT, with whom I collaborated during this project, including Jud Harward and the iLab Group, Jesus del Alamo and the WebLab Group, Steven Lerman and the CECI Group, the iCampus Group, and the Rutledge Research Group.

Finally, I would like to thank all of my friends at MIT and in San Francisco who have supported me through my endeavors. I would especially like to thank Milena B. Yamaykina who has been supportive and invaluable during the toughest moments of this project.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Purpose and Motivation

Because students learn more effectively through experimentation and collaboration, laboratory experience is a key component of many science and engineering degree programs. However, the high cost of equipment and personnel necessary to maintain a functional laboratory has introduced a cost barrier for colleges and universities which seek to maximize the educational experience of students. The paradigm of Internet laboratories, or iLabs, offers an educational model that maximizes the utilization and accessibility of expensive equipment by allowing students to conduct experiments through a web-based interface. By facilitating access to experimental apparatus, iLabs allow students to explore and engage physical phenomena. By slightly varying the parameters of each experiment run, students can efficiently and inexpensively follow their curiosity to better understand underlying principles in their discipline. Unlike conventional labs, iLabs enable students to conduct experiments around the clock with no need for constant lab monitoring by a professor or TA. Internet laboratories also allow professors to integrate laboratory experiments into a lecture setting, offering an enhanced visual aid for students. Additionally, universities can reduce costs by sharing Internet laboratories, thus distributing costs among universities and increasing students' experimental experience.

This paper focuses on the development and architecture of the Polymer Crystallization iLab, a remote Internet laboratory for conducting polymer crystallization experiments using optical microscopy. This standard experiment in undergraduate polymer science involves viewing the isothermal crystallization of polymers through a polarized light microscope. The polymer crystallization experiment has been difficult for students from MIT and other universities to access because of the limited availability of polarized light microscopes with the requisite heating stage and photographic equipment. The Polymer Crystallization iLab solves this cost barrier by multiplexing a single setup that allows users to remotely control a heating stage, a motorized microscope, and an image capture device. With these capabilities, students conduct the experiment and acquire digital images using optical microscopy. Through the use of digital image analysis, students can then analyze their data to derive the kinetic crystallization properties of a specific polymer.

For her Masters of Engineering Thesis, Paola Nasser developed a Remote Microscope consisting of a polarized light microscope and a digital camera controlled via a Java Applet on the local machine [1]. The client had capabilities to display images captured by the digital camera with the capability to view video at a rate of one frame every six seconds. Additionally, the client could adjust the light, objective, and polarizer settings of the microscope. This Remote Microscope provided a base for the implementation of the full Polymer Crystallization iLab.

This thesis expands upon Nasser's work to enable the user to control the microscope remotely on any platform. The video stream has been improved to support real-time streaming images at speeds of up to two frames per second. In addition, capability has been added to control a heating stage, the XY position, and the focus of the polymer sample, and to save experimental runs on the server. An on-line environment enables users to analyze images after experiments have been run and saved on the server. Finally, a reservation mechanism allows users to reserve the microscope for the time needed to complete their experiments.

1.2 Background

1.2.1 Polymer Crystallization Experiment

The polymer crystallization experiment involves heating a sample of polymer above its melting point and subsequently cooling it to various crystallization temperatures. Analysis of isothermal crystallization events yields an observable rate of nuclei formation and rate of growth for each crystallite. The rate of growth vs. crystallization temperature allows users to experimentally determine such properties as the activation energy, the fold energy for the growing crystals, and the Avrami exponent of thin film crystallization. Students can then relate the measured crystallization properties to their study of theoretical polymer crystallization kinetics and behavior [2].

The polymer crystallization experiment is well suited to an online environment because the experiment is hands-free and "memoryless." Once the sample is set, the experiment can be cycled without local intervention. In addition, once the sample has been melted, it has been essentially "reset" in the sense that prior experiments will not significantly impact future experimental results. Thus, students can conduct and repeat experiments without any major impact from each other's prior iterations.

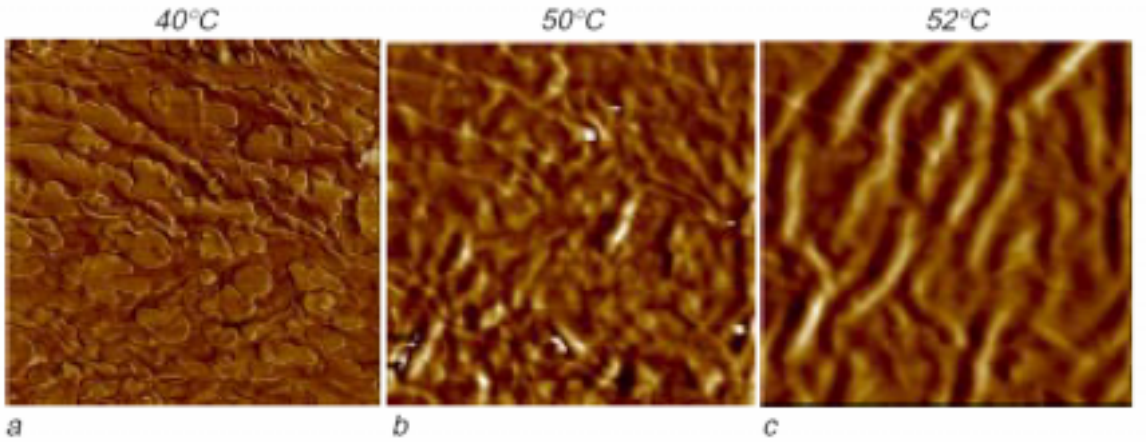


Figure 1: Isothermal crystallization of PEO (a) 40°C, (b) 50°C, and (c) 60°C.[3]

1.2.2 Related Work

There have been a number of independent projects initiated to develop a remote microscope for different applications. In 1996, James Kao [4] developed the Internet Remote Microscope to aid in the remote fabrication of integrated circuits as part of the Computer Integrated Design and Manufacturing project. This microscope took a single snapshot of an integrated circuit and sent it to multiple clients who could remotely inspect the IC and could confer with each other using an online chatting interface. This system was developed further by Somsak Kittipiyakul, who completed the automation of the Kao microscope [5]. Although a good start, this system was characterized by high latency in commands to change the microscope settings and to refresh the captured image.

Another project at MIT developed a remote, automated microscope for characterizing micro-electromechanical systems (MEMS). The MEMS project at MIT allowed MEMS designers to analyze the three-dimensional motion of a MEMS device during development and testing. The system, built for the MEMS group at MIT, was an improvement over the Internet Remote Microscope; however, it did have shortcomings which were noted by Daniel Seth in his Masters Thesis in 2001 and are summarized here [6]. The system used HTTP to pass messages from the client to the server, which required that the client initiate all communication. Thus a polling mechanism was instituted, which continually queried the server to see if the output was ready. This mechanism was far less efficient and scalable than full duplex communication. Another pitfall was the inability to manipulate, crop, mark, filter, or save images on the server in any modified format. The primitive video streaming simply used a single file on the server, which was continually refreshed by the client and overwritten by the server, introducing an inherent race condition. Finally, there were no security measures in place to ensure that users' data was not compromised and, moreover, that the system was not tampered with by unauthorized users.

Outside of MIT, there has been an ongoing, open-source project by the MEMS Exchange

[7] which is supported by the Defense Advanced Research Projects Agency (DARPA) and hosted by the Corporation for National Research Initiatives (CNRI). This project seeks to increase access to MEMS micro-fabrication resources and establish a distributed MEMS fabrication environment, organizing and connecting designers to the MEMS micro-fabrication resources located throughout the country. This project uses a simple but well-defined protocol of asynchronous ASCII text messages between the server and client to set the state of the microscope and request images from the mounted camera. Developed by A.M. Kuchling [7], this Microscope Networking Protocol Specification formed the basis for Paola Nasser's Remote Microscope.

1.2.3 iCampus Framework

iCampus is an alliance formed in 1999 between MIT and Microsoft to enhance university education through information technology. iCampus sponsors iLabs in a number of disciplines including microelectronics, mechanical engineering, thermodynamics, and civil engineering. A project sponsored by iCampus, the Framework Project, has attempted to abstract the common services used by the iLabs to prevent each iLab from having to implement redundant systems. Led by two Microsoft engineers visiting MIT, Dave Mitchell and Eric Carlson, the Framework Project sought to provide these common services in a number of compact modules, such as the Identity Service and Storage Service. The Identity Service and the Storage Service provided a functional abstraction for creating a user management system and storage management system, respectively. The Framework Project has since been discontinued; however, the iLab Project at MIT has taken over the responsibility for designing and implementing a shared architecture for the common services needed by iLabs.

Each of the iLabs under development represents a distinct set of experimental requirements which must be classified and addressed by the iLab Group. The Polymer Crystallization iLab serves as a prototype of an interactive experiment for the development of the iLab Project's shared architecture. During this experiment, the Microscope Server must supply the user with real-time status updates in the form of

streaming temperature readings and streaming video. This class of experiment is intended to closely model real-world laboratory conditions, where the user can interact with the experimental apparatus in real-time. In contrast to this paradigm, the Microelectronics WebLab [8] represents a "stateless" experiment where the parameters are batched into a command script which is used to run the experiment. Because of the relatively small time scale in electronic measurements, there is no user interaction during a "stateless" experiment. The user's job is to set up the experiment and let the experiment run without any user involvement.

1.3 Development

This section provides an overview of the programming languages and integrated development environments (IDE) used for the implementation of the various software components of the Polymer Crystallization iLab. Each language was chosen for its benefits to each specific module.

1.3.1 Java

The Microscope Client is currently implemented as a Java Applet. Java was initially chosen for Nasser's design because of its object-oriented structure, platform independence, and availability in most popular web browsers. Because the system uses full duplex communication between the client and server, a client program must be able to decipher messages sent over primitive sockets. Because Java Applets can be embedded into web pages, use of Java allows clients to run the applet from anywhere without having to download a full client application. The Java Virtual Machine (JVM) required by the Polymer Crystallization iLab is Java 1.4. Since many web-browsers do not come with Java 1.4, users will have to download the Java Plug-in supplied by Sun Microsystems.

The Microscope Client was developed using the Sun Open Net Environment (Sun ONE). This integrated development environment contains a number of coding, compiling, and debugging tools as well as a graphical Form Editor, which allows the developer to easily

manipulate and preview visual Java components.

1.3.2 Python

The Microscope Server and all associated hardware controllers are implemented in Python. Python is an object-oriented language that has a number of freely available modules to control everything from serial ports to image manipulation. Because it is written entirely in C, it is very efficient at string manipulation and dictionary management. Python allows communication with the hardware modules via serial ports and Microsoft's Component Object Model (COM), through simple yet efficient modules available in the Win32 package. Another available library, the Python Imaging Library (PIL), allows images to be manipulated by the Microscope Server.

In addition to the abundance of available modules, Python also comes with a small but useful IDE called PythonWin. PythonWin allows single line commands to be sent to the Python interpreter for easy testing of software components. It also contains tools for managing and debugging code.

1.3.3 C#

The Framework Server has been implemented using C# and Microsoft's new .NET Framework. The Framework Server, named after its origin in the Framework Project, is used for post-experimental data analysis and user management. C# was the language chosen for this component because of its seamless integration into ASP.NET web pages and its simple database access mechanism. Microsoft's .NET platform is an efficient and highly scalable environment for web applications. Analysis of data images requires a fast and efficient web server to respond to many users. Our Framework Server will inherit the scalability and ease of management from C#, ASP.NET, and Microsoft's Internet Information Service (IIS).

For the IDE we use the highly developed Visual Studio .NET. This environment simplifies the task of creating pages for a web application and for editing and managing the code for a large project. In addition, VisualStudio.NET simplifies the task of creating and using the web services required to extend the use of the iLab beyond MIT.

CHAPTER 2

Remote Microscope

A large part of the Polymer Crystallization iLab involves the use of a Remote Microscope. Based on Paola Nasser's design, the Remote Microscope uses a two-tier architecture, with a Microscope Client and a Microscope Server passing messages over TCP/IP sockets to remotely alter the state of the microscope settings and capture digital images. This thesis describes the enhancements made to the Microscope Server and the Microscope Client to allow users to conduct and save a complete remote polymer crystallization experiment. This chapter will give a brief overview of the Remote Microscope. An overview of how each of these two components fit into the Remote Microscope can be seen in Figure 2. For more information on the server and client developed by Nasser, the reader is directed to her Masters' of Engineering thesis [1].

2.1 Hardware Overview

The Remote Microscope uses hardware purchased from two vendors: Zeiss and Linkam. The Zeiss hardware consists of an Axioplan 2 Imaging microscope and an AxioCam MRc digital camera. The Axioplan is a motorized microscope whose component design allows the user the flexibility to add modules for several different applications. Two of these modules which are assembled with the Axioplan for our Remote Microscope are a polarized light filter and the mounted AxioCam MRc. With a resolution of up to 1300 x 1030 pixels and color binning capabilities, the camera provides the flexibility to produce both high resolution images for single image capture and compressed images for rapid

video capture. Both Zeiss components, the Axioplan and the AxioCam, can be programmatically controlled through the KS.300 software developed by Zeiss.

The Linkam hardware provides the temperature control for our system. The MDS600 Linkam heating stage, is mounted directly to the mechanical stage carrier of the Axioplan microscope. The MDS600 comes with three associated hardware controllers: the TMS94 temperature controller, the MDS600 directional controller, and the LNP94 liquid-nitrogen pump. The TMS94, which acts as a proxy for all of the Linkam hardware, is the only Linkam module directly connected to the computer through a serial port.

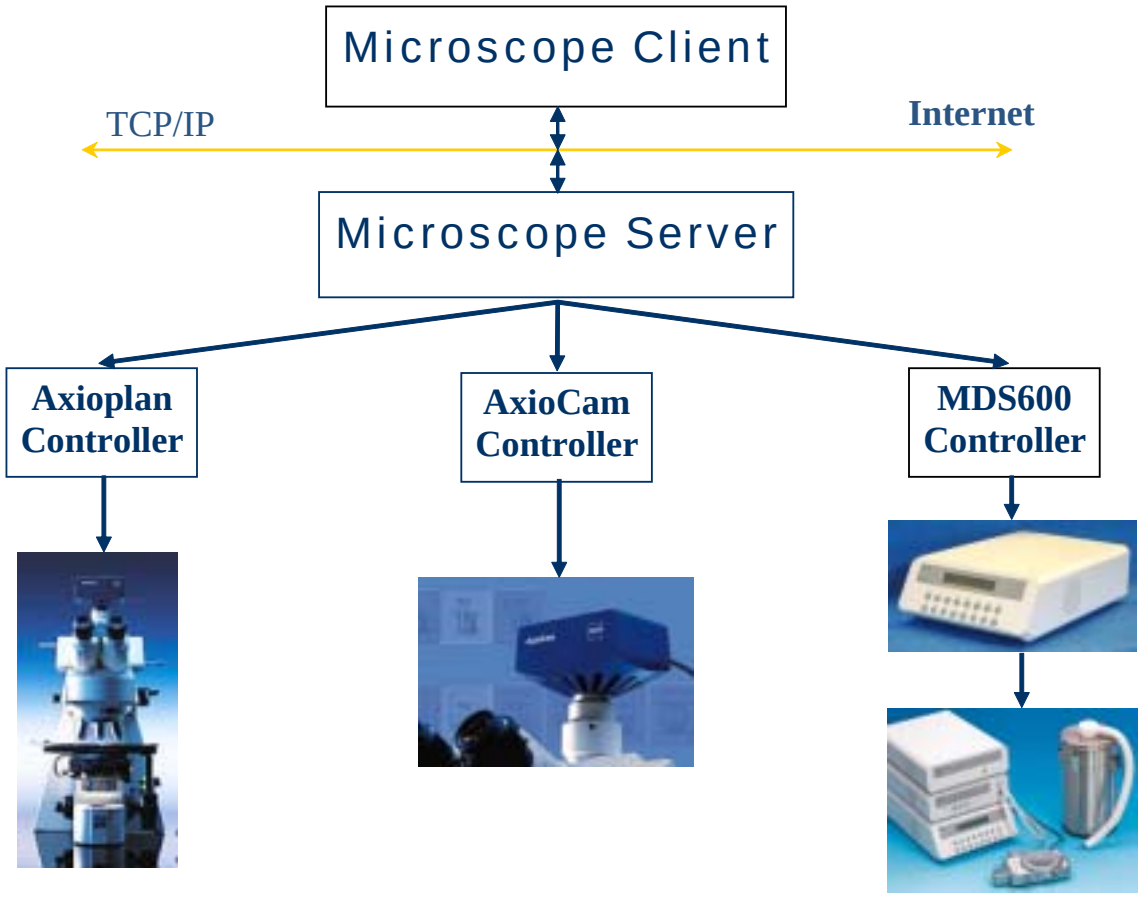


Figure 2: Remote Microscope System Architecture

In order to provide an interface between the Python Microscope Server and the hardware, the Controller class has been written in Python to wrap for the appropriate hardware functions. Since the Zeiss components can be controlled by vendor software, the Axioplan Controller and the AxioCam Controller classes both act as proxy classes to the

KS.300 software. These controller classes marshal commands using a dispatched KS.Application Component Object Model (COM) interface. The respective controller classes for the Axioplan and the AxioCam can use the KS.Application COM interface to change the objective, aperture, field stop, and reflector settings of the microscope and to capture images from the digital camera. In contrast to this high-level communication scheme, the Linkam hardware must be controlled by arcane ASCII text commands sent to the TMS94 using a serial port connection. Thus, the MDS600 Controller class provides a proxy for the Microscope Server to issue serial commands to the appropriate Linkam hardware.

