HP SharedX: A Tool for Real-Time Collaboration

With this real-time communication product, two or more remote users can share and interact with the same X-protocol-based applications from their workstations. Windows are shared in such way that it almost seems as if all the participants in the shared session are sitting at the same workstation, running the same application.

by Daniel Garfinkel, Bruce C. Welti, and Thomas W. Yip

HP SharedX is a communication tool that extends the industry-standard X Window System to enable real-time sharing of X-protocol-based applications between two or more remote users and displays. With HP SharedX users can share information with one another via a workstation without being in the same location.

Being able to share information over a network in real time is a very effective productivity tool. The following two examples show how displaying applications across a computer network can increase productivity:

- System administration. Mary is a system administrator for several networks distributed over several widely dispersed buildings. When a user on a particular network encounters what may be an application or system problem, Mary can, with the user's cooperation, establish a common (HP SharedX) window with the user's system. With this shared window Mary can see and diagnose the user's problem while the user is watching what she is doing. She can perform this task without leaving her desk, using the diagnostic tools available at her workstation.
- Remote demonstrations. For some new or complex software packages, there may be only a handful of people who are able to demonstrate the software effectively. HP SharedX gives these people an opportunity to have a virtual presence at remote locations on the network. Hewlett-Packard's Work Management Operation routinely uses HP SharedX to demonstrate advanced features of its HP WorkManager database product from the desks of engineers in Fort Collins, Colorado to prospective buyers all over the world. Also, various partners who are developing additions to WorkManager use HP SharedX to demonstrate their work in progress to the engineers in Fort Collins, allowing new features to be evaluated "live" without travel.

HP SharedX can accomplish display sharing because of the nature of the system it is built upon, the X Window System.[1,2] The X Window System, known simply as X, is a networked window system allowing X applications running on one computer to display on another (see "X Window System Client/Server Architecture" on page 25). This networked aspect allows sharing of expensive high-speed computers among several users on low-cost X-based display stations. Furthermore, X is designed to be interoperable in heterogeneous computing environments. Applications running on one vendor's hardware can display themselves on another vendor's display by adhering to the X protocol standard. Heterogeneity is accommodated by abstracting the application's view of the display subsystem, including the graphics hardware, input device control, memory management, and data formats. Interaction with the display station occurs through a well-defined protocol, which provides a consistent way for applications to work with a wide variety of display devices, from PCs to high-resolution, deep-color workstations.

X has become the industry standard window system for workstations in part because of its networked and heterogeneous nature. HP SharedX builds on the X Window System by retransmitting the X protocol stream to multiple X display stations, thus simultaneously displaying a single application on multiple computer displays. Some key features of HP SharedX are:

- No changes are needed to existing X applications to share them. HP SharedX allows users to share virtually any X application, rather than forcing users to use specially written, collaborative applications.
- Users see a copy of the application to the best ability of their display device. If necessary, HP SharedX will degrade the quality of the displayed images to match the capabilities of the display hardware.
- Running applications can be shared without prior setup. Users need not restart applications they want to share; the application can be shared in its current state. This is important in consulting environments where users may not know how they got their application into its current state.
- The receiving machines (machines receiving the shared application) need no modifications or special software installed. Since X protocol is the communication mechanism, the receiving machine can be any X-capable display device.
- Receivers of a shared application can interact with it as well as the sender (the machine sending the shared application). Receivers can demonstrate operation of the application rather than describing to the sender what to do.

