LACOME: Early Evaluation and Further Development of a Multi-User Collaboration System for Shared Large Displays

by

RUSSELL MACKENZIE

B.Sc. (Honours Computer Science), The University of British Columbia, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2010

Abstract

Large screen displays are becoming more prevalent as their prices continue to fall. For these large screens, the paradigm of one-user-per-display begins to break down because there is adequate space and resolution for a variety of simultaneous content. LACOME, the Large Collaborative Meeting Environment, is a set of software tools that allow multiple users to simultaneously publish their personal computer displays onto a large shared display using any VNC server. Once published, windows or even full desktops can be moved, resized, and iconified; optionally, they may even be controlled by other users. LACOME is groupware: multiple users can interact with the shared display simultaneously. Our work focuses on enhancing the stability and usability of the LACOME system, which previously existed only as a rough, proof-of-concept prototype. The LACOME Server was originally developed using C++ on Linux. To simplify administration and provide cross-platform support, the LACOME Server is ported to Java. A new engine provides performance-guaranteed rendering. The secure transmission of VNC passwords is now supported using SSL, instead of relying on default passwords. Interactive widgets such as an IP address display and a 'Return to Desktop' button support usability. The LACOME Client has been converted into a Java™ Web Start application, eliminating the need for installation or configuration by new users. Additionally, LACOME Client settings now persist across usage sessions and a new feature, "Publish Me," automatically determines the user's IP address to initiate a VNC connection. We also develop a new window manipulation technique, the Large Screen Optimized (LSO) technique, which features larger interaction handles. Clicking near the edge of a window 'snaps' the cursor precisely to the edge of that window, allowing a user to quickly place objects against the edges or corners of the screen. The addition of an explicit mode switch to take control of the contents of a window allows us to utilize the entire window area for manipulation and provide a 'zooming resize' feature. An initial user study shows that users preferred and were able to manipulate windows more quickly using the LSO technique than with a more traditional technique.

Preface

The study involving human subjects, described in Chapter 4, was completed with the approval of the UBC Behavioural Research Ethics Board under Ethics Certificate #H09-01534. This work was published as UBC Computer Science Technical Report TR-2010-03 (MacKenzie, Hawkey, Perswain, & Booth, 2010). I, Russell MacKenzie, jointly developed the LSO technique and experiment design with my collaborators Kirstie Hawkey, Presley Perswain, and Kellogg Booth, primarily during group design meetings. I completed the majority of the implementation of the software required to run the experiment on my own, with some coding by Presley Perswain. The study required two experimenters; Presley Perswain assisted me in running the experiment throughout. Finally, Kirstie Hawkey and I jointly performed the data analysis and prepared a manuscript for publication. All other work reported in this thesis was done by me unless otherwise noted.

Table of Contents

List of Tables

List of Figures

Glossary

DNS	Domain Name System
GUI	Graphical User Interface
I/O	Input / Output
IPv4/IPv6	Internet Protocol version 4 / version 6
LACOME	The “Large Collaborative Meeting Environment”
MDE	Multi-Display Environment
PCIe	Peripheral Component Interconnect Express
SDG	Single Display Groupware
VNC	Virtual Network Computing

Acknowledgements

First and foremost, I would like to give my supervisor Dr. Kellogg S. Booth my most sincere thanks for his continued support. Since taking me on as a fledging undergraduate, he has been a veritable fount of ideas and inspiration and an ardent believer in my abilities. He has held me to the highest standards and given me a thorough introduction to the world of Human-Computer Interaction and academic research at large. I would also like to thank Dr. Kirstie Hawkey for her frequent and practical advice, tips, guidelines, proofreading, and much more. I would have missed more than one deadline without her assistance. I thank Michiel van de Panne, my diligent second reader, for his comments.

I would also like to acknowledge the ICICS lab managers, Gable Yeung and Ron Fussell, for their assistance and enthusiasm. My fellow students have been excellent peers, critics and sources of inspiration, including Jen Fernquist, Leah Findlater, Zoltan Foley-Fisher, Idin Karuei, Sebastian Koch, Billy Lam, Joel Lanir, Evgeny Maksakov, Mohan Raj Rajamanickam, and Garth Shoemaker. I would like to single out Presley Perswain for his especial help on the LACOME project; he has been deeply involved in brainstorming, testing, and coding LACOME for as long as I have.

Finally, I would like to thank my wife Fauve for her unwavering love and for supporting me in my decision to pursue graduate studies.

My research was supported by the Natural Sciences and Engineering Research Council of Canada through the Discovery Grant program, the Alexander Graham Bell Canada Graduate Scholarship program, the Strategic Project Grants program, and the Strategic Network Grants program as part of NECTAR, the Network for Effective Collaboration Technologies through Advanced Research. The final stage of the research was partially supported by Graphics, Animation, and New Media (GRAND), a Network of Centres of Excellence and the spiritual successor to NECTAR. Additional support came from the Faculty of Graduate Studies and the Department of Computer Science at the University of British Columbia.

To my wife, Fauve

1 Introduction

Large screen displays are a pervasive feature of modern meeting rooms and workspaces. As prices continue to fall the average size and resolution of these displays is increasing. Many already dwarf typical computer monitors by either metric. Traditionally, meetings have operated in a one-user-per-display paradigm where a single user physically connects his or computer to the large screen display. While this approach works well for some types of meetings, such as presentations with a single presenter, we believe that a more flexible system is required to support a wider variety of collaboration patterns. In particular, current tools offer poor support for meetings with multiple presenters, especially meetings in which the cast of presenters changes in a dynamic and ad hoc manner. To allow different users to use a display, physical re-cabling is usually required. Even relatively advanced solutions such as KVM (keyboard-video-mouse) switches only allow a single user's desktop to be displayed at one time. Merely switching quickly between users is not enough; it is often important to show information from multiple users simultaneously.

LACOME, the "Large Collaborative Meeting Environment," consists of two custom software applications that, when paired with one or more VNC servers, allow multiple users to simultaneously publish their computer desktops to a shared large screen display, and also allow other users to interact with the displayed information on a variety of semantic levels. VNC, which stands for "Virtual Network Computing," is an industry-standard tool for controlling a computer remotely, regardless of operating system. Users of LACOME are free to use the VNC server of their choice. The LACOME Server is responsible for driving the shared display and is capable of connecting to and displaying multiple VNC sessions simultaneously. The LACOME Client is used to initiate connections and to redirect mouse and keyboard input from individual users' computers to the shared display space.

A user may enable “navigation” through a LACOME Client. When navigating, the user receives a cursor that is displayed on the shared screen. The cursor may be used to manipulate windows through such actions as moving, resizing, and iconifying. Each window contains a published computer desktop. A user may take control of a window in order to interact with its contents. While interacting with the shared display, the system cursor on a user’s own machine becomes locked in place to prevent the accidental clicking of other user interface elements in applications running the user’s machine.

Multiple users may interact with the shared display space either by publishing content, by interacting through mouse and keyboard input, or both. LACOME is an example of a Single Display Groupware (SDG) system. However, because each user typically also possesses his or her own computer, complete with a display, the LACOME meeting environment is also a Multi-Display Environment (MDE). Previous research in these two areas is discussed further in Section 1.5.

In the remainder of this chapter, we summarize briefly the contributions of this thesis and give an overview of its structure. We then discuss related work, followed by a background discussion of the LACOME system functionality.

1.1 Contributions

This thesis builds directly upon research completed by Zhangbo “Zephyr” Liu in the course of completing his 2007 M.Sc. thesis, also at the University of British Columbia (Liu, 2007). Liu’s thesis described the implementation of a prototype or “proof-of-concept” version of LACOME. This early version had several limitations: the LACOME Server ran only on Linux, only very simple window manipulations were provided, and the UI framework was not easily extensible. More importantly, both the LACOME Server and the LACOME Client were dogged by a number of usability issues.

The research reported in this thesis addresses each of these deficits. The LACOME Client’s user interface (UI) was modified to increase usability, and the installation process was greatly simplified through the use of Java WebStart. The LACOME Server

was reimplemented in the Java programming language to provide broad cross-platform compatibility. A new and extensible UI framework was implemented for the LACOME Server to allow for a variety of new interaction techniques and for the creation of interactive widgets. As part of this, we designed, implemented, and evaluated an innovative interaction technique termed the LSO, or "Large Screen Optimized," technique for window management. This technique introduces a mode-switch to distinguish between the actions of manipulating windows and interacting with window contents. Having a mode-switch allows the entire area of a window to be used for manipulation, providing much larger interaction handles. The LSO technique also introduces a novel zooming-resize technique and the ability to 'snap' to the edges or corners of a window in order to easily position a window with corner or edge constraints in the display.

A controlled user study was conducted to evaluate the effectiveness of the LSO technique. While the study was limited in scope and could not adequately validate all features of the LSO technique, those aspects that were tested showed consistently positive results. The LSO technique was widely preferred to the more traditional alternative technique used for comparison in the study.

With these improvements, the current version of LACOME offers a complete, robust, easy-to-install and easy-to-use system for publishing multiple computer desktops to a shared display space and for supporting a variety of collaboration patterns using the shared information.

1.2 Guiding Principles

In his thesis, Zhangbo Liu presented a set of guiding design principles that influenced the direction of the LACOME research. These same principles continue to guide the project just as strongly in this iteration, so his words are presented here virtually verbatim from his thesis (Liu, 2007):

Make it simple. We try to minimize changes to existing systems and keep individual systems as loosely coupled as possible so that when LACOME gets updated or modified, other system connecting with it won't be affected.

Make it compatible. Although we were initially motivated by a particular situation with a particular hardware and software configuration, we do not want LACOME to be used exclusively in only one setting. Instead, we want LACOME to be a general purpose system that can be easily deployed in many settings; wherever possible cross-platform approaches are adopted to achieve greater compatibility. The server side should be portable to other systems with minimum changes; the client side is kept light-weight and setup-free to minimize installation problems.

Rely on social conventions. Following the advice given by the iRoom team (Johanson, Hutchins, Winograd, & Stone, 2002), we do not intend to make LACOME "smart" nor have it automatically adapt itself to users. Instead, we leave it to the users themselves to coordinate their collaborative work with LACOME. It is likely that issues of human-human communication as well as human-computer interaction will rise in multi-user collaboration with the system. We provide smooth human-computer interaction in order to facilitate naturally arising social-based human-to-human interactions.

1.3 Overview of the Thesis

This first chapter describes LACOME and situates the current work in terms of previously published research. Additionally, it revisits much of the content published in Zhangbo Liu's 2007 thesis on the early implementation of the LACOME system (Liu, 2007), describes the LACOME system architecture, and introduces a number of crucial terms that will be key to understanding the discussions in subsequent chapters. This chapter is intended to give LACOME users a firm understanding of the operation of the system.

Chapter 2 discusses changes made to the LACOME Client. It is divided into two sections: the first addresses robustness and usability issues while the second describes a

number of new features introduced into the LACOME Client. This chapter contains some low-level implementation details, and so will be of interest primarily to future developers of LACOME. It also contains detailed discussions regarding design decisions as well as usability issues in the original LACOME Client.

Chapter 3 discusses changes made to the LACOME Server. Because most of the LACOME Server software was rewritten from the ground up, this chapter contains many technical, low-level details. The first section describes the software architecture, followed by details of the extensible UI framework. These sections will be of interest primarily for future developers of LACOME. The chapter ends with descriptions of a number of new features in the LACOME Server, including discussions of the underlying problems the features are designed to address. These discussions will be of value to advanced LACOME users, such as system administrators responsible for configuring the system.

Chapter 4 discusses the LSO technique for window manipulation, and should be useful to those interested in window manipulation techniques or interaction design for large screen displays. It also presents a quantitative user study that was conducted to evaluate the overall effectiveness of the LSO technique compared to a more traditional window manipulation technique that was used for comparison. Chapter 4 stands on its own and does not require the reader to have read any of the other chapters in this thesis, including this introductory chapter. It also eschews the low-level technical system details presented in Chapters 2 and 3, and instead focuses on interaction design and the methodology used in the user study.

Chapter 5 offers our reflections on this research and identifies a number of areas requiring future work.

The Certificate of Approval from the UBC Research Ethics Board for the study reported in Chapter 4 is included in Appendix A.

1.4 LACOME Background

In this section we describe the basic operation of LACOME and introduce key terminology. Chapters 2 and 3 discuss in detail the changes between the original version of LACOME and the current version, as well as new features that have been added. Here, we only present the current operation of LACOME at a high-level and we avoid discussing its development history except when necessary.

1.4.1 Overview

The LACOME system uses a client/server architecture. A single LACOME Server runs on a dedicated machine, which is connected to a large screen display. It is either left running at all times or is run when a meeting begins. Installation is intended to be straightforward; the administrator responsible for the server machine simply places the program files in the directory of his or her choice and executes a shell script or batch script. An optional settings file, described in detail in Section 3.3.3, may be created to specify the size of the window, the port on which to listen for LACOME connections, or the default address to be displayed onscreen. Many users are used to launching applications by double-clicking the application's icon. While supporting such a mechanism would be straightforward, doing so is platform specific and it was assumed that users responsible for launching the LACOME Server were likely to be comfortable executing a shell or batch script from the command line. No direct user interactions with the LACOME Server are supported, so once initialized the server machine is typically left untouched.

Each LACOME user is expected to operate a computer during a meeting. In most scenarios this will be the user's own laptop computer, although in some cases, such as in a computer lab or control room, other form factors might be used. The LACOME Client does not need to be installed in the traditional sense; instead, it may simply be run from a webpage using Java WebStart. On the first launch of the LACOME Client, settings files are created in the user's home directory. Should one desire not to use Java WebStart the LACOME Client can also be used in the same manner as the LACOME Server, using

shell scripts or batch scripts to launch it. Some users might be more comfortable double-clicking an icon to launch the application; we expect that these users will find the WebStart mechanism sufficient for their needs.

A sample system configuration diagram is shown in Figure 1.1. In this example, three computer systems are interacting with LACOME. Two of them, each labeled "Publisher," are sharing their desktops on the shared large display. Additionally, two of them, each labeled "Navigator," are interacting with the content on the shared display by redirecting their mouse and keyboard input. Note that a LACOME Client may be both a Publisher and a Navigator simultaneously; more information on these terms is presented below. The various connection types are labelled. Navigators communicate over a LACOME connection, while Publishers communicate over VNC connections. There are two types of VNC connection, regular and reverse: a regular VNC connection is initiated by the VNC Client (LACOME Server), while a reverse VNC connection is initiated by the VNC server. Both regular and reverse VNC connections are shown in the figure, though in practice each VNC server typically uses only one or the other at any given time.

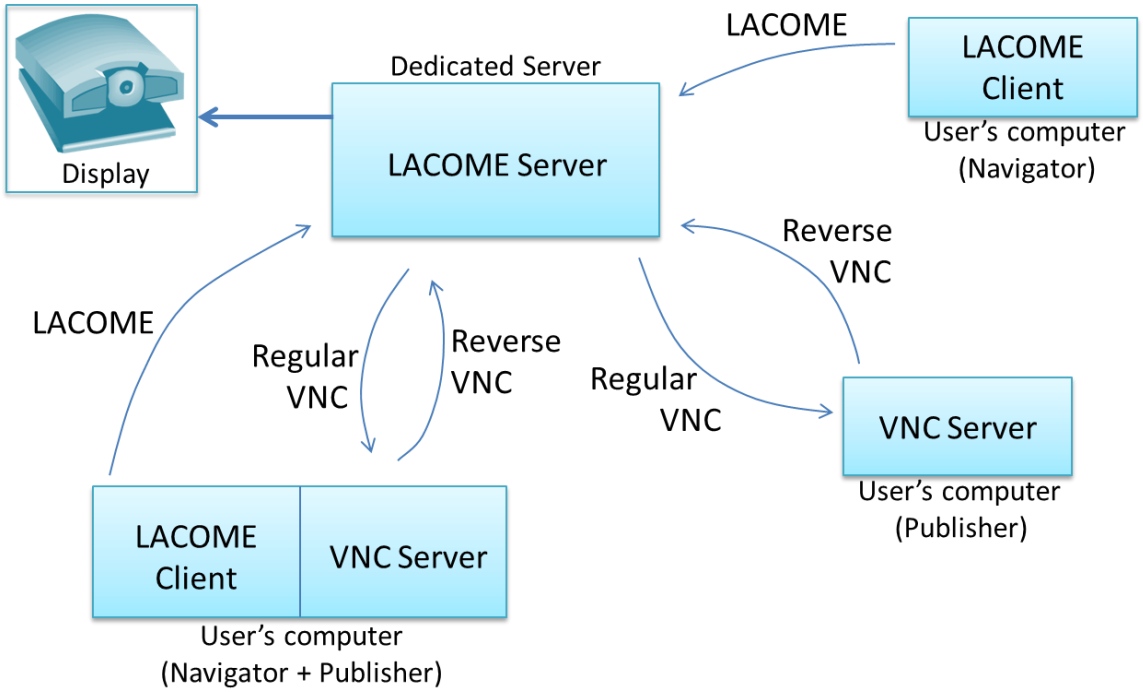

Figure 1.1 A typical LACOME configuration.

1.4.2 Connecting LACOME

To connect a LACOME Client to the LACOME Server, a user enters the address and port number for the LACOME Server into the text fields at the top of the LACOME Client, then presses the "Connect" button located in the top-right corner. An example is shown in Figure 1.2. After the Connect button is pressed the LACOME Client will attempt to connect to the machine `TheLACOMEServer` on port 2001. Immediately upon connection, the user's nickname is sent to the LACOME Server; the nickname may be changed later by pressing the "Send Nick" button. Nicknames are used to identify windows and cursors on the shared display. Once connected, the Connect button's text changes to "Disconnect." A subsequent press will disconnect the LACOME Client from the LACOME Server and the button will revert to "Connect."

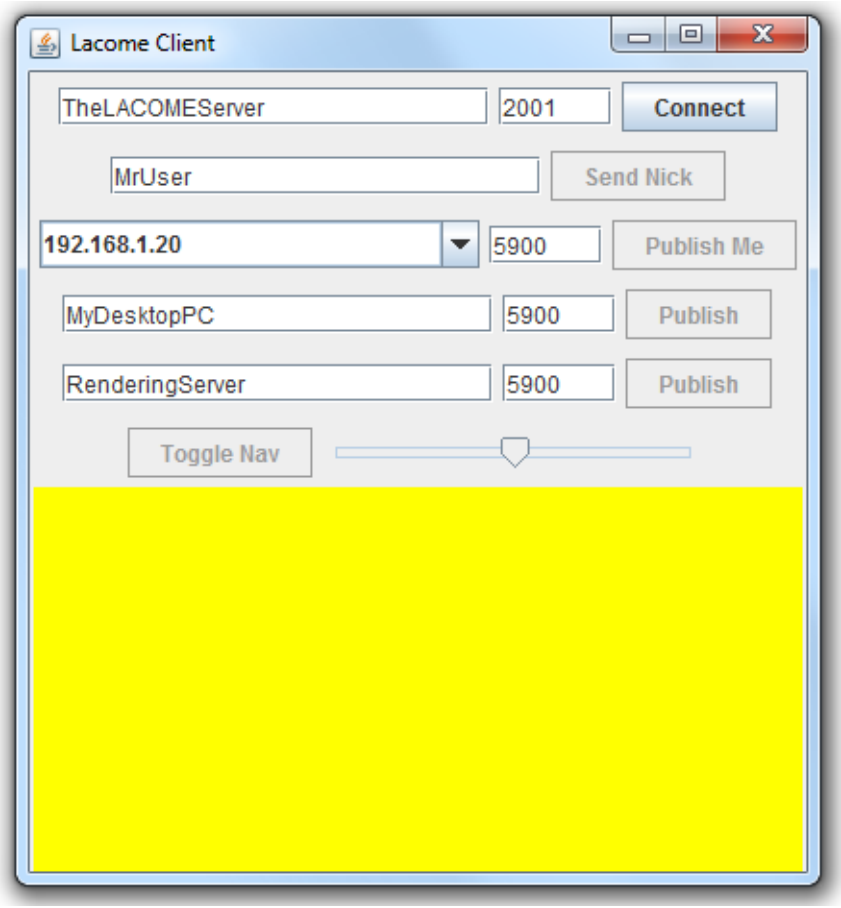

Figure 1.2 The LACOME Client.

1.4.3 Publishing

Users who wish to share their computer desktops on the shared display are required to run a VNC server. Any standard VNC server may be used, and there are several options available for each of the major operating systems. Our terminology for the act of sharing a computer desktop is "publishing," and the shared computer is termed a "Publisher." More information on the user interface for publishing desktops is presented below in Section 1.4.4 and Section 1.4.5.

A single user may publish multiple desktops, which may be on his or her own computer or on any computer that shares network connectivity with the LACOME Server. The LACOME Client that published a VNC session is known as that session's "Owner," though sessions initialized with a reverse connection are ownerless; see Section 1.4.5 for more information on reverse connections. When a LACOME Client disconnects from the LACOME Server, the VNC sessions owned by it are disconnected. This means that should a user leave a meeting, they are unlikely to leave sensitive information available to other, possibly untrusted, users.

Special consideration is given to publishing the user's own computer. The LACOME Client determines all the different IP addresses belonging to the computer and displays them in a drop-down list. The default choice, which is the computer's hostname, is typically sufficient; only in somewhat complex networking scenarios is a different choice required.

It should be noted that the LACOME Server is, in fact, a VNC client that has been modified to allow connections to multiple VNC servers simultaneously. Thus, the notions of "client" and "server" are deeply muddied: the LACOME Server is also a VNC client, and the machine running the LACOME Client also runs a VNC server. For this reason, full names such as "LACOME Server" or "VNC client" will always be used in this document, rather than the short form "server."

Two different methods may be used to publish a desktop: regular VNC connections and reverse VNC connections. For a regular connection, the network socket is initiated by the VNC client; recall that the LACOME Server is the VNC client. The network socket for a reverse connection is initiated by the VNC server, which connects to a listening VNC client. The LACOME Server has a dedicated thread waiting for incoming reverse VNC connections. Whenever it receives a request it accepts the socket connection and responds as a VNC Client.

1.4.4 Regular VNC connections

To initiate a regular VNC connection, the user enters the address and port number for a VNC server into the appropriate text fields, then presses the Publish button. For example, in Figure 1.2 the user is nearly ready to publish the machine `MyDesktopPC` using the VNC server on that machine's port 5900. Note, however, that the Publish button is disabled because the LACOME Client is not yet connected to a LACOME Server. Once connected to a LACOME Server, the Publish button can be pressed and the LACOME Client will prompt the user for the VNC password. It then passes this information along to the LACOME Server, which attempts to initiate a VNC session with the target VNC server. Again, the LACOME Client does not actually create the VNC connection: it merely collects the user input and passes it along to the LACOME Server, which initiates the connection.

1.4.5 Reverse VNC connections

The method for initiating a reverse connection depends on the particular VNC server used. For example, if using the command-line only "x11vnc" VNC server on Linux, one simply uses the "`-connect`" switch:

```
x11vnc -connect TheLACOMEServer
```

Other VNC servers differ. RealVNC on Microsoft Windows offers GUI options for establishing reverse connections. The user first right-clicks on the RealVNC icon in the system tray and chooses to "Connect to listening viewer," as shown in Figure 1.3. The

user then enters the VNC client's address in the dialog box that appears, as shown in Figure 1.4.

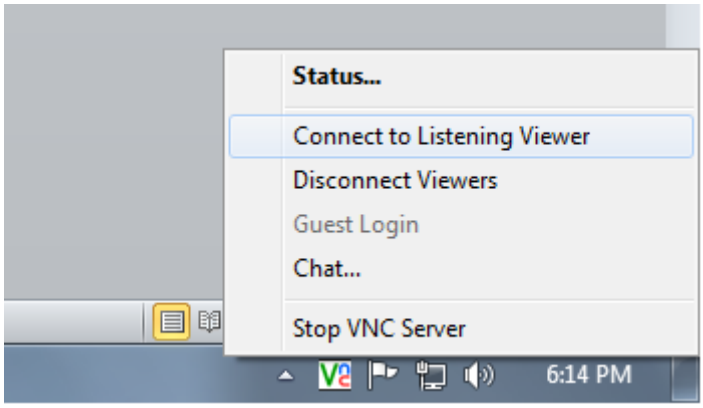

Figure 1.3 The first step for initiating a reverse connection from RealVNC.

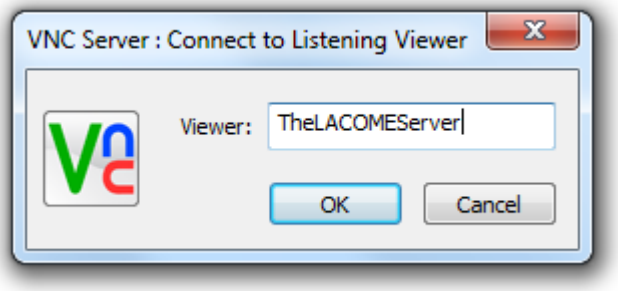

Figure 1.4 The second step for initiating a reverse connection from RealVNC.

1.4.6 Navigator: Input Redirection

Using the LACOME Client, a user may redirect his or her mouse and keyboard input to the shared display by pressing the "Toggle Nav" button, visible in the centre-left of Figure 1.2. When in this mode the user's computer is known as a "navigator," and the system cursor becomes locked to the centre of the navigation pane. The navigation pane is the yellow area at the bottom of Figure 1.2. While navigating, the user receives a cursor on the shared display that may be used for window manipulation, control of VNC sessions, or for annotations. The size of the cursor on the shared display may be changed using the slider bar in the LACOME Client. To cease navigation, the user may either double-click the "Return to Desktop" button at the top-right of the LACOME Server

display, or use the keyboard shortcut Ctrl+Shift+Space. The cursor on the shared display is then released and the system cursor on the navigator is also released.

A navigator may be in one of three states: Manipulator, Controller, or Annotator. Manipulators may move, resize, or iconify windows. Controllers may interact with the contents of a window. The Annotator mode is not implemented, but would allow a user to create sketches or add text to the shared display. These three states are defined in the following three sections.

1.4.7 Manipulator

When a LACOME Client begins navigating, its default state is that of "Manipulator." A manipulator may move, resize, iconify, or deiconify windows, or interact with onscreen widgets. The LSO technique is the default window manipulation technique in LACOME, though others may be implemented straightforwardly using the flexible UI framework. We briefly introduce the LSO technique for window manipulations here; the technique is described in greater depth in Chapter 4. For brevity, the figures presented in Chapter 4 are not reproduced here.

In the LSO technique, a user may interact with the entire surface area of a window because an explicit mode-switch is required to convert the manipulator into a "controller" state, which is discussed in the next section. The left mouse button is used for moving a window, while the right button is used for resizing. The window is divided into nine regions: there are four corner regions, four edge regions, and one centre region. The corner regions are 50x50 pixel rectangles, and the edge regions are 50 pixels wide and run between corner regions. The centre region comprises all of the remaining space. To prevent corner regions from overlapping, windows are constrained to always be larger than 100px tall.

If a user depresses the left mouse button in the centre region and moves the mouse, the point on the window underneath the cursor remains under it as it moves. If the user depresses the left mouse button in one of the edge or corner regions, the cursor is

immediately moved, or "snapped," to the exact edge or corner. From then on, the point on the exact edge or corner remains under the cursor as it moves. This snapping feature allows a user to quickly, easily, and precisely position a window against the exact edge or corner of the display because the cursor is constrained to never leave the display area.

If a user depresses the right mouse button on a snapping region, the cursor is again moved to the exact edge or corner. From then on, that point remains under the cursor as it moves. However, when resizing, there is always some point on the window that does not move. If a corner region was selected, the opposite corner remains stationary. If an edge region was selected, the corresponding point on the edge directly across from the cursor remains stationary. If the centre region was selected, the point underneath the cursor remains stationary, and the window is resized about that point as the mouse is moved up and down. This provides a "zooming-resize" effect, similar to the zooming feature commonly found in mapping software such as Google Maps (Google Inc., 2010) and Bing Maps (Microsoft, 2010e).

