Lacome

A Cross-platform Multi-user Collaboration System for a Shared Large Display

by

Zhangbo Liu

B.Eng., The University of Electronic Science and Technology of China, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University of British Columbia

December, 2007

Abstract

Lacome is a multi-user cross-platform system that supports collaboration in a shared large screen display environment. Lacome allows users to share their desktops or application windows using any standard VNC server. It supports multi-user concurrent interaction on the public shared display as well as input redirection so users can control each other's applications. Lacome supports separate types of interaction through a Lacome client for window management tasks on the shared display(move, resize, iconify, deiconify) and for application interactions through the VNC servers. The system architecture provides for *Publishers* that share information and *Navigators* that access information. A Lacome client can have either or both, and can initiate additional *Publishers* on other VNC servers that may not be Lacome clients. Explicit access control policies on both the server side the client side provide a flexible framework for sharing. The architecture builds on standard cross-platform components such as VNC and JRE. Interaction techniques used in the window manager ensure simple and transparent multi-user interactions for managing the shared display space. We illustrate the design and implementation of Lacome and provide insights from initial user experience with the system.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Kellogg S. Booth for his support and encouragement. Not only did he provide motivations and ideas on my research, he has also helped me establish confidence to pursue my life goals. I would like to thank my second reader Dr. Norman Hutchinson, for efficiently providing invaluable comments and suggestions on my thesis. I am also grateful to my lab manager Ron Fussell for his kindly support on everything during the project.

I would like to acknowledge my fellow students and friends for their support and feedback. This research could not have been successfully done without the help from Man Hon Chan, Joel Lanir, Yamin Htun, Jian Xu, Mike Tsai, Haoyuan Hu, Gang Peng, Garth Shoemaker, Tony Tang, Clarence Chan and Sherman Lai.

Last, but not least, my thanks go to my family for their endless support and unconditional love.

The research reported in this thesis was supported by The University of British Columbia and by the Natural Sciences and Engineering Research Council of Canada under the Discovery Grant program and the Strategic Network Grants program as a component of NECTAR, the Network for Effective Collaboration Technologies through Advanced Research.

Chapter 1

Introduction

In the last decade, personal computers have evolved from supporting an individual's work to supporting collaborative work for groups of people. Single Display Groupware (SDG) [53] is a model that supports multi-user co-located work with a shared display. There are a number of SDG systems [4, 7, 8, 10, 21, 23, 36, 39, 44, 52, 62] that have been developed or even commercialized. Multi-Display Environments (MDE), on the other hand, integrate multiple devices such as desktops, laptops, PDAs, large displays, and digital cameras into interactive workspaces [18, 32, 45, 54]. Our project bridges between these two models.

Lacome, which stands for "LArge COllaborative Meeting Environment", is a software system that supports multi-user, multi-platform co-located collaborative work with a shared display. Using Lacome, users can share information such as full desktops or selected application windows by placing them onto a shared display. Users can also simultaneously interact with the shared display and the information within it. This includes direct interaction with desktops or application windows on other computers. Lacome is a combination of SDG and MDE: it has a primary shared display that contains all of the shared information, but it shows multiple windows from different

devices on the shared display and it allows users to interact with either the shared display or with other users' personal displays.

The remainder of this chapter describes the motivation for our project, some guiding principles that we followed throughout our work, and a summary of the rest of the thesis.

1.1 Motivation

The motivation for the Lacome project comes from the following scenario. A group of co-workers comes to a meeting room for a weekly meeting. The meeting room has a projector. Anyone who wants to give a presentation can connect his or her laptop to the projector via a video cable so that others can see the information the presenter wants to show. When only one person presents, this works well. However, sometimes there are multiple persons who want to present during a meeting. That makes the switching of the video cables between people's laptops very frequent, which takes a certain amount of time, especially when those laptops have different configurations and the owners are not familiar with the projector. This affects the efficiency of the meeting. Lacome aims to solve this problem by providing a better collaborative meeting environment.

A secondary goal of Lacome is to integrate and fully deploy our existing facilities and resources. In our laboratories we have three wall-sized large display surfaces (16 feet wide by 9 feet high) each with an IBM Deep Computing Visualization (DCV) system [29] available for collaboration-related research. There is no existing software toolkit for us to fully leverage the

computational power of the DCV to support our research. We thus faced a number of challenges working with the DCV system.

The DCV system only supports standard OpenGL [5] programs on its application nodes, which distribute the rendering tasks to multiple rendering nodes. Non-OpenGL programs cannot run on the DCV. Many of our existing research projects and demos are not built on OpenGL. Thus, we needed software to bridge the gap between the software environment of the powerful hardware facility available to us and our own project-specific software.

We tackled these problems by building a software system that allows non-OpenGL applications to be shown on the large display via the DCV and allows user to manipulate the applications remotely instead of directly operating on the DCV machine.

The constraints of the system configuration was not the only reason to develop Lacome. Multi-touch pointing techniques have been attracting much attention. Products such as Apple's iPhone [3], Microsoft's Surface PC [4] and various novel techniques such as done by Han [25] and Izadi et al. [31] using touch sensitive displays have showcased the potential of this area and are getting increased attention from the general public. Systems that support single-user multi-touch and multi-user multi-pointing will become more popular and may eventually become standard in the next generation of personal computers. How to accommodate conventional single-pointer systems to new multi-pointer systems will become an important issue to resolve. With built-in multi-pointer support, Lacome has the potential to provide a transition from conventional single-pointer systems to next-generation multi-platform systems.

In addition, our research group has been doing research projects in several directions on human-computer interaction and computer supported cooperative work in large display environments. Our purpose in building such a software system was in part to integrate our knowledge and experience to build a framework for further research activities.

For single-user applications, VNC [47] can be used to display an application running on one personal computer and to let the owner of the second personal computer control the input to the application on the first personal computer. Our work uses VNC as one of its building blocks but substantially extends VNC to multiple users and multiple simultaneous input.

The design concept of Lacome has been inspired from the Integrated Tabletop by Nakashima et al. [39]. The Integrated Tabletop is based on the Remote Frame Buffer (RFB) protocol [46] which is widely used in VNC-based applications [47]. It supports multi-cursor interaction in an application window under the Windows system. We have adopted some of the successful experiences from the Integrated Tabletop during the development of Lacome, but tried to resolve some of its drawbacks and improve some existing features.

- The Integrated Tabletop only supports Windows; while Lacome supports all three mainstream operating systems for personal computers.
- The Integrated Tabletop requires the installation of a modified VNC server on users' machines; Lacome, on the other hand, requires no changes to the VNC server: Any standard VNC server can be used unmodified to connect to Lacome.

- The Integrated Tabletop has only one control layer. Window manipulation takes place only when the user drags the margin of the windows as with a normal GUI window manager; Lacome has multiple control layers to better support different modes of interaction with shared displays.

1.2 Guiding Principles

In order to guide our work, we have identified a list of design principles that we used throughout our research:

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 [32], 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 Contributions

By designing and implementing Lacome, we have made the following contributions: First, Lacome separates the types of interaction a *Navigator* client can do into windows management tasks on the shared display (move, resize, iconify, deiconify) and application interactions through the VNC servers. In addition there are tasks such as pointing or other annotation on the shared display that we can consider as a third *Annotator* mode. Although the *Annotator* mode is not novel, we include it into our design for the completeness of all task types. Second, Lacome separates the *Publisher* and *Navigator* functions so a client can have one or both, or perhaps multiple Publishers, and can initiate *Publisher* connections for other VNC servers that may not have Lacome clients. Third, Lacome adds an explicit access control policy with components that are on both the server side and the client side to provide a flexible framework for sharing. Fourth, Lacome provides an architecture that builds on standard cross-platform components such as VNC and JRE. Last but not least, we take care to make good choices concerning the interaction techniques used in the window manager tasks to ensure simple and transparent multi-user interactions for managing the shared display space.

1.4 Overview of the Thesis

The remaining of this thesis is organized as follows. Chapter 2 presents related work. It draws attention to the key techniques that we used in developing the Lacome system. Chapter 3 presents the system architecture of Lacome from a system designer's perspective. Details of the user interface design are elaborated in Chapter 4. Chapter 5 describes implementation details and experience we have gained during our works. Informal evaluation with pilot users, conclusions drawn from the result, and future work are discussed in Chapter 6.

Chapter 2

Related Work

Personal computers started with the desktop and later the laptop. These two types of computers are designed for single-user activities. When we collaborate with colleagues who are co-located with us, we may find that a single desktop or laptop, or even multiple single computers used together do not satisfy our requirement for collaboration. There has been a demand that urges computer scientists and engineers of the need to design a new paradigm for computers to support multi-user co-located collaborative activities.

Another branch of the personal computer evolution seeks to make the size of the hardware smaller, such as for handheld devices. These portable devices are easy to hold, but in many cases, not easy to use. A common drawback of handheld devices is that they are too small to be easily manipulated. Although the size of handheld devices is continuously becoming even smaller, users still want bigger screens to allow more information to be displayed. This is the presumable reason the relative size of the screens in many handheld devices is getting bigger.

In this chapter, we elaborate on Single Display Groupware and Multi-Display Environments, as well as on several techniques that are closely related to Lacome, such as multi-cursor window management and network-

based input redirection.

2.1 Single Display Groupware

The term Single Display Groupware (SDG) was coined by Stewart et al. in 1999 [53]. However, the history of development and research on SDG can be traced to early 1987 when Xerox PARC created their CoLab system to make meetings for researchers more effective [52].

With increasing demand to have a computer that supports multiple users for co-located collaboration with enough screen space for multiple users to work on, SDG arose to meet the requirement. SDG is a model for supporting collaborative work between people who are co-located with each other [53]. A SDG system involves only one shared display on which all users collaborate. A SDG system can be used in conjunction with other personal computing devices such as desktops or laptops [36, 52], or PDAs [38]. These additional devices are used for displaying personal information for an individual user. Early examples of SDG systems with this type of configuration are the CoLab system created by Xerox PARC and the CaptureLab system designed by University of Toronto that use desktops [36], and the Pebbles project done by Carnegie Mellon University that uses PDAs [38]. A SDG may also consist only of a single shared display for the entire system, without any other personal displays, like with the DiamondTouch developed by Mitsubishi Electric Research Laboratories (MERL).

Strictly speaking, an implementation of a SDG is a system with one shared display to support multi-user co-located collaboration. We can dis-

cuss SDG from several aspects according to various attributes of factors of the display:

Display Model. This factor divides all SDG systems into two categories: shared display, and public plus private displays. In the shared display model, all users are working on the same single display. No other display devices are involved. In the public plus private displays model, users have a shared public display that shows the information available to all users. Meanwhile individual users have their own private displays that contain non-public information. The different models target different usage requirements. The shared display model is usually used when participants are involved in group discussion, sitting around a tabletop or standing in front of a wall display. The public plus private model applies to large meeting rooms in which participants are holding a group meeting in which some will do a presentation to others.

Resolution and Size. This factor also divides the SDG into two parts: single-tile displays and multi-tile displays. The single-tile display uses either a rear or front projector, or a single LCD monitor as the shared display. The multi-tile display leverages arrays of multiple projectors or flat displays to form a large mosaic of displays that acts as a single high resolution, large physical size display. Here we are more interested in those tiled displays that can increase both display size and resolution rather than those that only increase the display size. The advantages and disadvantages of both types are apparent. Single-tile displays are cheap, easy to implement, but have low resolution and relatively small size. Multi-tile displays offer wonderful resolution and greater size of the display, but are often expensive and space

consuming. We observe that larger displays seem to be the trend of recent development for SDG.

A study by Swaminathan and Sato indicates that "when a display exceeds a certain size, it becomes qualitatively different" [55]. Larger displays enable users to create and manage many more windows, as well as to engage in more complex multitasking behaviors [20]. Tan et al. conducted a series of experiments [56, 57] to explore the effects of display size on user performance in reading comprehension tasks involving static text, spatial orientation tasks involving static 2D scenes, and path integration tasks involving interactive 3D virtual environments and found that physically large displays improve performance on spatial tasks. Ni et al. provide a summary of the benefits of large displays [40]. We envision that with the projectors and flat displays gradually becoming more affordable, multi-tile display will become more popular.

Orientation. A SDG has two orientation styles. One is vertical, which mounts the shared display on the wall. The other is horizontal, which puts the shared display on a tabletop. The wall-mounted display may require more space, especially if it is a projection-based display instead of a flat display. However, the wall-mounted display naturally avoids one problem that the tabletop display suffers which is the issue of orientation of the displayed objects [34]. With a vertical display all users see objects in an upright orientation, but displaying the same objects on the tabletop display cannot satisfy all users because users are located around the tabletop thus some of them can only see the objects in a reversed orientation.

Touch Sensitivity. Some SDG systems are touch sensitive, others are

not. Typical examples of touch sensitive displays are DiamondTouch [21] and Microsoft Surface PC [4]. The DiamondTouch table developed by MERL is a multi-user, touch-and-gesture-activated screen for supporting small group collaboration. DiamondTouch allows up to four users to simultaneously use two pointer inputs each on specialized touch table hardware. There are many projects that have explored thi [41, 51, 59, 60, 66] using the DiamondTouch Table and DiamondTouch SDK. Although it has some limitations such as an inability to differentiate between the two pointers from the same user, and a limitation on the number of users, it is popular among researchers. Microsoft Surface PC [4] is touted as Microsoft's next generation personal computer. Various technologies [16, 64, 65] have been developing over the past five years during the Surface PC development road map. For touch sensitive systems, several different techniques are used to achieve input sensing. DiamondTouch and SmartSkin [44] use a conductive pad under the display panel to detect multiple pointers. Perceptive Pixel [25, 26] relies on frustrated total internal reflection (FTIR), a technique familiar to the biometrics community where it is used for fingerprint image acquisition. Microsoft Surface PC employs a camera sitting underneath the table to recognize users' hands and physical objects. ThinSight, the latest technique to emerge for multi-touch sensing, adds a new layer behind a regular LCD that allows infrared sensing to detect multi-touch input [31].