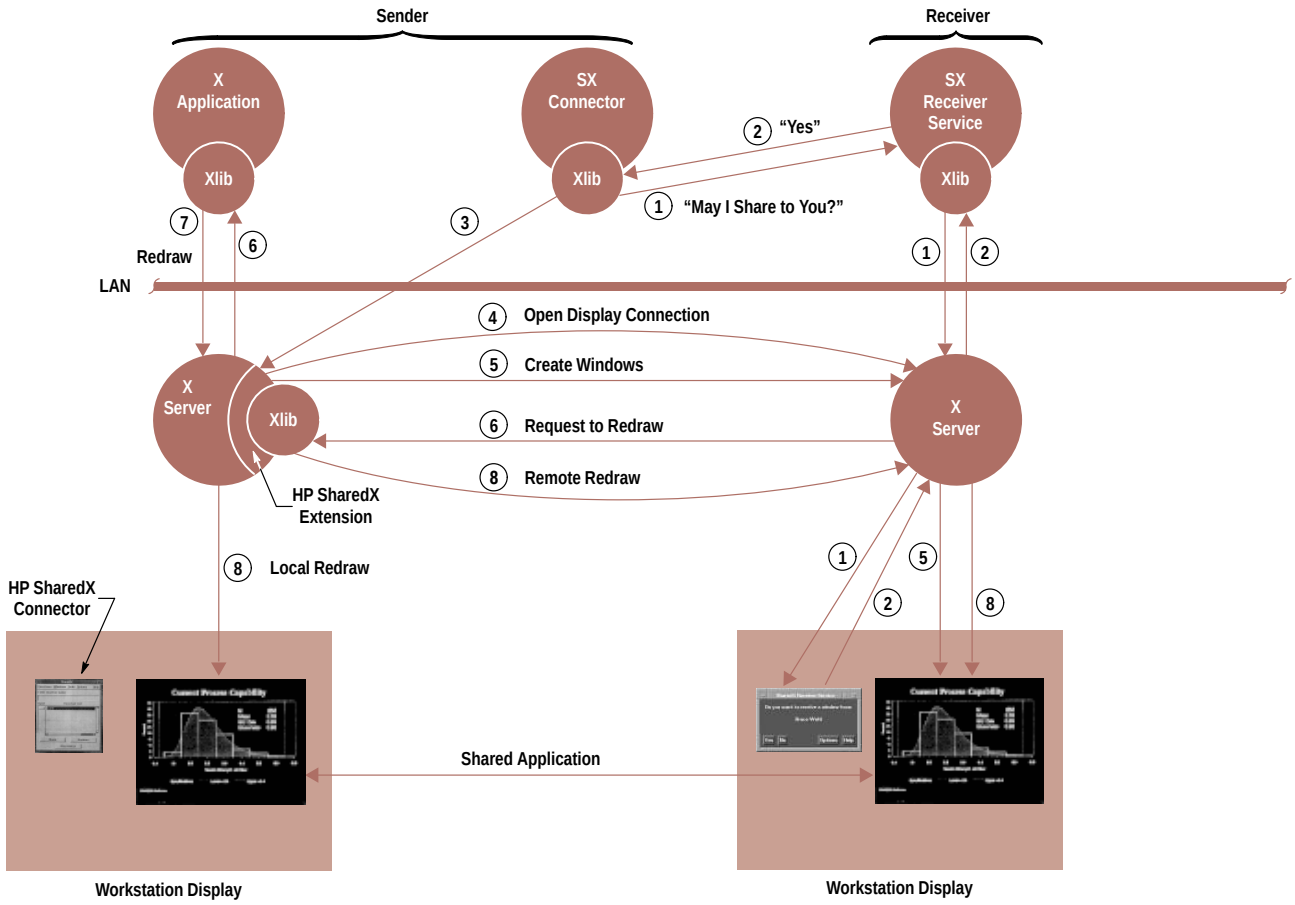

Fig. 1. HP SharedX sender/receiver architecture and data flows during initial display connection.

The rest of this article covers the architecture of the HP SharedX system, the user model and user interface for sharing, an overview of the low-level mechanism that multicasts the X protocol and merges input events, and finally, a description of how sharing is accomplished with displays of different visual types and different resources.

HP SharedX Architecture

HP SharedX consists of three components: the HP SharedX user interface (known as the connector), the HP SharedX receiver service, and the HP SharedX extension to the X server (see Fig. 1). The connector is the sender's control for sharing and unsharing windows, enabling and disabling receiver input, displaying sharing status, and so on. The receiver service is an optional process on the receiver's machine that simplifies sharing windows and increases security. The HP SharedX extension is a low-level mechanism that shares windows, keeps those windows up to date, and merges input from multiple users. These three components are described in more detail later.

An HP SharedX session begins when the user at the sending workstation specifies a receiver and pushes the Share button in the HP SharedX connector window (Fig. 2). After the Share button is pushed the sender selects the window to share by clicking the mouse button over the desired window. Once the desired window is selected the following events occur between HP SharedX processes on the sender and receiver workstations shown in Fig. 1. Each step corresponds to a number circled in Fig. 1.

Fig. 2. The HP SharedX connector window.

X Window System Client/Server Architecture

The X Window System, commonly referred to as X, is an industry-standard, network-transparent window system. X presents to the user a hierarchy of resizable overlapping windows providing device independent graphics. A graphical user interface is commonly included as an integral part of the X Window System.

The principal feature that distinguishes X from a conventional window system is its network transparency. The X Window System allows window applications or clients to access the display only through the X server, which is a separate process that arbitrates resource conflicts and provides display, keyboard, and mouse services to all clients accessing the display (Fig. 1). X can support a spectrum of hardware displays ranging from small monochrome units to advanced graphics systems with up to 32 bits of color per pixel.

The client and the server exchange information only by means of the X Window System protocol which can be implemented via any reliable byte stream. In the HP-UX* implementation of X, as in most others, this byte stream is implemented as a socket, which is a logical data connection between two processes on the network. Clients may reside locally with the display server or on a remote system across the local area network (LAN). A performance optimization bypasses the physical LAN when the client and the server are on the same computer.

Because the client program and the display server are two separate entities, the target display can be specified at the time the application client is run. The client program is indifferent. It sends out X protocol commands via calls to the X library (Xlib), which in turn calls the network service functions to communicate with the target X server.

