Virtually Shared Displays and User Input Devices

Grant Wallace and Kai Li
Department of Computer Science,
Princeton University, Princeton, NJ 08540

Abstract

This paper proposes making displays and input devices as first-class citizens in a networked system environment for collaborative applications. The paper describes a virtually shared model that enables users to use remote displays as extensions of their local displays and to allow multiple users to use multiple cursors and keyboards to input and control shared applications and their windows simultaneously. We have implemented a prototype system and deployed it to three DOE Fusion control rooms. The implementation performs well on today's hardware and our user feedbacks show that such a paradigm can substantially improve information sharing.

1. Introduction

We are moving into a new era of computing in which computers and networks are becoming ubiquitous. One of the associated challenges is to achieve seamless communication and visualization, i.e. to enable users with heterogeneous computing devices to communicate with each other and to visualize each other's information effortlessly and seamlessly.

We advocate that displays and user-input devices should be first-class citizens in a networked environment. They should be connected such that display information can be moved from one display to another seamlessly for collaboration purposes, functionally similar to connecting to external physical displays. Users should be able to view shared displays as extensions of their own displays, ideally independent of platforms, operating systems and applications. A scientist could walk into her colleague's office to show results by simply dragging the plots from her laptop onto a colleague's desktop display. After their discussion, the display information on the colleague's display may evaporate. In a collaborative group setting, multiple users can display, input and control information on a shared display simultaneously (Figure 1).

Current operating systems and window systems, however, have limitations to supporting such scenarios. They were designed for single-user use cases in a less network-centric era. A fundamental assumption has been that display devices and user-input devices are not first-class citizens in a network system environment. As Gettys pointed out in his paper on SNAP computing [Gett05], *"Today's computing mantra is 'One keyboard, one mouse, one display, one computer, one user, one role, one administration'; in short, one of everything. However, if several people try to use the same computer today, or cross administrative boundaries, or change roles from work to home life, chaos generally ensues."*

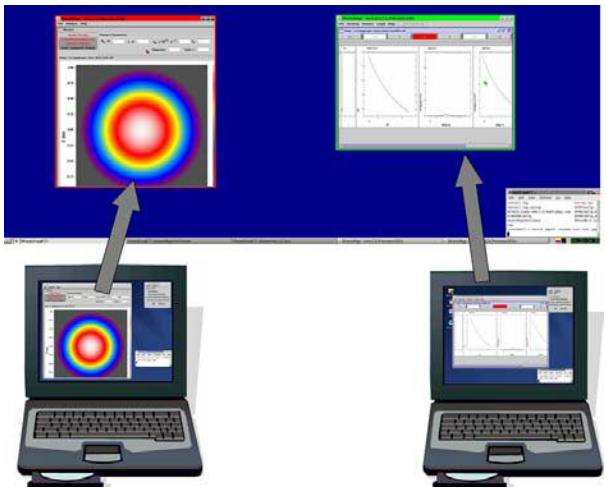


Figure 1: Application windows that can be readily moved or duplicated to other screens, and simultaneous multi-user input form the core of a collaborative display environment.

This paper proposes virtually shared display and user-input abstractions to create network enabled displays and input devices. The abstractions can be used to conveniently design systems to implement seamless information sharing for multi-user collaborations.

We have designed and implemented a shared display system by using the proposed abstractions and by leveraging VNC and x2x. We have released the implementation to the public domain and deployed our shared display system to three DOE fusion control rooms for their production use. The users' feedback shows that the proposed abstractions are indeed useful and shared displays can substantially improve information sharing in control room environments. Our performance evaluation shows that the implementation can provide interactive frame rates and impose reasonable network and computing resource requirements.

2. Virtually Shared Displays: Model and Abstractions

We propose a *virtually shared model* for displays and user-input in a networked environment. The goal is to conveniently connect components across a network so that peer participants can share application windows and provide user input across physical devices. There are several requirements to support this goal:

- Share display information at the granularity of application windows.
- Allow multiple users to input and control applications simultaneously.
- Support sharing in a many-to-many fashion.
- Be operating system independent and application transparent.