Projection Style. Depending on the hardware of the display and the space requirement, the projection style of a SDG can be front or rear-projected for a wall-mounted display, or top or bottom-projected for a table-

top display. A flat panel LCD display does not need projection. SMART Board [10] designed by SMART Technologies has a number of different versions that can be used as front or rear projection displays as well as flat touch sensitive tabletop or wall displays to satisfy a variety of user requirements. One disadvantage of many projection-based displays is that with a top or front-projected display a shadow of the user's arm or body will be cast on the display, which occludes the information shown in certain areas on the display. A bottom-projected display suffers from the bulky components that reside under the table such as projectors and mirrors. They often require users to stand or sit awkwardly for extended periods of time, which potentially impacts the comfort level of users [50]. A rear-projected display neither occludes information nor impacts the comfort level of users, however, it requires dedicated space which might be a concern for small rooms, or elaborate mirror geometry that may be difficult to calibrate and maintain.

We summarize the discussion above with Table 2.1, and fill in some typical SDG systems as examples.

We give brief descriptions for those system that we haven't introduced before:

The Integrated Tabletop [39] proposed by Nakashima et al. is a tabletop system that combines 2D and 3D projection together. The system is based on the remote frame buffer (RFB) protocol [46]. It supports multi-cursor interaction in an application window under Windows. The RFB protocol was used in VNC, which will be elaborated on later in this chapter. During the development of the Lacome system, we benefited from the experience

with the 2D-3D Integrated Tabletop. We tried to solve or remedy some problems it has exposed. This will be discussed in more detail in later chapters.

The SmartSkin system [44] designed by Rekimoto uses non-camera-based sensing techniques as describe above to support multi-pointer gestural input. Similarly, Perceptive Pixel [26], originally developed by Han based on an earlier technique he developed [25], also supports free hand multi-finger input. Both of their systems focus on bringing together multiple inputs. They do not distinguish which input comes from which user.

The PowerWall developed at University of Minnesota [8],the Scalable Display Wall developed at Princeton University [62], and the Interactive Wall developed at Stanford University [23] are typical wall display systems built by academic research. They all have very large sizes and resolutions.

Conventional SDG systems such as the CoLab [52] and the Capture-Lab [36], were built with one public display and a number of personal displays. Each participant was able to switch his/her input devices to the shared display, but no simultaneous multi-user input was supported. This issue has been solved in more recent systems in which smaller personal displays and the shared display are integrated into interactive workspaces. They are discussed in the next section.

2.2 Multi-Display Environments

As we saw in the prior section, SDG systems are not employed mutually exclusively with other personal displays. They are sometimes combined

together to form a Multi-Display Environment, sometimes also called Multiple Display Groupware (MDG) [15]. A common characteristic of these systems is that they all consist of a big, shared display and several small, personal displays. The Augmented Surface [45], iRoom [32], i-LAND [54] and ARIS [18] are just a few examples of Multi-Display Environments. Table 2.2 summarizes these systems and lists examples of the various types.

Augmented Surface developed by Rekimoto from Sony Computer Science Laboratories and Saitoh from Keio University [45] allows users to smoothly interchange digital information among their portable computers, tabletop and wall displays, and other physical objects such as a videotape or a document folder. It uses a technique called *hyperdragging*, supported by a camera-based object recognition system, that allows users to drag information beyond the screen boundaries to transfer to another screen.

i-LAND [54] is a Multi-Display Environment developed by Streitz et al. at the German National Research Center for Information Technology (GMD) and the Integrated Publication and Inforamtion System Institute (IPSI). i-LAND embeds different types of displays into office furniture to form several "roomware" components including an interactive electronic wall (DynaWall), an interactive table (InteracTable), two types of computer-enhanced chairs (CommChairs) and two "bridges" for the Passage-mechanism. The idea of integrating computing facilities into furniture follows the researchers' vision of future workspaces. Physical objects are supported in i-LAND via a concept called *Passage* that will detect uniquely physical objects with their weight to identify them. Other than using camera-based ob-

ject recognition systems like what the Augmented Surface does, the Passage implementation in i-LAND uses the weight of physical objects for identification and computer-controlled scales are built in the Bridge for detection.

ARIS is an interface for application relocation in Multi-Display Environments developed by Biehl and Bailey from the University of Illinois [18]. ARIS provides a direct manipulation interface that enables a user to visually relocate applications among computers in an interactive workspace. The interface uses an iconic map of the space to enable a user to perform application relocation. ARIS leverages the Gaia middleware framework [48] to handle application migration, which introduces a limitation that machines in the same space must be running the same operating system in order to support application migration.

The **iRoom** developed by Johanson et al. at Stanford University is another Multi-Display Environment. While i-LAND emphasizes the "roomware" that integrates computing facilities with furniture, the iRoom focuses on augmenting a dedicated meeting space with large displays, wireless or multimodal devices, and seamless mobile appliance integration [32]. iRoom has its own approach to moving objects between displays. Using the iROS middleware framework [42], it allows the data and state of the applications to be stored in a DataHeap. By sending events through an EventHeap, applications can be launched on another machine when the application window is dragged into its display. iRoom uses the PointRight system [33] to handle peer-to-peer pointer and keyboard input redirection in a multi-user multi-machine environment that will be discussed later.

In the next two sections, we discuss techniques that are critical to SDG

and Multi-Display Environments: including Multi-Cursor Window Management and Input Redirection. We then briefly summarize some other systems in the final section of this chapter.

2.3 Multi-Cursor Window Management

Multi-cursor support is necessary and important for both SDG and Multi-Display Environments, because they both support multi-user collaboration. Both have at least one shared display that allows concurrent multi-user collaboration. It is natural to imagine a scenario in which a group of participants work together on shared displays each with his own mouse or some kind of pointing device. One can also immediately realize that there might be issues with how to interpret input actions, especially when participants are working with legacy applications that are by default configured for single users.

Various researchers have studied a number of techniques to support multiple cursors [33, 38, 39, 59, 63]. These multi-cursor supporting techniques have several limitations and drawbacks. In the Integrated Tabletop system [39], the shared display can have multiple remote desktop windows and multiple cursors on it, but because the system only supports Windows, only one cursor can control a desktop window at any given time.

PointRight is similarly "restricted by the operating system to single cursor control per machine" [33], as is the SDGToolkit [59]. Moreover, if multiple users connect to the same display simultaneously, PointRight will "average" the cursor movements from all devices.

In work done by Wallace et al. [63], a limited number of virtual cursors time-share the system's single cursor. According to the authors' system experience, a user is usually interfered with by other users only when their windows are obscuring his.

In Lacome, we try to solve or remedy many of the above issues concerning multiple cursors. The machine on which Lacome is running acts as a dedicated server and does not deal with user inputs. Currently we are using a Linux machine, but this is transparent to end users and can be easily ported to other platforms. This makes Lacome a cross-platform system. Any desktop or application window can be shared on Lacome. For making Lacome a general purpose system that supports any platform, following our "make it compatible" principle, we do not perform any modification that is platform-specific. According to the original configurations of our targeted operating system's (Windows, Mac OS, Linux), the default setting only supports a single user. As a result a cursor can only control one window at a time in Lacome. Lacome also restricts the number of cursors that can simultaneously control certain single user application windows to avoid ambiguous behavior. The support of a semi-transparent window allows multiple users to collaborate together without being interfered with by others.

At the time of writing, the only technique that supports real multiple system cursors is the Multi-Pointer X Server (MPX) [28]. MPX is a groupware windowing system that natively supports SDG. When running on a SDG, MPX allows multiple cursors to interact with a single application simultaneously. The feature is not supported in any other multi-cursor systems so far. MPX directly modifies the X window system. As the first windowing system

supporting real multiple system cursors. MPX is still under active development with more features being added [27]. The current challenge of MPX and any other windowing systems that support multiple cursors is that the feature is not natively supported on either the hardware or the application level. Currently hardly any X window applications are designed with multiple users in mind. Thus if we apply MPX directly on these applications the outcome will be ambiguous and unpredicted due to the indetermination introduced by the load of serialization of cursor activity. We consider MPX to be a potential component of our Lacome system so that Lacome could potentially support multi-cursor manipulation on a single X window desktop that has MPX installed. We will discuss this possible extension in the final chapter.

2.4 Input Redirection

Input Redirection is crucial when we want to allow users to interact with content on a remote machine. There are many tools supporting this mechanism such as VNC [47], Integrated Tabletop [39], WinCuts [58], Rover [19], PointRight [33], Pebbles [38], ARIS [18], and Swordfish [24].

VNC is a popular software for remote control which we will discuss in the next section. Integrated Tabletop uses an extended VNC client to send the input redirection information to a specific machine. WinCuts manages remote machines by coupling them with a separate program called Visitor that handles input redirection. Rover, PointRight and Swordfish all use a spatial layout to arrange the screens so the mouse can travel between remote

screens that are adjacent in the layout. Pebbles allows a number of users to control multiple pointers on one remote display via PDA input. ARIS leverages its iconic map together with hot key combinations to accomplish the input redirection task.

In Lacome, our input redirection mechanism is similar to the one that is implemented in Integrated Tabletop. The major difference is that we implement multiple controlling modes instead of single controlling mode like all other systems. Details of the Lacome controlling mode will be elaborated in later chapters.

We treat the spatial layout for multiple remote windows as a separate type of action and provide users the capability to arbitrarily rearrange them independent of their ability to interact with the content of the applications running in the windows. In addition, we provide multi-user multi-pointer support similar to Pebbles, except that our pointers are controlled by mice instead of by PDA's, although we have the potential to extend this to support other external input devices such as PDAs and laser pointers without significant changes to the current system.

2.5 Other Techniques

We briefly summarize three techniques that are closely related to the development of the Lacome project and which form the basis for our work.

VNC

Virtual Network Computing (VNC) [47] is a software package that allows users to view and fully interact with a remote computer (the VNC server) using a simple program (the VNC viewer) running on another computer anywhere on the Internet [9] The VNC server can disable input from other VNC viewers for security and the VNC viewer can disable output to VNC servers for efficiency.

Currently there are several implementations of VNC such as RealVNC [9], TightVNC [12], UltraVNC [13] and OSXVNC [6]. They are all implemented based on the remote frame buffer (RFB) protocol [46] and have nearly the same features. Although VNC was originally developed for remote control of a computer, it can be applied to co-located collaboration [43]. We make use of the RFB protocol to build our Lacome system. We used the TightVNC server for our initial prototype testing.

Rover

Rover [19] by Booth et al. allows seamless switching between different machines under a multi-machine multi-platform environment to mimic a multi-screen computer. Rover provides a spatial layout for multiple screens in a virtual workspace. Moving the cursor out of one screen will let the cursor enter into the adjacent screen if that screen is available. Rover can be used by either a single user to seamlessly navigate through multiple computers under multiple platforms, or by multiple users to collaboratively access each other's desktops, subject to the constraint that when a computer/screen is

occupied by a user, other users who move their cursor into the screen will have to skip that screen immediately.

WinCuts

WinCuts is an interaction technique developed by Tan et al [58]. that allows users to replicate arbitrary window regions on the same or different computer. Each WinCut is a "live" view of a region of source window with which a user can interact. WinCuts supports replicating arbitrary window regions into independent windows. Manipulation of the WinCut source window will be continuously updated in the replicated window. The input redirection mechanism allows users to interact with the content in the replicated WinCut window and have the input redirected to the source window. The purpose of WinCuts is to more effectively use limited screen space.

2.6 Summary

In this chapter we have reviewed general concepts of Single Display Groupware and Multi-Display Environments as well as some typical systems. We also have discussed underlining key techniques that have been applied in this area. All of these impact our design and implementation of Lacome, which stands for "LArge COllaborative Meeting Environment". Starting in the next chapter, we elaborate the design of the system architecture and the user interface of Lacome. In addition, we provide selected implementation details, and some of the insights we have gained from our work.

System	Display Mode	Resolution Size(cm/inch) Orientation	Touch/ Max Points	Projection Style
DiamondTouch [21]	Shared	2736x2048 86x65/(34x26) Vertical	Yes/4	Top
Microsoft Surface PC [4]	Shared	1024x768 56x53(22x21) Horizontal	Yes/52	Bottom
Integrated Tabletop [39]	Shared	NA NA Horizontal	No	Top & Bottom
SmartSkin [44]	Shared	NA 80 x 90/(32x35) Horizontal	Yes/<=10	Top
SMART Board [10]	Shared	1600x1200 171x128/(67x50) Hori./Vert.	Yes/1	Rear Front Flat
Perceptive Pixel [26]	Shared	5120x1024 488x91(192x36) Hori./Vert.	Yes/NA	Rear
CoLab [52]	Pub. & Pri.	NA NA Vertical	Yes/1	Rear
CaptureLab [36]	Pub. & Pri.	NA NA Vertical	No	Rear
PowerWall [8]	Shared	3200x2400 183x244/(72x96) Vertical	No	Rear
Interactive Wall [23]	Shared	4096x2304 183x107/(72x42) Vertical	Yes/1	Rear
Scalable Display Wall [62]	Shared	6000x3000 244x549/(96x216) Vertical	No	Rear

Table 2.1: Examples of Typical SDG Implementation

System	Display Type	Data Proce. Tech. Sup. Physi. Obj.	Sensing Techniques
Augmented Surface	Computer, Wall Display, Tabletop	*Hyperdragging* [45] Yes	Camera-based
i-LAND	*DynaWall*, *CommChairs* *InteracTable*	BEACH [54] Yes	Weight Detection
ARIS	Computer, PDA, Wall Display	Gaia [48] No	NA
iRoom	Computer, PDA, Tabletop, Wall Display	iROS [42] No	Touch-based

Table 2.2: Multi-Display Environments Implementations

Chapter 3

System Architecture

In this chapter we discuss Lacome from a system designer's perspective. We first give an outline of the system, followed by the architectural design of both the server and client components. After that we elaborate on the access control policies we have incorporated into the system.