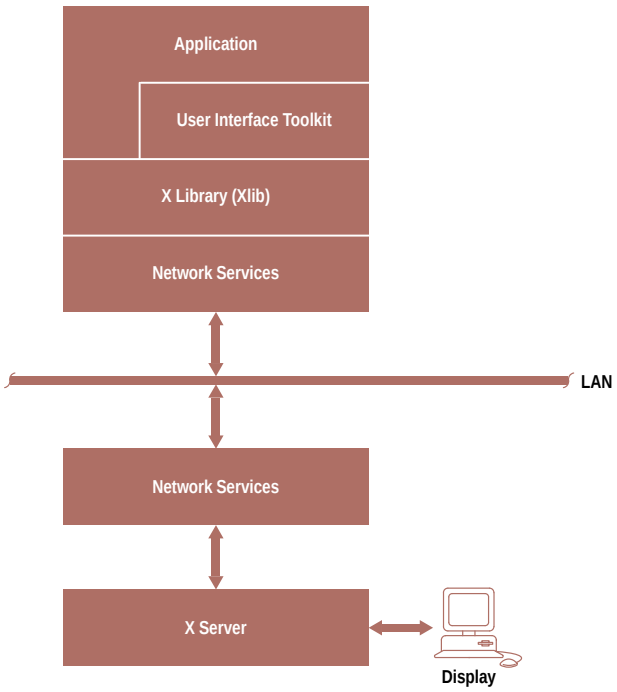

Fig. 1. X Window System client/server architecture.

1. The connector sends a message over the network to the HP SharedX receiver service on the receiver's machine. The receiver service displays a window asking the user at the receiver system to accept a shared window from the sender (Fig. 3).

2. The receiver passes back a yes or no response to the connector. If the answer is no, the sender is informed that access is denied to the receiver's machine.

3. If the answer is yes (or if the receiving machine does not have the receiver service), the connector sends a special X protocol request (share window W to receiver R2) to the sender's X server. The X server's main event loop recognizes the share window request as one directed at the HP SharedX extension. It calls the appropriate function in the extension to handle the request.

Fig. 3. The HP SharedX receiver service window.

4. The HP SharedX extension opens a connection over the network to the receiver's X server. This is done with the Xlib call XOpenDisplay.

5. The HP SharedX extension creates windows on the receiver's screen to match those on the sender's screen.

6. As the windows are created and mapped, the receiver's X server generates a request to redraw the newly created windows.

7. The application responds by redrawing the contents of all of its windows. X protocol drawing requests to redraw the windows are handled locally by the sender's X server, as usual, but since the application is now shared, the protocol is effectively echoed to the receiver (⑧ in Fig. 1). This is how the receiver is brought up to date with the sender when the share is initiated. Incidentally, none of the resources (graphics contexts, fonts, and so on) exist on the receiver when the requests start flowing from the shared application. Resources on the receiver are created when they are needed, that is, at the point they are first used, and not when they are created by the shared application. As a result, the input from the application to the sender's X server and output from the sender's X server to the receiver's X server might look like the following streams:

Input	Output
ClearWindow(win1)	ClearWindow(win1a)
DrawLine(win1, gc1, x, y)	gc1a = CreateGC(win1a, …)
DrawLine(win2, gc2, x, y)	DrawLine(win1a, gc1a, x, y)
	gc2a = CreateGC(win2a, …)
	DrawLine(win2a, x, y)

Graphics Glossary

The following are some graphics terms that may be of help in reading parts of this article that discuss graphics.

Color Map. A very high-speed look-up table that maps the numbers stored in the frame buffer (video memory) to actual color values which are converted to analog voltages and sent to the monitor.

Frame Buffer. The video memory of a display device in which each element represents one picture element, or pixel. The frame buffer is divided into two parts: onscreen memory (current visible image) and offscreen memory (graphics memory that is never visible).

Graphic Context. A set of attributes such as foreground and background colors, line styles, and fill patterns which are used by X clients to specify how the X server should render the drawing requests it receives.

Image Planes. The primary display memory on HP's display systems, used for rendering complex images.

Offscreen Memory. A portion of the frame buffer that cannot be displayed on the monitor. In all other respects, offscreen memory behaves the same as on-screen (visible) memory. Starbase and X use offscreen memory to hold character, cursor, pixmap, and scratch information for rapid transfers to onscreen memory.

Pixmap. A rectangle of image data maintained in offscreen memory when there is room, and in virtual memory when there is no room in offscreen memory.

Rendering. Any form of drawing operation, including text, line, and raster output. Rendering may occur to onscreen memory, offscreen memory, or virtual memory.

Visual Type. The color map capabilities of a given display. Common visual types supported on HP displays include 1-bit static gray (or monochrome), 8-bit pseudo color (having 256 color map cells of RGB values), and 24-bit direct color (using 8 bits each for red, green, and blue values).

Note that these are only representations of the drawing commands sent from the application and the sender's X server. The parameters win1a and win2a correspond to the different windows created in step 5. The identifiers win1a, gc2a, and so on represent resources on the sender's display which are mapped to resources win1a, gc2a, and so forth on the receiver's display.

