A Multi-Cursor* X Window Manager Supporting Control Room Collaboration

Grant Wallace, Peng Bi, Kai Li and Otto Anshus

Princeton Universtiy Computer Science
{gwallace,pbi,li,otto}@cs.princeton.edu

ABSTRACT

Existing window systems have been designed to support a single user and they are not suitable for control room collaboration. This paper describes how to extend the X-window system to support multiple simultaneous cursors. In a collaborative control room environment a multi-cursor X window system can allow multiple users to simultaneously drag, control and input to windows on a shared display. This paper discusses the design and implementation of a multi-cursor X window manager. Our early experience at a fusion control room show that our multi-cursor X window manager is an effective approach to support control room collaboration.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Synchronous interaction, Collaborative computing, Computer-supported cooperative work*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Windowing systems, Input devices and strategies, Graphical user interfaces (GUI)*; H.4.1 [**Information Systems Applications**]: Office Automation—*Groupware, Workflow management*; D.4.4 [**Operating Systems**]: Communications Management—*Input/Output*

Keywords

Multi-cursor, Multi-user, Simultaneous input, Synchronous input, Window manager, Desktop, Shared display, Group collaboration, X11, Xserver

1. INTRODUCTION

Ever since Douglas Engelbart's ground breaking demonstration in 1968 of the mouse, the cursor and remote collaboration [6], people have been inspired to create collaborative computing systems. The proliferation of computers, commodity software, and networks have provided an underlying platform for computer supported cooperation and sharing. However, in the advance of personal desktop computers, system support for multi-user input and collaboration has lagged behind. Today's window systems are single cursor based, assuming a single user per display. Although window systems such as X windows [15] and Microsoft Windows allow remote clients to show windows on a shared display, multiple clients have to time share the single cursor, each taking her turn to control windows or enter inputs. The serial input procedure is a barrier for control room collaboration where a large number of users need to show results generated from multiple applications simultaneously in order to make time-critical decisions.

As part of the Scientific Discovery through Advanced Computing (SciDAC) FusionGrid [16] project, we have been working closely with Fusion scientists at several DOE Energy Research labs investigating the requirements for technology to improve collaboration within fusion control rooms. Fusion control rooms are dynamic places where large amounts of data must be periodically generated and analyzed and group decisions made and carried out within short time intervals. Since the fusion experiments are precious and costly, it is crucial that fusion scientists make the best decisions possible when setting the parameters for the next experiment (or shot). A combination of individual and group data must generally be visible for exploration and comparison. This setting lends itself to a combined set of displays, some public and some private, on which people do their work. Since there are tens of collaborators in such a control room, it is highly desirable to support simultaneous navigation and concurrent control within the shared display environment; and, because users access a variety of existing software tools to analyze the dataset, the collaborative framework must support the simultaneous use of legacy applications.

In response to the needs of a collaborative control room, we have developed a multi-cursor X window manager. This window manager provides for multiple simultaneous cursors at the desktop level and allows multiple users to concurrently interact with all components of a desktop environment including applications, window position and size, and system menus. Our work differs from previous multi-user collaborative software research in that we add the multi-cursor support at the systems level rather than within specialized applications. This has the advantage of allowing si-

*We use the term ***cursor*** throughout the text to designate a combined mouse and keyboard input channel and its associated arrow on the screen.

multaneous control of multiple legacy applications. This is important to researchers in a control room who already have a set of single user tools for data analysis. The multi-user desktop allows them to simultaneously traverse the data set in side by side windows and compare results.

We have designed, prototyped and deployed a collaborative control room system. Our system is based around the multi-cursor window manager and leverages several other existing tools to do application sharing and tiled display alignment. A large, shared tiled-display provides the resolution necessary for simultaneous navigation and group viewing. An application sharing mechanism allows users to quickly move data views between personal and shared displays. The multi-cursor X window manager allows users to concurrently control and navigate independent windows created from multiple applications. Our early experience at the control room of Princeton Plasma Physics Lab (PPPL) shows that the multi-cursor window manager can indeed alleviate the single-cursor user interface bottleneck in a control room and it is an effective and scalable approach to supporting control room collaboration.

In this paper we describe our work on the collaborative control-room software, focusing in particular on the multi-cursor desktop. Section (2) describes related work. Section (3) discusses system requirements and design considerations. In section (4) we describe our implementation of a multi-cursor window manager and relate some system experiences in section (5). Finally, section (6) concludes and describes future work.