3.1 System Outline

In Lacome we used a client/server architecture to design and implement the system. The system consists of one server and an arbitrary number of two types of clients. The clients are all communicating directly with the server while the server decides what needs to be drawn on the large display and how to do it. Figure 3.1 illustrates the configuration for a typical Lacome usage scenario. There is no direct connection between clients. All communication goes through the server.

During our discussion on Lacome, we will use a few terms which are explained below:

The term ***Publisher*** refers to a machine that shares its workspace for others to access on the shared display via a VNC server running on the *Publisher*. This is a little ambiguous but in Lacome, a VNC server will be

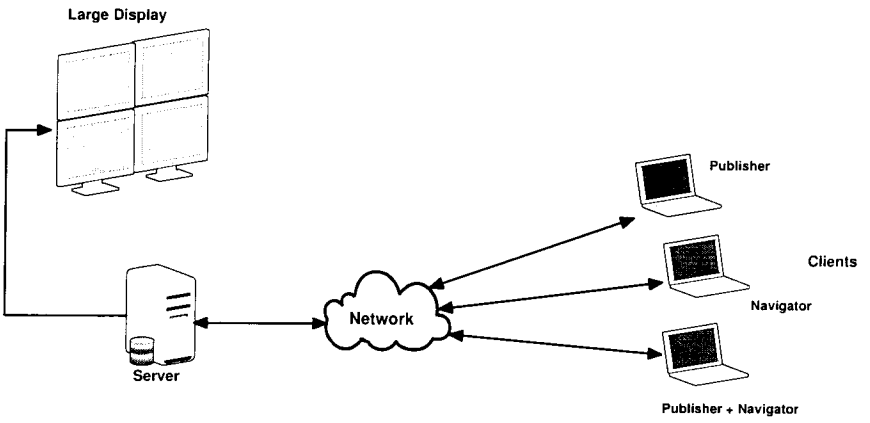

Figure 3.1: A Typical Hardware Configuration of Lacome

considered as a Lacome client that we call a *Publisher*.

The term ***Navigator*** refers to a machine that forwards its mouse and keyboard events onto the workspace to interact with the workspace.

The term ***Manipulator*** refers to an operating mode in which a *Navigator* can do several kinds of manipulation tasks such as move, resize, iconify and deiconify to the windows on the shared display. This is the default mode when a *Navigator* first gets initialized.

The term ***Controller*** refers to an operating mode in which a *Navigator* controls some *Publisher*'s workspace via the *Publisher*'s VNC server. All input events generated from the *Navigator* are redirected to the *Publisher* in the Lacome server.

For completeness, we use the term ***Annotator*** to refer to an operating mode in which a *Navigator* is used in a "drawing" or "sketching" mode to annotate or "decorate" information on the shared display. Our current

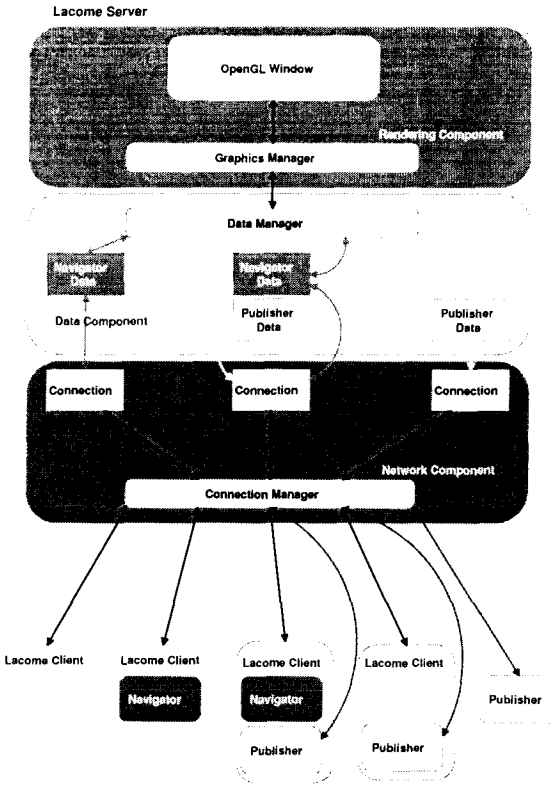

Figure 3.2: Lacome System Outline

implementation does not implement this mode, but a production system would.

Publisher and *Navigator* are not mutually exclusive. One client can be a *Publisher*, a *Navigator*, or both at the same time, as illustrated in Figure 3.1.

Figure 3.2 shows a more detailed view of the system architecture. Within

the Lacome server, there are three components:

The ***Network Component*** deals with all communication between the Lacome server and the clients. A connection manager is developed to manage all client connections. Each connection is associated with a particular client's data in the Data Component.

The ***Data Component*** is the core component of the server. On the one hand, it serves as a bridge between the two other components. It constantly extracts events from the Network Component and processes them. Moreover, it sends information to the *Rendering Component* so the appropriate data can be displayed to the users. Determining which information to send is one of the major tasks of this project.

The ***Rendering Component*** is responsible for drawing everything in a proper way.

On the client side, there are two components that correspond to the *Publisher* and *Navigator* roles, both introduced above. A client does not need to have both active. Each can be enabled and disabled on demand, which provides more flexibility.

There are two major types of data flow.

Client to Server: data flows as a *Publisher* client informs the server when to establish or close a VNC session and as it sends frame data to the server to update the display of a shared desktop or application window. Similarly a *Navigator* client sends data and events to the server to update its cursor information, such as size, position or color.

Server to Client: the server is responsible for establishing connections and disconnecting from clients or VNC sessions. Server also redirects

the *Navigator*'s inputs to the specific *Publisher* when the *Navigator* is in *Controller* mode.

3.2 Server Architecture

The Lacome server contains the three components introduced in Section 3.1. In this section we elaborate on each of them.

Network Component

The Network Component is responsible for all connections and data transmissions between the server and the clients. When the Lacome server starts running, a connection manager is installed. The connection manager consists of a client manager and a *Publisher* manager. The client manager first initiates a thread that keeps waiting and accepting new client connections. When a new connection has been established, a new thread is spawned and assigned to that connection. The new thread extracts data from the client connection and forwards the data to the *Data Component*. There are two modes by which *Publishers* can connect. Firstly, we establish a thread that passively listens to VNC requests from external *Publishers*. Moreover, a client may register a *Publisher* machine requesting the system to connect to the *Publisher*. The *Publisher* manager will then actively establish a VNC connection to the designated machine.

When a *Publisher* connection is established, the Lacome server acts as a VNC client to the VNC server installed on the *Publisher*. Similar to the client connection case, a dedicated thread is associated with each active

Publisher connection to handle tasks such as reading frame data from the *Publisher* and redirecting input events to the *Publisher*. A detailed graph of the Network Component is illustrated in Figure 3.3.

The connection manager is also responsible for maintaining the status of each connection and disconnecting inactive ones.

We have reused the RFB protocol [46] for transmitting frame data and input events, so our system is compatible with all standard VNC servers. That means any implementation of the standard VNC server such as TightVNC, RealVNC, etc. can be used to establish a *Publisher* and connect to the Lacome server without any changes.

Data Component

The *Data Component* manages the current status of each client, including information about its *Navigator* and *Publishers*. The structure of a client is shown in Figure 3.4. A client can be associated with at most one *Navigator* and an arbitrary number of *Publishers*. In our case, the *Navigator* must be exactly the same machine where the client is located, but the *Publishers* might come from either the local machine or other machines. A *Publisher* has an association with a client when that *Publisher* is registered by that client.

A client that is not associated with any *Navigator* or *Publisher* is called an **Empty Client**. By default, a client is empty when the client is initialized and first connected to the Lacome server. An empty client contains basic information such as IP, nickname, etc.

As suggested by its name, a **Non-empty Client** is a client that has at

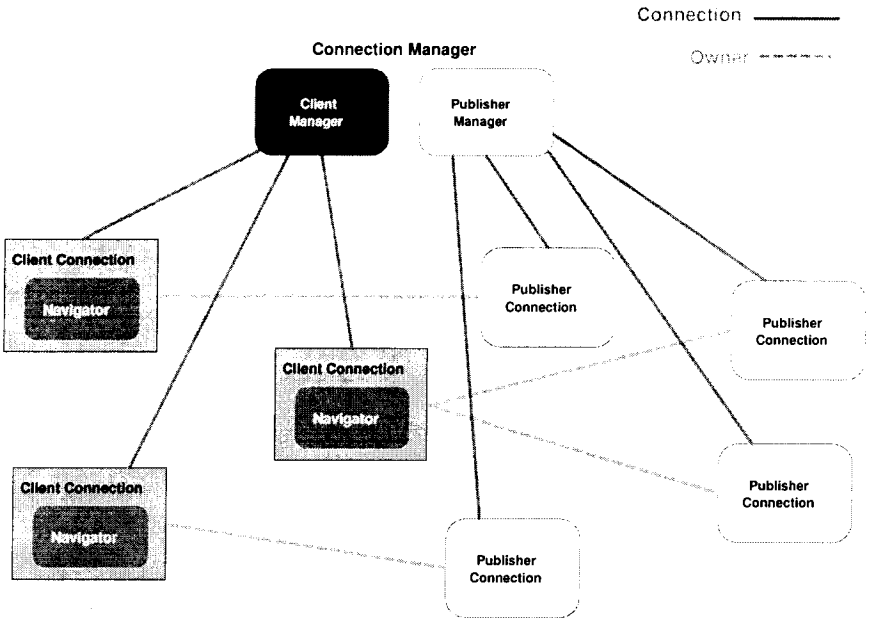

Figure 3.3: Network Component

least one *Navigator* or *Publisher*. For a non-empty client, any of its Navigator and Publishers will have a corresponding connection in the Network Component. A data manager is designed and implemented to keep track of all the data in the Data Component. All data are kept in specific data objects and the data manager organizes them into several logical sets. For example, the data manager keeps a list of references to the *Publishers* that have been iconified, so that it is very easy for the OpenGL thread to render them appropriately. The iconification and the OpenGL thread will be discussed in Chapter 4 and Section 3.2 respectively.

The data of a client can be grouped into three categories:

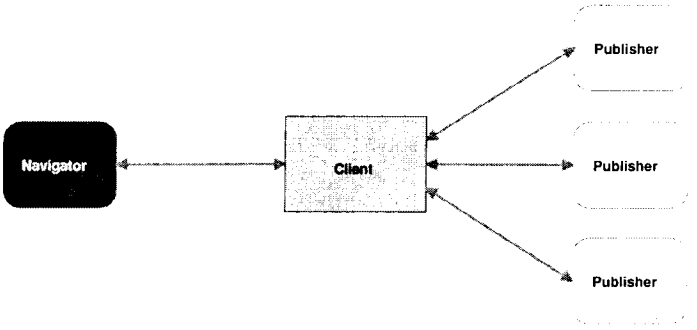

Figure 3.4: Client Structure in Data Component

Basic client information This includes client ID or nickname as the replacement of its IP, its control mode (discussed in Section 3.4), a list of references to the *Publishers* registered by it.

Navigator data This includes information such as the coordinates of the cursor and the button mask of the keyboard, both needed for manipulating multiple windows and redirecting input events. It also contains a reference to the *Publisher* window that is under its control, and the size and color of the cursor.

Publisher data This includes the scale and the aspect ratio of the window, the screen position of the window on the shared display, a backward reference to its owner as well as to its controller, and a reference to its frame buffer.

Rendering Component

The entire Rendering Component resides within the main OpenGL graphics thread. It handles the rendering tasks with the help of a modified version of the ivLib [61], which provides well defined advanced primitives such as vector. We discuss this in more detail in Chapter 4.

3.3 Client Architecture

The client/server subsystem supports multi-client collaboration. The structure outline for a single client is illustrated in Figure 3.4. When there is only one client, there is little need for structure. However, when there are multiple clients working together, the situation becomes more complicated than the single client case, so careful design and implementation are important.

Navigator

A Lacome client can have at most one *Navigator*. The *Navigator* is responsible for transmitting the client's cursor and keyboard events to the server to support the client's cursor on the server. The *Navigator* belongs to its client. When the client disconnects, the *Navigator* is also immediately disconnected

Publisher

A *Publisher* is a machine with a standard VNC server that shares a desktop, application window, or part of a desktop or application window with the server.

All *Publishers*, even when their VNC servers are not running on the same machine as some client, have a logical connection with the client that introduced them to the server, but the client does not control their resources.

Once connected to a Lacome server, a *Publisher* sends its frames to the Lacome server and may accept redirected input messages. When a *Publisher* is introduced to the Lacome server by a Lacome client, the Lacome client is the "owner" of the *Publisher*. As a result, when a Lacome client disconnects, all *Publishers* that have been brought to the Lacome server by that client are also disconnected.

3.4 Access Control Policies

As mentioned in Section 3.3, when there are multiple clients connected simultaneously, there are several situations that do not exist in a single-client scenario.

Scenario 1

Two users, A and B, are working together each on their laptops using *Navigators* to access a public machine serving as a *Publisher*. Both A and B need to get into the *Publisher* in *Controller* mode to control the shared desktop or application window. Since most operating systems on personal computers are currently designed for single user access, the result will be unpredictable when both A and B try to control the application at the same time. Lacome does not attempt to solve this problem but does provide features that facilitate social mechanisms for coordinating usage of access control to reduce

inadvertent collisions.

Scenario 2

Several users are working simultaneously on the large display. User A connects his laptop as a *Navigator* via his Lacome client while at the same time sharing his desktop using a standard VNC server as a *Publisher*. User B connects with another *Navigator*. B wants to control A's desktop when A's *Navigator* is controlling somebody else's desktop. User B might find that by controlling A's desktop he is indirectly controlling the desktop that is currently under A's control. Moreover, A loses control of both his own desktop and the one he is controlling. This is neither B's nor A's purpose.

