The "Mighty Mouse" Multi-Screen Collaboration Tool

Kellogg S. Booth, Brian D. Fisher, Chi Jui Raymond Lin, and Ritchie Argue
Department of Computer Science
University of British Columbia
Vancouver, BC, V6T 1Z4, Canada
E-mail: {ksbooth, fisher, rlin, ritchie}@cs.ubc.ca

ABSTRACT
Many computers provide seamless support for multiple display screens, but there are few cross-platform tools for collaborative use of multiple computers in a shared display environment. We describe a novel approach, tailored specifically for face-to-face collaboration, in which multiple heterogeneous computers (usually laptops) are viewed simultaneously (usually via projectors) by people working together using a variety of applications running on various platforms. Mighty Mouse is built on top of VNC, a virtual networking protocol that allows local display and interaction with remote computers. Mighty Mouse is a single display groupware tool. All participants view common displays, so Mighty Mouse uses only the remote input capability of VNC. But Mighty Mouse enhances VNC with various features to support flexible movement across the various platforms, "floor control" to facilitate smooth collaboration, and enhanced customization features to accommodate different user, platform, and application preferences in a relatively seamless manner.

KEYWORDS: collaboration, cut-and-paste, keyboard mappings, low-fidelity prototyping, single display groupware, virtual network computing.

INTRODUCTION
Many personal computers have fully integrated support for multiple display screens, both as separate screens and as tiled screens where the mouse moves seamlessly from screen-to-screen and windows may be split across screens as if they formed a single large display surface. Platform specific tools, such as Timbuktu for the Macintosh and more recently Windows [5], allow users on one computer (the *controller* or *home*) to interact with applications running on another (the *controllee* or *target*). This is accomplished by mirroring the controllee's display on the screen of the controller and forwarding keyboard and mouse events from the controller to the application running on the controllee. The X Windows system [4] has a client-server model in which the display and its associated input devices (keyboard and mouse) are separated from the host platform on which an application is running (a special case being when the two are the same). More recently, cross-platform protocols, most notably VNC, Virtual Network Computing developed at AT&T [6], provide a capability similar to Timbuktu that allows heterogeneous controller/controllee pairings and, for at least the **x2vnc** implementation of the VNC protocol [1], the ability to control multiple heterogeneous controllees in sequence.

LEAVE BLANK THE LAST 2.5cm
OF THE LEFT COLUMN
ON THE FIRST PAGE
FOR US TO PUT IN
THE COPYRIGHT NOTICE!

Our work builds on these previous ideas and uses VNC's network protocol as well as the VNC server on the controllee to handle all of the interactions with the host operating system and applications running under it. What we change is the client, especially the user interface on the controller, and the ability to control and adapt for multiple controllees.

A primary reason for changing the user interface is our focus on single display groupware [7], which has traditionally considered multiple users sharing a single screen displayed by a single computer, but which we enlarge to multiple users sharing multiple screens each displayed by a different computer. Timbuktu, X Windows, and VNC largely address the problem of a user not being co-located with the machine on which a computation is performed. Single display groupware addresses the problem of multiple co-located users sharing a single computer and display. We address the problem of multiple co-located users collaborating on tasks that are supported by applications running on multiple computers each with its own display, but with all of the displays visually shared by all of the users.

AN ILLUSTRATIVE EXAMPLE AND USAGE SCENARIO
To illustrate many of the basic features we want to provide, we briefly consider the example of a meeting held in a small boardroom. Each participant has brought a laptop computer to the meeting, many of them with documents that will be discussed and modified over the course of the meeting. One or more LCD projectors are on the table connected to some (probably not all) of the laptops. Over the course of the meeting, typical activities might include discussion and revision of a spreadsheet containing a proposed budget, the drafting of a document describing the decisions made at the meeting, and consultation of other documents on one or more of the laptops or on the web.

Without special tools, a typical experience is that users will move around the room from laptop to laptop as they take turns modifying or viewing the documents. Sometimes they

will move documents from one machine to another, and at times they may re-cable the projectors so that everyone in the room can see the particular document under discussion. Especially in this last case one often sees a person physically move to the laptop on which the document is hosted to assume temporary ownership of the document and the application that manages it.