2. RELATED WORK

Our work on collaborative control room software builds on a collection of research from several areas. These include research in control room systems, display walls, application-sharing, multi-user applications and collaborative environments.

In the area of simultaneous multi-cursor collaboration there have been many research projects. These include systems such as MMM[4], Liveboard[5], Tivoli[12], KidPad[2], M-Pad[13] and Pebbles[11]. These projects have experimented with users simultaneously interacting with and controlling specialized applications. The applications have been custom designed to support multiple simultaneous cursor interaction. We extend this work by adding multi-cursor support at the desktop level. This allows multiple people to simultaneously interact with the desktop or any applications running on the desktop, including legacy single-user applications. Within a multi-user desktop, users can work independently on different applications or interleave their input to the same application.

There is a collection of work on application-sharing. The research can be generally divided into *collaborative transparent* and *collaborative aware* sharing techniques. Systems such as Xmove[17], SharedX[7], XTV[1], VNC[14] and NetMeeting[10] seek to transparently share views and user interaction on existing applications. Alternatively, collaboratively aware systems typically need to be developed or configured for specific applications but can provide greater flexibility such as independent navigation, simultaneous input and heterogeneous platforms. Examples of these include multi-user games, simulators and command and control systems [3]. We base our application-sharing method on the collaborative transparent model, using Xmove[17], and extend it with a Java graphical user interface to ease use and configuration.

Work has also been ongoing in the area of control room systems. One interesting example in this area is the Courtyard system [18]. It interleaves private and shared displays in a multi-cursor application. They implement a power plant monitoring system. It uses a large shared display for group data visualization and provides for multi-user input; when an item is clicked by a user, detailed information is shown on their personal display. Our work differs from this in that while they develop a specialized application, we want to provide generic systems-level support for multi-user collaboration at the desktop level to avoid specialized application development or re-implementation.

Stanford Interactive Workspaces Project [8] creates a collaborative environment with several shared displays. It provides hardware support for moving personal displays to the shared area and multiplexes cursor input, although there is only one shared cursor. Our work adds system-level support for multiple cursors and integrates software support for application sharing to environments such as these.

Our work also relates to two-handed input research such as [9]. Multi-cursor support at the desktop level can facilitate two-handed input even on current single-input applications such as drawing and illustration packages. For example, one cursor can be used to control brush size and palette selection while the other draws lines or objects. This minimizes the back and forth movement necessary with one cursor when alternately selecting brushes or painting.

Several of the above research areas have been combined into an emerging research community called Single Display Groupware [2]. This area of research seeks to facilitate the collaboration of multiple co-located users simultaneously interacting with a shared display. It incorporates research in multi-user interfaces, application sharing, access control, shared displays and peer-to-peer networking. One of the stated desires of this community is to encourage systems developers to add true multi-cursor support at the OS level. Our work on a multi-cursor window manager is a first step in this direction. It can support simultaneous interaction in different windows of existing single-user applications. It can also support interleaved interaction within the same window. Another desire stated in the SDG papers is for simultaneous navigation by users. Multi-cursor desktops provide support for this by allowing users to traverse a shared data set in independent side-by-side applications.

3. SYSTEM DESIGN

3.1 Control Room Environment

Collaboration among engineers and scientists within a control room is critical to making sound decisions in a limited time period. Control rooms have historically had many types of information displays, including analog and digital readouts, individual computer displays and group visible displays; however, the group displays have typically been

pre-programmed to show certain data and have not lent themselves to dynamic use by collaborators.

Our user group consists of 20 to 30 fusion scientists in a reactor control room. They analyze data from previous experimental reactor runs and adjust parameters for the next experimental run. There is a 20-minute time period in which to do analysis and make decisions. Researchers will typically produce graphs and views of the data on their personal workstations. When they find an interesting result they want to share it with the group so it can be incorporated into the decision making process. Previously, this was accomplished by scientists walking to each other's computer displays to see the results. Our goal was to intermingle the use of personal and shared displays to make their collaboration more efficient.

3.2 Collaborative Components

We decided to focus our efforts in three areas: creating a large shared display, making it easy for users to move their application windows to the shared display, and providing for simultaneous control on the shared display. The applications used in the control room are primarily X11 based; however, the workstations are typically Windows or Mac. So, we could initially narrow our display environment to X11 with user tools targeted toward Windows and Mac. We sought to re-use or incorporate other tools as much as possible.