Thus we need to design a series of policies for access control to make the system robust. The problems described in the above scenarios are the major access control problems in our system. In order to tackle these problems, we carefully analyze our system and come up with the following policies:

Three-Mode Interaction. Nearly all existing window management systems leverage a single mode structure to handle access control. In our system, we use a new, three-mode structure which we believe provides a better sense of control and management. In the *Manipulator* mode, *Navigators* can freely manipulate any of the windows shared by the *Publishers*, moving, resizing, and iconifing them as they desire to arrange the shared space. Only when a *Navigator* enters *Controller* mode does it communicate with the application running on the *Publisher*'s computer. In *Controller* mode the cursor is restricted to stay within the window it is controlling. In *Annotator* mode the *Navigator* is able to do annotation on the shared

display space either with a single window or across multiple windows. The advantages of this three-mode structure will be discussed in detail in Chapter 4.

One Owner Interaction. We store a reference in every *Publisher* that points to the *Navigator* which controls it. At any given time a *Publisher* can be controlled by at most one *Navigator*. Other *Navigators* cannot access an occupied *Publisher* until the controlling *Navigator* relinquishes its control.

Tailorable Access Control. A flexible access management module is proposed to handle the complexity brought by the situation of single-client-multiple-roles to prevent unexpected outcomes. The basic strategy is to not allow a client to have multiple roles at the same time. When a client is already a *Publisher* and sharing its desktop or application window, there are several different options to solve the problem that occurred in Scenario 2:

- Force *Navigator* and *Publisher* to be mutually exclusive in a client. When there is already an active *Publisher* in a client, the system will forbid the *Navigator* in the same client to be turned on, and vice versa. This prevents the problem from happening.

- For a client who has an active *Publisher*, if a *Navigator* from another machine is trying to get access into the *Publisher*, the system will send a query message to the client to ask for a response. If the client approves, it will give up its *Navigator*. The client then needs to repel the controlling *Navigator* out before it can re-enable its own *Navigator*.

The access control in Lacome is crucial. In this section we have discussed

the system structure side of our access control policies. In Chapter 4 we will elaborate the interface side, to which we pay more attention.

3.5 Summary

In this chapter, we have discussed the system architecture of Lacome. We have gone through both the server side and the client side, as well as the design of the access control. With a general concept of the system architecture in mind, in the next chapter we will turn to the design of the user interface, which is another important component of the system.

Chapter 4

Interface Design

Interface design is considered the most important part of Lacome. Existing window systems are designed to support a single user and they are not suitable for multi-user co-located collaboration. Multi-user concurrent collaboration is our goal in Lacome. In this chapter we introduce the interface design of Lacome. We first list some design requirements that we set for the system. We then describe the multi-cursor window manager we designed, followed by a description of the client interface design.

4.1 Design Requirements

Building on the guiding principles set forth in Chapter 1, we adopted the following requirements during the design process to guide our work. We used these same requirements to test the prototype we built.

- The system should be cross-platform. It should support all mainstream operating systems including Microsoft Windows, Apple Mac OS, and Linux.
- Any user should be able to connect his/her own laptop to Lacome without complicated setup work. Users should be able to join, leave,

or re-join a Lacome session, at any time without disruption.

- The system should provide most, if not all, of the basic window management features that a typical window system provides, such as moving, resizing, iconifying and deiconifying windows. Moreover, these features should be intuitive enough for the users to recognize, learn and remember, without imposing heavy mental workload on first-time users.

- All users should be able to visibly identify their cursors on the shared large display. They should also be able to customize the size, shape, and color of their cursors.

- Mutual interference between multiple users should be minimized both when they are doing individual work and when they are collaborating.

- The access control model set by the system should be easy to understand with sensible defaults to ease the transition from novice to expert user.

4.2 Multi-Cursor Window Manager Design

Mainstream windowing systems only support a single-user and their window managers support only a single cursor. There are, however, research systems that provide multi-cursor window managers [33, 39, 59, 63]. Unfortunately, we cannot directly adopt these approaches. The major challenges are that they only support a single specific operating system [33, 39, 59, 63], and they require a fair amount of specialized setup. These factors make them less

portable. Moreover, there are other drawbacks such as only one single mode for access control [39], a limited number of cursors [39, 63], and undesirable interference between users when one user's window obscures another user's window.

Based on the experiences reported by other researchers, we took all these factors into consideration when designing Lacome. With respect to our three-mode access control structure, we divide this section into three parts. In the first part we describe the general features we develop for the Lacome system in the *Manipulator* mode including Cursor Customization, Drag and Drop, Scaling, Transparency, and Iconification. In the second part, we introduce more features we have developed exclusively for the system under the *Controller* mode. At last, we illustrate the *Annotator* mode which we have not yet implemented in the current version of Lacome. However, we explain its features for the sake of completeness.

Manipulator Mode

The features we describe here are not novel. Each of them have been implemented in some single user windows systems. However, to our knowledge, Lacome is the first multi-cursor window system that has combined all of these features together. We have reasons to believe that they should all co-exist within a multi-cursor window system. We elaborate them one by one and explain our design rationale for each.

Cursor Customization. Cursor customization is important in a multi-cursor window system. Individual users need to visibly differentiate their cursors from other users' cursors. Different users may also have different

preferences for their cursors. We enable cursor customization in three categories: color, size and shape. Figure 4.1 shows a picture of sizes and colors for the default arrow-shaped cursors. When a user starts the *Navigator* on

Figure 4.1: Cursors with Different Colors, Sizes and Same Default Shape

his/her client machine, the Lacome server will automatically assign a unique color to his/her cursor on the shared large display. The color of the cursor should be also shown on the client. This makes it easy to identify the cursor anywhere on the screen. When the user disconnects from the system, the Lacome server will "recycle" the color that the user was using so that it can be reassigned for another incoming user. Users can change the size of their cursors on demand through the Lacome client, which is described in Section 4.3.

Window Initialization. The initial states of all shared windows are pre-defined. The system sets the coordinates of the center of any new shared window to the same point on the shared large display.

It also sets a fixed initial value for the height of a shared window and calculates the width based on the actual aspect ratios of the source window provided by the *Publisher*. The width of the window will be recalculated whenever the aspect ratio for that window is updated on the *Publisher*. In

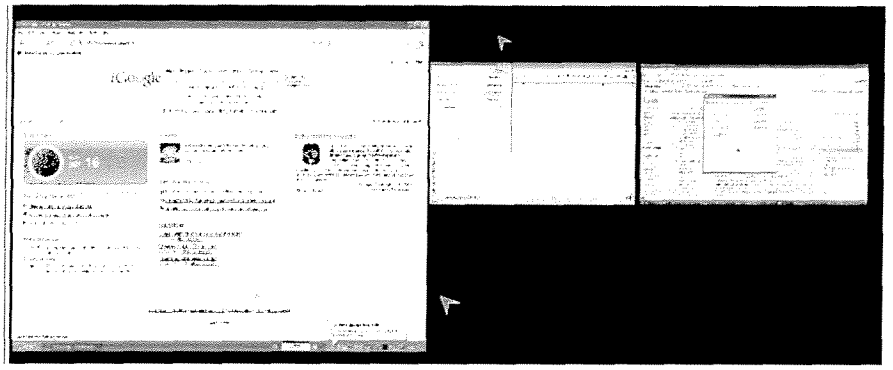

Figure 4.2: Initial States and Modified Windows

Figure 4.2, the window in the middle of the screen is at the initial location with the initial default height. The window on the right has the initial size but has been moved from its initial location. The two windows have the same height. The window on the right has a bigger width than the one in the middle because its aspect ratio is bigger than the other's. The window on the left is at a modified location with a modified size.

This design decision was made after investigating several approaches. There are two sets of parameters we need to consider: the initial location of a shared window and its initial height and width. Many window systems seem to believe the most obvious choice for the start up location is a random location. More sophisticated approaches employ heuristics such as calculating the area of unoccupied screen to find a place to locate the window to minimize occlusion of existing windows. All of these approaches have shortcomings:

Firstly, since the shared windows are popped up at a random location,

it is not easy for the users to find the one they want when there are multiple shared windows arriving quickly in succession, especially when they are initiated by multiple users. Moreover, the dynamics get more complicated when other users are concurrently manipulating existing shared windows so the mere fact that a window is changing on the screen does not identify it as being the new window.

Secondly, if a new shared window shows up at a location occupied by some window currently being manipulated by another user, the new window may interrupt the other user's work.

Based on these considerations, we decided to have all new windows show up at the same location to save available screen space and give users clear hints on where to find new windows. For similar reasons we chose to fix the initial height of each new window.

Drag and Drop. Drag and drop are the most common means of repositing individual windows in most window systems. We followed the conventions of other popular window systems for this feature and set the left mouse button as the trigger for drag and drop. Here we assume that the user of Lacome uses a three-button mouse (we later discuss the solution for mice with fewer buttons). A shared window is busy when there is a cursor holding (dragging or scaling) or controlling it. A window on the shared large display is either busy or available. When a cursor is hovering on or dragging an available shared window, the color of the window's border will become the same as the color of the cursor, as illustrated in Figure 4.3. This identifies the "owner" of the window.

There are problems that are specific to multi-user window systems that

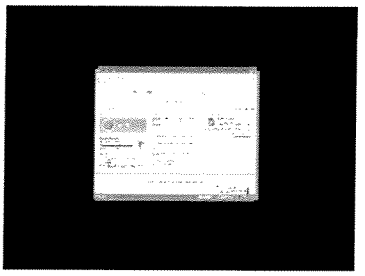

Figure 4.3: A Cursor Hovers on a Shared Window

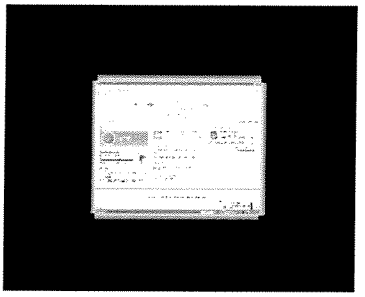

Figure 4.4: Two Cursors Hover on a Shared Window

are completely absent in the single-user scenario. One of them is that multiple cursors might be hovering on the same shared window. In that case the color of the border of the window brought by a cursor below will be overlapped by the one's above[39]. In Lacome, we solve this by making the extra border bigger than the existing one, as illustrated in Figure 4.4, so that users are aware that there are multiple cursors working in the region of this window and they can easily find theirs as well as others' cursors on the shared large display.

The difference in how drag and drop works between Lacome and almost all other systems, is that in other systems, drag and drop can only be performed by clicking on the window's border because there is only one mode of access control; when the cursor moves inside the window region, it starts to control the shared desktop on the machine where the desktop resides. In Lacome, we have three-mode access control to tackle this problem. Drag and drop operations are operating in the *Manipulator* mode, so a user can click and hold on any point within the window region or its border to move the window. Thus users need not to worry about whether their cursors are clicking exactly on the border of the windows if they want to drag the windows without interacting with the content of the window.

Resizing. Resizing is important especially in the large display environment. In order to save screen space, we set the initial window size to be relatively small. When a user wants to do a presentation with some slides or several people want to see the same window on the shared large display, they may want the window to become large enough for everyone to easily see it. We use a modified drag and drop with the right mouse button to set resizing. Similar to the drag and drop for positioning, the user holds and drags any point in the window region to resize it, instead of clicking only on the border.

Lacome tries to support multiple users without too much interference. To achieve this, we set a minimum and maximum values for the size of a shared window, so that a window will neither obscure the whole large display, nor become too small to be recognized. In Figure 4.5, the window in the middle has the initial size, the one on the left has maximum size and

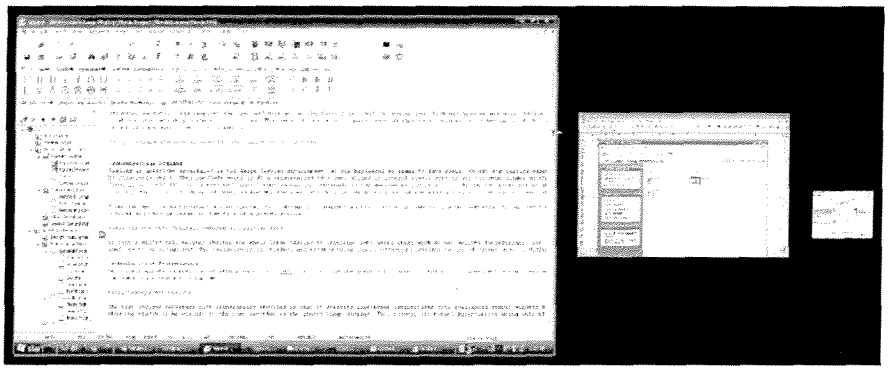

Figure 4.5: Scalable Shared Windows

the one on the right has minimum size. The threshold for the minimum and maximum size can be adjusted according to the physical screen size and resolution as part of the system specific parameters.

Transparency. Based on the experience provided by the Integrated Tabletop [39], we realized the value of rendering transparent windows in some circumstances. The Dashboard widgets in Apple Mac OS X have a similar semi-transparent windowing feature [2]. We decided to implement this feature in Lacome, as shown in Figure 4.6. The most obvious advantage transparency provides is that it supports concurrent interactions with overlapped shared windows by allowing more than one window to be visible at the same location. This reduces interruptions between multiple users.