If a user double-clicks the left mouse button anywhere on a window that is not currently being controlled or manipulated, the window becomes iconified. First, the window's current size and position are stored. Next, the window is shrunk to a small size and placed in the bottom-left corner of the display. The window continues to receive framebuffer updates while it is iconified. If multiple windows are iconified, they are placed in a line along the bottom edge of the display. All iconified windows are 24px tall and cannot be moved or resized. To deiconify a window, a user must double-click the iconified version of the window. The window is then restored to its original size and position.

1.4.8 Controller

A navigator may take control of a shared desktop. In this state, the navigator is known as a "controller". To interact with the contents of a shared desktop, a second level of input redirection must take place. The first level of indirection is when the LACOME Client captures the user's mouse movements and forwards them to the LACOME Server. The LACOME Server can then forward those movements again to a VNC server to which it is

connected. When a user middle-clicks on a window containing a shared desktop, the Navigator undergoes a mode-switch from being a Manipulator to being a "Controller." When controlling a window, the cursor is constrained to remain within the bounds of that window. Any mouse or keyboard events are forwarded a second time, from the LACOME Server to the target machine. The LACOME Server acts as a multiplexer, taking a user's input and forwarding it to one of many VNC servers. To cease control, the user must press the keyboard shortcut Ctrl+Alt+F1. Only one user may control a window at one time because almost all operating systems support only a single cursor. Additionally, a controlled window may not be manipulated by other users, although this could easily be allowed.

1.4.9 Annotator

In principle, a Navigator could have other modes besides Manipulator or Controller, though none have been implemented at this time. Some mechanism must be provided for switching between modes; the extensible UI framework of the LACOME Server provides a straightforward way to implement a series of buttons or toolbars. One important potential mode identified by Zhangbo Liu in his original LACOME thesis is that of "Annotator." In this mode, a user is able to enter text or create sketches with simple drawing tools. Some annotations could be affixed to particular windows, and follow them as they move, while other annotations could simply exist in the overall shared display space.

1.5 Related Work

By the mid-1970's, personal computers (PCs) such as the Apple I and Commodore PET were available to hobbyists, marking a transition from large, expensive, multi-user mainframe systems to smaller, relatively inexpensive, single-user systems (Press, 1993). Today, personal computing is ubiquitous. In 2009, despite widespread economic downturns and an 11.9% decline in sales compared to 2008, 257 million PCs were shipped worldwide (Gartner, Inc., 2009). However, single-user computing, where every user has a dedicated screen and interacts with applications on that screen, is not always

sufficient. The relatively recent research area of Single Display Groupware (SDG) strives to allow multiple collocated users to interact effectively with a shared display. Our work follows this trend, but combines it with some other emerging themes in personal computing.

The continuing evolution of the personal computer has led to a number of different form factors. In addition to desktop PCs, one currently finds laptops, netbooks, nettops, ultra-mobile PCs, tablet PCs, touchscreen tabletops, and a wide range of handheld devices and smartphones available for sale. Even video game consoles can be used for some tasks traditionally performed on personal computers, such as web browsing or instant messaging (Sony Computer Entertainment America, 2010). This cornucopia of devices means that users often find themselves in environments with multiple displays and computing platforms. The research area of Multi-Display Environments (MDE) explores this theme, and informs our LACOME research.

A number of research projects have used input redirection to allow users to interact with remote computers. While there is some information on this in the discussions of SDG and MDE, further details on input redirection are provided at the end of this section as a precursor to the LACOME input redirection methods.

Window manipulation is a major sub-topic within this thesis. Related work on collaborative meeting software, window manipulation techniques in operating systems, window management in practice, and novel window management techniques in a variety of settings is presented at the beginning of Chapter 4, because it is specific to the LSO technique and the controlled user study detailed there.

Much of the related work in this chapter was first identified by Zhangbo Liu in his thesis (Liu, 2007). Many of his references are presented again here, along with a discussion of new work performed in the intervening time between Liu's thesis and this thesis.

1.5.1 Single Display Groupware

The Single Display Groupware (SDG) model of collocated collaborative work was first formalized by Stewart et al. (Stewart, Bederson, & Druin, 1999). Its basic tenets are a shared user interface, shared feedback, and coupled navigation (if one user navigates to new content all users are affected). A number of earlier systems match this paradigm: the CoLab system from Xerox PARC (Stefik, Foster, Bobrow, Kahn, Lanning, & Suchman, 1987) strove to make meetings more effective through collaborative tools and the LiveBoard system (Elrod, et al., 1992) was intended to allow direct interactivity.

Public plus personal displays

The CoLab system allowed multiple users to interact with a shared database and numerous groupware applications (Stefik, Foster, Bobrow, Kahn, Lanning, & Suchman, 1987). As in LACOME, each user of CoLab interacted with a dedicated personal computer. Because the system predates the era of portable computing, these were part of a fixed installation. It is our belief that using a "personal" personal computer – that is, a PC belonging to the user full-time – will lower barriers to using the system and will provide more fluid access to a user's content. Other systems also used personal displays in addition to a shared display, such as the CaptureLab system (Mantei, 1988), which used desktops, and the Pebbles project (Myers, Stiel, & Gargiulo, 1998), which used PDAs.

In CaptureLab, users interacted with fixed computer terminals. Due to the size and weight of the terminals it was difficult to reconfigure the meeting room environment. Very close attention was paid to the precise layout of the meeting rooms, because it was felt that reduced eye contact could lead to a breakdown of social norms (Mantei, 1988). We have chosen not to address such issues because most users of LACOME have laptops, which allow enormous flexibility for dynamically reconfiguring a meeting space. While we expect that the shared display will be a primary focus during most meetings in which LACOME will be used, we remain agnostic about meeting room design and expect that user groups will form their own usage styles.

Pebbles allowed multiple users to control the mouse and keyboard of a single PC (Myers, Stiel, & Gargiulo, 1998). This caused some difficulties because some users inadvertently moved the single shared cursor when not attempting to do real work, thereby interfering with other people's attempts to work. In LACOME each user controls their own cursor; should they need to interact with a particular shared desktop they temporarily acquire full control of that machine and then release control when they are finished.

Large screen displays

There has been a trend in SDG research towards large screen displays. Bisgaard et al. summarize much of this work in their survey paper (Bisgaard, Heise, & Steffensen, 2009). LACOME assumes that there is enough screen space and resolution for multiple computer desktops to be used simultaneously. If one attempted to use LACOME on a display with only moderate resolution, such as a single 1280px x 1024px projector, many of the system's advantages would be lost. Swaminathan and Sato performed a study using a fairly early large screen display, which was 6'x3' (1.83m x 0.91m) with a resolution of 2400px x 1200px (Swaminathan & Sato, 1997). They determined that "a large display introduces new and qualitatively different issues into interaction design." We found this to be the case. One particular example of this is the thin window borders used by modern operating systems, which were extremely difficult to use on our large screen and led us to design the LSO window manipulation technique described later.

In a paper about the use of large displays in the automotive design industry, Buxton et al. conclude that "the social mores around a conventional workstation differ from those of a drafting table" – a drafting table being a traditional form of non-digital large screen display (Buxton, Fitzmaurice, Balakrishnan, & Kurtenbach, 2000). The transition to large displays "recaptur[es] some of the social and collaborative properties of the design studio that were lost during the first generation of computerization."

To analyze how people use large displays in public spaces, Peltonen et al. developed CityWall, a large public display showing images of Helsinki, Finland, and observed its usage over a period of one month (Peltonen, et al., 2008). The system was designed for

users to simply walk up and use. Peltonen et al. discovered that many users were attracted to begin using the system by observing others using it. While users mostly used the system independently, some were observed to engage in "interaction as performance" and demonstrated a wide variety of actions and user roles. Peltonen et al. conclude that designers should create systems that support "asymmetric and ad hoc role-taking." This research is ongoing; for example, new interaction techniques for the CityWall display were recently presented (Jacucci, et al., 2010).

According to Czerwinski et al., "large displays enable users to create and manage many more windows, as well as to engage in more complex multi-tasking behaviour" (Czerwinski, Robertson, Meyers, Smith, Robbins, & Tan, 2006; Robertson, et al., 2005). A recent study by Bi and Balakrishnan (Bi & Balakrishnan, 2009), in which users switched to using a 16'x6', 6144px x 2034px display for five days, supports this finding. Bi and Balakrishnan conclude that "a large display could benefit multi-window tasks and rich-information applications, enhance users' awareness of peripheral applications, and offer immersive working experiences." In LACOME, "multi-window tasks" could span multiple users' computers, and "peripheral applications" could include other users performing their own work in parallel.

In the LiveBoard system, users use a wireless pen or stylus to sketch freeform drawings or annotate slides (Elrod, et al., 1992). The wireless nature of the input device means that it can be passed freely among different users of the system. A key issue with the system is the accuracy of input; users had trouble with optical parallax because the sensor for the pen was located somewhat behind the screen.

Andrews et al. performed a recent study using a large display, with a resolution of 10,240px x 3200px that curves around the user (Andrews, Endert, & North, 2010). They determined that analysts given such a screen used the "space both as a form of rapid access external memory and as an added semantic layer in which meaning was encoded in spatial relationships."

In systems such as LACOME, users interact with the large screen at a distance, using local input devices. In other systems, such as a design studio environment at Alias Inc., users can interact with large displays through direct manipulation (Khan, Fitzmaurice, Almeida, Burtnyk, & Kurtenbach, 2004). Because of the large size of the displays, and their placement throughout the design studio's physical space, walking from display to display in order to complete direct manipulations is an expensive operation. At Alias, Khan et al. developed the Frisbee system to minimize physical travel between displays, using a local "telescope" view of a distant "target." The telescope view shows a replicated version of the target space, like a "picture-in-picture," and includes controls for remotely adjusting the size and position of the target. Each user may use their own "Frisbee."

Tabletop displays

Research on tabletop displays has seen a surge in popularity in recent years. Tabletops naturally foster communication and collaboration because users typically face each other across the display. However, applications with a strong sense of orientation often cause problems, because some users must view these objects upside down or sideways. Benko et al. performed a survey of tabletop researchers and developers that identified key obstacles to the mainstream adoption of tabletops, as well as input and ergonomic challenges to their use (Benko, Morris, Brush, & Wilson, 2009). Kruger et al. determined that window orientation plays several key roles when collaborating on tabletop displays, including comprehension, coordination, and communication (Kruger, Carpendale, Scott, & Greenberg, 2003). Because computer desktops are strongly oriented, we have chosen to avoid issues of orientation in LACOME by only using vertically mounted displays and vertically oriented windows, preserving the original desktop orientation.

Touch displays

Some Single Display Groupware screens are touch-sensitive. One important example is the DiamondTouch, which is a tabletop display that uses capacitively-coupled electric fields to isolate touch points per-user, for up to four simultaneous users (Dietz & Leigh,

2001). The DiamondTouch system has been used in a large number of research projects (Piper, O'Brien, Morris, & Winograd, 2006; Shen, Vernier, Forlines, & Ringel, 2004; Tse, Greenberg, & Shen, 2006). Other important technologies include frustrated total internal reflection, used by Perceptive Pixel (Han, 2005; Han, 2006) and Smart Technologies (Morrison, 2009); camera based systems, such as the Microsoft Surface (Hilliges, Izadi, Wilson, Hodges, Garcia-Mendoza, & Butz, 2009); and infrared sensing, used in the ThinSight system (Izadi, Hodges, Butler, Rrustemi, & Buxton, 2007).

LACOME does not currently support touch or multi-touch systems. It does not support input on the LACOME Server at all, relying instead on input redirection via the LACOME Client. However, because much of the user interaction in LACOME involves window manipulation, direct touch support could be beneficial. One can imagine a presenter standing near a large display: rather than redirecting input through her computer or asking an audience member to manipulate a window, she could simply and unobtrusively perform that action herself through direct manipulation on the large screen.

Mobile devices

The CoSearch system (Amershi & Morris, 2008) was designed to support collaborative web searching in locations with constrained hardware resources, such as libraries, schools, and developing regions. Users cluster around a single display and each user has a mouse for personal use; clicking links in a search result adds that page to a queue, colour-coded per user. Thus, pages of interest to each user can be visited in turn, without getting left by the wayside. While all of the users of CoSearch actively engage with exactly the same UI elements to serialize a set of shared activities, users of LACOME often use particular portions of the display individually for disjoint, parallel activity. In addition to mouse-driven interaction, CoSearch also supports the usage of mobile phones as input devices. Using the “Mouse mode” on the mobile phone allows users to mimic the functionality of the mice that would normally be used. Additionally, users may “Get Tabs” from the shared display, for personal browsing on the mobile phone.

1.5.2 Multi-Display Environments

In the previous section, several systems were introduced that might more properly be referred to as Multiple Display Groupware (MDG). These systems paired a large shared display with smaller personal displays. A wide variety of such systems exist, including the Augmented Surface, PointRight (part of the iRoom), i-LAND, ARIS, NiCE and IMPROMPTU. Also, Ritchie Argue developed a system that supports a diverse range of multiple-display topologies for Mac OS X (Argue, 2007).

The Augmented Surface project seeks to help users bridge the gap between portable/personal computers and pre-installed/public computers, projectors, digital whiteboards and other such devices (Rekimoto & Saitoh, 1999). Using vision-based tracking, users can drag-and-drop data between their own computer, other displays in the environment, and physical objects, using a technique they dub "hyper-dragging." LACOME similarly seeks to allow users to bring together content from multiple personal computers.

PointRight is a system that allows a single mouse and keyboard to provide input to a large number of disparate computing systems (Johanson, Hutchins, Winograd, & Stone, 2002). It is used in the Stanford iRoom, an interactive workspace that contains several large displays, digital whiteboards, and tabletop displays. Like the LACOME Client, the PointRight application captures user input and forwards it over the network. However, whereas the LACOME Server receives such input and uses it to drive multiple cursors simultaneously within a single system window on a large display, PointRight creates cursors that "jump" between displays, which may even be running different operating systems. The system supports only one cursor per underlying computer system at any given time. LACOME only allows a single cursor to control an underlying system, but it allows multiple cursors to work in close proximity on a single workspace that contains several computer desktops.

i-LAND extends the notion of "display" in "Multiple Display Environment" (Streitz, et al., 1999). Streitz et al. assert that future workspaces will be dynamic, flexible, and

support the ad hoc formation of teams. Rather than allowing only traditional computer displays, the i-LAND system seeks to incorporate the architectural environment and introduces the concept of "roomware," or computer-augmented objects such as doors and furniture.

ARIS is an "interactive space window manager" (Biehl & Bailey, 2004). It provides an iconic map of the workspace with which users can visually relocate supported applications among various screens in the workspace. When these screens are driven by different underlying machines, the system attempts to launch an equivalent program on the new machine and restore application state. In LACOME, applications are shown on displays attached to computers other than the computer on which the applications are running, but because of VNC, the application still runs on the original computer. In ARIS, it is possible that a target machine might have no applications capable of handling a relocated piece of information. This is not an issue for LACOME because the application continues to run uninterrupted on the original machine.

Other work has treated multiple displays as belonging to a single cohesive workspace, rather than separate private/public spaces. For example, Perspective Cursor warps the cursor as it travels from display to display so that it appears to be the same size and at the same orientation to the user, avoiding geometric distortions due to the placement of displays in a room (Nacenta, et al., 2007; Nacenta, Sallam, Champoux, Subramanian, & Gutwin, 2006).

The NiCE discussion room allows a seamless transition between traditional, paper-based interaction while seated at a table and interaction at a large digital whiteboard (Haller, et al., 2010). A special ballpoint pen simultaneously records pen strokes to the system's computer via Bluetooth while actually inking paper (Anoto Group, 2010). Drawings may thus may be presented digitally in real time, or shared afterwards.

IMPROMPTU allows users to replicate application windows to a shared display (Biehl, Baker, Bailey, Tan, Inkpen, & Czerwinski, 2008). Each window can be either *shared* or

shown, depending on whether other users may interact with the contents of the window. Future work to include such window-specific features in LACOME is discussed in Section 5.1.3. IMPROMPTU underwent one of the first field studies of a MDE system. Biehl et al. determined that the system was useful overall and was most used for facilitating opportunistic collaboration.

Plaue and Stasko evaluated the use of multiple shared displays in a laboratory study of groups collaborating on a data-intensive, sense-making task (Plaue & Stasko, 2009). They examined the impact of the presence of and location of shared displays, and determined that users were significantly more satisfied with side-by-side dual monitors than with a single shared display.

Argue's thesis presents a system for Mac OS X that supports a much wider range of display topologies than is typically available in major operating systems (Argue, 2007). In addition to the typical "mirror" and "extended desktop" modes, the system allows displays to logically overlap, be contained within other displays, or be spatially separated. It also attempts to remove physical connectivity limitations and allow multi-user connectivity to shared displays.

1.5.3 Multi-Cursor Window Management

When multiple users collaborate using computing systems, they must either share an input device or use multiple devices. While other researchers have previously studied turn-taking protocols (Inkpen, McGrenere, Booth, & Klawe, 1997), we focus on multi-cursor window management because LACOME supports multiple input devices.

A number of studies have examined techniques for using multiple cursors. A major problem identified by most researchers is that the vast majority of extant software supports only a single cursor. Stewart et al. note that "many applications store user state information in global variables leading to shared interface state such as a single pen colour or font" (Stewart, Bederson, & Druin, 1999). They instead chose to represent all application functionality as tools, using a "local tool" metaphor. "Local tools" use

"separate icons that lie on the data surface along with user data." Confounding this problem, modern operating systems typically only support a single cursor at the window-manager level (Hutterer & Thomas, 2007).

In CoSearch, users share one keyboard but each has a personal mouse (Amershi & Morris, 2008). Search results and queued browser tabs in CoSearch are colour-coded per user; Amershi & Morris determined that participants found colour coded cursors useful and that they were "one of the 'best things' about CoSearch." Other researchers have noted problems using colour-coding to differentiate between users, because some tools are themselves colour-coded (Myers, Stiel, & Gargiulo, 1998). For example, there would be an ambiguity whether a red pencil would draw a red line, or instead simply belong to the "red" user. In Mischief, Morajevi et al. differentiate users' cursors by using the letters of the alphabet with an arrow attached, pointing in one of the eight cardinal directions (Moraveji, Inkpen, Cutrell, & Balakrishnan, 2009; Moraveji, Lindgren, & Pea, 2009). The use of different arrows on different cursors reduces clumping when multiple users point at the same object.

As noted previously, PointRight only allows a single cursor per machine due to the fact that the underlying systems only accept one cursor (Johanson, Hutchins, Winograd, & Stone, 2002). In PointRight, each display is typically driven by a different PC. If multiple users attempt to control one system, their inputs are blended together. The system simply queues all incoming input events so they are executed in an interleaved manner. Similarly, the Pebbles system for using PDAs to control a shared display simply averages the inputs from each user (Myers, Stiel, & Gargiulo, 1998). This presented a particular problem because users were prone to idly moving the cursor even when not attempting to do meaningful work. Heimerl et al. recently presented a system that adds a "metamouse," which averages the position of each user's mouse (Heimer, Ramachandran, Pal, Brewer, & Parikh, 2009). Clicking is only allowed in the Metamouse system when all the cursors are near each other; that is, when all users have agreed on the correct location to make

progress. It is not clear what type of feedback is given to users of Metamouse, as only preliminary work has been published.

Wallace et al. developed a window manager for use in nuclear fusion research control rooms that allows users to simultaneously interact with multiple legacy applications (Wallace, Bi, Li, & Anshus, 2004). They use a time-sharing approach; events from the virtual cursors cause the system to quickly switch window focus to that cursor's window and re-dispatch the event using the system cursor. Because the regular system cursor is what generates the events that applications receive, legacy applications can be used unchanged. Like LACOME, their system uses per-user coloured cursors, and gives windows coloured borders that match the cursors' colours, in order to denote window focus.

The MPX (Multi-pointer X) system provides perhaps the best support for multiple cursors available today (Hutterer & Thomas, 2007). It provides support for up to 255 independent mouse cursors, and different users may interact with any number of applications simultaneously. The system provides "floor control" on a per-window basis, so legacy applications might allow only a single cursor at one time while SDG-enabled applications may allow an arbitrary number of cursors. A special application, "DeviceShuffler," supports input redirection from any computer on the network. The main drawback is that the system supports X Windows exclusively. In contrast, LACOME is fully cross-platform, both for the LACOME Server and for the LACOME Client.

Multi-pointer technology in applications does now seem to be gaining ground. This is perhaps driven by the increasing availability of multi-touch devices such as the Apple iPhone and iPad. Based on the Mischief system (Moraveji, Inkpen, Cutrell, & Balakrishnan, 2009; Moraveji, Lindgren, & Pea, 2009), which is a system for classroom interaction using large numbers of mice, Microsoft has released the Windows Multipoint Mouse SDK; it is targeted for educational uses (Microsoft, 2010b). Another product, Microsoft Multipoint Server (Microsoft, 2010c), allows a desktop PC with multiple mice

and keyboards to give each user a standalone experience. Instead of Single Display Groupware, Windows Multipoint Server essentially offers what might be described as "Multiple Display Individualware."

LACOME currently supports only a very limited range of tools. It will be crucial to employ techniques such as the "local tools" metaphor as new functionality is added. LACOME does not rely on workarounds such as time-sharing the system cursor; in fact, the system cursor is not used at all. All cursors are virtual and contained within the LACOME window, which typically occupies the entire screen. However, each published desktop within LACOME supports only one cursor, which is a limitation imposed by VNC; once a user begins controlling a desktop other users are prevented from interacting with that window.

1.5.4 Input Redirection

LACOME relies heavily on input redirection: using their own mice and keyboards, users are able to interact with content on a shared display. Many other projects, including several already mentioned (Johanson, Hutchins, Winograd, & Stone, 2002; Myers, Stiel, & Gargiulo, 1998; Biehl & Bailey, 2004), also use input redirection. LACOME provides this using a client-server relationship. Users run the LACOME Client on their own computers; it captures their mouse and keyboard and forwards it to the LACOME Server.

VNC, or "Virtual Network Computing", was developed at Olivetti Research Laboratory to allow users to remotely control their PCs (Richardson, Stafford-Fraser, Wood, & Hopper, 1998). A VNC client sends cursor and keyboard events to a VNC server, and the VNC server sends image updates back to the VNC client. The LACOME Server is, in part, a modified version of the TightVNC client. It is able to connect to multiple VNC servers at once. This portion of LACOME is responsible for the second stage of the input redirection process, from the LACOME Server to a published desktop. The first stage of input redirection in LACOME, from the user's machine to the LACOME Server, is achieved through the LACOME Client. The LACOME Server is in the middle, and functions much like a multiplexer or switching network.

Some systems, such as the Integrated Tabletop (Nakashima, Machida, Kiyokawa, & Takemura, 2005) and Mighty Mouse (Booth, Fisher, Lin, & Argue, 2002), use an extended VNC client for input redirection. Others instead use custom applications, such as PointRight (Johanson, Hutchins, Winograd, & Stone, 2002) and Pebbles (Myers, Stiel, & Gargiulo, 1998). WinCuts, a program that allows users to annex display space in order to reconfigure user interfaces, uses a custom application called Visitor; user input is redirected to the shared screen when the cursor is dragged off the edge of the user's personal display, and multiple users are supported based on a turn-taking protocol (Tan, Meyers, & Czerwinski, 2004).

Swordfish is a tool that lets users dynamically bind displays in a MDE together (Ha, Inkpen, Wallace, & Ziola, 2006). When a user drags the cursor to the edge of his or her screen the system begins broadcasting user input events as UDP packets over the network.

All of these systems use relative cursor motions. The developers of Pebbles (Myers, Stiel, & Gargiulo, 1998) attempted to use absolute coordinates, but the resolution of the input device was much lower than the output device, making it impossible to achieve fine-grained positioning. The LACOME Client currently supports only relative positioning, although the LACOME Server also accepts absolute positions. This topic is discussed further in Section 3.2.

1.5.5 Summary

In this section we briefly summarize the related work described above, in a tabular format. Whenever possible, we use a short system name (e.g. LACOME) chosen by the designer of each system; in the few cases where system names are not provided, we created a short title enclosed in quotation marks. For brevity, only the first author's last name is provided. Full references are available in the bibliography. Additionally, the year of publication and the location of the system or the primary research lab are provided, as an aid for memory recall. The table then identifies which of the following aspects a system utilizes or analyses: shared displays, used by multiple users; multiple display

environments; input redirection techniques; touch screens or direct input devices such as styli; and large screen displays.

Table 1.1 Tabular summary of related work.

System Name	First Author	Year	Location	Shared Display	Multiple Displays	Input Redirection	Touch/ Direct Manipulation	Large Screen Displays
CoLab	Stefik	1987	Xerox PARC	●	●	●		●
CaptureLab	Mantei	1988	U. of Toronto	●	●	●		●
Pebbles	Myers	1998	CMU	●		●	●	●
CityWall	Peltonen	2008	Helsinki, Finland	●			●	●
"Daily Work"	Bi	2009	U. of Toronto					●
LiveBoard	Elrod	1992	Xerox PARC	●			●	●
"Analyst's Workstation"	Andrews	2010	Virginia Tech		●			●
Frisbee	Khan	2004	Alias, Inc.	●	●	●	●	●
CoSearch	Amershi	2008	U. Washington / Microsft	●		●		
Augmented Surface	Rekimoto	1999	Sony / Keio U., Japan	●	●		●	
PointRight	Johanson	2002	Stanford	●	●	●		●
i-LAND	Streitz	1999	GMD/IPSI, Germany	●	●	●	●	●
ARIS	Biehl	2004	U. of Illinois	●	●	●	●	●

System Name	First Author	Year	Location	Shared Display	Multiple Displays	Input Redirection	Touch/ Direct Manipulation	Large Screen Displays
Perspective Cursor	Nacenta	2006	U. of Saskatchewan	●	●			●
NiCE	Haller	2010	Upper Austria University	●	●		●	●
IMPROMPTU	Biehl	2008	Microsoft, Redmond, WA	●	●	●		●
"Multi-cursor X Windows"	Wallace	2004	Princeton	●	●	●		●
Multipointer X	Hutterer	2007	U. of South Australia	●		●		
Integrated Tabletop	Nakashima	2005	Osaka University	●	●		●	
Mighty Mouse	Booth	2002	U. of British Columbia	●	●	●		●
WinCuts	Tan	2004	Microsoft, Redmond, WA	●	●	●		●
Swordfish	Ha	2006	Dalhousie	●	●	●		

2 The New and Improved LACOME Client

The LACOME Client is a relatively small piece of software that runs on each user's computer, captures user input, and relays it to the LACOME Server. The bulk of the software is located in approximately a half-dozen Java classes, and is composed of about 2000 lines of code. This chapter describes the continued development of the LACOME Client and several new features that have been implemented.

2.1 Robustness and Usability

The original LACOME Client (Liu, 2007) was functional, but it had a number of problems and usability issues that needed to be addressed. It was very easy to lose mouse focus when simultaneously moving the mouse and clicking. The original LACOME Client did not support IPv6, and the entry of ports to correctly connect VNC Servers was not mandatory. Some key data, such as the user's nickname and selected cursor size, were not sent upon connecting to a LACOME Server. Additionally, some UI components, such as the "Toggle Nav" button that switched mouse and keyboard input to the shared screen, were still active even when not connected. Finally, the UI did not change appearance based on the connection state.