The main abstractions for this model are networked user-input devices, networked application windows, and networked displays. These components form communication links such that input-devices connect to applications, and applications connect to displays.

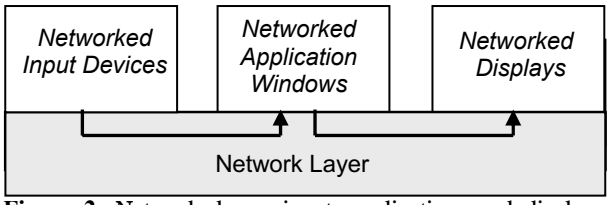


Figure 2: Networked user-input, applications and displays are the basis of a shared display environment.

A ***networked window*** is an abstraction associated with an application that sends graphical outputs and receives user inputs across the network. Networked windows must support several operations related to these tasks.

For graphical output, a networked window must support the operations of *replication*, *migration* and *orphaning*. Replication allows network windows to be shown on multiple networked displays simultaneously. Migration allows a networked window to be moved from one networked display to another. Orphaning allows an application to continue running even when no network display resources are available.

A networked window must also be able to receive input from networked input-devices. If the networked application is multi-user aware it can allow for simultaneous input; otherwise, it can serialize multiple input streams that it receives.

These networked window operations permit a variety of use cases such as collaborative sharing (replication), mobile computing (migration), and display composition to view high-resolution spanned windows.

A ***networked display*** is an abstraction that implements a display surface for computer systems in a network to display application windows. The main difference from the traditional display abstraction is that it allows for multiple focused windows per display to support multiple simultaneous networked input devices. The main operations of a networked display are *attach/detach window*, *attach/detach cursor*. Attach window is the operation that allows a networked application window to begin showing content on the networked display. Attach cursor is the operation that allows a networked user-input device to provide input to the networked display. Note that only networked windows on a display can be seen or controlled remotely by others. In other words, the networked or non-networked property of a group of windows serves as the protection domain for information sharing in this paradigm.

A ***networked input device*** is an abstraction that virtualizes a physical user-input device for composing systems in a network. The goal of the abstraction is to treat input devices also as first-class citizens in a network to enable building flexible multi-user systems.

Traditionally, user input is forwarded to the focused application by the display server or window manager. In our model, the networked input devices can instead make direct connections to their focused networked applications. In this case, the networked display still tracks cursor-to-window focus relationships but, instead of forwarding input, it provides the URL address of the networked application when focus relationships are established. This can provide for a direct communication model with better security and performance.

The main operations of a networked input device are *attach/detach* to or from a networked application. This abstraction allows dynamic binding with networked window abstractions. Thus, they can be used to compose an implementation that allows for multiple simultaneous inputs.

3. Previous Work

It is desirable to leverage previous work in this area in order to establish a collaborative display system for fusion scientists without excessive engineering effort. The final system must meet the four requirements listed in section 2: many-to-many, window-granularity sharing, with multi-user input across varied hardware platforms.

The X windows system provides a network-based display protocol that makes it possible to connect to remote display servers. However such connections can only be established at application startup and so dynamic replication or migration of windows is not supported [Gett04]. Several X11-based proxy servers have been created such as SharedX, Xmove and DMX to allow display redirection. However, they also require applications to connect at startup and aren't cross-platform compatible. These limitations do not lend themselves to the type of ad-hoc and dynamic collaboration required.

A number of collaborative systems provide a one-to-many sharing paradigm such as LiveMeeting, NetMeeting, Remote Desktop, Citrix and WebEx. These systems allow one person at a time to share display information and provide input. They, unfortunately, do not support many-to-many or peer-to-peer type sharing where multiple participants simultaneously share windows and provide user input.

Another class of collaborative display systems is based on VNC (Virtual Network Computer). VNC uses a pixel-based approach to replicate all desktop pixels from one computer to another [Rich98]. The advantage of this type of approach is its support for dynamic, cross-platform sharing. One variant of VNC called MetaVNC [Sato04] allows remote desktop windows and local windows to appear side-by-side on the desktop. This is accomplished by making the background of the remote desktop appear transparent. The main drawback of MetaVNC and other VNC implementations is that sharing is done at the granularity of the desktop. When connecting to a remote MetaVNC server, all desktop windows will be shared. We would like to restrict the sharing granularity to the window level for privacy reasons.