Iconification. Windows not currently in use can take up valuable screen space and obscure other windows. To prevent this from happening, we implement iconification as do most window managers. A window is iconified when a user double-clicks on it. An iconified window is shrunk to a standard

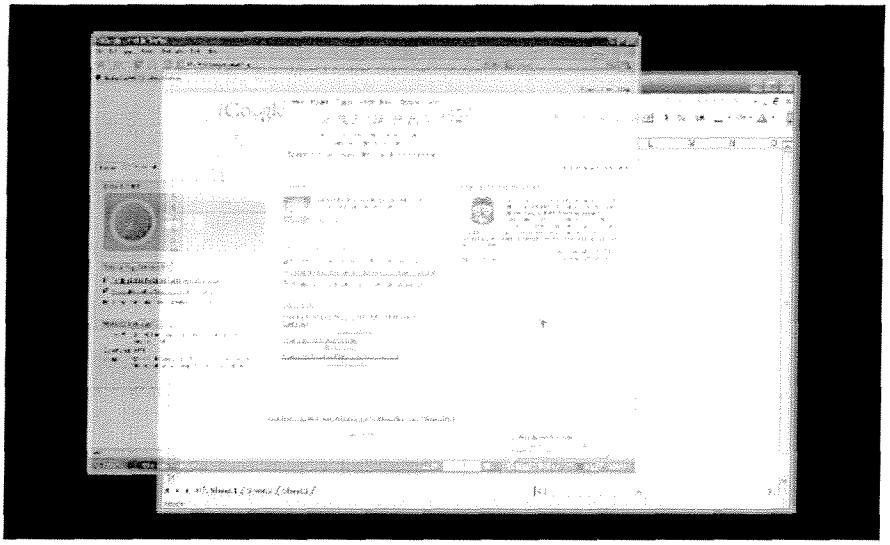

Figure 4.6: Transparent Windows

sized icon that resides at the bottom of the shared screen. It will recover to its previous size and location when someone double-clicks on the iconified window. Figure 4.7 illustrates this. This iconified version of a window is simply a smaller sized rendering of the window. We do not substitute an iconified window with a static icon because Lacome is a multi-user system and we cannot expect each user to know or remember what every icon made by others is. So we show a miniature of the window to give users enough information to identify it.

Following a technique by [37], we have added a window expanding effect to the icon list, as illustrated in Figure 4.8. When a cursor is moving close enough to a window icon, that window icon will temporarily expand to

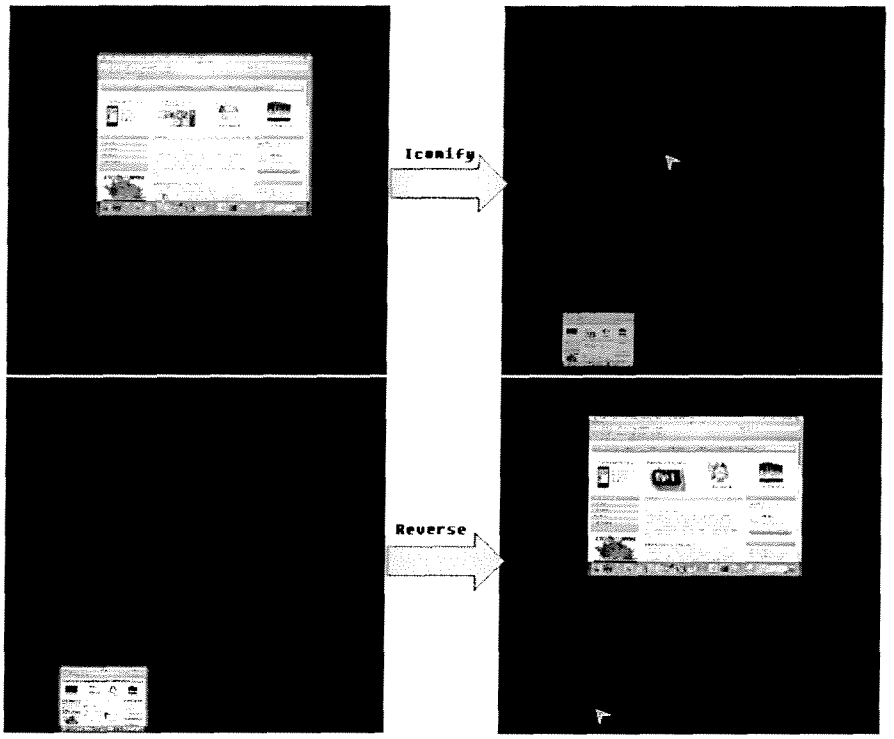

Figure 4.7: Windows Iconification

become bigger so that it will be easy to reach and read.

Controller Mode

As mentioned previously, there is a three-mode access control structure in Lacome. In Section 4.2 we introduced the features in the first mode, known as the *Manipulator* mode. In this section, we discuss the *Controller* mode. We first discuss the frame stack, which is an essential concept to understanding the Lacome interface. We then elaborate on input redirection.

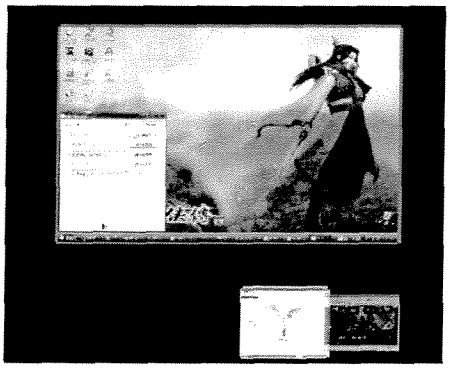

Figure 4.8: Expanding Window Icons

Mode Switching. To switch from the *Manipulator* mode to the *Controller* mode, a user clicks the middle mouse button on a target window. Depending on which access control policy the system is using (see Section 3.4), the server will either grant or refuse the access request of the user. When a cursor is in the *Controller* mode, its movement will be restricted to within the area of the target window. Figure 4.9 shows a *Navigator* going into *Controller* mode and interacting with a shared desktop window.

In *Manipulator* mode a cursor can move and resize a target window, but cannot get into the desktop or interact with the machine on which the desktop resides. We have chosen a key combination so the cursor will not accidentally switch out of *Controller* mode. The hot key combination *Crtl-F1* is used to switch from *Controller* mode back to *Manipulator* mode. We believe interaction requirements are distinctly different under different modes. This isolation provided by separated modes allows us to tailor our interface to facilitate people working in each mode. More details are discussed later.

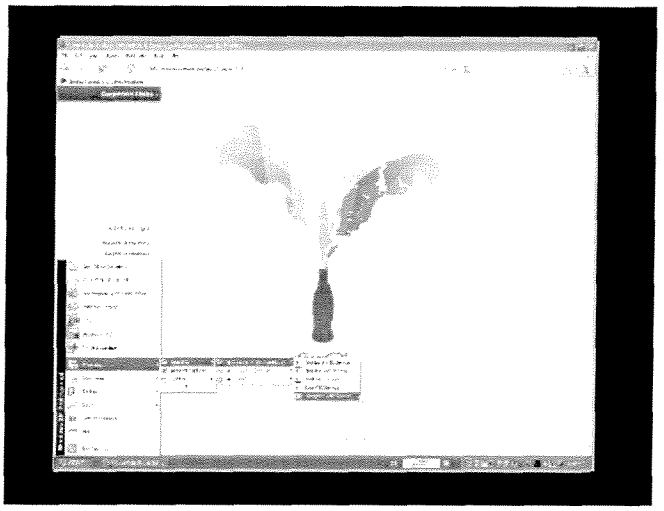

Figure 4.9: Controller Mode

Frame Stack. The frame stack provides an ordering of the shared windows, telling which windows should be displayed in front of others. We have implemented the following policies in our prototype design:

Windows controlled by cursors in the *Controller* mode have highest priority and reside on the top of the stack. They are rendered on top of all other windows, even those being manipulated by cursors in *Manipulator* mode. Windows that are not being controlled or manipulated have the lowest priority and reside at the bottom of the stack. As a result, shared windows under the *Controller* mode and shared windows under the *Manipulator* mode will be on top of all windows that are available.

Secondly, whenever a window enters a new mode (from available to *Manipulator* or *Manipulator* to *Controller*), it will be put on top of everything in that mode. As a result, the ordering of the windows within a mode will

be determined by the order they entered that mode: oldest rendered first, most recent rendered last.

The above design policies indicate that shared windows residing in the *Controller* mode will always be on top. Intuitively, windows in the *Controller* mode are actively interacting with users and should not be obscured if another window is merely moving across them in the shared display.

Input Redirection. The input redirection feature utilizes the RFB protocol [46], which is used by VNC. As a result, any machine with a standard VNC server is able to connect to Lacome and share its desktop and/or application windows. When Lacome starts processing input redirection between a client (*Navigator* in *Controller* mode) and a VNC server (*Publisher*), it forwards the mouse and keyboard events it receives from the Lacome client to the corresponding *Publisher* client (a VNC server). One may question why the two interacting parties are not directly connected instead of having a server redirecting input events. There are several reasons behind this design.

Primarily, Lacome holds the context on the cursor coordinates. Recalling that each shared window could be resized and moved, we must take these parameters into account when we translate the mouse coordinate between different desktops. More specifically, we must first translate the controller coordinates from its local desktop to the shared coordinate of the large display. After that, we have to figure out the position of the cursor relative to the controlled target window left top corner, with appropriate scaling, before we can correctly redirect input events. In addition, the *Controller Navigator* may have a different desktop resolution than the one it is controlling. For

example, the input machine may operate with a resolution of 1024 x 768 while the target machine operates on 1600 x 1200. In a standard VNC implementation, the controller will have scroll bars to accommodate the difference in the area. Lacome, on the other hand, has built-in support to accommodate different window sizes by resizing, so it can easily handle these cases.

The second reason Lacome serves as an intermediary is that it is not a common practice to have a single VNC server accept multiple client connections. Unless specifically set by the user in the VNC server, a VNC server typically either rejects further incoming connections or disconnects the current connection in order to accept a new connection. Both of these cases are undesirable. When multiple connections occur, we observe a significant increase in system workload on the *Publisher*. This increase is unnecessary because it is handling two clients of which one is only out-going (Lacome) while the other one is only in-coming (input redirection). Because our VNC servers are often running on laptops or mobile devices, it seems more appropriate to shift the workload to the dedicated server running Lacome.

The third reason is that Lacome can act as a security checkpoint to avoid exposing IP addresses or user identities. In some cases, *Publishers* are behind firewalls that impose restricted access. It may be easier to add a Lacome server as a trusted endpoint thus to add multiple *Navigator* clients. Moreover, adding each client which may be interacting only for a limited amount of time to a firewall exception list is tedious and prone to security flaws.

The final reason is that Lacome serves as a buffer for faulty connections.

In the situation where one of the parties is suddenly disconnected, the Lacome can maintain all of the data and gracefully handle the disconnection and reconnection if the missing party returns.

The user of a *Controller* forfeits control of a window by pressing *Ctrl-F1*. We have found that this combination is seldom used by most software and operating systems. We have little worry that this key combination would be intercepted by the underlying operating system. After some trial-and-error on several operating systems (Microsoft Windows, Apple Mac OS, Linux), we have confirmed that *Ctrl-F1* is a good choice.

Although we cannot expect Lacome to monitor all key stroke events and detect if *Ctrl-F1* was pressed, we leave this task to the Navigator client. We are expecting a much higher data flow rate in the *Controller* mode, so we believe that monitoring keys strokes may introduce unnecessary processing overhead. As well, we envision that there will be scenarios where the client would customize the *Navigator* to use a different key combination. Our design makes this change invisible to Lacome and to the *Publisher* client. Upon detecting the forfeit key combination, the *Navigator* client would send a message to inform the Lacome server to switch the *Navigator* back to *Manipulator* mode. The *Publisher* would not be informed of this.

Annotator Mode

One significant difference between *Annotator* mode and the other two modes is that the *Annotator* mode has nothing specific to do with the *Publisher*. Its manipulation target is the shared display space. By allowing users to do annotation on the shared display space, Lacome provides the feature of

collaborative painting that is also provided by other collaboration toolkits or systems such as GroupKit [49]. When entering the shared space, a *Navigator* will get into *Manipulator* mode by default. Then it can change its mode to either *Controller* or *Annotator*, depending on the user's purpose. A *Navigator* in *Controller* mode or *Annotator* mode will have to switch back to *Manipulator* mode before switching to the third mode.

In the current version of Lacome we have not implemented the *Annotator* mode. However, we want to include it in the future extension of Lacome for its widely applicable area as well as function completeness. In addition, it is not difficult to implement it under our framework.

4.3 Client Design

Compared to the Lacome server, a Lacome client is relatively small, but that does not mean it is of little consequence. To the contrary, the client is the middleware that end users use to interact with the system. The design of the client is directly related to the usability, and hence the success of Lacome.

Functional Components

A complete version of an interface for a Lacome client (shown in Figure 4.10) consists of the following three components:

Server Connection. This has a text field to provide the IP address of the running Lacome server and its port. Currently, a user has to know the exact IP address and the port on which the Lacome server is running

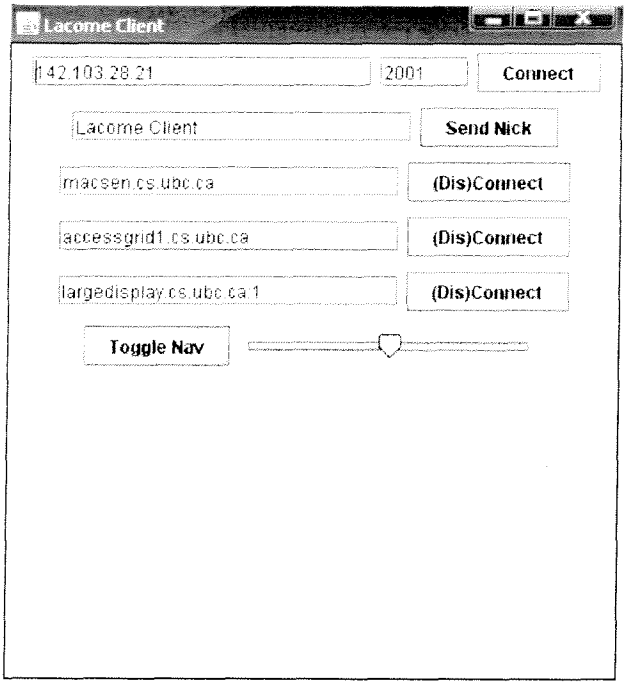

Figure 4.10: Lacome Client

in order to be connected to the server. In later versions, we will add a discovery browser or favorite list to enable quick connection to the Lacome server. Users are also able to choose a nickname for others to use to reference them on the Lacome server. See Figure 4.11 for a screenshot of the text field.

