Online Polymer Crystallization Experiment

by

Derik A. Pridmore

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

January 28, 2005

Author_____
Department of Electrical Engineering and Computer Science
January 28, 2005

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

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

Online Polymer Crystallization Experiment

by
Derik A. Pridmore

Submitted to the Department of Electrical Engineering and Computer Science
January 28, 2005
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

An architecture for online remote operation of a polymer crystallization experiment was refined, beta tested in actual use conditions, and extended based on feedback from those tests. In addition, an application for graphically simulating macroscopic crystal spherulite growth was developed for use as an educational tool. Finally, the experiment was used in the design process for modifying the generic iLab framework to incorporate interactive functionality. Specifically, a reservation model and design changes to the experiment storage and service broker were proposed based on the Polymerlab, and the experiment was used as a testbed for initial implementation of some of the proposed systems.

Thesis Supervisor: Gregory C. Rutledge
Title: Professor of Chemical Engineering

Acknowledgments

I would like to thank Prof. Greg Rutledge, Enid Choi, Daniel Talavera, Jon Salz, Sid Henderson, and Dan Lovell for their help and support.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Objective

This paper focuses on the development and deployment of the polymer crystallization iLab, a remote Internet laboratory for conducting polymer crystallization experiments using optical microscopy. Using the polymer crystallization iLab, students learn about the fundamentals of polymer physics while remotely conducting this experiment and recording data in the form of digital images from an optical microscope. Through the use of digital image analysis, students analyze the data obtained to derive the crystallization properties of a specific polymer and characterize the crystallization kinetics.

This project had several objectives. The first was to evaluate the security, functionality, and utility of the polymer crystallization iLab, or Polymerlab, system by deploying it in beta testing for the first time with student users in an actual test, and then to respond to design shortcomings identified through user feedback. An additional goal was to modify the existing Polymerlab framework to enable multiple users to view and control the experiment at once, to further the goal of real-time remote collaboration between users. Another goal was the development of a graphical crystal simulation tool that would allow students users to compare their experiments to theoretical predictions in an intuitive way. Finally, initial steps in the integration of the Polymerlab experiment, the only

iLab experiment to use real-time user control in its experiments, with the generic iLab framework were taken.

1.2 Purpose and Motivation

There are several motivating factors behind the development of the Polymerlab and iLabs in general. One of the most important is to enable scarce and expensive equipment to be used by students who might not otherwise be able to. In the case of the Polymerlab, the optical microscope, associated controllers, and sample stage necessary to carry out the experiment are quite costly. Since only one set of equipment is available, and it is delicate, using the lab in a typical classroom setting would require both extreme care and tight scheduling. In addition, lab personnel are required to supervise whenever students might make use of the experiment. Thus, while making experimentation an integral part of a student's curriculum enhances the educational experience, there are cost barriers to doing so. Developing a web mediated interface for the experiment allows for more efficient sharing of resources in order to overcome cost barriers, while also enabling built in hardware protection to reduce the need for supervision. These cost savings might be distributed over a group of universities not bound geographically in order to expand the experimental apparatus available to their respective students. Furthermore, the ability to access the experiment via the web allows for innovations in the classroom, by allowing instructors to illustrate their lectures with live demonstrations of experiments. Internet laboratories can also be used 24/7, not just during scheduled lab hours. This gives students more flexibility and allows more efficient use of time.

Another motivation for developing the Polymer crystallization iLab is to facilitate group learning. An important trend in education has been the integration of both experimentation and collaboration in degree programs, especially in the areas of science and engineering. Once a system is developed to allow single users to share a resource such as the Polymerlab equipment remotely via the internet, allowing remote collaboration between multiple users is only a matter of extending the software design. In contrast, the physical apparatus of the experiment might not be amenable to such group collaboration due to, for example, laboratory space constraints or equipment design.

1.3 Background

1.3.1 Polymer Crystallization Experiment

The polymer crystallization experiment is used as part of an undergraduate course in the Chemical Engineering Department called 10.467, Polymer Science Laboratory. The experiment, a standard example used in teaching polymer physics, involves heating a polymer sample (in this case PEO, or polyethylene oxide) above its melting point and subsequently cooling it to various controlled temperatures in order to observe the characteristics of crystal formation under isothermal conditions. Crystals are imaged using a polarized light microscope, which can be difficult to obtain.

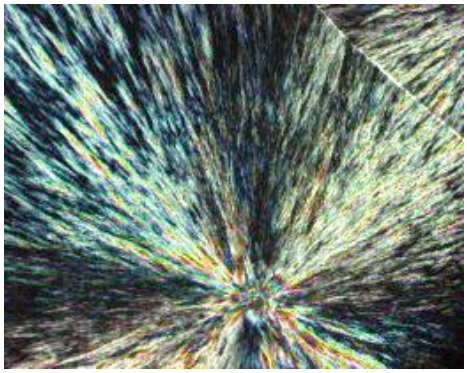


Figure 1: PEO crystal imaged using Polymerlab

When imaging crystallization, spherulites are observed. Spherulites are crystal particles which grow radially and exhibit spherical symmetry. By observing the rate of formation of crystal spherulites and the rate of growth of those spherulites as a function of temperature, students can determine certain fundamental physical properties of the sample. By comparing the functional form of the dependencies of transformed volume or crystal growth rate on temperature to those predicted by theory students may obtain physical characteristics such as activation energy, average surface energy, and Avrami exponent of the thin film crystallization [1]. The exact manner of determining these values will be discussed later in Chapter 5.

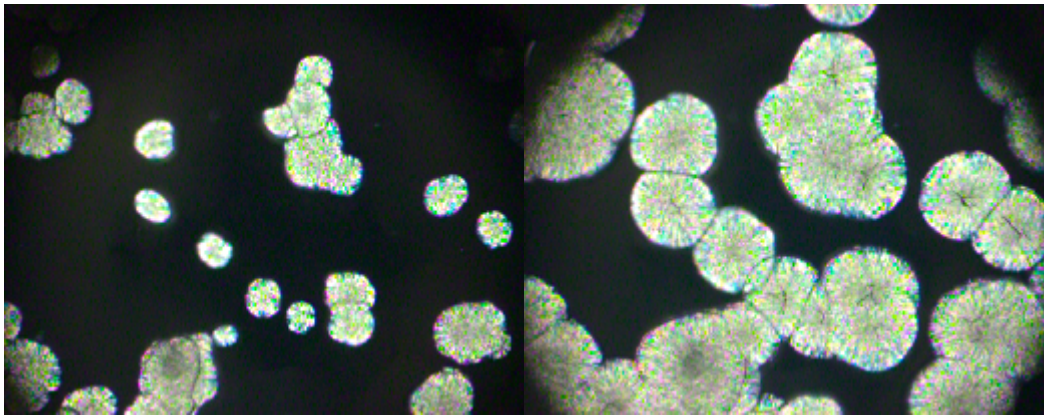


Figure 2: Growth of spherulites during crystallization. The image on left shows initial crystallization; the image on the right shows the same sample at a later time.

Unlike some experiments, the polymer crystallization experiment may be cycled repeatedly with no user intervention to change or renew the polymer sample. In theory, once the experiment has been set up, it is self-contained and may be used continuously. In addition, with proper sample preparation, the experiment should be memoryless: previous crystallization results should not affect subsequent runs. Therefore, the polymer crystallization experiment is especially suited to the goals of an online laboratory. In particular, the experiment can be run at any time convenient for the student without instructor oversight, preparation, or intervention.

1.3.2 Related Work

The work contained in this thesis has been a direct continuation of work performed by Paola Nasser and Daniel Talavera in connection with their Masters of Engineering theses. In her thesis, Paola Nasser developed a Remote Microscope consisting of a light microscope and a digital camera controlled via communication between a Java Applet on a user's local machine and a microscope server [2]. The client had capabilities to display images captured by the digital camera and 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. Daniel Talavera's thesis expanded both the microscope client and server to enable additional functionality, including the capability to control a heating stage, the XY position of the sample in the microscope, and the focus of the polymer

sample, and to save experimental runs on the server. Additionally, the video streaming rate of the system was improved and an on-line environment was created to enable users to store and analyze data, as well as manage laboratory reservations [3].

Much of the development done in connection with the Polymerlab experiment has been directly influenced by previous projects related to remote control of microscopy systems. Among these are James Kao [4] and Somsak Kittipiyakul's [5] Internet Remote Microscope, a remote automated microscope developed by a MEMS research group at MIT [6], as well as a DARPA sponsored open-source project [7]. Each of these systems impacted the design of the Polymerlab in multiple ways. The most important of these, the MEMS project, was developed by A. M. Kuchling and uses a simple asynchronous message passing protocol ((the Microscope Networking Protocol Specification) between the microscope and user (server and client) to set microscope parameters and request images. This protocol formed the basis for Paola Nasser's Remote Microscope design.

1.3.3 iLab Project

The iLab project began in June of 2000 as part of the iCampus initiative, a 5 year, $25 million research alliance between MIT and Microsoft to enhance university education through information technology [9]. The goal of the iLab project was to create internet accessible laboratories. Among these laboratories were the polymer crystallization experiment, a heat exchange experiment, and a microelectronics lab.[1] Additional experiments were added as the project grew. Each of these experiments represents a distinct set of requirements for an online experiment. As discussed above, the Polymer Crystallization iLab serves as a prototype of an interactive experiment. In contrast, the Microelectronics WebLab represents a "stateless" experiment [8]. 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 additional interaction.

---

[1] The Heat Exchange experiment may be found at http://heatex.mit.edu, and the Microelectronics WebLab may be found at http://ilabserv.mit.edu.

After these labs met their initial goals, work shifted to designing a common support infrastructure that would allow the more general aspects of online laboratories to be maintained as part of a common system, to reduce redundancy and make code maintenance easier. This system would separate the administrative components which would be common to all online laboratories, such as user authentication and result storage, from experiment specific components. While part of the work detailed in this thesis was being carried out, the first laboratory supported by this general iLab framework debuted in January of 2004. This framework, however, included only support for a general type of experiments known as "batched" experiments. These experiments do not require user interaction. After this initial implementation, work was begun on modifying the batched experiment architecture to support user-interactive experiments. Experiment interfaces and communication protocols would need to be changed to address the problems inherent to real-time control of experiments by users. Indeed, the very concept of a reservation system would need to be introduced, as the previous "batched" experiments could be scheduled automatically by a server. The details of many of these modifications will be discussed later in this thesis. The current Polymerlab serves as a testbed for software design to support interactive experiments.

1.4 Development

This section provides a description of the various programming languages and programming environments used in creating components of this lab. This description is necessary not only to detail the earliest design choices made in implementing components but also for future maintenance.

1.4.1 Java

Both the Microscope Client and the Simulation module are implemented in Java. Java was originally chosen by Paola Nasser to implement the earliest version of the Microscope client, because of its simple native support for messaging over sockets, its

easy use of pre-packaged GUI (Graphical User Interface) components, its object oriented structure, and the relative pervasiveness of the Java within most browser applications. Because the system uses full duplex communication between the client and server, the client must be able to decipher messages sent over sockets. The use of Java allows for a thinner client to be downloaded by users by relying on built-in Java classes and functionality, which saves bandwidth and time, as well as ensures compatibility with most if not all computing environments. 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. (In Appendix D, the User Survey Responses, a number of browsers which have been used with the experiment are listed.)

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. The Simulation module was developed using a simple Emacs editing environment, but both the Microscope client and simulation module can easily be opened and edited in any of a number of standard Java editing environments and IDEs (Integrated Development Environment).

1.4.2 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. Python was chosen due to its efficient string manipulation and dictionary management. The hardware modules implemented in Python communicate with the various hardware devices using serial ports and Microsoft's Component Object Model (COM), through simple yet efficient modules available in the Win32 package. Another freely available library, the Python Imaging Library (PIL), allows images to be manipulated and converted to different formats.

Editing and debugging of the Python code was done using an 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, compiling, and debugging code.

1.4.3 C#

The Framework Server, used for post-experimental data analysis and user management, is implemented using C# and Microsoft's .NET framework. C# was chosen as the development language for this component because of its seamless integration into ASP.NET web pages and its simple database access mechanism. Both properties of the language greatly simplify the task of adding web services such as user verification and management. In addition, .NET applications are easily hosted using Microsoft's IIS web server. VisualStudio.NET was used as the IDE for developing, debugging, and extending the Framework server.

CHAPTER 2

System Overview

The polymer crystallization iLab is made up of several independent yet interconnected software and hardware systems. The system was implemented in order to best reflect the use and structure of the polymer crystallization experiment itself, and to achieve maximum design flexibility. On the software side, for example, the system provides a level of abstraction between client and the microscope servers which in some respects mimics the actual physical use of such an experiment, yet still maintains design flexibility to allow for changes of the type discussed later in this thesis. Additionally, the separation of these components allows for development in a respective language that offers the most powerful benefits, as has been discussed previously. On the hardware side, the layout and interoperability of components was, of course, influenced greatly by the commercial availability and compatibility of the individual components, but has still been optimized to suit the desired functionality of the system. The following chapter provides a detail overview of these system components and their operation.

2.1 System Architecture

The general architecture of the Polymerlab system includes three separate components: the Microscope Client, the Microscope Server, and the Framework Server. Figure 3 provides a simple graphical representation of these three components and the flow of communication between them. In keeping with the original description of the system, the Microscope client and Microscope server, shown grouped together at the bottom of the figure, are known collectively as the Remote Microscope. While the Framework server handles the administrative tasks of user authentication, account management, reservation

scheduling, group management, and data storage and retrieval, the Remote Microscope provides the instrument functionality of the various microscope and sample manipulation hardware that one would typically associate with such a laboratory experiment. Communication occurs between each of the three primary software elements of the Polymerlab system. While the details of each leg of this communication is discussed below, it is important to note that this three-way communication is currently unique to the Polymer crystallization iLab, and is crucial for the user interaction and image streaming needed for such an experiment. This detail will be discussed later in Chapter 9 in relation to the current state of the generic iLab architecture.

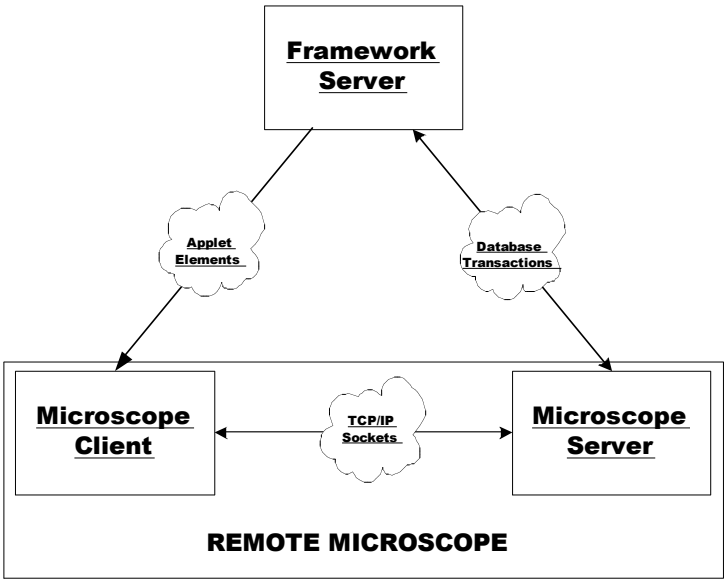


Figure 3: General architecture overview

2.2 Framework Server

As discussed above, the Framework server handles the administrative aspects of managing the Polymer Crystallization iLab. It is a three-tier system consisting of a series of dynamically generated web forms and services backed by database transactions which are served through any standard web browser. The Framework server uses Microsoft's .NET platform to generate these services. This platform has many characteristics that make it an ideal choice. The C# programming language used in developing .NET applications is an object oriented language, allowing for a very intuitive internal

representation of procedures and components. The development environment also enables complex functions to be compiled and run locally, rather than transmitted to the user to be run, which saves time for users. It also has built-in APIs for database transactions. Easy compatibility with Microsoft's IIS web server makes hosting .NET web services relatively simple, which is a consideration in the long term maintenance of the Polymer Crystallization iLab.

Within the Framework server, the individual components are closely related but can be loosely described as a three-tier system. The first tier is actually the client browser. Because of the ubiquitous availability of numerous web browsers and because .NET applications are compile and run on the host server, this first tier is remarkably thin and fast, requiring no lengthy download or installation and using very little bandwidth resources. Figure 4 shows the typical page served to a student user. The details of the layout and control flow of the interface provided to students and administrators are omitted here, and readers are referred to Daniel Talavera's thesis.

The middle tier of the Framework server consists of a series of web services implemented in ASP.NET which provide the actual mechanisms for user authentication, account management, and data retrieval. Most of these services make integral use of a system of object oriented classes implemented in C# which provide wrappers for use by the web services in accessing and modifying information stored in the systems database. Examples include User objects, Role objects which are assigned to users and help define their system privileges, and Experiment Run objects which correspond to individual experiment runs. These classes will be discussed later when describing additions made to allow Multicasting. Figure 4 shows an example of a web interface dynamically generated by the Framework Server showing a user's experiments.

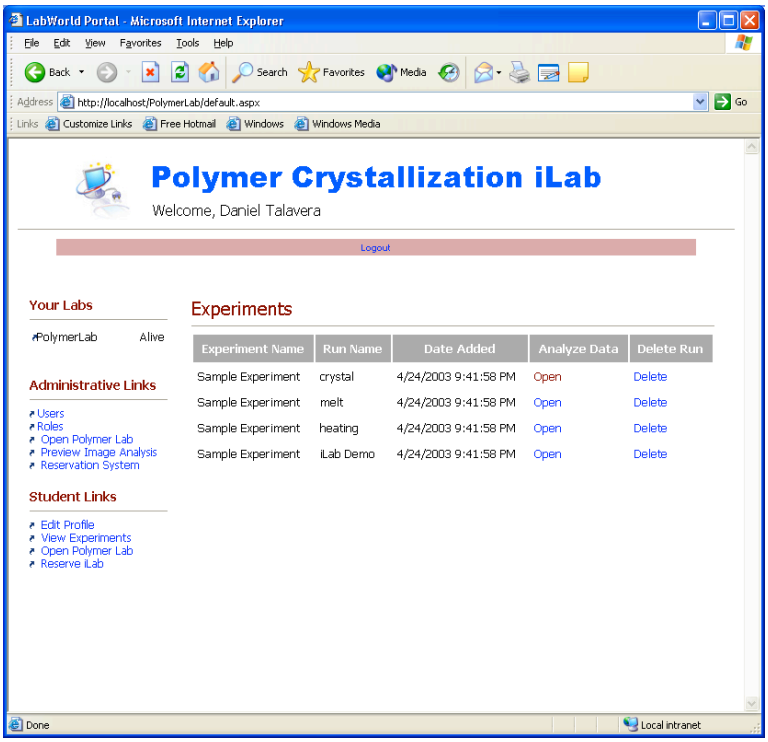


Figure 4: Typical student user interface

The final tier of the Framework server is the underlying database. Implemented using Microsoft SQL, the database allows for the storing of user account information, reservation details, experiment run information, etc, and allows that information to persist between user sessions and throughout the lifetime of the polymer crystallization experiment. It also serves as a medium through which the Framework server communicates with the Microscope server. The Microscope server has the ability to verify with the Framework server information passed to it by the Microscope client, and likewise to communicate to the Framework server the location of data stored during an experiment run by performing database queries. Appendix A contains the scripting file needed to generate the latest version of the database, and has proven useful in the process of restoring the system after reinstallation. This script contains various structures and stored procedures which are essential for operation of the latest version of the polymer crystallization iLab, and should supersede the script contained in Talavera's original thesis.

The diagram in Figure 5 below outlines the structures and dependencies of the database as it is currently implemented, including changes made to allow Multicasting which will be discussed later.