3.2.1 Multi-Cursor Window Manager

We wanted to support concurrent interaction on the shared display since, in typical use, scientists would be modifying data views in side-by-side application windows. We searched for other projects that would allow multiple users simultaneous control of a desktop. All projects we found allow shared control of a desktop via a single cursor that is shared among multiple collaborators like VNC[14]. We found no X extensions or other references on how to support multiple cursors in X11. Although we did get some hints from the XFree86 developer's group into how we might time-slice the X server's single user paradigm to create a multi-user interface, we still needed to start from scratch in the creation of a simultaneous multi-user desktop.

3.2.2 Shared Display

We decided on a two-projector tiled display at the front of the control room. The display is 6x16 feet and 7 feet off the ground. It's large enough to be easily visible from anywhere in the room. We automatically align the projectors using DeskAlign[19] and drive them from a single PC with a dual-headed graphics card.

3.2.3 Shared Windowing Configuration

After searching and evaluating many application-sharing projects and tools[1] we decided to use Xmove [17]. Xmove has an X11 pseudo-server which runs on a client computer. The `$DISPLAY` environment variable can be set such that when applications are started, the application window will then be sent to the pseudo-server and looped back to the local display. At any subsequent time the application window can be redirected to a different X server by utilizing the xmovectrl command line tool. We developed a Java GUI application to help simplify the use of Xmove (figure 1). It starts the pseudo-server, sets the `$DISPLAY` variable, and lists the applications displayed locally or on the shared display. Applications can be moved to or from the shared display by selecting them from the list and clicking the appropriate arrow button.

[1]Most application-sharing projects in the literature are either proprietary or inactive without downloadable code.

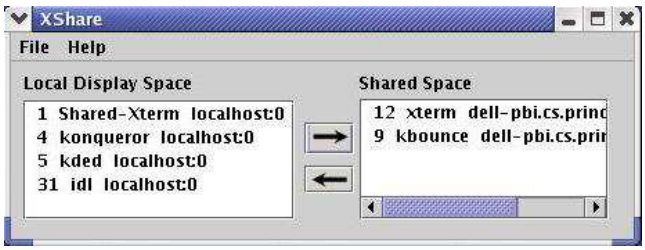

Figure 1: Application-Sharing GUI

3.3 Multi-Cursor Desktop Design Choices

Starting from the paradigm of using a multi-cursor desktop to support concurrent user interaction, and extending the ideas upon which a single-user desktop is based we developed the following assumptions.

Multi-Cursor Desktop Assumptions

- Multi-cursors operate concurrently and can control any application including menus and items on the desktop.
- A cursor can only grab the focus of one window at a time.
- If multiple cursors interact with the same single-user application, the event stream is the subsequent interleaving of cursor events.
- If multiple people control the same cursor, their interactions will be combined and interleaved.
- Cursors must be easily distinguishable from one another.
- Cursor-application focus associations must be easily distinguishable.
- Legacy X11 application should run in the environment.

Based on these assumptions, we considered several layers into which desktop multi-cursor support could be added. Note first that adding multi-cursor support to individual applications is not considered due to the requirements of interacting with a desktop environment and supporting legacy X applications. Essentially, there are three levels at which multi-cursor support can be added: the display server layer, the window manager layer, or a remote desktop layer.

Modifying the X11 display server is the most elegant solution. This would give not only a multi-user desktop with support for legacy applications, but could also provide a system library for future multi-user aware applications to

utilize when registering for user input channels. A less elegant but much easier approach is to implement a multi-cursor window manager. This has the advantage of enabling a multi-user desktop supporting legacy applications while requiring only a small development effort with no system-level modifications. Both the display server and window manager implementation have the advantage of providing good performance. The third choice is to add multi-cursor support to a remote desktop server such as VNC[14]. This provides utility similar to that at the window manager layer, with the added potential for cross-platform support, but with potentially decreased performance and a larger software development effort.

Weighing these three options, and keeping in mind that X11 was our target platform, we decided to prototype our system in the simpler window manager layer in order to quickly gain real-world use experience. We detail our window manager implementation in the next section.

4. MULTI-CURSOR WINDOW MANAGER IMPLEMENTATION

A multi-cursor window manager (MCWM) can provide a convenient way to prototype a multi-cursor desktop. It allows us to gain some experience with simultaneous application-sharing and editing on a control-room shared display. There are several constraints involved in implementing a MCWM. These constraints derive from the fact that the Xserver internally supports only one cursor. This has ramifications which include: only one cursor is drawn on the display, only one window has focus, there is only one event queue, and there are no data fields for differentiating cursors. These were all challenges we needed to solve in order to successfully implement our MCWM.