2.1.1 Loss of Mouse Focus

The most troublesome problem that had to be addressed was that it was very easy to cause the original LACOME Client to lose mouse focus when in Navigation mode. In this mode, mouse movements and button clicks are redirected from the user's own computer to the LACOME Server running on the shared screen. To accomplish this, the cursor is automatically placed in the centre of the LACOME Client's mouse panel, which is a large yellow region at the bottom of the LACOME Client window. When the mouse is moved the LACOME Client sends a message to the LACOME Server. In the original LACOME Client, the cursor moves back to the center of the mouse panel just when it is

about to exit the mouse panel. This mechanism did not always work; if the cursor happened to move far enough to be outside of the LACOME Client altogether and then be clicked, the operating system reassigned mouse focus to a different program. In this case, the LACOME Client stopped receiving mouse events. This is a particularly damaging failure mode because the user's attention is almost certainly directed on the shared screen and not on his or her own display; a common reaction by LACOME users when the cursor stops moving is to continue waving the mouse about and clicking wildly for a few seconds while continuing to look at the shared screen, in an attempt to determine if there is a hardware problem or system lag. By the time a user realizes that mouse focus has been lost, any number of unintended consequences could have occurred in other applications running simultaneously on the user's desktop. The cursor promptly does snap back to the mouse panel when it is again placed atop the LACOME Client, but by then the user's attention has been diverted from the primary task and it has to be re-focused again on the shared screen. To reduce the frequency with which this problem occurs, the cursor repositioning scheme was modified so that the cursor is moved to the centre of the mouse panel after every mouse movement, rather than only on an exit event. This significantly reduces the risk of losing mouse focus because the cursor must now move far enough in a single system-level mouse event to escape the LACOME Client from the centre of the mouse panel. By increasing the size of the mouse panel the probability of this happening can be reduced further, but at the expense of screen real estate. While there is currently no system-independent way to implement a transparent mouse panel, there is hope that this feature may be available in the upcoming Java 7 release. Ideally, the user's entire display would be covered with a transparent panel and the cursor hidden, in order to prevent any confusion about where mouse focus lies.

2.1.2 Accept IPv6 Addresses and Make Port Entry Explicit

To mirror how the entry fields for entering the address of the LACOME Server are displayed, the address fields for VNC servers have been separated into address and port fields. We believe this makes the need for port definition more explicit and may serve to remind the user that proper port selection is important, which could aid in the

troubleshooting process if a non-default port is used and it has not been entered properly. However, network addresses that include port information may still be entered inadvertently into the address text field, particularly if a user has copy/pasted the address from another source such as a webpage, email, or instant message.

Rather than simply fail when a port is specified in the address field, the LACOME Client detects this state and handles it appropriately. If the port is the same in both the address field and the port field, it is simply stripped from the address field. If the ports are different, the dialog shown in Figure 2.1 is presented. Whichever port is selected in the dialog box is placed into the port field, and the port is stripped from the address field.

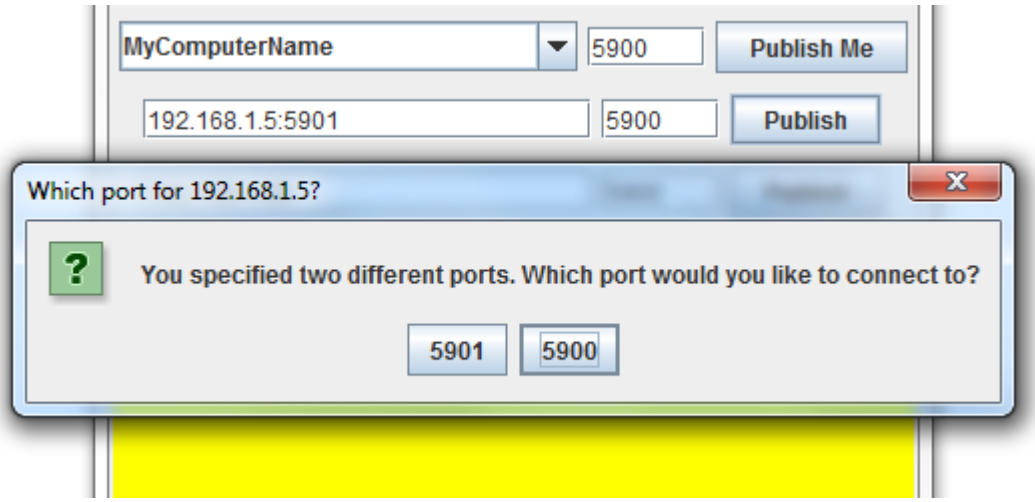

Figure 2.1 The dialog box text shown when two different ports have been specified.

The new LACOME Client is fully compatible with IPv6. In particular, the above discussion on the specification of ports includes ports in IPv6 addresses. In IPv4, the octets in a numeric IP address or the labels in a host name are separated with periods; a single colon separates the address from the port. In IPv6, each 16-bit group is separated by a single colon. A double colon represents a run of groups with the value 0 (there may only be one double colon in an address). For example, `::1`, the local loopback address, is an equivalent notation to `0000:0000:0000:0000:0000:0000:0000:0001`. Additionally, the last 32 bits may use the IPv4 "dotted-quad" (also known as "dot-decimal") notation; for example, one could specify the address

`::ffff:192.168.0.1`, which is an IPv6 "IPv4 mapped address" corresponding to `192.168.0.1`. To specify a port in an IPv6 address, the address is wrapped in square brackets and a single colon followed by the port is added to the end. For example, the address `[::1]:5900` might refer to a VNC server running on the default port, 5900, on the local host.

2.1.3 Send All Relevant Data When Connecting to a LACOME Server

When a LACOME Client first connects to a LACOME Server it is termed an *empty client*, because it is not associated with any VNC publishers. Such a LACOME Client is not intended to be completely devoid of information; it should still contain basic data such as the socket on which it is connected, the user's cursor size and cursor colour, and the user's nickname. However, in the original LACOME Client two key pieces of data, the user's nickname and cursor size, were not transmitted, either as part of the original connection request or immediately afterwards. Instead, these were only updated through explicit user action, requiring either the 'Send Nick' button to be pressed or the 'Cursor Size' slider to be dragged. With a blank nickname, it is difficult to know to whom a window belongs on the shared screen, requiring out-of-band communication between users and placing increased demands on users' memories. If the cursor size is not sent when a connection is initiated, it is possible that the LACOME Client and Server could be inconsistent; the Server would simply choose a default cursor size with no regard for what position the user's slider bar was in. To address these issues, these two data values are now sent immediately after a connection is established.

2.1.4 Disable Components When Not Connected

The missing data problem just described was compounded by the fact that all user interface (UI) elements in the original LACOME Client were *enabled* at all times. In the Java Abstract Window Toolkit (AWT) and Swing toolkit, an enabled component is capable of responding to user input and generating events. Disabled components are typically rendered in a faded gray color and do not receive input or generate events. The original LACOME Client is shown in Figure 2.2.

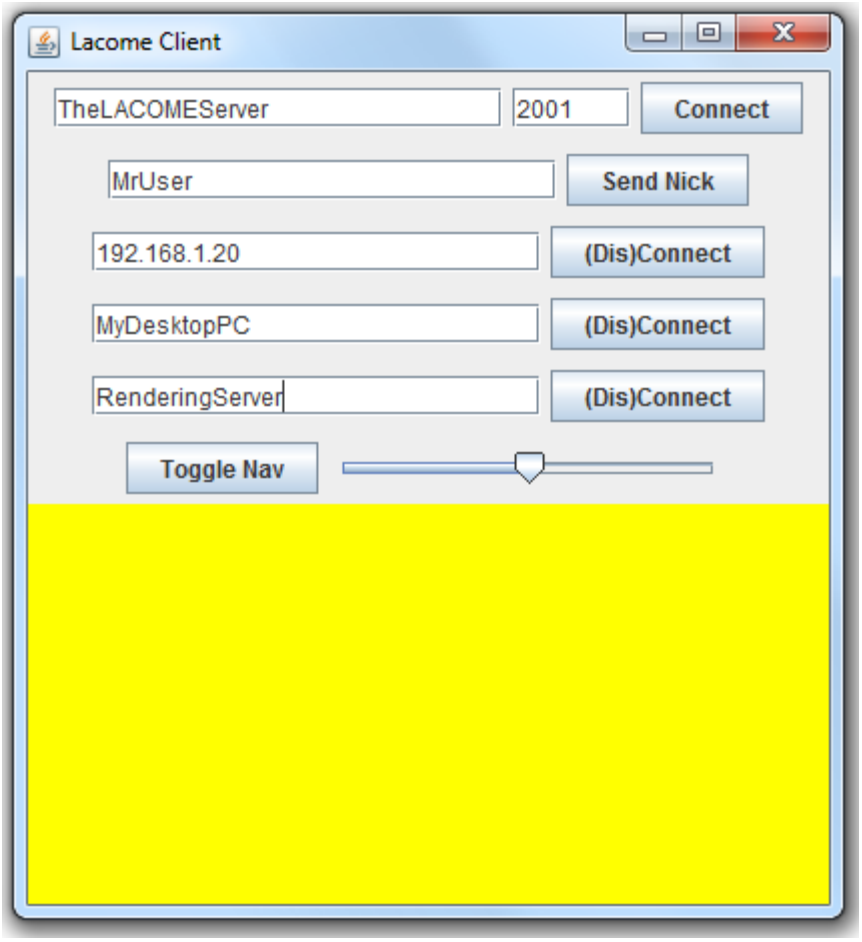

Figure 2.2 The original LACOME Client with all UI elements enabled at all times.

When not connected to a LACOME Server, a user could interact with UI elements such as the Cursor Size slider without realizing that it was not having any effect. Similarly, one could press the Toggle Nav button and begin input redirection despite the fact that there was no destination to which the input could be forwarded. To prevent such confusing behaviours from occurring, the UI was altered to disable certain components when not connected to a LACOME Server, as shown in Figure 2.3.

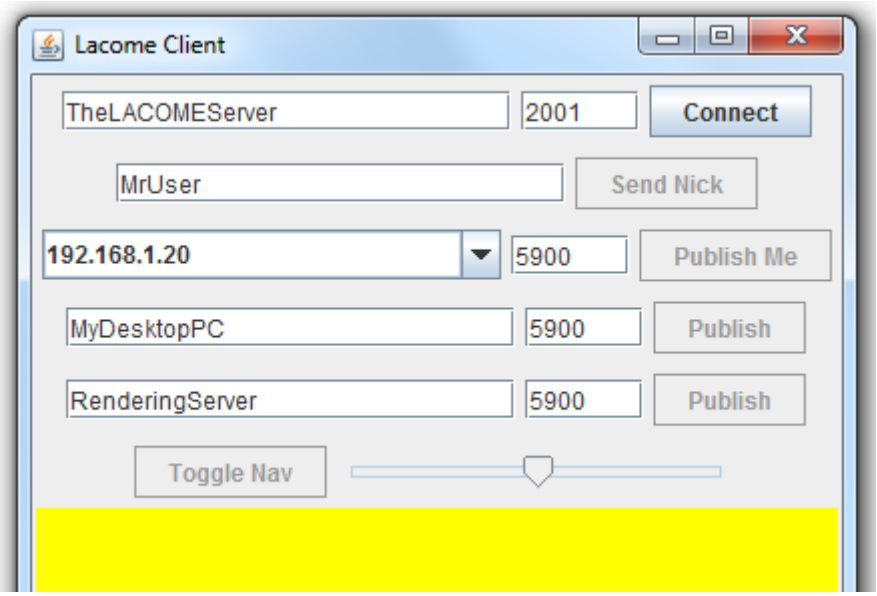

Figure 2.3 The new LACOME Client with certain UI elements disabled when not connected to a LACOME Server.

In particular, those components that normally trigger messages to be sent to a LACOME Server are disabled, while editable fields that do not trigger immediate messages are not disabled. This allows a user to modify the settings for a VNC publisher he or she intends to publish at a later time, even when not connected to a LACOME Server. Input redirection is disabled in this mode, avoiding confusing mouse 'lock-ups'. The subdued "greyscale" rendering of the disabled buttons along the right-hand edge serves to draw attention upward to the still-enabled 'Connect' button at the top-right.

2.1.5 Utilize Connection Information

Additional usability issues arose from the fact that the LACOME Client was not aware of, did not track, and did not utilize connection information of various kinds. This "statelessness" caused a number of problems. As mentioned previously, the "Toggle Nav" button was enabled even when not connected to a LACOME Server. If the user had connected to a LACOME Server previously during the usage session, enabling navigation caused a flurry of error messages related to socket closure; if no LACOME Server had been connected to, the program crashed with a `NullPointerException`.

The same result occurred whether the user had intentionally disconnected from the LACOME Server, the network connection had been severed, or the LACOME Server had crashed. In the current version of the LACOME Client, problematic buttons are disabled when not connected to a LACOME Server. While the original LACOME Client did attempt to internally track whether a VNC publisher was connected or not, this information was not fully utilized. As depicted in Figure 2.2, the button for publishing a VNC server in the original LACOME Client was labelled "(Dis)Connect," indicating both "Connect" and "Disconnect." This has been changed to read "Publish" when the VNC server is not connected and "Unpublish" when the VNC server is connected, thereby giving a clearer indication of the action to be taken based on the current state. Finally, the original LACOME Server did not inform a LACOME Client when one of that LACOME Client's published VNC servers disconnected. This caused a problem wherein the LACOME Client had imperfect information and naively attempted to disconnect a VNC server that was not connected. In the new version of the LACOME Server, when a VNC server disconnects the LACOME Server notifies that VNC server's LACOME Client owner, if one exists. The LACOME Client then reverts the "Unpublish" button back to its default "Publish" state.

2.2 New Features

Several new features were added to the LACOME Client, intended to make the software easier to install and use. Users are now prompted for VNC passwords, and entered data, including these passwords, is now retained between usage sessions. The LACOME Client can be launched from the LACOME website using Java Web Start, which is now fully supported on many platforms. Finally, the Publish Me feature automatically determines the user's IP address.

2.2.1 Prompt for Passwords

Previously, the LACOME Client was never aware of VNC passwords; instead, the LACOME Server contained a default, hardcoded password to which all VNC servers had

to be set. This was both awkward and potentially a source of security problems. A new dialog box, shown in Figure 2.4, is now presented when the Publish button is pressed.

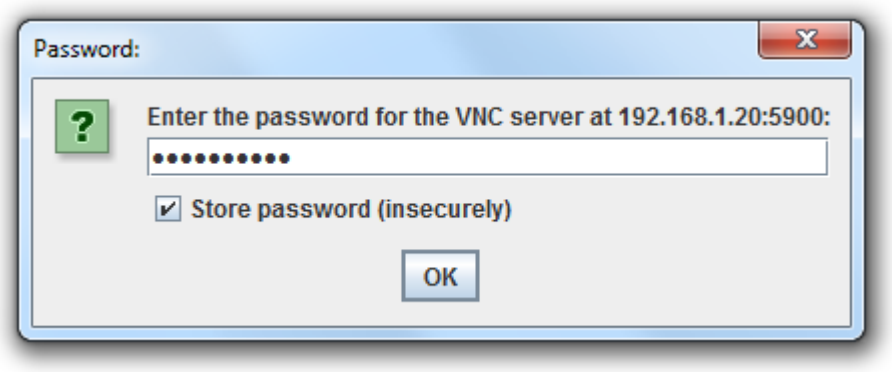

Figure 2.4 The password dialog box in the LACOME Client.

If the Store Password checkbox is left unchecked, any previously stored passwords are removed. After the password is forwarded to the LACOME Server the character array in which it is stored is filled with default values, in order to reduce the time window during which a memory-sniffing attack could occur. (Morana, 2009). If the Store Password checkbox is checked, the password continues to reside in memory after being sent to the LACOME Server instead of being over-written, and is stored on disk when the program terminates.

2.2.2 Persist User Data Between Usage Sessions

The original LACOME Client did not store any data persistently, requiring the user to re-enter data, such as the addresses of the desired LACOME Server and VNC servers and the user's nickname, during each usage session. Because many users will interact with only one LACOME Server and will always use the same nickname, this need for data entry is redundant. In the current version of the software, this type of data is saved when the LACOME Client quits. To minimize the amount of configuration required, the path to which the settings files are saved is not user-configurable. Through its System Properties feature, Java provides a platform-independent way of querying for the user's home directory. Using this, two settings files are created within a folder named "`.lacome`" in the user's home directory. Users are expected to have write permissions within their own

home directory, so it should normally be possible to create this directory. Java 6 and earlier versions provide no direct method for creating hidden folders. On Linux and Mac OS X, folders that begin with a period are hidden. Because there is no access to the "hidden" state of a folder the LACOME settings directory remains unhidden on Windows, although the upcoming Java 7 release provides additional I/O functionality including access to metadata such as the 'hidden' state for Windows folders. The two settings files are named, straightforwardly, "`lacomeSettings.dat`" and "`lacomePasswords.dat`". The settings file contains newline-separated plain text; the file format is described in Table 2.1. Note that although the number of sets of VNC server UI components is specified at compile-time, the settings file supports an arbitrary number of VNC Servers.

Table 2.1 Format for LACOME Client settings file.

Line Number	Contents
1	LACOME Server address
2	LACOME Server port
3	Nickname
4	Publish Me address
5	Publish Me port
6	*n*, the number of VNC servers
6 + *i*	i^{th} VNC server address
6 + *n* + *i*)	i^{th} VNC server port
6 + 2**n* + 1	Cursor Size, from 0 to 256

The contents of the passwords file are encrypted before they are saved to disk, using the "PBEWithMD5AndDES" algorithm (Oracle, 2010b). Internally, passwords are stored in a `Map<String, char[]>`, which maps from a VNC server address string to the password for that VNC server, stored in a character array. Using the Java Serialization API, this data structure is written to a byte array prior to encryption. The use of an encryption scheme prevents a casual user from opening the passwords file and seeing the

lists of passwords within as plain text, but it will not stop a sophisticated intruder. Most problematic is the fact that the encryption key is hardcoded directly into the source code, a violation of the CERT Secure Coding Standards (CERT, 2010). By disassembling the "`.class`" file, which contains the compiled code, an attacker could learn the encryption key and use it to recover the passwords. Encryption keys should instead be stored in a file residing in a secure directory, which was deemed to require too much development effort, or they should be requested from the user for each use, which was deemed too intrusive. A second attack vector lies in the fact that the passwords are loaded as the LACOME Client initializes, and remain in memory until the program is closed. An attacker could analyze a memory dump and locate the passwords. A better solution is to read in the passwords as late as possible and then overwrite them with meaningless values as soon as possible, to provide the minimum time interval for the attack to take place (Morana, 2009). Given that the encryption key is vulnerable, this was deemed to be a relatively small additional risk.

2.2.3 Java Web Start

Java programs, such as the LACOME Client, are typically run at the command line. For the standalone version of the LACOME Client, a pair of shell scripts is provided for Linux and Mac OS X and batch scripts are provided for Windows. One script builds the program from source, if the Java Software Development Kit (SDK) is installed and the Java SDK's '`bin/`' directory has been added to the system's 'Path' environment variable; the other script simply runs the program assuming it has already been compiled. No installation is necessary; the settings and password files will be created the first time the program is run. A user may simply unzip the LACOME Client and run one of the scripts.

While the above installation process is not overly complex, an even simpler approach is available. Java Web Start allows users to launch Java applications from the Internet (or other network) using a web browser. The user need simply click a link to a Java Network Launching Protocol (JNLP) file on a webpage, which the Java Virtual Machine (JVM)

uses to download the corresponding JAR file and in turn launch the application. Unlike Java applets, Web Start applications are not embedded into web pages and they are not subject to the same security restrictions as applets. To maintain security, all Web Start applications must be digitally signed. If the signing certificate is not trusted, a user will be presented with a dialog box similar to the one shown in Figure 2.5 when he or she attempts to launch the LACOME Client using Web Start.

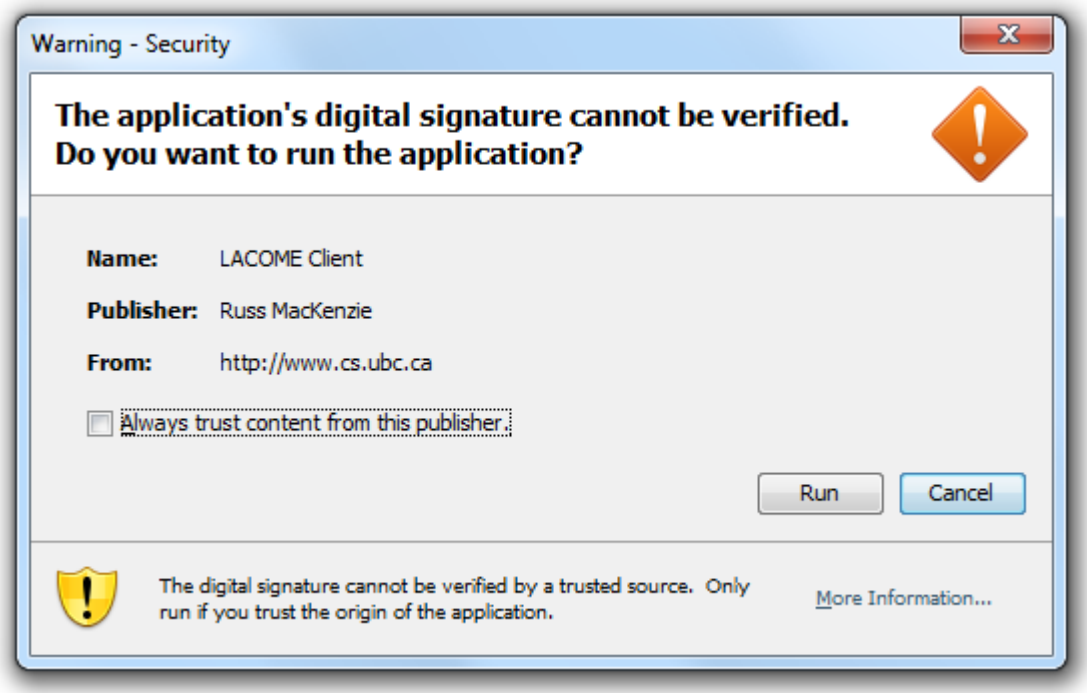

Figure 2.5 The security warning for an untrusted Web Start application.

If the certificate is trusted, or if the user has previously chosen to "Always trust content from this publisher," this dialog is not shown and the application launches immediately. This is as easy as launching a regular application but with the added benefit of always running the most up-to-date version automatically.

The LACOME Client has been tested over Web Start using Windows XP, Windows 7, Mac OS X 10.4 and 10.5.8, and Ubuntu Linux. Tested web browsers include Internet Explorer 8, Mozilla Firefox 3.6, Google Chrome 5, and Safari. Table 2.2, presented at the end of this section, lists the platform/browser combinations that were tested. As well, the

LACOME Client only requires the Java SE 5.0 Runtime (also known as the Java SE 1.5 Runtime). This is important because Apple computers only gained support for Java 6 in Mac OS X 10.5.2, and even then only on Intel x64 processors (Apple Inc., 2008). Computers with PowerPC or Intel x86 processors, as well as machines running older versions of the operating system, still only support Java 5.0.

The results of compatibility testing, performed in July 2010, are shown in Table 2.2. Three key features were tested in each VNC Server: whole-desktop sharing, individual window or partial desktop sharing, and reverse connections. While all VNC servers support whole desktop sharing, only a subset of them support the other two features. Some VNC servers, such as TightVNC Version 1.3, support sharing an individual window rather than the entire desktop; alternatively, a user may specify that only a rectangular subsection of the screen be shared. Additionally, some VNC servers do not support reverse connections, where the socket connection is initiated by the VNC server rather than by the VNC client. On two VNC servers for Linux, Linuxvnc 0.9.7 and TurboVNC 0.6, support for reverse connections was claimed in the documentation but the feature appeared to be unusable: we were unable to successfully use this feature during testing on these two servers, even when attempting to establish reverse connections to a variety of VNC clients.

A result of "Yes," which is paired with a green background, denotes that the feature was both present and was functional during testing with LACOME. A result of "n/a," with a yellow background, denotes that the feature is not supported by that VNC server. Finally, a result of "Error" with a red background denotes that the feature was unable to be tested because it could not be successfully used even with other standard VNC clients. Notably, no server/feature combinations were found to be incompatible with LACOME. That is, in every case where a server and feature worked with other standard VNC clients that server and feature also worked with the LACOME Server, as intended.

Table 2.2 Results of testing various VNC servers.

OS	VNC Server	Version	Whole Desktop Sharing	Window / Partial Desktop Sharing	Reverse Connection
Microsoft Windows	RealVNC	4.1.3	Yes	n/a	Yes
	TightVNC	1.3	Yes	Yes	Yes
	TightVNC	2.0	Yes	n/a	Yes
	TurboVNC	0.6	Yes	Yes	Yes
	UltraVNC	1.0.8.2	Yes	n/a	Yes
Mac OS X (10.5.8)	Remote Desktop	-	Yes	n/a	n/a
	VINE	3.0	Yes	n/a	Yes
Linux (Ubuntu 10.04)	RealVNC (vnc4server)	4.1.1	Yes	n/a	Yes
	TightVNC (Xvnc)	1.3	Yes	n/a	Yes
	TurboVNC	0.6	Yes	n/a	Error
	Krfb (KDE VNC server)	4.4.2	Yes	n/a	n/a
	Linuxvnc	0.9.7	Yes	n/a	Error
	X11vnc	0.9.9	Yes	Yes	Yes
	VINO (GNOME VNC Server)	2.28.2	Yes	n/a	n/a

2.2.4 Publish Me

The Publish Me feature allows a user to easily initiate a connection from the LACOME Server to a VNC server running on his or her desktop. When started, the LACOME

Client determines the hostname and various IP addresses associated with the computer on which it is run. These addresses are listed in a drop-down box, as shown in Figure 2.6. Note that the port field, located immediately to the right of the drop-down box, is populated with the value “5900” by default. This is the default port for VNC servers.

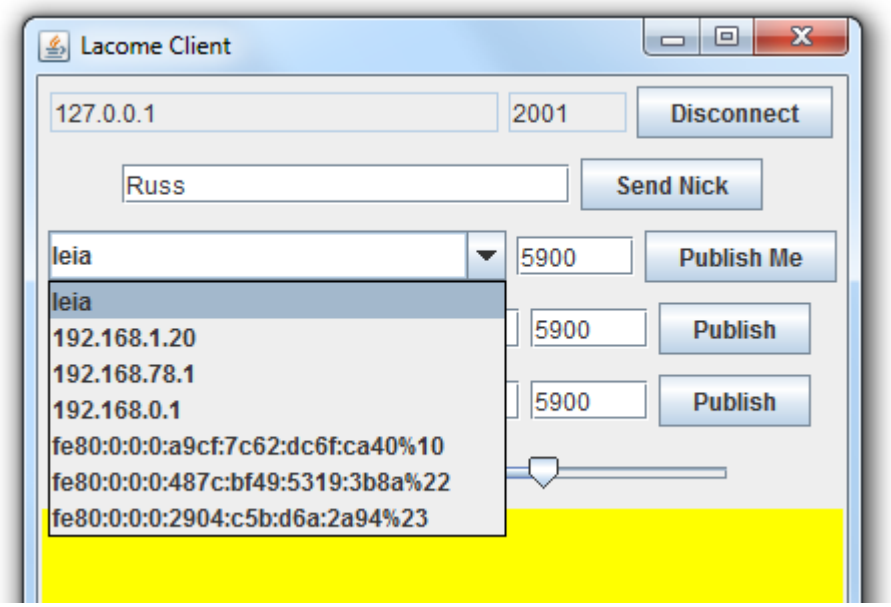

Figure 2.6 LACOME Client with multiple network interfaces.

The addresses come from two sources. The first entry, and the one that is selected by default on the first run of the program, is the fully qualified domain name for the computer’s IP address. In most cases, this address is the correct choice and is also the most meaningful name to a user. According to the Java API specification, this is determined using a “best effort method,” although we have not yet found a way to cause the method to fail (Oracle, 2010a). The remaining addresses are determined by the system and are always numeric IP address as opposed to domain names requiring DNS resolution. A single computer may have multiple network interfaces and be attached to multiple networks simultaneously. For example, the machine shown in Figure 2.6 has three network interfaces, and each interface has both an IPv4 and IPv6 address. To successfully publish a VNC server, a user must know which address will be visible to the LACOME Server. Most users are expected to only attach to one network at a time, or to

be sufficiently advanced to know which address belongs to which network interface. Typically, the fully qualified domain name is sufficient; otherwise, a small amount of trial-and-error may be required to determine the correct choice.