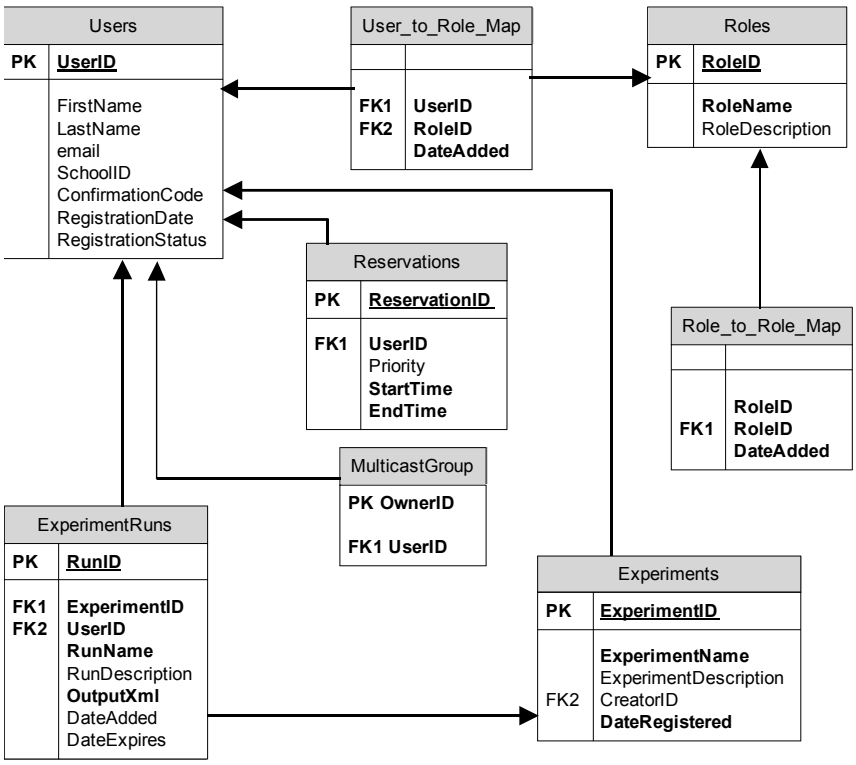


Figure 5: Database schematic representation

2.3 Microscope Client Overview

The Microscope Client is the interface through which students interact with the experiment. It is designed to allow students to quickly and easily manipulate the hardware while still restricting inputs to commands that are valid, so as not to damage any of the hardware. As previously discussed, the Microscope Client is implemented as an applet to allow for maximum compatibility with user operating systems. The applet is handed off to users after authentication by the Framework Server. As shown in Figure 3, unidirectional communication exists between the Framework Server and the Microscope client. This communication occurs in the form of applet tag parameters which are generated dynamically by the Framework Server. (This information is drawn from a corresponding User object which is saved in the Framework Server's session context

when a user first logs in.) These tags include information regarding a user's identity and reservation. The reservation information is used to prevent users from occupying the lab indefinitely by giving the applet an HTML redirect to logout after the specified amount of time. Additionally, this information is used in subsequent communication between the Microscope Client and Microscope Server to determine who is using the Microscope Server and, as will be seen later, whether that user is the "primary" user.

Communication between the Microscope Server and Client occurs via TCP/IP sockets. The system uses a simple, human-readable ASCII message passing protocol based on work done by A.M. Kuchling, as previously mentioned. This system offers the benefit that it is easily extensible, as long as both the client and server recognize the messages being passed. In actuality, there are two socket connections between the Microscope Client and Server. One is uni-directional, and is dedicated to transmitting streamed images from the Server to the Client. The other is an actual duplexed connection for passing messages between the entities. It is important to note that messages are passed both ways, and that the communication protocol is asynchronous: neither side waits for a response from a command. For instance, a communication error which causes a command to the server to grab an image to be dropped does not cause the client to wait forever; instead, it simply displays a new image whenever it receives on. Each message may also include any number of parameters for use by the receiver. For full discussion of the details of the messaging syntax, readers are referred to Daniel Talavera's thesis.

Users cause the Microscope Client to communicate with the Microscope Server by manipulating the client GUI. As discussed in Talavera's thesis, the ScopeFormApplet class which implements the GUI associates messaging events with each of the Java Swing controls which are able to be manipulated by the User. The Microscope Client includes several panels which separate out the functionality of the experiment: a temperature panel, which is used to control the temperature of a sample by submission of an "experiment"; an image panel, which allows users to toggle between single image capture mode and streaming video mode, translate the sample stage in the "xy" plane, and focus the image; a microscope control panel which allows users to control various settings such as magnification, aperture, field stop, exposure time, and autofocus, in an

attempt to obtain the best image possible; and finally, a message area, where status and error messages regarding the state of the experiment are displayed. The implementation of the ScopeFormApplet allows for ease of editing and extending the functionality of the GUI, which will be discussed later with respect to additions to it. Figure 6 shows a screen capture of the Microscope Client before the modifications implemented in this thesis.

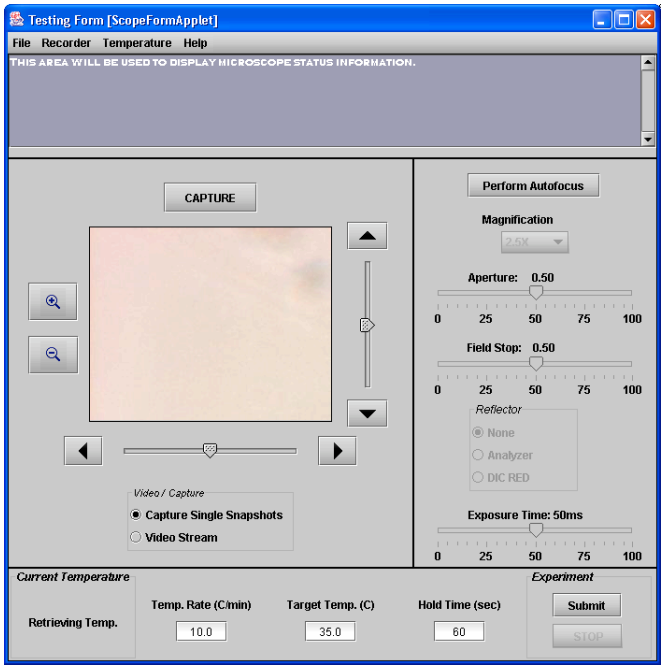


Figure 6: Remote Microscope Client GUI

2.4 Microscope Server Overview

The final component of the Polymerlab architecture is the Microscope Server. The Microscope Server must not only communicate with both the Framework Server and Microscope Client, but must also marshall the commands sent to it and dispatch them to control the actual hardware implementing the experiment. The Microscope Server is a Python object responsible for establishing and maintaining the socket connections to clients. It contains listener and writer threads which continuously interpret and relay commands between lower level hardware controllers and the Microscope Client. In addition, the Microscope Server is responsible for communicating with the Framework Server's database to identify a user's data directory and storing data from experiments there.

The Microscope Server functions mainly by interpreting commands from the Microscope Client and then delegating them to an internal object known as the Device Manager. Briefly, the Device Manager takes commands and determines to which of the available hardware controllers to relay a command. The design of the Device Manager makes the addition of new hardware components as easy as following a template for constructing it hardware controller. Figure 7 depicts graphically the communication between the Microscope Server, the Microscope Client, and the hardware components of the lab. For a more detailed description of the Device Manager, please see Paula Nasser's thesis.

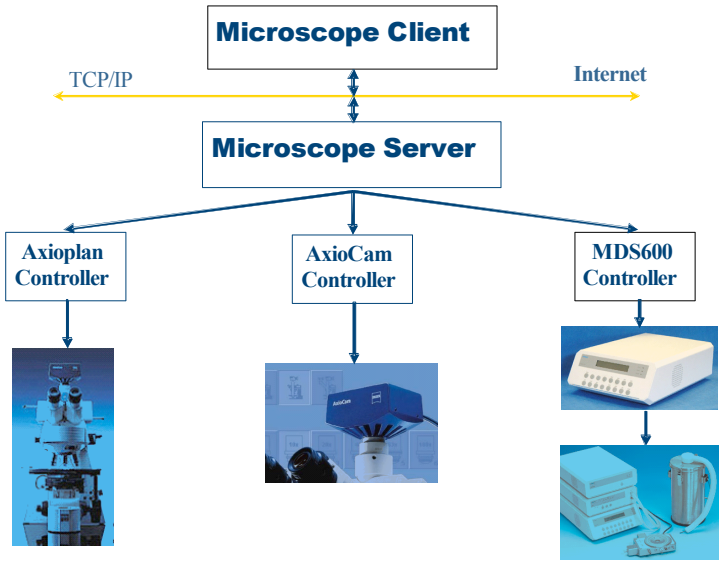


Figure 7: The Remote Microscope

The Polymerlab system currently uses three such hardware controllers mentioned above. They are the Axiocam class, the Axioplan2 class, and the MDS600 class. The controllers each correspond to one of the hardware systems used: the Axiocam controller class controls the Zeiss Axiocam digital camera, the Axioplan2 controller class controls the Zeiss Axioplan2 microscope, and the MDS600 controller class sends commands to the Linkam TMS94 temperature programmer and the Linkam MDS600 motorized heating stage. Each of these devices is pictured in Figure 7. In addition, they are briefly described below. Figure 8 shows a more detailed object dependency between the controllers and hardware.

2.4.1 Axioplan2 Controller

The Axioplan2 Microscope is a Zeiss device. It has multiple light sources and magnification settings, as well as the ability to insert various lenses. It is controlled directly by Zeiss KS300 software. This software may be controlled by hand, but in the case of the Polymerlab system commands are relayed to the KS300 software through Microsoft's COM interface (via a standard Python32 COM module) from the Axioplan2 controller. (This can be seen in Figure 8.) In this manner, the Axioplan2 controller is able to control the microscope's settings after having interpreted them through the Configuration subclass. The structure of the Controller class is discussed in detail in Paola Nasser's thesis.

2.4.2 Axiocam Controller

Like the Axioplan2 controller, the Axiocam controller relays commands through a COM interface to the KS300 software. Unlike the Axioplan2 controller, it makes use of saved configuration scripts which may be opened though the KS300 software by using the *tvload* function. This allows for the specification of image quality, size, and other related properties.

2.4.3 MDS600 Controller

The MDS600 controller controls the TMS94 Temperature Programmer and MDS600 Heating Stage. Therefore, both temperature and XY movement are controlled via the MDS600 controller. Unlike the previous two controllers, the MDS600 controller communicates to the TMS94 through low level serial port commands. These commands are simple text commands and are listed in the Linkam programming guide, "Serial Communication Manual for the T92, T93, and T94 Series Programmers."

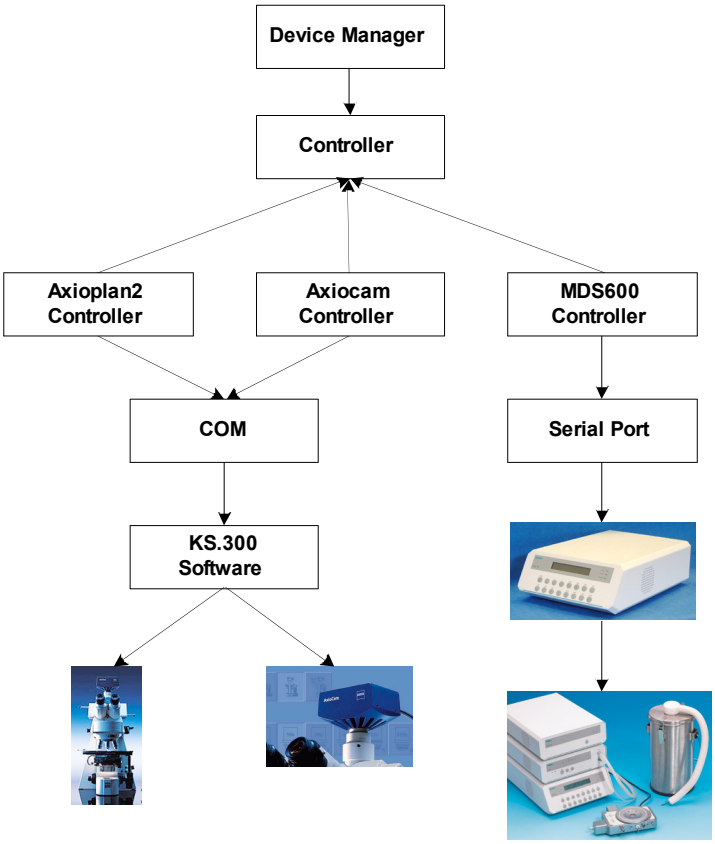


Figure 8: Module Dependency Diagram of Hardware Controllers

CHAPTER 3

Beta Testing and Deployment of Polymerlab

The ultimate goal for the polymer crystallization iLab is to be integrated in a sustainable manner into a course on polymer science, to be used as an aid to teach polymer science. During the Fall Term of 2003, the polymer crystallization experiment was beta tested in such a setting. MIT students taking the course 10.467, Polymer Science Laboratory, used the experiment for the first time in a laboratory setting with a typical user load. The following section describes the details and results of that testing.

3.1 Overview of Experiment

The polymer crystallization experiment represented one of several labs which were performed by students in 10.467, and was the only remotely operated, internet accessible lab in the course. The class consisted of twelve students who worked in groups of 2 or 3. These students were assisted by two lab teaching assistant, as well as one computer assistant who monitored the experiment and responded to bugs/errors and other student questions. Curricular materials describing the experiment setup and procedure, including experiment user manuals, were drafted and distributed to the class. A one hour lecture was given which provided adequate background with the physical principles underlying the experiment. In addition, a one hour demonstration was given to explain all the necessary features required to perform the online experiment.

Each student group performed several runs through the experiment over the course of two weeks. Groups were free to determine how to allot the work among themselves. Each run consisted of making a reservation for the experiment, performing the experiment by controlling the hardware remotely, taking data by recording streamed images, and subsequently analyzing those images. Each experiment run involved heating a Polyethylene oxide, or PEO, sample above the melting temperature of the polymer, and then holding the temperature there for a period of time long enough to thoroughly melt the crystalline sample and erase any memory effects. The students then cooled the samples using ambient air at a controlled rate to one of several predetermined temperatures, several degrees below the crystallization temperature of the polymer. With a polarizing filter in place, the students then maintained this temperature while observing the polymer sample using streamed images. Once the students observed the appearance and growth of birefringent crystallites, they recorded and saved images for as long as they deemed necessary. The students then re-heated the sample and noted that temperature at which this new isothermally crystallized polymer re-melted. They repeated this procedure for each of the several given crystallization temperatures. The students then analyzed the recorded images from each of their temperature runs to determine the nucleation density and crystal growth rates. By analyzing the temperature dependence of the melt transition temperatures and growth rates, the students extracted information about the thermodynamic equilibrium transition temperature, the dimensionality of the crystals, and the activation energy for crystal growth are determined. Each group was required to submit a report detailing the results of their experiments and analysis, which included a determination of the equilibrium melting temperature, the mean surface energy of the crystal lamella, and the Avrami exponent.

3.2 Results of Use

As a tool to evaluate the usefulness of the polymer crystallization iLab, each student was asked to complete written questionnaires asking them to assess various aspects of the lab. A copy of this questionnaire, including a summary of all student responses, is given in Appendix D. In the questionnaire, students gave feedback about issues such as experimental availability, ease of use, adequacy of parameters such as frame rate, and overall instructional merit of the lab. Their feedback was largely positive, and was invaluable in determining which portions of the lab required modification. For instance, students found the initial registration process to be straightforward. They generally found

the introductory demonstration of the experiment useful. However, when asked about reserving and using the experiment, many of them requested longer reservation slots in order to more effectively complete the experiments. This feedback is useful for lab administrators when striking a balance between length of reservations for a single student and overall lab availability. The students were satisfied with the availability of time slots, suggesting that the experiment could support larger class sizes.

The time of day in which students used the experiment varied widely, and they overwhelmingly found the 24 hour availability of the experiment useful. This in itself is a strong validation of one of the underlying assumptions which led to the development of internet laboratories. When asked if they would have preferred increased instructor interaction, students were neutral except in cases where computer problems associated with beta version of the system necessitated additional help. With subsequent, more stable versions of the experiment, it is likely that students will require less interaction. With respect to the interface of the experiment, students found the microscope controls easy to understand and use, and agreed that the frame rate of the streamed microscope images was adequate. (To increase the frame rate, image sizes were previously decreased by Daniel Talavera from 1300x1030 pixels to 260x206 pixels, and image encoding was changed to decrease post-capture processing time. This allows for a frame rate of two images per second.) The students reported that the procedure used to analyze their data was clear, though some found the image analysis tool which was provided too complicated. (This tool, JImage, is a free, Java-based image analysis tool released by the NIH.) Finally, the students overwhelmingly reported that the experiment increased their knowledge of polymers and polymer crystallization. Only one student slightly disagreed with the statement “I enjoyed using this experiment,” and the majority strongly agreed with it.

The initial beta testing of the Polymerlab system was not without problems, however. Most important of these was a hardware problem in the computer hosting the Framework and Microscope Servers. Faulty RAM in this computer caused numerous computer crashes and required frequent rebooting of the host machine. Students found this frustrating. In addition, the KS300 software used to control the Axioplan2 microscope was not intended for operation on a Windows XP platform, causing additional occasional problems during startup. Eventually the faulty RAM was identified and removed, an updated version of the KS300 software was installed, and the rest of the system was as well during the end of the fall 2004 term. This included upgrading the operating system

to Windows XP from Windows NT. (The updated KS300 software was meant for a Windows XP system, as opposed to the earlier software, which was not meant for a Windows NT platform.)

As with any newly developed system, numerous bugs were also discovered. The most important of these was an error in the way that XY movement commands sent from the Microscope Client were validated and ensured to be within the MDS600 stage's safe operating range. The error resulted in damaged hardware which caused movement in one direction to be disabled for the remainder of the test. The equipment was later sent for repairs. An additional bug involved naming conventions for saved experiments. Users frequently used unexpected characters which caused the Microscope Server to enter an error state, resulting in crashes. Additionally, while the experiment is theoretically able to be cycled with no real interaction on the part of administrators, it was found that sample quality and preparation were critical in obtaining meaningful results. For example, a sample which is contaminated with dust and debris has far too many nucleation events to be useful. An additional pathological mode of crystallization which was observed involved the formation of crystals only at the edge of a sample. Because such samples make it impossible for the experiment to be performed properly, it is critical to prepare a proper sample. Appendix F outlines the procedure for preparing a proper sample.

One of the major results of the student feedback was the need for a real-time collaboration utility. Limiting visibility of the scope to a single, registered use at any one time is necessary to avoid hardware control conflicts, but is not well-suited to the potential for learning and collaboration among students at different locations, or even computer terminals. During the beta testing period, it was often the case that groups were forced to come to a central computer lab and huddle around a single computer in order to work together. This obviously contradicts one of the main purposes of developing internet laboratories, especially when group use is foreseen as a primary mode of teaching. This result verified the necessity of one of the initial goals of this thesis, and development efforts were devoted to permitting multiple users to join a "group", wherein all members of which can view the experiment simultaneously, and who can take and cede control of the experiment in real time.

CHAPTER 4

Multicasting

This chapter discusses the addition of the Multicast feature to the Polymerlab system. Multicasting allows users to form groups and simultaneously view and use the polymer crystallization experiment in order to further the goal of user collaboration and allow for remote oversight. Changes were made to each of the components of the Polymerlab system: the Framework server, the Microscope Client, and the Microscope server.

4.1 Motivation

Although the addition of multicasting was already a goal at the outset of this project, the results of the beta testing and user survey were useful verifications of the design goal. The project had already identified that by allowing users to collaborate, the Polymerlab experiment would encourage students to learn from one another. Having observed students working in groups to use the original experiment, it was apparent that collaboration was essential if students were to be able to take full advantage of the remote nature of the lab. This makes intuitive sense, because collaboration is a natural part of a conventional lab setting; preserving this aspect in the virtual lab is important. Collaboration would also allow instructors to assist students remotely and guide them through the experiment. In short, enabling multicasting would make Polymerlab an experiment that could be completely operated remotely, from start to finish, in a typical use scenario.

4.2 Design Additions/ changes

Enabling Multicasting required changes to each of the three major components of the Polymerlab architecture: the Framework Server, the Microscope Client, and the Microscope Server. While these modifications were sufficient enough that the Multicast versions of each of the components would be inoperable in use with the older design, it is important to note that the basic architecture of the system remained unchanged. Furthermore, many of the advantages previously discussed with respect to the design of each component came into play during modifications. As a testament to the thought put into the original design, modification of the components was relatively straightforward.

In general, the goal was to introduce the desired capability with a minimum of redesign. One important feature that needed to be maintained included asynchronous communication between clients and the Microscope Server. Additionally, it was decided that communication should occur only between the server and clients, and not between clients. The Microscope Server would mediate all communication between clients. Furthermore, the maintenance of the video streaming frame rate was of prime concern. Students had found the rate adequate, and introducing more clients could potentially slow the distribution of the streamed images.

In addition to the maintenance of some features of the original design, choices about the operational details of Multicasting had to be made. First, it was important that Users have the ability to define their own groups. When interacting with the experiment, it was clear that there would need to be two types of users: a controlling user, and observing users. This is critical not only to prevent race conditions in situations that could potentially damage hardware, but also to manage the complexity of having many users participate in an interactive experiment. The data from an experiment would be stored within the account of the user controlling the experiment. As a consequence, in order to prevent ambiguity over data storage, it should not be possible to pass control of the system while an actual experiment (in the sense of temperature controlling) is in progress. In order to reduce the number of interactions between the three components of the system during initialization of the client applet and communications with the Microscope Server, first control of the experiment would be assigned to the first user to open the applet. While others in a user's Multicast group may open and view the experiment and its status, only

the first user to arrive is first given control. This prevents wasted reservation sessions. In addition, no "daisy-chaining" should be allowed: only direct members of a user's Multicast group may access their experiment. Members of members may not. Considerations such as these influenced how the Multicast feature was implemented.