Once the redraw step is finished, the share connection is established. From here on, the two displays are kept in synchronization by equivalent X protocol being sent to each server.

Connector

The HP SharedX connector (Fig. 2) is an X Window application that contains all the controls that enable users to set up and control application sharing. Using task analysis, we designed the HP SharedX connector to match the user's task flow, thus providing intuitive controls to the complex operation of sharing an application. Some of the usability features of the connector window include:

- An address book (Fig. 4), accessible from the Receivers menu item in the connector window, to keep and organize common receivers and allow users to specify receivers by user name rather than machine name
- A shared pointer (called a telepointer), available from the Windows menu item in Fig. 2, for users to point to areas of shared applications during collaboration

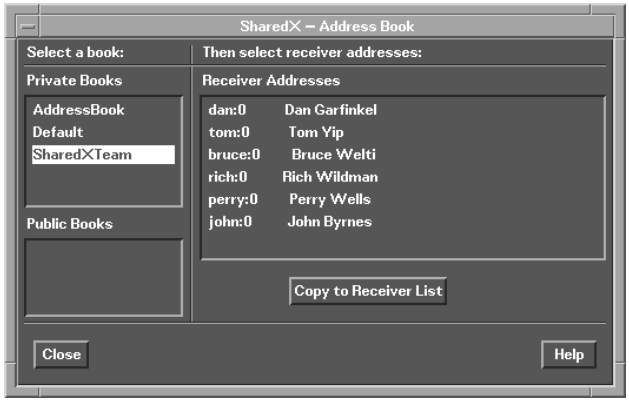

Fig. 4. A window showing the address book.

- Several cues, like changing a window title so that the user can keep track of the windows being shared, the users receiving shared windows, and the receivers that can send input to the shared application
- Various levels of dialog information (For example, when the user starts using HP SharedX, a Normal dialog level is set giving the user complete status information about the quality of the shared application. When the user sets the dialog level to Experienced, only critical errors are reported.)
- Extensive online help information, including a troubleshooting section for diagnosing and fixing problems when sharing applications.

After the first release of HP SharedX, we gathered information on the product to determine major usability issues. From this analysis three major issues emerged:

- The need to annotate a shared window
- The need for a "shared whiteboard" to assist in collaboration
- The difficulty and security implications of the receiver's granting display access to the sender for shared applications.

The first two usability issues are addressed by the HP SharedX Whiteboard (see "Whiteboard: A New Component of HP SharedX" on page 28). The third issue is addressed by the HP SharedX receiver service (described below), a program that is invoked on the prospective receiver's machine when sharing is attempted.

Receiver Service

When a sharing request is made to the connector via the Share button, the connector software first tries to invoke the receiver service program on the receiver's computer. The receiver service displays a dialog box on the receiver's display (Fig. 3). The dialog box informs the user that the sender wants to share a window and asks for a yes or no response. If the user selects yes, the receiver service grants access to the receiver's display for sharing.

The connector can get one of three responses from the receiver service. The receiver can answer yes, in which case the connector will share the specified window. The receiver can answer no, in which case the connector will display a dialog box informing the sender that access permission was denied. Lastly, the receiver service may not be present, in which case the connector will attempt to share with the receiver's machine.

Fig. 5. Different application sharing architectures. (a) Centralized pseudoserver architecture. (b) Replicated architecture. (c) Sharing-library architecture. (d) Integrated architecture.

Once HP SharedX has shared windows to the receiver, it tells the receiver service to remove display access from the sender. This prevents processes other than HP SharedX on the sender from making unauthorized connections to the receiver's display.

HP SharedX Extensions

The HP SharedX extensions are low-level routines responsible for intercepting data streams that flow in and out of the X server during a sharing session. These routines merge input from multiple receivers and ensure that the receivers' displays are updated properly.

Several groups of researchers have attempted application sharing in the X window system environment.[3] These implementations can be classified into four different architectures, each with inherent strengths and weaknesses[4] (see Fig. 5). The main goal of sharing X applications is to ensure that the application and the basic X server don't have to change to work in a sharing environment. This requires some sort of interface mechanism between the application and the X server.

By far the most popular architecture for window sharing in X is the centralized pseudoserver architecture (Fig. 5a). This architecture places a process responsible for output multicasting and input merging between the shared applications and the X server.† This architecture is the most flexible, but suffers from performance problems, even by unshared applications, because it places a process between the application and the X server.

Other application sharing systems have used a replicated architecture for sharing windows (Fig. 5b). The replicated architecture runs a copy of the application for each receiver of the shared application and keeps these copies synchronized by sending identical input to each. Although this architecture optimizes the image for each receiver of the shared application, it has been shown to be difficult to keep the instances of the application synchronized and nearly impossible to add users to an existing sharing session.