With the software we have developed, and with modest assumptions on additional hardware support, scenarios for the meeting could be quite different. First, we assume that all of the laptops can be connected via a network, either through cables or wireless. Increasingly this is the case, often with the network part of the building infrastructure, but if not, a small networking hub or wireless base station is affordable, portable, and compatible with most recent laptops. If we further assume that there are video switches connecting the many laptops to the smaller set of projectors, it is possible to dynamically choose which subset of the laptop displays will be visible to everyone at the meeting. This can be done manually, or under computer control if more expensive video switches are used. Again, these capabilities are becoming available in some conference rooms and we can expect that with the growing popularity of LCD screens and digital video standards, they will become increasingly present as standard features in many settings.

What then remains is to provide software support that will permit any user to interact with any application running on any of the laptops over the course of the meeting, and to do this in a way that naturally supports both the individual users and the group of users. This is what Mighty Mouse does. At the end of this TechNote we will describe some other ways in which it can be used, but for now we concentrate on the collaborative meeting scenario, which is an example of single display groupware but with multiple heterogeneous platforms.

The term "single display groupware" originally meant literally a single screen, but the fundamental issues are those that arise through the visual sharing of the screen, and the collaborative and often simultaneous interaction of multiple users with the application(s) whose information is being displayed. We generalize this to multiple displays. We can think of this as simply being a much larger virtual display made up of the individual displays. But we restrict ourselves to the case where users interact sequentially with a single application and leave for another discussion the problem of simultaneous interaction with an application. We discuss this last point a bit in the final section of this TechNote.

IMPLEMENTATION

Our current implementation of Mighty Mouse uses the VNC protocol. The VNC server (on the controllee) and a new VNC client (on the controller) are used. The software has been modified to support multiple pairs of controllers and controllees, and floor control support has been added. We describe each of these features and give brief comments on their implementation where appropriate. As with similar systems, the basic operation on the controller is that mouse and keyboard events are captured by Mighty Mouse (adapting the VNC client for each platform) and either re-directed to the controllee or left for the host platform's O/S to handle normally (the "idle" state).

(a) Spatial metaphor for implicit switching

At any instant, a controller is either idle or is connected to a single controllee. There is a configurable spatial arrangement similar to what is provided on the Macintosh and other platforms for multiple display screens. Mouse movement on the controller causes a corresponding movement of the cursor on the controllee screen. If the cursor moves off the left boundary of the controllee screen the connection to that controllee is broken and a new one is established with the screen that is logically to the left. Similarly, movement off the right, top, or bottom switches connections to those screens.

The cross-platform version supports only left-right implicit switching, but an earlier prototype demonstrated the effectiveness of fully 2D implicit switching. We have not used this in the current version for two reasons. The first is that it is generally a good idea to block cursor movement beyond whichever edge holds the menu bar or similar GUI features. This is because these features are generally not targeted directly, but rely on "clipping" at the edge of the screen to "grab" the mouse cursor for faster access to the menu bar. If a full 2D spatial layout is allowed this cannot be supported. The second reason is that the control panel layout (see (b) below) is currently a linear strip and we did not want to take up a lot of screen real estate on the controller with a more general 2D icon. We do support a circular layout, so the leftmost display is logically to the right of the rightmost.

One existing implementation of the VNC protocol, **x2vnc**, also supports multiple connections and a spatial metaphor for implicit switching for X Windows clients [1].

(b) Dynamic keyboard and mouse mappings

Different platforms have different keyboards, especially for the special function and modifier keys. The VNC protocol already provides for automatic key and modifier mappings between the controller and the controllee. We are extending these to include mouse click mappings to accommodate differences between one-, two-, and three- button mice. We are also making the mappings dynamic as a function of the controller and the controllee platform types, the user, and the application. Our current implementation requires manual selection of application-specific mappings (see (c) below).

(c) Control panel for explicit switching

It is not always desirable to have to move the cursor to a

new screen when the focus of activity changes. Especially when there are intermediate screens, this leads to tedious mouse movement and possibly disruptive behavior on the intermediate screens as they are captured and then released by the controller as it moves to the final target screen. For this reason we have added a control panel that resides on the screen of the controller (Figure 1). This permits explicit switching between controllees and also provides a convenient mechanism for managing key mappings.