4.2.1 Client Changes

The most visible changes to the system took place in the Microscope Client. Users would need a mechanism to give and take control of the experiment, as well as identify times when control was available, etc. The ScopeFormApplet class underlying the client applet was modified to introduce a new button panel alongside the message pane. It includes two buttons: one to cede control of the experiment and the other to take it. While the Microscope Server itself tracks which user has control of the experiment, as will be discussed later, the applet also includes a layer of protection against users who do not have control manipulating the experiment. All of the input mechanisms in the applet were modified to be active contingently based on a number of Boolean fields: the **haveControl** field indicates whether a client currently has the ability to send commands, while the **controlAvailable** and **experimentRunning** fields determine at any time whether a user may take control. These fields are initially set by the Framework Server when the applet is handed off to a user, and update immediately through communication with the Microscope Server. The Microscope Server immediately communicates the state of control. When the **haveControl** field is set to false, a user may see the experiment controls and any images streamed or captured by the controlling user, but may not manipulate the controls. The controls at such a time appear grayed out and are inaccessible. Figure 9 shows the modified Microscope Client applet.

While the observing user cannot modify the controls, the controls still reflect any manipulations performed by the controlling user. Whenever an action is performed, an updating message with all the necessary information is sent to each of the clients. This is important if users are to stay oriented during the course of the experiment. For example, if users saw the quality of images begin to deteriorate during an experiment run they might not know whether the sample was simply melting or whether the controlling user had modified the settings, such as exposure time. Due to the asynchronous nature of communications between the Microscope Server and Client, there already existed an

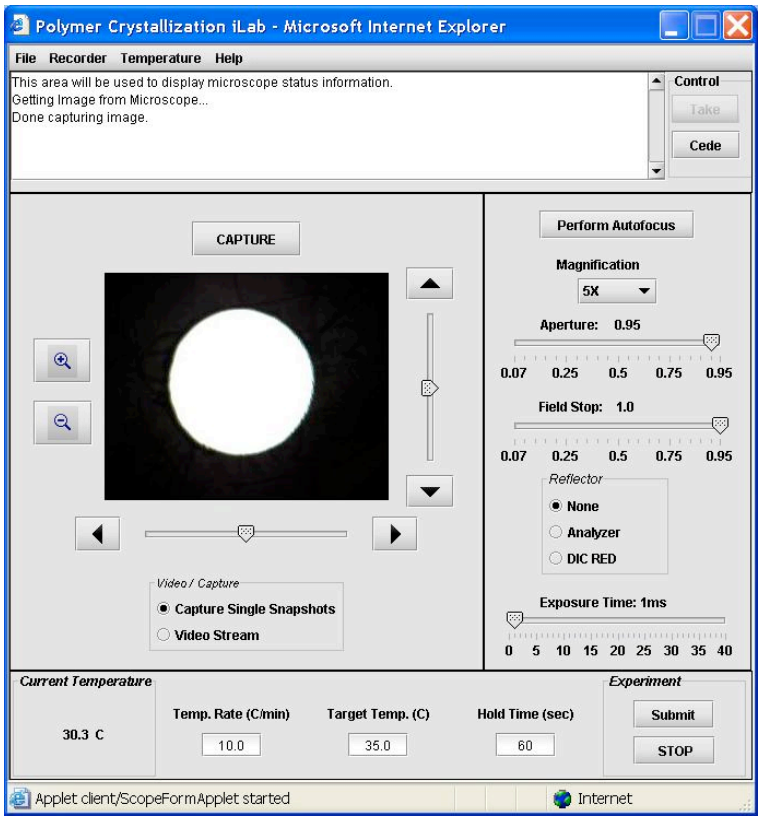


Figure 9: Modified Microscope Client applet.

infrastructure for distributing information to client about the state of the experiment. This occurs through the **cmd_scope()** function of the IScopeProtocol interface which is implemented by the ScopeFormApplet. (The IScopeProtocol interface is simply a list of commands which the client must be able to recognize from the server.) Now, after every command, the Microscope Server then simply sends this command to every client connected to it, regardless of control status. Of course, the IScopeProtocol interface did have to be extended to enable the ceding and taking of control by users. The handling of two methods, the **cmd_occupied()** and **cmd_available()** methods, were rewritten to configure the applet based on availability. Three other methods were added. A listing of these commands is shown in Table 1. Readers are referred to Table 2 of Daniel Talavera's thesis for a full listing of the commands.

COMMAND	EXPLANATION
TAKE_SUCCESSFUL	Informs a client that control of the experiment has successfully been taken. All other clients receive the TAKEN command.
TAKEN	*UserID* Informs client that another user, identified by the accompanying string, has taken control of the experiment.

CEDE	Informs client that control of the experiment has been given up, and is now available to be taken.

Table 1: Commands from the Server to the Client.

4.2.2 Microscope Server additions

Previously, the Microscope server only kept track of a single user. The server trusted connections from any applet, since the verification of users was left to the Framework server. If an additional user tried to make a connection to the server, it was sent an "occupied" message automatically. The concept of serving a group of users, rather than just one, with one user having "control" had to be introduced. Since additional users could be expected now, this behavior was changed to include an access of the Framework Server's database to determine whether the user had a reservation or is a member of the proper Multicast group. This provides an additional layer of protection against unauthorized users accessing the lab, rather than relying on the assumption that any applet contacting the Microscope server was properly authenticated by the Framework Server. A user queue was implemented, in which every current user resides. The AsynchronousServer class of the Python was actually given three such lists of socket connections: clients, unauth_clients, and image_socket_clients. Now images being streamed are sent to every client socket connection in the image_socket_clients list. Likewise, commands for recognizing new users or removing current users involve adding or removing their socket connections from the lists. An additional user field, currentUser, was introduced to track the user who currently has control of the experiment. In order to execute any command, a user must be the currentUser. The only exception to this occurs when the currentUser gives up control of the experiment. At that time, the **take** command is recognized by the first client to respond. Commands issued by the currentUser are executed and updating messages are sent to all authenticated clients. The take and cede functions were two new commands which were introduced to the Microscope Server which it had to be able to understand from the clients. These functions are described below in Table 2.

COMMAND	PARAMETERS AND EXPLANATION
TAKE	userID=*value* Request to take control of the experiment. If control is available, causes a user to be set to currentUser.
CEDE	If a user is the currentUser, gives up control of the experiment and

	notifies all other clients of the availability of control.

Table 2: Commands from Client to Server.

4.2.3 Framework server additions

In order for the authentication process to acknowledge Multicast groups, and in order for them to be managed dynamically by users, the Framework Server had to be modified. These modifications occurred in three main areas: the creation of new object classes which would correspond to this new structure and would allow the code backing the aspx pages to store relevant information in a user's session context; the creation of new aspx pages for group management; and the modification of the database to introduce new data structures and stored procedures.

The new C# object class MulticastGroup was modeled after the User class. In the interest of maintaining a compact structure, however, it has no dependencies on the User class. Instead, it tracks its members via one GUID corresponding to the groups owner, and a separate list of GUIDs corresponding to the group's members. (A GUID, or global unique identifier, is a number which, within a system, is guaranteed to be unique. This allows it to be used as an efficient representation of an object.) Like the User class, it contains a variety of helper methods which connect to newly written stored procedures in order to produce MulticastGroup objects from data in the Framework Server database. These methods are also used to identify members and are used in the authentication process. The new MulticastGroups database table merely identifies owner/member GUID pairs, which are used to construct MulticastGroup objects when the proper helper methods are called. The newest database generation script can be found in Appendix A. In order for users to manage their own groups, new .aspx pages were created. For instance, Figure 10 below shows the group management page, where users may be deleted. By simply following a link to the page shown in Figure 11, the user can add any other valid user. Finally, Reservation authentication was changed in

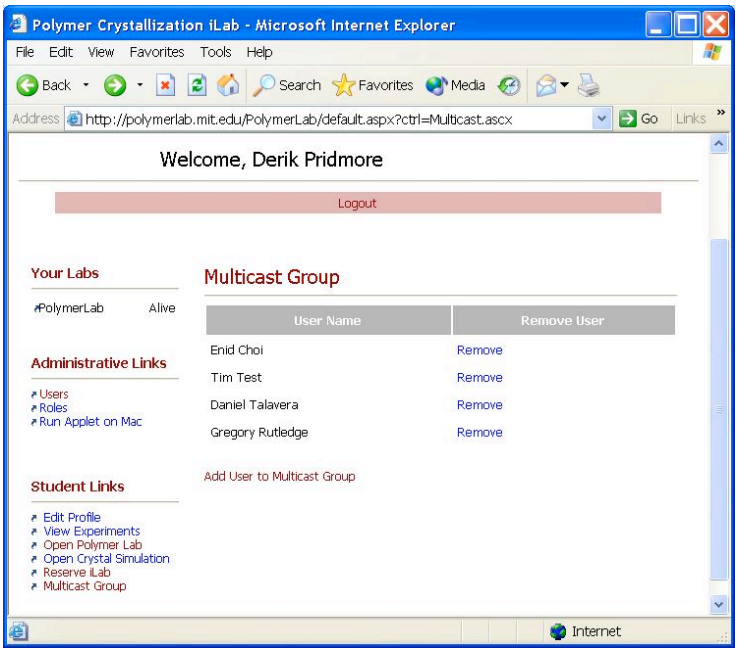


Figure 10: Framework Server Multicast Group management page

the Framework Server to allow the handoff of applet and applet parameters to members of a Reservation owner's group. This involved modifying the code behind the applet access page to use certain MulticastGroup object helper functions.

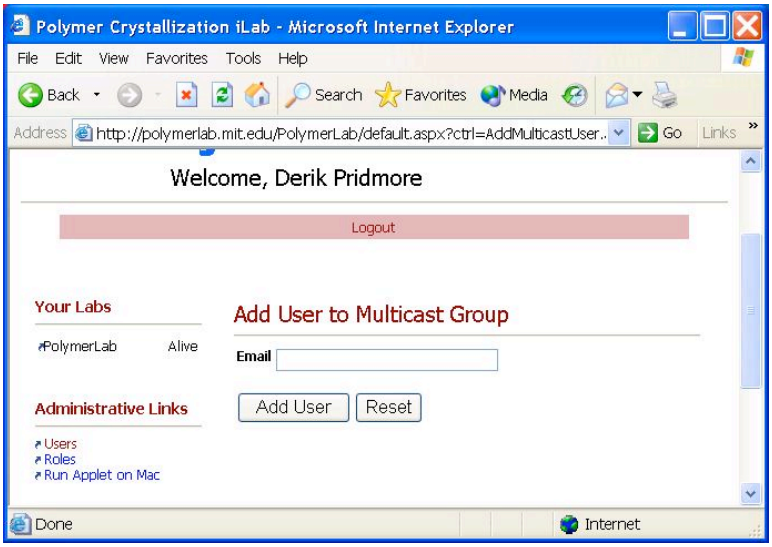


Figure 11: Adding User to a Multicast Group.

4.3 Performance issues

Given that much work had previously been done on the Polymerlab system to ensure that

frame rates would be adequate for students, it was important to verify that the introduction of potentially many clients using the system at once would not cause the rate at which images were streamed to suffer greatly. Figures 12 and 13 below illustrate that this is not the case. Figure 12 shows the time taken to output images (averaged over approximately 5 images) to all clients for a varying number of such clients. The data

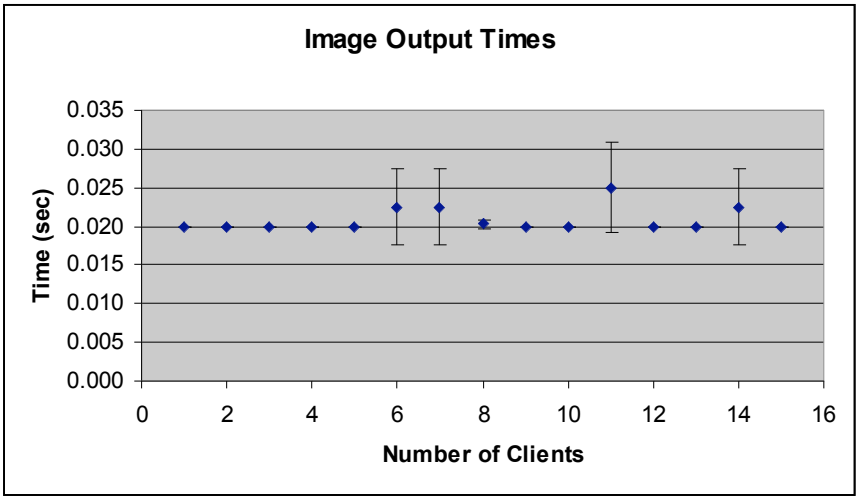


Figure 12: Microscope Server image output times versus number of clients.

indicate that, for reasonable numbers of clients, factors other than the actual time required to send output images to all of the image sockets of clients dominates. In addition, it is clear from Figure 13 that the time required to actually grab an image from the Axiocam camera and perform any processing necessary absolutely dominates the approximately 6 frames/second streaming rate. Thus, the addition of multiple clients accessing the microscope and receiving images at once is of no great concern in terms of frame rates.

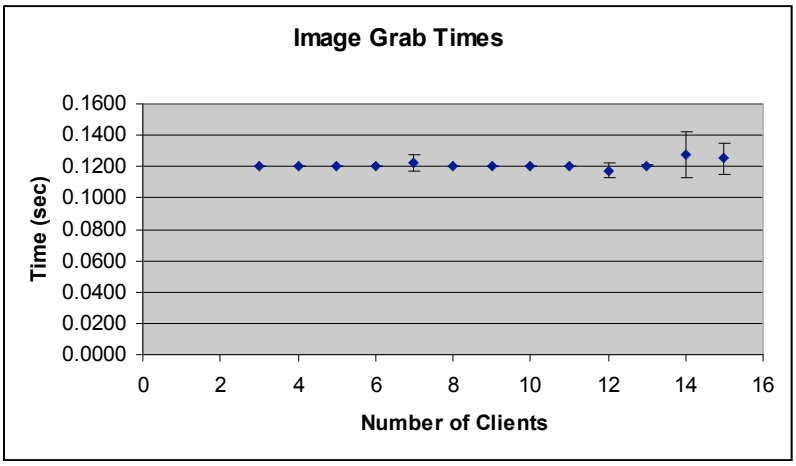


Figure 13: Microscope Server image acquisition times versus number of clients.

CHAPTER 5

Graphical Simulation Application

This chapter outlines the elements and algorithms behind the crystal simulation tool. The motivation behind the application is simple: to provide an easy-to-use tool which students can use to compare theoretical crystallizations with actual results from the online microscopy experiment. For instance, variances in sample quality and/or preparation can cause memory effects that in turn cause nucleation events to occur non-randomly. Similarly, temperature gradients may cause variations in crystallization rate or melting temperature. In addition, some of the assumptions used in the crystallization model may be oversimplifications, which could cause predictions to be slightly different from observed phenomena. A tool that would allow students to quickly and easily explore the similarities and differences between observation and prediction would make a valuable contribution to their understanding. More simply, such a tool could be used as a sanity check: students could "rerun" the experiment virtually using the same values as the original experiment and determine whether visually whether it generates similar results.

5.1 Crystal Growth Model

The primary goal of the polymer crystallization experiment is to use a model of the temperature dependence of the rate of nucleation and rate of growth of crystal spherulites to identify physical properties of the PEO sample and compare these to known values. This is a test of both the experimental technique of the students and the validity of the model used to understand the crystal growth.

The standard goals of crystallization models are to explain the observed morphological features and their kinetic features and, in a related manner, provide analytical methods by which measured properties can be reliably predicted. Typical models take into account variables such as temperature, stress, strain, and composition, but in the present polymer crystallization experiment students are only concerned with temperature dependence. The two main features which the polymer crystallization experiment investigates are primary and secondary nucleation rates and their dependence on temperature. (Secondary nucleation refers to nucleation of a new layer of crystalline material on an existing crystallite.) A model of crystallization kinetics should try to explain or at least be consistent with several features of crystal growth: the morphology of polymer crystals (lamellar crystal growth, the magnitude of lamellar thickness, and the fact that at crystals formed at intermediate temperatures have straight facets whereas those formed at lower temperatures have curved facets), the temperature dependence of the growth velocity, and the effect of molecular weight on growth kinetics. (Molecular weight effects are not addressed in this experiment.) The model which is discussed below is a summary of an excellent and clear discussion found in Schultz of the most basic models explaining crystallization [1].

In general, there is consensus regarding the modeling of the primary nucleation of crystallites. Primary nucleation theory derives from classical nucleation theory used to describe the formation of water droplets. Generally, that theory considers water molecules in vapor form in a temperature and pressure below the boiling point of water. Thus, it is thermodynamically favorable for the water to exist in liquid form. However, a small droplet has a high surface area to volume ratio because of its size, and may actually be energetically unfavorable. By comparing the surface energy associated with a droplet of size r and the corresponding energy difference between that drop in liquid and vapor form, a critical drop size r* can be determined. Any drop formed at this size will continue to grow. Thus, r* defines a critical nucleus. By determining the distribution of embryonic nuclei in a sample (which is a Boltzman distribution) and the rate at which sub-critical droplets are accumulated the primary rate of crystallization can be determined.

Analogously, primary nucleation theory of polymer crystals considers an embryonic crystal consisting of strands of polymer molecules stretched out and aligned side by side. For such an embryo having v polymer strands of length l and cross-section a, the total free energy difference between the embryonic crystalline state and the liquid melt state

can be written as:

$$\Delta G(v,l) = lav\Delta G_v + 2av\sigma_e + Cl\sqrt{av}\sigma_s ,$$

where Gv is the difference in free energy per unit volume between the crystallized and melt states, C is a constant relating to the cross-section shape of the strands, and σe and σs are the end and side surface energies respectively. By differentiating with respect to the number of strands v and the length l, one can obtain the critical values for the length l and number of strands v and eliminate them from the equation. Using for ΔG_v the formula (which is valid for undercooling)

$$\Delta G_v = -\Delta h_f \frac{T_m^o - T_x}{T_m^o} ,$$

where T_x is the crystallization temperature, yields a critical value for the energy barrier of

$$\Delta G^* = \frac{2C^2\sigma_s^2\sigma_e(T_m^o)^2}{\Delta h_f{}^2(T_m^o - T_x)^2} .$$

By using arguments about the energy differential involved in adding one strand to a critical embryo or subtracting a strand to reduce a v+1 embryo to a critical embryo based on the free energy difference formula given above, a net growth rate of embryos can be determined. (Demonstration of this calculation requires a substantial number of approximations and is avoided here.)

The most commonly used model to explain secondary nucleation was proposed by Lauritzen and Hoffman. It basically rests on the idea of a molecularly flat growth interface on which a new crystal layer must nucleate and grow. In the model, growth occurs as adjacent sequential strands are deposited in an entire layer, one layer after another. The first strand bonds to the original layer, and then subsequent strands, which are part of the chain containing the first strand, bond to both the original layer and also the previous adjacent strands. The chain winds along the surface in the manner shown below in Figure 14.

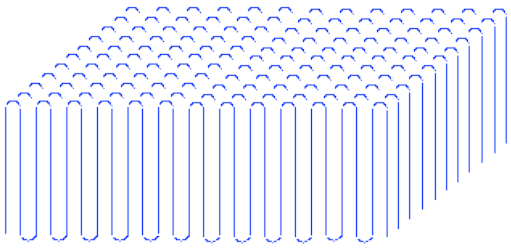


Figure 14: Example showing winding of strands at the crystal surface.

(The challenges to this idea have to do with the molecular surface being rough, with the

nature of the nucleus that starts the new layer and with whether the barrier to nucleation is predominantly enthalpic or entropic in origin. Hoffman has also proposed a "Regime" model that accounts for roughness of the growth surface as a consequence of competing rates of nucleation for a new layer and lateral growth to complete an existing layer. This is different from roughness of the fold surface, which requires entropic arguments.)

Starting in a manner very similar to the primary nucleation analysis, the secondary nucleation analysis begins with an expression for the free energy change caused by the addition of the first strand. By subsequently using this expression determining the net rate of deposition of first strands on potential sites, a functional form for the growth rate is obtained [1]. A typical model of the growth velocity of crystal lamella obtains the formula

$$v(l',T) = G_o e^{\frac{U^*}{R(T-T_\infty)}} e^{\frac{-K_g}{T\Delta T}}.$$

G_o is a constant that contains some temperature dependence itself and also wraps up the dependence on strand length, etc. The growth constant K_g is given by

$$K_g = \frac{4b\sigma_s\sigma_e T_m^o}{k\Delta h_f}.$$

It should be noted that the leading constant in the formula depends on the assumptions made about the exact manner of formation of new monolayers on the crystal surface. The value of 4 above corresponds to the case in which one monolayer completes before another begins. Furthermore, the scaling of the temperature in one exponential of the growth velocity formula from an absolute level to one dependent on a value T_∞ reflects the fact that strand mobility ceases not at absolute zero but around the glass transition temperature T_g.