In all other respects, the Publish Me components operate in the same manner as the other publishing components. The password dialog discussed in 2.2.1 is presented when the "Publish Me" button is clicked. Once connected, the text changes to "Unpublish Me," and the LACOME Client is properly notified when the VNC server disconnects from the LACOME Server. The selected address persists when the program exits and is reselected when the program next starts again. Should that address no longer be valid, the fully qualified domain name is chosen instead.

3 The New and Improved LACOME Server

The LACOME Server is responsible for driving the shared screen, viewable by all parties involved in a meeting. Its primary purpose is to connect to and display multiple VNC sessions simultaneously; it also supports input direction via multiple LACOME Clients to allow window manipulations to be performed on the VNC sessions by any or even all of the meeting participants, simultaneously.

A number of improvements have been made to the original LACOME Server. Most significantly, it has been completely rewritten from the ground up, using Java, to achieve platform independence. Additionally, a new rendering engine provides guaranteed rendering performance and ensures mouse movements remain as smooth as possible even when many VNC sessions are being displayed. An extensible UI framework was implemented to provide the necessary infrastructure required for the experiment reported in Chapter 4; the experiment tested a new window manipulation technique which was implemented for the LACOME Server. Several other new features were added such as an IP Address display, a “Return to Desktop” button, the use of a configuration file rather than hardcoded values for various parameters, and IPv6 support.

The new LACOME Server is a medium-sized software project and consists of approximately 6000 lines of Java code located in about 50 source code files.

3.1 Reimplementation in Java

The reimplementation of the LACOME Server in Java was undertaken for a number of reasons. First and foremost was the desire for platform independence, and the desire to simplify the setup and administration process. Second, the existing codebase was complex and in need of simplification. Finally, a major overhaul offered the opportunity

to implement a new rendering engine that is more efficient and that offers guaranteed rendering performance.

3.1.1 Platform Independence

The original LACOME Server was written in C++, and relied on POSIX Threads (pthreads), a threading library primarily available in Linux systems. It was possible to run the program on Windows using Cygwin, a Linux API emulation layer (Cygwin, 2010); however, this greatly increased the complexity of installation. Additionally, there appeared to be performance issues using Cygwin that were not present when the software was run natively on Linux. We were unable to get the original LACOME Server code to run on Mac OS X despite considerable effort.

According to research performed by NetApplications.com, which uses web browser usage statistics as a proxy for gathering data on operating system usage, Microsoft Windows operating systems held 91.46% of the market, while Mac held 5.16% and Linux held only 1.07% (NetApplications.com, 2010). The bulk of the remainder was attributed to mobile phones and a small fraction to internet-enabled gaming devices such as the Nintendo Wii and Sony PlayStation. Such estimates show that a primarily Linux-based LACOME Server may be difficult for many users to accommodate.

The use of the Java programming language ensures platform independence; a few native library files[1] are required for each platform, but these are small enough to distribute with the main program. Should the LACOME Server use Java Web Start in the future, signed and secure versions of these native library files will be automatically selected and downloaded behind the scenes from JogAmp.org, the maintainers of JOGL.

[1] Required libraries are JOGL (JSR-231, Java Bindings for OpenGL), and its associated tool GlueGen.

3.1.2 LACOME Server Architecture

A key feature of the new LACOME Server is its updated rendering engine and new system architecture. The program uses a highly modified version of the Java-based TightVNC client. This section begins with a discussion of the overall system architecture followed by more details of the rendering portion.

The LACOME Server uses a number of different threads, responsible for different jobs. These threads communicate through a small number of shared data objects. There are two types of service to which the LACOME Server must be connected: VNC servers and LACOME Clients. For each of these services, a simple "manager" thread is created that listens for and initializes new connections. A summary diagram of the architecture is presented at the end of this section.

VNC manager

For VNC servers, the manager thread is the `VNCManager` class. The `VNCManager` does double duty as it both creates outgoing connections, which are "regular" VNC connections, and accepts incoming connections, which are "reverse" VNC connections. Reverse connections are typically used to circumvent firewall restrictions that prevent a user's computer from accepting incoming connections to its VNC server; because a reverse connection is an outgoing connection (from the perspective of the machine running the VNC server) it is typically allowed by most firewalls. For more detail, refer to Chapter 1. The `VNCManager` thread accepts incoming connections. Outgoing connections are simply made via a call to the `AddVNC()` method by whichever LACOME Server thread seeks to create a new connection. The other threads that may create a VNC connection are all LACOME Client worker threads, discussed below. Once a connection has been established, either by regular or reverse connection, a `VNCClient` object is created and stored in `vncs`, which is a list of `VNCClient` objects stored within the `VNCManager`. Once the VNCClient is created, the VNCManager returns to listening for more incoming connections. The LACOME Server can connect to many VNC clients simultaneously.

Each VNCClient object represents a connection to a VNC server. Roughly speaking, it is divided into two halves. The first half, embodied by the VNCCanvas class, interprets the messages received from the VNC server and produces drawing requests, while the second half, known as the VNCDisplay class, performs the actual drawing and maintains the display window within LACOME. Communication between the two halves occurs through the VNCConsumer interface, which is implemented by VNCDisplay. The principal reason for the separation of these two halves is that actual rendering can only take place when there is an active OpenGL context. Because of this, rendering can only take place within the single rendering thread, mentioned below, and not within the VNCCanvas. Individual work units of rendering are queued by the VNCDisplay and rendered at an appropriate time by the rendering thread. Further information on the rendering algorithm is presented in Section 3.1.3.

LACOME Client manager

The LacomeClientManager thread waits for incoming connections from a LACOME Client. Unlike the VNCManager, no outgoing connections are ever initiated by the LacomeClientManager. When a connection is received, a LacomeClient object is created and stored in lacomeClients, which is, as one might expect, a list of LacomeClients. A LacomeClient object represents a worker thread (on the LACOME Server machine) that services a LACOME Client (on a user's machine). This is perhaps a nonstandard naming scheme, and it is inconsistent with that used for the VNCClient class described above; however, if we had named these objects LacomeServers they could be confused with the larger LACOME Server program, of which they are only a part. Once a LacomeClient object has been created, the LacomeClientManager returns to listening for new connections. There may be many LACOME Clients connected simultaneously.

A LacomeClient object holds a fairly small amount of data: the position, size and colour of a user's cursor on the shared display, and the user's connection information and nickname. The nickname is used primarily to display ownership of windows. A user may

request a new VNC connection through their `LacomeClient`. The `LacomeClient` processes the connection request and passes it along to the `VNCManager`. Currently, the `LacomeClient` object does not directly track the `VNCClient`s that it has published; instead, each `VNCClient` keeps track of the `LacomeClient` that published it in its `owner` field.

Shared state

As discussed above, program state is shared between the various program threads through several shared data objects. Synchronization of access to these objects is achieved through intrinsic locks, which are a core part of the Java language. In Java, every object is associated with an intrinsic lock. To provide mutually exclusive access to an object, a thread first acquires that object's lock; when finished, the thread releases the lock. A request for a lock is a blocking call; the requesting thread will pause until it successfully acquires the lock. The shared data structures are `vncs` in `VNCManager`, `lacomeClients` in `LacomeClientManager`, and `mainRenderables` and `overlayRenderables` in `Renderer`. Additionally, each `VncCanvas`-`VNCDisplay` pair shares state through a queue named `helpers`, which contains individual rendering work units. Proper concurrent behaviour for `helpers` is achieved through the use of a `ConcurrentLinkedQueue`, although the size of the queue is tracked through a separate `helpersSize` variable that is synchronized on the `helpersLock` object, because determining the size of the `helpers` queue would otherwise be an `O(n)` operation due to the implementation within the Java Collections API.

Miscellaneous other classes

Three other classes deserve a brief mention: `Main`, `Display`, and `InputHandler`. `Main` is a small class that contains the `main()` method, which is where the program begins. `Main` is responsible for initializing the display and spawning the manager threads, `VNCManager` and `LacomeClientManager`. Additionally, `Main` responds

to window events; if the user attempts to close the display window `Main` terminates the program.

`Display` is responsible for setting up and keeping track of the window frame and the OpenGL canvas, which together constitute the display window. It should be noted that `Display` is distinct from `VNCDisplay`, which performs the OpenGL drawing commands for a single VNC session. The `Display` object is created as the program initializes, and is connected to the `Renderer`. From this point on, the `Display` invokes a series of callback methods in the `Renderer` that instruct it to initialize or refresh the display, or inform it that the size of the OpenGL canvas has changed.

The `InputHandler` listens to mouse and keyboard events generated by the display window. Because users typically interact with the LACOME Server over a network, very little local input is allowed. If a user inputs the letter 'Q', for "Quit," the program terminates. If the user inputs the backquote character, '`', the window decorations are toggled on or off. The window decorations include the window's border, title-bar, and the minimize/restore/exit buttons. By maximizing the LACOME Server window, then toggling off the window decorations, a user can make the server occupy the full screen. As an implementation detail of JOGL, removing window decorations causes the window `Frame` and its OpenGL context to be destroyed; this in turn destroys any OpenGL textures that may have been created. After toggling window decorations, the `Display` object tells the `Renderer` to reinitialize itself; the chief consequence of reinitialization is that all textures must be reloaded into OpenGL.

Summary

A graphical summary of the above discussion is presented in Figure 3.1. Note that the `VNCManager`, `LacomeClientManager`, `Renderer`, and `Main` classes all follow the "singleton" design pattern. A label preceded by a plus symbol ("+"), such as the "+ vncs" within the `VNCManager` class, denotes a public field. The arrows between classes

represent containment or composition; the multiplicities of the relationships are shown beside the arrowheads.

Looking at Figure 3.1, we can see that the `VNCManager` class contains a list of `VNCClients` named `vncs`. Each `VNCClient` is comprised of two classes, a `VNCCanvas` and a `VNCDisplay`. The `LacomeClientManager` class contains a list of `LacomeClients` named `lacomeClients`. The `Renderer` class contains two lists of `Renderable` objects, named `mainRenderables` and `overlayRenderables`. Finally, the `Main` class contains a single `Display` object and a single `InputHandler`.

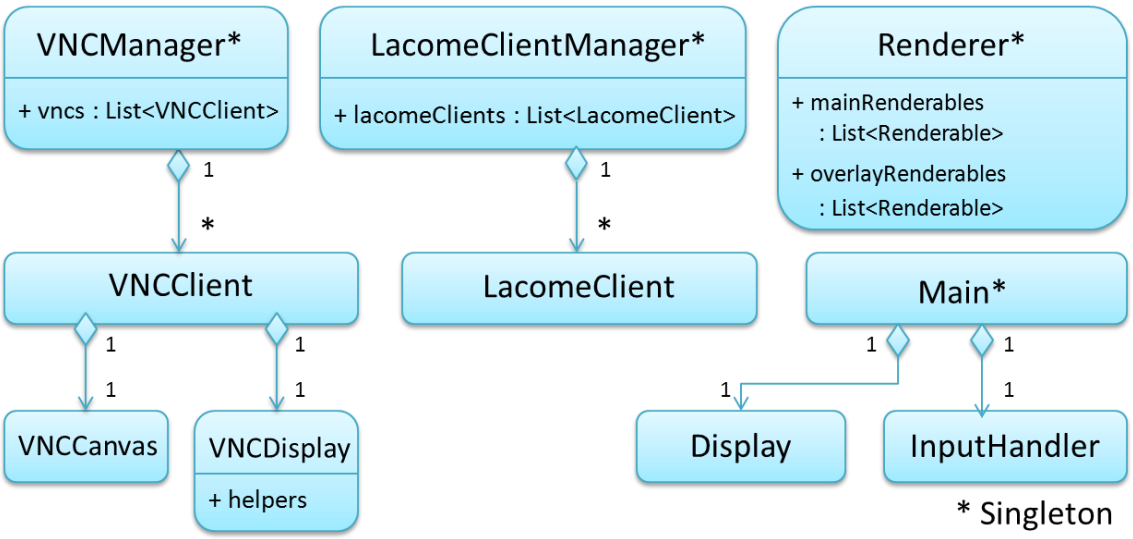

Figure 3.1 A UML-like summary of the LACOME Server architecture.

3.1.3 Rendering Using Textures

Rendering in LACOME consists of three steps. First, rendering commands are processed from any connected VNC network streams, converted into an in-memory representation of a pixel buffer, and queued for later rendering. This first step is, in essence, a pre-rendering step. The second step is to upload these texture updates to the graphics hardware. This merely updates the texture map stored in high-speed memory on the graphics card, and does not directly update the display. The third step of rendering is to

actually draw the textures to the display. This is a simple matter of rendering basic texture-mapped quadrilateral shapes, or "quads."

There are four potential bottlenecks in the rendering system. First, network bandwidth is limited and is largely outside of the control of LACOME. The VNC protocol attempts to minimize network traffic. The second potential bottleneck is that of memory and processor resources in the LACOME Server. In our experience, these have not been limiting factors during realistic usage. The third bottleneck is the connection to the graphics card; on modern systems this is the PCIe bus, while on older systems it was the AGP or PCI bus. The fourth bottleneck is the rendering performance of the graphics hardware: namely, its ability to perform operations such as drawing lines or texture-mapped quads. In practice, the cost of rendering the quads to the screen is miniscule on modern graphics hardware. The most important bottleneck is uploading texture updates to the graphics card. In the rendering algorithm described in the next section, special care is taken to minimize the number of update commands that must be issued.

3.1.4 Guaranteed Rendering Performance

As mentioned above in Section 3.1.2, each `VNCClient` is split into two parts: a `VncCanvas` and a `VNCDisplay`. The `VncCanvas` reads and processes VNC commands from the network and queues rendering commands in the `VNCDisplay`; the `VNCDisplay` is responsible for actually performing the rendering in OpenGL. The primary reason for this division of labour is that OpenGL commands can only be called when there is an active OpenGL context, and this condition is only satisfied within the main display thread and not within the various VNC threads.

By keeping track of how much time has elapsed during the rendering of a frame and allowing for early termination, before all queued rendering commands have been executed, we can ensure that each frame takes no longer than a predesignated minimum threshold to render. Currently, a frame time of 33ms has been chosen, corresponding to a guaranteed frame rate of 30 frames per second (FPS). This ensures that mouse

movements and window manipulations on the shared LACOME Server display are smooth and fluid, even when many VNC clients are simultaneously connected.

In this discussion that follows, it is assumed that the cost of actually drawing texture-mapped quadrilaterals to the display is small enough to be ignored. Only texture updates are potentially delayed to future frames of animation. While textures updates may be postponed all textures are drawn to the display, even if they have only been partially updated.

For each frame of animation, the same series of steps are performed. First, the start time is noted. Next, the rendering commands associated with the various `VNCClients` are executed until they have all been performed or until the maximum allowable time has passed. Finally, all renderable objects are drawn onscreen in six passes: one can imagine six panes of glass stacked atop one another as shown in Figure 3.1. Pseudo-code of the overall rendering algorithm structure is shown in Figure 3.3.

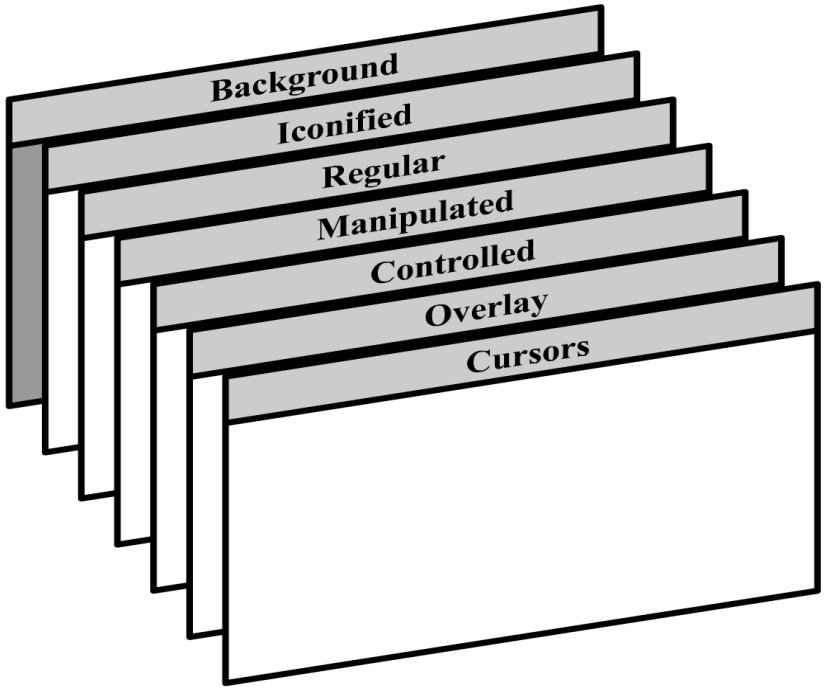

Figure 3.2 Compositing layers for rendered objects.

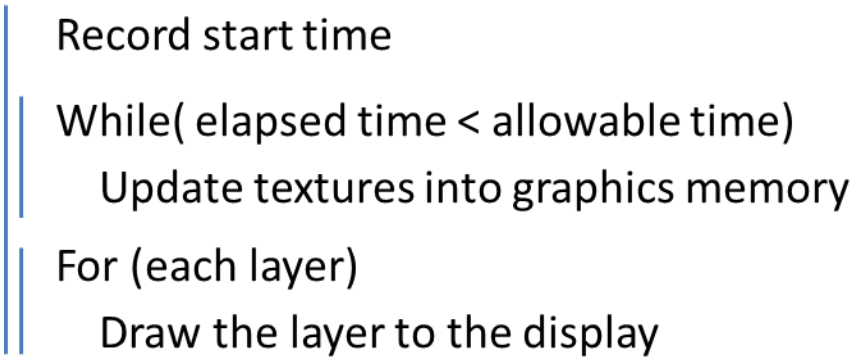

Figure 3.3 Pseduo-code of the rendering algorithm.

While objects on some layers are simply overlaid atop the lower layers, other objects are semi-transparent and are alpha-blended with content on the lower layers. The default choice is to use a black background, although this is easily modified. Starting with the deepest layer, which is potentially occluded by all other layers, iconified objects are drawn. Next are so-called "regular" objects that are not being manipulated or controlled, followed by objects that are being manipulated. Manipulation may entail either moving or resizing of a window. On the fourth layer are objects that are being controlled. Finally, overlay objects such as the IP address display or "Return to Desktop" button are drawn on the fifth layer and cursors are placed on the topmost layer, which is never occluded.

Each `VNCClient` maintains its own queue of rendering commands. When updating the textures it is critical to ensure that all `VNCClients` are serviced, even during periods of high demand when it is necessary to terminate rendering prematurely. To achieve this, we begin each display call by compiling a list, named `nonEmptyVNCDisplays`, of all `VNCDisplays` that contain queued rendering commands. As rendering progresses, a `VNCDisplay` is removed from this list if it has been serviced completely and there are no remaining rendering commands. A while-loop continuously loops over `nonEmptyVNCDisplays` until no `VNCDisplays` are left or until the maximum elapsed time has passed. On each pass through this while loop, which checks if the allotted time budget has been consumed, every element of `nonEmptyVNCDisplays` completes one update action. This ensures that `VNCDisplays` near the back of the list

are not starved of processing resources. More detailed pseudo-code of the rendering algorithm, including the above-mentioned details about how texture updating occurs, is shown in Figure 3.4.

```
Record start time
Construct nonEmptyVNCDisplays list
While( elapsed time < allowable time)
   For (each VNCDisplay in nonEmptyVNCDisplays)
      Partially update that VNCDisplay's texture
      If(VNCDisplay is empty)
         remove VNCDisplay from nonEmptyVNCDisplays
For (each layer)
   Render layer
```

Figure 3.4 More detailed pseudo-code of the rendering algorithm.

During each iteration over `nonEmptyVNCDisplays`, more than one rendering command may be executed per `VNCDisplay` as a performance enhancement. Each rendering command actually involves updating a rectangular portion of a texture; if several rendering commands target adjacent locations they may be collapsed into one larger command that can be issued with a single OpenGL call. VNC rendering commands often target grid-aligned 16px x16px squares. In the current implementation, all rendering commands of the same height with targets adjacent to each other on a single row are executed at once.

Figure 3.5 illustrates this process. The figure shows a representation of a 101px x 88px texture for a VNC session, divided into 16px x 16px squares, with smaller rectangles at the boundaries. Twelve rendering commands have been issued by the VNC Server; they are labelled in order with white numbers. The first five commands are all adjacent, so

they are collapsed into a single texture update command. Adding the 6th command would create a non-rectangular region, so the first texture update is issued. The 7th command is adjacent to the 6th, so they are combined and the texture update issued. The 8th command is not adjacent to either the 7th or the 9th command, so the 8th command is not combined and is instead simply executed. Note that although the 12th command is adjacent to the 8th command spatially, it is not combined with the 8th command because it is further down the queue and the queue is processed serially. The 9th and 10th commands are adjacent, so they are combined. The 11th command is adjacent to the previous two and so it is also combined with the 9th and 10th commands, despite the fact that it is located on the left side of them. Finally, the 12th command is issued on its own.

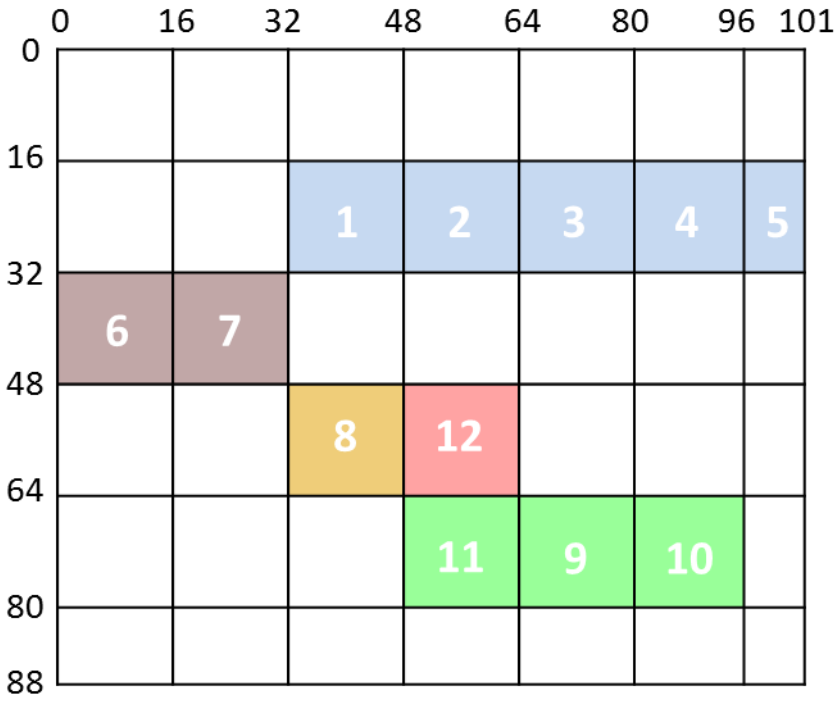

Figure 3.5 Rendering commands are combined before updating OpenGL texture. Twelve individual commands, marked by white numbers, have been collapsed into five texture updates.

3.2 Extensible UI Framework

The original LACOME Server presented a very simple user interface. Only two types of objects, VNC displays and cursors, could be rendered, and each of these was explicitly handled by the main rendering and mouse handling functions. In contrast, the new rendering engine supports a wide variety of renderable objects, each with unique display properties and interactions. This is a modular architecture, making it possible to separate display from behaviour. The development of this UI framework was stimulated by the need for diverse interaction techniques for the experiment detailed in Chapter 4, as well as by the desire for new interactive widgets such as the IP Address Display described in Section 3.3.1 or the "Return to Desktop" button described in Section 3.3.2.

Renderable objects reside in one of three locations in the LACOME Server. As mentioned in Section 3.1.2, "LACOME Server Architecture," the `Renderer` class contains two lists of renderable objects entitled `mainRenderables` and `overlayRenderables`. Objects in the `overlayRenderables` list are composited atop those in `mainRenderables`, as shown in Figure 3.2. Special UI elements that should not be hidden are added to `overlayRenderables`, while all other UI elements are added to `mainRenderables`.

Cursors are associated with a particular `LacomeClient` object. To render them, the display function simply iterates over the `lacomeClients` collection within the `LacomeClientManager`. Cursors are rendered on the topmost pane, because they should never be hidden.

Two software interfaces lie at the heart of the UI framework: `Renderable` and `PointerListener`. All displayable objects implement `Renderable`. This interface provides methods to query the object's size, position, and manipulation state. Additionally, a `Renderable` object can report whether a given point lies within it or not, and a `Renderable` object can have `PointerListener`s added to it.

A “pointer” in LACOME is a synonym for a cursor. When a mouse event occurs, such as a mouse move or mouse drag event, window entrance or exit event, or button press, release, click, or double-click event, the system determines over which object the cursor was located. Each `PointerListener` that belongs to the object is notified, in turn, of the mouse event. `PointerListener`s are notified in the order in which they were added to the `Renderable`. With this information, complex interaction techniques can be designed and implemented. In addition to mouse events, keyboard events are also handled in an identical manner.

While the LACOME Client sends all mouse motion as relative movements, the LACOME Server also accepts absolute positions. In the user study described in Chapter 4, all user interactions were recorded and could be played back at a later date. Because the LACOME Server accepts absolute positions, it is straightforward to “scrub” through the input data. One can jump to an arbitrary position in the data stream without needing to traverse a long chain of relative movements.

To illustrate the use of the UI framework the architecture underlying the LSO window manipulation technique is presented here. For a full description of the technique, including diagrams, see Section 4.2. In the LSO technique, a `VNCDisplay` is shown onscreen. The VNC session is a textured rectangle of fixed aspect ratio. To provide a border and title-bar, `VNCDisplay` inherits from a class named `FrameWindow`, which itself inherits from `Renderable`. `FrameWindow` is an abstract class that assumes a fixed aspect-ratio content pane; the only abstract method is `renderContentPane()`, which `VNCDisplay` implements by simply drawing the VNC texture in the correct place. `FrameWindow` provides methods to determine if a point lies within the title-bar, and to resize itself based on the desired size of the content pane.

When a `FrameWindow` is created, several helper objects are also created. Each `FrameWindow` holds a list of `Renderable`s called `overlays`; to this list is added a `BorderOverlay` and a `SnapRegionsOverlay`. A `BorderOverlay` is both a

`Renderable` and a `PointerListener`; using window entrance and exit mouse events the `BorderOverlay` keeps track of which cursors are currently atop its `FrameWindow`. It then draws outlines on its window in the colours corresponding to those cursors. If a cursor is actively manipulating or controlling the `FrameWindow` only one outline, of that cursor's colour, is drawn. A `SnapRegionsOverlay` is also both a `Renderable` and a `PointerListener`; it notes when a cursor moves into each of eight different "snap regions." When entered, a snap region turns white and opaque and then slowly fades back to total transparency after one second.

Three other `PointerListener` helper objects complete the LSO interaction mechanism for `FrameWindow`s. An `IconifyPointerListener` listens for double-click events and toggles the iconification state of the window. A `SnapMoveListener` allows the user to move the window, while a `SnapResizeListener` allows them to resize it. When the left mouse button is pressed while the cursor is atop the window the `SnapMoveListener` records the initial state of the window and toggles the manipulation state of the window to "manipulated." If the cursor is currently in one of the snap regions, it is moved to a corresponding location on the exact edge or corner of the region. Until the left mouse button is released, any mouse movement causes the position of the window to change accordingly. When the mouse button is released, the `SnapMoveListener` sets the window's manipulation state back to "free."