THINC is another virtual display system that allows networked desktop sharing. It is implemented at the device driver level and as such can support dynamic sharing and achieve good performance [Bara05]. However, because it operates at the device driver level, it does not track application window boundaries and so doesn't support window granularity sharing. Additionally it currently only has a Linux implementation.

Previous work on supporting multiple simultaneous user input has been done in the Computer-Supported-Cooperative-Work (CSCW) community. It focuses on multi-user computer-human interaction [Cars99]. Related work has also been done in the Single-Display Groupware community to look at multi-user interaction on a shared display [Myer99]. These classes of research have typically looked at human-interface needs and application support for collaboration rather than at systems level requirements. They have not looked at OS- and Window Manager- level support for multiple cursor interaction and in particular do not address multi-user interaction on legacy systems and applications.

A recent effort has been made on a multi-pointer X11 server [Hutt06]. This server allows multiple mice to be plugged into the same computer to create multiple cursors and simultaneous interaction. This work is very relevant and could be extended in the future to support a networked input abstraction.

4. Design and Implementation

In implementing the proposed collaborative display abstractions, our approach is to leverage existing software components where possible. VNC can form a good basis for display sharing because it supports cross-platform sharing; however, it must be modified to add support for window granularity and many-to-one sharing. Additionally, x2x forms a good basis for creating networked user input. It is X11 based, but can easily be supported on Windows systems using Cygwin. It, however, must be modified to support capturing and distinguishing multiple simultaneous user input.

Leveraging these software systems, we have designed and implemented the three networked components as described below and shown in figure 3. We will discuss the implementation of these components in the remainder of this section.

- ***Networked application windows*** – supported with a modified VNC server that allows capturing and sharing pixels at window granularity.
- ***Networked user input*** – supported with a modified version of x2x that can generate distinct input from multiple users.
- ***Networked displays*** – supported with a modified VNC viewer that can display windows from multiple servers and with a modified window manager that supports simultaneous multi-cursor input.

As mentioned earlier, VNC is a pixel-based protocol that allows easy cross-platform sharing. However, VNC only provides functionality to share the entire desktop, not individual windows. This has undesirable consequences for both privacy and utility in collaborative display systems. Privacy issues arise because users typically have some content they want to share, such as a data graph, and other content they want to keep pri-

vate, such as their email. Additionally, utility is limited when an entire desktop is shared because application windows from different collaborators cannot be placed side-by-side for comparison and discussion.

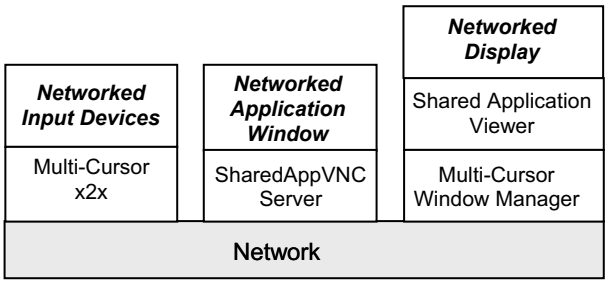


Figure 3: Shared display system architecture using virtually shared display and input device abstractions.

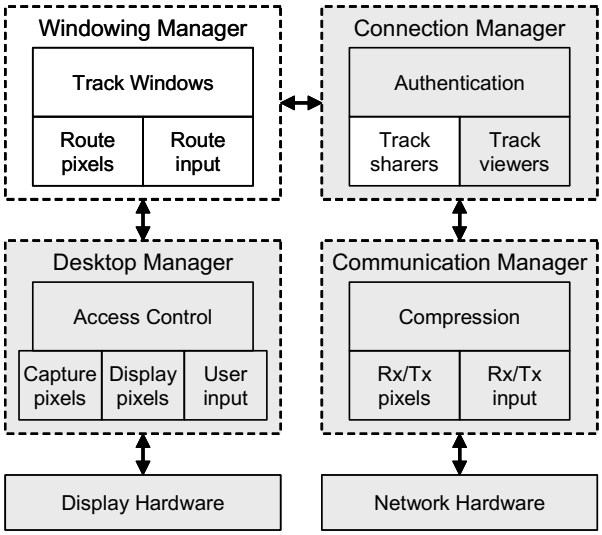


Figure 4: Original VNC provides display and management of pixels at the granularity of the desktop. We extend this by adding a windowing manager to allow window-granularity sharing between multiple sources.