T_∞ is identified empirically to be around T_g + 30°K. In the polymer crystallization experiment this corresponds to a value of 176 K. U* has a value of 29.3 kJ/mole. Δh_f, the heat of fusion of PEO, has a value of 2.43x10^9 ergs/cm3. The parameter b is the diameter of the polymer molecule, which for PEO is approximately 0.5-1.0 nm.

A good model of crystallization must simultaneously take into account both the primary and secondary nucleation processes discussed above, since even while one crystal has already formed and may be quite large another primary nucleation event may just be occurring. Such a "continuous nucleation" (nucleation at a constant nonzero rate)

contrasts with "instantaneous nucleation," in which all nucleation takes place at t=0, and the subsequent nucleation rate is zero.) The first is appropriate for homogeneous nucleation, while the second describes heterogeneous nucleation. An analysis that incorporates both primary and secondary nucleation leads to a prediction of total transformed crystalline fraction of a sample as a function of model parameters such as temperature. The Avrami equation yields just such a relationship.

The Avrami analysis begins by observing that for a polymer sample in which spherulites nucleate randomly at a constant rate of n per second, and grow at a steady velocity v, a spherulite that begins at time τ will have a volume

$$\Omega_1 = \frac{4\pi}{3}(v(t-\tau))^3 .$$

Note that this assumes the crystal grows as a sphere, but this does not affect the overall analysis; the generalized result is given below. If one did not have to worry about crystals overlapping, the volume transformed in a small period of time *dt* would be

$$d\Omega'(t) = \frac{4\pi}{3} n(vt)^3 dt$$

The difficulty arises in determining the total volume Ω transformed, since once crystal cannot nucleate and begin to grow within an area already occupied by another. Denoting the fraction of the sample which has been transformed as Φ(t), the amount of sample which is transformed at any time t is simply the amount that would have been transformed with overlap possible scaled by the fraction of the sample which remains to be transformed, given by the equation

$$d\Omega(t) = [1-\Phi(t)]d\Omega'(t) .$$

Dividing through by the total volume of the sample and integrating, an expression for Φ(t) is obtained which describes the total fractional transformation. In the general case of this dependency is known as the Avrami equation and is given by:

$$\Phi(t) = 1 - e^{-kt^m} .$$

The constant m represents the Avrami exponent. The Avrami exponent is equal to the growth dimensionality plus one. (The factor k also depends on the growth dimensionality and is a function of the growth rate v raised to a power that goes as the dimensionality.) The equation given above also depends on the time distribution of nucleation events. In particular, it assumes homogeneous nucleation, meaning that nucleation events occur at a constant rate throughout time. The opposite of homogenous nucleation is known as

heterogeneous nucleation, and describes the situation in which all of the nuclei appear and begin growing at one moment and only that moment. The dependence of facet or edge shape mentioned earlier depends crucially on the distribution of nucleation events and was a key consideration in the simulation applet which will be discussed below.

Knowing the functional form of the volume transformation and temperature dependence of the rate of crystal growth, students using the Polymerlab are equipped to compare the results of their experiments to those predicted by their model of crystallization kinetics. The goal in implementing the graphical simulation applet was to adequately represent the predicted behaviors.

5.2 Voronoi Diagrams

Given the positions and growth rate of spherulites formed at a uniform temperature in a sample, graphical simulation of the growth of these crystals becomes essentially the problem of calculating boundaries between growing spherulites. This problem, especially in the simplified case of heterogeneous nucleation, may be addressed through the use of Voronoi diagrams.

Given a set of points in a two dimensional grid, the set of all points closer to a given point than to any other point in the set forms a structure known as the Voronoi polygon for the point. The union of the all the Voronoi polygons for a point set is known as the Voronoi diagram for that set [10]. Examples of uses for Voronoi diagrams include route planning algorithms for robots, operational logistics for supplying to a specified set of locations, astronomy, etc. Figure 15 shows a set of points along with the associated Voronoi diagram (dark) and the Delaunay triangulation (light).

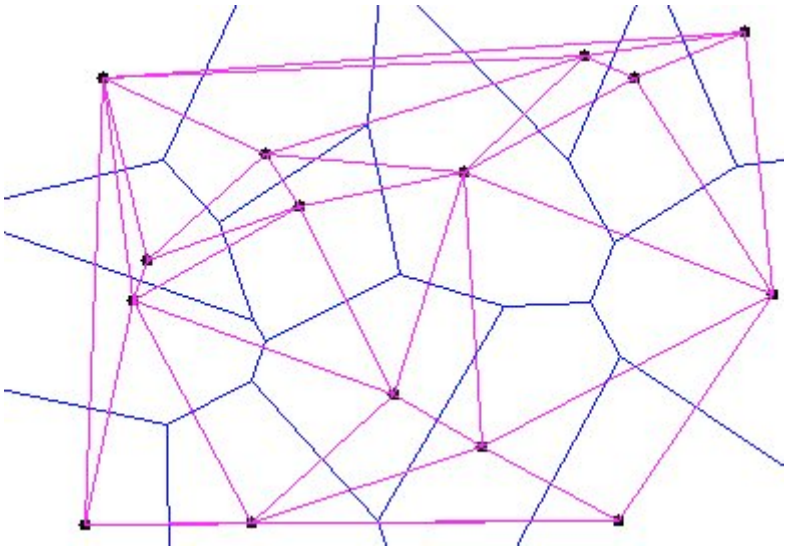


Figure 15: Voronoi diagram and the dual Delaunay triangulation

The dual of the Voronoi diagram problem is the problem of calculating the Delaunay triangulation of a set of points. The Delaunay triangulation is given by the set of lines drawn between each point and the set of points "closest to" it. The problems are dual in that any algorithm or computation which calculates one problem may also be trivially converted to calculate the other. Points in the XY plane are connected by a line in the Delaunay triangulation if their Voronoi polygons share a common edge in the Voronoi diagram. As can be seen in Figure 15, the edges of the Voronoi diagram form perpendicular bisectors of the lines in the Delaunay triangulation.

The Voronoi diagram of a set of points has several properties that lend themselves to the construction of efficient algorithms. The most important of these properties is that the number of lines in both the diagram and its dual is linear in the number of points N, and this linearity is typically of order a small constant times N. Algorithms exist based on the convex hull problem which can solve the Voronoi triangulation problem in O(NlogN) time and O(N) space. This algorithmic efficiency can be shown to be optimal [10].

5.3 Heterogeneous nucleation applet

In order to first approach the most tractable problem, the design of the simulation applet began by addressing the problem of heterogeneous nucleation. Heterogeneous nucleation was chosen because, independent of crystal growth rate, the boundaries between crystals

formed during heterogeneous nucleation always form straight lines. In contrast, homogeneous nucleation in general causes the boundaries to form hyperbolas, although this is not always apparent at low crystal growth rates. The assumption of heterogeneous nucleation simplified the problem of graphical implementation in Java and in terms of the pre-computation of boundaries.

Given the definition of the Voronoi diagram problem, the parallels between the problem of calculating boundaries for crystals formed during heterogeneous nucleation crystallization are obvious. Since each crystal starts growing at the same time and at the same rate, the boundaries between crystal centers will be formed along perpendicular bisectors. In addition, the assumption of a uniform growth rate allows the problem of graphical rendering to be broken into two simple steps. The first is painting the area filled by at least one crystal. By painting all crystals in a similar monochrome manner, it does not matter which crystal is painted first or whether crystals overlap. (Note that this is not true if crystals grow at different rates.) This saves the work of having to calculate the exact time and point of every contact between crystals and modifying their boundaries to be some arbitrary shape. The second step, then, involves painting the pre-calculated boundaries. By painting these boundaries the same color as the background, it is only apparent at an impinging location between two or more crystals. This method yields a convincing appearance of crystal growth and impingement to the eye as long as the boundaries are properly calculated. Appendix B contains the Java code for the initial design of the simulation applet implementing this strategy for heterogeneous nucleation.

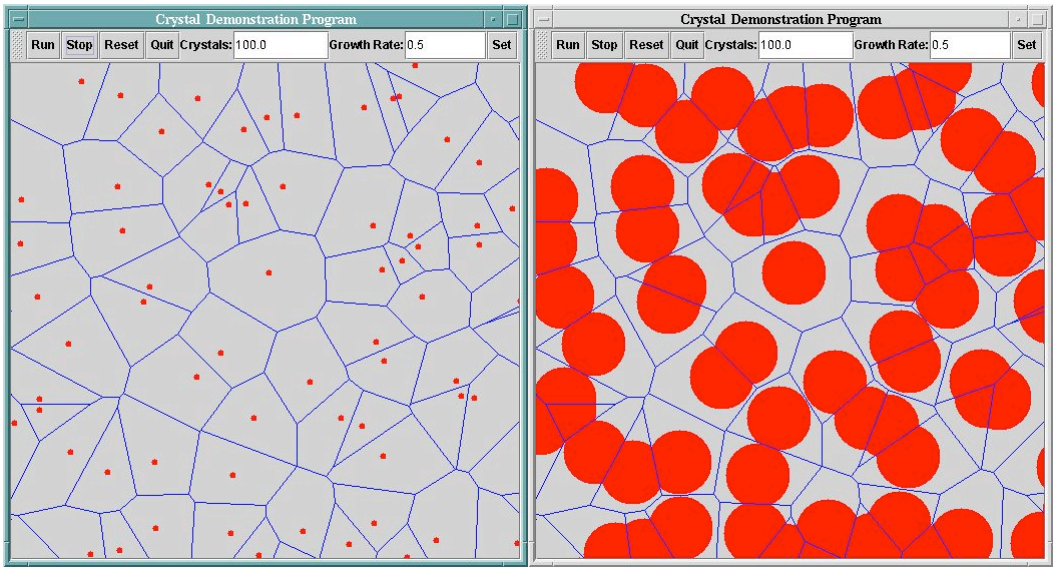


Figure 16: Heterogeneous simulation applet showing crystal growth.

Figure 16 demonstrates the previously discussed approach. In general, the logical flow of the simulation is as follows. Random nucleation sites are calculated based on an input (which was initially simply of number of crystals.) For each site, a crystal with zero radius and a center located at the nucleation site is created. The boundaries between the crystals are calculated using Voronoi diagram algorithms discussed below. The animation is then started, with time iterating in intervals of 20 milliseconds. (Future version may base this rate on growth rate of crystals to ensure they are smoothly rendered.) At every times step, each crystal object is given a message to "grow," which alters its radius field. Each crystal in the window is then asked to paint itself. Finally, the boundaries of the crystals are painted over the crystals and time is iterated. In Figure 16, the boundaries are painted a shade obviously different from the background in order to demonstrate that these pre-existing boundaries are in fact correct.

While the heterogeneous animation applet accomplished the goals set out for it, it still has limited usefulness since the actual crystallization process observed in the polymer crystallization experiment involves homogeneous nucleation. For this reason, more realistic methods of calculating and rendering crystal boundaries were investigated.

5.4 Additively Weighted Voronoi Diagrams

There exist extensions of the Voronoi diagram to problems more general than simple heterogeneous, uniform growth rate crystallization. In the most common statement of the problem, discussed in the previous section, the function used to determine whether a particular location in the XY plane is closer to one point than another is simple Euclidean distance. However, the Voronoi diagram problem can handle more general functions. One such extension of the standard Voronoi diagram is known as the additively weighted Voronoi diagram. Additively weighted Voronoi diagrams are defined in the same way as the Voronoi diagrams previously described, except that the Euclidean distance used to determine the location of boundaries is modified for each point by a "weighting." For instance, in the case of calculating the boundary between crystals, each crystal may be given a weighting equal to the time for which the crystal has been in existence. Crystals which formed early might have a large weighting, while relatively recent crystals might have a small weighting. This weighting is used to modify the Euclidean distance to any point. Thus crystals started earlier are considered to be "closer" to any location than later

crystals. It should be noted that there exist still other forms of the Voronoi diagram which correspond to crystals whose individual growth rates are different in addition to different start times. The solutions for such problems involve complex spirals. [11]

Like the simplest Voronoi diagram problem, optimal algorithms exist for additively weighted diagrams. These diagrams are also of order O(NlogN). However, not only are these algorithms difficult to implement, their implementation using the Java APIs chosen for this project would require writing complex representations of curved lines. Since an initial implementation would have involved numerical approximations, using existing graphics functions, of the hyperbolic lines needed, it was decided first to attempt an outright numerical calculation of the crystal boundaries. Because the boundary calculation would still be done as a pre-computation, optimization of the algorithm would not be critical.

5.5 Homogeneous nucleation applet

In the heterogeneous applet, the algorithm for calculating the boundaries is a brute force, $O(N^3)$ iteration. The basic idea is to iterate over every crystal combination, calculating the hyperbolic curve that would form the boundary between those two crystals if no other existed. This curved boundary is then graphically approximated by short, straight line segments which are added to a set of boundary lines, contingent on the fact that no other crystal is closer to that location in the sense of the additively weighted distance function. This function is calculated using Euclidean distance minus the difference between product of the start time and growth rate of the subject point and any other point. This in effect handicaps younger crystals by subtracting from their present radius the radius of the older crystal. Figure 17 shows two screenshots of the heterogeneous applet. Crystals forming at different times can be seen to grow and impinge on one another.

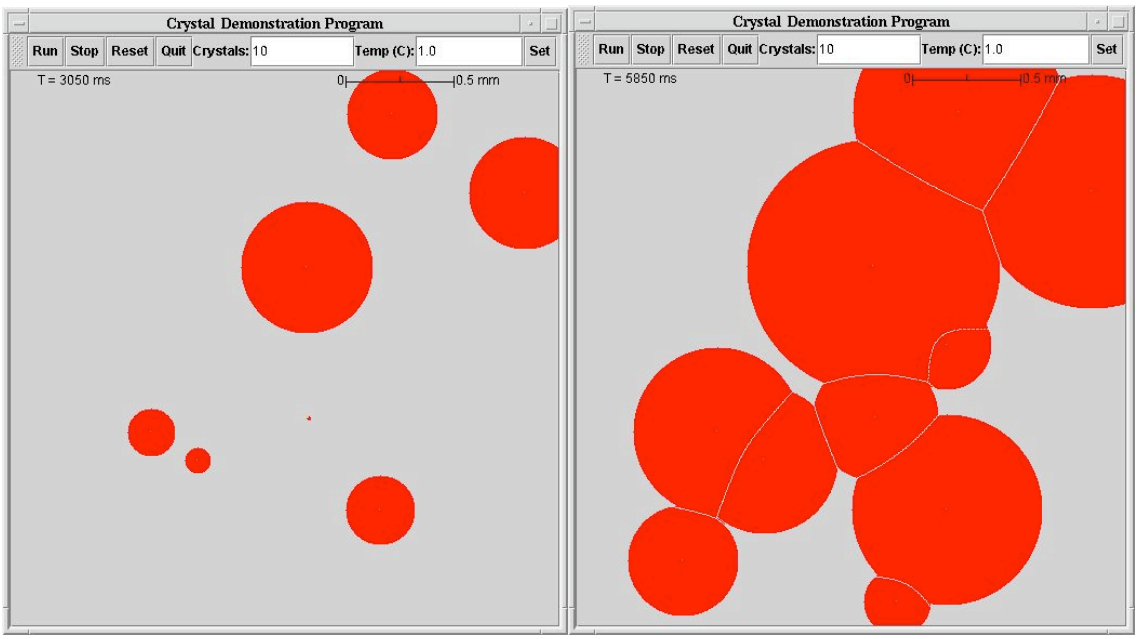


Figure 17: Graphical simulation using Additively weighted Voronoi diagram.

The heterogeneous applet must, of course, also provide a method for adding crystals throughout time. At present, crystals are simply added at a constant, preset rate. After inputting a number of crystals, the algorithm calculates a random location for the first crystal. Given the next crystal "nucleation" time, the algorithm then calculates a new random point. If this point lies inside the previous crystals, it is discarded and another is chosen. If it does not, a crystal with that location and start time is added to a Vector holding all the crystals. This continues until the desired number of crystals is achieved. While this method provides an excellent first pass and yields good results, as will be seen later, there are several problems that may potentially be improved upon. The first is that as a large number of crystals are added, the probability of finding an available site to place an additional crystal becomes extremely small. Because the algorithm described above would simply loop until one is found, a hard coded upper limit was used to prevent such a scenario. The second manner in which this technique could cause a problem is in the analysis of the Avrami exponent, since the analysis given before assumed a constant rate of nucleation. A third area of possible improvement is that the nucleation rate is simply programmed in and is constant. A more realistic approach might take into account the temperature at which the crystals are formed.

5.6 Results

Given the model discussed in section 5.1 and the numerical values of constants for PEO, it should be straightforward to generate realistic results for the crystal animation. However, as discussed before, certain assumptions and simplifications were made during the development of the graphical animation applet. Therefore, as a test the applet was used to measure crystal growth rates at 4 randomly chosen temperatures. (Appendix F, the Student User Manual, contains details on how to run the applet.)

Parameter	Value
$(\sigma\sigma_e)^{1/2}$	3.35E-6 J/cm^2
b	1 nm
T_m	70.3 C
T_∞	-97.15 C
U*	29.3 kJ/mole
Δh_f	2.43x10^9 ergs/cm^3
R	8.3144 J/mol*K

Table 3: Parameters from applet test run.

Table 3 shows the values used in the simulation applet to calculate growth rates. The sizes of crystals were recorded at various times while the graphical simulation ran for each temperature. Each crystal was measured at least three times and the results of those measurements were averaged and plotted against time for each temperature. This analysis follows the same steps as those taken by the MIT students using the Polymerlab. The analysis allows one to obtain a value of the growth constant, K_g. Using that constant and the previously defined relationship, the value of the mean surface energy, $(\sigma_s\sigma_e)^{1/2}$, was calculated. The results yielded a value for $(\sigma\sigma_e)^{1/2}$ of 3.32E-6 J/cm^2, which is in good agreement with the accepted value of 3.35E-6 J/cm^2. Because, in the simulation, the actual number driving the evolution of the crystals is the correct one, this level of accuracy reflects a typical measurement inaccuracy that can be expected by normal users. The intermediate results are shown in Figure 18 and Figure 19.

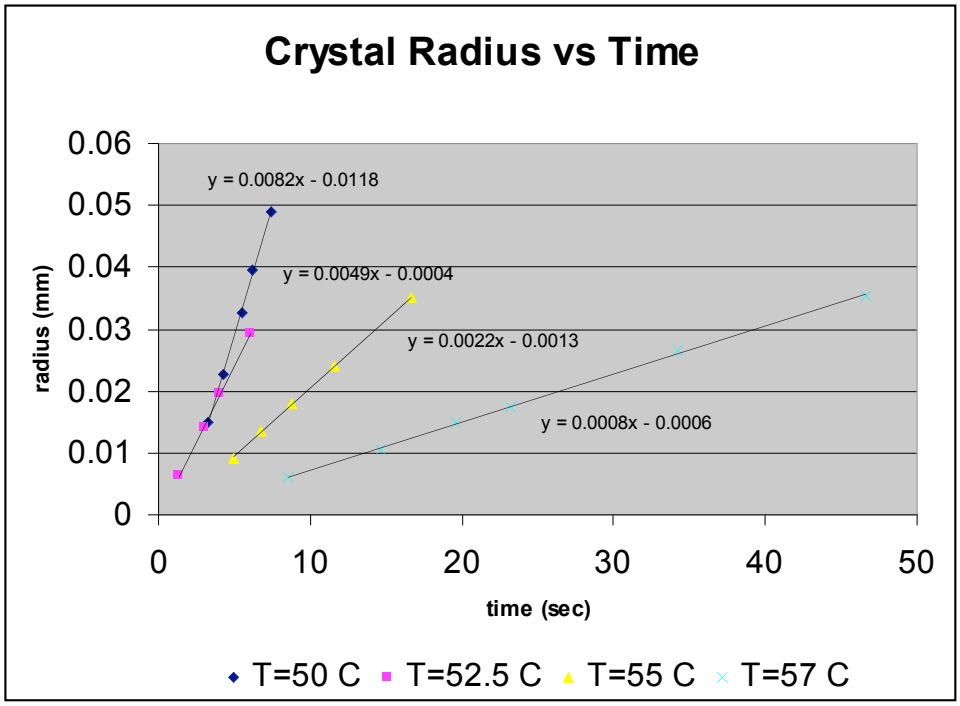


Figure 18: Crystal growth versus time for animations at selected temperatures.

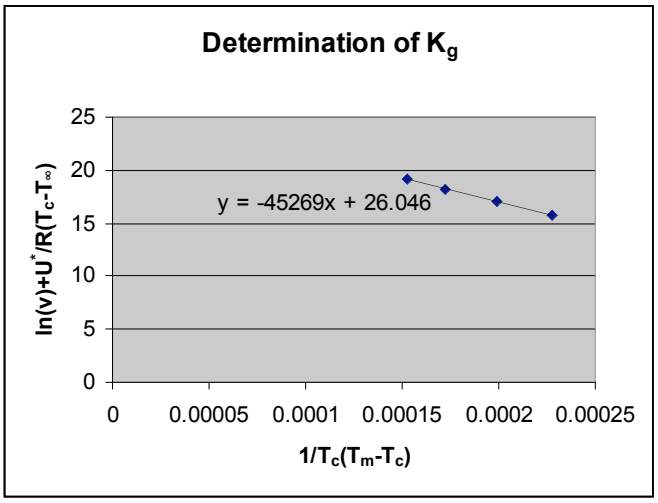


Figure 19: Determination of K_g from animation data.