2.2 Microscope Client Overview

Together, the Microscope Client and the Microscope Server comprise the Remote Microscope. The roles of the Microscope Client are to provide students with a Graphical User Interface and to parse and process messages to and from the server. The Microscope Client applet contains a number of GUI controls with which the user can manipulate the state of the microscope, camera, and heating apparatus by sending messages to the Microscope Server through TCP/IP sockets. In turn, the Microscope Client receives updates through these sockets, which it must parse and display for the user. The Microscope Client, described further in CHAPTER 6, must allow students to quickly and easily manipulate the hardware, while validating all commands, so as not to damage any of the hardware.

The Microscope Client has four primary tasks: to display graphic controls for the user to manipulate hardware settings, to message-pass with the Microscope Server, to display the state of the hardware, and to display the image of the polymer sample. These tasks are accomplished by using a Java Applet for the user's Graphical User Interface (GUI), as pictured in Figure 3. By manipulating this GUI, students create and send messages to the Microscope Server, where these messages are decoded and processed. In turn, the Microscope Client receives updates through messages sent by the server, which the client parses and displays as feedback for the user. Java Swing components are used to manipulate the state of the microscope, camera, and heating apparatus and to validate any commands the user sends.

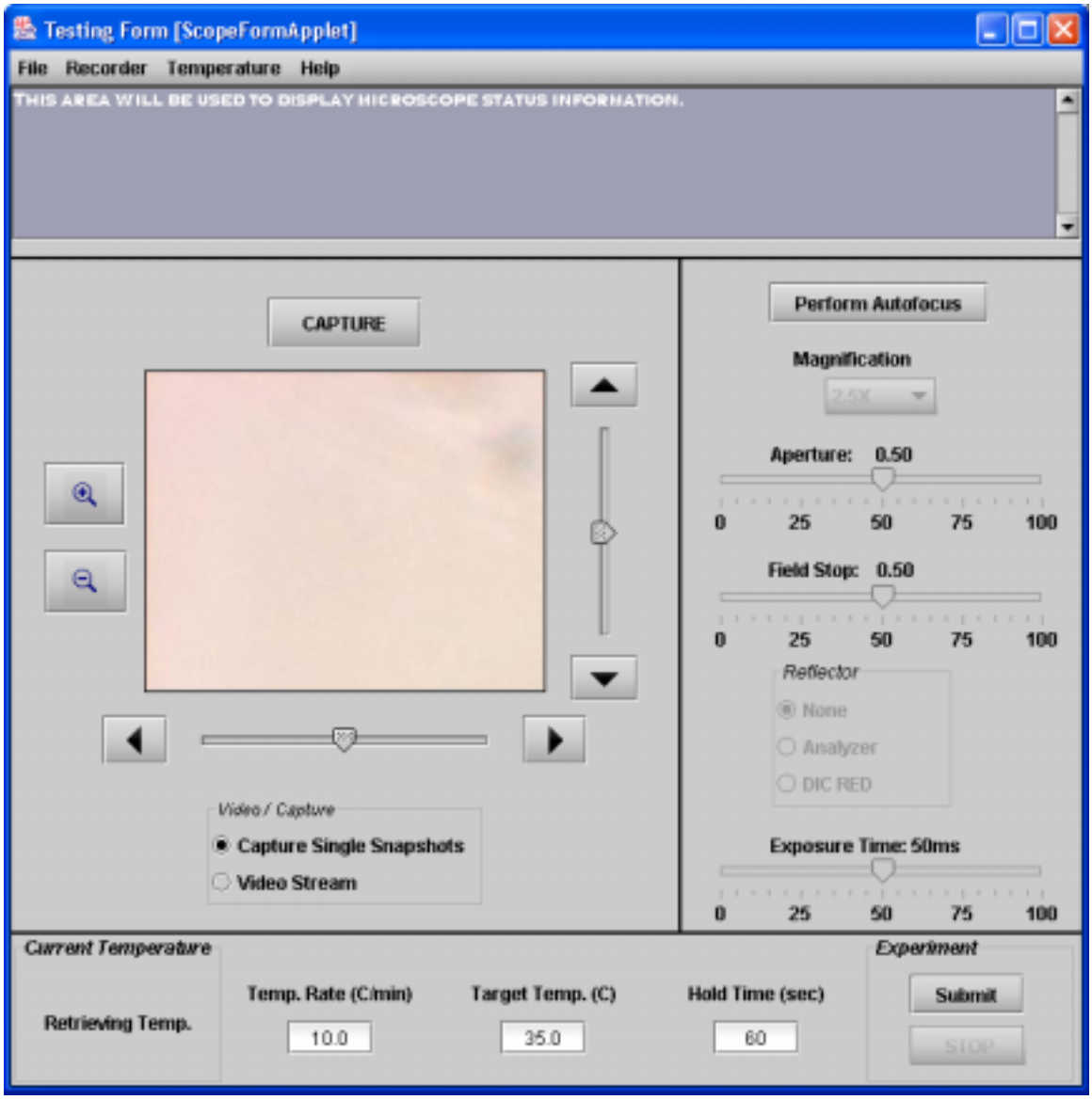


Figure 3: Remote Microscope Client GUI

The user interface is broken into four bordered sections. The northern section, called the Message Panel, is used to display the status of the client-server communication and any errors which occur during command execution. The west section, called the Image Panel, is used to display the 260 x 206 image, to subscribe to the video stream from the Microscope Server, to focus the image, and to navigate around the polymer sample. In the east section, called the Microscope Panel, the user can manipulate the GUI tools provided to control the state of the various microscope and camera settings. From this panel, the user can control the magnification, aperture, and reflector settings of the

microscope as well as the exposure time of the camera. Finally, in the southern section of the applet, called the Temperature Panel, the user can view the real-time temperature of the sample and can submit an experiment, in which the user inputs a temperature rate, a target temperature, and the hold time for the target temperature. In addition to all these panels, a menu provides the user with additional functionality such as the ability to save an experiment and all the associated images on the server. The Microscope Client is discussed in more detail in CHAPTER 6.

2.3 Microscope Server Overview

The Microscope Server is the second software component for the Remote Microscope. The Microscope Server is responsible for maintaining the overall state of the system, accepting client connections, processing messages sent from the client to the appropriate hardware controller module, and saving experiments to the database and images to the file system. The server, written in Python, is based on Kuchling's Remote Microscope implementation for the MEMS Exchange project.

The Remote Microscope's Config and Options classes are used to set the various configuration options for the Microscope Server. The most useful options for developers are the debug option and the timing option. These options cause the server to display debugging information for the developer such as the latency of a command. Another component, the Device Manager, is used as a proxy class for all the hardware controllers used by the microscope. The Device Manager is in charge of deciding which hardware controller to use for a given input message. The reader is directed to Chapter 6 of Paola Nasser's thesis for more information on these components.

2.4 Hardware Controllers

The Controller class and all of its subclasses are responsible for sending commands to specific hardware modules, either using COM interfaces to vendor software or low-level serial port commands directly to the hardware. The Controller class is described in Chapter 7 of Nasser. Each controller has a set of properties maintained in a local dictionary called `settings`. Each of these properties can be easily accessed using the appropriate `get_<control_name>` method in the specific controller and can be modified

using the appropriate `set_<control_name>`. For a complete list of the settings of the appropriate modules, please refer to Appendix B.

CHAPTER 3

Hardware and Controller Design

This chapter first describes the hardware used for the Polymer Crystallization iLab, and then describes the controllers used for each of the hardware components.

3.1 Hardware Setup

The Polymer Crystallization iLab is an extension of the Remote Microscope project begun by Paola Nasser[1]. Two hardware components of the Remote Microscope hardware purchased by Nasser are re-used in this project: the Axioplan 2 Imaging microscope and the AxioCam MRc digital camera. The modifications made to their respective hardware controllers are described in the next section.

Although Nasser's project details the use of the Linkam LTS350 heating stage and LUDL motorized XY stage, the Polymer Crystallization iLab uses neither of these components. The small size of the viewing aperture in the LTS350 and the inability to mount the heating stage onto the LUDL XY stage make both hardware components unsuitable for this project. Although the LTS350 stage could be mounted directly to the microscope, it has no motorized XY movement. This inability to maneuver the slide in the XY direction would prevent students from being able to navigate the sample and experiment on different regions of the polymer. Thus, we use a different Linkam heating stage, the MDS600, and its associated hardware, the TMS94 and the LNP, in this implementation of the iLab.

3.1.1 TMS94 Temperature Programmer

The TMS94 is the temperature programmer for the MDS600 stage. This programmer connects to the computer using an RS232 serial port. From the serial port, low-level ASCII text commands are sent to the TMS94 to control three separate modules: the MDS600 stage, the MDS600 controller, and the Liquid Nitrogen Pump (LNP94). The TMS94 thus serves as a proxy for the other associated Linkam hardware. The connection to the MDS600 stage controls the heating block and monitors the temperature; the connection to the MDS600 controller directs the XY movement of the polymer sample carrier inside the stage; and finally, the LNP94 connection allows the TMS94 to control the flow of liquid nitrogen through the LNP94, thus controlling the cooling rate of the sample. The TMS94 is shown in Figure 4.

Figure 4: TMS94 Temperature Programmer (left) with MDS600 Heating Stage (right)

3.1.2 MDS600 Heating Stage and Controller

The MDS600 heating stage is a large area heating stage with a built-in servo for movement in the XY direction. This heating stage, pictured in Figure 4, can operate in a temperature range between -196ºC and 600ºC with heating and cooling rates from 0.01ºC/min to 130ºC/min. The MDS600 connects directly to the Axioplan microscope using a small sub-stage mounting device. Thus, the heating stage itself cannot move in the XY direction; however, a slide carrier inside the stage allows the polymer sample to

travel inside the heating stage above the heating block in a circle with a radius of 15 mm.

3.1.3 Liquid Nitrogen Pump (LNP94)

In order to allow cooling at rates faster than the ambient cooling rate, the MDS600 comes with a LNP94 cooling system. The LNP94 has two pumps that are automatically controlled by the TMS94. These pumps regulate the flow of liquid nitrogen from a 2L Dewar flask to the MDS600 heating stage to cool the encased polymer sample. The liquid nitrogen in the chamber allows us to lower the temperature of the sample to -196ºC. The LNP94 is shown in Figure 5.

Figure 5: Liquid Nitrogen Pump (LNP94) and Dewar Flask

3.2 Hardware Controllers

The Polymer Crystallization iLab uses three hardware controller classes: Axiocam, Axioplan2, and MDS600. Each of these classes is written in Python and is a subclass of the Controller class. Figure 6 shows a module dependency diagram of each of the hardware controllers. Both the Controller class and the associated Device Manager are discussed in detail in Nasser (Chapter 7). Because of the inheritance of each controller, there is an inherent dependency on the Controller superclass. The following subsections discuss the lower-level dependencies and the command flow in more detail.

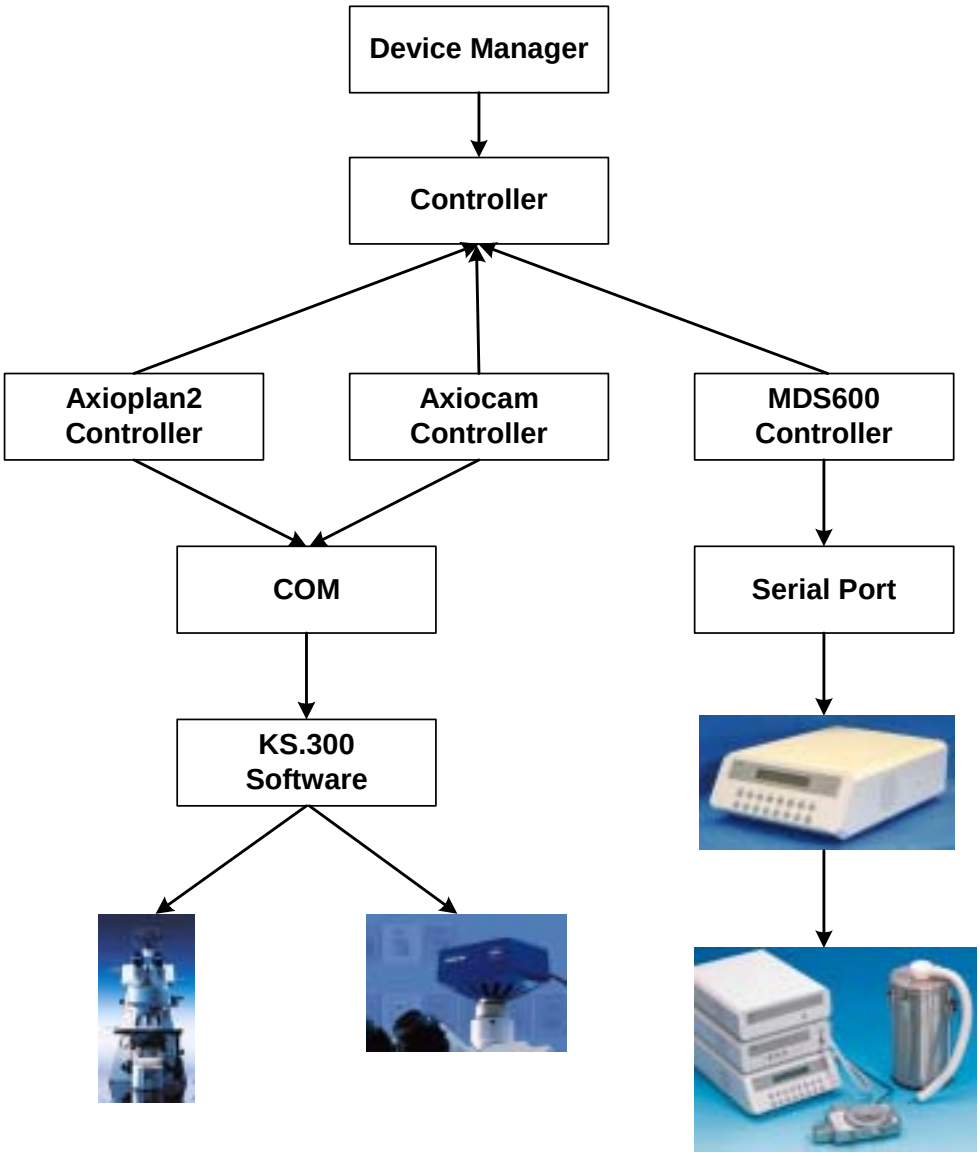


Figure 6: Module Dependency Diagram of Hardware Controllers

3.2.1 Axioplan2 Controller

As the diagram shows, the Axiocam Controller's dependency hierarchy includes COM, KS.300, and the actual microscope hardware. This controller is responsible for setting and maintaining the state of all adjustable microscope settings. To do this, the Controller dispatches commands to the KS.300 interpreter using the KS.Application COM interface. By dispatching a KS.Application object using the Python Win32 package, the functions available to the KS.300 interpreter are exposed to the Python controller modules. Thus,

the COM interface makes all functionality of the KS.300 software available to Python. More information on the KS.300 interpreter can be found in the help documentation of the KS.300 software.

3.2.2 Axiocam Controller

The dependencies for the Axiocam Controller mirror those of its related Axioplan Controller. The Axiocam Controller is responsible for the exposure time property and for capturing frames from the camera. The Axiocam Controller issues commands to the camera by dispatching them to the KS.300 interpreter using the same COM interface as the Axioplan2.