To overcome these limitations we created an extended VNC protocol and implementation as depicted in Figure 4. Original VNC implements the gray components including the capture, transmission and display of pixels at the granularity of the desktop. In addition, it handles one-to-many connection management, which allows one presenter to share to many viewers. Our implementation has added the functionality represented by the white boxes. A windowing manager is added to maintain window-level knowledge, such as the location, size and relationship of all desktop windows. This permits the sharing of certain windows and exclusion of others. In addition, we extend the viewer connection manager to handle multiple simultaneous connections. This allows many-to-one sharing where many users share content to the same viewer, such as a display wall.

The multiple shared windows on the viewer are each placed in their own frame and so look identical to native windows of the viewer display. The shared windows include all parts of the application display including menus. The shared windows can be rearranged independently of their positions on the originating computer, and so can be placed anywhere on a networked display. This allows for easy side-by-side comparison of shared windows. If a shared window becomes occluded on its originating server, it will stop sending display updates for that region and that viewer content will remain static until the occlusion is removed. Our implementation, SharedAppVNC, is released to the public domain (http://shared-app-vnc.sourceforge.net/).

Allowing multiple users to simultaneously interact on the collaborative display was the second priority in our prototype system. Current windowing systems only have data structures supporting a single cursor, so to accomplish simultaneous interaction we created a specialized X11 window manager [Wall04].

The window manager renders multiple cursor arrows on the screen by drawing small 16x16 pixel windows and utilizing the XShape extension to make their shape identical to a normal cursor. Each multi-cursor is rendered with a unique color to easily distinguish it from the others. Cursor events are sent to the window manager using a modified version of x2x which packs the cursor id into 3 unused bits in the XEvent *state* field. This allows the distinction of 8 unique cursors. The cursor id allows the window manager to maintain multi-cursor state information including the current location, focused window and activated control keys.

When the window manager receives a multi-cursor event, it applies that cursor's saved state to the system cursor and then resends the event through the normal event loop (figure 5). This process essentially time-slices the system cursor between the multi-cursors. The time-slicing will be imperceptible to simultaneous users because user input such as typing or dragging is of low bandwidth compared to the system and display update response. Also, we suppress the z-order and window decoration changes that normally happen during a keyboard focus event. This makes keyboard focus changes unnoticeable to users. Window decorations and z-order are only changed when a multi-cursor establishes focus on a window. At that point the window is decorated with the cursor's color to designate the focus relationship where input will be directed.

Operations that involve mouse dragging are the one instance where events cannot be effectively interleaved. For this situation we allow users to obtain an exclusive lock for dragging. By pressing the shift button while dragging, all other multi-cursor events are suppressed. The other cursors will appear as an X until the drag operation is completed.

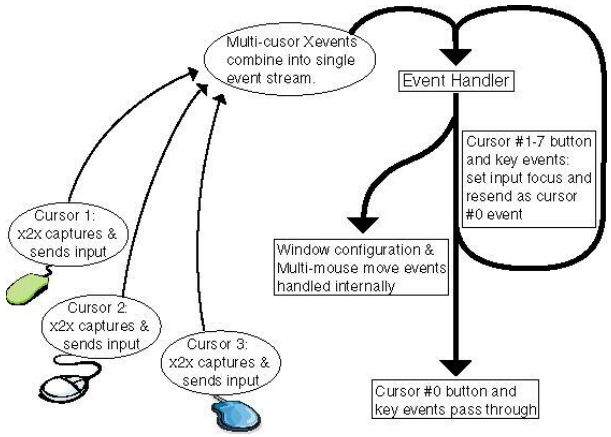


Figure 5: Our prototype system accomplishes simultaneous multi-user input by time-slicing the system cursor provided by a standard operating system.