While measurements were being made, the total volume of crystal transformed was also being recorded in order to complete the Avrami analysis. Figure 20 shows the results of that analysis. In particular, one would expect that, with the two dimensional disks used to depict crystal growth, the Avrami analysis would yield a value of three for the Avrami exponent. Figure 20 demonstrates that this was very nearly the case. The most important

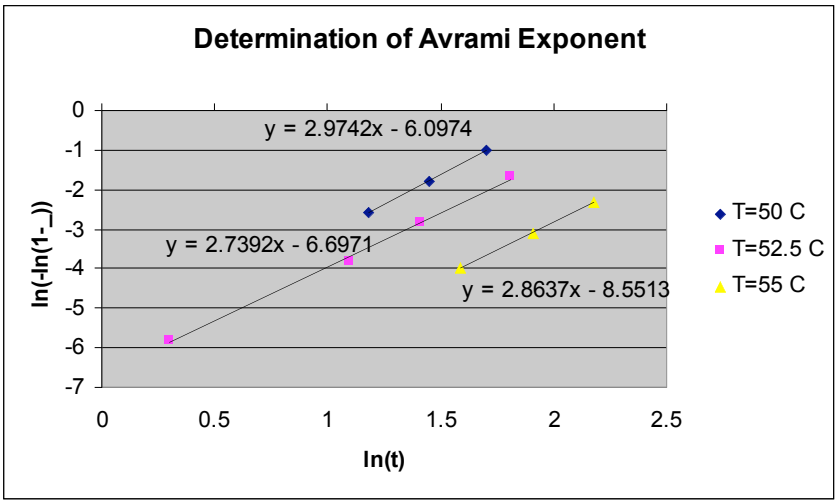


Figure 20: Determination of Avrami exponent from graphical animation data.

thing to realize here is that even with the simplified manner of adding the crystals, the heterogeneous nucleation analysis still holds quite well. It should be noted that the total time required to complete the experiment using the animated applet was on the order of forty-five minutes. This time includes running the simulation, making measurements, and performing analysis. This is a drastic reduction over the amount of time required to complete the actual experiment.

5.7 Conclusion

To summarize, an applet enabling fairly realistic graphical simulation of crystal growth was developed using a Voronoi diagram representation and a numerical calculation for the boundaries. The applet enables a quick yet accurate completion of the same analysis required for the actual Polymerlab experiment. Using this applet, student will be able to quickly test their results or learn through trial and error the correct analysis, while being confident that their measured results are correct. Areas of possible improvement in terms of the model used include better process for inserting nuclei and a temperature dependent nucleation rate. Graphically, a better paint implementation for the Crystal class which took into account boundary collisions and modified the internal boundary representation accordingly would allow for crystals to paint the Maltese cross patterns shown in the first chapter. However, this would be difficult as it might require a more analytical computation of the boundaries of each crystal.

CHAPTER 6

Incorporation of Generic iLab Architecture

The final goal of this project was to facilitate the development of a generic iLab architecture able to handle user interactive experiments like the polymer crystallization experiment, and to incorporate these architectural enhancements into the Polymerlab system where appropriate. Limited progress was made in incorporating the latest design changes into deployed Polymerlab components due to the lack of a completed API suitable for interactive experiments. However, the Polymerlab system was used as a model to direct changes in the communication architecture of the generic iLab framework. Furthermore, the Polymerlab system was used as a model and stub testbed for development and testing of the reservation system needed to facilitate user interactive experiments, including design suggestions that are not currently possible in the Polymerlab setup.

During the development of the first generic iLab APIs and system architecture, the MIT Microelectronics laboratory served as a model system that allowed the designers of the architecture to identify abstract issues that would face internet laboratories [12]. At the same time, it enabled them to stay focused on a concise design, by virtue of constraining them to implement the architecture for that actual system. This initial iteration worked closely with the Microelectronics Weblab team and prioritized their requirements to establish some functionality to the API's core methods. While this system served its purpose well in that the designers were able to implement their APIs to yield a fully functional and robust platform, it also served to limit their design scope to laboratories which use a "batch" architecture. Batch laboratories do not require user input during the actual execution of an experiment. (Other examples of experiment types are interactive

experiments and sensor experiments, such as the MIT flagpole project, in which raw sensor data from a flagpole is streamed to a user.) Batched experiments are the simplest, where a specification is made and submitted, and experiments run without user interaction; they are not subject to many of the demands that a real-time, user interactive experiment must meet. As a result, the iLab architecture also lacked the capability to meet these demands.

6.1 "Batch" iLab Architecture

The following section discusses the iLab architecture as it was developed at the outset of this thesis. Specifically, it describes the implementation which paralleled the Microelectronics Weblab [13]. In general, most remotely run experiments operate in a similar manner: users are identified/ authenticated, they submit input parameters which characterize an experiment to be run, the inputs are validated, the experiment specification is submitted, the experiment is run, and the results are retrieved. By design, the components of the generic iLab architecture correspond to the most general aspects of any internet laboratory, and their functionality and domains strongly embody this concept.

The generic iLab architecture can be broken down into three main components. The first and central of these components is the Service Broker. The Service Broker is an attempt to abstract out the most general and common administrative features of internet laboratories. The guiding force in the design of the Service Broker was the desire to separate out domain-independent functions. Domain-independent functions are those that are common yet not exclusively associated with any particular type of lab. The iLab development team roughly identified five such necessary functions: an experiment storage mechanism for experiment parameters and results, a user authentication/ security mechanism, an authorization mechanism to specify user privileges, a reservation mechanism, and an administrative mechanism to manage user accounts. The Service Broker APIs specifically attempt not to incorporate hardware or vendor specific details. The need is for a component that can facilitate all these common functions and communicate with lab components. The functions provided are not exposed to the lab equipment; instead, only what is necessary for communicating with lab hardware is exposed. The Service Broker is designed to be platform independent and interoperable. In view of this, the Service Broker is implemented as a group of web services, in no small

part due to the presence of accepted and open web standards. In general, the Service Broker is designed to be customizable as far as administrative policies are concerned.

The next main component of the iLab architecture is the Lab Client. The Lab Client is the interface through which users (students) interact. It accepts inputs to be transmitted to lab hardware, and displays experiment results when received. The Lab Client, however, is not able to directly communicate with lab hardware in the original iLab design. In order to enforce an abstraction barrier that would, among other things, allow independent developers to create their own clients to suit their specific needs, all communication with the actual lab hardware is passed as messages through the Service Broker. The Service Broker acts as a middle man, exposing a standard interface to the Lab Client. On the other side of the Service Broker lies the Lab Server. The Lab Server is responsible for converting the input messages specifying experiment conditions and for controlling lab hardware. Like the Lab Client, the Lab Server can only communicate with the Service Broker through a series of standard web services. Figure 21 shows the flow of communication between the three main components in the original iLab architecture. It is useful to contrast this figure with Figure 3 from Chapter 2 (a similar graphic representation of the Polymerlab system.)

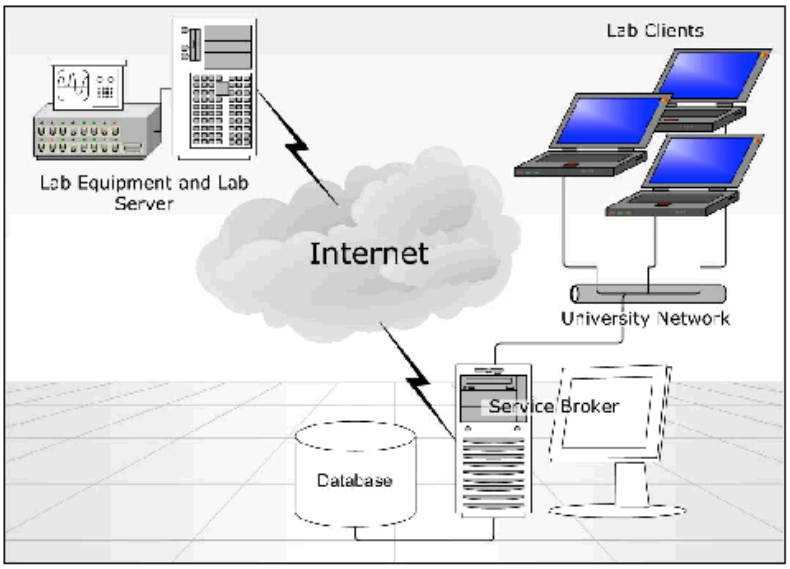


Figure 21: Communication between components of general iLab architecture.

The structure of the original iLab architecture posed several formidable challenges for anyone wishing to implement an interactive experiment using the APIs. Most of these

challenges relate to latency/ performance issues, which are critical in real-time user controlled experiments. The first big performance hit comes about because the initial external architecture, which must be implemented by the Internet-accessible labs in order to communicate with the Service Broker, was designed so that the Service Broker takes in **all** message exchanges centrally before passing them on, as can be easily seen in Figure 9. Two APIs, the client-to-Service-Broker API and the Service Broker-to-lab API establish this centralized communication. The prototype allows messages to be exchanged only between the Lab Client and the Service Broker, or the Lab Server and the Service Broker. No direct communication between server and client is possible. This is a key difference between the Polymerlab system and the iLab general architecture. It allows a third party developer to construct a client to their satisfaction, rather than have one customized client provided by the original developer, as is the case in the Polymer crystallization iLab; however, it also introduces unacceptable latency for several reasons. The SOAP-based web service methods which the APIs facing both the Lab Server and Lab Client are formed around require interpretation using the Web Services Description Language (WSDL). Subsequently, the data must be packaged into XML SOAP request and response messages that are transmitted using HTTP. The parsing, validation, and transmission times introduce large amounts of latency. Furthermore, all transmissions to and from the Service Broker are encrypted, adding yet another layer of latency. In a similar manner, the communication design of the initial architecture dictates that experiment results be written to a database in the Service Broker before being passed back to the client. In a system where experiment results might comprise digital images requiring high bandwidth, the database calls needed to access the records might be overly time intensive.

Not all of the issues preventing interactive experiments are performance related, however. Because the model experiment was a batched experiment which relied on a simple FIFO queue to determine the order to execute experiments, a reservation system was the only area of the above identified domain-independent services which was not implemented in the internal APIs of the Service Broker. While it may seem that this is merely a trivial matter of implementation, the interactions between users on the client end, administrators who set reservation policies, and the need to make connections contingent upon time-based reservation makes such a system difficult in the context of the initial architecture.

In light of all of these concerns, work was begun to identify weaknesses relating to the

implementation of a user interactive experiment and to modify the existing architecture accordingly. In the course of this thesis, work was done with two other Masters students to use the Polymerlab to identify design needs and problem areas, to serve as a model experiment in much the same way as the Microelectronics WebLab did for the initial architecture. Karim Yehiya worked to propose architectural changes to the Service Broker which would address some of the concerns above, while Jedidiah Northridge specifically addressed the development of a Scheduling Service.

6.2 Service Broker Design Changes

As discussed above, the centralized message passing architecture underlying the original Service Broker design introduces inherent problems for interactive experiments. In the Spring term of 2004, Karim Yehia worked with the administrator of the Polymerlab system to identify critical areas of the generic iLab architecture which needed redesign in order to accommodate similar experiments. As a result, three major areas were identified. As part of his thesis, Karim and the iLab development team proposed architecture changes and new APIs to support changes in these areas. These changes primarily focused on the Service Broker [14].

The first and most important design change required by the Service Broker was the ability to allow Lab Clients and Lab Servers to communicate directly with one another via an internal domain-specific protocol. To meet this need the iLab development team introduced the idea of General Ticketing. The primary idea behind General Ticketing is that many of the services previously hosted by the Service Broker should be broken out as stand-alone entities. These stand-alone entities, known as process agents, may run on different systems than the Service Broker. The process agents allow users to bypass the Service Broker for certain method calls, given that they have a valid ticket issued by the Service Broker. This ticketing mechanism closely resembles the token passing from the Framework Server to the Microscope Client in the polymer crystallization iLab. In the new system, the Service Broker has the authority to write tickets and dispense them to clients. Rather than being specific to one particular Lab Server, the general ticketing mechanism offers access to one of any number of resources (process agents) which are known to and advertised by the Service Broker. These resources may include many individual Lab Servers, in keeping with the Service Broker's more general architecture, which allows one Service Broker to provide domain independent services for multiple lab

servers. More importantly, these resources may include services that were formerly fully integrated into the Service Broker, such as an Experiment Storage Service, or a Reservation Service. It is precisely these services which were the next necessary modifications.

6.2.1 Experiment Storage Service

In the first iteration of the iLab architecture, experiment results were communicated directly from the Lab Server to the Service Broker, where they were stored and later retrieved by the Lab Client. For the aforementioned reasons, this system prohibits interactive streaming of results. The ESS changes the treatment of experiment objects and the method they are retrieved. The original API treated experiments as individual units, which doesn't suit the idea of a client making multiple experiment runs (in our case temperature runs) by a client during a single reservation session. The ESS defines experiment objects as sequences of experiment records, each holding some type of payload representing data, error messages, etc. Records are added to the experiment as long as the single session persists. The ESS provides a mechanism for storing large binary results such as images. Instead of transferring the results as an attachment in a web service call, the data is streamed. In the Polymerlab, it happens that the "ESS" which the Microscope Server writes to is the same system which hosts the Framework Server. However, the generic architecture of the iLab ESS allows for this functionality to be separated out, making it more flexible.

6.2.2 Scheduling Server

General ticketing eliminates the static communication pathway between Service Broker and Lab server by allowing web services to be invoked by anyone possessing the proper credentials, in the form of a ticket. This has broader implications for the types of communications possible, as discussed previously, but in the context of scheduling it allows for a reservation system to piggyback on top of the built in time-dependent features of the ticket system.

The development of a reservation system was begun in tandem by Jedidiah Northridge, a Masters student at MIT [15]. Like Yehia, Northridge also made use of the Polymerlab as

a case study to illustrate design needs. Furthermore, modified components of the Polymerlab system were made available to him to use as a stub testbed for the test implementation of the system he designed. Unlike the current implementation of the Polymerlab system, the general iLab architecture wanted to enable a general concept of scheduling which might include factors other than first come first serve. Lab administrators might want to enable auctioning of time slots, for example. In addition, administrators need the flexibility to implement generic rules for describing and restricting reservations, if the system is to be applicable for many types of experiments.

The Scheduling Server is designed to be an optional component that handles the details of creating tickets according to some desired scheduling system. It is designed to operate as both a web service and a web application. The application will allow users to enter interact with the Server to shape the scheduling. Administrators would interact with it in order to lay out reservation policies, while students would use the application to choose reservation times that best fit their schedule. This system is, of course, quite similar to the current Polymerlab system, except that administrative policies regarding scheduling are separated by an abstraction barrier from the details of implementation. Rather than editing code to ensure maximum reservation times, for example, the administrator is able to implement the rules generally through a web interface. The figure below depicts a beta test of Northridge's reservation system which used a modified version of the Microscope Client and Server (bypassing the Framework Server.) In addition to setting up a Scheduling Server, which he developed, and a Service Broker to fill the role occupied by the Framework Server, Northridge modified two components of the Polymerlab system in several ways. The Microscope Server was modified to support the new Service Provider API, as well as process Lab Server tickets received from clients in order to grant access to the server. The Microscope Client was also modified to recognize and use tickets, as well as interface with the Service Brokers web services. These modifications were successful, and Northridge was able to create a reservation system, be passed the applet from the Service Broker, and access the Microscope Server using tickets.

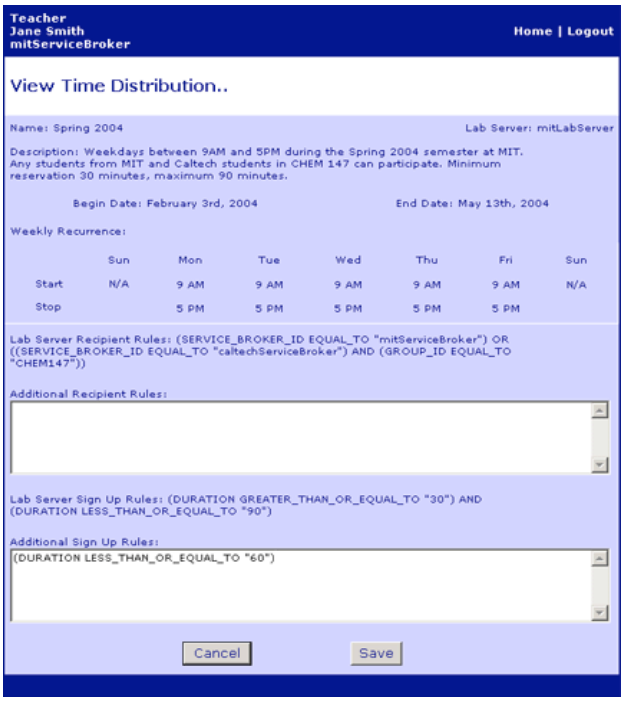


Figure 22: Screenshot from test implementation of Reservation System using Polymerlab

6.3 Future Work

The changes to the Service Broker outlined above and the development of a Scheduling Service position the generic iLab architecture for implementation of a full interactive experiment soon. In order for this system to support the Polymerlab system, several issues must be addressed. One of the most important will be the development of a method for introducing modular internal domain-independent functions that can be customized for a particular lab. This would allow for components such as the ESS to be tweaked for optimal use with the Polymerlab system. For instance, the design outlined by Yehiya for the ESS does not allow individual temperature runs completed during the same user session to be stored or deleted separate from each other. In addition, the Microscope Server and its components would need to be modified to connect to the ESS in the manner desired. While Northridge's test implementation verified the ability to modify the Microscope Client and Server to consume tickets, it did not address the ESS at all. With respect to Multicasting, in general, the General Ticketing mechanism introduced by Yehiya et al. seems general enough to take on this functionality in the same way the Scheduling Server implementation was able to piggy back on top of that system.

CHAPTER 7

Concluding Remarks and Future Work

The Polymerlab system is now able to allow multiple users to collaborate and control the experiment at the same time. Initial testing has supported the idea that such an experiment will better engage students and serve as a useful teaching device. Future steps will include rolling out the simulation applet for use with students to determine its usefulness to them.

While the Polymerlab now has the functionality initially envisioned for it, issues of support become a concern. Although the lab equipment and sample itself are actually stable for extremely long periods of time without user interaction (to change light bulbs, or polymer samples, for example), it is ironic that the component of the system that shows its age most quickly is the software. Without periodic updates to the operating system and network software, the host system may become vulnerable to attacks from outside intruders. This aspect of online laboratories will exist for as long as network security remains a problem. Another aspect of security and support involves the upkeep of the code implementing the Framework Server, Microscope Client, and Microscope Server. To this end, the generic iLab architecture previously discusses has now begun to address the needs of an interactive experiment such as Polymerlab, and indeed is in the process of rolling out the first complete APIs and specifications. Once this is completed, it will be possible to implement the pre-multicasting version of Polymerlab using that architecture. While this will represent a step backward in terms of functionality (because of the lost of multicasting), it will represent a step forward in ensuring that the lab is supported over its lifetime, because the general architecture will have the combined support of every laboratory implementing it.

There are also several areas in which the graphical simulation applet could be improved. One potential improvement includes making the frame rate dependent on growth rate. Additionally, the algorithm used for inserting crystals is quite naïve and should be improved. At the same time the assumption of a pre-set nucleation rate could be modified

to include temperature dependence. The most important area in which the applet could be modified is in the internal representation of the Crystal object. Currently these objects are represented as simple circles that grow and overlap. A more sophisticated internal representation of its boundaries, and importantly is paint method, would allow for a very realistic rendering including the characteristic Maltese cross pattern. Once this is achieved, the applet could very well stand in completely for the actual physical experiment.