One significant difference between this controller and the previous one is that the Axiocam uses KS.300 configuration scripts. These configuration scripts included in the application, allow an administrator to locally open the KS.300 software and create a configuration for the images that the Axiocam captures. The Python controller can then load a configuration for the camera using the *tvload* command from the KS.300 interpreter. Previously, the Remote Microscope camera always captured images at the maximum resolution, and then used the Python Imaging Library (PIL) to manipulate the size and encoding of images before they were sent to the user. This level of indirection caused an awkward latency, on the order of six seconds, between the time the user requested an image and the time an image was displayed in the Microscope Client. Video streaming was also very choppy, with a frame rate of one frame every six seconds. Using configurations to specify the camera's encoding of the image instead of post-processing the image allows the system to maintain a video stream of two frames per second and provides a huge reduction in latency during single-image capture.

3.2.3 MDS600 Controller

The MDS600 Controller shows another alternation of Paola Nasser's implementation of the Remote Microscope. Whereas Nasser's implementation called for the use of the LTS350 heating stage and the LUDL XY stage, the incompatibility of the two stages required us to purchase the MDS600, the TMS94, and the LNP94. The dependency diagram shows that the MDS600 Controller is solely dependent on the TMS94 through a

serial port RS232 connection. All commands to alter the temperature of the heating stage and the position of the sample inside the stage are issued to the TMS94 using archaic single-line text commands as specified in the Linkam programmers guide, "Serial Communications Manual for the T92, T93, T94 Series Programmers."

The design decision was made to incorporate both the location and temperature capabilities of the TMS94 into the MDS600 Controller because of the primitive nature of communication between the computer and the TMS94. Since serial ports can only have a single connection open at one time, separation of the two modules would require an incessant opening and closing of serial port communications, introducing an overhead in each command sent to the TMS94. In addition, since multiple threads access the controller, a complex blocking scheme would have to be employed at high levels in the architecture to prevent two serial commands from being issued at the same time. By combining both capabilities in one module, the system avoids the overhead and complexity arising from such problems.

The TMS94 has limited temperature control functionality. The TMS94 accepts the rate of temperature change and the target temperature as inputs. When a user commands the system to take control of the heating stage, the TMS94 ramps to the target temperature and holds that temperature indefinitely. More sophisticated control of the heating stage temperature is achieved by the Microscope Server and discussed in Section 7.2.

For movement in the XY direction, the TMS94 issues differential positioning commands to the MDS600 (i.e. move +/-50μm). Therefore, the MDS600 Controller class is responsible for tracking the absolute position of the sample in the heating stage. Since the heating stage is not equipped with a reference motor, centering of the sample inside the stage must be done when the Microscope Server is started (*see Appendix A*). At this time, the location of the sample holder is initialized to (0,0). All further directional commands use this initial starting position as the reference position until the Microscope Server is restarted.

The operation of the Liquid Nitrogen Pump has not yet been tested. Although the pump does have its own set of commands in the Linkam programmer's manual, the TMS94 has the capability to monitor and automatically adjust the flow rates for liquid nitrogen. Using the current temperature of the stage, the TMS94 sets the pump flow rate to achieve the desired rate of temperature change until it reaches the target temperature.

CHAPTER 4

Database Schema

The database for the Polymer Crystallization iLab allows us to preserve data across web sessions and provides a clear interface between the various components of our system. The Microscope Server saves peripheral data about experiments to the database, for use by the Framework Server during post-experiment analysis. Likewise, the Framework Server uses the database to save information about users, roles, reservations, and experiments, which is retrieved and used by the Microscope Server. More information about these interfaces can be found in the next chapter. The data model for the database design is shown in Figure 7. The following sections describe the details for each of the tables in the database.

4.1 Users

Our data model includes a Users table which holds all user information that is maintained by the system. The table stores only a few fields of information, such as name, email, and school, to allow future versions of the Polymer Crystallization iLab to port the current architecture easily to a shared architecture, such as the one proposed by the iLab Project. Under such a shared architecture, it is important to limit the amount of user information available to the system, since the underlying authentication and authorization mechanism may not have access to extensive user information. A globally unique

identifier (GUID) is used as the primary key in the Users table. This GUID is used by other database tables as a foreign key to reference a User in the system. In addition, a Confirmation Code uniquely specifies a user and is sent to that user to confirm the registered email address. The Registration Status describes the state of registration for a particular user as described in Section 8.4.

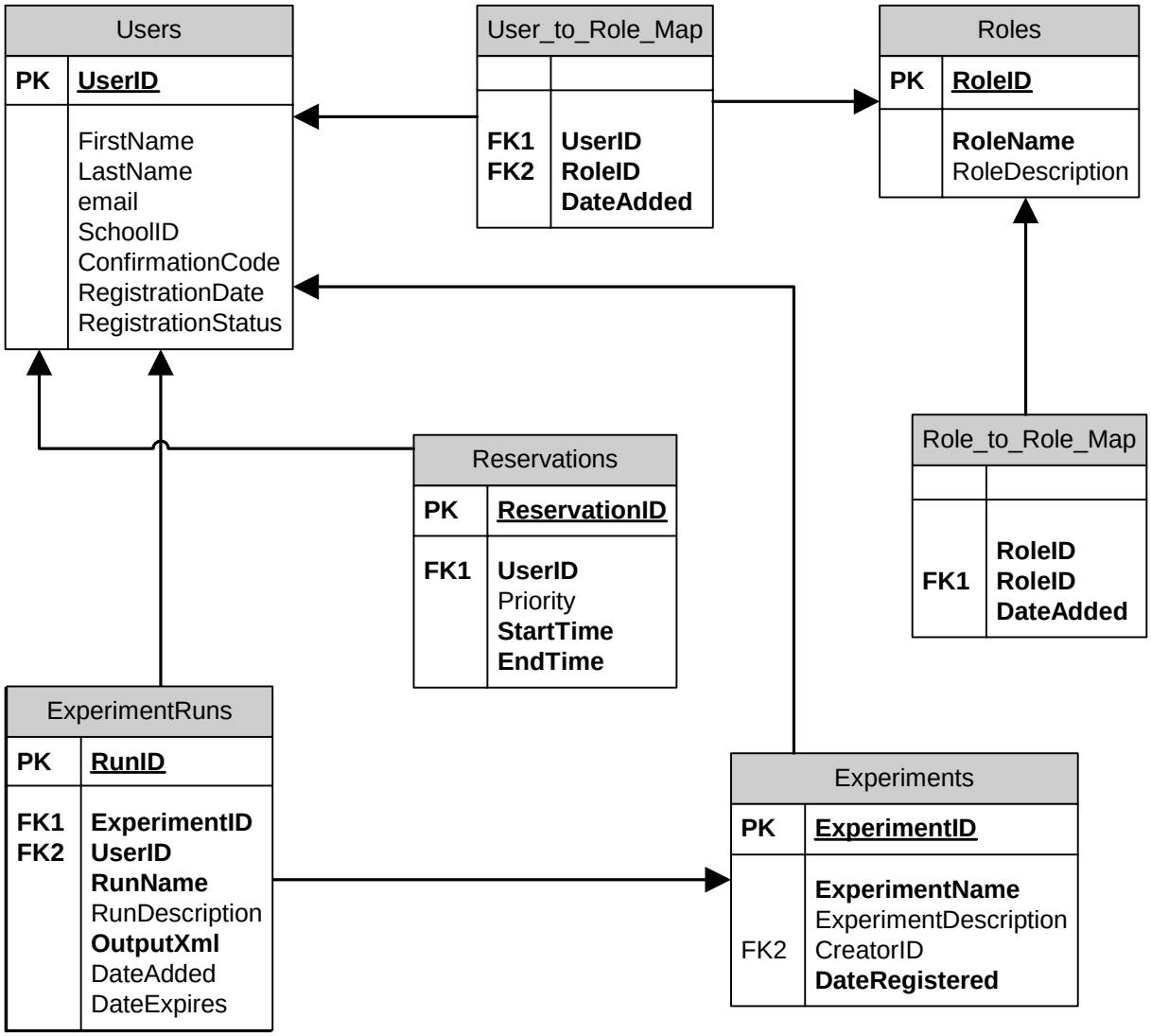


Figure 7: Data Model for the Database

4.2 Roles

In order to allow a role-based authorization mechanism to be employed, the data model includes a Roles table. Roles, indexed by a GUID, are linked to users through the User_to_Role_Map table. Based on Philip Greenspun's user management model in *The Internet Application Workbook* [8], this data model allows us to represent group membership in first-normal form: meaning data that can be derived from the joining of tables is not duplicated. The Roles table includes fields for a role name and a role description. Another table in the database, Role_to_Role_Map, allows us to map roles to other roles. This mapping table allows us to maintain data in first-normal form, while allowing different group memberships to inherit multiple permissions. This design will be useful when the system is deployed at several universities. In that case, there can be a separate user group representing undergraduate students at University A and undergraduate students at University B. Both of these groups can be linked to an Undergraduate role to simplify the administration of the system. In addition to a GUID, the Roles table contains fields for a Role Name and a Role Description. Neither of these fields is guaranteed to be unique, and both are included to simplify administration of roles by a lab administrator.

4.3 Experiments

The data model for the system includes an Experiments table used to identify the experiments that the microscope can conduct. Currently, the Polymer Crystallization iLab uses Polyethylene Oxide (PEO) as the polymer sample. In order to differentiate between experiment runs carried out using PEO and experiment runs using an alternate polymer sample, each experiment run is linked to a particular experiment type. The Experiments table also allows a lab administrator, such as a teaching assistant or a professor, to filter all experiment runs of a particular type. For example, if Class A is conducting Experiment X and Class B is conducting Experiment Y, the professor for Class A could choose to view only experiment runs of type X. Each row in the Experiments table is uniquely identified by a GUID, and contains an Experiment Name, an Experiment Description, a Creator ID that references the Users table, and a Registration Date. All of this information is provided for administrative purposes and is used in the Framework Server.

4.4 Experiment Runs

Each time a student performs an experiment, it is cataloged in a database table called Experiment Runs. A GUID again serves as the primary key for a row in this table. In addition, the Experiment ID and the User ID columns contain unique identifiers, which are used to reference the Experiments table and the Users table, respectively. The Run Name also identifies an experiment run for a particular user and is used to locate the directory where all experimental files can be found. Images for a particular run are stored in a public user directory with a naming convention of $PUBLIC_DIRECTORY/*user_email/run_name*, requiring the Run Name to be unique for a specific user.

Another column in the table, the Run Description column, is supplied by the user to identify and document an experiment run. The Date Added field is generated by the system when an experiment is created, so a user or a TA can reference experiment runs by a specific date or a range of dates. The Date Expires field is included for system management. A supervisor can review and delete experiment runs which have expired in order to free up system resources for new runs. Finally, the Output XML field, which is currently `null`, contains the XML result that is output by the system after an experiment run has been completed. This XML string contains all the information a user needs for the review and analysis of an experiment run. For more information on this XML output string, the reader is directed to Section 5.4.1.

4.5 Reservations

The Reservations table stores information about when a student is authorized to use the Remote Microscope. A Reservation consists of a GUID that to uniquely identify a Reservation and another GUID to identify the user who has reserved the microscope. In addition, a Start Time and an End Time record when the Reservation becomes active and when it expires. Finally, a priority for the reservation has been included in the database. Although this field is not currently used, a priority scheme is a useful extension for allowing a professor or a TA to reserve the microscope for a demonstration or for maintenance, even though a routine user may have already reserved the system.

4.6 Stored Procedures

To relate the programming logic to the database model, a number of stored procedures have been created in the database. Each class in the Framework Server that is represented by a database table has a number of static functions which it can use to load objects from the database. Objects are loaded based on specific properties in the database columns; for example, a user with a specific GUID or email address can be loaded from the database. Thus, stored procedures are named according to the following convention: `<StaticCaller>_Get<BusinessObject>From<PropertyName(s)>`. For example, the ExperimentRun class has a static function, which can return a set of ExperimentRuns from a UserID and ExpID. The database contains a corresponding stored procedure called ExperimentRun_GetRunFromUserIDAndExpID which selects the appropriate rows from the appropriate tables. A list of all stored procedures can be found in Appendix C.

CHAPTER 5

System Architecture

This chapter familiarizes the reader with the software system architecture for this project. The Polymer Crystallization iLab consists of three largely independent software components: the Microscope Client, the Microscope Server, and the Framework Server. Each of these modules is written in a separate language to leverage the advantages of a particular programming language for the necessary task. With this benefit in efficiency comes the necessity to design a clear interface between modules. Figure 8 shows an overview of the interface mechanisms between components. Two of these components, the Microscope Client and the Microscope Server, have been introduced in CHAPTER 2. In this chapter, the final software entity, the Framework Server, is introduced, followed by a detailed description of the communication mechanisms between modules. More information about the Microscope Client, the Microscope Server, and the Framework Server can be found in CHAPTER 6, CHAPTER 7, and CHAPTER 8, respectively.

5.1 Framework Server Overview

The Framework Server manages the software system. The Framework Server is built as a traditional three-tier system. The first tier consists of a lightweight client viewed in any traditional web browser. The middle tier consists of an ASP.NET server with C# objects such as Users, Roles, Experiments, and Experiment Runs, which are manipulated through the web interface. Finally, the database provides the backend of the three-tier

architecture, allowing the user to save data across web sessions and to access data across modules.

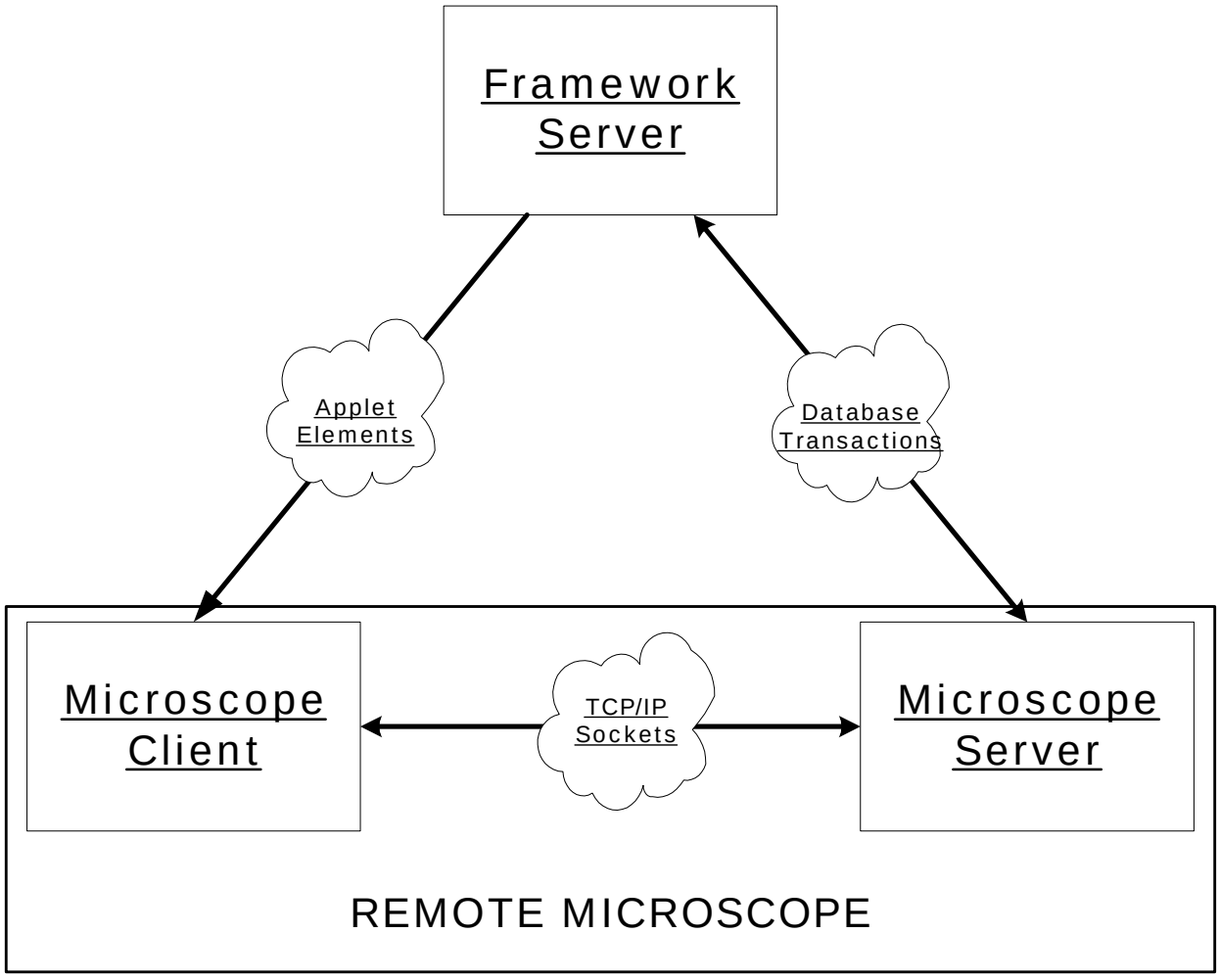


Figure 8: System Architecture Diagram

Using Microsoft's IIS to host a number of dynamically generated web pages for the user, the Framework Server provides an interface for a user to login to the system, reserve the microscope, manage user-specific information such as user profile and experiments, analyze past experiments, and use the Remote Microscope. These dynamically generated web pages provide the software logic used to manage the system. The Framework Server extracts information from the database, which it passes to the Microscope Client through applet parameters. The applet then uses these parameters to send information about the current User to the Microscope Server. After experiments are saved by the Microscope Server, they are written to the database. The Framework Server can then retrieve past

experiment runs for the user to process. For details about the Framework Server, the reader is directed to CHAPTER 8.