Similarly, the `SnapResizeListener` listens for right-click events on the window. When one of these events occurs, it records the initial state of the window and sets the manipulation state of the window to "manipulated." Again, if the cursor is currently in one of the snap regions, it is moved to a corresponding location on the exact edge or corner of the region. Until the right mouse button is released, mouse movements cause the window to be resized. For full details of resizing behaviour in each snap region, see Chapter 4. When the mouse button is released, the `SnapResizeListener` sets the window's manipulation state back to "free."

In the previous several paragraphs, we have seen how several `Renderables` and `PointerListeners` can work together to create an exciting widget, including interesting renderings and complex interaction techniques. We now show how the UI framework can support dynamically changing needs.

The experiment detailed in Chapter 4 contrasted the aforementioned "snapping" or "LSO" technique with a more familiar "Windows-style" window manipulation technique. To provide the alternate functionality, two new helper classes called `WindowsStyleMoveListener` and `WindowsStyleResizeListener` were used in place of `SnapMoveListener` and `SnapResizeListener`, respectively. Additionally, the `SnapRegionsOverlay` was not used for "Windows-style" `FrameWindows`. However, the `FrameWindow` class itself was able to be used unchanged between the two conditions. With a rather trivial programming effort, it is possible to allow a user to toggle back and forth between the two interaction mechanisms.

An additional example of the flexibility afforded by the UI framework also comes from the experiment in Chapter 4. Live VNC sessions were not used in the experiment; instead, one of two static images was shown and tinted either red or green. This was a simple modification: instead of using a `VNCDisplay`, we simply created a new class called `ExperimentDisplay` that also inherited from `FrameWindow` and rendered the content pane differently. All classes related to the interaction technique could be used without modification, despite the change in the display.

3.3 New Features

Several new features have been added to the LACOME Server, primarily to improve usability. An IP Address display helps users initiate connections from their LACOME Clients, while the Return to Desktop button provides an easy way for a LACOME Client to cease navigation. Basic configuration of the server can now be completed through the use of a simple configuration file. Finally, the system now provides full IPv6 support.

3.3.1 IP Address Display

Users wishing to utilize the LACOME system must specify to which LACOME Server they would like to connect. A user may connect to different LACOME Servers at different times, such as when travelling between several different meeting rooms or classrooms. The current version of the LACOME Client, an image of which is shown in Figure 2.3, has two text fields and a "Connect" button at the top of its user interface. The leftmost text field is for the LACOME Server address.

In the original LACOME Server, information on the address of the LACOME Server had to be supplied to the user out-of-band; that is, through ad-hoc or improvised communication channels not built into the LACOME system itself. Over several months of light usage of the system, at least three methods of communicating the LACOME Server's IP address were observed in practice. Most simply, a user requiring this information could simply ask the meeting group and receive a verbal reply. Another participant who had already connected would simply read the address aloud, and the user would type it in. Often, whoever had set up the meeting knew the address and bootstrapped the process. Verbal communication of an IP address presents two difficulties, however; it is slow and error-prone, and it disrupts the meeting at large. Thus, an alternate behaviour developed. When conveniently positioned, a user could ask a neighbouring user what the IP address was; the neighbour would turn his or her laptop so the first user could simply read and copy the address. This behaviour somewhat reduced the disruption of the group, but it was not always feasible and it was still slow and error-prone. In the final method, the person who set up the meeting would sometimes use a command line tool (`ipconfig` on Windows, or `ifconfig` on Linux) on the machine running the LACOME Server to display the IP address and then leave that window visible throughout the meeting. This had the advantage that the address was visible to all participants without any disruption, and required no use of a verbal communication channel. However, this technique used valuable screen real estate on the shared display and meant that the LACOME Server could not be run fullscreen.

The IP address display advances upon this last technique. Rather than display the address using a separate application, it is now displayed directly within the LACOME Server display. This minimizes the amount of wasted screen real estate. When the LACOME Server initiliazes, it determines all IP addresses belonging to the machine on which it is run. One of these addresses is chosen to be the default address displayed. If an address has been specified in the configuration file it is chosen as the default address; however, if the system determines that the specified address does not belong to the host machine a "?" is appended to the display. If no address has been pre-specified, the first address returned by `InetAddress.getAllByName()` is used. The IP address display is shown in Figure 3.6; the left side shows only the default address, while the right side shows all addresses used by the host machine because the cursor has moved onto the IP address display.

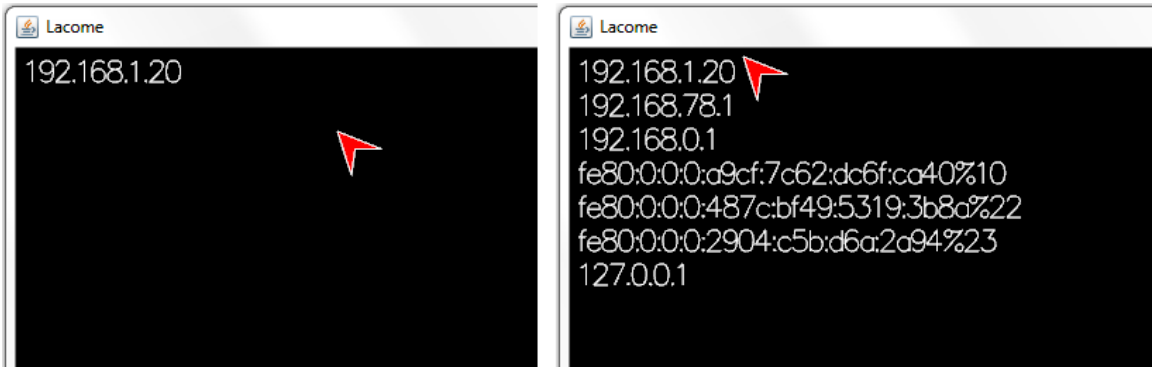

Figure 3.6 The LACOME Server's IP address display showing only the default address (left) and all addresses (right).

A computer may be attached to multiple networks. For example, a machine running a LACOME Server could be attached to the public internet and also be attached to a high-speed router for local connections. Different users may need to connect using different network interfaces, and therefore will need to specify different IP addresses for the LACOME Server. However, permanently displaying all IP addresses may consume too much screen real estate. To accommodate these conflicting needs, the IP address display has been made interactive; when a cursor hovers over it for 500ms, the display expands to list all IP addresses. When all cursors have moved off the display, it shrinks again to

show only the default address. The IP address display is shown on the overlay pane shown in Figure 3.2.

Internally, the IP address display consists of a single class, appropriately named `IPAddressDisplay`. It implements `Renderable`, `ScreenReshapeListener`, and `PointerListener`. It simply renders the IP addresses as text; a viewport clips those addresses that should not be visible at any given time. When a mouse enter event occurs the `IPAddressDisplay` begins timing, and after the fixed delay period expands the viewport over time to reveal the other addresses. By listening to screen reshape (resize) events, the `IPAddressDisplay` is able to maintain its position in the top left of the screen.

The IP Address display fulfills a fundamental need in LACOME, which is that end-users must be able to specify to which LACOME Server they would like to connect. Displaying the address onscreen is a simple and direct approach, although it has some limitations. Most users, who are not fluent touch-typists, must glance back and forth between their personal screen and the shared screen while they attempt to transcribe the address from the LACOME Server display to the appropriate location with the LACOME Client's interface. This is a somewhat tedious and error-prone process, and one that must be repeated each time the user connects to a different LACOME Server. An alternative would be to use a "lobby" server. A lobby server would reside at a permanent fixed address, which could then be stored in the LACOME Client's settings file. Upon initialization, the LACOME Client would connect to the lobby server and download a list of all currently active LACOME Servers. The user would then simply choose the LACOME session to which they would like to connect from a drop-down list, located in the LACOME Client interface where the LACOME Server address field is currently placed. Such a solution would prevent typographic errors and allow meeting-specific connections strings, such as "Alpha Team meeting," rather than requiring machine names or IP addresses.

3.3.2 Return to Desktop

When users click the “Toggle Nav” button in the LACOME Client, their mouse becomes trapped in the yellow navigation region. A cursor associated with them appears on the shared display, and mouse and keyboard input is redirected to the shared screen. A consequence of this is that the user is unable to use the GUI on his or her own display while navigating. In the original LACOME system, a special keyboard sequence was used as an escape command to end navigation, remove the user’s cursor from the shared screen, and return control to the user’s own display. The sequence Ctrl+Shift+Backspace was chosen because it is not in common use in other software applications. This shortcut is, however, cryptic, difficult to remember, and not easily discoverable by novice users.

Because users are free to use the LACOME Server’s GUI while navigating, it is sensible to put UI elements on the shared screen. A new “Return to Desktop” button has been added to the top-right of the LACOME Server, which is a familiar location for an exit button for Windows and Linux users. The button, shown in Figure 3.7, consists of a miniature computer desktop pointed to by a large green arrow.

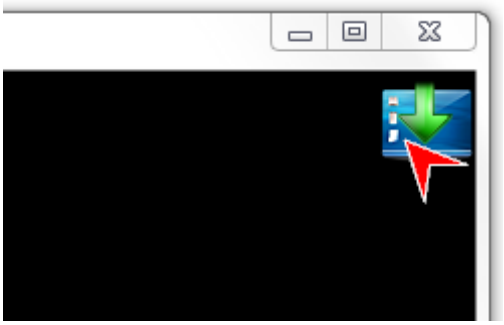

Figure 3.7 A cursor poised to click the Return to Desktop button.

When the Return to Desktop button is double-clicked, the LACOME Server sends a message over the network to the user’s LACOME Client instructing it that the user would like to cease navigation. The LACOME Client then ceases navigation and releases the mouse. Because the LACOME Server knows which LACOME Client is associated with the double-click, only that LACOME Client is affected.

Internally, the Return to Desktop button is an instance of the `ExitButton` class, which inherits from `Button`. `Button` implements `Renderable`, and simply displays an icon loaded from a file on disk. The `ExitButton` class also implements `ScreenReshapeListener`; when the window is reshaped (resized) the `ExitButton` repositions itself to be in the top-right corner. A simple class named `ExitButtonPointerListener` is added to the `ExitButton` and listens for double-click events. The Ctrl+Shift+Backspace sequence still works.

3.3.3 Configuration File

A very simple text file is used to configure the LACOME Server. It currently only supports four options: the default IP address to be displayed, the port on which to listen for LACOME Clients, and the initial screen width and height. In the original LACOME Server these values were all hard-coded and required re-compilation if changed. Each value is assumed to be formatted correctly and each value is separated by a newline character. The IP address is optional. As discussed in Section 3.3.1, if no IP address is specified the first one found by the system is displayed as the default instead. If no configuration file can be found, one is created with hard-coded, default settings.

3.3.4 IPv6 Support

The new LACOME Server fully supports IPv6, whereas the original LACOME Server did not. The key problem is that Microsoft Windows is a single-stack platform. This refers to a "protocol stack," also known as a "communications stack" for networking. IP is Layer 3 of the seven-layer OSI model. On Windows, the implementation of this layer supports only a single family of IP addressing, either IPv4 or IPv6. A socket created on Windows must specify which family it supports. To listen on both network interfaces, two sockets must be created. Linux, however, is a dual-stack platform; only one socket need be used.

In Java, dual-stack support for IPv6 was added for the Windows operating system in Java 5.0 (Oracle, 2010c). This makes adding IPv6 support completely transparent and

automatic. Only one socket is required to service both network stacks when using Java on any platform.

4 Large Screen Window Manipulations

The LACOME system, as described in the first three chapters of thesis, provides a functional system for publishing content from users' personal computers to a large shared display. However, interacting with content on large displays can be difficult when using traditional interaction techniques. This chapter describes a window manipulation technique, named the LSO or "Large Screen Optimized" technique that was designed for use with LACOME. The technique could also be used with other systems that utilize large screen displays. In addition to details of the LSO technique, this chapter presents a literature review specific to window management research, and describes a controlled user study that compares it to a more traditional window manipulation technique.

Introduction

Display technology continues to make rapid advances, providing ever-larger physical sizes, greater resolutions, and higher contrast at ever lower costs. It is now feasible to have multiple high-definition displays available in even small meeting rooms, and it is plausible that large wall-size displays will soon become commonplace. Such displays can improve performance on navigation, search, and comparison tasks in information rich virtual environments (Ni, Bowman, & Chen, 2006), and are inherently well-suited for collaborative activities. However, it is often difficult to switch which user controls the display, and there is no broad support for multiple simultaneous users. While individually small, these technical hurdles disrupt collaborative activities and prevent effective use of available displays.

Research into collocated collaborative meeting systems for single display groupware (SDG) and multi-display environments is ongoing. Systems such as WinCuts (Tan, Meyers, & Czerwinski, 2004), Integrated Tabletop (Nakashima, Machida, Kiyokawa, & Takemura, 2005), IMPROMPTU (Biehl, Baker, Bailey, Tan, Inkpen, & Czerwinski, 2008), WeSpace (Wigdor, Jiang, Forlines, Borkin, & Shen, 2009) allow multiple users to share content from their personal computers on public displays and to allow others to

interact with that content. Window manipulation typically occurs only when a user drags the margin of the windows (title bar, border) as in traditional window management systems. In contrast, LACOME utilizes multiple control layers to better support differentiated control modes, so that the window manipulation (e.g., resizing, moving, iconifying) is distinct from interacting with content displayed within published windows. Because of the design choice to require a mode switch in order to interact with the content within a window, LACOME is able to support alternative window manipulation techniques.

Traditional windowing interfaces such as those of Microsoft Windows or Mac OS X may not be as usable on a large display as they are on personal displays. Swaminathan and Sato (Swaminathan & Sato, 1997) found that when a display exceeds a certain size, it becomes "qualitatively different." Larger displays enable users to create and manage many more windows (Bi & Balakrishnan, 2009), as well as to engage in more complex multitasking behaviours (Czerwinski, Robertson, Meyers, Smith, Robbins, & Tan, 2006). A recent week-long study (Bi & Balakrishnan, 2009) comparing usage of a large display to single and dual monitor configurations for daily work suggests the need for different window management techniques for a large wall-size display. Users of the wall-size display performed more moving and resizing operations than in their normal environments. They also had difficulty accurately selecting the corner of an application window for resizing, particularly when the window was located in the peripheral region of the screen. Collocated meetings may provide additional challenges for large screen display use; users may face multiple cognitive demands simultaneously and are rarely in the ideal position to view a shared display. Moreover, when multiple users with multiple cursors interact on shared displays, larger target sizes may ease selection (Moraveji, Inkpen, Cutrell, & Balakrishnan, 2009).

We are interested in exploring alternative window manipulation techniques, specifically for use on large displays during collocated collaboration when users' desktops are published to a very large display. We exploit the fact that, in such situations, window

manipulations such as moving and resizing are more common than direct interaction with a window's content. We assume that the collaborative system either explicitly provides multiple control layers or that mode switches could be accomplished by a hotkey or some similar approach. In this chapter we present a novel window manipulation technique, LSO, which stands for "Large Screen Optimized." LSO provides manipulation handles that overlap content, providing a larger target area for moving and resizing windows. We experimentally validate our technique in a within-subjects laboratory study in which participants used both LSO and a traditionally-based technique to move and resize windows on a wall-sized display. Our results show that participants were faster when using LSO and that they found it easier to use. While our technique was designed specifically for collaborative use of large displays, it may also be suitable for individual use on large screens or for use on personal computers. In particular, those users with larger monitors or secondary displays may more frequently arrange multiple visible windows on the screen (Hutchings & Stasko, 2004).

We review background and related work specific to window manipulation techniques. A discussion of more general related work applicable to the LACOME system as a whole was presented in Chapter 1. This chapter also includes a detailed description of the LSO technique, the experimental design for a quantitative user study measuring the effectiveness of the technique, and the results and conclusions drawn from that study.

4.1 Background and Related Work

We briefly present related work on collaborative meeting software and background on standard window manipulation techniques in modern operating systems before describing research investigating users' windows management practices. We then summarize research investigating new window manipulation techniques.

4.1.1 Collaborative Meeting Software

There have been many research projects developing systems to support collocated meetings and the sharing of multiple private desktops and windows on shared public displays. We highlight some of the more recent ones.

WinCuts (Tan, Meyers, & Czerwinski, 2004) is an extension to the VNC protocol enabling a VNC server to replicate an arbitrary region of a desktop or window instead of an entire desktop. WinCuts has the potential to optimize use of limited screen space. IMPROMPTU (Biehl, Baker, Bailey, Tan, Inkpen, & Czerwinski, 2008) enables users to share windows both on a public group display and on their own personal displays. IMPROMPTU is tied to MS Windows. It encompasses a number of user interface elements that enable easy sharing of application windows, and visibility of who is sharing what via the system. Similarly WeSpace (Wigdor, Jiang, Forlines, Borkin, & Shen, 2009) enables the sharing of application windows on both a large shared wall display and a multi-touch table display, using VNC as the basis of its sharing mechanism. WeSpace implements a number of new features, such as a custom API for developing applications to extend the collaborative ability of the system.

All of these systems use traditional window manipulation techniques as supported by standard operating systems. For example, WinCuts (Tan, Meyers, & Czerwinski, 2004) exposes only the immediately relevant sections of a window rather than the whole window, by placing the 'cut' region into a new window. The system relies on the operating system to control window manipulations such as moving and resizing on the new window.

4.1.2 Window Manipulation in Operating Systems

Microsoft Windows and Apple's Mac OS X both build window manipulation functions directly into the operating system. On Unix and Linux, window manipulation is handled by a 'window manager,' a special X11 program that varies depending on the Linux distribution used. We will focus on Metacity and Compiz, the default window managers

for Ubuntu, Debian, and Fedora, which are three of the most common Linux distributions, as well as Windows and Mac OS X techniques.

Standard approaches for window manipulation in commercial operating systems quickly emerged after the introduction of multiple windows in personal computer. Chepuis et al. provide a comprehensive history of window management techniques (Chapuis & Roussel, 2005). Dragging the title-bar to move a window is standard in all three systems (Apple Inc., 2010; Compiz, 2010a; Gnome, 2010a; Microsoft, 2010a). Mac OS X also allows moving by clicking on optional toolbars and bottom bars that together with the title-bar are collectively referred to as the 'window frame' (Apple Inc., 2010). Linux has the unique capacity to move a window by holding down the 'ALT' key on the keyboard and dragging from anywhere in the window (Compiz, 2010a) (Microsoft, 2010a). This is an example of an explicit mode switch for window manipulations. While Linux defaults to allowing the user to interact with a window's contents, LACOME defaults to allowing window manipulations.

Microsoft Windows and Linux both enable resizing a window by dragging the small (approximately 5 pixel) border around the window (Compiz, 2010b) (Gnome, 2010a) (Microsoft, 2010a); however, Windows 7 has increased the border to 8 pixels. Mac OS X uses a different method for resizing that involves dragging the 'resize control' located in the lower right hand corner of the window. Mac OS X windows cannot be resized anywhere else (Apple Inc., 2010). Microsoft Windows and Linux both offer similar resize controls in the lower right hand corner; however, those controls are considered obsolete in the most recent version of Microsoft Windows (Microsoft, 2010d) and are only sporadically implemented in Linux applications (Gnome, 2010b). Additionally, Compiz on Linux offers the ability to resize a window by holding down the 'ALT' key and dragging with the middle mouse button, anywhere in a window (Compiz, 2010b).

4.1.3 Window Management in Practice

Researchers have investigated how users manipulate their windows in practice ever since windowed systems became popular. In 1986, Bury and Darnell (Bury & Darnell, 1986)

reported that people performed more accurately in windowed systems than non-windowed systems, but they also performed more slowly. The reason for the slower performance is that people spent more of their time engaged in screen-management activity.

More recently, Hutchings and Stasko (Hutchings & Stasko, 2004) examined how users managed their windows on regular single and dual display desktop computers. They classified users into three categories: maximizers, near maximizers, and careful coordinators. Users of larger screens preferred careful coordination; they simultaneously arrange multiple visible windows on the screen, which in turn requires more fine-grained window manipulation.

Bi and Balakrishnan (Bi & Balakrishnan, 2009) conducted a week-long user study to compare usage of a large display to single and dual monitor configurations for daily work. Users of the wall-size display performed higher percentages of moving and resizing operations than in their normal environment. Furthermore, users had difficulty accurately selecting the corner of an application window for resizing, particularly when the window was located in the peripheral region of the screen. Bi and Balakrishnan also found that many people had difficulty resizing windows due to small resize targets and long resize distances. They recommended that new methods of window manipulation be developed for large screen displays. These methods should focus on improving the interactions of resizing and moving windows, which are common on large screen displays, and deemphasize less frequently used operations such as minimizing and maximizing.

Moraveji et. al. (Moraveji, Inkpen, Cutrell, & Balakrishnan, 2009) found that for large groups, such as those with more than 16 members, group performance of multiple cursors on a single display degraded with smaller target sizes. Because most windowing systems use relatively small interaction points for window manipulations such as move and resize, this could become a relevant point as the size of groups interacting on shared displays grows.

4.1.4 Window Management Research

A number of research projects address problems with window management. Most of these systems are designed specifically for single-user systems and may not be as appropriate for a multi-user shared display. Elastic Windows (Kandogan & Shneiderman, 1996) reduces the number of move and resize operations that need to be carried out manually by grouping windows together and moving and resizing them as a group. However, in multi-user systems resizing a second window due to manipulations of the first could disrupt another individual's work. QuickSpace (Hutchings & Stasko, 2002) involves moving and resizing other windows in response to operations conducted on the active window. Unlike Elastic Windows, QuickSpace does not require window groupings, making it more general. However, similar to Elastic Windows, QuickSpace's automatic move and resize operations could disrupt others in a multi-user environment. E-conic (Nacenta, et al., 2007) uses knowledge of the user's perspective to draw a consistent representation of objects in multi-display environments. This can reduce the problem of distant objects being hard to see and manipulate on large displays. However, because the system is based on a single user's perspective, objects appear skewed to all other participants, making the system impractical in co-located group interactions on shared displays. Metisse (Chapuis & Roussel, 2005) is a window system that allows researchers to easily design, implement, and evaluate new window management techniques, such as ZoomOutAndMaximizeHeight and position-dependent window transformations that are indirectly applied when a user moves a window (e.g., shrinking the window as it is moved to the periphery).

Some systems require semantic information about window content. It is an open question as to whether such automated systems will work well with multiple users. For example, AdWiL (Haraty, Nobarany, DiPaola, & Fisher, 2009) utilizes a genetic algorithm to place windows for people. Shrinking Window Operations (Hutchings & Stasko, 2004) reduces the size of windows based on their relevant semantic content, similar to WinCuts (Tan, Meyers, & Czerwinski, 2004). However, this requires access to the semantic content of the window and the ability to determine what is relevant.

Other approaches do not address the problem of moving and resizing windows. For example, Chapuis and Roussel (Chapuis & Roussel, 2007) propose a system to automatically expose windows during copy and paste, in order to minimize the number of moves and resizes required to finish a task. Hoffmann et al. (Hoffmann, Baudisch, & Weld, 2008) propose a series of methods for visually cueing window notifications and switching in large screen environments, an approach which may add confusion when multiple users are working in different regions of the screen. Sugawara and Maruta propose to replace many window management activities with a push-pin metaphor (Sugawara & Maruta, 2009). However, their system is designed for use on small screen devices and may not be suitable for shared large displays. Also, the total replacement of the window metaphor may confuse people more accustomed to traditional windowing systems.

4.2 Large Screen Optimized (LSO) Technique

Our technique takes advantage of the fact that to interact with the content of a window in LACOME the user must explicitly take control of that window. This allows the entire window to be used for manipulations instead of only the title-bar and borders. The LSO technique introduces several novel features. It allows window manipulations to take place anywhere in the window, including within the content pane. It includes 'snapping regions' that automatically move the cursor to the boundary of the window. Finally, it supports a 'zooming' resize method.

A video figure illustrating the LSO technique is available at: http://www.cs.ubc.ca/labs/imager/video/2010/LargeScreenDisplayWindowManipulations/LSOTechnique.mp4, and is recommended for readers interested in the details of the technique.

In the LSO technique, windows are divided into nine regions: four edge regions, four corner regions, and the remaining centre region. For our setup, the edge regions were 50 pixels wide and the corners were 50 pixels square. A distance of 50 pixels corresponds to

approximately 10 centimeters of physical length on our large screen display. Further details of the hardware used can be found in Section 4.3.5.

To move a window, a user can click anywhere in the window, not merely the title-bar. If we consider initiating a move operation as a Fitt's Law task, the ability to click anywhere in the window greatly increases the target width compared to the Traditional technique, detailed below. While the left mouse button is depressed, the cursor is replaced by an arrow-cross. A click in the central region, which is that area which is neither a corner-snapping nor an edge-snapping region, causes the point under the cursor to remain under the cursor as the cursor moves around. If the click is in the snap region near a corner, the cursor is automatically moved to the exact corner. Similarly, if the click is in the snap-region of an edge, the cursor is automatically moved to the nearest point on that edge. Because the cursor can never leave the screen, this gives users a fast and precise method to position a window against an edge or in the corner of the screen, as shown in Figure 4.1. In the left-most pane of the figure, a user positions the cursor over an edge-snapping region. In the actual program, the snapping regions are temporarily highlighted when rolled-over; for clarity, all snapping regions are drawn in this figure. In the centre pane, the user presses the left mouse button, causing the cursor to change to an arrow-cross and snap to the exact edge. In the right-most pane, the user drags the mouse upwards to position the window against the top of the screen. Continued upwards movement of the mouse does not move the cursor further, as it is forbidden to leave the screen.

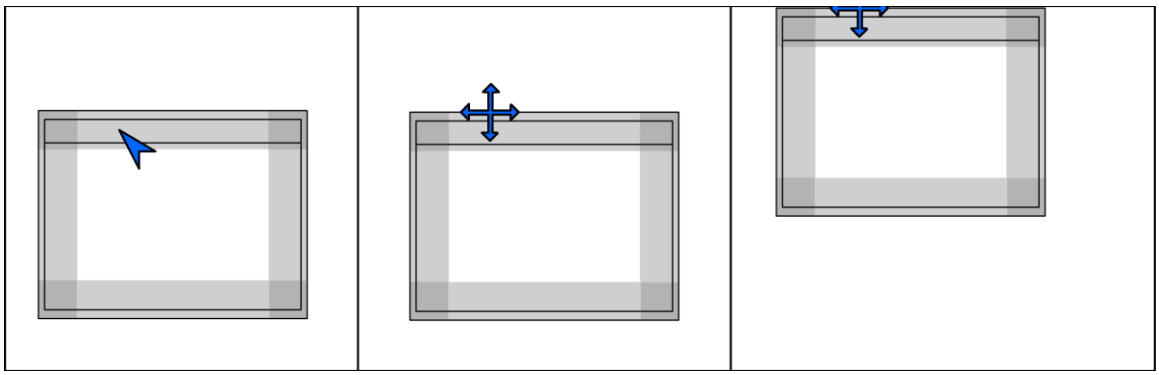

Figure 4.1 LSO move technique.

In Fitt's Law terminology, the edge or corner of the screen is a target of infinite width; the user need merely move the cursor as rapidly as possible towards the edge and it will stop when it reaches the edge. Since snapping positioned the cursor at the exact edge or corner of the window at the beginning of the move action, it will be properly positioned against the edge or corner of the screen after moving. The snap regions are indicated to the user by a semi-transparent white overlay that appears when the cursor rolls over the snap region, and linearly fades after one second to full transparency. It should be noted that the snap regions do not change size; they are simply highlighted to indicate to the user that the cursor is within that snap region. Implementation details on the rendering of overlay regions can be found in Section 3.2.

To resize a window in the LSO technique the user clicks with the right-mouse button, anywhere in the window. While the right mouse button is depressed, the cursor changes to an arrow pointing in the direction it will move when dragged, as shown in Figure 4.2. If she clicks near a corner, the cursor is moved to the exact corner; this is shown by the left and centre panes of the figure. As in the traditional technique, the opposite corner remains fixed and the selected corner remains under the cursor as it moves. If the user clicked near an edge, the cursor is moved to the nearest point on that edge and that point remains under the cursor as it moves. As the user drags the top-right corner, as in the right pane of the figure, the window is resized and the bottom-left corner remains fixed in position.