Appendix A: SQL Database Script

The following script initializes all the necessary tables and stored procedures, including modifications from previous versions of the polymer crystallization iLab. It should be imported in the standard way through the Enterprise Manager of MS SQL.

```
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].[Reservation_CheckConflict]') and OBJECTPROPERTY(id,
N'IsProcedure') = 1)
drop procedure [dbo].[Reservation_CheckConflict]
GO

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

```
drop procedure [dbo].[Reservation_CreateNew]
GO

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

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

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

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

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[Reservation_Redeem]') and OBJECTPROPERTY(id,
N'IsProcedure') = 1)
drop procedure [dbo].[Reservation_Redeem]
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].[MulticastGroups]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1)
drop table [dbo].[MulticastGroups]
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 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].[MulticastGroups] (
      [ownerID] [uniqueidentifier] NOT NULL ,
      [memberID] [uniqueidentifier] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Reservations] (
      [UserID] [uniqueidentifier] NOT NULL ,
      [ReservationID] [uniqueidentifier] 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)=NULL,
      @UserID uniqueidentifier,
      @AddDate datetime,
```

```
      @OutputXml ntext =NULL
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 OFF
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

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.Reservation_CheckConflict
      @Start datetime,
                @End datetime
AS
      select COUNT(*)
      from Reservations where ((StartTime >= @Start) and (StartTime <=
@End)) or ((EndTime >= @Start) and (EndTime <= @End)) or ((StartTime <=
@Start) and (EndTime >= @End)) or ((StartTime>= @Start) and (EndTime<=
@End))
GO
SET QUOTED_IDENTIFIER OFF
GO
```

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.Reservation_CreateNew
      @ResID uniqueidentifier,
      @UserID uniqueidentifier,
      @StartTime datetime,
      @EndTime datetime
AS
      insert into Reservations
      (ReservationID, UserID, StartTime, EndTime)
      values (@ResID, @UserID, @StartTime, @EndTime)
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.Reservation_GetResFromDay
      @Day datetime
AS
      select ReservationID, UserID,  StartTime, EndTime
      from Reservations where (DAY(StartTime) = DAY(@DAY)) and
(MONTH(StartTime)=MONTH(@DAY)) and (YEAR(StartTime)=YEAR(@DAY))
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.Reservation_GetResFromID
      @ResID uniqueidentifier
AS
      select ReservationID, UserID,  StartTime, EndTime
      from Reservations where ReservationID=@ResID
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
```

```
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.Reservation_GetResFromUserID
      @UserID uniqueidentifier
AS
      select ReservationID, UserID,  StartTime, EndTime
      from Reservations where UserID=@UserID
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.Reservation_IsReservation
      @ResID uniqueidentifier
AS
      select COUNT(*)
      from Reservations where ReservationID=@ResID
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE dbo.Reservation_Redeem
      @UserID uniqueidentifier,
                @Now datetime
AS
      select ReservationID, UserID,  StartTime, EndTime
      from Reservations where UserID=@UserID and (DAY(StartTime) =
DAY(@NOW)) and (MONTH(StartTime)=MONTH(@NOW)) and
(YEAR(StartTime)=YEAR(@NOW)) order by StartTime
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

Appendix B: Heterogeneous Simulation Applet

This appendix provides all Java code necessary to compile and run the heterogeneous τ_o nucleation graphical simulation applet. This applet was an alpha version which calculated boundaries analytically using simple Euclidean distance weighted Voronoi diagram.

B.1 SimTwo.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

import java.util.*;
import javax.swing.Timer;


  // Note the very indirect way control flow works during an animation:
  //
  // (1) We set up an eventListener with a reference to the animationWindow.
  // (2) We set up a timer with a reference to the eventListener.
  // (3) We call timer.start().
  // (4) Every 20 milliseconds the timer calls eventListener.actionPerformed()
  // (5) eventListener.actionPerformed() modifies the logical
  //     datastructure (e.g. changes the coordinates of the ball).
  // (6) eventListener.actionPerformed() calls myWindow.repaint.
  // (7) Swing schedules, at some point in the future, a call to
  //      myWindow.paint()
  // (8) myWindow.paint() tells various objects to paint
  //     themselves on the provided Graphics context.
  //
  // This may seem very complicated, but it makes the coordination of
  // all the various different kinds of user input much easier.  For
  // example here is how control flow works when the user presses the
  // mouse button:
  //
  // (1) We set up an eventListener (actually we just use the same
  //     eventListener that is being used by the timer.)
  // (2) We register the eventListener with the window using the
  //     addMouseListener() method.
  // (3) Every time the mouse button is pressed inside the window the
  //     window calls eventListener.mouseClicked().
  // (4) eventListener.mouseClicked() modifies the logical
  //     datastructures.  (In this example it calls ball.randomBump(), but
  //     in other programs it might do something else, including request a
  //     repaint operation).
  //


class AnimationWindow extends JPanel {
  // overview: an AnimationWindow is an area on the screen in which a
  // crystal animation occurs.  AnimationWindows have two modes:
  // on and off.  During the on mode the crystal grows, during the off
  // mode the crystal doesn't grow.

  private AnimationEventListener eventListener;
  private Vector crystals;
  private AddWeightVoro awvoro;
```

```
private String message;
private double displaytime;
private double pixpermm;
int x1,y1,x2,y2;
boolean mouseclick;

private Timer timer;
private boolean mode;

public AnimationWindow() {

  // effects: initializes this to be in the off mode.



  super();         // do the standard JPanel setup stuff
  pixpermm = 100/0.5;
  mouseclick=false;
  // this only initializes the timer, we actually start and stop the
  // timer in the setMode() method
  eventListener = new AnimationEventListener();
  // The first parameter is how often (in milliseconds) the timer
  // should call us back.  50 milliseconds = 20 frames/second
  timer = new Timer(50, eventListener);

  mode = false;
}

public void initializeCrystals(int n, double r) {
  // this method will eventually take in the number of crystals and the rate
  // and insert the crystals.

  // may be able to use getSize like methods (see Main.java line 153) when
  // creating crystals so that they always get created in the right area
  // no matter if the window has been resized.

  // FOR NOW we assume that crystals are added every 10 time clicks.
  x1 =0;
  x2=0;
  y1=0;
  y2=0;
  mouseclick=false;
  int timeinc = 10;
  int time = 0;
  displaytime=0.0;
  crystals = new Vector();
  Crystal c;
  //double w = (double) super.getSize().width;
  //double h = (double) super.getSize().height;
  double w = 510.0;
  double h = 530.0;

  while (crystals.size() != n) {
    c = new Crystal((int) (w*Math.random()), (int) (h*Math.random()));
    c.setRate(r);
    c.setStartTime(time);
    boolean ok = true;
    for (int i = 0; i < crystals.size(); i++) {
      if (((Crystal) crystals.get(i)).inside(c)) {
          ok = false;
        }
    }
    if (ok) {
        crystals.addElement(c);
        time += timeinc;
    }
  }


  /*
```

```
  c =  new Crystal(100,100);
  c.setRate(r);
  c.setStartTime(30);
  crystals.addElement(c);
  c =  new Crystal(300,300);
  c.setRate(r);
  crystals.addElement(c);
  c =  new Crystal(400,100);
  c.setStartTime(50);
  c.setRate(r);
  crystals.addElement(c);
  c =  new Crystal(450,150);
  c.setRate(r);
  crystals.addElement(c);
  */
  awvoro = new AddWeightVoro(crystals,r);

}

public void resetAnimationWindow(double n, double r) {

  //Resets the animation window to start over
  mouseclick=false;
  initializeCrystals(((int) n),r);
  repaint();
  // remember to add code to clear any imput window. that'll have to happen
  // higher up.
  // this only initializes the timer, we actually start and stop the
  // timer in the setMode() method
  eventListener = new AnimationEventListener();
  // The first parameter is how often (in milliseconds) the timer
  // should call us back.  50 milliseconds = 20 frames/second
  timer = new Timer(50, eventListener);

  mode = false;
}

// This is just here so that we can accept the keyboard focus
public boolean isFocusTraversable() { return true; }

public void paint(Graphics g) {
  // modifies: <g>
  // effects: Repaints the Graphics area <g>.  Swing will then send the
  //          newly painted g to the screen.

  // first repaint the proper background color (controlled by
  // the windowing system)
  Color b = Color.black;
  super.paint(g);

  Color c = new Color(0,0,255);
  paintCrystals(g);
  g.setColor(c);
  awvoro.paint(g);
  g.setColor(b);
  g.drawString("T = "+Math.round(displaytime*1000)+" ms",25,13);
  if ((! mode) && mouseclick) {
    g.drawLine(x1,y1,x2,y2);
    double length = Math.pow((double) ((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)),0.5);
    length = length/pixpermm;
    g.drawString(Math.round(length*1000) +" nm",x2+10,y2+10);
    x1=0;
    y1=0;
    x2=0;
    y2=0;
    mouseclick=false;
  }

}

public void paintCrystals(Graphics g) {
```

```
  // modifies: <g>
  // effects: Repaints the Graphics area <g>.  Swing will then send the
  //          newly painted g to the screen.

  // calls the paint method of each crystal in the crystals vector.
  //
  for (int i = 0; i < crystals.size(); i++) {
    ((Crystal) crystals.get(i)).paint(g);
  }
  for (int i = 0; i < crystals.size(); i++) {
    ((Crystal) crystals.get(i)).paintCenter(g);
  }
}



public void setMode(boolean m) {
  // modifies: this
  // effects: changes the mode to <m>.

  if (mode == true) {
    // we're about to change mode: turn off all the old listeners
    addMouseListener(eventListener);
    addMouseMotionListener(eventListener);
    removeKeyListener(eventListener);
  } else {
    removeMouseListener(eventListener);
    removeMouseMotionListener(eventListener);
  }

  mode = m;

  if (mode == true) {
    // the mode is true: turn on the listeners
    removeMouseListener(eventListener);
    removeMouseMotionListener(eventListener);
    addKeyListener(eventListener);
    requestFocus();           // make sure keyboard is directed to us
    timer.start();
  }
  else {
    timer.stop();
  }
}



class AnimationEventListener extends MouseAdapter
implements MouseMotionListener, KeyListener, ActionListener
{
  // overview: AnimationEventListener is an inner class that
  // responds to all sorts of external events, and provides the
  // required semantic operations for our particular program.  It
  // owns, and sends semantic actions to the crystal and window of the
  // outer class

  // MouseAdapter gives us empty methods for the MouseListener
  // interface: mouseClicked, mouseEntered, mouseExited, mousePressed,
  // and mouseReleased.

  // for this example we only need to override mouseClicked
  public void mouseClicked(MouseEvent e) {
  }

  public void mousePressed(MouseEvent e) {
    //System.out.println("mousePressed "+e.getX()+","+e.getY());
    x1=e.getX();
    y1=e.getY();
  }

  public void mouseReleased(MouseEvent e) {
```

```
      //System.out.println("mouseReleased "+e.getX()+","+e.getY());
          x2=e.getX();
          y2=e.getY();
          mouseclick=true;
          repaint(0,0,510,530);
    }



    // Here's the MouseMotionListener interface
    public void mouseDragged(MouseEvent e) { }
    public void mouseMoved(MouseEvent e) { }

    // Here's the KeyListener interface
    public void keyPressed(KeyEvent e) {
      // modifes: the ball that this listener owns
      // effects: causes the crystal to stop growing but
      //          only if one of the keys A-J is pressed.

      // Come back and rewrite this? Is this the desired functionality?
      // Better to have it call the stop button function. As currently
      // written it permanently stops growth.

      int keynum = e.getKeyCode();

      if ((keynum >= 65) && (keynum <= 74)) {
        System.out.println("keypress " + e.getKeyCode());
        //for (int i = 0; i < crystals.size(); i++) {
          //  ((Crystal) crystals.get(i)).stopGrowth();
          //}
      }
    }
    public void keyReleased(KeyEvent e) { }
    public void keyTyped(KeyEvent e) { }

    // this is the callback for the timer
    public void actionPerformed(ActionEvent e) {
      // modifes: both the crystal and the window that this listener owns
      // effects: causes the crystal to grow and the window to be updated
      //          to show the new position of the ball.

      // MAY BE ROOM TO IMPROVE PERFORMANCE BELOW. MAYBE NOT THOUGH.
      Rectangle rec;
      Rectangle repaintArea;
      Crystal c;
      for (int i = 0; i < crystals.size(); i++) {
          c = ((Crystal) crystals.get(i));
          rec = c.boundingBox();
          c.growCrystal();
          repaintArea = rec.union(c.boundingBox());
          //repaint(repaintArea.x,
          //repaintArea.y,
          //repaintArea.width,
          //repaintArea.height);
          repaint(0,0,510,530);
      }
      // NOTE THAT THE TIME INCREMENT DEPENDS ON THE FRAME RATE
      // We assume below that the frame rate is 20 frames/sec.
      displaytime += 0.05;

      //repaint();

      // Have Swing tell the AnimationWindow to run its paint()
      // method.  One could also call repaint(), but this would
      // repaint the entire window as opposed to only the portion that
      // has changed.

    }
  }
}
```

```
public class Simulator extends javax.swing.JApplet {
  // overview: An ApplicationWindow is a top level program window that
  // contains a toolbar and an animation window.

  protected AnimationWindow animationWindow;
  JTextField numfield;
  JTextField ratefield;
  public double pixpermm = 100/0.5;
  public double num = 10.0;
  public double rate = 1.0;

  public void init() {
    // effects: Initializes the application window so that it contains
    //          a toolbar and an animation window.


    // Title bar
    //super("Crystal Demonstration Program");



    //Create the toolbar.
    JToolBar toolBar = new JToolBar();
    addButtons(toolBar);

    //Create the animation area used for output.
    animationWindow = new AnimationWindow();
    // Put it in a scrollPane, (this makes a border)
    JScrollPane scrollPane = new JScrollPane(animationWindow);

    //Lay out the content pane.
    JPanel contentPane = new JPanel();
    contentPane.setLayout(new BorderLayout());
    contentPane.setPreferredSize(new Dimension(510, 550));
    contentPane.add(toolBar, BorderLayout.NORTH);
    contentPane.add(scrollPane, BorderLayout.CENTER);
    setContentPane(contentPane);
     //Initialize the crystals.
    animationWindow.initializeCrystals(10,1.0);
  }

  protected void addButtons(JToolBar toolBar) {
    // modifies: toolBar
    // effects: adds Run, Stop, Reset, and Quit buttons to toolBar

    JButton button = null;
    numfield = null;
    ratefield = null;
    JLabel label = null;


    button = new JButton("Run");
    button.setToolTipText("Start the animation");
    // when this button is pushed it calls animationWindow.setMode(true)
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        animationWindow.setMode(true);
      }
    });
    toolBar.add(button);

    button = new JButton("Stop");
    button.setToolTipText("Stop the animation");
    // when this button is pushed it calls animationWindow.setMode(false)
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        animationWindow.setMode(false);
      }
    });
    toolBar.add(button);
```

```
  button = new JButton("Reset");
  button.setToolTipText("Reset the animation");
  // when this button is pushed it calls animationWindow.setMode(false)
  // and animationWindow.resetAnimationWindow()
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      animationWindow.setMode(false);
        animationWindow.resetAnimationWindow(num,rate);
    }
  });
  toolBar.add(button);

  button = new JButton("Quit");
  button.setToolTipText("Quit the program");
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      System.exit(0);
    }
  });
  toolBar.add(button);

  label = new JLabel("Crystals:");
  label.setToolTipText("Number of crystals to appear");
  toolBar.add(label);

  numfield = new JTextField("10");
  numfield.addFocusListener(new java.awt.event.FocusAdapter() {
          public void focusLost(java.awt.event.FocusEvent evt) {
              numfieldFocusLost(evt);
          }});
  toolBar.add(numfield);

  label = new JLabel("Temp (C):");
  label.setToolTipText("Temperature of the sample");
  toolBar.add(label);

  ratefield = new JTextField("1.0");
  ratefield.addFocusListener(new java.awt.event.FocusAdapter() {
          public void focusLost(java.awt.event.FocusEvent evt) {
              ratefieldFocusLost(evt);
          }});
  toolBar.add(ratefield);

   button = new JButton("Set");
  button.setToolTipText("Set the crystal values");
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      setButtonPressed();
    }
  });
  toolBar.add(button);
}

private void setButtonPressed() {
  animationWindow.setMode(false);
  num = Double.parseDouble(numfield.getText().trim());
  if (num < 0 || num > 20) {
    num = 10.0;
  }
  rate = Double.parseDouble(ratefield.getText().trim());
  if (rate < 0 || rate > 10) {
    rate = 1.0;
  }
  animationWindow.resetAnimationWindow(num,rate);
}

private void numfieldFocusLost(java.awt.event.FocusEvent evt) {
  try {
    //num = Double.parseDouble(numfield.getText().trim());
    //if (num < 0 || num > 1000) {
        //do something about popping up an error box.
```

```
      //}
    } catch (NumberFormatException e) {
      //
    }
  }

  private void ratefieldFocusLost(java.awt.event.FocusEvent evt) {
    try {
      //rate = Double.parseDouble(ratefield.getText().trim());
      //if (rate < 0 || rate > 1000) {
          //do something about popping up an error box.
      //}
    } catch (NumberFormatException e) {
      //
    }
  }

}
```

B.2 Crystal.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

import java.util.*;

public class Crystal {
  // Overview: A Crystal is a mutable data type.  It simulates a
  // crystal growing inside a two dimensional box.  It also
  // provides methods that are useful for creating animations of the
  // crystal as it grows

  // Note that may want to change the dimensions of the box.

  private int x = (int)((Math.random() * 600.0) - 0.0);
  private int y = (int)((Math.random() * 600.0) - 0.0);

  private double rate = 1.0;
  private double radius = 0.0;
  private Color color = new Color(255, 0, 0);
  private Color color2 = new Color(0,255,0);
  private int start = 0;
  private Point center = new Point(x, y);


  public Crystal(int a, int b) {
    center = new Point(a, b);
    this.x = a;
    this.y = b;
  }

  public Crystal() {
  }

  public boolean inside(Crystal c) {
    Point p = c.getCenter();
    int d = center.distanceToPoint(p);
    int diff = c.getStartTime()-start;
    if (d<=diff*rate)
      return true;
    else
      return false;
  }

  public void setStartTime(int i) {
    start = i;
```

```
}

public int getStartTime() {
  return start;
}

 public void changeColor() {
  color = new Color(0,255,0);
}

public Point getCenter() {
  return center;
}

public double getRadius() {
  return radius;
}

public void setRadius(double rnew) {
  this.radius = rnew;
}

public void setRate(double i) {
  rate = i;
}

public void growCrystal() {
  if (start == 0) {
    radius += rate;
  }
}

public void stopGrowth() {
  rate = 0.0;
}

public void paint(Graphics g) {
  // modifies: the Graphics object <g>.
  // effects: paints a circle on <g> reflecting the current position
  // of the ball.

  // the "clip rectangle" is the area of the screen that needs to be
  // modified
  if (start == 0) {
    Rectangle clipRect = g.getClipBounds();
    int rad = (int) radius;
    // For this tiny program, testing whether we need to redraw is
    // kind of silly.  But when there are lots of objects all over the
    // screen this is a very important performance optimization
    if (clipRect.intersects(this.boundingBox())) {
        g.setColor(color);
        g.fillOval(x-rad, y-rad, rad+rad, rad+rad);

    }
  } else {
    start = start - 1;
  }
}

public void paintCenter(Graphics g) {
  if (start == 0) {
    Rectangle clipRect = g.getClipBounds();

    // For this tiny program, testing whether we need to redraw is
    // kind of silly.  But when there are lots of objects all over the
    // screen this is a very important performance optimization
    if (clipRect.intersects(this.boundingBox())) {
        g.setColor(color2);
        g.fillOval(x-1, y-1, 1, 1);
    }}
    }
```

```
  public Rectangle boundingBox() {
    // effect: Returns the smallest rectangle that completely covers the
    //         current position of the crystal.