5.2 Framework Server => Microscope Client Communication

The unidirectional communication from the Framework Server to the Microscope Client shown in Figure 8 is achieved by passing applet parameters. As mentioned in the overview, the Framework Server dynamically generates a set of web pages for a specific user. When a user logs in to the system, a corresponding User object is saved in the Framework Server's session context. When the user requests to use the microscope, the Framework Server generates a web page that includes parameter tags for the user's GUID, a reservation ticket GUID, and an experiment GUID. This web page directs the browser to load the Microscope Client. These parameters are passed to the Microscope Server when the Microscope Client tries to authorize itself, so the Microscope Server in turn, knows who is using the system. Simple HTML tags are added to the dynamically generated web-page to redirect the browser to a logout page after the reservation has expired. These tags prevent users from logging on to the Microscope Client indefinitely. For more information on users, experiments, and reservations, the reader is directed to CHAPTER 8.

5.3 Microscope Server ⇔ Microscope Client Communication

Bidirectional communication between the Microscope Server and the Microscope Client is achieved through two low-level TCP/IP sockets using a simple but well-defined message protocol based on Kuchling's Microscope Networking Protocol Specification [7]. One socket is a half-duplex binary socket used by the server solely to transmit images to the client. The other socket is a full duplex socket through which ASCII messages are passed between the server and client. These messages consist of a single line of text terminated by a new line character, '\n'. The first word of the line is the command name, followed by parameters in the form of [parameter_name=value] pairs. Commands can have any number of parameters, some of which may be optional. Since the protocol is asynchronous, neither the client nor the server locks while waiting for a

response to any of the messages; thus any packet drops due to network errors are handled gracefully by the system.

The Microscope Client acts as a messenger between the user and the server. Users can pass messages to the server by altering the appropriate GUI controls in the Microscope Client. Commands are then generated and sent across the network to the Microscope Server, where they are parsed and marshaled through a Device Manager to the appropriate hardware Controller class. The hardware controller is then responsible for changing the state of its hardware, after which an acknowledgement message is returned to the client.

There are a number of advantages to this messaging abstraction between the client and the server. The human-readable ASCII text messages provide a clear interface between the two entities. In addition, this interface does not impose any restrictions on the implementation of the client and server, so long as both modules can send and process messages. A full description of the messaging protocol is included in the following subsections: 5.3.1 and 5.3.2.

5.3.1 Client Commands

The Microscope Client initiates communication with the Microscope Server using an AUTH message. Based on the user's GUID, the reservation GUID, and the experiment GUID, the Microscope Server determines whether or not to allow the user to access the microscope. Each of these parameters is passed to the Microscope Client by the Framework Server through simple applet tags. After authorization, the Microscope Client can send any combination of commands based on user manipulation of GUI controls. The following table shows all commands sent from the client to the server.

COMMAND	**PARAMETERS AND EXPLANATION**
AUTH	userID=*value* reservationID=*value* experimentID=*value* Request to authorize this client. The parameter values are

	GUIDs for the user, the reservation, and the experiment. They are used in database transaction by the Microscope Server in this client session.
AUTOFOCUS	Performs auto focusing by searching for the best z-focus position.
SET	x=*value* y=*value* focusPosition=*value* magnification=*value* lightMode=*value* aperture=*value* fieldStop=*value* reflector=*value* exposureTime=*value* Changes the hardware settings and status. All parameters are optional. Command can be sent with any number of parameters. A list of valid parameter values can be found in Appendix B.
CAPTURE	Config=*value* The parameter is the name of a predefined acquire configuration. For more details on configurations, see section 3.2.2. Currently, the only configuration used is the "iLab" configuration which captures color JPEG images with a resolution of 260x206 pixels.
VIDEO_START	Config=*value* Requests video streaming to start. The argument is the same as in the CAPTURE command above.
VIDEO_STOP	Requests the video streaming to stop.
SERVER	Prints out the server information such as threads, port connections, and client connections. This command is usually used only for debugging.
STATE	Print out the server state information. This command is usually used only for debugging.
RUN_EXPERIMENT	targetTemp=*value* temperatureRate=*value* holdTime=*value* Commands the server to take control of the heating stage, ramp the temperature to the targetTemp at a rate of temperatureRate, and hold for targetTemp for holdTime seconds.
STOP_EXPERIMENT	Commands the server to cede control of the heating stage
START_RECORDER	archiveName=*value* overwrite=*{yes, no}* Commands the recorder to start archiving images and saving experimental data. The overwrite flag tells the server whether or not to overwrite an existing experiment run with archiveName
STOP_RECORDER	Commands the recorder to stop and tells the server to persist the experiment in the database
QUIT	Terminates the client connection. The server removes the connection and close the TCP/IP socket.

Table 1 - Client Commands to Server

5.3.2 Server Commands

Microscope Server to Microscope Client communication exposes the consequences of our asynchronous messaging protocol. Although we are using TCP/IP sockets, which are "guaranteed" to send all network packets in order, we would like our system not to pause while awaiting network transmissions. To prevent this delay, we use asynchronous messages. However, because our commands are sent asynchronously, we must also acknowledge client commands with an update message from the server. For example, if the user submits an experiment, the RUN_EXPERIMENT message is passed to the Microscope Server with the appropriate parameters. The Microscope Client does not assume that the message was received and does not change any values on its display. The Submit button remains enabled until the Microscope Server sends an EXPERIMENT message back to the client to acknowledge the request. Similarly, the RECORDER message in the table below acknowledges a START_RECORDER message from the client; a SCOPE message acknowledges a SET message from the client; and an OCCUPIED or AVAILABLE message acknowledges an AUTH message from the client. The following table describes all possible commands that are sent from the server to the client.

Command	**Explanation**
IMAGE	*length* \n *binary data* Transmit image to client as binary data. The message includes the length of the image data, followed by a new-line, and the image data.
SCOPE	x=*value* y=*value* focusPosition= *value*magnification=*value* lightMode=*value* aperture=*value* fieldStop=*value* reflector=*value* exposureTime=*val* Inform clients of the microscope's current settings (current state). For a list of valid parameter values see the controller's details in Appendix B.
STATUS	*msg* Sends a message to client specifying the status of an issued command

ERROR	*msg* Sends an error message to client.
TEMP	*temperature* Sends a temperature update for the client to display
EXPERIMENT	*{heating, cooling, holding, cancelled}* Notifies the client of the progress of the experiment. Also acts as an acknowledgement that a command was received
RECORDER	*{overwrite?, recording, stopped}* Notifies the client of the status of recording. Also acts as an acknowledgement that a command was received
OCCUPIED	Message sent upon initialization to declare the availability of the scope.
AVAILABLE	Message sent upon initialization to declare the availability of the scope.

Table 2 - Server Commands to Client

5.4 Microscope Server ⇔ Framework Server Communication

The Microscope Server and the Framework Serve communicate via database transactions. Reservations, Users, and Experiments are all initially created using the Framework Server, as described in CHAPTER 8. When the Microscope Client starts a session, it passes user, reservation, and experiment GUIDs obtained from the Framework Server to the Microscope Server. The Microscope Server retrieves these objects from the database to check their validity. For example, the Microscope Server retrieves a Reservation from the database and checks to ensure that the appropriate person is using the microscope. Thus, the reservation, user, and experiment which were initially created in the Framework Server, are communicated to the Microscope Server using database transactions.

The Microscope Server to Framework Server communication is accomplished through a similar mechanism. When a user starts the recorder using the Microscope Client and supplies an archive name, the Microscope Server uses this archive name to create a

directory inside the users' public directory. Subsequent images are saved to server under this newly created public directory. When the user completes the experiment run and stops the recorder, the run is saved in the database and a Synchrnoized Media Integration Language (SMIL) output file is written by the server to the archive directory. After an experiment run has completed, the Framework Server allows users to navigate through a series of web pages dynamically generated with ASP.NET. The Framework Server uses the database to retrieve experiment runs for a specific user for analysis and management of past experiments. Thus, by saving the experiment to the database, the Microscope Server communicates with the Framework Server and saves experiment runs.

5.4.1 SMIL

The Synchronized Media Integration Language (SMIL) [10] is a W3C Recommendation for combining audio, video, text, and graphics into a unified presentation. The central function of the language is to time multimedia components and to schedule their display either in parallel or sequentially. The language is written as an XML application, which allows authors or programs to create multimedia presentations by simply outputting a text file. This text file contains URLs to the specific media elements which are retrieved by the SMIL interpreter. Currently, Real Player and Quick Time both implement the W3C Recommendation and are able to read SMIL files.

When an experiment run is saved to the system, a SMIL file is created allowing users to play back their experiments. These files create a slide show presentation with two regions superimposed on a Root Pane, as shown in Figure 9. The SMIL presentation contains a sequence of parallel elements. Each parallel element contains textual data describing the frame number for a particular image and a JPEG image saved during an experiment run. The Image Region displays the image file, while the Caption Region displays the frame number of each image. As the SMIL timeline progresses, these parallel elements sequentially display images with their corresponding captions, allowing the user to replay the experiment run.

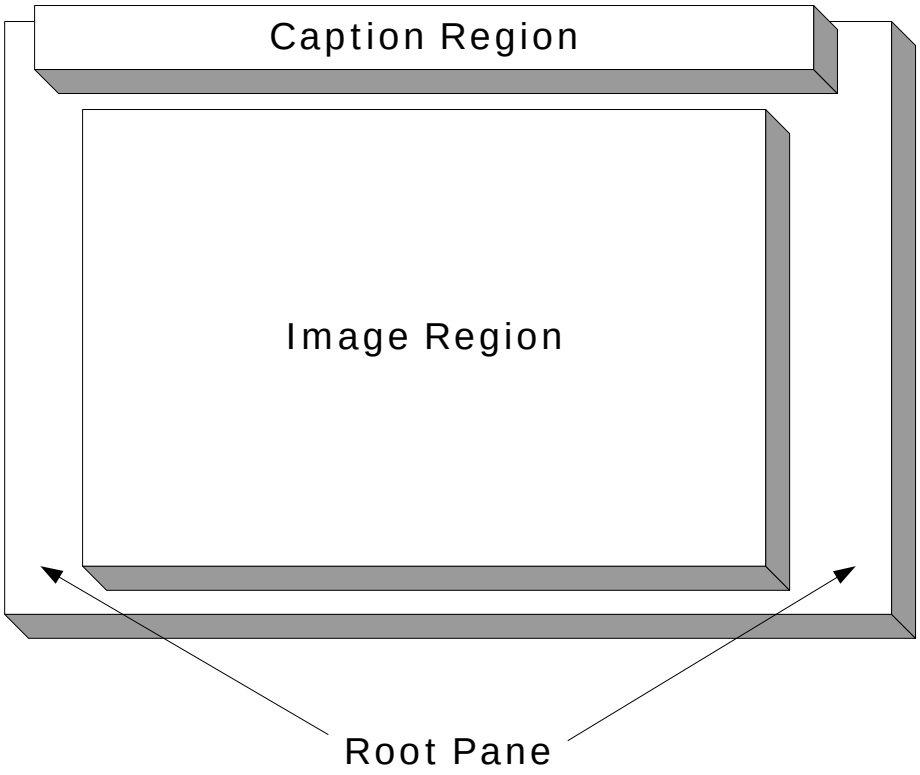


Figure 9: Layout for the SMIL presentation of an experiment run.

CHAPTER 6

Microscope Client

The Microscope Client provides the GUI for the user to control the Remote Microscope. As shown below in Figure 11, this module has been largely rewritten to reflect a number of development improvements, interface improvements, and application enhancements. This chapter details each of these changes to the Microscope Client.

6.1 Development Improvements

The Remote Microscope Client described by Nasser (Chapter 5) consisted of a single Java Panel, the ScopeGUI, which performed the majority of the functionality for the applet. This panel contained three subsections: the northern, eastern, and western panels described in Section 2.2. Since the ScopeGUI was a Graphical Form in Sun's Forte IDE, it and its constituent components could easily be manipulated using the IDE's Form Editor. This Form Editor allowed GUI components to be visually manipulated so that the developer could preview what the components of the ScopeGUI would look like to a common user.

In this new version of the Polymer Crystallization iLab, the entire applet has been converted to a Graphical Form, so that the developer can easily preview and manipulate the entire applet. The class ScopeFormApplet contains all the necessary functionality for the Microscope Client. Since the entire ScopeFormApplet class is a Graphical Form in the new Java Sun ONE IDE, it and any of its components can easily be altered by future developers using the IDE's Form Editor. The added functionality with the development

of the Polymer Crystallization iLab has required the addition of a menu to improve user interface. From this menu, users can command the Microscope Server to take control of the heating stage. Users can also issue commands to save an experiment using the new menu. For more details about GUI usage, please refer to the Student User Manual in Appendix B.

6.2 Temperature Panel

The largest modification to the interface of the Remote Microscope Client is the addition of the Temperature Panel. The Temperature Panel, found in the southern part of the Microscope Client and pictured below in Figure 10, consists of five subsections. The left-most subsection displays the current temperature of the heating stage. Real-time update messages are sent from the Microscope Server to the Microscope Client every two seconds. The messages in this real-time data stream are processed by the Microscope Client and displayed to the user in this section of the panel.

Figure 10: Temperature Panel

The middle three subsections of the Temperature Panel contain three input fields for an experiment run: Target Temperature, Temperature Ramp Rate, and Target Hold Time. Each of these fields has a validator. When the focus of the text field is lost, the validator ensures that the fields contain valid parameters for an experiment submission. The current valid values allow a temperature rate within 0.1-45ºC/min, a target temperature between 35-150ºC, and a hold time within 0-300 seconds. If the value of a text field is invalid when a *focus lost* event is raised by the applet, a dialog box notifies the user that the input is invalid and the applet returns focus to the appropriate text field.

The right-most subsection contains the buttons to submit and stop an experiment. When the submit button is depressed, the values in the three fields are again validated. If valid, the Microscope Client sends a RUN_EXPERIMENT message to the Microscope Server

with the appropriate parameters. The MDS600 Controller takes take control of the heating stage and brings it to the specified temperature before it is released to cool to the ambient temperature or until another experiment is submitted.

It is important that the Microscope Client ***not*** assume that the Microscope Server has received any messages. Therefore, the Microscope Client must wait for an EXPERIMENT message from the Microscope Server before disabling the SUBMIT button and enabling the STOP button. This creates an inherent race condition if two experiments are submitted by the client's having depressed the submit button twice before the EXPERIMENT message is received by the client, but this risk is necessary because of the asynchronous communication used in our protocol. More information can be found in Section 5.3 of this paper.