Rather than start completely from scratch, we started with the base code of an existing, but minimal, single cursor window manager called wm2.

4.1 Creating and Differentiating Events from Multiple Cursors

We first needed to create a way to distinguish between events generated from distinct input sources. We searched through the XEvent structure for any unused fields or bits common to all event types. We found that the *state* field only uses bits 1-13. The *state* field is declared as *unsigned int*, but we've observed the Xserver to send only 16 bits of data. This leaves bits 14-16 available to specify a cursor number, allowing eight distinct cursors. We differentiate between cursor 0 and cursors 1-7. Cursor 0 we call the system cursor, it has zeros in bits 14-16 and therefore is the Xserver's normal input channel. Cursors 1-7 we call multi-cursors. They must be generated through some other means.

To generate multiple input sources we modified the x2x program. X2x is a client application that captures keyboard and mouse input and sends the corresponding Xevents to a remote Xserver. This essentially allows a user to attach their keyboard/mouse to a remote display. We modified x2x to pack the cursor number into the top 3 bits of the *state* field before sending events. We also added a command line option to specify the cursor number.

4.2 Displaying Multiple Cursors

Normally, cursor display is handled through the Xserver, but there is only support to display one cursor. So to physically display multiple cursors we constructed a separate technique. For each cursor (1-7) that registers with the window manager, we open a new X window sized 16x16 pixels. We then use the XShape extension to make this window look like a cursor. We color the window one of seven pre-selected colors to distinguish cursors from one another. Cursor motion events are then very easy to handle within the window manager: we simply change the location of the corresponding cursor-shaped window. Cursor 0 is drawn by the Xserver as usual.

4.3 Creating Multiple Focused Windows

We keep a data structure of information for each cursor. Among the information kept is the (x,y) location, focus window, and cursor color. When the event handler receives a multi-cursor button click event, it uses XWarpPointer to move the system cursor to that location and then XQueryPointer to find what window occupies that position. That window then becomes the focus of the corresponding multi-cursor. Previous associations of that window with other cursors are relinquished. We then change the border color of this window to match the cursor color. This makes it easy for the user to identify the focus of their cursor.

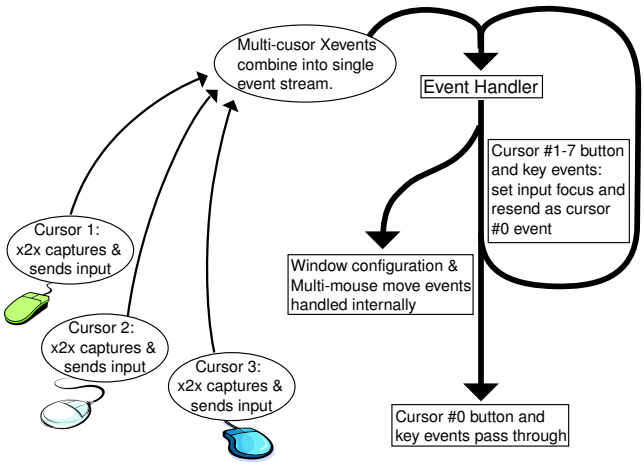

Figure 2: Muli-Cursor Event Loop

4.4 Simulating Multiple Event Queues

The window manager event handler receives all window configuration events, in addition, we register for button, key, pointer and visibility events. We modified the event handling routine of wm2 to create two distinct code paths; one for the system cursor (cursor 0) and one for the multi-cursors (1-7) (figure 2). The system cursor code path is mostly unchanged from the original wm2 source code. The multi-cursor path sets the input focus to the window associated with the cursor and then resends the event through the system cursor path. Normally a window manager will change the shading or color of a window's frame as the focus is gained or lost. We suppress this and only change the color of the window frame when associations between cursors are established or relinquished. When the events are related to window configurations such as moving, resizing or focus,

the window manager consumes the event without resending it through cursor 0.

4.5 Implementation Evaluations

4.5.1 Strengths

One of the main strengths of this implementation is its simplicity. The implementation works within the unmodified X11 framework. All aspects, from creating and sending cursor events to receiving and handling the events, are done using X11 function calls and data structures. This allows us to do minimal changes to existing window managers without making changes to the underlying X11 system.