    // a Rectangle is the x,y for the upper left corner and then the
    // width and height
    int rad = (int) radius;
    return new Rectangle(x-rad, y-rad, rad+rad+1, rad+rad+1);
  }
}
```

B.3 Point.java and Line.java

```
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

public class Point {
  // Overview: A Point is just what it seems.  It has an x and a y value,
  // and a few methods for determining distances to other points.
  private int x;
  private int y;

  // Constructor
  public Point(int x, int y) {
    // Creates a new point
    this.x = x;
    this.y = y;
  }

  public int xval() {
    return x;
  }

  public int yval() {
    return y;
  }

  public void setX(int xval) {
    this.x =  xval;
  }

  public void setY(int yval) {
    this.y =  yval;
  }

  public int distanceToPoint(Point p) {
    // modifies:
    // effects:
    // returns: distance to Point p
     double px = (double) p.xval();
     double py = (double) p.yval();
     double dx = (double) x;
     double dy = (double) y;
     int d = (int) Math.sqrt( Math.pow((px-dx),2.0) + Math.pow((py-dy),2.0));
     return d;
   }

}

public class Line {
  // Overview: A Line is just what it seems.  It has two endpoints which each
  // have an x and a y value.

  public int x1;
  public int y1;
  public int x2;
  public int y2;
```

```
  // Constructor
  public Line(int x1, int y1, int x2, int y2) {
    // Creates a new point
    this.x1 = x1;
    this.y1 = y1;
    this.x2 = x2;
    this.y2 = y2;
  }

  public int x1val() {
    return x1;
  }

  public int y1val() {
    return y1;
  }

  public void setX1(int xval) {
    this.x1 =  xval;
  }

  public void setY1(int yval) {
    this.y1 =  yval;
  }

   public int x2val() {
    return x2;
  }

  public int y2val() {
    return y2;
  }

  public void setX2(int xval) {
    this.x2 =  xval;
  }

  public void setY2(int yval) {
    this.y2 =  yval;
  }

}
```

Appendix C: Homogeneous Simulation Applet

This appendix provides the Java code necessary to compile and run the homogeneous nucleation graphical simulation applet. This applet calculates boundaries numerically using simple additively weighted Euclidean distance. (The code for the new numerical AddWeightVoro class is also included.) This applet takes in a temperature and yields a correct crystal growth rate for that temperature. It also includes a built-in crystal measuring ability to simplify analysis of the animation results.

C.1 Simulator2.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

import java.util.*;
import javax.swing.Timer;


  // Note the very indirect way control flow works during an animation:
  //
  // (1) We set up an eventListener with a reference to the animationWindow.
  // (2) We set up a timer with a reference to the eventListener.
  // (3) We call timer.start().
  // (4) Every 20 milliseconds the timer calls eventListener.actionPerformed()
  // (5) eventListener.actionPerformed() modifies the logical
  //     datastructure (e.g. changes the coordinates of the ball).
  // (6) eventListener.actionPerformed() calls myWindow.repaint.
  // (7) Swing schedules, at some point in the future, a call to
  //      myWindow.paint()
  // (8) myWindow.paint() tells various objects to paint
  //     themselves on the provided Graphics context.
  //
  // This may seem very complicated, but it makes the coordination of
  // all the various different kinds of user input much easier.  For
  // example here is how control flow works when the user presses the
  // mouse button:
  //
  // (1) We set up an eventListener (actually we just use the same
  //     eventListener that is being used by the timer.)
  // (2) We register the eventListener with the window using the
  //     addMouseListener() method.
  // (3) Every time the mouse button is pressed inside the window the
  //     window calls eventListener.mouseClicked().
  // (4) eventListener.mouseClicked() modifies the logical
  //     datastructures.  (In this example it calls ball.randomBump(), but
  //     in other programs it might do something else, including request a
  //     repaint operation).
  //


class AnimationWindow extends JPanel {
  // overview: an AnimationWindow is an area on the screen in which a
  // crystal animation occurs.  AnimationWindows have two modes:
  // on and off.  During the on mode the crystal grows, during the off
  // mode the crystal doesn't grow.
```

```
private AnimationEventListener eventListener;
private Vector crystals;
private AddWeightVoro awvoro;
private String message;
private double displaytime;
private double pixpermm;
int x1,y1,x2,y2;
boolean mouseclick;

private Timer timer;
private boolean mode;

public AnimationWindow() {

  // effects: initializes this to be in the off mode.



  super();         // do the standard JPanel setup stuff
  pixpermm = 100/0.05;
  mouseclick=false;
  // this only initializes the timer, we actually start and stop the
  // timer in the setMode() method
  eventListener = new AnimationEventListener();
  // The first parameter is how often (in milliseconds) the timer
  // should call us back.  50 milliseconds = 20 frames/second
  timer = new Timer(50, eventListener);

  mode = false;
}

public void initializeCrystals(int n, double r) {
  // this method will eventually take in the number of crystals and the rate
  // and insert the crystals.

  // may be able to use getSize like methods (see Main.java line 153) when
  // creating crystals so that they always get created in the right area
  // no matter if the window has been resized.

  // FOR NOW we assume that crystals are added every 10 time clicks.
  x1 =0;
  x2=0;
  y1=0;
  y2=0;
  mouseclick=false;
  int timeinc = 10;
  int time = 0;
  displaytime=0.0;
  crystals = new Vector();
  Crystal c;
  //double w = (double) super.getSize().width;
  //double h = (double) super.getSize().height;
  double w = 510.0;
  double h = 530.0;

  while (crystals.size() != n) {
    c = new Crystal((int) (w*Math.random()), (int) (h*Math.random()));
    c.setRate(r);
    c.setStartTime(time);
    boolean ok = true;
    for (int i = 0; i < crystals.size(); i++) {
      if (((Crystal) crystals.get(i)).inside(c)) {
          ok = false;
       }
    }
    if (ok) {
        crystals.addElement(c);
        time += timeinc;
    }
```

```
  }


  /*
  c =  new Crystal(100,100);
  c.setRate(r);
  c.setStartTime(30);
  crystals.addElement(c);
  c =  new Crystal(300,300);
  c.setRate(r);
  crystals.addElement(c);
  c =  new Crystal(400,100);
  c.setStartTime(50);
  c.setRate(r);
  crystals.addElement(c);
  c =  new Crystal(450,150);
  c.setRate(r);
  crystals.addElement(c);
  */
  awvoro = new AddWeightVoro(crystals,r);

}

public void resetAnimationWindow(double n, double r) {

  //Resets the animation window to start over
  mouseclick=false;
  initializeCrystals(((int) n),r);
  repaint();
  // remember to add code to clear any imput window. that'll have to happen
  // higher up.
  // this only initializes the timer, we actually start and stop the
  // timer in the setMode() method
  eventListener = new AnimationEventListener();
  // The first parameter is how often (in milliseconds) the timer
  // should call us back.  50 milliseconds = 20 frames/second
  timer = new Timer(50, eventListener);

  mode = false;
}

// This is just here so that we can accept the keyboard focus
public boolean isFocusTraversable() { return true; }

public void paint(Graphics g) {
  // modifies: <g>
  // effects: Repaints the Graphics area <g>.  Swing will then send the
  //          newly painted g to the screen.

  // first repaint the proper background color (controlled by
  // the windowing system)
  Color b = Color.black;
  super.paint(g);

  Color c = new Color(0,0,255);
  paintCrystals(g);
  g.setColor(c);
  awvoro.paint(g);
  g.setColor(b);
  g.drawString("T = "+Math.round(displaytime*1000)+" ms",25,13);
  if ((! mode) && mouseclick) {
    g.drawLine(x1,y1,x2,y2);
    double length = Math.pow((double) ((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)),0.5);
    length = length/pixpermm;
    g.drawString(Math.round(length*10000) +"E-4 mm",x2+10,y2+10);
    x1=0;
    y1=0;
    x2=0;
    y2=0;
    mouseclick=false;
  }
```

```
}

public void paintCrystals(Graphics g) {
  // modifies: <g>
  // effects: Repaints the Graphics area <g>.  Swing will then send the
  //          newly painted g to the screen.

  // calls the paint method of each crystal in the crystals vector.
  //
  for (int i = 0; i < crystals.size(); i++) {
    ((Crystal) crystals.get(i)).paint(g);
  }
  for (int i = 0; i < crystals.size(); i++) {
    ((Crystal) crystals.get(i)).paintCenter(g);
  }
}



public void setMode(boolean m) {
  // modifies: this
  // effects: changes the mode to <m>.

  if (mode == true) {
    // we're about to change mode: turn off all the old listeners
    addMouseListener(eventListener);
    addMouseMotionListener(eventListener);
    removeKeyListener(eventListener);
  } else {
    removeMouseListener(eventListener);
    removeMouseMotionListener(eventListener);
  }

  mode = m;

  if (mode == true) {
    // the mode is true: turn on the listeners
    removeMouseListener(eventListener);
    removeMouseMotionListener(eventListener);
    addKeyListener(eventListener);
    requestFocus();           // make sure keyboard is directed to us
    timer.start();
  }
  else {
    timer.stop();
  }
}



class AnimationEventListener extends MouseAdapter
implements MouseMotionListener, KeyListener, ActionListener
{
  // overview: AnimationEventListener is an inner class that
  // responds to all sorts of external events, and provides the
  // required semantic operations for our particular program.  It
  // owns, and sends semantic actions to the crystal and window of the
  // outer class

  // MouseAdapter gives us empty methods for the MouseListener
  // interface: mouseClicked, mouseEntered, mouseExited, mousePressed,
  // and mouseReleased.

  // for this example we only need to override mouseClicked
  public void mouseClicked(MouseEvent e) {
  }

  public void mousePressed(MouseEvent e) {
    //System.out.println("mousePressed "+e.getX()+","+e.getY());
    x1=e.getX();
```

```
  y1=e.getY();
}

public void mouseReleased(MouseEvent e) {
  //System.out.println("mouseReleased "+e.getX()+","+e.getY());
      x2=e.getX();
      y2=e.getY();
      mouseclick=true;
      repaint(0,0,510,530);
}



// Here's the MouseMotionListener interface
public void mouseDragged(MouseEvent e) { }
public void mouseMoved(MouseEvent e) { }

// Here's the KeyListener interface
public void keyPressed(KeyEvent e) {
  // modifes: the ball that this listener owns
  // effects: causes the crystal to stop growing but
  //          only if one of the keys A-J is pressed.

  // Come back and rewrite this? Is this the desired functionality?
  // Better to have it call the stop button function. As currently
  // written it permanently stops growth.

  int keynum = e.getKeyCode();

  if ((keynum >= 65) && (keynum <= 74)) {
    System.out.println("keypress " + e.getKeyCode());
    //for (int i = 0; i < crystals.size(); i++) {
      //  ((Crystal) crystals.get(i)).stopGrowth();
      //}
  }
}
public void keyReleased(KeyEvent e) { }
public void keyTyped(KeyEvent e) { }

// this is the callback for the timer
public void actionPerformed(ActionEvent e) {
  // modifes: both the crystal and the window that this listener owns
  // effects: causes the crystal to grow and the window to be updated
  //          to show the new position of the ball.

  // MAY BE ROOM TO IMPROVE PERFORMANCE BELOW. MAYBE NOT THOUGH.
  Rectangle rec;
  Rectangle repaintArea;
  Crystal c;
  for (int i = 0; i < crystals.size(); i++) {
      c = ((Crystal) crystals.get(i));
      rec = c.boundingBox();
      c.growCrystal();
      repaintArea = rec.union(c.boundingBox());
      //repaint(repaintArea.x,
      //repaintArea.y,
      //repaintArea.width,
      //repaintArea.height);
      repaint(0,0,510,530);
  }
  // NOTE THAT THE TIME INCREMENT DEPENDS ON THE FRAME RATE
  // We assume below that the frame rate is 20 frames/sec.
  displaytime += 0.05;

  //repaint();

  // Have Swing tell the AnimationWindow to run its paint()
  // method.  One could also call repaint(), but this would
  // repaint the entire window as opposed to only the portion that
  // has changed.
```

```
    }
  }
}

public class Simulator2 extends javax.swing.JApplet {
  // overview: An ApplicationWindow is a top level program window that
  // contains a toolbar and an animation window.

  protected AnimationWindow animationWindow;
  JTextField numfield;
  JTextField ratefield;
  public double pixpermm = 100/0.05;
  public double framespersec = 20.0;
  public double num = 10.0;
  public double rate = 0.21553;
  public double temp = 55.0;

  public void init() {
    // effects: Initializes the application window so that it contains
    //          a toolbar and an animation window.


    // Title bar
    //super("Crystal Demonstration Program");



    //Create the toolbar.
    JToolBar toolBar = new JToolBar();
    addButtons(toolBar);

    //Create the animation area used for output.
    animationWindow = new AnimationWindow();
    // Put it in a scrollPane, (this makes a border)
    JScrollPane scrollPane = new JScrollPane(animationWindow);

    //Lay out the content pane.
    JPanel contentPane = new JPanel();
    contentPane.setLayout(new BorderLayout());
    contentPane.setPreferredSize(new Dimension(510, 550));
    contentPane.add(toolBar, BorderLayout.NORTH);
    contentPane.add(scrollPane, BorderLayout.CENTER);
    setContentPane(contentPane);
     //Initialize the crystals.
    animationWindow.initializeCrystals(10,0.21553);
  }

  protected void addButtons(JToolBar toolBar) {
    // modifies: toolBar
    // effects: adds Run, Stop, Reset, and Quit buttons to toolBar

    JButton button = null;
    numfield = null;
    ratefield = null;
    JLabel label = null;


    button = new JButton("Run");
    button.setToolTipText("Start the animation");
    // when this button is pushed it calls animationWindow.setMode(true)
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        animationWindow.setMode(true);
      }
    });
    toolBar.add(button);

    button = new JButton("Stop");
    button.setToolTipText("Stop the animation");
    // when this button is pushed it calls animationWindow.setMode(false)
    button.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
      animationWindow.setMode(false);
    }
  });
  toolBar.add(button);

  button = new JButton("Reset");
  button.setToolTipText("Reset the animation");
  // when this button is pushed it calls animationWindow.setMode(false)
  // and animationWindow.resetAnimationWindow()
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      animationWindow.setMode(false);
        animationWindow.resetAnimationWindow(num,rate);
    }
  });
  toolBar.add(button);

  button = new JButton("Quit");
  button.setToolTipText("Quit the program");
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      System.exit(0);
    }
  });
  toolBar.add(button);

  label = new JLabel("Crystals:");
  label.setToolTipText("Number of crystals to appear");
  toolBar.add(label);

  numfield = new JTextField("10");
  numfield.addFocusListener(new java.awt.event.FocusAdapter() {
          public void focusLost(java.awt.event.FocusEvent evt) {
              numfieldFocusLost(evt);
          }});
  toolBar.add(numfield);

  label = new JLabel("Temp (C):");
  label.setToolTipText("Temperature of the sample");
  toolBar.add(label);

  ratefield = new JTextField("55.0");
  ratefield.addFocusListener(new java.awt.event.FocusAdapter() {
          public void focusLost(java.awt.event.FocusEvent evt) {
              ratefieldFocusLost(evt);
          }});
  toolBar.add(ratefield);

   button = new JButton("Set");
  button.setToolTipText("Set the crystal values");
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      setButtonPressed();
    }
  });
  toolBar.add(button);
}

private void setButtonPressed() {
  animationWindow.setMode(false);
  num = Double.parseDouble(numfield.getText().trim());
  if (num < 0 || num > 15) {
    num = 10.0;
  }
  temp = Double.parseDouble(ratefield.getText().trim());
  if (temp < 48 || rate > 62) {
    rate = 0.21553;
    temp = 55.0;
  } else {
```

```
      rate = Math.exp(26.147-(29300.0/(8.3144*(temp+97.15)))-
45816.0/((temp+273.15)*(70.3-temp)))*pixpermm/framespersec;
    }
    animationWindow.resetAnimationWindow(num,rate);
  }

  private void numfieldFocusLost(java.awt.event.FocusEvent evt) {
    try {
      //num = Double.parseDouble(numfield.getText().trim());
      //if (num < 0 || num > 1000) {
          //do something about popping up an error box.
      //}
    } catch (NumberFormatException e) {
      //
    }
  }

  private void ratefieldFocusLost(java.awt.event.FocusEvent evt) {
    try {
      //rate = Double.parseDouble(ratefield.getText().trim());
      //if (rate < 0 || rate > 1000) {
          //do something about popping up an error box.
      //}
    } catch (NumberFormatException e) {
      //
    }
  }

}
```

C.2 AddWeightVoro.java

```
import java.awt.Graphics;
import java.applet.Applet;
import java.awt.Color;
import java.util.*;
public class AddWeightVoro {
  Color col1,col2,col3;
  double pi=3.14159265358979;
  int num;
  int winHeight,winWidth;
  int i,j, k, m;
  double dist,be,dist2,xc,yc,yy,slope,th,miny,maxy;
  int rr,xj;
  double a1r,a2r,yr;
  double ymr,x2r,y2r,d3,d4;
  int x2rI,y2rI,rwe;
  int yjI,minyI,maxyI,x10I,y10I;
  double b1y,b2y,b3y,b4y,x0,x10,y10,d5,d6;
  int br2,br3;
  double rate;
  double[] x1;
  double[] y1;
  double[] w1;
  int[] x;
  int[] y;
  int[] w;
  double[] s;
  String[] sss;
  Vector crystals;
  boolean drawn;
  Vector lines;
  Line l;

  public double dou(String doublestring){
    double d;
    d = (Double.valueOf(doublestring)).doubleValue();
```

```
  return d;
}

public AddWeightVoro(Vector cs, double r){
  //Takes in a Vector of Crystals.
  this.crystals = cs;
  rate = r;
  int n = cs.size();
  col1=new Color(204,204,204);
  col2=Color.yellow;
  col3=Color.blue;
  drawn = false;
  lines = new Vector();

  winWidth=510;
  winHeight=530;
  num=n;
  x1=new double[n];
  y1=new double[n];
  w1=new double[n];
  x=new int[n];
  y=new int[n];
  w=new int[n];
  s=new double[n];
  sss=new String[n];
  for(k=0;k<num;k++){
    x1[k]=((Point) ((Crystal) crystals.get(k)).getCenter()).xval();
    y1[k]=((Point) ((Crystal) crystals.get(k)).getCenter()).yval();
    w1[k]=((Crystal) crystals.get(k)).getStartTime();
    x[k]=(int)(x1[k]+0.5);
    y[k]=(int)(y1[k]+0.5);
    w[k]=(int)(w1[k]+0.5);
    sss[k]=""+w[k];
  }
}


public double pow(double a,double b){
  double ans;
  ans=Math.pow(a,b);
  return ans;
}
public double atan(double a){
  double ans;
  ans=Math.atan(a);
  return ans;
}
public double sin(double a){
  double ans;
  ans=Math.sin(a);
  return ans;
}
public double cos(double a){
  double ans;
  ans=Math.cos(a);
  return ans;
}


void heap(double weight[],double xval[],double yval[],int n){
  int k,half,i,j,m;
  double b1,b2,b3,c1,c2,c3;
  half=(int)(n/2);
  for(k=half;k>=1;k--){
    i=k;
    b1=weight[i-1];
    b2=xval[i-1];
    b3=yval[i-1];
    while(2*i<=n){
      j=2*i;
      if(j+1<=n){
```

```
        if(weight[j-1]<weight[j]){
          j++;
        }
      }
      if(weight[j-1]<=b1){
        break;
      }
      weight[i-1]=weight[j-1];
       xval[i-1]=xval[j-1];
       yval[i-1]=yval[j-1];
      i=j;
    }//wend
    weight[i-1]=b1;
    xval[i-1]=b2;
    yval[i-1]=b3;
  }//next k
  for(m=n-1;m>=1;m--){
    c1=weight[m];
    c2=xval[m];
    c3=yval[m];
    weight[m]=weight[0];
    xval[m]=xval[0];
    yval[m]=yval[0];
    i=1;
    while(2*i<=m){
      k=2*i;
      if(k+1<=m){
        if(weight[k-1]<=weight[k]){
          k++;
        }
      }
      if(weight[k-1]<=c1){
        break;
      }
      weight[i-1]=weight[k-1];
       xval[i-1]=xval[k-1];
       yval[i-1]=yval[k-1];
      i=k;
    }//wend
    weight[i-1]=c1;
    xval[i-1]=c2;
    yval[i-1]=c3;
  }//next m
}
public void paint(java.awt.Graphics g){
  //g.setColor(col3);
  //g.drawLine(winWidth-150,10,winWidth-50,10);
  //g.drawLine(winWidth-150,5,winWidth-150,15);
  //g.drawLine(winWidth-100,5,winWidth-100,10);
  //g.drawLine(winWidth-50,5,winWidth-50,15);
  //g.drawString("0",winWidth-158,13);
  //g.drawString("100",winWidth-48,13);
  g.setColor(col1);
  if (! drawn) {
    heap(w1,x1,y1,num);
    //g.setColor(col3);
    for(i=1;i<=num-1;i++){
        for(j=i+1;j<=num;j++){
          dist=pow(pow(x1[i-1]-x1[j-1],2)+pow(y1[i-1]-y1[j-1],2),0.5);
          be=w1[j-1]-w1[i-1];
          be = (int) (be*rate);
          // The following if-statement checks to make sure the crystal can
          // event occur, though this should already be insured. If
          // it can't, nothing happens.
          if(dist>be){
            dist2=dist/2;
            xc=(x1[i-1]+x1[j-1])/2;  // find the mid x dist between the points
            yc=(y1[i-1]+y1[j-1])/2;  // find the mid y dist between the points
            yy=yc-y1[i-1];           // find the delta_y from point 1 to mid
            slope=yy/dist2;             // slope is a misnomer
```

```
            th=atan(slope/pow(1-slope*slope,0.5)); // calculate the angle the line makes
to x axis
            if(x1[i-1]<x1[j-1]){
              th=pi-atan(slope/pow(1-slope*slope,0.5));
            }
            miny=0;
            maxy=0;
            rr=0;
            xj=0;
            a2r=0.5/be;
            /*
            while(rr==0){    // do this while inside the bounds of the viewing window or
xj<100
              a1r=16*dist2*dist2*xj*xj-4*be*be*xj*xj-4*dist2*dist2*be*be+pow(be,4);
              if(a1r>=0){
                  yr=a2r*pow(a1r,0.5);
                  // sometimes change sign below to see effect
                  ymr=yr;
                  x2r=xc-cos(-th)*xj-sin(-th)*ymr;
                  y2r=yc-sin(-th)*xj+cos(-th)*ymr;
                  if(x2r>0 && x2r<winWidth && y2r>0 && y2r<winHeight){
                    d3=pow(pow(x2r-x1[i-1],2)+pow(y2r-y1[i-1],2),0.5)+((int)w1[i-
1]*rate);
                    br2=0;
                    for(k=1;k<=num;k++){
                      if(k!=i && k!=j){
                        d4=pow(pow(x2r-x1[k-1],2)+pow(y2r-y1[k-1],2),0.5)+((int)w1[k-
1]*rate);
                        if(d3>d4){
                            br2=1;
                            break;
                        }//if d3>d4
                      }//if k!=i...
                    }//next k
                    if(br2==0){
                      x2rI=(int)(x2r+0.5);
                      y2rI=(int)(y2r+0.5);
                      g.drawLine(x2rI,y2rI,x2rI,y2rI);
                      l = new Line(x2rI,y2rI,x2rI,y2rI);
                      lines.addElement(l);
                    }//if br2==0
                    if(ymr<miny){
                      miny=ymr;
                    }
                    if(ymr>maxy){
                      maxy=ymr;
                    }
                  }//if x2r>0 && ....1950
              }//if a1r>=0 1950
              xj++;
              rwe=0;
              if(x2r<0 || x2r>winWidth || y2r<0 || y2r>winHeight){
                  rwe++;
              }
              if(rwe==1){
                  rr=1;
              }
              if(xj<100){
                  rr=0;
              }
            }//while rr==0
            */
            minyI=-winHeight;//(int)(miny+0.5);
            maxyI=winHeight;//(int)(maxy+0.5);
            for(yjI=minyI;yjI<=maxyI;yjI++){
              // old line was: b1y=4*pow(dist2*be,2)+4*pow(be*yjI,2)-pow(be,4);
              b2y=1/(4*dist2*dist2-be*be);
              b1y=pow(dist2*be,2)+pow(be*yjI,2)-pow(be,4)/4;
              b3y=b1y*b2y;
              if(b3y>=0){
                  // added the be/4 term below.
```

```
                    x0=-(pow(b3y,0.5));
                    // Note that I flipped the signs on the x0 and yjI terms.
                    x10=xc+cos(-th)*x0+sin(-th)*(yjI); // ok,
                    y10=yc+sin(-th)*x0-cos(-th)*(yjI); // these coord transforms check out
                    if(x10>0 && x10<winWidth && y10>0 && y10<winHeight){
                      d5=pow(pow(x10-x1[i-1],2)+pow(y10-y1[i-1],2),0.5)+((int)w1[i-
1]*rate);
                      br3=0;
                      for(k=1;k<=num;k++){
                        if(k!=i && k!=j){
                          d6=pow(pow(x10-x1[k-1],2)+pow(y10-y1[k-1],2),0.5)+((int)w1[k-
1]*rate);
                          if(d5>d6){
                              br3=1;
                              break;
                          }//if d5>d6
                        }//if k!=i
                      }//next k
                      if(br3==0){
                        x10I=(int)(x10+0.5);
                        y10I=(int)(y10+0.5);
                        g.drawLine(x10I,y10I,x10I,y10I);
                        l = new Line(x10I,y10I,x10I,y10I);
                        lines.addElement(l);
                      }//if br3==0
                    }//if x10>0...
                  }//if b3y>=0
                }//next yjI
              }//if dist>be
            }//next jmw
        }//next imw
        drawn = true;
      } else {
        // paint all the lines in lines.
        for (int i = 0; i < lines.size(); i++) {
            l = (Line) lines.get(i);
            g.drawLine(l.x1,l.y1,l.x2,l.y2);
        }

      }

      g.setColor(Color.black);
      g.drawLine(winWidth-200,10,winWidth-100,10);
      g.drawLine(winWidth-200,5,winWidth-200,15);
      g.drawLine(winWidth-150,5,winWidth-150,10);
      g.drawLine(winWidth-100,5,winWidth-100,15);
      g.drawString("0",winWidth-208,13);
      g.drawString("0.5 mm",winWidth-98,13);
    }
}
```