The multi-cursor window manager is implemented for X11 displays; a release version based on IceWM is available at http://multicursor-wm.sourceforge.net/.

Figure 6: Our shared display system deployed in the Princeton Plasma Physics Lab control room.

Our initial system has been evaluated by several fusion facilities and is currently part of the production environment in the control rooms of General Atomics in San Diego, and the Princeton Plasma Physics Lab. This system has changed how the display walls in those rooms are used. Instead of just showing pre-determined content, the shared displays are now a dynamic forum for user discussion. In a typical discussion, 2-5 users will push application windows onto the wall and compare their results side-by-side. We anticipate that future stages of our system will also integrate remote users into the collaborative discussions.

5. Evaluation

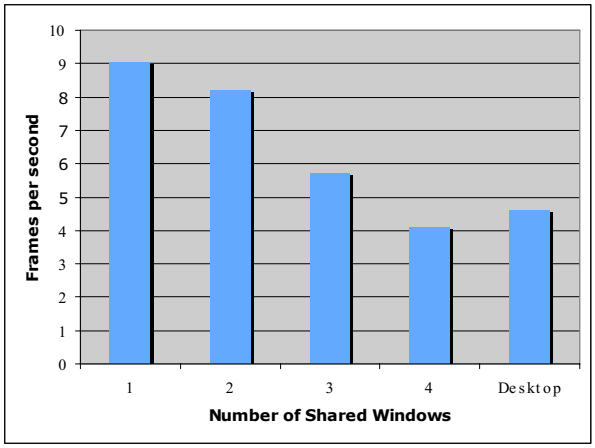
We have evaluated our system on its ability to provide adequate frame-rate and response time. In a Fusion control room, users typically share a few types of data including 2D plots, animated 2D plots and video clips from the Fusion engine camera. We test the system with similar workloads.

In the first experiment, since video clips are the most resource-intensive to share, we measured the frame-rates achievable when sharing multiple video clips from one workstation to the shared display. For this experiment we used a 3.2GHz Pentium 4 with 2GB of memory running Fedora Core 5 for both the scientist workstation and the computer driving the shared display. They were connected by 100Mbit Ethernet and the scientist workstation had a resolution of 1280x1024. The scientist workstation ran 4 video windows, each occupying about 25% of the screen (640x480).

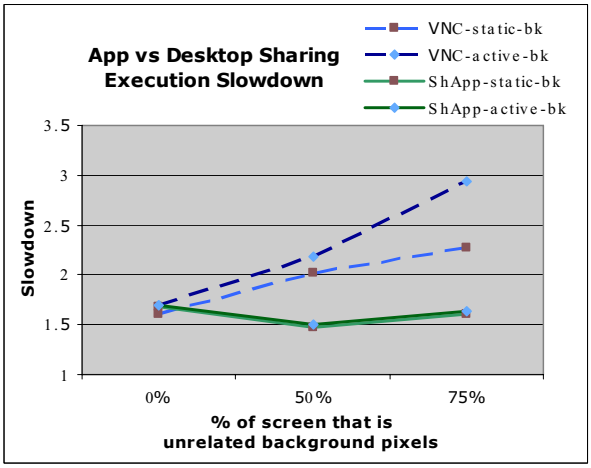
We measured the video frame-rate achievable while sharing 1, 2, 3 or all 4 windows simultaneously and compare that with sharing the entire desktop with unmodified VNC (graph 1). Sharing one or two windows achieved a frame-rate of about 9 and 8 fps respectively, still satisfactory for typical control room simulation videos. As additional videos were shared, frame-rates fell off as expected, giving about 4 fps with 4 windows shared. This validates our assumption that sharing smaller portions of the screen should provide better performance. For comparison, sharing the whole desktop using normal VNC gives about 4.5 fps, about equivalent to sharing all 4 windows with SharedAppVNC.