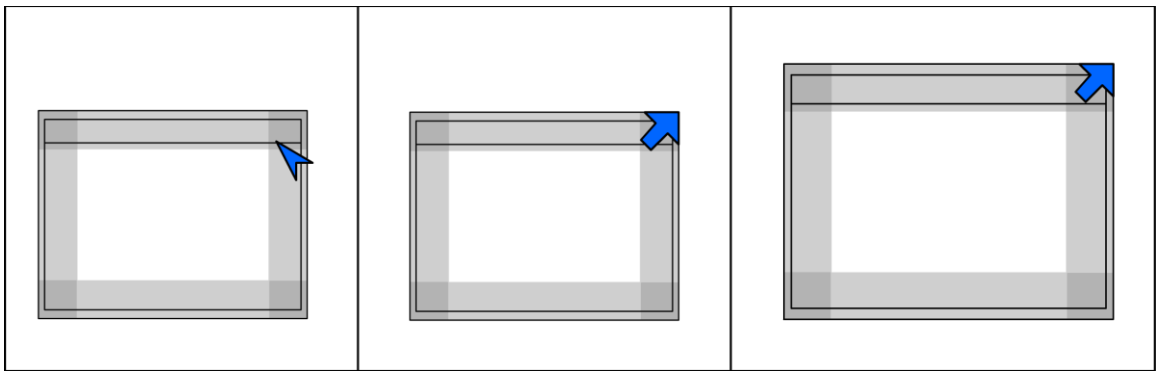

Figure 4.2 LSO resize technique.

The zooming resize technique is shown in Figure 4.3. If the user right-clicks in the central portion of the window, the cursor remains stationary and the point under it remains fixed. The rest of the window scales about that point as the mouse is moved up and down, analogous to 'zooming' in and out of that point. This is shown in the figure; in the time between when the centre pane and the right pane were rendered, the user moved the mouse upwards to increase the size of the window. While the right mouse button is depressed, the cursor becomes an arrow pointing to the top-right and remains stationary throughout the manipulation.

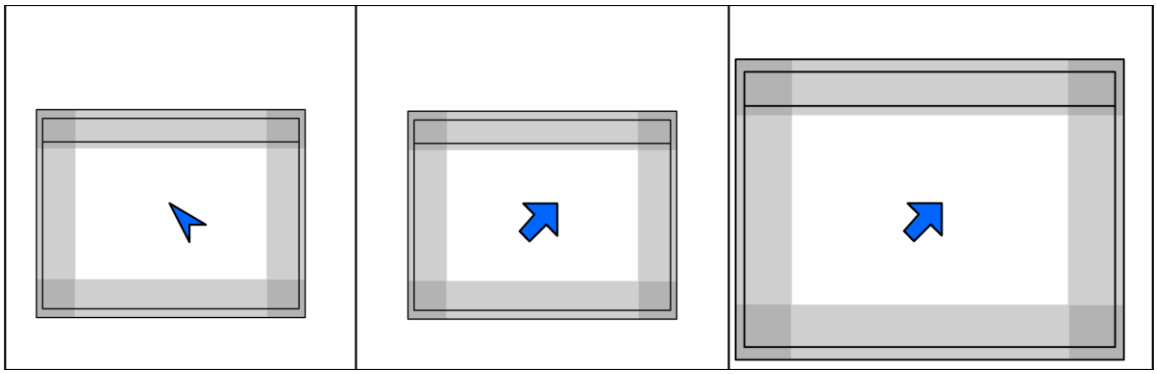

Figure 4.3 LSO zooming resize technique.

4.3 Experiment

We conducted an experiment to examine whether the LSO technique was more effective for moving and resizing windows on a large screen display than a traditional manipulation approach. While one of the motivating scenarios for our technique is collaboration, we limited our evaluation to single users to allow us to focus on the effectiveness of the technique for window manipulations typical during large screen display use.

4.3.1 Experimental Design

A within-participant, crossed factorial design was used. As summarized in Figure 4.4, we varied the window manipulation technique ('Traditional', LSO), start button position

(25% from bottom, 25% from top; always centered horizontally), initial window size (355x236, 1033x614), target size (355x236, 1033x614), and target locations (12: each screen corner, center of each screen edge, 30% towards center from each corner). Only a single start position (center of screen) for the window was used. Between subjects, we counterbalance the ordering of window manipulation technique ('Traditional-then-LSO', 'LSO-then-Traditional'), and content pane image (green or red-tinted pebbles).

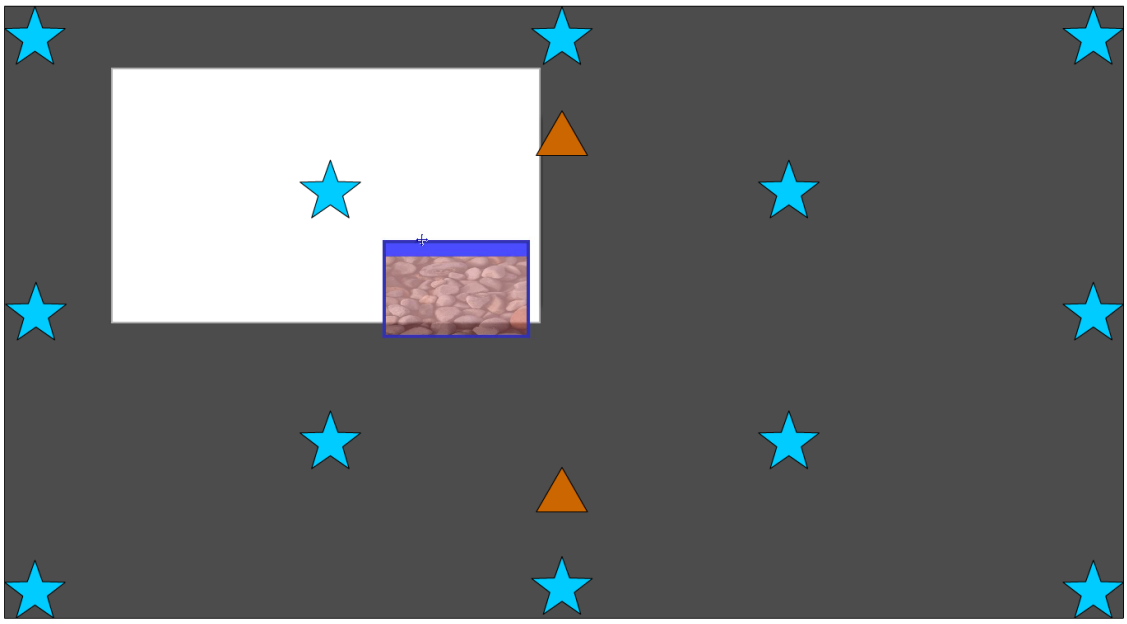

Figure 4.4 Placement of objects in the experiment design. Screenshot of a trial in progress, with overlays of the 12 possible target window positions (blue stars) and 2 Start Button Positions (red triangles). The white target window is the "large" size, and the pebbled experiment window is the "small size."

4.3.2 Window Manipulation Techniques

The two window manipulation techniques are the LSO technique, described in Section 4.2, and a 'Traditional' technique that was designed to be as familiar as possible, with many similarities to the windowing mechanism of Microsoft Windows. To move a window in the traditional technique, the user left-clicks the title-bar and drags the mouse.

To resize a window the user left-clicks the window border, either on a corner or on an edge, and drags the mouse. If a corner was chosen, the position of the opposite corner remains fixed and the selected corner stays under the cursor as the cursor moves. If an edge was chosen, the position of the point directly across from the selection point remains fixed, and again the selected point stays under the cursor. When the cursor is placed over the title-bar it changes to an arrow-cross ("cross barby") to indicate that clicking will initiate a move; when over the window border, the cursor changes to an arrow pointing in the direction to which the cursor will be constrained while resizing.

For the purposes of the study, both techniques share the property that they preserve the aspect ratio of the content pane. This is motivated by the fact that the use case for these windows is publishing computer desktops on a large shared display; reshaping a window would thus entail either distorting or cropping the content of the window. While the window is being resized the cursor is constrained to move only along a path that will preserve the aspect ratio. That is, the cursor will only move diagonally if a corner was selected, horizontally if the left or right edge was selected, and vertically if the top or bottom edge was selected. In the experiment an aspect ratio of 2.375:1 was used for the content pane. The design of the experiment windows is shown in Figure 4.5; while the real windows were coloured and textured, the window is shown here in wireframe for clarity.

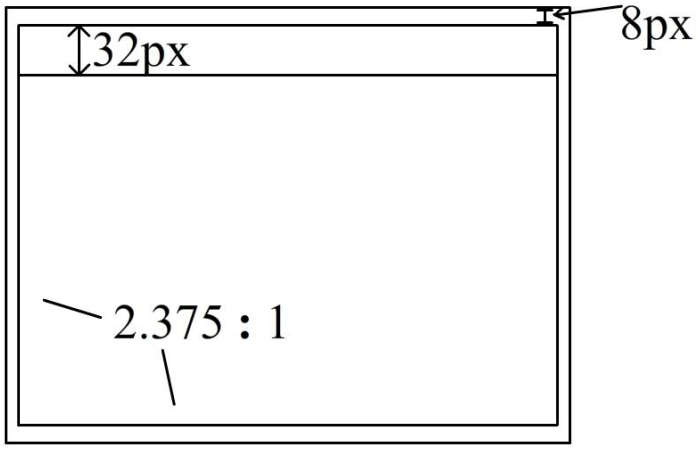

Figure 4.5 Design of windows used in the experiment.

4.3.3 Task

Participants were required to perform window manipulations involving moving and/or resizing. On each trial a 'Start' button is centered horizontally, and positioned either 25% from the bottom or 25% from the top of the screen. Two positions were utilized in order to investigate the effect of distance to the target. For the Traditional technique, the top position is closer than the bottom position, while for the LSO technique the distance is the same. The start button is 135x50 pixels in size. When clicked, the start button disappears and the trial starts. Immediately, a window centered on the screen appears. Each window, as illustrated in Figure 4.4, shows an image of pebbles in its content pane for the experiment. Above the content pane is a solid-color title-bar, with a height of 32px (Figure 4.5). The title-bar has no buttons or other decorations within it. Surrounding both the content pane and the title bar is an 8px solid-color border.

The task is to fit the window over the target area, which appears in one of twelve locations when the start button is clicked. For some trials the window is initially the same size as the target, so it is only necessary to move the window. On other trials, the window also needs to be resized. For example, the window shown in Figure 4.4 needs to be resized to fit the larger white rectangle. It is not required that the match be pixel-perfect; we consider a trial completed when all four corners of the window are within 10px of the

corners of the target. Upon achieving a successful fit both the window and the target area disappear, and the Start button reappears for the next trial.

To allow the target behind the window to be visible at all times, the window is made semi-transparent. A close look at Figure 4.4 reveals that the white target rectangle is visible through the experiment window. While being manipulated it has an alpha value of 0.5 (equal blending); otherwise it has an alpha value of 0.8 (mostly opaque).

4.3.4 Study Protocol

Each participant was guided through three initial trials to teach them the basic mechanics of the task. They were given 30 training trials with continued verbal instruction about the technique, read from a script. Questions were encouraged during these training trials. Participants next completed a block of 48 trials, after which they were asked to move to a different table and complete a maze, which was intended to reduce fatigue or boredom effects from performing a large number of repetitive tasks. This was followed by a second block of 48 trials and a short questionnaire regarding the technique they had just used. This process (Training, Block 1, Maze, Block 2, and Questionnaire) was repeated for the second window manipulation technique. Finally, participants completed a questionnaire comparing the two conditions.

Each half of the experiment consisted of 96 unique trials, split into two blocks. The blocks were balanced, fully crossed on initial window size, target size, and start button position. Six of the target locations were used in each block (2 corners, 2 edges, 2 floating). See Figure 4.4 for their placement. The 24 participants completed 96 blocks in total. The 48 trials within each block were intended to be randomly ordered. Unfortunately, due a programming error only four random orderings were actually used. Each of the first 12 participants saw the same ordering for Block 1 of each technique, and a second ordering for Block 2 of each technique. Similarly, each of the second 12 participants saw a third ordering for the Block 1, and a fourth ordering for Block 2.

4.3.5 Apparatus

We used a large screen wall display measuring 5.31m in width and 2.97m in height (17.4'x9.8'). The display is comprised of an array of 12 projectors in a 4x3 tiling shining onto a single frosted-glass vertical surface. The bottom of the screen is at floor level, and it is centered in an approximately 10m x 6m room. The display is depicted in Figure 4.6. A dual-link DVI output feeds into a small network of XPO3 video processors from Cyviz LLC, which 'chop' the output into 12 portions and blend the edges together.

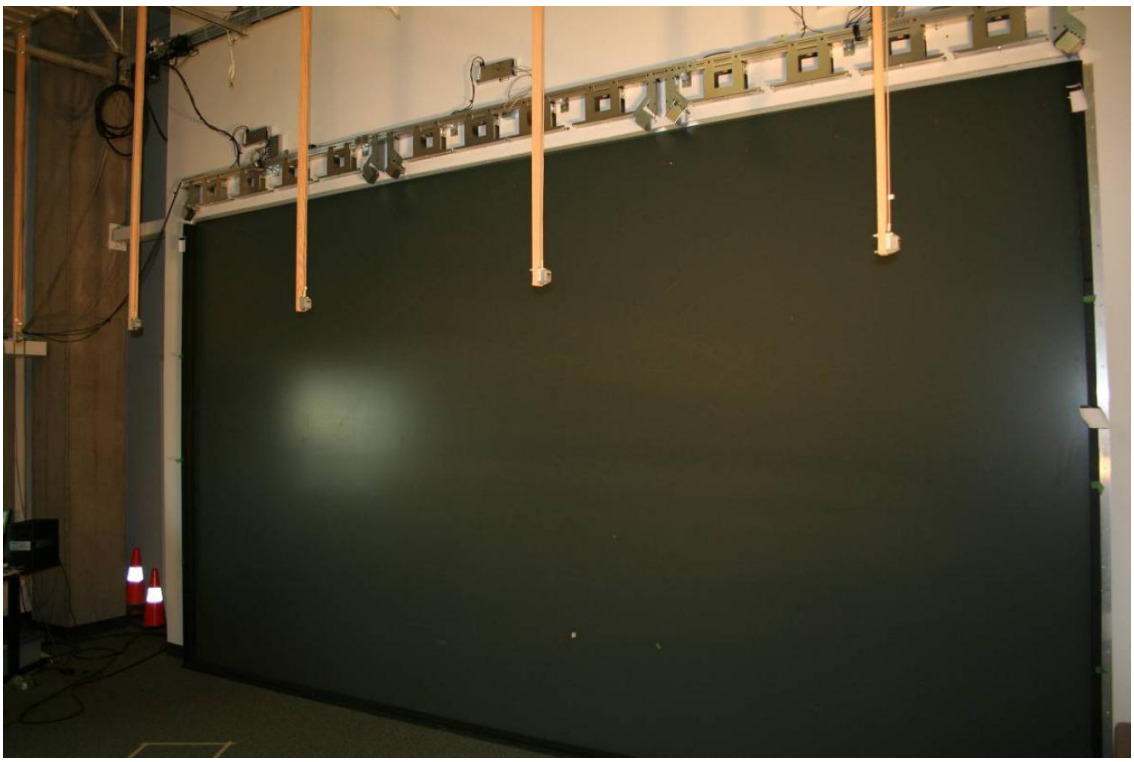

Figure 4.6 Photograph of large screen display used in the user study.

The projectors each have a native resolution of 800px x 600px, and were given a 160px (~30 cm) blended overlap with their neighbours in both the horizontal and vertical directions, for a total resolution of 2720px x 1480px. The final blending functions were not implemented at the time of the experiment so some variation in the brightness and color of the individual projected tiles was still present. Additionally, there were a small number of projector alignment errors that led to blurriness or tearing of the composite

image. These alignment errors are most visible where there are high-contrast edges and/or fine details, such as black text on a white background. To minimize the effect of these alignment errors and blending variations we used solid colors of medium lightness for the screen background, window title-bars, and window borders, and an image of mottled pebbles for the window content.

Participants were seated at a table that is 1.5m long, 0.75m wide, and 0.7m high. The edge nearest the participant was centered and positioned 2.7m from the screen. This distance was chosen for a number of reasons. Earlier work by Bi and Balakrishnan (Bi & Balakrishnan, 2009) suggests that a distance of 2.0-2.5 meters is preferred for a screen of this width. However, at 6' tall their screen was shorter than ours, and pilot participants noted that a greater distance was required to allow natural viewing of the corners of the screen. Additionally, our screen is seated at ground level so shorter pilot participants could not see the bottom of the screen because of occlusion by the far edge of the table, which required us to use a table of somewhat narrow width.

Participants used a "Microsoft Wireless Optical Mouse 2000" to move the on-screen cursor. As the resolution was quite high, pilot participants found it difficult to move the cursor to the corners in one motion and still have the fine control necessary to complete the task, unless cursor acceleration was left on. We enabled cursor acceleration and set the cursor speed to 6 out of 11 in the Windows dialog box for mouse settings. The experimental software ran full-screen so users were unaware of any details of the underlying computing environment.

4.3.6 Participants

We recruited 24 participants (15 male, 9 female) from the university community. Participants were required to have normal or corrected to normal vision and a lack of colour blindness. All but one participant was right-handed. Only 10 participants indicated that they regularly use (more than once per week) a single type of input device; almost all (23/24) were regular users of mice, but touch pads (15), touch screens (3), track point (1) and stylus (1) were also in use. Roughly half (13/24) regularly use multiple operating

systems: Microsoft Windows (21), Linux (12), and Mac (5). No participants indicated that they regularly use a large screen display, but 11 indicated that they had performed window manipulations on wall/table top displays (5) and projected wall displays (6) in the past.

4.4 Results

We first present manipulation time results and then subjective results from the questionnaires. We qualitatively support our findings with participants' comments about the ease of use and relative advantages of the two techniques.

4.4.1 Performance Data

Preliminary analysis revealed that, as expected, content pane image (green-tinted pebbles, red-tinted pebbles) had no impact on the findings, so we dropped that variable from our analysis. The combination of initial window size and target size results in two types of Action: Move (small to small, large to large) and Move+Resize (small to large, large to small). The dependent variable was manipulation time, which began when the start button was clicked and ended when the window was correctly positioned over the target.

Changes in scores on the dependent variable were analyzed in a 2 (technique order: LSO-1st, LSO-2nd) x 2 (technique: Traditional, LSO) x 2 (Action: Move, Move+Resize) x 3 (Final Position Type: corner, edge, floating) x 2 (Block: first, second) x 2 (Start Button Position: top, bottom) mixed analysis of variance (ANOVA) with a between subjects measure on the first variable. Adjustments were made to the ANOVA results, using the Huynh-Feldt correction for the Action x Final Position Type interaction to account for the violation of the sphericity assumption revealed by Mauchly's Test of Sphericity.

Table 4.1 provides a summary of the main effects and shows the six interactions that reached significance. Significant main effects were found for all the within-subjects variables, but not for the between-subjects variable. We will examine the interesting findings in turn.

Table 4.1 Main effects and interactions for mean manipulation time.

†Huynh-Feldt correction applied		Manipulation Time		Test Statistics		
*p<0.05	IV	μ	SD.	F	p	η^2
Main Effects (Between Subjects)						
Technique Order	LSO-1	5.41	3.48	0.508	0.484	n/a
	LSO-2	5.04	3.03			
Main Effects (Within Subjects)						
Technique	Traditional	5.57	3.25	21.660	.000*	.496
	LSO	4.89	3.36			
Action	Move	3.09	1.46	223.506	.000*	.910
	Move+Resize	7.36	3.29			
Final Position Type	Corner	5.38	3.64	11.046	.000*	.334
	Edge	5.29	3.33			
	Float	5.01	2.95			
Block	First	5.37	3.59	7.652	.011*	.258
	Second	5.09	3.02			
Start Button Position	Bottom	5.30	3.39	12.602	.002*	.364
	Top	5.15	3.25			
Significant Interactions						
Technique x Technique Order				11.136	.003*	.336
Start Button x Technique Order				5.224	.032*	.192
Technique x Action				8.411	.008*	.277
Action x Final Position Type				3.961	.032†	.153
Action x Block				4.635	.043*	.174
Final Position Type x Block				10.101	.000*	.315

4.4.2 Main Effects

There was a statistically significant main effect of Technique, validating the usefulness of LSO technique for window manipulation on a large screen display. The mean performance advantage is 0.68 seconds per manipulation. As discussed below, there were interactions of Technique with both Technique Order (Section 0) and Action (Section 0).

As expected, there was a statistically significant main effect of Start Button Position on manipulation time. Because a window in the Traditional condition must be dragged by the title-bar, participants must move the cursor further to begin movement when the start button is in the lower position. We had expected to see an interaction between Start Button Position and Technique, because clicking anywhere in the window begins a move action in the LSO technique so there should be little no difference between the two start positions. While the mean difference in times for the LSO technique (Δ=0.074 s) was smaller than that of the Traditional technique (Δ=0.216 s), the difference was not statistically significant ($p = 0.291$).

There were also significant main effects of Action and Final Position Type. This is expected, as the Move+Resize condition requires strictly more interaction than the Move condition, and the corners, edges, and floating targets are all at different distances from the window's starting position.

There was also a significant main effect of Block, with manipulation time decreasing in the second block for each condition. If we plot manipulation time over all four blocks performed by each participant, there is a continued downward trend; we suspect that performance had not yet plateaued at the end of the experimental trials.

Technique order interactions

There was no main effect of technique order; however, there was an interaction between both Technique Order and Technique and between Technique Order and Start Button Position. In the interaction of Technique Order and Start Button Position, those participants who began with the LSO technique were slower when starting from the

lower position. Presumably, this is because they were unused to the precision required for this movement when they began to use the Traditional technique in the second half; see Section 4.5.1 for a more complete discussion. We argue similarly to explain the Technique Order by Technique interaction: participants who began with LSO were unused to the precision required in the Traditional technique, and their performance suffered when they were required to use the more difficult technique.

Technique x Action

There were main effects for both technique and for action. There was also an interaction between the two. While LSO was somewhat faster for moving windows (Δ=−0.31s), it was much faster for resizing (Δ=−1.06s). This is well supported by the qualitative feedback from our questionnaires; users reported much difficulty resizing windows using the Traditional technique.

Action x Final Position

There was a statistically significant interaction effect between Action and Final Position, which was not hypothesized in advance. The Move+Resize action is slower in corners (μ=7.59s) and at edges (μ=7.47s) than for floating targets (μ=7.02s) because participants would overshoot the target, and the window's resize handles would become inaccessible. When this happened a recovery action was needed, such as moving the window so it was completely visible once again. This did not occur when only moving because the resize handles were not needed, nor did it occur when the target was at a floating final position, as no portion of the window was likely to go off-screen. As all higher-order interactions were non-significant, the Action x Final Position Type interaction generalizes across all other variables.

The effect of this phenomenon was reflected in the questionnaire data; several participants explicitly mentioned the snapping regions when commenting on their preference for the LSO technique when moving and resizing window in the corners and edges of the screen. We examined the use of these snapping regions in the LSO technique and the impact that the use had on manipulation time. For those targets located on the

edge of the screen and in the corners, the snapping technique would be most useful. We classified a trial as having an optimal snapping action (for a corner target, in the matching corner region; for an edge target, in the matching edge region or the adjacent corner regions) or not (snapping to another region, or not snapping at all). As seen in Table 4.2, participants more frequently took appropriate snapping actions for corner targets (48% of corner trials using LSO technique) than for side targets (27%). This is likely due to the fact that for edge targets there was still a need to position the target along the edge, while for corner targets the window gets effectively locked in place by the corner. Snapping appropriately resulted in improved performance times for Move+Resize (Δ=−0.17s) actions to an edge target and both Move (Δ=−0.21s) and Move+Resize (Δ=−1.11s) in corner targets; however, there was no performance gain for a Move to the edge.

Table 4.2 Mean manipulation times when snapping was or was not used optimally.

Action	Target Position	Optimal?	Frequency	μ (s)	S.D. (s)
Move	Edge	Yes	108	2.97	0.91
		No	276	2.96	1.56
	Corner	Yes	169	2.86	2.37
		No	215	3.07	1.72
Move + Resize	Edge	Yes	98	6.91	2.94
		No	286	7.08	3.85
	Corner	Yes	199	6.52	3.33
		No	185	7.63	4.78

Final Position Type x Block and Action x Block

The Final Position Type x Block interaction, graphed in Figure 4.7, shows a distinct improvement in fitting windows to the corners, far greater than the improvement attributable solely to block. In the Action x Block interaction, participants showed a larger improvement for Move+Resize trials (Δ=−0.47s) than for Move trials (Δ=−0.08s).

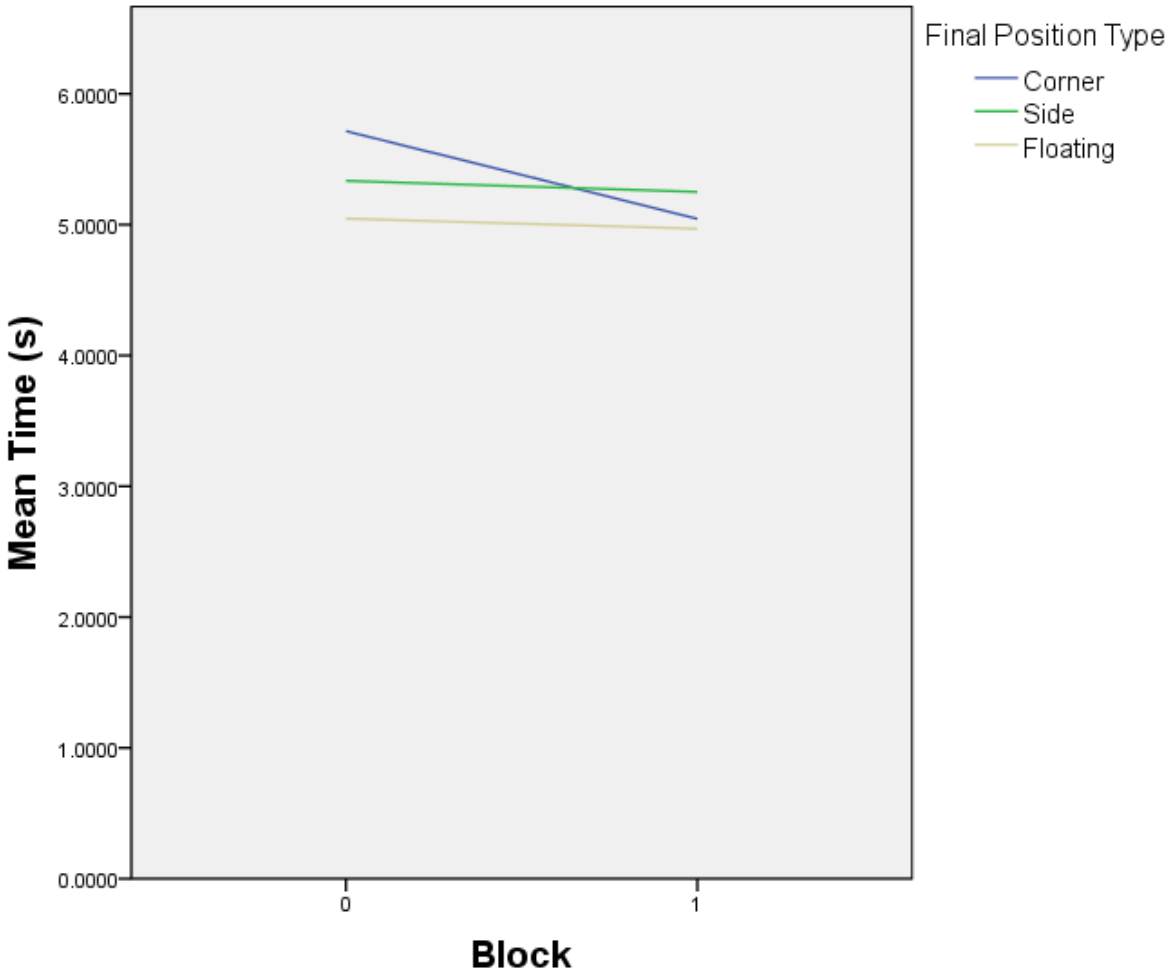

Figure 4.7 Graph of Final Position Type x Block interaction.