Appendix D: User Survey Results

Repeated below is a survey given to each of 10.467 students involved in the Beta testing of the Polymerlab system. Included are summaries of their responses along with charts showing the distribution of answers for questions that required quantitative answers.

Polymerlab Survey

Please answer the following questions. This survey is intended to help us evaluate and improve the iLab experience. Your answers will remain anonymous.

When applicable, please circle a number from 1 to 7 to indicate whether you agree with the statement. (1 = Strongly Disagree, 2 = Disagree, 3 = Mildly Disagree, 4 = Neutral, 5 = Mildly Agree, 6 = Agree, 7 = Strongly Agree).

Registration

1) The registration process was straightforward and easy to use. 1 2 3 4 5 6 7
[2, 3, 5, 6, 7, 7, 7]

2) Please describe any problems you encountered while registering to use the site.
One student encountered errors stating they had already registered. Another stated the process was difficult to understand.

3) Please list any other comments you have on the registration process.
One student asked for earlier registration availability. Another asked for a shorter time required to be verified in the system.

Reservations

4) The reservation process was straightforward and easy to use. 1 2 3 4 5 6 7

[3, 5, 6, 6, 7, 7, 7]
5) The maximum reservation time is adequate. 1 2 3 4 5 6 7 **[1, 4, 2, 6, 3, 2, 3, 1]**
6) I was able to make scheduled reservations for the times I needed. 1 2 3 4 5 6 7 **[7, 5, 6, 5, 7, 5, 7, 5]**
7) I used all of my reserved time slots. 1 2 3 4 5 6 7 **[7, 6, 6, 5, 7, 4, 7, 7]**
8) The 24/7 availability of the experiment was useful to me. 1 2 3 4 5 6 7 **[7, 7, 7, 6, 7, 6, 7, 6]**

9) For what length of time did you typically use the experiment in a single session?
All said 90 min.

10) Did you typically use all of a scheduled reservation? yes / no
Answers were overwhelmingly yes.

11) What time of day did you typically log into the experiment?
(morning / afternoon / evening / night)
Students were fairly evenly distributed over the time slots, though a surprising number also signed up for times which were their normal lab time.

12) Currently, the maximum reservation time is 90 minutes. Do you think it should be longer or shorter? If so, how long/ short?
All students asked for more time. Most suggest 2 hours or more.

13) How much time was needed to complete the entire experiment? (If your group divided the experiment, please give the estimated total time. Do not include server down time.)
[5, 6, 6, 10, 2, 6, 3-4, 5 hours]

Running the Experiment

14) I found the instructions for the lab clear and easy to use. 1 2 3 4 5 6 7 **[7, 4, 6, 7, 5, 4, 7, 6]**
15) I would like to have had the lab instructions available online as part of the iLab experiment. 1 2 3 4 5 6 7

[1, 5, 4, 6, 7, 4, 5, 3]
16) The demo performed by Prof. Rutledge in the Chem. Eng. PC cluster was helpful to my understanding of how to run the experiment. 1 2 3 4 5 6 7 **[7, 6, 7, 4, 7, 6, 7, 5]**
17) I would have found the capability to have another teammate or TA view the experiment remotely as I ran it useful. 1 2 3 4 5 6 7 **[4, 5, 6, 5, 3, 7, 7, 7]]**

18) How many times did the iLab server "freeze" while you were trying to use it?
[1, 1, 0, 5, 6, 6, 6, 4]

19) From where did you successfully log onto the experiment (Athena cluster, RESNET, off-campus housing, Chem. E. PC cluster, etc.)? (List all that apply.)
Chem. E. PC cluster, Athena. Students repeated these answers.

20) Was there any place where you were unsuccessful in logging onto the experiment? If yes, please describe.
Off campus housing, personal computer. Students repeated these answers.

21) What operating system(s) did you successfully log onto the experiment (Windows NT/2000/XP, Redhat Linux, Sun Unix, etc.)? (List all that apply.)

Sun UNIX, Windows XP. Windows XP. Students repeated these messages.

22) Were there any operating systems on which you tried unsuccessfully to run this experiment? If so, please list them.
Windows 2000. N/A.

23) What browser(s) did you use (Internet Explorer, Mozilla, etc.)? (List all that apply.)
Internet Explorer, Mozilla, Netscape. Students repeated these answers.

24) Were there any browsers on which you tried unsuccessfully to run this

experiment? If so, please list them.
Netscape, Internet Explorer, N/A. Students repeated these answers.

25) How much of the experiment were you able to complete (i.e. how many isothermal crystallizations)?
2 crystallizations. All. All. All. All. All. All.

26) If you experienced any problems or difficulties with the lab instructions or explanations, please describe them.

<blank>. N/A. <blank>. <blank>. None. None. Procedure #8 was briefly unclear for one student.

TA Interaction

27) I would have found it helpful to interact more with the lab TA's to understand and run the experiment. 1 2 3 4 5 6 7 **[4, 4, 3, 4, 4, 4, 5, 3]**
28) I would have found it helpful to interact more with the computer TA (Derik) to understand and run the experiment. 1 2 3 4 5 6 7 **[6, 5, 3, 5, 7, 5, 7, 7]**
29) The lab TA's response time to questions/ issues was adequate. 1 2 3 4 5 6 7 **[7, 4, 5, 4, 5, 4, 5, 3]**
30) The computer TA's response time to questions/ issues was adequate. 1 2 3 4 5 6 7 **[4, 5, 5, 7, 7, 5, 7]**

31) How many times was it necessary for you the contact the lab TA's with questions about the lab procedure?
2-3 times. None. 1 time. None. None. None. 1 time.

32) How many times was it necessary for you the contact the computer TA with questions about the lab server?

10+ times. 4-5 times. 3-4 times. 4 times. 3 times. 1 time. 1 time.

33) Please describe any problems/ issues you had.

Microscope server freezing, unable to login. <blank>. <blank>. <blank>. Microscope freezing. Server down. Server crashed during movement of stage. Time warnings were wrong.

Experiment Interface

34) The user interface for the microscope was easy to use. 1 2 3 4 5 6 7 **[7, 5, 5, 5, 6, 5, 7, 6]**
35) The feedback panel was useful. 1 2 3 4 5 6 7 **[7, 5, 6, 6, 4, 4, 5, 6]**
36) The heat controls/ experiment run controls were easy to use. 1 2 3 4 5 6 7 **[7, 5, 6, 6, 6, 3, 5, 6]**
37) When streaming images, the image refresh rate was adequate. 1 2 3 4 5 6 7 **[7, 6, 6, 7, 7, 5, 6, 6]**
38) The procedure for recording a temperature run was easy to use. 1 2 3 4 5 6 7 **[7, 5, 3, 7, 4, 2, 6, 6]**
39) I would liked to have had a real image of the equipment (microscope, hotstage) to refer to. 1 2 3 4 5 6 7 **[4, 5, 3, 5, 7, 3, 4, 7]**

40) Were there any aspects of the experiment controls that were confusing?
Video recording vs. video stream. No. Student thought should be able to hold a temperature indefinitely. No. No. No.

41) What changes if any would you make to the experiment controls/ user interface?
<blank>. N/A. More colorful. <blank>. None. None. Ability to resize applet.

Data Analysis

42) The procedure for analyzing the data was clear. 1 2 3 4 5 6 7 **[7, 6, 6, 6, 4, 6, 5, 6]**
43) I found the Jimage tool useful. 1 2 3 4 5 6 7 **[7, 7, 6, 1, 6, 1]**

44) I found the JImage tool easy to use. 1 2 3 4 5 6 7

[7, 6, 6, 1, 6, 1]

45) Did you have any problems retrieving saved experiments? If so, please describe.

Student described the process as “tedious and said that he ended up converting the files to bitmap. No. No. No. No. No, but student though procedure led to hurriedly naming files.

46) Did you use the JImage applet to analyze your data? If so, did you experience any problems using the applet?

No. Student tried to use it but crashed. Yes. Yes. No. Yes. Yes.

47) If you did not use the JImage applet, what technique did you use to analyze your data?

Bitmap. Windows Paint. Students repeated these answers.

Science Objectives and Overall Experience

48) The lab has increased my understanding of the properties of polymers and polymer crystallization. 1 2 3 4 5 6 7

[7, 5, 6, 6, 7, 6, 6, 6]

49) I enjoyed using this experiment. 1 2 3 4 5 6 7

[4, 5, 6, 5, 6, 4, 5, 3]

50) I learned as much using this experiment as I would have using a hands-on lab in building 31-068. 1 2 3 4 5 6 7

[7, 4, 7, 4, 7, 3, 3, 5]

52) Please describe any other changes or additions you would make to the lab, and give any other comments you have.

One student stated technical problems and having to share the lab made the lab too time consuming. Another student asked for a live picture of the equipment. One student asked for access to data taken by other members of her group. Confirmation box for deleting data runs.

Appendix E: Reinstallation procedures

E.1 Backing up System

Before any attempt to reinstall components of the Polymerlab system, an important first step should include backing up the current Polymerlab system. There are three main areas that must be backed up. The first two include the Microscope server code and the Framework and Microscope Client code. The Microscope Server code, including all Python code and Perl scripts for launching the server, may be found in the C:\iLab directory. The Microscope Client Java code, as well as the Framework .NET project code and user recorded data, may be found in the C:\Inetpub directory. Both of these directories should be backed up in their entirety.

The third area which must be backed up is the SQL database. While it is theoretically possible to try to recreate the database from a simple copy of the files in various directories, this is not straightforward. It is especially difficult if the reason for the backup is an upgrade to a newer version of the database software. Therefore, there are two steps to backing up the database. The first is to generate a database script that can be used to recreate the format of the database. An example of this script is given in Appendix A. The procedure for generating this script is falirly simple: in Enterprise Manager, right click on the Polymerlab database, and select the option to generate script. Within the option box that appears, select the option to script all objects, including stored procedures, and give the file a name. This file can later be imported to generate the structure of the database. However, when generated this way the database will not be populated with data. To backup the database with all data intact, right click on the Polymerlab database within Enterprise Manager and select backup database. This will generate a file containing both the structure and data from the database. Reinstating the database is as simple then as importing the backup file from within Enterprise Manager.

It is important to note that these two methods are redundant: the full database backup is much more complete than the script generation. However, because the script is stored as a text file it is more easily duplicated, for instance in this thesis.

E.2 Program Requirements

After refomatting, obtaining a new computer, or an unexpected system crash, the next step in reinstalling the Polymerlab system is to reinstall the operating system and off-the-shelf software components. The newest versions of the KS software require Windows XP as the operating system. (Before installing however, the latest versions of each of the components below should be researched and their requirements determined.) In order to provide services over the web, the machine will need IIS and .NET software installed.

- 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/
- Any standard Java IDE, though even a simple text editor may be used.

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 or later.
  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

E.3 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 go to C:\ILab\RemoteMicroscope and do:

  ```
  jar -cvf client\data\PCiLab.jar client/*.class client/images/* com/*
  move client\data\PCiLab.jar C:\inetpub\wwwroot\ TODO
  ```

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

E.4 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.

E.5 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.

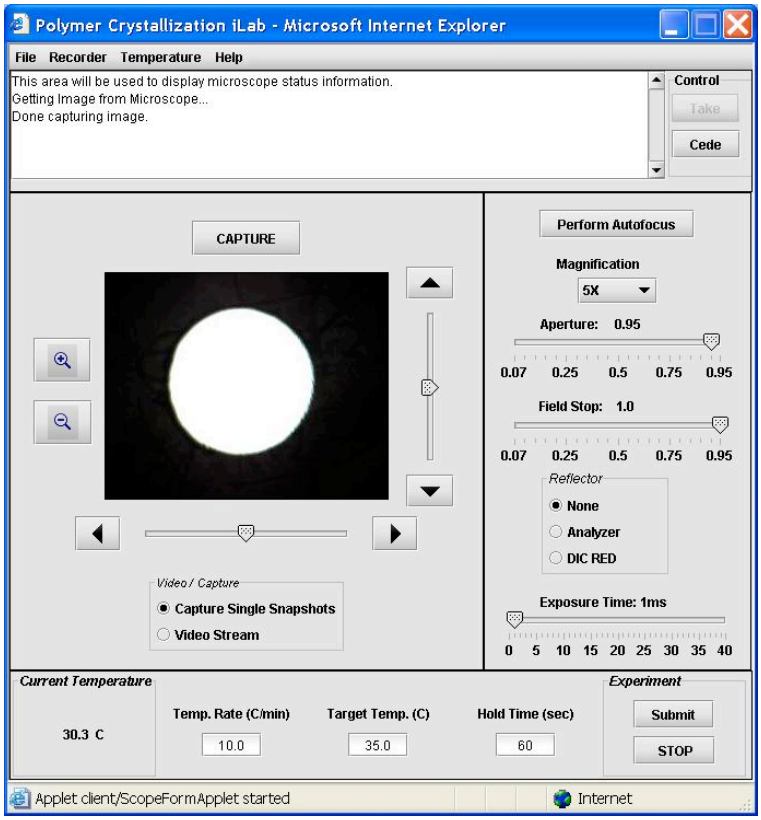
Virtual Directories can be added to expose any directories on the file system to the Web Server. Currently, there is a virtual directory called iLab, accessible through http://polymerlab.mit.edu/iLab. This virtual directory points to the local directory C:\inetpub\wwwilab\. Under this virtual directory, 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 F: Student User Manual

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

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 located to the side 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; a popup dialog window will flag invalid parameters. After the desired parameters have been entered, press the SUBMIT or RUN button at the bottom 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 temperature, 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.

Ceding Control

When a user has control and is not running an experiment, the CEDE button in the upper right hand corner of the applet will be enabled. By pressing this button, other users who are also viewing the experiment will be given the chance to take control. Control is taken on a first come, first serve basis. All users will see a message informing them that control is available, and once another user has taken control another message will be sent to all clients informing them of that fact.

Taking Control

When control of the Microscope becomes available, a message will be displayed in the text area, and the TAKE button will be enabled. The first user to press TAKE gains control of the experiment. When control has been taken, a message will be sent to all clients and displayed in the text message area. The TAKE button, and other control buttons, will become enabled if a user has control of the experiment.

Appendix G: Polymer Sample Preparation

Because air pockets, dirt, and detritus can contaminate the crystal sample, causing crystallization to occur non-randomly, it is important to prepare the sample with extreme care. The first step should include cleaning the round quartz slide using Kimwipes with water and then alcohol. After cleaning and drying the slide so that no residual polymer or debris remains, place it back inside the circular holder in the MDS600 hot stage. (Note that the slide has a preferential orientation in the holder- this can be seen by carefully examining the edge of the slide. It should be returned in the proper orientation, and the sample, of course, should be placed on top.)

The PEO sample should be prepared as a solution, using Polyethylene oxide, molecular weight 100,000. Using a 20 mL scintillation vial, add approximately 1.5 g of PEO. Slowly stir in about 15mL of purified water. Use a stirrer at a medium setting until the PEO is well dissolved. (Previous samples were stirred for approximately 24 hours.)

Draw approximately 2 mL of the solution into a syringe. Add an Acrodysc syringe filter to the tip of the syringe. With the magnification pieces moved to the open slot, to allow access to the quartz slide, reach in and place several large drops on the sample holder. (It will be difficult to force the drops out.) The liquid should cover the entire circular slide. When observing directly though the eyepiece on the Axioplan2, the sample should appear clean.

Using the Polymerlab, make a reservation and open the experiment. (Do not replace the cover to the hot stage yet.) Set the temperature controller to a rate of 30 deg/min, the hold temperature to 100 degrees, and the hold time to 300 seconds. Submit the experiment. Once the temperature begins to reach 100 degrees the water from the solution will begin to evaporate. When drying the sample, do not completely seal the MDS600, as water vapor will condense inside. Simply set the cover atop the instrument to keep dust from settling on the sample. Leave enough room to observe the sample drying.

Continue submitting the experiment until the sample is dry. When it is, slowly lower the temperature to around 52 degrees, and observe the crystals formed. They should appear

and grow relatively quickly at that temperature. They should also appear distinct and sparse enough to allow student to successfully track the growth of several crystals for an extended period of time. They should also appear across the entire sample, not just near the edges, etc.

References

[1] Schultz, J.M. Polymer Crystallization, 129-147. Oxford University Press, 2001.

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

[3] D. Talavera. "*On-Line Laboratory for Remote Polymer Crystalliation Experiments Using Optical Microscopy*", M.Eng. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2003

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

[10] Sedgewick, Robert. Algorithms in C++, 407-411. Addison-Wellesley Co., 1992.

[11] De Berg, Mark et al. Computational Geometry: Algorithms and Applications. 2nd ed. Springer-Verlag, 2000.

[12] iCampus Project Webpage. *http://icampus.mit.edu,* 2004.

[13] iLab Service Broker Webpage. *http://ilab.mit.edu,* 2004.

[14] K. Yehia. "*The iLab Service Broker: a Software Infrastructure Providing Common Service in Suppose of Internet Accessible Laboratories*", M.Eng. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2004.

[15] J. Northridge. "*A Federated Time Distribution System for Online Laboratories*", M.Eng. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2004.