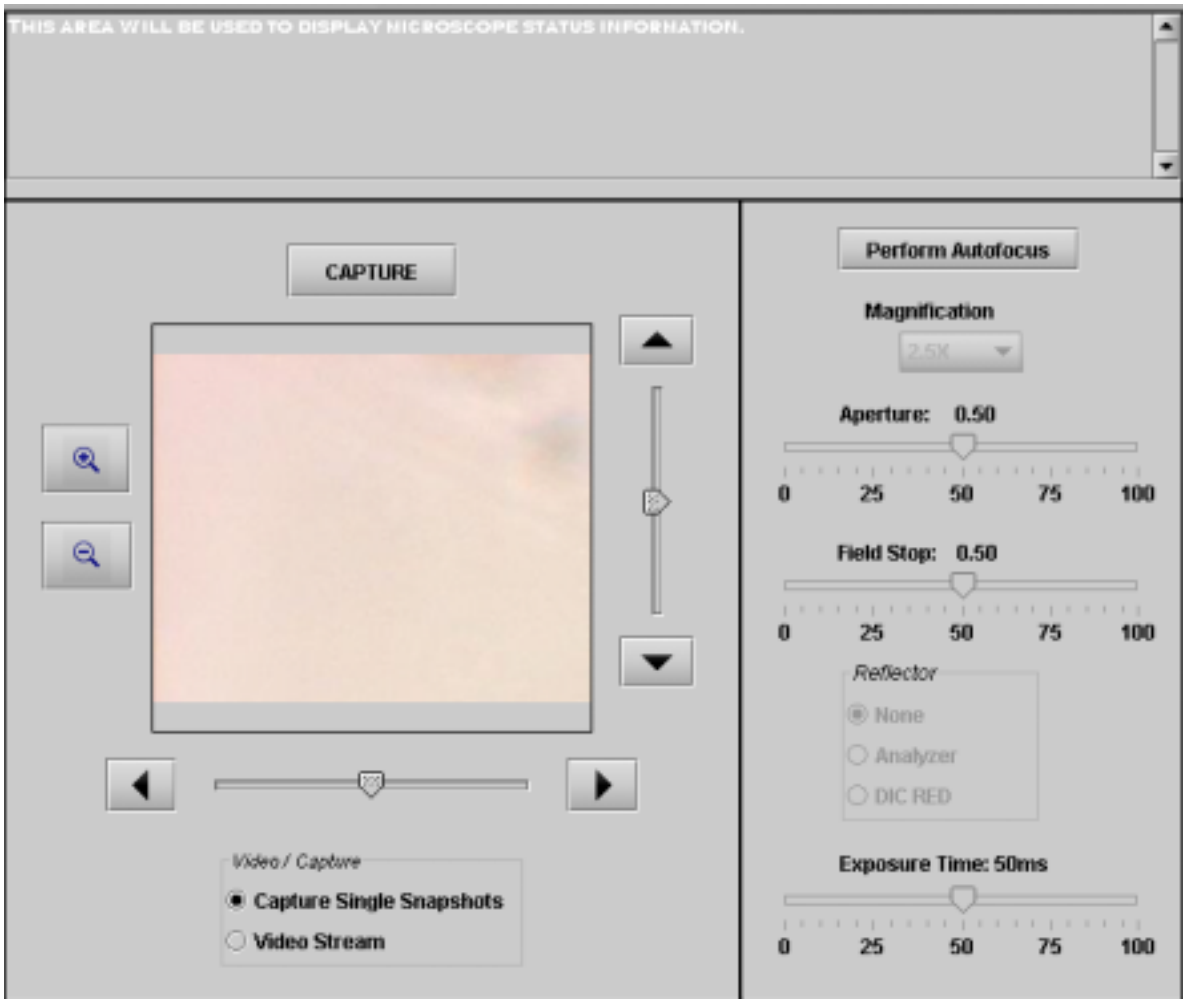


Figure 11: Message Panel (top), Microscope Panel (right), and Image Panel (left)

6.3 Other Interface Modifications

Apart from the addition of the Temperature Controller panel discussed in the previous section, a number of changes were made to the existing Remote Microscope Client in Figure 11. The Microscope Panel remained largely intact from Nasser's implementation; however, a feature was added to update the image every time a microscope setting was changed. Because of the improved speed of image updates, this change does not add excessive latency to the system. All other interface modifications are discussed in the following subsections. A complete student user guide can be found in Appendix B.

6.3.1 Message Panel

The original Remote Microscope had a Message Panel with a two lines of text: one for status updates and one for error updates. Evaluation of that design showed that some messages are sent by the server but never displayed. If two messages arrived quickly enough, the first message is impossible to read before it is overwritten by subsequent messages. To prevent important messages from being missed by the user, a string buffer was included in the Microscope Client to maintain a history of the status messages received by the client. The Message Panel was then altered to allow a scrolling text window, where users could access previous messages that were sent from the Microscope Server.

6.3.2 Image Panel

The original Image Panel had left, right, up, and down buttons which the user could use to change the XY positioning of the sample. Evaluation of this design showed that at high magnifications, a click of a directional button would produce show an entirely different region of the polymer sample. On the other hand, if pressing the button caused a smaller amount of change, at lower magnifications, this change would not be perceivable, and users would need to press the button excessively to move the sample. Therefore, since the resolution of XY movements depends on the magnification objective the users have in place, a more flexible navigation mechanism had to be implemented.

To solve this problem, a slider was added to the XY direction with which allows the users to specify the granularity of XY movement. Users can move the slider the appropriate amount to modify the location of the polymer sample and the region of the image capture.

Another modification to the Image Panel was the addition of focusing capability. To accomplish this enhancement, a Plus button and a Minus button were added to the left of the image. Since, like the XY position, the desired focus position depends on the magnification being used, a weighting factor is used to compute the actual Z direction positional change of the heating stage. When the Plus button is pressed, the Microscope Client computes the new position and validates that the new position of the stage does not exceeds its maximum height. This client-side validation ensures that the client does not issue a command which would crash the objective lens into the heating stage. If the input is valid, the Microscope Client sends a message to the Microscope Server to change the focus position of the Microscope.

6.3.3 Applet Menu

With the added functionality of the fully operable Polymer Crystallization iLab, a menu interface was added to reduce the cluttering of the Microscope Client. Instead of future developers adding more buttons to the GUI, they can add menus for the appropriate controls. Users can use the appropriate menus tell the Microscope Server to start and stop the recorder and to take or cede control of the heating stage, although this notion of control is not currently implemented. When a user starts the recorder using the Recorder Menu, the user is prompted to enter an experiment title. After entering a title, a "START_RECORDER <*title*> <*overwrite*>" message is sent to the Microscope Server. The server then parses and processes the message as described in the next chapter. A complete list of these messages can be found in Section 5.3.

6.4 Microscope Authorization

When a web page to load the Microscope Client is dynamically generated by the Framework Server, applet parameters are included to inform the applet of the user ID, reservation ID, and experiment ID used in the current session. The Microscope Client then sends these parameters to the Microscope Server to ask for authorization to use the

microscope. The server then attempts to verify the reservation ticket for this user against the information in the database and returns an authorization notification, an occupied notification, or an unauthorized notification. Since users are required to reserve the microscope before use, this requirement ensures that only the appropriate user can access the server. The GUID representing the reservation ID is a 128-bit key which gives us sufficient probability that malicious attackers cannot penetrate the system. In addition, since the authorization message is sent to the document base of the applet, malicious users cannot save and run the JAR file from their local machine to try to access the Microscope Server.

6.5 Look and Feel

The Look and Feel (L&F) of the ScopeFormApplet has been modified to create a more sophisticated look for the Microscope Client. Using the pluggable look and feel package (PLAF) from Sun Microsystems, the applet employs the Kunststoff L&F implementation. This modification has been made for purely aesthetic purposes. The Kunststoff L&F includes a scheme to create buttons and menus with color gradients as opposed to the bare and unadorned appearance of the traditional Metal L&F of most java applets. Similarly, additional look and feel packages can be used to create an even more elaborate GUI for the Microscope Client.

CHAPTER 7

Microscope Server

Under Paola Nasser's implementation, the Remote Microscope allowed users to control all the necessary hardware for microscope control and image capture. With the addition of the Linkam hardware and the associated temperature controllers, the Microscope Server had to be enhanced to allow the real-time streaming of temperature updates. In addition, with all the necessary controllers in place to remotely conduct the polymer crystallization experiment, software logic had to be added to allow the Microscope Server to accept experiment submissions, monitor the progress of experiments, and allow users to save experimental data to the server. This chapter describes the enhancements made to the Microscope Server.

7.1 Temperature Updates

The TMS94 does not come with any built-in software or firmware to notify the serial port when the temperature of the stage has changed. In the absence of such an event-driven system, a polling mechanism had to be implemented to pull information about the current temperature from the hardware. On startup, the Microscope Server starts a temperature thread, along with the reading, writing, and listening threads of the original Remote Microscope to allow real-time temperature updates. This temperature thread sends status queries every two seconds to the TMS94 using the MDS600 Controller. After the

Microscope Server queries the hardware for the current temperature, if a client is connected to the server, the server sends a temperature update message to the client. The two second period between temperature updates is parameterized in the Microscope Server. The two second delay between updates allows for a reasonable latency while not burdening the system with excessive overhead.

Because of the consistency of the temperature updates, the temperature thread is also used to check the connections to all the Microscope Clients to ensure that none of the connections are broken. Every three hundred temperature updates, all of the connections to all Microscope Clients are checked to ensure that none of the links between the server and the clients have failed. This prevents the server from locking up its resources in case of a network failure. Although the system only currently allows one client, the addition of future clients for collaboration purposes is both a plausible and useful extension to this project.

7.2 Running Experiments

The Microscope Server has been expanded to allow students to conduct the polymer crystallization experiment remotely. This experiment requires students to observe the isothermal crystallization of a polymer at a number of different temperatures. As mentioned in the chapter on hardware controllers, the TMS94 takes a target temperature and target rate as inputs. Once the target temperature is reached, the Linkam hardware holds that temperature indefinitely. To enable the user to hold a certain temperature for a desired amount of time, the Microscope Server must start a Hold Timer thread to monitor the amount of time that a temperature has been held. When the specified time has passed, this Hold Timer thread releases control of the heating stage.

To start an experiment, the Microscope Client submits a RUN_EXPERIMENT message to the Microscope Server with the appropriate parameters. Before creating and sending the message, the Microscope Client ensures that the parameters fall within a specified value range. In addition to this client-side validation, the parameters are also checked server-side. This server-side validation protects against malicious attacks on the server. Because of the simple protocol used to control the microscope, the Microscope server is vulnerable to messages sent to the appropriate socket port. If a malicious attacker manages to validate himself to the system, the server-side validation ensures that all

incoming experiment parameters will not damage the hardware.

7.3 Recording Experiments

When the appropriate menu item is selected, the Microscope Client creates and sends a START_RECORDER message to the Microscope Server. This message is sent with an `aquireName` parameter which is used to create a new directory under the users directory. At that point the Microscope Server resets the frame counter and begins to save images to the server under the newly created directory. Each image is stored with a strict naming convention consisting of four alphanumeric character stings separated by an underscore character, '_', and ending in the appropriate file extension. The character strings represent the following: frame number, time elapsed * 10, temperature * 10, and magnification. Thus, the following file name, `0035_0154_0825_10X.JPG`, identifies a JPEG image in this experiment run. Parsing the file name identifies this image as the 35th image in the experiment run taken 15.4 seconds after the recorder started when the polymer sample was 82.5ºC and the microscope objective was set to 10X. This convenient naming convention allows us to extract all of the information about an image by simply parsing the file name.

When a user completes an experiment run and stops the recorder, the Microscope Client sends a STOP_RECORDER message to the Microscope Server. At that point, the Microscope Server stops saving images and writes the SMIL file used to replay the experiment run. To create the file, the Microscope Server iterates through the archived directory and extracts the necessary information for the SMIL caption from the file name. Two versions of this file are written to the archive directory, output.smil contains the well-formed SMIL XML code and output.mov contains a Quick Time modification to the SMIL file. The files can then be accessed by the user through the Framework Server as described in the following chapter.

A key consideration for the future development of the Polymer Crystallization iLab is the design of an XML schema which can define a complete experiment run. Currently, our system embeds meta-data about individual images in the file name. In this case, we are limited by the length of the name allowed by the file system. A separation of the presentation information in the SMIL file and a generalized XML schema to describe experiments would increase the flexibility of experiment analysis.

CHAPTER 8

Framework Server

The Framework Server serves as a manger of the overall software system. The three-tier architecture of the Framework Server gives our system performance gains with respect to scalability, flexibility, and speed. The lightweight client for the Framework Server, the first tier, is simply a set of HTML pages viewed in a typical browser. The multi-layer middle tier consists of a web server written in ASP.NET and an application server written in C#. Finally, the third tier, a SQL database, provides the backend data management to ensure that information is reliable and consistent throughout our distributed environment.

The Framework Server provides process management for all system users. Since only one user can logically control the microscope at a given time, the Remote Microscope necessarily has a two tier design, where a single client is in control of the server. However the Framework Server, with which users analyze their data and administrators manage the system, can benefit from the performance enhancements of a distributed, three-tiered architecture allowing multiple users to interact with the system simultaneously. In addition, the potential for using web services to distribute user management of the system across many universities lends itself to the separation of the application logic in the top two tiers and data logic in the bottom tier. This chapter describes the implementation of the Framework Server and how this module is able to control the process flow for the Polymer Crystallization iLab.

8.1 Class Summary

The classes used by the Framework Server act as wrappers for the database tables. Each column in the Users and Roles tables corresponds to a property of the User and Roles Classes, respectively. Both classes also have an array of Roles, which is obtained by compiling an exhaustive mapping from the User_to_Role_Map and Roles_to_Role_Map tables. User and Role objects are saved to the database using the `Persist()` method of their respective classes and can be loaded from the database using one of the static functions available in each class. For example, users can be loaded from the database using any of the following static methods, which call the appropriate stored procedure in the database: `getUserFromGUID()`, `getUserFromEmail()`, `getAllUsers()`, and `getUsersInRole()`.

The Experiment and Experiment Runs classes also contain properties mirroring the columns in their respective tables. The properties are loaded from and stored to the database using the `Persist()` method and other static methods included in the two classes. These static methods include the functionality to load a subset of experiments such as all the experiments runs for a specific user or all the experiment runs of a specific experiment type.

The Reservation Class follows the programming logic of the other classes, again acting as a database wrapper class. Currently, the priority field is not used; however, it will be useful for enabling administrative overrides of lab user reservations.

8.2 Identity Service and Lab Service

Currently, the system uses one level of indirection before accessing the C# business objects. Instead of explicitly calling the static methods of a given class, two library service classes, the Identity Service and the Lab Service, are used to manipulate objects. These classes are a legacy from the Framework Project, and were originally intended to be fine-grained web wervices that would allow multiple universities to easily collaborate with each other, while simplifying the development cycle for iLab implementation. They have been preserved for easy integration with the iLab Project's future design cycles. At the time this thesis was put together, the iLab Project focused its efforts on designing a shared architecture specifically for batched experiments. Future editions of this

architecture should be amenable to the implementation requirements of interactive experiments allowing for the integration of the Polymer Crystallization iLab into this infrastructure.

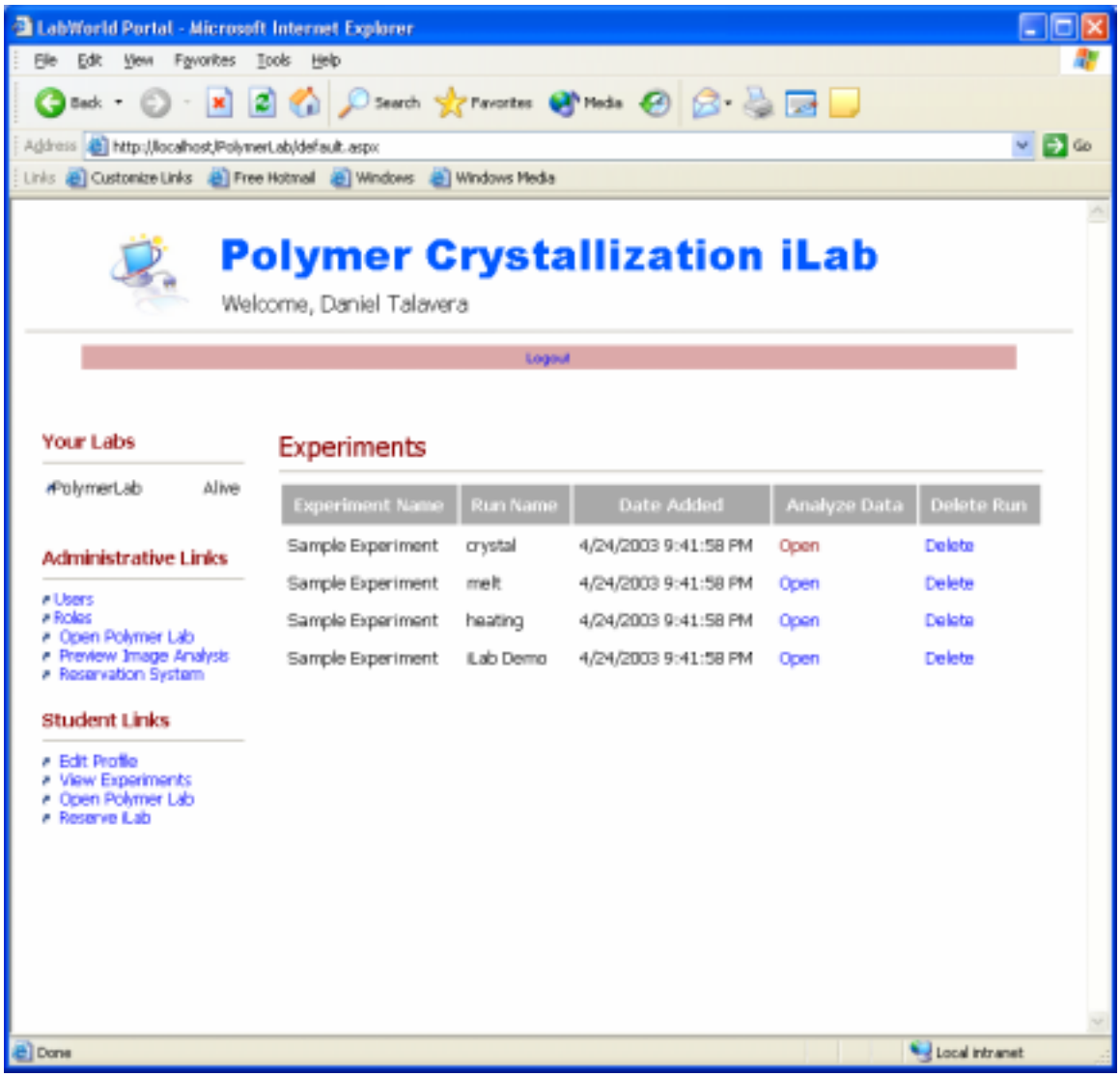


Figure 12: Framework Server User Interface

The Identity Service contains all functionality for user management. Though a large API of available functions are exposed through the Identity Service, a usable, distributed architecture will likely only contain a subset of these methods because of the performance constraints of web services. The current implementation of the Identity Service contains low-level administrative functions which work well for local calls; however, in a distributed environment, web service calls are more costly because of the

latency introduced by the network and by the conversion to and from XML. Therefore, a lower granularity of functions must be exposed in a web service implementation to decrease the number of required function calls.