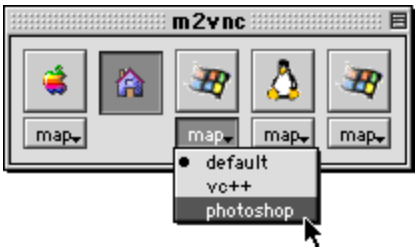

Figure 1. The control panel, showing the drop-down menu for setting keyboard mappings, with the home button second from the left.

The control panel has an iconic button for each target machine and a special button for the home (controller) machine, which is located within the control panel according to its logical spatial relationship with the controllees. The icons indicate the type of the operating system: Macintosh, Windows, Linux/Unix, or home. Home is not identified by its type, but instead is differentiated by it being the controller. Background color provides state information (which of the buttons is the current controllee) and a drop-down menu at the bottom of each button allows various key mappings to be selected.

When the controller is idle, the control panel functions like any other widget on the desktop. Clicking on one of the buttons initiates a connection to that controllee using the currently selected key mapping. If the drop-down menu is activated, the current key mapping can be examined (see Figure 1) and the selection changed; this does not activate a connection. When Mighty Mouse is first launched, a user might configure the key mappings for each connection. We have not provided a customization facility, but an obvious extension would be to specify default key mappings in a user-defined configuration file.

When the controller is not idle, mouse and keyboard events are forwarded to the controllee. The control panel on the controller is not accessible. A mouse-down event with a special meta-key combination returns to the idle state with the cursor centered on the icon of the just-released controllee. This is configurable, but in our prototype the meta-key combination is CTRL-SHIFT-ALT. At that point a user can click on a different icon to explicitly switch to a new controllee, adjust the key mappings and click again on the same controllee, or resume normal activity on the host controller.

(d) Consistent apparent mouse speed

Differences in mouse gain across platforms and varying screen resolutions can cause apparent changes in velocity during implicit switching. We have experimented with various techniques for adjusting the mouse gain. These scale mouse movements so that visually continuous velocities are perceived by the user. This requires that some state information regarding fractional cursor position be maintained, similar to what is done in Bresenham's algorithm for digital line drawing.

(e) Smart cut-and-paste

The **x2vnc** implementation supports a basic cross-platform cut-and-paste operation, but only for text. This is invoked using the Edit menu on the frame of the window on the controller screen in which the mirror of the controllee is displayed. Mighty Mouse does not mirror the controllee's display because of our assumption that all of the displays are visible, so there is no frame and hence no Edit menu. We could add this to the control panel, but we have not done so. We have, however, prototyped a number of extensions to handle a limited range of more advanced cut-and-paste mechanisms.

When both platforms support a desktop metaphor, standard drag-and-drop operations or cut-and-paste from the desktop and folders are possible. The semantics would be an implicit **ftp** operation from one machine to another. When the target machine (or window) is a command line interface, alternate semantics need to be used. Drag-and-drop could still imply an **ftp** operation to the current directory associated with the command line window. More elaborate mechanisms might include configurable mappings that launch applications that are registered for various file types. We have experimented with a number of these ideas, but have not implemented them as yet.

(f) Floor control for collaborative usage

When only a single user is switching between a number of controllees, Mighty Mouse provides essentially the same functionality that **x2vnc** provides but across all platforms (**x2vnc** is an X Windows client). Users can of course sequentially assume control of various controllees, with all others remaining in the idle state working on just their home machines. But fluid collaboration requires that multiple simultaneous connections be supported.

To achieve this, Mighty Mouse includes a floor control mechanism that oversees the management of connections. We assume that there is always a set of mutually exclusive pairwise connections, with every machine being exactly one of in-the-idle-state, a controller, or a controllee. This assumption precludes multiple users from simultaneous access to an application on a single machine. Our belief is that, while clearly desirable, this is still not easy to provide because many popular applications are not fully collaboration-aware. We have thus concentrated on the low-hanging fruit and for now make the assumption of only

one-to-one connections.