Navigator Control. This GUI component is used to toggle between the *Navigator* and the client's system cursor. When a user clicks the *Toggle Nav* button, the system cursor will be mounted into the rectangular region (highlighted with a yellow background of the user's screen) as shown in

Figure 4.11: Server Connection Component

Figure 4.12. When the system cursor reaches the boundary of the rectangle, it will be reset to the center of the rectangle. Thus the system cursor will not be able to get out of the rectangle. As a result, the cursor is virtually in an unbounded 2D plane. Meanwhile, the relative displacement of the system cursor movement as well as the mouse button events are sent to the Lacome server. The Lacome server will draw a virtual cursor on the shared large display which updates accordingly.

For the same reason as in *Controller-Manipulator* switching, a user needs a unique hot key combination to switch between the *Navigator* and the local system cursor. Currently we have chosen *Shift+Backspace* to achieve this. The control bar beside the toggle button is used to adjust the size of the *Navigator* cursor. Users sitting at different distances to the large display may need to adjust the size of theirs cursors accordingly. We have only implemented the cursor size customization feature in our client. There may be other customization options which we could add in the future.

Publisher Control. Using this component, a user is able to instruct the Lacome server to connect to any available VNC server as long as he/she has the password to that VNC server. The machine on which the VNC server resides does not need to be a Lacome client (see Figure 4.13). This is an optional component. For those users who do not need to share desktops

Figure 4.12: Navigator Control Component

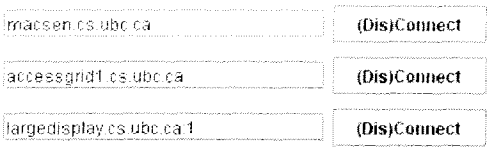

Figure 4.13: Publisher Control Component

or application windows, a compact version of the Lacome client can be used that does not have this component.

Role-Based Configuration

Among the functional components described above, the *Server Connection* and *Navigator Control* are mandatory for a client, while the *Publisher Control* is optional. According to different scenarios, we can choose to either include the *Publisher Control* into the client or not. In the default system configuration like the one shown in Figure 3.1, the clients normally

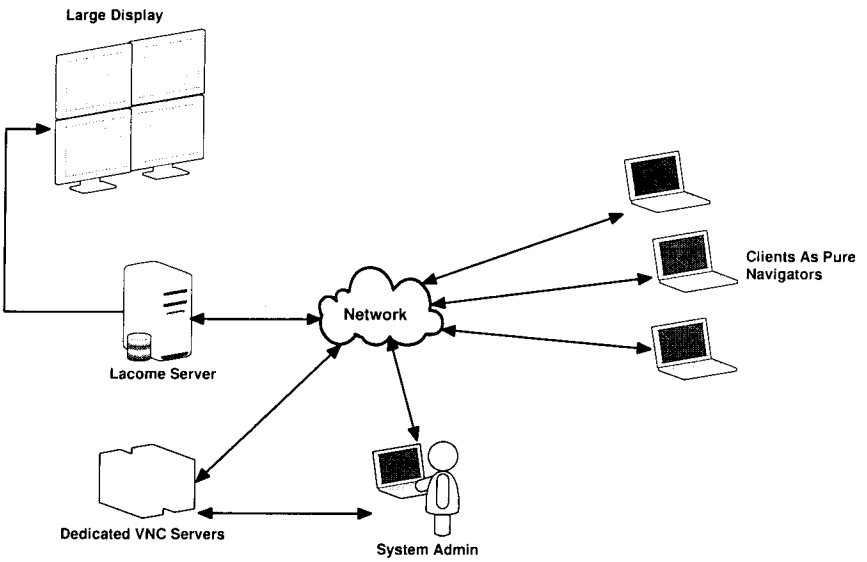

Figure 4.14: Configuration with a System Administrator

include the *Publisher Control* component, but in a meeting with a system administrator and several dedicated machines as servers, we may remove the *Publisher Control* component from the clients and only make this option available to the administrator. In that case, the system administrator is responsible for bringing all shared content to the meeting. Each user will use his/her laptop to connect to the Lacome server as a pure *Navigator*, as shown in Figure 4.14.

4.4 Summary

In this chapter we have elaborated the design details of the interfaces in Lacome. With this and Chapter 3 we now have a relatively complete un-

derstanding on both the internal system architecture and the external user interface. In next chapter, we discuss specific details of the implementation and the experience we have gained from our work.

Chapter 5

Implementation

In the previous two chapters we introduced the Lacome system architecture and the user interface, respectively. In this chapter we discuss some of the details, as well as the difficulties we encountered implementing the system. Aditional details are in the source code, which is commented. We first describe the hardware and software environment for our implementation, and then we present selected details of the implementation.

5.1 Hardware

The Lacome system is part of the collaboration infrastructure maintained by the Large Shared Display Group (LSD-G) at the University of British Columbia. In our laboratories we have three large tiled display surfaces that are each 16 feet wide by 9 feet tall. Behind each surface there are projector arrays with 12 projectors, each having a resolution of 1280 x 1024 pixels, as shown in Figure 5.1 (one display surface has stereo-capable projectors having a resolution of 1024 x 780 pixels). The projector arrays are connected to an IBM DCV (Deep Computing Visualization) graphics cluster with each display driven by one application node and six rendering nodes comprising dual CPU Intel Xeon 3.6GHz processors with 8Gbytes of memory. Including

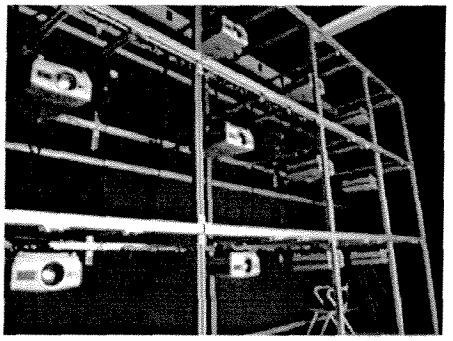

Figure 5.1: Projector Array

an additional administrative node, there are 22 dual CPU processors, all communicating through an Infiniband high-speed interconnect [29].

Each rendering node controls the display of two projectors. The Lacome server runs on the application node using the associated rendering nodes to update the display surface. The laboratory is part of the UBC campus wireless network so users easily gain connectivity to the Lacome system. In addition, there are some desktop PCs in the laboratory that can be used as clients or data servers to test the system. This accommodates a variety of meeting requirements. The configuration is illustrated in Figure 5.2. The IBM graphics cluster provides a powerful processing capability that we have used in our research. However, the Lacome system is designed to be largely independent of the particular hardware environment. A variety of high performance architectures could be used instead for large display surfaces, or just a single projector driven by a personal computer could be used for smaller display surfaces.

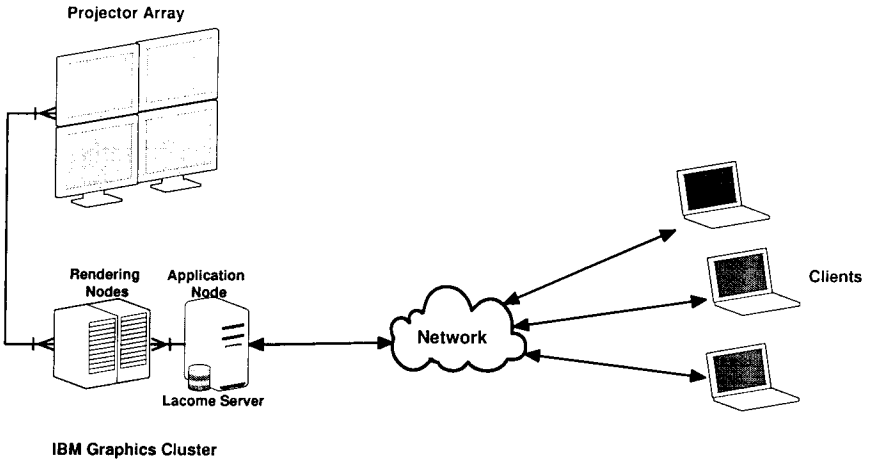

Figure 5.2: A typical hardware configuration for Lacome

5.2 Software

The Lacome server and its clients are implemented separately, using different approaches because of the different purposes they have. The server is implemented in C++ under Red Hat gcc-4.1.1 in order to accommodate the working environment of the application node of the IBM graphics cluster; the client is implemented in Java to achieve cross-platform functionality.

DCV

The IBM Deep Computing Visualization (DCV) software [29] installed on the IBM graphics cluster is used to manage the parallel rendering for the projector array. The architecture is designed so we can consider the display of the shared workspace onto the large screen to be a black box. We describe some of the salient details of the server and the client to illustrate some of

the choices made in the architecture.

Server

The current version of the Lacome server contains about 5000 lines of commented C++ code, excluding the ivLib graphics library. The implementation has three components as described in Chapter 3. These are discussed in turn here.

Network Component

The Network Component is implemented based on the RFB protocol. Currently most standard implementations of the VNC [6, 9, 12, 13] are configured so that one or more VNC clients can connect to a VNC server but a VNC client can only connect to one VNC server. In the Lacome architecture the VNC servers are actually running as Lacome *Publisher* clients, while the VNC client software is running on the Lacome server. Thus the notion of "client"and "server" is reversed from its normal meaning with respect to VNC. This distinction should be kept in mind during the rest of this chapter to avoid any confusion.

The Lacome server has to realize multiple VNC client connections, one to each VNC server that is an active *Publisher*. At first we manually created multiple VNC clients on the server side, but after developing the Network Component, we employ multi-threading to dynamically allocate multiple connections to the VNC servers, each of which runs on a Lacome client. This is realized by a *Publisher Manager* that receives a connection request from a Lacome client. The request includes the IP address and port for a re-

mote VNC server. The *Publisher Manager* creates a new thread running an instance of a standard VNC client and establishes a network connection between that thread and the remote *Publisher*. When a *Publisher* disconnects, the system resources are recycled and the thread is terminated.

The same approach is used by the *Navigator Manager* to dynamically create a new thread to manage each remote *Navigator* after receiving a connection request from a Lacome client.

These two are deleted when the remote client terminates. In this way the Network Component can provide connection-on-demand services so users can join and quit a meeting at any time.

Access to VNC servers is normally controlled through password authentication. Currently we assume a default password for all VNC servers that connect to Lacome. It makes sense for a co-located collaboration among a group of co-workers who trust each other to share the same password. Future versions will need to implement more stringent password protection if Lacome is to be used on a public network to support multi-group distributed collaboration.

In a more robust implementation either the Lacome client that initiates a connection to a VNC-based *Publisher* would provide its password, or the Lacome server might provide a repository for passwords. There are advantages to each approach. If the Lacome server is trusted, it might be preferable to have all password authentication done by it, with Lacome clients accessing *Publishers* only through the the Lacome server. If the Lacome server is not trusted, it is problematic how it could connect to a *Publisher* without having the *Publisher*'s password, unless a client acted as an intermediary. This is

not supported by the current architecture, which has the Lacome server as the central control point for communication with all *Publishers*.

Data Component

The Data Component must support full Lacome clients with *Navigator* and *Publisher* components, as well as *Publishers* that do not reside on machines running a Lacome client. The Data Component is designed to manage data efficiently from a number of perspectives.

The Data Component keeps track of the state of each client and makes this information available to the other components. The primary data is the list of *client connections*. As described in Section 3.2, a Lacome client may be associated with one *Navigator* and an arbitrary number of *Publishers*. Each has a connection to the Lacome server. These connections are recorded in the Data Component. Every *Publisher* and every *Navigator* must have a unique Lacome client as its owner, as well as a corresponding connection in the Network Component. *Publishers* and *Navigators* connected to a Lacome server will be disconnected if their owner client disconnects. The Data Component initiates the disconnections when it detects that a *Navigator* or *Publisher* no longer has a Lacome client with an active connection. Figure 5.3 illustrates the relationship between data sets in the Data Component. Lacome clients, *Navigators* and *Publishers* are maintained in three sets. Each *Navigator* or *Publisher* is "owned" by a client. Note that a *Publisher* does not have to be in the same machine with its owner client, but a *Navigator* has to reside in the same machine with its owner client. Maintaining a single set for each data type ensures no duplication occurs.

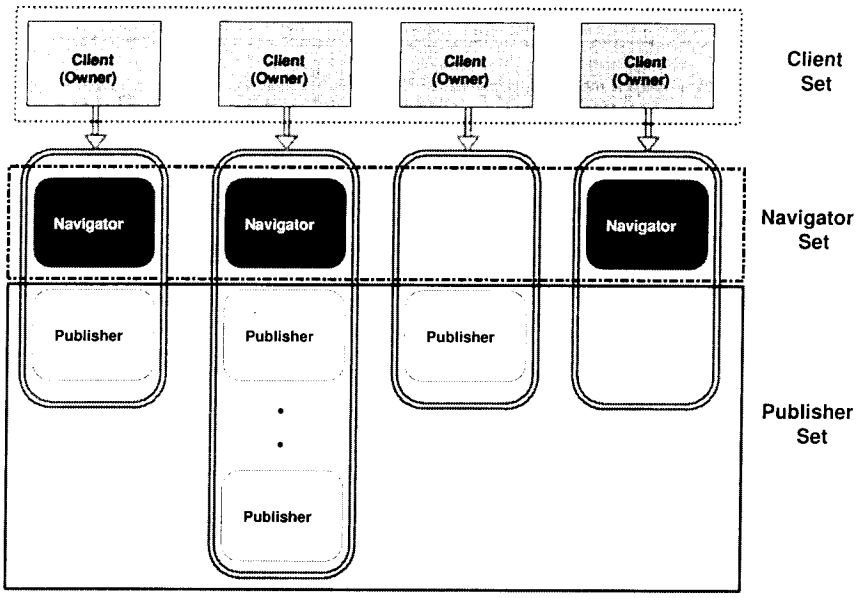

Figure 5.3: Data Structures Maintained in Data Component

It is also easy to implement extensions based on this model.

A *Publisher* may be iconified (or not) and it may be under the control of a *Navigator* (or not), and it has a current location and size. Similarly a *Navigator* may be in one of three states, corresponding to whether it is in *Manipulator*, *Controller*, or *Annnotator* mode. If it is a *Manipulator* or *Controller* it is associated with the *Publisher* it is manipulating or controlling.