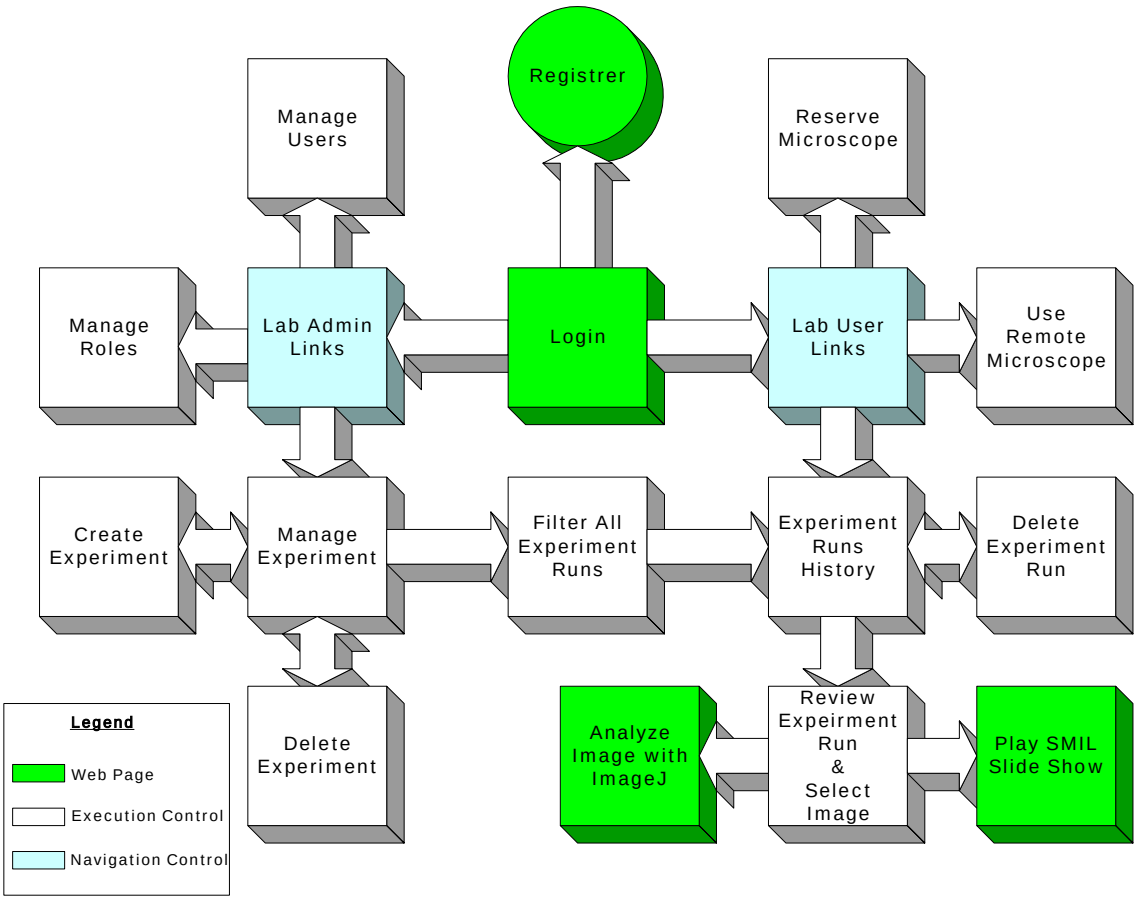


Figure 13: Page Flow Diagram for Framework Server Interface

8.3 User Interface

The user interface of the Framework Server consists of a number of dynamically generated web pages viewed through any web browser. After a user logs into the system, he is taken to the appropriate web application. The layout of the web pages in the web application consists of a Header Pane, a Left Pane, and a Content Pane, as shown in Figure 12. The Header Pane contains all the header information for the website such as the title, the user's name, and a link to the logout page. The Left Pane and the Content Pane are determined by the Roles of the current user. When a user logs into the system,

their user information is stored in the session context. This session context is used to determine the links provided in the Left Pane and the subsequent web controls accessible in the Content Pane. The system currently recognizes two types of Roles with their corresponding interfaces: Administrators and Default Users. The links provided are shown in Figure 13. The contents of the interfaces for the two roles are described in subsections 8.3.1 and 8.3.2.

8.3.1 Administrative Interface

The administrative interface was created to facilitate the management of users, roles, and experiments. When an administrator logs into the system, the Admin Links web control is loaded into the Left Pane. Using these links, the user can navigate the provided management pages to add, delete, and edit users, roles, and experiments in the system. Clicking on any of these links loads and renders the appropriate web control in the Content Pane. Figure 13 shows the links administrators can click to manage users, roles, and experiments. To preserve space, the management web controls do not show the full web control flow accessible through the management links. A typical scenario is provided below.

If an administrator wishes to manage the roles in the system, he clicks the appropriate administrative link for roles. The Mange Roles web control, shown in Figure 14, appears in the content pane showing a list of all roles in the system. From this web control, Administrators can follow the appropriate links to add a new role or edit an existing role. Following the link to edit a role causes the web control found at the bottom of Figure 14 to appear in the Content Pane. From here, a lab administrator can change a role name and description. In addition, the administrator can associate users with a specific role to give them access to different parts of the system. Management for users and experiments closely follows the above scenario, giving our administrative interface an intuitive feel.

Lab administrators are also allowed to view the experiment runs for all of the users in the system. To do this, they can navigate to the Filter Experiment Runs page, where they can select which experiment runs to display based on a combination of the experiment type or the users who created an experiment run. This allows lab administrators such as a TA to view an experiment run performed by a user for grading or to help the student analyze the results.

Figure 14: Page Flow for Role management

Currently, the Polymer Crystallization iLab can only logically support one experiment at a given time Since a lab administrator must change the polymer sample to change the type of experiment currently active, the Framework System must support the notion of the current experiment. To notify the system of the active experiment, administrators can specify the active experiment in the Mange Experiments page. When a student uses the Remote Microscope, the system embeds the active experiment GUID in the web page, notifying the Remote Microscope of the type of experiment being run.

8.3.2 Student Interface

Students will typically use the Framework Server to conduct experiment runs with the

Polymer Crystallization iLab, to reserve the system for future use, and to manage and analyze their past experiment runs. The general interface for students follows directly from the administrative interface. Students are provided a list of student links which are loaded into the Left Pane of the web page. Students can use these links to navigate to the desired pages. The page flow for student links is more complex than that for administrative links. The control flows for the links to analyze past experiments, reserve the system, and use the Remote Microscope are explained in detail in the following sections.

8.4 Registration

There are two separate ways that new users can be added to the system. The first way involves a lab administrator adding the user through the appropriate administrative web pages. An alternate and preferred means of adding a user is by having a user register with the system through the registration page. User registration follows a finite state machine modeled closely after the one described by Philip Greenspun in *The Internet Application Workbook* [8]. To become an active user in the system, a user must first either register from the registration page or be added by a lab administrator. Then, the user must confirm his email address in order to be considered an active user. From that point, a lab administrator can add the confirmed user to any of the active roles. Note that until the user has confirmed his email address, he is **not** visible in the system.

8.5 Reservation System

All users wishing to conduct experiments with the Remote Microscope must first make a reservation. Since the Polymer Crystallization iLab requires students to interact with the experimental apparatus in real-time, only one user can be in control of the system at one time. Therefore, a reservation system is required to mediate times that users can access the Remote Microscope. In addition, the reservation requirement provides a security feature for our system. Since the system only allows users with a 128-bit reservation ticket previously saved in the database, malicious users are prevented from successfully authorizing themselves.

To reserve the system, lab users load the appropriate reservation web control consisting

of a calendar and a list of reservations. From this page, users can select a date on which to reserve the system. After the user selects a date, the Framework Server queries the database to retrieve all reservations for the selected date and displays these to the user. The user then inputs a desired start time and end time during which they would like to use the Remote Microscope. The Framework Server checks to ensure that their desired time period does not exceed the maximum period allowed by the system and ensures that the reservation does not conflict with another user's reservation. If the request is valid, a new reservation is created and written to the database. The following section describes how a reservation is subsequently redeemed when a user attempts to use the Remote Microscope.

8.6 Using Remote Microscope

When a user wishes to redeem a reservation and use the Remote Microscope to conduct a polymer crystallization experiment, he must follow the appropriate student links to open the polymer lab. When the user clicks this link, the Framework Server checks the database to see if this user has a valid reservation in the database. If the user does have a valid reservation, the system creates a web page with applet parameters denoting the user GUID, reservation GUID, and active experiment GUID. The Microscope Client is then loaded into the user's browser, reads the applet parameters, and attempts to authorize itself with the Microscope Server. If the Microscope Server is in use, it sends an OCCUPIED message to the client; otherwise, it checks the appropriate parameters, reads the reservation from the database, checks the user GUID against the reservation's creator, and sends an available message to the Microscope Client. An AVAILABLE message from the server to the client begins a session with the Remote Microscope.

To enforce the reservation system, the Framework Server checks the reservation table and sets a timeout for the web page by computing the time between when the page is loaded and when the user's reservation expires. While generating the web page to use the Remote Microscope, the Framework Server embeds a META tag that directs the browser to "refresh" the page after the specified timeout. An example META tag is shown below:

```
<META HTTP-EQUIV="Refresh"
CONTENT="1500;URL=http://polymerlab.mit.edu/PolymerLab/logout.aspx">
```

The refresh directive loads a logout page after the specified time to force the user to cede control of the Remote Microscope when his reservation expires.

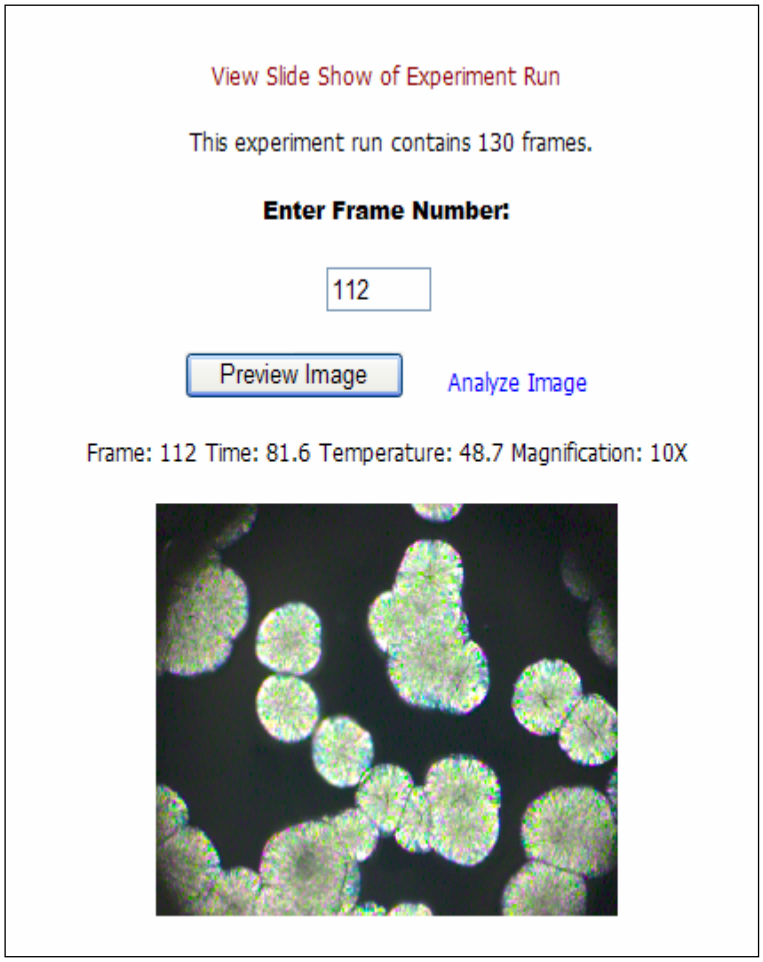


Figure 15: Analyze Experiment Web Control

8.7 Analyzing Experiment Runs

After an experiment has been run and the Microscope Server saves an experiment run to the database, the Framework Server provides users with the functionality to delete and analyze past experiment runs. Figure 12 shows the experiment history web control loaded in the Content Pane. When a user clicks on the link to open an experiment for analysis, a web control like the one shown in Figure 15 is loaded into the Content Pane.

From this control, the view slide show link opens a new browser window with a SMIL slideshow presentation to replay the experiment run. The SMIL presentation can currently be run using either Quick Time or Real Player. A user can alternatively select the desired frame number of the image they would like to preview. This causes the appropriate image to appear in the Preview Pane of the web control. The meta-data associated with a specific image, such as the frame number, time, temperature, and magnification, is parsed from the file name and displayed for the user. When the page is created, the Framework Server encodes the identity of the image in the Analyze Image link. By pressing this link, the user can navigate to a page where an image analysis applet, ImageJ, is loaded into the browser with the specified frame preloaded for analysis. The following subsection describes ImageJ in greater detail.

8.7.1 ImageJ

ImageJ is a public domain image analysis tool written in Java and inspired by the National Institute of Health's Image program. ImageJ runs as an applet that is preloaded with a specific image for analysis. ImageJ can edit, display, process, and analyze images in a number of formats. It can calculate area and pixel value statistics for a user defined area, create density histograms, and perform all standard image processing functions such as contrast manipulation, sharpening, smoothing, edge detection, and binary threshold.

ImageJ was chosen as the analysis tool to allow the greatest flexibility for user analysis. The wide variety of functions available in ImageJ allows users to explore a variety of ways to analyze their experimental data. ImageJ gives users maximal functionality while requiring users to evaluate the best possible means of analysis. This flexibility in using the image analysis software is in keeping with a primary goal of the Polymer Crystallization iLab, to create a laboratory experience with the greatest possible educational value.

CHAPTER 9

Evaluation of the Polymer Crystallization iLab

The Polymer Crystallization iLab is ready for full deployment in the Undergraduate Polymer Science Laboratory class, 10.467. The iLab has been tested by the author and a few selected individuals; however, the full deployment and evaluation will occur after the completion of this thesis. This chapter gives the reader a preliminary evaluation of the educational value of the iLab as well as a scenario describing a typical use.

9.1 Educational Value

The primary goal of the Polymer Crystallization iLab is to deliver a rich and interactive educational experience in polymer crystallization. To accomplish this goal, we have created an interactive Remote Microscope with real time streaming of images and data. This Remote Microscope closely mimics a real laboratory experience, where the user has full control over the experimental apparatus. The architecture is designed to limit the amount of automation, while maximizing the amount of user interaction and control during an experiment.

The educational principle at the heart of the Polymer Crystallization iLab is that students are more motivated and can learn better when they can conduct experiments to compare real world data to simulations, can collaborate with each other, and can explore following

their curiosity. This iLab solves the problems of high cost of multiple setups, insufficient laboratory space, and continuous staffing seen in traditional laboratories. We achieve a huge economy of scale by multiplexing a single setup among multiple users, by allowing the lab to be accessed from anywhere at any time, and by incorporating the flow of the experiment into the software interface and reducing the dependence on a TA.

The full educational value of the Polymer Crystallization iLab will be realized after the iLab is deployed and user feedback is received from actual students. Great care has been taken to make the necessary components scalable and to make the system secure. Evaluation will have to be made as to the continuing integrity of the polymer sample, to ensure that excessive use of a single sample does not significantly alter experimental results.