4.5.2 Limitations

Using the existing X11 system also has limitations. We are essentially time-sharing the system's single cursor between multiple clients, and this creates scalability issues and potential race conditions. In particular, we already know that the number of cursors is limited by the number of unused bits in the XEvent structure. Also, performance may decrease faster than expected from a true multi-cursor display server as concurrent cursors are added.

Because there is only one focus window at a time, race conditions must be avoided by setting window focus and sending window events atomically. Setting the focus and re-casting the event to the system cursor takes two passes through the event loop in our implementation. If events from other cursors get interleaved during this transaction period, they must be re-queued until the transaction is complete.

5. SYSTEM EXPERIENCES

Our main goal in this project has been to improve collaboration within control rooms by allowing users to simultaneously interact with and navigate data sets on a shared display. Figure 3 shows a prototype of our system running in the PPPL control room. In this section we would like to evaluate some experiences from using the system.

Figure 3: A shared display running multi-cursor X in fusion Control Room

5.1 Using concurrent cursors

We have had up to three users simultaneously interacting with our multi-cursor desktop (see figure 4). The ability to distinguish cursors and focused windows by color is quite effective and we have not encountered problems with users identifying their cursor or focused window. Simultaneous typing is seamless. Users concurrently working in xterms or other text-based applications will not notice any difference from a single-user desktop. Controlling cursors, moving and resizing windows is similarly transparent.

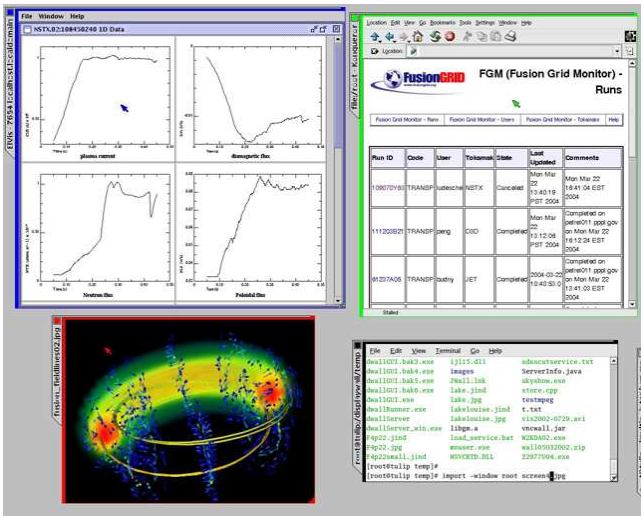

Figure 4: Example of 3 users interacting with shared display: red user viewing toroidal data, blue user making 2D plots, green user looking at fusion web page

5.2 Interference from other users

One observation regarding the multi-cursor desktop is that screen real estate becomes important. When several people are working simultaneously on a personal-size display we've found that they interfere with each other frequently. There is no way to share the z-order, so at any given time some window is on top, obscuring the view of other windows. Multi-cursor interaction on a small desktop is primarily effective when users are working together on the same application and so are not interfering with each other. To accommodate more users it is necessary to increase the screen size. For a dual projector display, two users can easily work without interference and three users is acceptable.

5.3 Setup and use

The control room has about 20 workstations running mostly Mac OS and Microsoft Windows, but the data analysis applications are X11 based and run from an application server. To accommodate our window sharing, we run the Xmove pseudo-server on the application server and run our Java GUI client on the workstations. The Java client can run on Mac, Windows or Linux and can connect to the application server to direct the window to the local or shared display.

To accommodate transferring the cursor to the shared display we run x2x with the `-east` setting on the user workstations. With this configuration, when a user drags the mouse

off the right hand edge of the workstation screen, the corresponding color-coded cursor appears on the left edge of the shared display. Dragging the cursor off the left edge of the shared display brings it back. This makes it very easy to switch between controlling the local desktop or the shared display.

One limitation to implementing the multi-cursor support in the window manager is that it defines other aspects of the shared display user interface. Window managers effect how the desktop will behave, including how windows look and are controlled. It would be nice to be able to easily change the interface to accommodate what users are most accustomed to.