Current floor control is very simple. Any machine in the idle state is available as a controllee to any controller that selects it, either implicitly or explicitly. A machine that is currently a controller or a controllee cannot be selected. This means that implicit selection through the spatial metaphor is ignored when a controller moves logically across the screen of a controller or a controllee. We expect to replace this with a more elaborate protocol based on our earlier analysis of the give-and-take paradigm for using multiple mice in single display groupware [2,3].

A SINGLE-USER SCENARIO

The discussion so far has focused on Mighty Mouse as a collaborative tool. The original impetus was to support just a single user, especially the situation in which one person is using two or more heterogeneous computers. An extreme scenario of this might be an office equipped with Macintosh, Windows, and Linux/Unix platforms each being used for specific tasks. Shared file systems mounted on all of the platforms would make data interchange easy, but a user would still have to switch back and forth from one keyboard and mouse set to another as activities moved from machine to machine. Just as in the collaborative case, our assumption is that the multiple monitors will all be visible to the user, so the main problem is seamless switching from machine to machine in a manner that mimics a multi-display system.

Before implementing Mighty Mouse we built a number of low-fidelity prototypes, mostly using video techniques to simulate various usage scenarios. The most elaborate of these used three computers (one Macintosh, one Windows, and one Unix) each driving LCD projectors displayed side-by-side on large wall-mounted screens. We simulated a variety of features illustrative of the mechanisms for switching between platforms/screens, "smart" cut-and-paste, and mode control. A single video camera captured the activity on the three large screens, which appeared in the video as if they were three desktop monitors placed side-by-side. The computers were not networked together. Instead, we had a different person operating each of them working in concert to provide the illusion of functional software. When one person moved the cursor to the left edge of her screen, the person to her left began moving his cursor from the right edge of his screen as if it were a continuation. This low-fidelity prototyping allowed us to test alternative designs with different scenarios before we began implementation.

From this iterative approach emerged a design for the prototype. We implemented two single-user versions of the design. One operated across both Windows and Linux/Unix platforms; the other was specific to the Macintosh. After additional testing and refinement of the design, we moved to the implementation of the full cross-platform version. At the same time we enhanced the design to include additional features for collaboration (multiple simultaneous controllers and controllees, as well as floor control mechanisms) and we adopted the VNC protocol as the basis for our implementation.

DISCUSSION, CONCLUSIONS, AND FUTURE WORK

There are a number of ways in which the current implementation can be extended. Certainly more elaborate floor control mechanisms could be added. These need to be evaluated by user testing in realistic settings. Adding controllers for PDAs or other hand-held devices would substantially increase the usefulness of Mighty Mouse. In the collaborative board room scenario, not everyone is likely to have a laptop, but they may have PDAs or other pen-based devices. More automatic keyboard mappings might be possible, especially if the controllees can provide information about the application that is running (not always easy to do if background tasks are running).

The video clip that accompanies this TechNote illustrates the Mighty Mouse control panel and the spatial metaphor developed in the low-fidelity prototypes.

ACKNOWLEDGEMENTS

The Natural Sciences and Engineering Research Council of Canada provided funding for this work.

REFERENCES

1. Hubinette, F. (2002). **x2vnc 1.31** (home page). www.hubbe.net/~hubbe/x2vnc.html

2. Inkpen, K., Booth, K.S., Gribble, S.D., and Klawe, M.M. (1995). Give and take: Children collaborating on one computer. *Proc. of ACM CHI '95* (May 7-11), pp. 258-259.

3. Inkpen, K.M., Booth, K.S., Klawe, M.M., and McGrenere, J. (1996). Turn-taking protocols for mouse-driven collaborative environments. *Graphics Interface '97* (May), pp. 138-145.

4. Jones, O. (1989). *Introduction to the X window system.* Prentice-Hall, Inc. Englewood Cliffs, NJ 07632, pp. 234-241.

5. Netopia (2002). **Timbuktu** (home page). www.netopia.com/en-us/software/products/tb2/

6. Richardson, T., Stafford-Fraser, Q., Wood, K.R., and Hopper, A. (1998). Virtual network computing. *IEEE Internet Computing*, Vol.2 No.1 (Jan/Feb), pp. 33-38.

7. Stewart, J., Bederson, B.B, and Druin, A. (1999). Single display groupware: A model for co-present collaboration. *Proc. of CHI '99*, pp. 286-293