9.2 Usage Scenario

In a typical scenario, a student begins a session by logging on to the Framework Server. If the student has previously set up a reservation, he can directly click the link to open the Remote Microscope; if no such arrangements have been made, the student must first reserve the microscope for the desired amount of time. After a reservation has been successfully made, the student can follow the links, redeem the reservation, and begin to use the Remote Microscope.

Once a Remote Microscope session has been authorized and initialized, the student is free to use the Microscope Client within the boundaries imposed by the validated fields. At that time, the user can adjust the microscope settings, scan the polymer sample for a suitable region for the experiment, enter the parameters for an experiment run, and command the temperature controller to melt the polymer. The student then sets the analyzer in place and subscribes to the video thread so he can see when the polymer sample has melted. In addition, the student can optionally start the recorder if he wants to record the melting. If the analyzer is in place when the polymer sample melts, the user sees the polymer start to disappear as it forms a Maltese cross and the field of vision becomes blacked out. This pattern is seen in the progression of images in Figure 16.

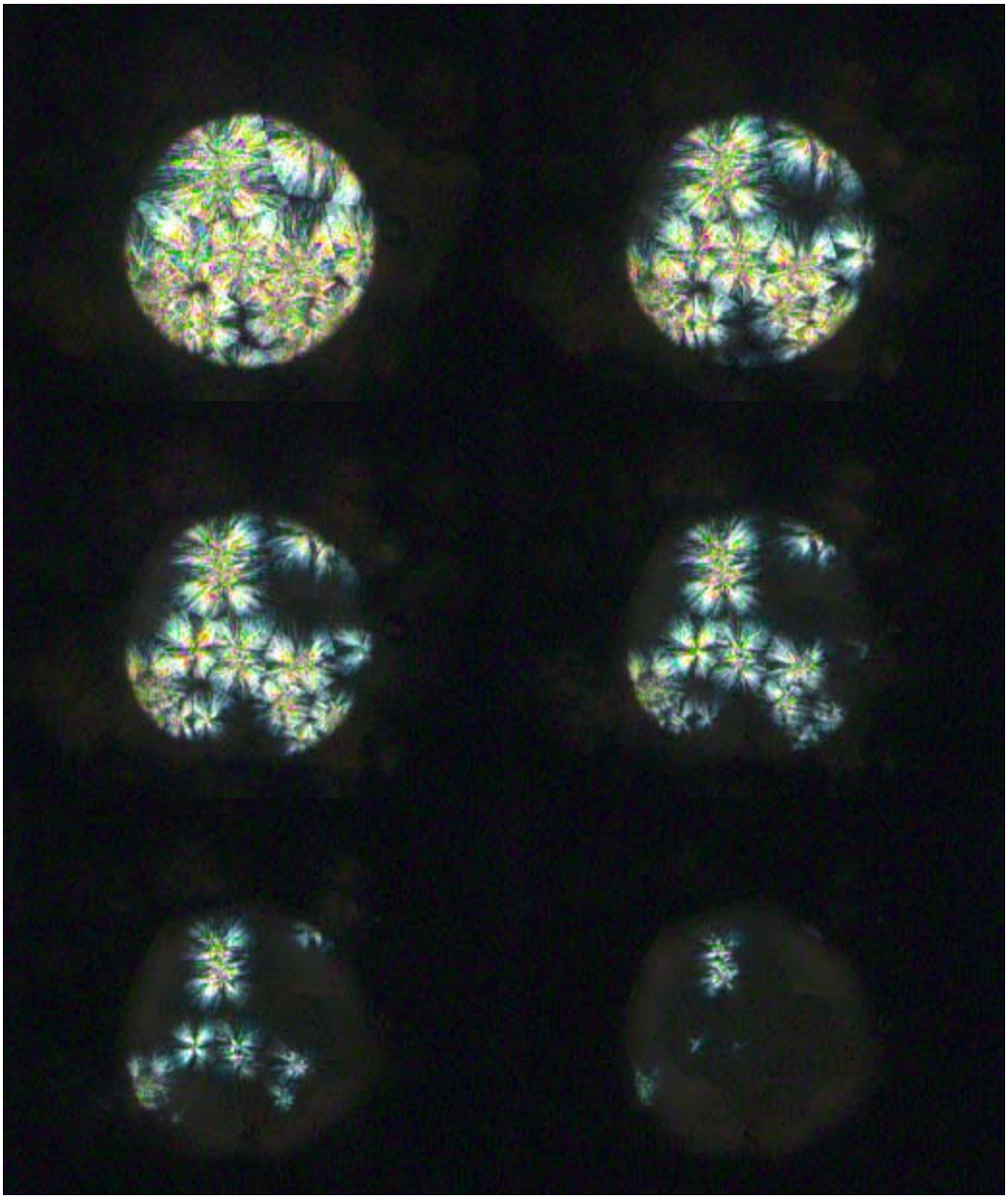


Figure 16: Maltese Cross Pattern in Melting PEO

Once the sample has melted, the student can either leave the temperature controller in control, or can stop the current experiment and input the temperature fields for the desired cooling and subsequent crystallization event. The student again commands the temperature controller to take control of the heating stage and watches as the temperature cools to the desired target temperature. The student then starts the recorder so the subsequent images are saved to the server. If the analyzer is still in place, the user sees a

number of nucleation events and the resulting crystals as each begins to grow. A sample crystallization event recorded by the system can be seen below in Figure 17.

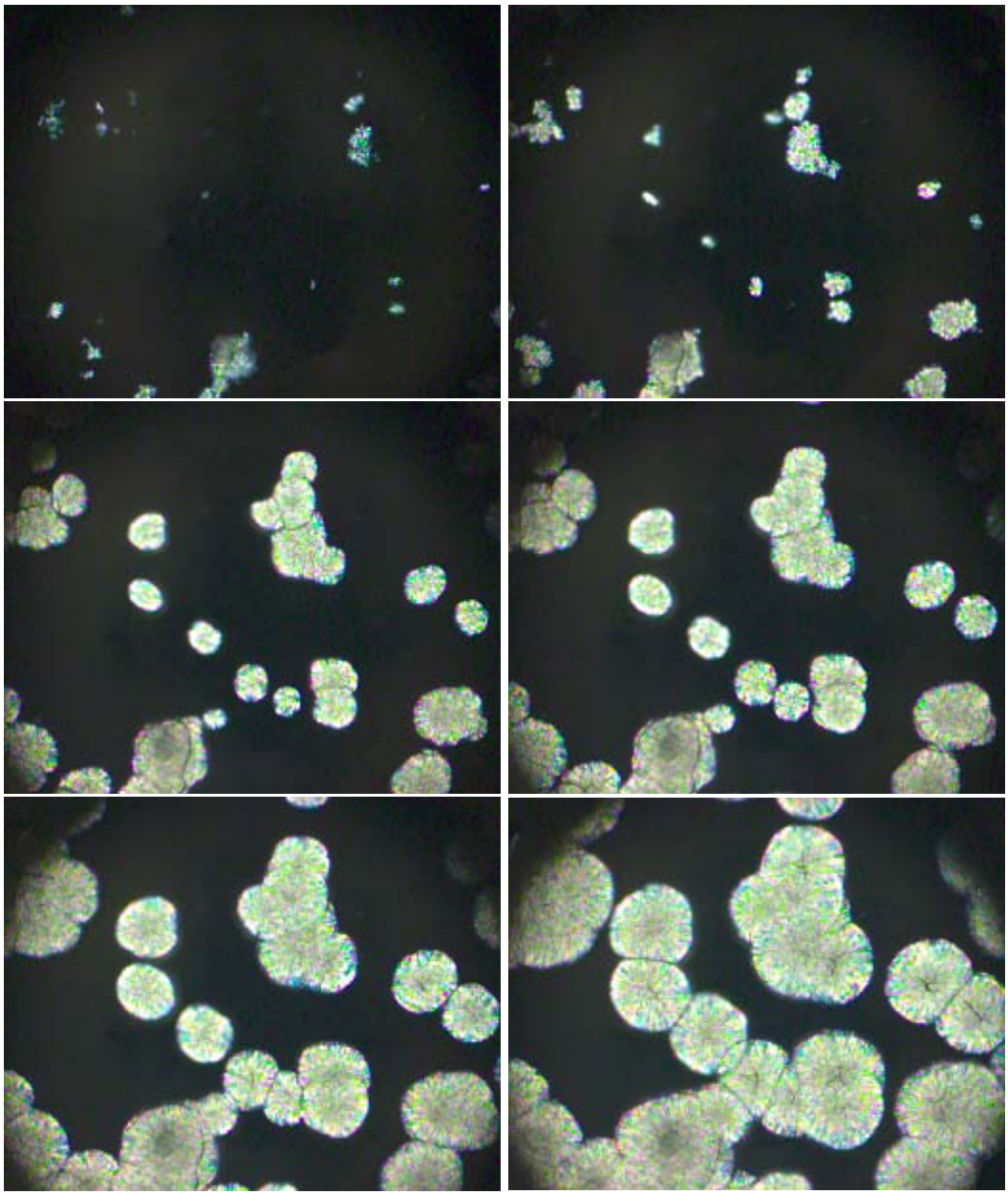


Figure 17: Crystallization Event for PEO

After the experiment has concluded, the student can repeat the experiment at different crystallization temperatures. The student quits the applet after completing all experiments and logs in to the Framework Server to analyze the collected images.

CHAPTER 10

Concluding Remarks and Future Work

The Polymer Crystallization iLab makes possible an educational experience not previously available to most science and engineering undergraduates. By creating a software system with a web-based interface to laboratory equipment, we have enabled students to remotely conduct polymer crystallization experiments using optical microscopy. Such experiments allow students to characterize the crystallization properties of polymers by observing and analyzing polymer crystallization events. Students in laboratory classes and in traditional engineering classes can use this educational online laboratory to solidify the concepts studied in their coursework. The cost barrier to equipment is solved by multiplexing a single setup and allows students to access the equipment at any time of day. Staffing requirements are kept to a minimum by designing software logic that directs students and by embedding security and safety checks into the software system.

The Polymer Crystallization iLab provides an example of an interactive online laboratory, where students can remotely control the microscope and associated hardware with minimal latency. Students can receive real-time streaming temperature and image updates as if they were viewing the experiment in the laboratory. In addition, the necessary software to process images has been included so students can analyze their

results after an experiment has been conducted.

Additional capabilities can be added to the Polymer Crystallization iLab to enhance the educational experience. For example, students could benefit from a collaborative environment if multiple clients were allowed to access the system simultaneously. This added functionality would require a distinction between a lab controller and a lab observer, to ensure that only one person had control of the microscope at a given time. This functionality could easily be added by employing a token passing scheme and using the structure maintained in the Microscope Server from the MEMS project [7].

The next stage of development for the Polymer Crystallization iLab could also involve an integrated simulation tool, so students can study the relationship between the theoretical models encountered in class and actual observations from the laboratory. In addition, the ImageJ applet can be altered so that students can engage in analysis specific to the polymer crystallization experiment. Currently, the entire image analysis tool is provided to the students; however, a tool with more limited and directed functionality would prove beneficial to assist students in their analysis. Finally, the Polymer Crystallization iLab will undergo extensive testing as it is deployed in the Fall 2003 undergraduate polymer laboratory at M.I.T. Development revisions likely will be made to the system based on user feedback.

Educational online laboratories will augment students' understanding of scientific and engineering concepts by allowing them to evaluate theoretical ideas applied in a laboratory setting. This paradigm of online laboratories offers engineering educators the opportunity to provide an enhanced educational experience to their students. Future technologies will further develop and expand the number of online laboratories in deployment and will help to broaden laboratory experiences among all science and engineering students.

Appendix A: Administrative User Manual

The following Administrative User Manual assumes that the Microscope Server and the Framework Server are running on a machine with the hostname, PolymerLab.mit.edu, and that the Microscope Client and Microscope Server code can be found in the following directory: C:\ILab\RemoteMicroscope.

A.1 Program Requirements

This application requires *Windows 2000* or *Windows XP* operation system with IIS and the .NET Framework installed. To be able to run and compile the Microscope Server, the following packages need to be installed on the hosting machine:

- Python 2.2.2 : http://www.python.org/
- The Python Imaging Library 1.1.4 (Windows version) : http://www.pythonware.com/products/pil/
- Python Win32 Extension Package for Python 2.2, Win32all version 152 : http://starship.python.net/crew/mhammond/win32/Downloads.html
- Serial Communication Extension Package, SioModule22 : http://starship.python.net/crew/roger/

To edit and compile the Microscope Client applet you will need the following installed:

- Java 2 SDK version 1.4.1_02 : http://java.sun.com/
- Java SunONE Studio 3 Update 1 Community Edition: http://java.sun.com/

To run the Microscope Client applet, users will need the following installed:

- Java Plug-in 1.4.0 or higher:
  http://java.sun.com/
- Quick Time Plug-in with SMIL interpreter (optional for viewing slide show)
  http://quicktime.apple.com
- Real Player (optional for viewing slide show)
  http://www.real.com/

To run and compile the Framework Server, the following software must be installed:

- .NET Framework Version 1.1
  http://www.microsoft.com/net/
- VisualStudio.NET
  From VisualStuido.NET CDs
- Internet Information Services
  Included in Windows XP CD
- SQL Server 2000 with Service Pack 3
  http://www.microsoft.com/sql/downloads/default.asp and SQL CD

A.2 Compiling Instructions

For the Microscope Server no compilation is necessary because the Python language is an interpreted language that does not require compiling. Although Python files can be compiled into .pyc or .pyo files, there currently is no benefit to doing so. Therefore, the Microscope Server is currently used as a scripted application. Modifications need only be made and saved to the corresponding .py to alter the behavior of the Microscope Server.

To compile the client application using the command line:

- Go to the appropriate directory:

  cd C:\ILab\RemoteMicroscope

- To compile the code:

  javac –classpath C:\ILab\RemoteMicroscope client\ScopeFormApplet.java

- To deploy a release version of the code you need to create a jar file and move it to the C:\inetpub\wwwroot\PolymerLab directory, which is the directory where the Framework Server resides. To do this a batch script has been included in the directory C:\ILab\RemoteMicroscope called deploy.bat. Running this batch script will deploy a release version in the appropriate directory.

To compile the Framework Server, it is suggested that Visual Studio .NET be used. Although it is possible to compile the program outside of Visual Studio, this is not recommended because of the complexity of the project. The Framework Server is stored under the Visual Studio project called PolymerLab.

A.3 Running Instructions

To run the Microscope Server, first make sure that the sample is centered in the MDS600 heating stage. Since the MDS600 is not equipped with a reference motor, the initial starting position will be used as a reference position for the center of the polymer sample:

- Go to the appropriate directory:

  cd C:\ILab\RemoteMicroscope

- Run the starting script:

  startServer.py [options]

- The available options for the script are the following:

-d, --debug	Display debugging trace messages
-f, --cfg <file>	Use specified configuration file
-h, --help	Display this help message

-t, --timing	Display timing trace messages
-v, --verbose	Display verbose activity messages

To run the client applet, log on to the Framework Server and follow the links to reserve and open the polymer lab.

A.4 How to add more files to the HTTP server

The HTTP server can read files that are located in the following directory:

C:\inetpub\wwwroot

The HTTP server is a development version of Microsoft's IIS web server. On Windows XP Professional, there are a maximum of 10 simultaneous sessions supported. In order to allow more simultaneous connections, the operating system will have to be upgraded wo Windows XP Server or downgraded to Windows 2000 Server.