4.4.3 Questionnaire Data

Participants filled out questionnaires with ratings and rankings.

Post-technique results

After each technique, participants rated the technique according to its ease of use overall and for each Action/Final Position Type combination. The results are shown in Table 4.3, and the questions used a 5-point Likert scale where "1" denoted "hard" and "5" denoted "easy." LSO was rated easier to use than the Traditional technique for both Move and Move+Resize). Non-parametric Wilcoxon matched-pairs signed-ranks tests (Z-score

reported), with a Bonferroni adjustment of the significance level to .007 to compensate for multiple comparisons, revealed this difference was significant except for free-floating target locations (marginally significant for Move+Resize to floating targets).

Table 4.3 Participant ratings of window manipulation techniques for ease of use.

P<.007*		**Traditional**		**LSO**		**Test stats**	
Action	**Target**	**μ**	**S.D.**	**μ**	**S.D.**	**Z**	**P**
Move	Float	4.33	.963	4.67	.637	1.604	.109
	Edge	3.46	1.062	4.54	.509	3.696	**.000***
	Corner	3.42	1.176	4.46	.833	3.452	**.001***
Move + Resize	Float	3.42	.830	4.17	1.007	2.452	.014
	Edge	3.17	1.049	3.92	.929	3.366	**.001***
	Corner	3.13	1.191	4.00	1.103	3.827	**.000***
Overall		3.21	.448	4.29	.690	4.245	**.000***

Post-session preferences

After the participants had completed both techniques, they completed a questionnaire that asked them to rank the techniques according to their overall preference, as well as to indicate their preferred technique for the various moving/resizing target locations (floating, edge, corner). The vast majority (23/24) of participants preferred the LSO technique over the Traditional window management technique; and in each specific condition at least 19/24 (79.2%) participants preferred the LSO technique, as shown in Figure 4.8. Chi-square analyses, with a Bonferroni adjustment of the significance level to .007 to compensate for multiple comparisons, revealed that this preference was significant in all cases ($p<.002$).

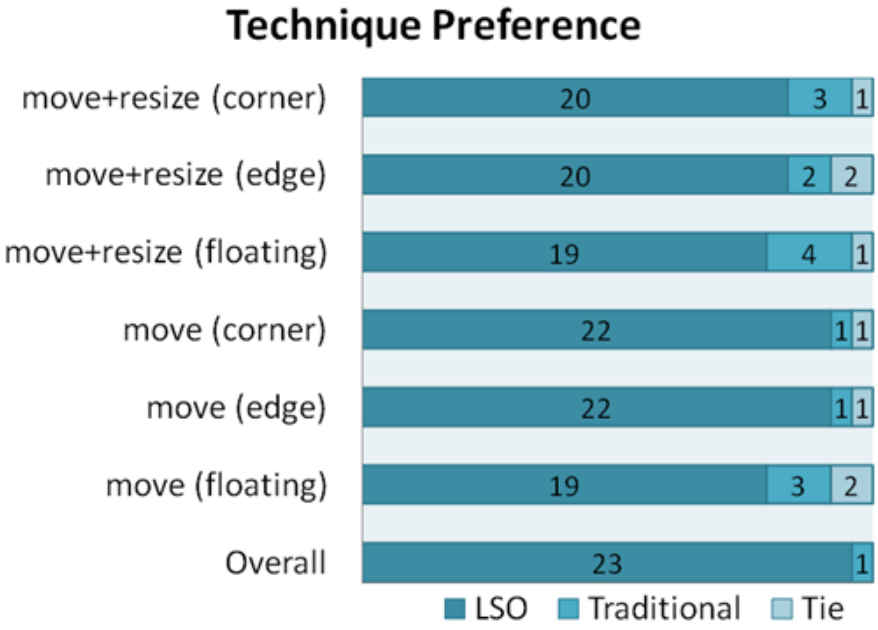

Figure 4.8 Preferences for the window manipulation techniques.

An examination of individual rankings for each combination of action and target location revealed that 15 participants consistently indicated a preference for LSO and 1 consistently indicated a preference for the traditional technique. Interestingly, the single participant who preferred the traditional technique was the only participant who was not a regular mouse user. This participant was also an outlier in terms of mean window manipulation time.

We examined participant comments for the reasons behind their preferences. For the Move and Move+Resize to the floating and corner target positions, about one third of participants explicitly mentioned the snap-to feature of the LSO technique. The ability to click anywhere in the LSO technique was mentioned for all targets (i.e., 16/19 for Move-floating, 7/22 for Move-corner), with some noting that the technique was "less rigid" than the traditional technique and that LSO "allows traditional manipulation if wanted." While most participants did not explicitly comment on the use of two buttons in LSO, the few that did were mixed in their opinions. For example, one said (for Move+Resize-floating) "using right click is much more convenient." However, another consistently made the

comment “using both R&L buttons is confusing.” It was interesting that three of the participants specifically noted that the overloaded left mouse button in the traditional techniques was problematic, as one noted “there were also several occasions when I’d want to move the window via the top bar and would accidentally resize it instead (or vice versa).”

One aspect of the LSO technique that appears to have been problematic was the zooming-resize technique, shown earlier in Figure 4.3. After using LSO, three participants made negative comments about this technique, with one commenting that it should resize more slowly. Two others commented that it was more difficult to shrink a window than to expand it.

Tradeoff: two buttons, larger selection space

We asked participants to choose between using (a) both the left and right mouse buttons with large selection regions (i.e., handles), or (b) only one button with smaller selection regions; 22/24 participants indicated a preference for two buttons and a larger selection space. Having larger handles was the primary concern for six participants. While one commented, “I don’t like using two buttons, but I like a larger target more,” five others felt that using two buttons was not much more complex, although some training might be needed. A few participants considered the ramifications of using the right mouse button for resizing during normal use, with one commenting that their choice depended upon the importance of window manipulation to the task and another saying, “I think we can right click for resizing and design another key to replace right click’s tasks [context menu].”

4.5 Discussion

In this section we present discussion related specifically to the LSO technique and the experiment described in this chapter. A broader discussion of the LACOME system as a whole follows, in Chapter 5.

The LSO technique outperformed the Traditional window management technique for both moving and resizing, regardless of the start button position or the final target

location. The ability to click anywhere in the window when moving reduced the distance participants had to travel and increased the target size. The performance advantage was greater for resizing than for moving, in part due to the snapping regions in LSO that constrain users from moving a window past the screen edge when appropriately snapped. Furthermore, the use of the right mouse button helped prevent participants from inadvertently selecting the wrong handle and moving rather than resizing or vice versa. Participants subjectively rated LSO as being easier to use than the Traditional technique and they preferred it for use on a large screen display.

4.5.1 LSO Beyond Large Collaborative Displays

While our LSO technique was designed specifically for collaborative use of large displays, it may also be suitable for general use on personal computers, particularly for those users with larger monitors or secondary displays who may more frequently arrange multiple visible windows on the screen (Hutchings & Stasko, 2004).

We developed the LSO technique to improve window manipulation on large displays. We have since learned that some of the window managers in distributions of Linux provide the similar technique of clicking anywhere within a window for moving (ALT+drag with left mouse button) (Compiz, 2010a) (Gnome, 2010a) and resizing (ALT+drag with middle mouse button) windows (Compiz, 2010b). We could find no evaluation of these techniques in the literature; however, the success of our technique during this study provides validation for this approach.

In order to utilize the LSO technique in other systems, we would have to provide some mechanism for a mode switch; the successful integration of modifier keys for this purpose in the Linux window managers shows that this approach to mode switching works. As our questionnaire data revealed, our participants were quite willing to complicate the manipulation process by using two different mouse buttons to initiate the correct action in return for the larger selection areas. Of course, they were not required to perform mode switches during the experiment, so further study is required to determine if this switch would prove to be a hindrance in a real-world system.

4.5.2 Learning and Transfer Effects

We found a significant effect of Block and significant interactions involving Technique Order, which indicate learning and transfer effects. Despite the apparent simplicity of the task, it appears that participants continued to improve overall across all four blocks; as a methodological issue, one should expect a longer acclimation period when users must adapt to using a large screen, even when using a fairly familiar technique such as our Traditional technique.

In this study, we used technique orderings of A-B and B-A (A=Traditional, B=LSO). To establish a better baseline of learning curves, we could also use A-A and B-B orderings; that is, have participants use one technique for four consecutive blocks. This would help us separate transfer effects from learning effects.

As shown in Figure 4.9, participants who began with the Traditional technique did extremely well when they switched to the LSO technique; those who started with LSO showed only minor improvements in the Traditional technique compared to those who started with Traditional. Because the questionnaire data showed that participants found the Traditional technique harder, we believe that those who started with Traditional were expending much more practice effort. After using the narrow edges and minute corners of the Traditional technique, these participants found the large regions of the LSO technique very easy to use. Those who began with the easier LSO technique did not receive this practice benefit, and so showed only minor improvement.

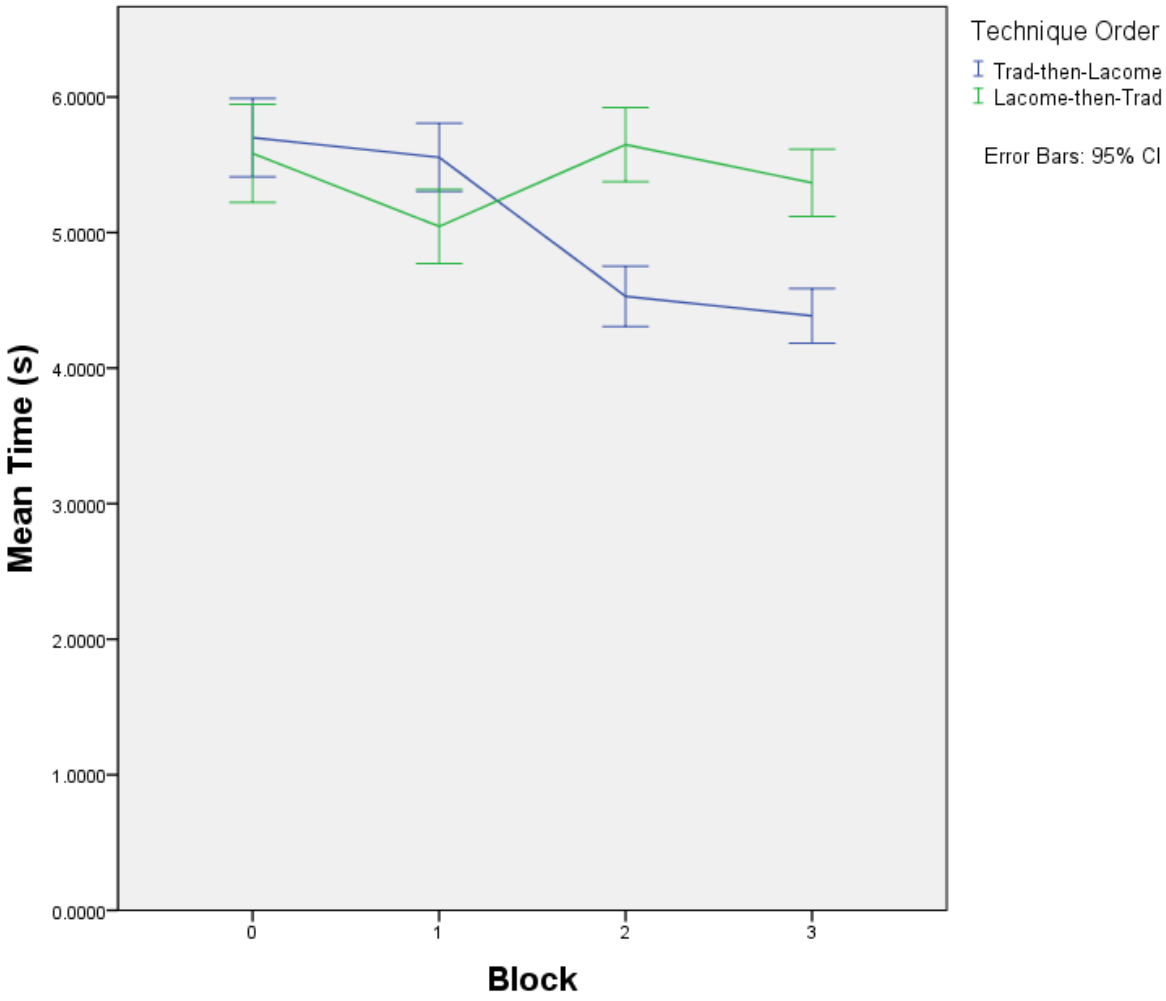

Figure 4.9 A plot of learning effects based on technique order. Each line has two Traditional blocks and two LSO blocks; the four higher points are Traditional and the four lower are LSO. Learning effects are apparent in each pair of blocks. Those who start with Traditional do much better in LSO, as shown by the blue line from top-left to bottom-right. The mean time is in seconds.

This unexpected asymmetric transfer effect calls into question the validity of using a within-subjects experiment design. However, examining only the first two blocks in Figure 8, which occur before any transfer effects can happen, we see that participants using LSO already see statistically significant gains. Additionally, using a within-subjects design allowed us to collect data such as preference information, which would have been impossible in a between-subjects experiment.

4.5.3 Limitations

Our study was an initial look at evaluating the LSO window management technique. It was intended to show that practical and significant gains could be made over a traditional windowing system. However, windowing systems are used in many ways and in many contexts, and no single study can demonstrate the efficacy of the LSO technique across all of them. We next discuss a number of limitations to our evaluation that must be resolved through further study.

Users found that they were unable to achieve the accuracy required for the experimental task using the 'zooming' resize method, and the feature was consequently little used. We believe that this technique remains promising for real-world situations and should be validated with different tasks, or different methodologies. Additionally, the required precision was determined through pilot testing and not through analogy to real-world data. In some contexts, users may be perfectly willingly to place windows with less accuracy than was required of them in this experiment.

Our study was conducted with single users who were squarely facing the display. While we believe that our results should hold for multiple users and alternate seating positions, further validation is required for collaborative meeting environments.

The level of accuracy required in this experiment may have been higher than users typically require in practice. While users strongly preferred the LSO technique for the task given, it remains to be seen if this result holds during long-term real-world usage when the moving and resizing actions are interspersed with other tasks.

All windows in this study were non-semantic; the improvements offered by the LSO technique should translate directly to systems where windows are primarily viewed, moved, and resized. If the LSO technique is to be used in more general systems where users frequently interact with the semantic contents of a window, an effective mode toggle must be found and the performance cost of performing this mode switch must be

measured. Our future work will include a comparative study that better evaluates the impact of mode switching.

5 Conclusion and Future Work

In this chapter we identify future research pathways that promise to further evaluate the LACOME system and how we might apply it to alternate domains. We then present some final conclusions.

5.1 Future Work

LACOME, in its current form, provides good support for collocated collaborative meetings. However, a great deal of future work is required to validate the current system and extend it for new uses.

While the experiment described in Chapter 4 validates the basic usability of the LSO window manipulation technique, more validation is required both for the LSO technique and for the LACOME system as a whole. Additionally, we imagine a number of alternate scenarios to which LACOME could be applied, including classroom settings, large lecture halls, and distributed meeting environments. Not all meetings are collaborative. Access control could be used to selectively give users access to the shared display space. Large-screen displays may also have a variety of form factors. Some environments could have multiple screens. The current LACOME system, which relies on standard VNC servers, does not support shared audio. Positional audio could be a useful extension. Finally, different facilities may have different networking requirements, and in some situations the use of a proxy server may be required. We briefly describe possible future work in each of these areas.

5.1.1 Further Validation of the LSO Technique

The LSO technique, presented in Chapter 4, was designed with particular usage in mind. The user experiment presented in Chapter 4 was only able to test a narrow range of usage.

In particular, that study used only a single participant at a time. LACOME is designed to support collaborative work, which necessarily entails multiple users. Future studies must verify that the performance enhancements found in the initial study continue to hold when several users simultaneously perform window manipulations. Additionally, in the initial study the single user was seated at an ideal viewing angle. When multiple users share the screen some will necessarily be positioned at sub-optimal viewing angles, which could prove to be a significant factor.

An important subtlety of the LSO technique is that it relies heavily on mode-switching. In one mode user inputs cause window manipulations, while in another mode user inputs actually interact with the contents of a window. This mode switching surely adds some cost to user performance. The initial study did not require users to switch modes and may therefore have reported overly optimistic results for the LSO technique. Future studies should include realistic mode switching to determine its true cost.

Finally, the LSO technique includes a "zooming-resize" feature. When a user right-clicks in the central region of a window and drags the mouse up or down, the window is resized about that point. The interaction is familiar to many users, and is similar to the zoom feature of Google Maps or Microsoft's Bing Maps. However, the initial experiment required a high degree of accuracy in both window position and window size and most users avoided using the zooming-resize feature, presumably because the technique alters both size and position simultaneously. We believe that under a more realistic workload the zooming-resize feature is valuable. Future studies should further test this feature to determine its efficacy.

5.1.2 Validation of the Overall LACOME System

While the LSO technique has seen promising early evaluation, the LACOME system as a whole has only seen limited laboratory use. The observation of extensive real-world use of the system is critical for finding usability flaws, identifying new features, and informing future design cycles.

One key aspect of the system requiring further validation is users' initial use of the system. For moderators and system administrators, this chiefly involves understanding the computer networking requirements of LACOME. Even if the system is running in a restrictive networking environment, LACOME can often still be used, although with some limitations. Administrators must be given enough information to make informed decisions regarding modifications of their network settings. Additionally, any limitations imposed by the networking system must be described to end users of the system. It remains to be seen if the LACOME ecosystem, including its documentation, facilitates such knowledge transfer from relatively expert administrators, responsible for setting up LACOME and the networking environment, to the relatively novice end users who actually use LACOME.

Many end users of LACOME are likely to be unfamiliar with computer networking or the computer security tools running on their computers. In particular, it may be necessary for users to allow LACOME access through their personal system firewalls. Future studies should analyze whether the LACOME system is intuitive to novice users and whether the LACOME documentation is sufficient to guide end users through the process of configuring their system settings and VNC server settings.

Additionally, our personal experience is that the use of the LACOME system can dramatically and positively impact the flow of a meeting, allowing more fluid collaboration and the incorporation of multiple digital artifacts spread across multiple computing systems. It is important to fully understand how users adopt the technology and what impact it has on both the meeting process and meeting outcomes.

5.1.3 Access Control

People involved in collocated meetings may share a variety of relationships, not all of which are completely trusting. For this reason, various levels of access control will be required in some settings. There are a number of different actions that could be restricted. The most basic action is simply connecting to the LACOME Server. Currently anyone may connect and use an arbitrary nickname, but in the future users might be required to

set up an account and use a consistent nickname or even use a verified real-world name. Another security concern is the ability to toggle navigation and thereby gain an onscreen cursor. Even if a cursor is not allowed to perform any actions, it may still be used for pointing. One step up is the ability to move, resize and iconify windows. In a very large meeting, such as in a classroom or seminar setting, a troublemaker could disrupt a meeting through malicious window manipulations and it might be difficult to determine, in real-time, which user was the troublemaker. The ability to publish windows could be abused by users publishing inappropriate content. While regular VNC connections can be easily tracked to their owner, reverse VNC connections cannot if the network address is ephemeral. Finally, using VNC, LACOME can give a user the ability to directly control a published desktop. This requires a great deal of trust between users. Robust, authenticated access control would be appropriate.

By default, LACOME does not limit any user actions but instead relies on social norms to mediate the interactions between users. For example, if a user is viewing a window and another user moves a window in front of it, the first user may simply ask the second user to reposition the offending window. Such social norms are not always sufficient, particularly if highly sensitive data is being shared or if users are antagonistic.

If a user does not want any other users to interact with a published desktop, he or she may use the VNC server's own settings to prevent access. However, this scheme does not allow just a subset of users to interact while forbidding others. It is instead a blanket rule applied to all meeting participants. Identity-based access control would allow different users to have different permission levels.

Access control could make use of the notion of a moderator. A moderator is an all-powerful user who can set the permission level of other users. The moderator would set up the LACOME Server ahead of time. Assuming that users are pre-registered, the moderator could allow only specific users to join the meeting. Alternately, the moderator could allow new users or guests to join, but only subject to authentication. Roles could be defined. For example, registered users could have full permissions including publishing

and control, while guests might only have the ability to connect, navigate, and manipulate windows. Additionally, groups could be defined, and individuals or roles could have group-specific permissions. An example where this would be useful would be negotiations or arbitration, where multiple teams with competing interests might be using a LACOME Server to access a shared display.

We can imagine a hypothetical negotiation scenario between two parties. In this case a moderator, responsible only for technical operation of the system, creates three groups and three roles. The three groups are "Arbitrators," "Union," and "Management." The first role is "arbitrator." The arbitrator may connect, navigate, manipulate windows, and publish. A user's role is arbitrator if and only if he or she is a member of the Arbitrators group. The second role is "leader." One leader is chosen for each side of the negotiation, who may connect, navigate, or manipulate his or her group's windows, and publish. The third role is "member," which is assigned to other members of the negotiation teams. Members may connect, navigate, and manipulate and control their own group's windows, but they may not publish. The intent of this scheme is to reduce conflict between opposing parties. No one from the Union may interact with a Management window or vice versa. If they would like a window from the other party moved they may ask the arbitrator or the other party to do it. Because only the leader of each group may publish, when a new window appears all participants know who was responsible and may direct their questions and comments to that individual. Similarly, only the leader may unpublish the windows belonging to his or her group.

Of course, one could design many other roles and groups, or even other access control schemes altogether. The actual implementation of an access control system is left for future work, The behaviour described above is merely a motivating example and an initial design consideration.

5.1.4 Classrooms and Lecture Halls

We imagine extending the use of LACOME to other facilities where large screen displays are used, beyond simple meeting rooms. This entails moving beyond collaborative

meetings and considering different user processes. One setting we have considered is education, particularly in classrooms and lecture halls. Many technological tools for education, such as chalkboards, websites, and large screen displays, still focus primarily on getting information from the instructor to the students. Technological support for getting information from students to the instructor, or to other students, is still relatively poor.

A key feature of LACOME is that any user may publish content to the shared display space, within the limits imposed by access control as discussed in Section 5.1.3. In a classroom setting, this means that not only would the instructor be allowed to publish, but students would be allowed to publish as well. This could be useful in a number of ways. For example, an instructor could ask a question to the class and students could respond visually by publishing their displays, rather than only responding verbally. Alternately, a student with a question could use his or her on-screen cursor to point at content on the shared screen rather than needing to explain verbally the visual content to which they are referring. Finally, a student could spend time preparing a question, including digital media such as diagrams created in a graphics editor like Microsoft Paint or The GIMP and then use LACOME's publishing capability to share it with the class.

In some classroom or lecture settings, social norms may be sufficient to maintain order and allow efficient communication between all parties. In other settings, however, the LACOME system could be open to abuse. In traditional classrooms, students signal that they have questions by raising their hands. The instructor is free to choose whether or not to respond to the question immediately. She may address the question immediately, ignore it altogether, or otherwise signal to the student that she will return to the question later. LACOME, in its current configuration, does not provide strong support for such mechanisms. A student publishing without the instructor's permission, whether intentional or not, could be very distracting for the entire class.

We propose an extension to the access control mechanisms discussed above. Two important innovations are dynamic roles or permissions, and queuing. In the access

control scheme described in Section 5.1.3, roles were pre-configured by a moderator prior to a meeting. In a classroom setting, students move between roles. Most of the time, a student is not permitted to interact with the LACOME system in any way. He or she must ask permission from the instructor. The instructor may switch the student's role to allow pointing (navigation) or to allow publishing. Once finished, the instructor can return the student to the default role and continue the lecture. Queuing helps the instructor manage the permission requests made by students. Just as students may raise their hands at any time, a student may request elevated permissions at any time. The instructor could be presented with a list of students who are requesting permission, in the order in which they made requests. Any student in the queue could be called upon, that is, any student could have their role switched by the instructor. When returned to the default role, the student is removed from the queue.

A preliminary implementation of similar features was completed by Joel Lanir, using a modified version of the LACOME Client, for the MultiPresenter system, described in his dissertation (Lanir, 2009). That system allowed students to publish static images rather than entire interactive desktops, and did not use VNC technology on either the students' machines or on the machine driving the shared display. It did provide integration with Lanir's presentation slide software, allowing students to flip back and forth within the slide deck once they were called upon. Key elements of future work are to implement such features in LACOME, and to evaluate their usage in real-world classroom and lecture settings.

5.1.5 Multiple Screens

Meeting rooms and other collaborative spaces may hold more than one shared display. The LACOME Server currently assumes a single rectangular display and runs in a single window. If multiple monitors happen to be adjacent to each other, the LACOME Server could be stretched across several displays similar to conventional extended desktop capabilities. However, there is currently no support for more complex display geometries.

The Stanford iRoom offered support for fairly arbitrary display topologies with its PointRight software (Johanson, Hutchins, Winograd, & Stone, 2002).

It would be technically straightforward to create additional display windows and render LACOME display elements to them. However, it is less clear how to develop compelling and intuitive interaction designs that will work well across arbitrary geometries. A variety of related work addresses these issues, discussed in Section 1.5.2 and Section 1.5.4. An understanding of how multi-display installations are used and adopted by real-world meeting participants is essential to making informed design decisions in this area.

An additional factor that impacts systems utilizing multiple screens is resolution disparity. In LACOME, content from one device is published to a display on a different device. These displays will often have different resolutions: LACOME assumes that the shared display is of a much higher resolution than the publisher. If this assumption does not hold, LACOME still functions. However, shared displays may appear fuzzy or show distinct aliasing or "jaggies". If a future version of LACOME were to support multiple screens, some of those screens may have very different resolutions. To provide a consistent user experience some aspects of the system which are currently defined in terms of resolution (pixels) might need to be modified to function in terms of size. For example, the size of the snapping regions in the LSO technique should have a consistent spatial size, such as 10cm, on all displays driven by LACOME.

5.1.6 Distributed LACOME

LACOME was designed to support collocated collaborative meetings. Because the system depends heavily on computer networking, adding additional support for distributed collaboration is a realistic option. Simply cloning the displays across two locations would be a simple matter. One need only forward all LACOME messages and VNC packets to each LACOME Server. However, a distributed system brings in security and privacy constraints that do not typically apply to collocated systems, and network bandwidth and delay concerns as well.

For example, in a collocated setting a user may be confident, by means of visual inspection, that any content they place on the shared display is not being recorded by other meeting participants, effectively affording an "eyes-only" level of security. In a distributed system a user cannot know if remote users are using audio-visual recording devices.

In addition, a user of a distributed system cannot know if other parties are physically present on the other side, unless they also connect to the LACOME Server. Unless some member of the remote party can be trusted to give an accurate roster of all attendees, it may not be possible to securely share sensitive information.

If a user in a collocated meeting requires a one-on-one interaction with another user, such as to ask if it is permissible to move or resize the user's window, they may simply gesture or whisper to each other. In a distributed meeting, they must use a communication channel shared between all meeting participants, typically a telephone conference call, or have some other means of communication that is specific to them. One-on-one interaction intrudes on the flow of the meeting and is a distraction for everyone. This issue could be solved through the creation of user-to-user instant messaging, but this raises issues of trust and privacy.