The sharing-capable library (Fig. 5c) moves the output multicasting and input merging found in the pseudoserver architecture into the library of X functions (Xlib) linked into the application. By moving the sharing into an existing process and removing the pseudoserver process, performance is enhanced, but applications in such a system cannot be sure of sharability, either because they have not linked with this special library, or because they are running on a machine in the network that does not have the sharing-capable library.

The integrated sharing architecture, which is used in HP SharedX, moves the output multicasting and input merging routines inside the sender's X server (Fig. 5d). This architecture ensures that all applications are sharing-capable, while eliminating the pseudoserver process.

(continued on page 29)

† Output multicasting is the generation of multiple streams of requests from a single stream. Input merging involves coalescing multiple streams of events into a single stream.

Whiteboard: A New Component of HP SharedX

Whiteboard, a general-purpose image viewer and annotation X application, is a new addition to HP SharedX. Whiteboard evolved from users' needs to annotate graphic images in shared applications. Although it is impractical to annotate images in live X applications because of limitations in the X Window System, Whiteboard allows users to share snapshots (portions) of their display and to annotate those snapshots. Not only is Whiteboard useful for adding annotation to computer images, it can also be used to create original images. As an X application, Whiteboard is treated just like any other X application when it is shared in the HP SharedX environment.

Some of the other features that make Whiteboard convenient, powerful, and very functional when shared include:

- Annotation performed on an image can be erased without destroying the original image.
- Any region of the user's screen can be copied and pasted into the Whiteboard drawing area, even if the region contains areas of different X visual types. This ability to copy an arbitrary region containing multiple visual types is a new capability for X applications.
- When shared, Whiteboard knows when input is coming from the sender or receiver. Thus, it can respond differently according to the source of input. This capability is used to ensure that receivers are restricted in what screen regions they can copy to the Whiteboard.

Fig. 1 shows a typical Whiteboard display.

Annotating Images

The main feature that differentiates Whiteboard from other drawing applications is the concept of erasable and unerasable layers. Maintaining two layers allows users to erase image annotations without affecting the image being annotated.

When an image is loaded into Whiteboard, it is in the unerasable layer. Annotation can be performed in the erasable layer and then erased without destroying the original image. This is analogous to drawing an image on a real-world whiteboard in indelible ink, annotating the image in erasable ink, and then erasing the board leaving the original image. This user model is convenient in collaborative situations where changing just the image annotation is desired. For convenience, Whiteboard allows annotation to be hidden temporarily to reveal the unerasable image.

Copying Multivisual Regions

Copying a region from the screen is not as trivial as one might suspect. To an X programmer, using the Xlib function XCopyArea to copy a root window region into an XPixmap buffer might seem like an obvious solution. However, XCopyArea cannot handle cases in which the selected region includes areas of multiple visual types or visual types different from that of Whiteboard; the source and destination visual types must be the same to use XCopyArea. In more recent X displays, a screen can simultaneously support multiple visual types differing in depth and types of color maps. Therefore, it is possible to have child windows of differing visual types in the window hierarchy, which is not an uncommon situation.

An algorithm was developed to convert all image data in the selected region to the destination visual type. This algorithm involves scanning the selected region for all windows and parts of windows, coalescing those of the same visual types into subregions, then reading and converting each subregion to the corresponding subregion of the destination pixmap (paste buffer). If any subregion in the selected region matches the visual type of the buffer, a simple XCopyArea call is used to transfer the image information. Otherwise, the conversion algorithm uses functions in HP's Image Library to convert each subregion to the destination visual type.

If the X server on which Whiteboard is running supports deeper visual types than the default, Whiteboard keeps the paste buffer in the deepest visual type available, even though Whiteboard is running in the default (shallower) visual type. This technique maintains the buffer contents at the highest possible color fidelity, so that when a scaled paste is done, making the image larger, the resulting image does not contain pronounced dither patterns.

Recognizing the Source of User Input

Whiteboard is the first application that is able to detect the source of user input when the program is shared. Whiteboard uses a feature of the HP SharedX extensions to the X server on the sender to examine events and determine the source of input. Based on whether the sender or receiver generates the input, Whiteboard performs a different operation when the Copy Region button is pressed. When the sender presses the Copy Region button, Whiteboard allows the user to select a region on the sender's screen to copy. However, for security reasons a receiver should not be able to copy the sender's whole display. Instead, a Copy Region operation by the receiver confines the function to the Whiteboard drawing area.