Rendering Component

The Rendering Component uses a light weight OpenGL based graphics library ivLib [61] to draw graphics primitives. Specifically, we only make use

of the vector and matrix class defined in the library. The library is independent of the Lacome system so any other suitable graphics library could be used in a Lacome implementation. Everything we draw on the screen is in pure OpenGL. Most of the rendering is from bitmaps provided by the RFB protocol, so mostly we rely on texture-mapped rectangles to render images of what *Publishers* send us with some additional vector graphics for cursors and borders of windows. A display function is implemented to provide a mechanism to draw all the objects on the shared display, including windows and cursors on the different layers.

To effectively use the IBM DCV system, our system must be a standard OpenGL program. Following the OpenGL framework, the Rendering Component resides in the main thread. However, this can be changed based on the system requirements of a particular Lacome implementation. The rendering function gathers data such as window positions, cursor colors, etc., from the Data Component and displays them on the OpenGL window. The window is refreshed every time a new RFB frame has arrived or when any of the client states changes.

Client

Our goal in designing the Lacome client was to build a light-weight, platform-independent application with minimal functionality so most of the processing would be done on the Lacome server. We implemented our Lacome client in Java. The current source code contains about 1000 lines of commented Java code. In previous chapters the structure and design of the Lacome client was discussed. We describe a few of more implementation details in

following sub-sections.

Main Connection

A Lacome client must first connect to the Lacome server. This is done by opening a socket. After a successful connection, the Lacome client can start invoking other features by initiating connections for its Navigator or for one or more Publishers.

Publisher Registration

As described in Chapter 4, a Lacome client can register an arbitrary number of *Publishers* with a Lacome server. The Lacome client uses its socket connection to tell the Lacome server the IP address and port number of a VNC server running on the *Publisher* machine. In doing so the Lacome client does not directly connect to the *Publishers*. This keeps the networking and processing overhead relatively low. It also means that we do not need to install the Lacome client on every machine that we want to use as a source of data for the shared display. This is especially useful for some machines that are not physically present, but may still have useful information for a meeting.

Navigator

The *Navigator* uses a custom version of the RFB protocol to talk to the Lacome server. The client wraps all user input events in a format similar to the normal RFB protocol and sends them to the Lacome server. We made this implementation decision because we want to minimize the processing

overhead when we are performing input redirection. The only information that needs to recalculated from what is provided by the Navigator's host windowing system is the cursor coordinates with respect to the shared screen based on the size and location of the window the Navigator is controlling.

5.3 Implementation Tips and Experience

This section is written for researchers who intend to delve into the implementation details of the Lacome system. The source code of Lacome is licensed under the GPL and is free for distribution. However, in order to use the graphics library that Lacome uses, one must have a license for ivLib [61] according to its software license agreement. Otherwise it can be replaced with some other graphics library. The following tips are useful when using our current Lacome implementation or developing further features based on it:

- All external libraries or open source code should be maintained in a highly independent form. In our implementation we use two external libraries, namely the ivLib graphics library and the RFB protocol library. We have tailored both of them to fit our use (for example, we eliminate the display of the remote desktop on the VNC client side). In addition, we have made them as minimally coupled with other modules as possible so that future updates on the library will not affect the code base much.

- Different versions of the same library might not be compatible with each other and that may raise issues. In our case, the GL library

on the DCV system cannot recognize the routines *glIsTexture* and *glBlendColor* while the library on our developing machine can. We have to compromise our code to achieve the most compatibility when we cannot upgrade the library on the machine that we might not be able to configure.

- The bmp image texture has a restriction that the size (both width and height) of the texture must be a power of 2. This is a popular issue in computer graphics. Some extensions of OpenGL have already provided routines to solve this problem. In our system we don't have certain extensions in our GL library, so we solve this issue simply by using *glTexImage2D* to create an empty texture with size a power of 2, and using *glTexSubImage2D* to update the image texture. This gives us an advantage since *glTexSubImage2D* waives the restriction on the size of the input data.

- In order to make the OpenGL window update repeatedly we use the callback routine *glutIdleFunc* instead of *glutTimerFunc*, since we don't know when the frames arrive.

- In an OpenGL window, the Y axis is pointing upward; while in the VNC desktop window, the Y axis is pointing downward.

- We use threads extensively in our Lacome implementation. Keeping the system thread-safe is crucial throughout the development procedure and should be paid high attention in further extensions. Mutexes and condition variables are widely used in our code base. However,

we notice that excessive use of them will potentially slow down the program and thus impact the efficiency of the system. We currently maintain the minimum number of mutexes and condition variables in our code base via careful design. Future development should also make sure they are not abused.

- During the development of the client, we were trying to make the *Navigator* mouse panel transparent and hide the cursor. We wish to provide minimal obstruction to the workspace. However, even in Java, these functionalities cannot be implemented in a cross-platform fashion. Moreover, we would have to install extra packages on each Linux machine running the Lacome client to support transparency. In addition, mouse right clicks on a transparent window on a Mac machine is not detectable.

- Currently in our implementation, we use a three-bit binary code to represent the color of the *Navigator*'s cursor. More colors can be easily added by extending the number of bits of the binary code. However, care must be taken to ensure that the chosen colors are easily distinguishable.

- The hot key combination we set for the *Navigator* to toggle between different modes is done by taking all hot key combinations from popular applications on multiple platforms into consideration. We are confident that it works in most cases. Further extensions should also be careful in deciding similar platform independent features.

5.4 Summary

In future versions of the system we plan to integrate an enterprise version of RealVNC [9] to support application-level window sharing and collaboration, rather than only full desktop sharing.

In this chapter, the implementation details of the Lacome system have been discussed. In addition, we have listed some tips that we believe will be helpful to colleagues who intend to work in related projects. In the next chapter we will talk about our early system experience with pilot users. After that we draw conclusions from our work and discuss the further extensions of the current Lacome system.

Chapter 6

System Experiences, Conclusion and Future Work

In Chapters 3, 4 and 5 we discussed the design and implementation of the structure and interface of the Lacome system. In this chapter we discuss our experience using the system and draw some conclusions about the project. We then describe several avenues for future development.

6.1 System Experiences

We have successfully run a demo session in our laboratory with the Lacome system (see Figure 6.1) and invited our colleagues to participate in the demo. Eight people (consisting of faculty members, technical staff and graduate students, all with computer science or engineering backgrounds) came in with their laptops or used the dedicated testing machines and joined the demo session. First, we asked participants to download and run our Lacome client from the project webpage[1]. The Lacome client is written in Java and requires a Java Runtime Environment(JRE) [11] installation on the participants' machines. However, JRE is very common and all our par-

[1] http://www.cs.ubc.ca/~zephyr/lacome

ticipants had JRE already installed and experienced no difficulties running our Lacome client on their machines no matter which platform they were using. Next, we provided several dedicated *Publishers* and allowed our participants to freely register those *Publishers* onto the Lacome server and manipulate those windows with their *Navigator* cursor in the shared workspace on the large display. After a brief introduction on the Lacome system and explanation on the hot-key configurations, participants were asked to try any features they wanted. They were able to simultaneously interact with the windows smoothly in the shared workspace without obvious interference from other participants. Those participants who were interested in trying to share their desktop specific application windows were asked to install a VNC server on their machine. After that they could choose to share a window from their machines to the shared workspace. Thus, other participants could either manipulate the window or enter the *Controller* mode to control the shared window's machine. We employed the user-level communication principle that if a participant intended to access into another participant's computer, they should communicate verbally and the subject would disconnect his *Navigator* from the server before the other participant got in. Thus we avoided encountering security issues and undefined behaviors brought by multiple input redirections.

Participants have provided positive comments as well as some constructive suggestions including enabling the privacy control so that no default password would be used, refining the layout of the client interface to enable adding arbitrary number of *Publishers* instead of 3, and adding some visual/audio indication when users are switching modes to support the aware-

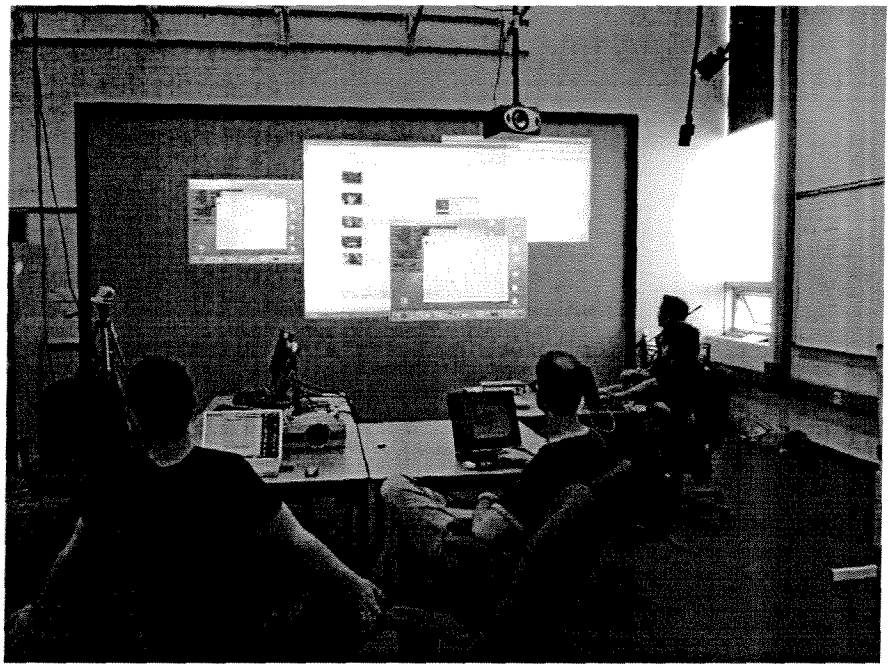

Figure 6.1: System Demo With Group Users

ness. In Section 6.3 we will talk about more future extensions of the Lacome system.

6.2 Conclusion

So far in this thesis, we have discussed the design and implementation of the Lacome system which supports multi-user cross-platform collaboration with a shared display and a number of personal computers.

Lacome supports separate types of interaction through a Lacome client for window management tasks on the shared display (move, resize, iconify,

deiconify) and for application interactions through the VNC servers. The system architecture provides for *Publishers* that share information and *Navigators* that access information. A Lacome client can have either or both, and can initiate additional *Publishers* on other VNC servers that may not be Lacome clients. Explicit access control policies on both the server side the client side provide a flexible framework for sharing. The architecture builds on standard cross-platform components such as VNC and JRE. Interaction techniques used in the window manager ensure simple and transparent multi-user interactions for managing the shared display space.

The original idea of Lacome was inspired by the Integrated Tabletop [39] proposed by Nakashima et al. Compared with their system, Lacome has achieved a number of enhancements. Firstly, their system only supports Windows; while Lacome is a cross-platform system. Secondly, their system requires the installation of a modified VNC server on user's computer; while in Lacome any standard VNC server will work and no specialized modification is needed. Last but not the least, their system has only one control layer and the window manipulation takes place only when user drags the margin of the windows; while Lacome has multiple control layers to better support different control modes.

An OpenGL based server has been developed to enable the sharing of multiple windows and manipulating via multiple cursors. We currently run the sever on our Red Hat Linux system and it should be able to migrate to other platforms with few or no modifications. Lacome provides a cross-platform, light-weight, setup-free client for the end users to easily get involved in collaborative interaction with the shared display. The client pro-

vides a user-friendly and simple interface that is fully-functional and easy to use. We have designed Lacome with flexibility to accommodate different system requirements and configurations based on a broad literature review in the related areas such as SDG and Multi-Display Environments. The thesis describes the structure and interface design of all the components of the Lacome server and client module. We have also elaborated the implementation details of the Lacome system. Moreover, we provide the experiences we have gained during the development of the Lacome system and hope they are useful for other colleagues. Early experience from our demo participants has indicated the promising future of our Lacome and encourages us to take more efforts into further development.

6.3 Future Work

The successful development of the Lacome system has aroused several possible extensions in certain directions. In this section we are going to describe those we believe should be completed in the next steps.

Access Control Support

In current version of Lacome, we do not have a full implementation of the access control module. However, this has already been taken into consideration during the design phase and the framework has already been created in the code base. Next we are going to implement the module on the framework which needs no modification on the current system.

Privacy Control in Document Sharing

In our usage scenario, there is a privacy concern. When a participant shares his/her desktop or application window onto the shared workspace other participants can see all the information on it. However, sometimes the provider only wants to show part of the information in the window and hide others. For example, a project manager wants to present the project budget to his/her team. The budget includes other project costs as well as the employees' salaries. The manager wants to show other costs and the total salary amount but not the details of individual employee's. WinClone introduced by Berry et al. [17] and extended to work distributively by Lanir et al. [35] aimed to address this issue. We plan to integrate WinClone with Lacome to enable privacy control in Lacome. Since WinClone is also VNC based software, it should not be hard to make the combination.

Multi-Cursor Support in a Single Client

Currently, Lacome supports each client with one *Navigator* (cursor) on the shared workspace. Moreover, only one cursor can get into a shared window at any given time, constrained by the platform restriction. As described inSection 2.3, MPX allows multi-cursor manipulation on a single desktop running under Linux. Integrating MPX into Lacome will bring more flexibility and diversity to our meeting environment. Several participants can then choose to either use a single machine to navigate on the shared workspace each with their own cursor or get into the same machine and manipulate it simultaneously.

Handheld Device Support

Handheld device support is the immediate extension of our Lacome system. Sometimes participants may not bring a laptop with them, but they may have a PDA in their pocket. To support handheld devices in Lacome, we need to develop another version of our client for the handheld device. Fortunately, our Lacome client is written in Java, so it would be fairly simple to convert it to a J2ME version.

Video and Audio Support

Access Grid is an ensemble of resources including multimedia large-format displays, presentation and interactive environments, and interfaces of Grid middleware and visualization environments [1]. These resources are used to support group-to-group interactions across the Grid. The WestGrid [14] community is using the Access Grid for video conferences and virtual seminars. Currently the seminars are supported with video and audio feeds provided by the Access Grid system plus the shared slides presentation provided by a VNC client. One possible next step is to try to enable video and audio support in the Lacome system and ultimately integrate Lacome into the future version of the Access Grid system.