In addition to adequate frame-rate, interactive applications must provide good response time to be useable. To measure response time we made a trace of a user interacting with a 2D dataset application. We then used a robot tool to replay the user input from a remote display and measured the execution time. For comparison we measured the execution time of the trace on the local computer without sharing; this formed our base execution time, normalized to 1. We measured the execution time of the trace application occupying 100%, 50%, or 25% of the originating screen; this corresponds to background pixel activity occupying 0%, 50% or 75% of the screen. We ran two cases, one with the background pixels static and the other with the background pixels active. The active background case con-

sists of an image slide-show with image transitions every 1 second. The results are shown in graph 2. For single-window sharing using SharedAppVnc, the slowdown is typically a factor of about 1.5 for both the static background content and the slide-show background (solid lines graph 2). This is as expected because individual window sharing can ignore the background pixel data. We expect a slight performance improvement with smaller windows, a trend we see initially when the window reduces to 50% of the screen.



Graph 1: Frame-rates achieved using SharedAppVNC to share multiple video windows. SharedAppVNC achieves higher frame-rates by limiting the screen-area shared.



Graph 2: Execution slowdown of an application shared using VNC or SharedAppVNC. The window size and background pixel activity is varied. SharedAppVNC keeps relatively constant performance by avoiding sending background pixels.

We also measured the slowdown using normal VNC sharing the entire desktop (dotted lines, graph 2). Unlike with window-sharing, desktop-sharing slowdown is affected by the background pixel activity. For a full screen trace-window, the relative slowdown is equivalent to that of SharedAppVnc – about 1.5. For smaller trace-window sizes, the relative slowdown for VNC increases because VNC must also transfer the background pixel activity to the remote client.

These experiments validate our experience that SharedAppVnc can achieve better performance when smaller areas of the screen are shared, and that the performance slowdown for the remote user is tolerable.

6. Conclusion

We have proposed a virtually shared model for networked displays and user-input devices. Using this model we have implemented a prototype system, released it to the public domain, and deployed it into the control rooms of two DOE fusion research labs where it has been incorporated into daily production use.

Some quotes from scientists indicate their appreciation for the collaborative system. "[Previously] everyone had their own screen, or hardcopy. To collaborate, they usually looked over someone's shoulder. [The collaborative software] allows easy side-by-side comparisons of data from different people…[and] lets scientists make connections and correlations between displays and data sets that would be difficult without the wall."

These positive results and feedback encourage us to continue future research and enhancements to shared display environments such as implementing a direct communication model between networked components so that latency can be improved and permissions settings supported.

7. References

[Bara05] Baratto, R., Kim, L., and Nieh, J., "*THINC: A Virtual Display Architecture for Thin-Client Computing*", ACM Symposium on Operating Systems Principles (SOSP 2005).

[Cars99] Carstensen, P., Schmidt, K.," *Computer supported cooperative work: New challenges to systems design*". In K. Itoh (Ed.), Handbook of human factors, 1999.

[Gett05] Gettys, Jim, "*SNAP Computing and the X Window System*", Linux Symposium, July 2005.

[Gett04] Gettys, J., Packard, K., "*The (Re)Architecture of the X Window System*", Linux Symposium, July 2004.

[Hutt06] Hutterer, P., MPX, http://wearables.unisa.edu.au/mpx

[Meng94] Menges, J., Jeffay, K., "*Inverting X: An Architecture for a Shared Distributed Window System*", Workshop on Infrastructure for Collaborative Enterprises, 1994.

[Myer99] Myers, B. and Stiel, H., "*An implementation architecture to support single-display groupware*", CMU Technical Report, CMU-CS-99-139, 1999.

[Rich98] Richardson, T., Stafford-Fraser, Q., Wood, K., Hopper, A. "*Virtual Network Computing*", IEEE Internet Computing, 2(1), Jan/Feb 1998.

[Sato04] Satoshi, U., *MetaVNC*, http://metavnc.sourceforge.net

[Wall04] Wallace, G., Bi, P., Li, K., Anshus, O., "*A Multi-Cursor X Window Manager Supporting Control Room Collaboration*", Princeton University, Computer Science, Technical Report TR-707-04, July 2004.