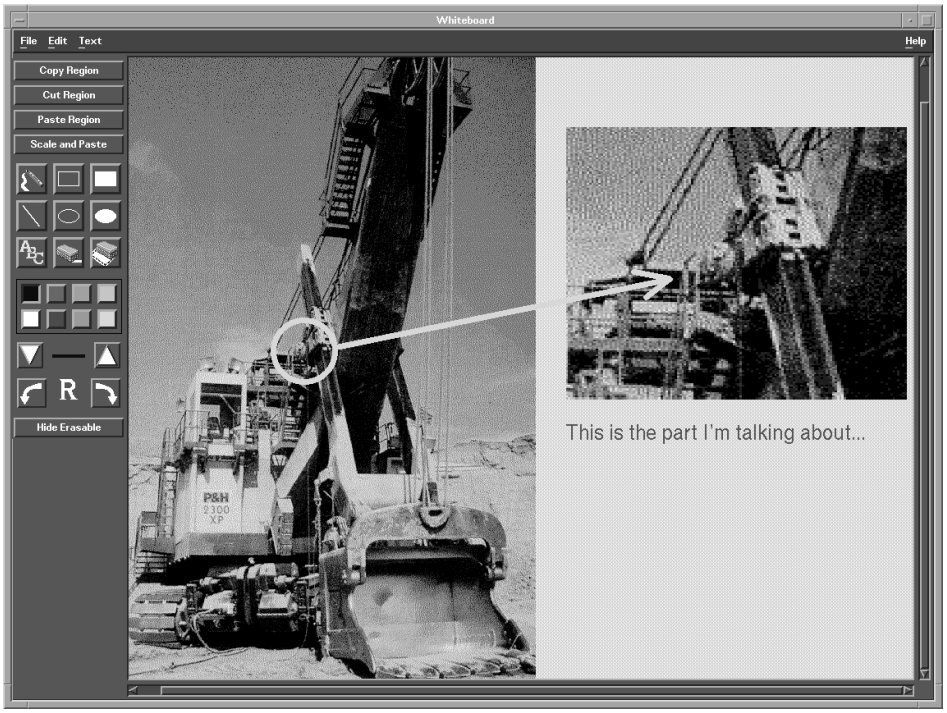

Fig. 1. A typical Whiteboard display. The white arrow with the circle on the end and the text illustrate a typical annotation. The picture on the right, which shows a zoomed-in portion of the main image, is an example of copying and pasting a portion of an image into the Whiteboard.

Ideally, Whiteboard should perform the copy from the display from which input is initiated. Unfortunately, because Whiteboard does not have a direct connection to the receiver's display when the application is shared, no copying operations can be initiated from the receiver's screen. Although the implemented Copy Region function does not present the most ideal solution, it does maintain complete functionality for the sender while providing the receiver with a reasonable substitute for the copy operation.

The advantages of the HP SharedX extension architecture (integrated sharing architecture) include:

- Any X application can be shared at any time
- Unshared applications do not suffer a performance penalty
- Sharing performance is optimized because state information about the X server is directly accessible to the sharing mechanism. (Other implementations require the sharing mechanism to query the X server state over the network, slowing down the sharing process.)

The main disadvantage of the HP SharedX architecture is the need to modify the X server, which requires access to the source code for the X server. Using the sharing-library approach, an application can be linked with a sharing-capable Xlib and can then be run with any standard X server.

Although these architectures for sharing applications differ, the basics of sharing an application are the same for each (with the exception of the replicated architecture). These basics include making instances of windows on receivers' displays, echoing the rendering requests of an application to each receiver, matching resources on the sender machine with those on the receiving machines, and merging input events from the multiple X servers into a single stream and returning the information back to the application. The following sections give a detailed look at how HP SharedX deals with some of these issues in sharing applications.

Limitations of HP SharedX

Applications are displayed on HP workstations in one of two ways. They are either displayed via the X server using X protocol, or displayed directly using direct hardware access (DHA).[5] DHA allows applications to bypass the X server to render graphics on the display. Since HP SharedX operates by retransmitting X protocol to the receiver's display, applications based on X can be shared, while DHA programs cannot be shared directly. The static images from DHA applications can be shared by copying their windows into the Whiteboard application described on page 28.

Programs that render through the DHA mechanism include those that use the Starbase graphics package or PEX† programming interface. However, the programming interface does not make the final determination of the rendering path. For example, Starbase programs can be run with a graphics driver that emits X protocol instead of using DHA. While there is usually some performance penalty for using these drivers, it does allow the application to be shared.

Even if an application is based on X, it may not share perfectly. For example, the audio application of HP MPower uses X protocol to display its control panel but interacts with the audio server to generate sound. Even though the control panel shares properly, when the receiver presses the play button, the audio is played only on the sender's audio server (see Fig. 6).

† PEX are 3D enhancements to the X protocol.

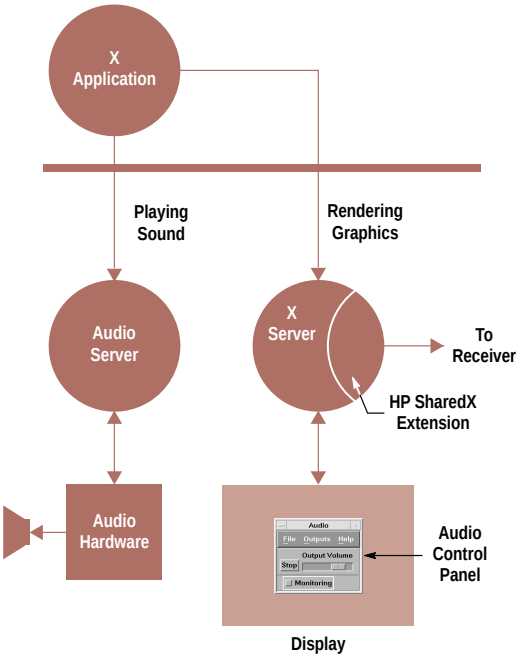

Fig. 6. An HP SharedX configuration in which output from a shared application is not echoed on the receiver. In this case the output is going to the audio server which knows nothing about the receiver.