Annotator Mode

The emphasis in our current work has been on the *Manipulator* and *Controller* modes for Lacome clients. The *Manipulator* mode allows each user to manipulate the arrangement (location, size, and iconification) of the shared

display objects (desktops or application windows). The *Controller* mode provides access to the applications "underneath" the display objects that run on *Publisher* clients. The *Annotator* mode works only on the shared display by overlaying information ("annotations") either on the full screen of the shared display, or on top of display objects, perhaps with some degree of transparancy so both the underlying content and the annotations can be seen together. Annotations on the full screen are in absolute locations and sizes, whereas annotations on a display object are relative to that object and scale accordingly when the object is scaled.

We have not yet implemented *Annotator* mode because it does not require any changes to the framework and thus was not of particular interest for this stage of our research. Like *Navigator* mode, *Annotator* mode uses mouse and keyboard information from a *Navigator* as the input for manipulating the shared display, but in this case it is running a custom drawing or sketching program that supports shared annotation by multiple users. There are many examples of these, ranging from basic "what you see is what you get" systems [22] to more elaborate structured drawing programs that maintain a model of the annotations with explicit concurrency control [30]. Any of these approaches could be implemented within the Lacome framework.

6.4 Summary

In this chapter, we have discussed the early user experiences with the Lacome system and drawn our conclusions from the current phase of the project. We consider the development of the Lacome system as a successful step in

the research of interactive group collaboration tools. The promising future extensions show the flexibility and extensibility of the Lacome system.

Bibliography

[1] Access Grid. http://www.accessgrid.org, retrieved 2007-10-21.

[2] Apple Dashboard Widgets. http://www.apple.com/downloads/dashboard/, retrieved 2007-11-24.

[3] Apple iPhone. http://www.apple.com/iphone, retrieved 2007-10-28.

[4] Microsoft Surface PC. http://www.microsoft.com/surface/, retrieved 2007-08-11.

[5] OpenGL. http://www.opengl.org/, retrieved 2007-08-08.

[6] OSXVNC. http://www.redstonesoftware.com/, retrieved 2007-08-16.

[7] Perceptive Pixel. http://www.perceptivepixel.com, retrieved 2007-08-11.

[8] PowerWall. http://www.lcse.umn.edu/research/powerwall/powerwall.html, retrieved 2007-08-12.

[9] RealVNC. http://www.realvnc.com/, retrieved 2007-08-16.

[10] Smart Technologies, SMART Board. http://www2.smarttech.com/st/en-US/Products/SMART+Boards/, retrieved 2007-09-26.

[11] Sun Microsystems Java Runtime Environment. `http://java.sun.com/j2se/desktopjava/jre/index.jsp`, retrieved 2007-10-17.

[12] TightVNC. `http://www.tightvnc.com/`, retrieved 2007-08-16.

[13] UltraVNC. `http://www.ultravnc.com/`, retrieved 2007-08-16.

[14] WestGrid. `http://www.westgrid.ca`, retrieved 2007-10-21.

[15] Ritchie Argue. Advanced Multi-display Configuration and Connectivity. Master's thesis, Dalhousie University, August 2007.

[16] Hrvoje Benko, Andrew D. Wilson, and Patrick Baudisch. Precise Selection Techniques for Multi-touch Screens. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1263–1272, New York, NY, USA, 2006. ACM.

[17] Lior Berry, Lyn Bartram, and Kellogg S. Booth. Role-based Control of Shared Application Views. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 23–32, New York, NY, USA, 2005. ACM Press.

[18] Jacob T. Biehl and Brian P. Bailey. ARIS: An Interface for Application Relocation in an Interactive Space. In *GI '04: Proceedings of Graphics Interface 2004*, pages 107–116, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[19] Kellogg S. Booth, Brian D. Fisher, Chi Jui Raymond Lin, and Ritchie Argue. The "Mighty Mouse" Multi-screen Collaboration Tool. In *UIST*

'02: Proceedings of the 15th annual ACM symposium on User interface software and technology, pages 209–212, New York, NY, USA, 2002. ACM Press.

[20] Mary Czerwinski, George Robertson, Brian Meyers, Greg Smith, Daniel Robbins, and Desney Tan. Large Display Research Overview. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 69–74, New York, NY, USA, 2006. ACM Press.

[21] Paul Dietz and Darren Leigh. DiamondTouch: A Multi-user Touch Technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226, New York, NY, USA, 2001. ACM Press.

[22] Saul Greenberg and Ralph Bohnet. GroupSketch: A Multi-user Sketchpad for Geographically Distributed Small Groups. In *Graphics Interface '91*, pages 207–215, June 1991.

[23] François Guimbretière, Maureen Stone, and Terry Winograd. Fluid Interaction with High-resolution Wall-size Displays. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 21–30, New York, NY, USA, 2001. ACM Press.

[24] Vicki Ha, Kori Inkpen, Jim Wallace, and Ryder Ziola. Swordfish: User Tailored Workspaces in Multi-display Environments. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1487–1492, New York, NY, USA, 2006. ACM Press.

[25] Jefferson Y. Han. Low-cost Multi-touch Sensing Through Frustrated Total Internal Reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, New York, NY, USA, 2005. ACM.

[26] Jefferson Y. Han. Multi-touch Interaction Wall. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Emerging technologies*, page 25, New York, NY, USA, 2006. ACM.

[27] Peter Hutterer. Mpx Blog. `http://wearables.unisa.edu.au/mpx/`, retrieved 2007-08-16.

[28] Peter Hutterer and Bruce H. Thomas. Groupware Support in the Windowing System. In *AUIC '07: Proceedings of the eight Australasian conference on User interface*, pages 39–46, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[29] IBM. Deep Computing Visualization. `http://www-03.ibm.com/servers/deepcomputing/visualization/index.html`, retrieved 2007-08-08.

[30] Claudia-Lavinia Ignat and Moira C. Norrie. Draw-together: Graphical Editor for Collaborative Drawing. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 269–278, New York, NY, USA, 2006. ACM.

[31] Shahram Izadi, Steve Hodges, Alex Butler, Alban Rrustemi, and Bill Buxton. ThinSight: Integrated Optical Multi-touch Sensing Through

Thin Form-factor Displays. In *EDT '07: Proceedings of the 2007 workshop on Emerging displays technologies*, page 6, New York, NY, USA, 2007. ACM.

[32] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):67–74, 2002.

[33] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. PointRight: Experience with Flexible Input Redirection in Interactive Workspaces. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 227–234, New York, NY, USA, 2002. ACM Press.

[34] Russell Kruger, Sheelagh Carpendale, Stacey D. Scott, and Saul Greenberg. How People Use Orientation on Tables: Comprehension, Coordination and Communication. In *GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 369–378, New York, NY, USA, 2003. ACM Press.

[35] Joel Lanir, Lior Berry, and Kellogg S. Booth. WinClone: Role-based Control of Distributed Application Views. CSCW'06 demo, Banff, 2006.

[36] Marilyn Mantei. Capturing the Capture Concepts: A Case Study in the Design of Computer-supported Meeting Environments. In *CSCW '88: Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, pages 257–270, New York, NY, USA, 1988. ACM Press.

[37] Michael McGuffin and Ravin Balakrishnan. Acquisition of Expanding Targets. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 57–64, New York, NY, USA, 2002. ACM Press.

[38] Brad A. Myers, Herb Stiel, and Robert Gargiulo. Collaboration Using Multiple PDAs Connected to a PC. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 285–294, New York, NY, USA, 1998. ACM Press.

[39] Kousuke Nakashima, Takashi Machida, Kiyoshi Kiyokawa, and Haruo Takemura. A 2D-3D Integrated Environment for Cooperative Work. In *VRST '05: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 16–22, New York, NY, USA, 2005. ACM Press.

[40] Tao Ni, Doug A. Bowman, and Jian Chen. Increased Display Size and Resolution Improve Task Performance in Information-rich Virtual Environments. In *GI '06: Proceedings of Graphics Interface 2006*, pages 139–146, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[41] Anne Marie Piper, Eileen O'Brien, Meredith Ringel Morris, and Terry Winograd. SIDES: A Cooperative Tabletop Computer Game for Social Skills Development. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 1–10, New York, NY, USA, 2006. ACM Press.

[42] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, Extensibility and Robustness in iROS. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 11, Washington, DC, USA, 2003. IEEE Computer Society.

[43] Holger Regenbrecht, Michael Wagner, and Gregory Baratoff. MagicMeeting: A Collaborative Tangible Augmented Reality System. *Virtual Reality*, 6(3):151–166, 2002.

[44] Jun Rekimoto. SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 113–120, New York, NY, USA, 2002. ACM Press.

[45] Jun Rekimoto and Masanori Saitoh. Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 378–385, New York, NY, USA, 1999. ACM Press.

[46] Tristan Richardson. The RFB Protocol. `http://www.realvnc.com/docs/rfbproto.pdf`, retrieved 2007-08-11.

[47] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[48] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A Middleware In-

frastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83. 2002.

[49] Mark Roseman and Saul Greenberg. Building Real-time Groupware with GroupKit, a Groupware Toolkit. *ACM Transaction on Computer-Human Interaction*, 3(1):66–106, 1996.

[50] Stacey D. Scott, Karen D. Grant, and Regan L. Mandryk. System Guidelines for Co-located, Collaborative Work on a Tabletop Display. In *ECSCW'03: Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work*, pages 159–178, Norwell, MA, USA, 2003. Kluwer Academic Publishers.

[51] Chia Shen, Frédéric D. Vernier, Clifton Forlines, and Meredith Ringel. DiamondSpin: An Extensible Toolkit for Around-the-table Interaction. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 167–174, New York, NY, USA, 2004. ACM Press.

[52] Mark Stefik, Gregg Foster, Daniel G. Bobrow, Kenneth Kahn, Stan Lanning, and Lucy Suchman. *Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings*, pages 335–366. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[53] Jason Stewart, Benjamin B. Bederson, and Allison Druin. Single Display Groupware: A Model for Co-present Collaboration. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 286–293, New York, NY, USA, 1999. ACM Press.

[54] Norbert A. Streitz, Jörg Geißler, Torsten Holmer, Shin'ichi Konomi, Christian Müller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-LAND: An Interactive Landscape for Creativity and Innovation. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 120–127, New York, NY, USA, 1999. ACM Press.

[55] Kishore Swaminathan and Steve Sato. Interaction Design for Large Displays. *Interactions*, 4(1):15–24, 1997.

[56] Desney S. Tan, Darren Gergle, Peter Scupelli, and Randy Pausch. With Similar Visual Angles, Larger Displays Improve Spatial Performance. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 217–224, New York, NY, USA, 2003. ACM Press.

[57] Desney S. Tan, Darren Gergle, Peter Scupelli, and Randy Pausch. Physically Large Displays Improve Performance on Spatial Tasks. *ACM Trans. Comput.-Hum. Interact.*, 13(1):71–99, 2006.

[58] Desney S. Tan, Brian Meyers, and Mary Czerwinski. WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1525–1528, New York, NY, USA, 2004. ACM Press.

[59] Edward Tse and Saul Greenberg. Rapidly Prototyping Single Display Groupware Through the SDGToolkit. In *AUIC '04: Proceedings of*

the Fifth Australasian User Interface Conference, pages 101–110, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[60] Edward Tse, Saul Greenberg, and Chia Shen. GSI Demo: Multiuser Gesture/Speech Interaction Over Digital Tables by Wrapping Single User Applications. In *ICMI '06: Proceedings of the 8th international conference on Multimodal interfaces*, pages 76–83, New York, NY, USA, 2006. ACM Press.

[61] James M. Van Verth and Lars M. Bishop. *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide.* Morgan Kaufmann, March 2004.

[62] Grant Wallace, Otto J. Anshus, Peng Bi, Han Chen, Yuqun Chen, Douglas Clark, Perry Cook, Adam Finkelstein, Thomas Funkhouser, Anoop Gupta, Matthew Hibbs, Kai Li, Zhiyan Liu, Rudrajit Samanta, Rahul Sukthankar, and Olga Troyanskaya. Tools and Applications for Large-scale Display Walls. *IEEE Comput. Graph. Appl.*, 25(4):24–33, 2005.

[63] Grant Wallace, Peng Bi, Kai Li, and Otto Anshus. A Multi-cursor X Window Manager Supporting Control Room Collaboration. Computer Science Report No. TR-0707-04, Princeton University, 2004.

[64] Andrew D. Wilson. Playanywhere: A Compact Interactive Tabletop Projection-vision System. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 83–92, New York, NY, USA, 2005. ACM.

[65] Andrew D. Wilson. Robust Computer Vision-based Detection of Pinching for One and Two-handed Gesture Input. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 255–258, New York, NY, USA, 2006. ACM.

[66] Mike Wu and Ravin Balakrishnan. Multi-finger and Whole Hand Gestural Interaction Techniques for Multi-user Tabletop Displays. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 193–202, New York, NY, USA, 2003. ACM Press.

Appendix A

Message Protocol of Lacome

Lacome uses a message protocol that extends the existing RFB protocol used by VNC. Each message has at least one integer header that identifies the message type followed optionally by additional fields that have type-specific information.

Send Nick Name	int	int	int	byte[ ]
	1	4	string number in bytes	string

Send Cursor Size	int	int	float
	1	6	size

Request VNC Connection	int	int	int	int	byte[ ]
	2	0	session id	string number in bytes	string

Terminate VNC Connection	int	int	int
	2	1	session id

Enable *Navigator*	int	int
	3	1

Disable *Navigator*	int	int
	3	0

Send Mouse Event	int	int	int	int
	64	dx	dy	button mask

Send Key Down Event	int	int	int
	65	key code	1

Send Key Up Event	int	int	int
	65	key code	0

Switch Out From *Controller* mode	int
	5

Send Toggle Iconification Request	int	int	int
	-6	dx	dy