5.4 Multi-cursor benefits

We've found three classes of interaction that benefit from a multi-cursor window manager. The first is multiple users working independently side-by-side. For instance, they may be looking at different views of the same data set by using separate application instances. The second type is when several users are working on the same task. This could be interacting with the same application or pointing out details on the same screen. In this case, the multiple cursors are essentially used one at a time, but the fact that everyone has one makes it more efficient to transition control. The system essentially remembers where each cursor is and so there is less dragging the mouse back and forth as users switch turns. The third case is when a single user interacts using two mice. This is particularly useful in drawing and illustration type applications such as the GNU Image Manipulation Program (gimp). A user normally has to move a cursor back and forth between different windows when alternating between tool selection and drawing. If the use has two mice, the left-hand mouse can be stationed by the tool area while the right-hand mouse is in the drawing area. This makes selecting different brushes and palettes considerably faster.

6. CONCLUSION AND FUTURE WORK

A multi-cursor window manager provides for efficient control room collaboration by alleviating the user input bottleneck that occurs from traditional single cursor systems. The multi-cursor environment scales to allow many users simultaneous control of applications on a shared display. This is important when users must traverse and compare data results that are time critical in nature.

X windows, which is designed for a single cursor and user, can be extended to support multiple concurrent cursors by modifications to a window manager. The window manager event loop must be modified to recognize multi-cursor events and render multiple cursor arrows on the screen.

We have designed, prototyped and deployed a multi-user collaborative system for control room collaboration. The multi-cursor window manager is used in conjunction with a large shared display and an application-sharing mechanism. This system has had initial deployment at the Princeton Plasma Physics Lab, and feedback from the control room users has been positive. They have come to rely on the shared display for their daily work activity.

In future work, we plan to move support for simultaneous cursors into the Xserver layer. In addition to supporting concurrent control of legacy applications, it will provide for more configurability as well as system libraries that future multi-user applications can utilize to access concurrent input channels. We will also continue to improve the application sharing tools, supporting heterogeneous system platforms.

7. REFERENCES

[1] H. Abdel-Wahab and M. Feit. Xtv: A framework for sharing x window clients in remote synchronous collaboration. *IEEE Tricomm*, 1991.

[2] B. Bederson, J. Stewart, and A. Druin. Single display groupware. *Sig CHI*, 1999.

[3] R. Bentley, T. Rodden, P. Sawyer, and I. Sommerville. An architecture for tailoring cooperative multi-user displays. *CSCW*, 1992.

[4] E. A. Bier, S. Freeman, and K. Pier. Mmm: The multi-device multi-user multi-editor. *SIGCHI*, 1992.

[5] S. Elrod, R. Bruce, R. Gold, D. Goldberg, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, J. Tang, and B. Welch. Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. *SIGCHI*, 1992.

[6] D. C. Engelbart. A research center for augmenting human intellect. *Proceedings of FJCC*, 33(1):395–410, 1968.

[7] D. Garfinkel, P. Gust, M. Lemon, and S. Lowder. The sharedx multi-user interface user's guide, version 2.0. *HP Research report, no. STL-TM-8907*, 1989.

[8] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing Magazine*, 1(2), Apr/Jun 2002.

[9] P. Kabbash, W. Buxton, and A. Sellen. Two-handed input in a compound task. *SIGCHI*, 1994.

[10] Microsoft. Netmeeting. *http://www.microsoft.com/windows/netmeeting/*.

[11] B. Myers and H. Stiel. An implementation architecture to support single-display groupware. *CMU Technical Report, CMU-CS-99-139*, 1999.

[12] E. R. Pedersen, K. McCall, T. P. Moran, and F. G. Halasz. Tivoli: an electronic whiteboard for informal workgroup meetings. *SIGCHI*, 1993.

[13] J. Rekimoto. A multiple device approach for supporting whiteboard-based interactions. *SIGCHI*, 1998.

[14] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1), JAN/FEB 1998.

[15] R. Scheiffler and J. Gettys. The x window system. *ACM Transaction on Graphics*, 5(2):79–109, 1986.

[16] SciDAC. Fusiongrid. *http://www.fusiongrid.org/*.

[17] E. Solomita, J. Kempf, and D. Duchamp. Xmove: A pseudoserver for x window movement. *The X Resource*, 1(11):143–170, JULY 1994.

[18] H. Tani, M. Horita, K. Yamaashi, K. Tanikoshi, and M. Futakawa. Courtyard: Integrating shared overview on a large screen and per-user detail on individual screens. *SIGCHI*, 1994.

[19] G. Wallace, H. Chen, and K. Li. Deskalign: Automatically aligning a windows desktop. *PROCAMS*, 2003.