Another example of applications that do not share perfectly are those applications that use X extensions. The X extension is a mechanism for extending the capability of the X Window System. While the core X protocol is standard among all X servers, X extensions vary among X server implementations. HP SharedX only retransmits core X protocol, so applications that use X extensions will have limitations when shared. For example, if an application uses input devices other than the keyboard and mouse via the X input extension,†† the receiver will see a copy of the window but will not be able to interact with the application with these additional input devices.

There are parts of the core X protocol that HP SharedX cannot handle properly. One example is the X mechanism for cut and paste. This mechanism uses a standard interprocess messaging system for transferring data from one X program to another. However, a receiver cannot cut and paste between a shared application and one on the local machine. The two applications cannot communicate since they are not connected directly to the same X server.

Establishing and Maintaining Connections

When a share request is made to a receiver, HP SharedX must first establish an X connection to the receiver's X server. It is over this connection that HP SharedX will manage the shared windows, retransmit rendering requests, and get receiver input. The display connection is made using the X library (Xlib).

Xlib is the client-side library of functions used by all X applications. These functions create the protocol requests that

†† The X input extension allows applications to receive input from input devices such as graphics tablets, knob boxes, button boxes, and so on.

become the X protocol packets sent over the display connection to the X server to request creation of windows, drawing to those windows, and so on. Although the HP SharedX extension is in the sender's X server, it communicates with the receiver's X server using Xlib. This makes the sender's X server appear to the receiver's X server the same as any other X client.

The HP SharedX extension, however, has some requirements that are different from other X clients, so we made some changes to Xlib to accommodate these requirements. The first change we made to Xlib was to create a mechanism to recover from broken display connections. In X, programs that lose their connection to the X server normally print an error message and exit. For most X programs, their one connection to the X server is their lifeblood, and if it is broken, they may as well terminate. The sender's X server still has plenty to do if it loses a sharing connection, and thus it should definitely not terminate. Xlib calls a user-specified routine, XIOErrorHandler, when a display connection is broken. In HP SharedX's version of XIOErrorHandler, the routine cleans up data structures related to the broken display connection, sends a notification to the HP SharedX user interface, and jumps back to a location inside the X server to continue processing.

A similar problem occurs when the remote X server is not responding. This can happen for a variety of reasons such as that another program may have taken exclusive access to the X server, the network between the two computers may be very busy, or the receiver's X server may be busy handling requests from other clients. In any case, X programs wait indefinitely for requests to be serviced before proceeding, which is not the desired behavior for the sender's X server. Therefore, HP SharedX has added a timeout handler to Xlib that allows the sender's X server to recover from a nonresponsive receiver. After 30 seconds of waiting on a receiver's X server, HP SharedX closes the display connection and notifies the sender that the receiver's system is not responding.

The final change we made to Xlib handles failures to establish an X (display) connection. Normally when a display connection fails, the program is not given a reason for the failure. A mechanism was added to extract the reason for display connection failure from Xlib and to return that information to the user interface. This information allows the user to diagnose problems more easily when attempting to share windows.

Managing Shared Windows

When users share an application, they usually have a clear idea of what constitutes the application and what set of windows should be shared. Most applications consist of a single program with a single connection to the X server. For these programs, it is easy for HP SharedX to share the right windows. However, an application can consist of more than one program (e.g., HP SoftBench), or a single program can display several windows that should not be grouped together (e.g., the HP VUE file manager can display multiple windows, each looking at a different directory). A user of the HP VUE file manager usually wants just one window to be shared, while a user of HP SoftBench may expect the debugger to be shared along with the static analyzer.

From the point of view of the X server, windows partake of two relationships. One is a parent-child relationship, which defines a window tree with all windows as descendants of the root window. The other relationship is that in which each window is owned by a display connection. Each window is uniquely identified by its window identifier, a 32-bit number in which seven of the bits are the same for windows owned by the same client. These seven bits are called the resource base.

Applications typically consist of one or more top-level windows in which applications display information. Because windows are relatively inexpensive to create, they are used extensively inside an application's top-level window for everything from drawing areas to buttons and menus.

When the user selects an application to share, HP SharedX must select a group of windows that best represent the application to send to the receiver's display. HP SharedX makes two assumptions: all top-level windows belonging to a single client connection (i.e., having the same resource base) belong to a single application, and all child windows of a shared top-level window should be shared with that top-level window.