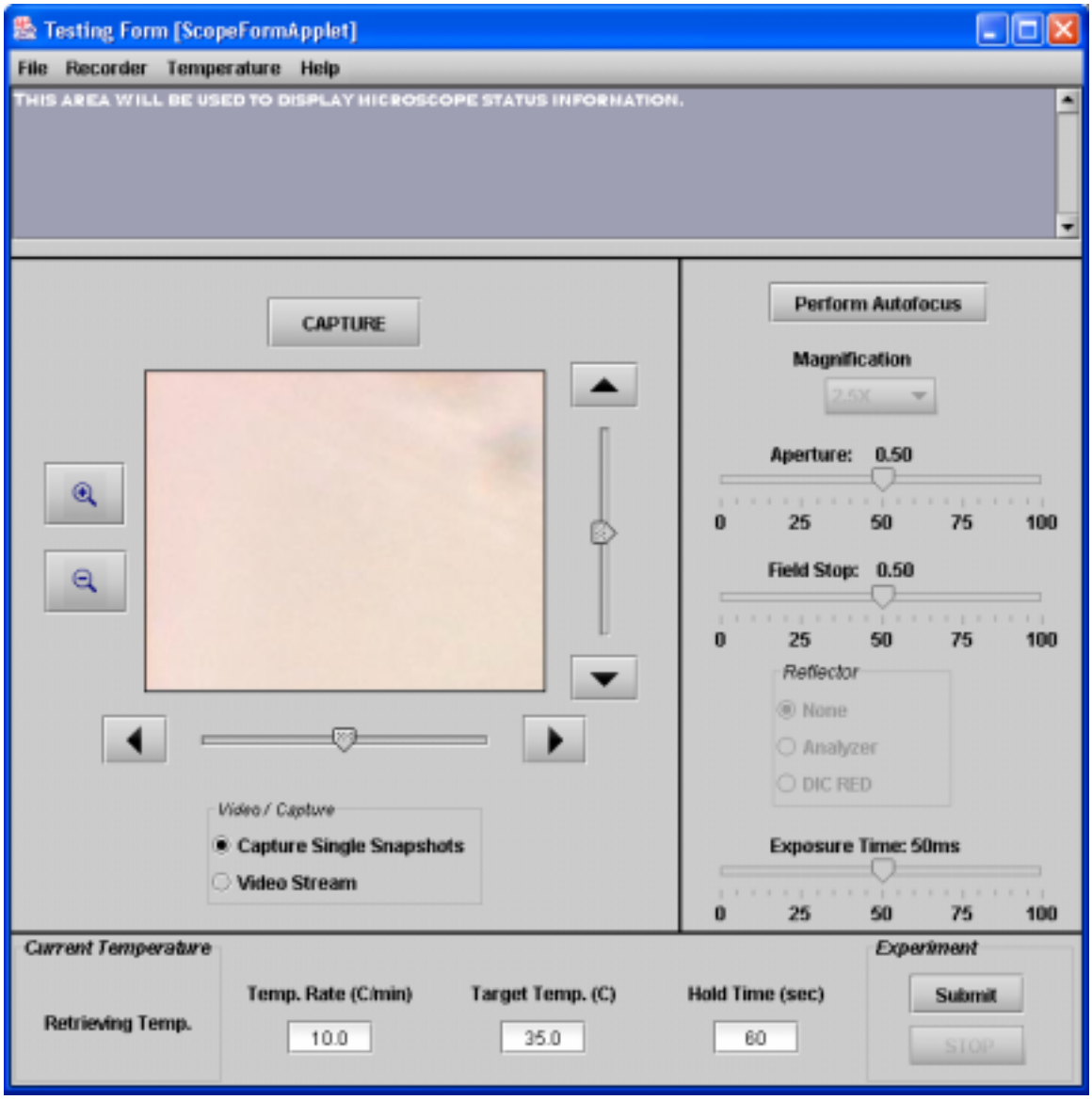
Virtual Directories can be added to expose any directories on the file system to the Web Server. Currently, there is a virtual directory called iLab, accessible through http://polymerlab.mit.edu/iLab. This virtual directory points to the local directory C:\inetpub\wwwilab\. Under this virtual directory, the users directory is where all experiment run images and SMIL output files are saved to the file system.

More web applications can be added to the web server by adding new web applications in VisualStudio.NET. This IDE automatically makes the necessary changes to the file system to allow the web application to be run remotely.

Appendix B: Student User Manual

This user manual is intended to explain how to use the student client interface for the Microscope Client. It explains what each graphical component does, and how it should be used.

Microscope Client



Capturing Images

To capture microscope images, first the user has to make sure the “Capture Single Snapshots” is selected on the *Video/Capture* control. Then every time the CAPTURE button is pressed an image is captured and it displayed on the screen.

Video Streaming

To start video streaming select the “Video Stream” selection on the *Video/Capture* control. The capture button is disabled, and a sequence of images is captured and displayed on the screen.

Moving the Image

To move the image around use the arrow buttons and sliders located to the right and bottom of the image display panel. Whenever any of these arrow buttons are pressed the XY stage moves, and if “Capture Single Snapshots” is selected a new image is captured and displayed on the screen immediately after the stage finishes moving.

Perform Autofocus

The *Perform Autofocus* button calls the one-time autofocus function, and returns a new focused image that is displayed on the image panel. The autofocus main purpose is to search for the right z stage position for adequate focusing.

Magnification

The *Magnification* control changes the objective lens being used. The possible magnifications are: 2.5X, 5X, 10X, 20X, and 50X.

Aperture

The *Aperture* control changes the condenser aperture setting. A condenser has the role of collecting, controlling and concentrating the light from the lamp onto the specimen. The aperture of the condenser serves to control the angle of the cone of light emerging from the top of the condenser. When the aperture is set to the maximum (0.95) the objective provides maximum resolution, but some glare may be present, which reduces image contrast. If the aperture is adjusted to about 0.70 the glare is reduced and contrast is improved, without significant lose of image detail. Lowering the aperture increases contrast but image detail will be lost.

The aperture setting should only be lowered for magnifications greater than 10X, because in lower magnifications the field of vision is greatly reduced when lowering the aperture. Therefore, for optimal performance maintain the aperture above 0.70 when using 2.5X and 5X objectives.

Field Stop

The field stop allows you to control the amount of light entering the system as well as the field of view. The field stop is basically a plate with a hole on it placed on the optical axis. This control is useful mainly to control the light illumination for the lower magnifications such as 2.5X, and 5X. For higher magnifications the field stop should be set to the highest value, and only use the Aperture control to adjust brightness and contrast.

Reflectors

With the reflector control you are able to select between an analyzer, a DIC_RED reflector, and no reflector at all. A much higher exposure time is always needed when using the DIC_RED reflector or the analyzer, than when not using a reflector at all.

Exposure Time

The exposure time controls the shutter speed of the camera. The normal setting for the exposure time is 1 ms. If the user is using the analyzer or the DIC_RED reflector then to get a clear image the exposure time has to be increased to around 20 ms.

Running Experiments

Experiments can be run by inputting the desired target temperature, ramp rate, and hold time into the Temperature Panel at the bottom of the Microscope Client. Only valid parameters will be accepted; invalid parameters will be flagged by a popup dialog window. After the desired parameters have been entered, press the SUBMIT or RUN button at the bottom right of the Microscope Client. The analyzer should be in place to view the polymer melting event. In addition, the Video Stream button should be selected to view images at the fastest possible rate. Once the polymer has melted, the appropriate cooling rate, target isothermal crystallization, and hold time should be entered into the Temperature Panel and the SUBMIT button should be pressed to regain temperature control conditions.

Recording Experiments

The Recorder menu allows a user to specify when experiments are recorded and images are saved to the server. NOTE: The video streaming button must be pressed to save images, as the images are currently saved in the Video Stream. When sufficient images have been recorded (the number should not exceed 150), the recorder can be stopped, and the experiment can be accessed through the Framework Server.

For further client instructions, please refer to the Full Student Manual at http://polymerlab.mit.edu.

Appendix C: SQL Database Script

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[FK_Experiment_Runs_Experiments]') and OBJECTPROPERTY(id, N'IsForeignKey') = 1)
ALTER TABLE [dbo].[Experiment_Runs] DROP CONSTRAINT FK_Experiment_Runs_Experiments
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_Add]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_Add]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_Delete]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_Delete]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_GetRunFromExpID]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_GetRunFromExpID]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_GetRunFromID]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_GetRunFromID]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_GetRunFromName]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_GetRunFromName]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_GetRunFromUserID]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_GetRunFromUserID]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_GetRunFromUserIDAndExpID]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_GetRunFromUserIDAndExpID]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_IsRun]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_IsRun]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[ExperimentRun_Update]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[ExperimentRun_Update]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiment_CreateNew]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[Experiment_CreateNew]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiment_GetAllExp]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[Experiment_GetAllExp]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiment_GetExpFromID]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[Experiment_GetExpFromID]
```

```
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiment_GetExpFromName]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[Experiment_GetExpFromName]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiment_IsExperiment]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[Experiment_IsExperiment]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiment_Update]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
drop procedure [dbo].[Experiment_Update]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiment_Runs]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Experiment_Runs]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Experiments]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Experiments]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Reservations]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Reservations]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Role_To_Role_Map]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Role_To_Role_Map]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Roles]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Roles]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[User_To_Role_Map]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[User_To_Role_Map]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Users]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[Users]
GO

CREATE TABLE [dbo].[Experiment_Runs] (
        [RunID] [uniqueidentifier] NOT NULL ,
        [ExperimentID] [uniqueidentifier] NOT NULL ,
        [RunName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Description] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [UserID] [uniqueidentifier] NOT NULL ,
        [AddDate] [datetime] NOT NULL ,
        [OutputXml] [ntext] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [expdate] [datetime] NOT NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

CREATE TABLE [dbo].[Experiments] (
        [ExperimentID] [uniqueidentifier] NOT NULL ,
        [ExperimentName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Description] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [RegistrationDate] [datetime] NOT NULL ,
        [CreatorID] [uniqueidentifier] NULL
) ON [PRIMARY]
GO
```

```
CREATE TABLE [dbo].[Reservations] (
	[UserID] [uniqueidentifier] NOT NULL ,
	[Priority] [tinyint] NOT NULL ,
	[StartTime] [datetime] NOT NULL ,
	[EndTime] [datetime] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Role_To_Role_Map] (
	[RoleIDA] [uniqueidentifier] NOT NULL ,
	[RoleIDB] [uniqueidentifier] NOT NULL ,
	[DateSubmitted] [datetime] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Roles] (
	[RoleID] [uniqueidentifier] NOT NULL ,
	[RoleName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[RoleDescription] [varchar] (7000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[User_To_Role_Map] (
	[UserID] [uniqueidentifier] NOT NULL ,
	[RoleID] [uniqueidentifier] NOT NULL ,
	[DateSubmitted] [datetime] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Users] (
	[UserID] [uniqueidentifier] NOT NULL ,
	[FirstName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[LastName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[Email] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[Password] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[SchoolID] [int] NULL ,
	[RegistrationDate] [datetime] NOT NULL ,
	[ConfirmationDate] [datetime] NULL ,
	[ConfirmationCode] [uniqueidentifier] NULL
) ON [PRIMARY]
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO


CREATE PROCEDURE dbo.ExperimentRun_Add
	@RunID uniqueidentifier,
	@ExpID uniqueidentifier,
	@RunName varchar(50),
	@Desc varchar(500),
	@UserID uniqueidentifier,
	@AddDate datetime,
	@OutputXml ntext
AS
	insert into Experiment_Runs
	(RunID, ExperimentID, RunName, Description,  UserID, AddDate, OutputXml)
	values
	(@RunID, @ExpID, @RunName, @Desc, @UserID, @AddDate, @OutputXML)

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF
GO
```

```
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.ExperimentRun_Delete
	@RunID uniqueidentifier
 AS

	delete from Experiment_Runs where RunID = @RunID
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.ExperimentRun_GetRunFromExpID
	@ExpID uniqueidentifier
AS
select RunID, ExperimentID, RunName, Description, UserID, AddDate, OutputXml
from Experiment_Runs where ExperimentID=@ExpID order by AddDate desc


GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO


CREATE PROCEDURE dbo.ExperimentRun_GetRunFromID
	@RunID uniqueidentifier
AS
	select RunID, ExperimentID, UserID, RunName, Description, AddDate, OutputXml
	from Experiment_Runs
	where RunID = @RunID

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO


/****** Object:  Stored Procedure dbo.FW_ExperimentRun_GetRunFromName    Script Date: 9/26/2002 1:22:07 PM ******/
CREATE PROCEDURE dbo.ExperimentRun_GetRunFromName
	@RunName varchar(50)
AS
	select RunID, ExperimentID, UserID, RunName, Description, AddDate, OutputXml
	from Experiment_Runs
	where RunName = @RunName

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
```

```
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.ExperimentRun_GetRunFromUserID
	@UserID uniqueidentifier
AS
select RunID, ExperimentID, RunName, Description, UserID, AddDate, OutputXml
from Experiment_Runs where UserID=@UserID order by AddDate desc


GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.ExperimentRun_GetRunFromUserIDAndExpID
	@UserID uniqueidentifier,
	@ExpID uniqueidentifier
AS
select RunID, ExperimentID, RunName, Description, UserID, AddDate, OutputXml
from Experiment_Runs where ExperimentID=@ExpID and UserID=@UserID
order by AddDate desc


GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO


/****** Object:  Stored Procedure dbo.FW_ExperimentRun_IsRun    Script Date: 9/26/2002 1:22:07 PM ******/
CREATE PROCEDURE dbo.ExperimentRun_IsRun
	@RunID uniqueidentifier
AS
	select count(1) from Experiment_Runs where RunID=@RunID

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO


CREATE PROCEDURE dbo.ExperimentRun_Update
	@ExpID uniqueidentifier,
	@Name varchar(50),
	@Desc varchar(500),
```

```
	@RunStatus varchar(50),
	@UserID uniqueidentifier,
	@AddDate datetime,
	@RunID uniqueidentifier,
	@OutputXml ntext
AS
	update Experiment_Runs
	set ExperimentID=@ExpID, RunName=@Name, Description=@Desc,
	@UserID=UserID, AddDate=@AddDate, OutputXml=@OutputXml
	where RunID = @RunID

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO



CREATE PROCEDURE dbo.Experiment_CreateNew
	@ExpID uniqueidentifier,
	@ExperimentName varchar(50),
	@Desc varchar(500),
	@RegDate datetime,
	@CreatorID uniqueidentifier
AS
	insert into Experiments
	(ExperimentID, ExperimentName, Description, RegistrationDate, CreatorID)
	values (@ExpID, @ExperimentName, @Desc, @RegDate, @CreatorID)


GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.Experiment_GetAllExp
AS
	select ExperimentID, ExperimentName, Description, RegistrationDate, CreatorID
	from Experiments order by ExperimentName

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.Experiment_GetExpFromID
	@ExpID uniqueidentifier
AS
	select ExperimentID, ExperimentName, Description, RegistrationDate, CreatorID
	from Experiments where ExperimentID=@ExpID
```

```
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.Experiment_GetExpFromName
	@ExpName uniqueidentifier
AS
	select ExperimentID, ExperimentName, Description, RegistrationDate, CreatorID
	from Experiments where ExperimentName=@ExpName

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.Experiment_IsExperiment
	@ExpID uniqueidentifier
AS
	select count(1) from Experiments where ExperimentID=@ExpID


GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO


CREATE PROCEDURE dbo.Experiment_Update
	@ExpID uniqueidentifier,
	@ExperimentName varchar(50),
	@Desc varchar(500),
	@RegDate datetime,
	@CreatorID uniqueidentifier
AS
update Experiments set ExperimentName=@ExperimentName,
Description=@Desc, CreatorID=@CreatorID, RegistrationDate=@RegDate
where ExperimentID=@ExpID

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

References

[1] P. Nasser. "*Remote Microscope For Polymer Crystallization WebLab*", M.Eng. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, September 2002.

[2] L. Wu, M.S. Lisowski, S. Talibuddin, and J. Runt. "*Crystallization of poly (ethylene oxide) and melt-miscible PEO Blends*". Macromolecules, 1999.

[3] S. Magonov, and Y. Godovsky, "*Atomic Forces Microscopy of Polymers: Studies of Thermal Phase Transitions*". Digital Instruments, 2001

[4] J. Kao. "*Remote Microscope for Inspection of Integrated Circuits,*" S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, September 1995.

[5] S. Kittipiyakul. "*Automated Remote Microscope for Inspection of Integrated Circuits,*" S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, September 1996.

[6] D. Seth. "*A Remotely Automated Microscope for Characterizing Micro Electromechanical Systems (MEMS)*", S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2001.

[7] A. Kuchling. "*Internet Access to an Optical Microscope,*" *http://www.mems-exchange.org/software/microscope/publications/ipc7-abstract.html*, March 2002.

[8] Microelectronics WebLab at MIT. http://weblab.mit.edu

[9] E. Anderson, P. Greenspun, A. Grumet. "*The Internet Application Workbook,*" *http://philip.greenspun.com/internet-application-workbook/*

[10] *"W3C SMIL Recommendation," http://www.w3.org/AudioVideo/*

[11] L. Hui. "*I-Lab Webpage*," *http://i-lab.mit.edu.* January 2001.

[12] J. Kao, D. Troxel, S. Kittipiyakul. "*Internet Remote Microscope*," [Telemanipulator and Telepresence Technologies III ,SPIE Proceedings, 2901, (Nov. 18-19, 1996) , Boston, MA.].

[13] M. Perez. "*Java Remote Microscope for Collaborative Inspection of Integrated Circuits*," M. Eng. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May, 1997.

[14] iCampus Project Webpage.
http://www.swiss.ai.mit.edu/projects/icampus/index.html, 2002

[15] "*What is Python*?" *http://www.python.org/doc/Summary.html*

[16] "*Java Foundation Classes (JFC)," http://java.sun.com/products/jfc/#components*