A distributed implementation of LACOME may be well suited to some types of meetings, such as between remote offices of the same company, individual workers working from home, or between researchers at different universities. There are, however, many situations when such a system will not be appropriate, such as when sensitive "eyes-only" information is shared. The access control issues raised in Section 5.1.3 are no doubt of increasing importance in these situations.

5.1.7 Audio

LACOME currently only supports sharing computer desktops. In the future we would like to support the sharing of audio streams as well. There are two motivations for this feature. First, meeting rooms with a dedicated large screen display and LACOME Server

are likely to contain powerful or state-of-the-art sound systems that are much more suitable for group use than are internal laptop speakers. Just as LACOME already helps users move away from a one-user-per-display paradigm, audio sharing would allow them to also move away from a one-user-per-speaker-system paradigm. Second, when sharing audio-visual content it may be more pleasant and less intrusive to have localized sound come from the same direction as the video content. As a window containing a computer desktop is moved from one side of the LACOME Server's display to the other, its audio stream can be re-mixed in stereo or full surround sound in order to follow it. While we have begun to explore the use of Pulseaudio, a cross-platform networked sound server (PulseAudio, 2010), this is still in preliminary stages and is largely left as future work.

5.1.8 Complex Networking

LACOME relies on networking for all of its operations. There are two types of connections: LACOME connections, which connect a LACOME Client to the LACOME Server, and VNC connections, which connect the LACOME Server to a VNC server. In the current implementation of LACOME, the LACOME Client always initiates LACOME connections. As discussed in Section 3.1.2, a thread in the LACOME Server listens for these connections. In contrast, a VNC connection can be initiated by either the LACOME Server or the VNC server.

Background

The most common method of initiating a VNC connection within LACOME is through the LACOME Client. In this method, the user enters the address and port of the VNC server into the appropriate text fields of the LACOME Client and then clicks the Publish button. A dialog box prompts the user to enter the VNC password. This information is sent to the LACOME Server, which initiates a connection to the specified VNC server. The LACOME Server then associates the connection with the user who initiated it. The owner's nickname is displayed in the title-bar of the window, and if the user disconnects from LACOME the VNC connections associated with that user are terminated.

Alternately, a user may initiate a connection from the VNC server, using what is known as a "reverse connection." The exact mechanism depends on the particular VNC server chosen. Some offer a GUI element while others require the use of command line options. The only input required is the address of the listening VNC client. In these systems, the LACOME Server acts as a VNC client, so the user merely needs to provide the address of the LACOME Server. In this case, there is no way to determine who initiated the VNC connection, so the connection has no user associated with it. The user must take special care to ensure that they eventually cease publishing a VNC server when using a reverse connection, because LACOME will not terminate the connection automatically. As future work, it would be straightforward to allow users to associate themselves with ownerless VNC connections using onscreen GUI elements on the LACOME Server so they can better manage the connections.

As a security enhancement, many computer systems and networks forbid incoming network traffic. This is often or even usually the case with computers used by end-users, which is exactly the user group that is likely to want to run VNC servers and the LACOME Client. Communication may be restricted at more than one level. For example, a user may be blocked by both a firewall running on his or her own computer, such as the Windows Firewall, or by a firewall or routing settings located on the network itself. Depending on the nature of the organization, end users may or may not be permitted to alter their system configurations.

Possible network configurations

A grid of connection possibilities is shown in Table 5.1. It should be noted that for simplicity it is assumed in this discussion that a machine either allows both VNC and LACOME connections, or it allows neither. It is possible to configure a machine to allow one type of connection and not the other, but such settings are not particularly interesting and are not discussed further.

Table 5.1 Connection types required under various network configurations.

<table>
<tr><td></td><td></td><td colspan="3">LACOME Server</td></tr>
<tr><td></td><td>Permitted connection directions</td><td>In Only</td><td>Out Only</td><td>Both</td></tr>
<tr><td rowspan="3">LACOME Client</td><td>In Only</td><td>A) No connections possible.</td><td colspan="2">B) No LACOME or reverse VNC connections. Regular VNC connections possible but no easy way to initiate them.</td></tr>
<tr><td>Out Only</td><td rowspan="2">C) Reverse VNC connections required, normal LACOME connections.</td><td>D) Proxy server required for both LACOME and reverse VNC connections. Regular VNC connections are impossible.</td><td>E) Reverse VNC connections required, normal LACOME connections.</td></tr>
<tr><td>Both</td><td>F) Proxy server required for LACOME connections and reverse VNC connections; regular VNC connections may be used.</td><td>G) All connections possible.</td></tr>
</table>

If a user is unable to receive incoming connections then regular VNC connections are impossible and reverse connections must be used. If a user cannot create outgoing connections, LACOME connections and reverse VNC connections are impossible. Fortunately, outgoing connections are permitted in most networking situations.

Configurations that support full functionality

Cell G: If both the LACOME Server and the LACOME Client accept both incoming and outgoing connections, no special restrictions apply and all types of connections may be used freely. This is shown in cell “G.”

Configurations that require the use of a proxy server

A proxy server is a computer that receives network traffic and re-transmits it virtually unaltered, in order to overcome networking difficulties.

Cell D: If both the LACOME Client and LACOME Server support only outgoing connections, as in cell "D," then communication may be achieved through the use of a proxy. The implementation of a proxy server is left for future work. In this case the LACOME Server connects to the proxy server at start-up. LACOME Clients connect to a proxy server in the same way they connect to a LACOME Server, by entering its address and port and pressing the "Connect" button. Once these socket connections have been established the proxy can forward messages from one machine to the other. Although the connections are outgoing from the perspective of both the LACOME Server and the LACOME Clients, bidirectional communication can occur over the socket. Because a LACOME Client in this scenario cannot accept incoming connections only reverse VNC connections may be used, and they must travel through the proxy server. This is potentially a very common scenario. Many systems and networks default to blocking incoming connections. Extra effort must be applied to allow incoming connections and sometimes it is simply not possible.

Cell F: If the LACOME Client supports both incoming and outgoing connections but the LACOME Server supports only outgoing connections, as shown in cell "F," a proxy must be used for LACOME connections and reverse VNC connections but no proxy is required for regular VNC connections. Because VNC traffic uses much higher bandwidth than LACOME traffic, such a scheme greatly reduces congestion on the proxy server, and network congestion more generally. This is a fairly uncommon scenario. Because a LACOME Server is often used in a dedicated installation, it is more likely to have had the extra effort expended to open firewall ports. Additionally, a LACOME Server is typically set up in advance of a meeting, so there may be adequate time to modify the network and system settings when this is permitted. LACOME Clients, on the other hand,

are often transient and might only use a LACOME Server once. They typically only connect at the start of a meeting, so there is little opportunity to reconfigure network settings.

Configurations that necessitate reverse VNC connections

Cells C and E: A LACOME Server is more likely to support both incoming and outgoing connections than a LACOME Client. A LACOME Client is likely to only support outgoing connections. This situation is shown in cell "E." LACOME connections may be established normally, but regular VNC connections cannot be used. Reverse VNC connections must be used instead. The same conditions hold for the cell "C."

Difficult or impossible configurations

Cell A: If both the LACOME Client and the LACOME Server accept only incoming connections, as in cell "A," communication between the two is impossible (unless initiated by a third party).

Cell B: If the LACOME Server is unable to receive incoming connections, LACOME connections and reverse VNC connections are impossible. Regular VNC connections would be possible, except that they cannot be initiated without a LACOME connection unless the LACOME Server is hard-coded to make the connection or if the LACOME Server accepts direct input from users. This scenario is shown in Cell "B" of Table 2.1 If the LACOME Server cannot make outgoing connections, regular VNC connections are impossible and reverse VNC connections must be used instead.

5.2 Conclusions

LACOME was originally motivated by a need to support collocated collaborative meetings. The support mechanism it provides allows users to publish and share their personal computer displays onto a shared display space. The envisaged scenario takes place in a typical meeting room, in either a professional or an academic workplace, augmented with a large, high-resolution shared display visible to all occupants.

We believe that LACOME adequately supports this scenario. The installation process for the new LACOME Server has been greatly simplified and the LACOME Server is now platform independent, so it can be used in diverse computing environments. The new LACOME Client is even simpler to use. It can be launched directly from a webpage using Java Web Start, so it need not even be installed by end users. The process of initiating either LACOME or VNC connections has been simplified by the auto-detection of IP addresses. No longer do users need to locate these IP addresses using external tools. These features allow new users to begin using LACOME right away and they reduce training overhead involved with using the system.

We have considered the usability of the LACOME system not only during setup and configuration but also during usage. By doing a better job of keeping the user's system cursor trapped in the yellow box of the LACOME Client while navigating, we prevent confusing error conditions where mouse focus is lost. New interactive widgets such as the IP address display and Return to Desktop button allow more complex interactions with the LACOME Server through a graphical user interface. The extensible UI framework supports the creation of other widgets in the future. Guaranteed rendering performance helps keep animation smooth even when there is a heavy VNC load. Finally, the new LSO window manipulation technique helps users move and resize windows more easily and supports the placement of windows in corners or against edges with its "snapping" feature. A controlled user study showed that the LSO technique was faster to use and preferred by study participants when compared to a traditional window manipulation technique. We believe the LSO technique makes window manipulation much easier on any large display.

Future work will further evaluate the efficacy of LACOME for fostering collaboration during collocated meetings. With the addition of new features, LACOME can be applied to other domains such as meetings with access control issues, educational settings such as classrooms and lecture halls, distributed meetings, and meeting environments with complex display geometries.

Bibliography

Amershi, S., & Morris, M. (2008). CoSearch: a system for co-located collaborative web search. *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, (pp. 1647-1656).

Andrews, C., Endert, A., & North, C. (2010). Space to think: large high-resolution displays for sensemaking. *Proceedings of the 28th international conference on Human factors in computing systems*, (pp. 55-64).

Anoto Group. (2010). Retrieved August 30, 2010, from Anoto: http://www.anoto.com

Apple Inc. (2008, May 6). *Java for Mac OS X 10.5 Update 1 adds support for Java SE 6*. Retrieved July 9, 2010, from Apple.com: http://support.apple.com/kb/ht1856

Apple Inc. (2010). *Apple Human Interface Guidelines: Windows*. Retrieved August 10, 2010, from Apple Developer: http://developer.apple.com/mac/library/documentation/UserExperience/Conceptual/AppleHIGuidelines/XHIGWindows/XHIGWindows.htm

Argue, R. (2007). *Advanced Multi-Display Configuration and Connectivity.* Master's Thesis, Dalhousie University.

Benko, H., Morris, M., Brush, A., & Wilson, A. (2009). *Insights on Interactive Tabletops: A Survey of Researchers and Developers.* Microsoft Research.

Bi, X., & Balakrishnan, R. (2009). Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. *Proc. CHI '09*, (pp. 1005-1014).

Biehl, J., & Bailey, B. (2004). ARIS: an interface for application relocation in an interactive space. *Proceedings of Graphics interface 2004*, (pp. 107-116).

Biehl, J., Baker, W., Bailey, B., Tan, D., Inkpen, K., & Czerwinski, M. (2008). IMPROMPTU: A new interaction framework for supporting collaboration in Multiple Display Environments and it's field evaluation for co-located software development. *Proc. CHI '08*, (pp. 939-948).

Bisgaard, J., Heise, M., & Steffensen, C. (2009). A Literature Survey of Single Display Groupware Interaction for Large Screen Collaboration. *Information Systems Research*.

Booth, K. S., Fisher, B. D., Lin, C. J., & Argue, R. (2002). The "mighty mouse" multi-screen collaboration tool. *Proceedings of the 15th annual ACM symposium on User interface software and technology - UIST '02*, (pp. 209-212).

Bury, K., & Darnell, M. (1986). Window Management in Interactive Computer Systems. *SIGCHI Bul. 18:2*, (pp. 65-66).

Buxton, W., Fitzmaurice, G., Balakrishnan, R., & Kurtenbach, G. (2000). Large displays in automotive design. *IEEE Computer Graphics and Applications, 20*(4), pp. 68-75.

CERT. (2010, June 23). *MSC03-J. Never hardcode sensitive information*. Retrieved July 09, 2010, from CERT Secure Coding Standards: https://www.securecoding.cert.org/confluence/display/java/MSC03-J.+Never+hardcode+sensitive+information

Chapuis, O., & Roussel, N. (2005). Metisse is not a 3D desktop! *Proc. UIST '05*, (pp. 13-22).

Chapuis, O., & Roussel, N. (2007). Copy-and-paste between overlapping windows. *Proc. CHI '07*, (pp. 201-210).

Compiz. (2010a). *Plugins/Move*. Retrieved July 22, 2010, from Compiz.org: http://wiki.compiz.org/Plugins/Move

Compiz. (2010b). *Plugins/Resize*. Retrieved July 22, 2010, from Compiz.org: http://wiki.compiz.org/Plugins/Resize

Cygwin. (2010). *Cygwin Information and Installation*. Retrieved August 22, 2010, from Cygwin.com: http://www.cygwin.com/

Czerwinski, M., Robertson, G., Meyers, B., Smith, G., Robbins, D., & Tan, D. (2006). Large Display Research Overview. *Extended Abstracts of CHI '06,* (pp. 69-74).

Dietz, P., & Leigh, D. (2001). DiamondTouch: a multi-user touch technology. *Proceedings of the 14th annual ACM symposium on User interface software and technology* (pp. 219-226). ACM.

Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., et al. (1992). Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. *Proceedings of the SIGCHI conference on Human factors in computing systems,* (pp. 599-607).

Gartner, Inc. (2009). *Gartner says PC Industry will suffer sharpest unit decline in history in 2009.* Press Release.

Gnome. (2010a). *Manipulating Windows*. Retrieved July 22, 2010, from Gnome User Guide: http://library.gnome.org/users/user-guide/stable/windows-manipulating.html.en

Gnome. (2010b). *Statusbars*. Retrieved July 22, 2010, from Gnome Documentation Library: http://library.gnome.org/devel/hig-book/stable/controls-status-bars.html.en

Google Inc. (2010). Retrieved August 15, 2010, from Google Maps: http://maps.google.com/

Ha, V., Inkpen, K., Wallace, J., & Ziola, R. (2006). Swordfish: user tailored workspaces in multi-display environments. *CHI'06 extended abstracts on Human factors in computing systems*, (pp. 1487-1492).

Haller, M., Leitner, J., Seifried, T., Wallace, J., Scott, S., Richter, C., et al. (2010). The NiCE discussion room: integrating paper and digital media to support co-located group meetings. *Proceedings of the 28th international conference on Human factors in computing systems*, (pp. 609-618).

Han, J. (2005). Low-cost multi-touch sensing through frustrated total internal reflection. *Proceedings of the 18th annual ACM symposium on User interface software and technology*, (pp. 115-118).

Han, J. (2006). Multi-touch interaction wall. *ACM SIGGRAPH 2006 Emerging technologies on - SIGGRAPH '06*, (p. 25).

Haraty, M., Nobarany, S., DiPaola, S., & Fisher, B. (2009). AdWiL: adaptive windows layout manager. *Extended Abstracts of CHI '09*, (pp. 4177-4182).

Heimer, K., Ramachandran, D., Pal, J., Brewer, E., & Parikh, T. (2009). Metamouse: Multiple mice for legacy applications. *2009 International Conference on Information and Communication Technologies and Development (ICTD)*, (p. 490).

Hilliges, O., Izadi, S., Wilson, A., Hodges, S., Garcia-Mendoza, A., & Butz, A. (2009). Interactions in the air: adding further depth to interactive tabletops. *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, (pp. 139-148).

Hoffmann, R., Baudisch, P., & Weld, D. S. (2008). Evaluating visual cues for window switching on large screens. *Proc. CHI '08*, (pp. 929-938).

Hutchings, D. R., & Stasko, J. (2004a). Revisiting display space management: understanding current practice to inform next-generation design. *Proc. of GI '04*, (pp. 127-134).

Hutchings, D. R., & Stasko, J. (2004b). Shrinking window operations for expanding display space. *Proc. AVI '04*, (pp. 350-353).

Hutchings, D., & Stasko, J. (2002). QuickSpace: New Operations for the Desktop Metaphor. *Extended Abstracts CHI '02*, (pp. 802-803).

Hutterer, P., & Thomas, B. (2007). Groupware support in the windowing system. *Proceedings of the eight Australasian conference on User interface*, *64*, pp. 39-46.

Inkpen, K., McGrenere, J., Booth, K., & Klawe, M. (1997). Turn-taking protocols for mouse-driven collaborative environments. *Proceedings of Graphics Interface*, (pp. 138-145).

Izadi, S., Hodges, S., Butler, A., Rrustemi, A., & Buxton, B. (2007). ThinSight: integrated optical multi-touch sensing through thin form-factor displays. *Proceedings of the 2007 workshop on Emerging displays technologies: images and beyond: the future of displays and interacton.*

Jacucci, G., Morrison, A., Richard, G., Kleimola, J., Peltonen, P., Parisi, L., et al. (2010). Worlds of information: designing for engagement at a public multi-touch display. *Proceedings of the 28th international conference on Human factors in computing systems*, (pp. 2267-2276).

Johanson, B., Hutchins, G., Winograd, T., & Stone, M. (2002). PointRight: experience with flexible input redirection in interactive workspaces. *Proceedings of the 15th annual ACM symposium on User interface software and technology*, (pp. 227-234).

Kandogan, E., & Shneiderman, B. (1996). Elastic windows: improved spatial layout and rapid multiple window operations. *Proc. AVI '96*, (pp. 29-38).

Khan, A., Fitzmaurice, G., Almeida, D., Burtnyk, N., & Kurtenbach, G. (2004). A remote control interface for large displays. *Proceedings of the 17th annual ACM symposium on User interface software and technology*, (pp. 127-136).

Kruger, R., Carpendale, S., Scott, S., & Greenberg, S. (2003). How people use orientation on tables: comprehension, coordination and communication. *Proceedings of the 2003 international ACM SIGGROUP Conference on Supporting Group Work*, (pp. 369-378).

Lanir, J. (2009). *A paradigm for classroom presentations on large, high-resolution displays.* Ph.D. Dissertation, University of British Columbia, Computer Science.

Liu, Z. (2007). *Lacome: A Cross-platform Multi-user Collaboration System for a Shared Large Display.* Master's Thesis, Dept. of Computer Science, University of British Columbia.

MacKenzie, R., Hawkey, K., Perswain, P., & Booth, K. (2010). *Evaluating Two Window Manipulation Techniques on a Large Screen Display*. Technical Report, University of British Columbia, Department of Computer Science.

Mantei, M. (1988). Capturing the capture concepts: a case study in the design of computer-supported meeting environments. *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, (pp. 257-270).

Microsoft. (2010a). *About Windows*. Retrieved July 22, 2010, from Windows User Experience Interaction Guidelines: http://msdn.microsoft.com/en-us/library/ms632597 (VS.85).aspx

Microsoft. (2010b). *Windows MultiPoint Mouse Software Development Kit*. Retrieved August 13, 2010, from Microsoft.com: http://www.microsoft.com/multipoint/mouse-sdk/

Microsoft. (2010c). *Windows Multipoint Server 2010 Home Page*. Retrieved August 13, 2010, from Microsoft.com: http://www.microsoft.com/windows/multipoint/default.aspx

Microsoft. (2010d). Retrieved July 22, 2010, from Windows User Experience Interaction Guidelines: Window Management: msdn.microsoft.com/en-us/library/aa511262.aspx

Microsoft. (2010e). Retrieved August 15, 2010, from Bing Maps: http://www.bing.com/maps/

Morana, M. (2009, January 17). *Java Security: Why Not To Use String Objects For Storing Secrets*. Retrieved July 09, 2010, from Writing Secure Software: http://securesoftware.blogspot.com/2009/01/java-security-why-not-to-use-string.html

Moraveji, N., Inkpen, K., Cutrell, E., & Balakrishnan, R. (2009). A mischief of mice: examining children's performance in single display groupware systems with 1 to 32 mice. *Proc. CHI '09*, (pp. 2157-2166).

Moraveji, N., Lindgren, R., & Pea, R. (2009). Organized mischief: comparing shared and private displays on a collaborative learning task. *Proceedings of the 9th international conference on Computer supported collaborative learning*, *2*, pp. 65-67.

Morrison, G. (2009). *Patent No. WO/2009/012586.*

Myers, B., Stiel, H., & Gargiulo, R. (1998). Collaboration using multiple PDAs connected to a PC. *Proceedings of the 1998 ACM conference on Computer supported cooperative work - CSCW '98*, (pp. 285-294).

Nacenta, M., Sakurai, S., Yamaguchi, T., Miki, Y., Itoh, Y., Kitamura, Y., et al. (2007). E-conic: a perspective-aware interface for multi-display environments. *Proc. UIST '07*, (pp. 279-288).

Nacenta, M., Sallam, S., Champoux, B., Subramanian, S., & Gutwin, C. (2006). Perspective cursor: perspective-based interaction for multi-display environments. *Proceedings of the SIGCHI conference on Human Factors in computing systems*, (pp. 289-298).

Nakashima, K., Machida, T., Kiyokawa, K., & Takemura, H. (2005). A 2D-3D integrated environment for cooperative work. *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '05*, (pp. 16-22).

NetApplications.com. (2010, July 13). Retrieved July 13, 2010, from Operating System Market Share: http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8

Ni, T., Bowman, D., & Chen, J. (2006). Increased Display Size and Resolution Improve Task Performance in Information-rich Virtual Environments. *Proc. of GI '06*, (pp. 139-146).

Oracle. (2010a). *Java Platform, Standard Edition 6 API Specification*. Retrieved July 10, 2010, from Oracle.com: http://download.oracle.com/docs/cd/E17409_01/javase/6/docs/api/

Oracle. (2010b). *JavaTM Cryptography Architecture Standard Algorithm Name Documentation for JavaTM Platform Standard Edition 6*. Retrieved July 09, 2010, from Oracle Java SE Documentation: http://download.oracle.com/docs/cd/E17409_01/javase/6/docs/technotes/guides/security/StandardNames.html

Oracle. (2010c). *Networking IPv6 User Guide for JDK/JRE 5.0*. Retrieved July 22, 2010, from Oracle.com: http://download.oracle.com/docs/cd/E17409_01/javase/6/docs/technotes/guides/net/ipv6_guide/index.html

Peltonen, P., Kurvinen, E., Salovaara, A., Jacucci, G., Ilmonen, T., Evans, J., et al. (2008). It's Mine, Don't Touch!: interactions at a large multi-touch display in a city centre. *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, (pp. 1285-1294).

Piper, A., O'Brien, E., Morris, M., & Winograd, T. (2006). SIDES: a cooperative tabletop computer game for social skills development. *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* , (pp. 1-10).

Plaue, C., & Stasko, J. (2009). Presence & placement: exploring the benefits of multiple shared displays on an intellective sensemaking task. *Proceedings of the ACM 2009 international conference on Supporting group work*, (pp. 179-188).

Press, L. (1993). Before the Altair: the history of personal computing. *Communications of the ACM, 36*(9), pp. 27-33.

PulseAudio. (2010). *PulseAudio*. Retrieved July 13, 2010, from PulseAudio.org: http://www.pulseaudio.org/

Rekimoto, J., & Saitoh, M. (1999). Augmented surfaces: a spatially continuous work space for hybrid computing environments. *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, (pp. 378-385).

Richardson, T., Stafford-Fraser, Q., Wood, K., & Hopper, A. (1998). Virtual network computing. *IEEE Internet Computing, 2*(1), pp. 33-38.

Robertson, G., Czerwinski, M., Baudisch, P., Meyers, B., Robbins, D., Smith, G., et al. (2005). Large Display User Experience. *IEEE Computer Graphics and Applications, 25*.

Shen, C., Vernier, F., Forlines, C., & Ringel, M. (2004). DiamondSpin: an extensible toolkit for around-the-table interaction. *Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 167-174).

Sony Computer Entertainment America. (2010). *Playstation 3 Features*. Retrieved August 14, 2010, from Playstation.com: http://us.playstation.com/ps3/features/

Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S., & Suchman, L. (1987). Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Communications of the ACM, 30*(1), pp. 32-47.

Stewart, J., Bederson, B., & Druin, A. (1999). Single Display Groupware: A Model for Co-present Collaboration. *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, (pp. 286-293).

Streitz, N., Geißler, J., Holmer, T., Konomi, S., Muller-Tomfelde, C., Reischl, W., et al. (1999). i-LAND: an interactive landscape for creativity and innovation. *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, (pp. 120-127).

Sugawara, K., & Maruta, R. (2009). A novel intuitive GUI method for user-friendly operation. *Knowledge-Based Systems Vol. 22*, (pp. 235-246).

Swaminathan, K., & Sato, S. (1997). Interaction Design for Large Displays. *interactions, 4*(1), pp. 15-24.

Tan, D., Meyers, B., & Czerwinski, M. (2004). WinCuts: manipulating arbitrary window regions for more effective use of screen space. *CHI'04 extended abstracts on Human factors in computing systems*, (pp. 1525-1528).

Tse, E., Greenberg, S., & Shen, C. (2006). GSI demo: multiuser gesture/speech interaction over digital tables by wrapping single user applications. *Proceedings of the 8th international conference on Multimodal interfaces*, (pp. 76-83).

Wallace, G., Bi, P., Li, K., & Anshus, O. (2004). *A MultiCursor X Window Manager Supporting Control Room Collaboration.* Princeton University.

Wigdor, D., Jiang, H., Forlines, C., Borkin, M., & Shen, C. (2009). WeSpace: The Design Development and Deployment of a Walk-up and Share Multi-surface Visual Collaboration System. *Proc. CHI '09*, (pp. 1237-1246).

Appendices

Appendix A UBC Research Ethics Board Certificate of Approval

The University of British Columbia
Office of Research Services
Behavioural Research Ethics Board
Suite 102, 6190 Agronomy Road, Vancouver, B.C. V6T 1Z3

CERTIFICATE OF APPROVAL - MINIMAL RISK

PRINCIPAL INVESTIGATOR:	INSTITUTION / DEPARTMENT:	UBC BREB NUMBER:
Kellogg S. Booth	UBC/Science/Computer Science	H09-01534

INSTITUTION(S) WHERE RESEARCH WILL BE CARRIED OUT:

Institution	Site
UBC	Vancouver (excludes UBC Hospital)

Other locations where the research will be conducted:
N/A

CO-INVESTIGATOR(S):
Russell MacKenzie
Kirstie Ann Hawkey
Presley R. Perswain

SPONSORING AGENCIES:
Natural Sciences and Engineering Research Council of Canada (NSERC) - "ARTIFACT, and NSERC strategic project grant held jointly with Dr. Sheryl Staub-French, Civil Engineering. I cannot figure out how to find this in the list of projects. Perhaps this is because I am not the PI. I would like someone to please tell me how I am supposed to do this so that next time I get it right."

PROJECT TITLE:
Study of Window Management in Lacome, a Local Area Collaborative Meeting Environment

CERTIFICATE EXPIRY DATE: July 16, 2010

DOCUMENTS INCLUDED IN THIS APPROVAL:	DATE APPROVED: July 16, 2009	
Document Name	Version	Date
Protocol:		
Lacome Window Management Research Protocol	1	July 8, 2009
Consent Forms:		
consent form	1.0	June 19, 2009
Advertisements:		
Lacome Window Management recruitment flyer	N/A	July 8, 2009
Lacome Window Management recruitment text	N/A	July 8, 2009
Questionnaire, Questionnaire Cover Letter, Tests:		
Initial Questionnaire	1	July 8, 2009
Post-Condition Questionnare	1	July 8, 2009
Final Questionnare	1	July 8, 2009
Other:		
N/A		

The application for ethical review and the document(s) listed above have been reviewed and the procedures were found to be acceptable on ethical grounds for research involving human subjects.

Approval is issued on behalf of the Behavioural Research Ethics Board and signed electronically by one of the following:

Dr. M. Judith Lynam, Chair
Dr. Ken Craig, Chair
Dr. Jim Rupert, Associate Chair
Dr. Laurie Ford, Associate Chair
Dr. Anita Ho, Associate Chair