Using these assumptions, if the sender shares an HP VUE file manager window, HP SharedX will send to a receiver all file manager windows. Users have the option of selecting Share Alone from a pull-down menu, but they could easily forget which applications to do this for. However, HP SharedX can be configured to respond differently to the main Share button for different applications. For many applications, HP SharedX is shipped preconfigured to share only the application's selected top-level window and its children.

The other problem mentioned above, that of multiple programs that should be shared together, has not been addressed directly. While application developers could use the HP SharedX command line interface to tie their applications together, HP SharedX does not provide a simple mechanism for grouping windows from different programs. Typically, for these applications the user must manually share all the desired windows.

Input Management

HP SharedX allows both the sender and the receivers of a shared application to interact with an application. In X, application input consists of events from the user's input devices and queries that the application makes about these devices. For example, the application can query the position of the mouse cursor, which, if the application is shared, has as many values as there are viewers of the application.

Most application-sharing systems implement one of two input merging policies: floor passing or free-for-all. The floor-passing scheme allows one user at a time to input to the application, with the choice of the user who has the floor manually controlled by a moderator. The floor-passing scheme ensures that input is not mixed from multiple users, but it requires explicit user action to change the input floor. The free-for-all policy allows anyone to input without explicit action, but it is prone to intermixed input from multiple users.

The HP SharedX input model, called *dynamic floor passing,* is a hybrid of these two methods. In this scheme, only one user at a time can give input to the shared application. Thus, user input does not become intermixed. However, input may change among users dynamically without explicit action, but only after a specific period of inactivity by the current user giving input.

In addition to answering the problem of intermixed input, dynamic floor passing solves the problem of input queries. This method always returns the state of the user's input devices currently or most recently interacting with the application.

The final challenge of handling input from multiple users is translating the actual input event that occurred on a receiver into an event that could have occurred on the sender. Translating a window identifier is trivial because HP SharedX maintains a mapping of which receiver windows correspond to which sender windows. Translating the keycodes in the event message is more difficult.

The keycode specified in an event is an index into a mapping table of logical key symbols that can vary from one X server to another. For example, one server can map keycode 52 to the letter "a", while another server can map the same keycode to the letter "z". The keycode mapping table is loaded into the shared application when the display connection is made, but the mapping can be changed dynamically.

When an event with a keycode is received by HP SharedX from a receiver, it is translated into a key symbol based on the current mapping for the receiver's X server. HP SharedX searches the sender's keyboard mapping to see if any keycode matches that same key symbol. If it does, the matching keycode is returned in an input event to the application. If no corresponding keycode on the sender's machine exists, the key event is discarded since no logical keycode can be sent to the application.

The flow diagram in Fig. 7 shows the operation of the input merging routine, including dynamic floor passing control and event-code translation.

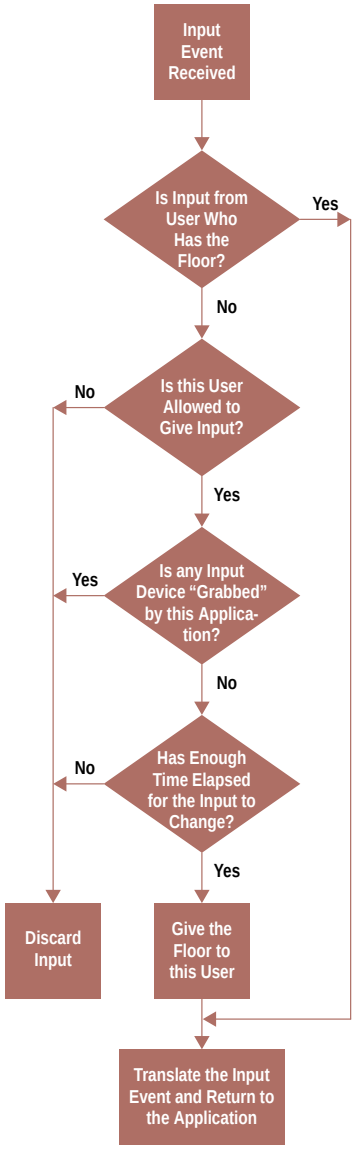

Fig. 7. Flowchart for the HP SharedX input merging routine.

Allocating Display Resources and Rendering

HP SharedX uses a "lazy" allocation scheme for display resources such as colors, pixmaps, graphics contexts, and fonts. To minimize initial sharing time and the impact on the receiving machine, display resources are allocated only when needed for a rendering request. For example, a shared application may allocate a large pixmap that is only displayed for some error condition. If that error condition never occurs during the sharing session, it would be a waste of time and resources to make an instance of the pixmap on the receiver's machine.

Once allocated, the display resource mapping is maintained so that future requests for that same display resource are satisfied without contacting the receiving X server. For example, if an application requests a line be drawn to a shared window, HP SharedX performs the following procedure:

```
draw the line to the local display;
for each remote instance of the window
begin
    if a remote instance of the graphic context has already
    been allocated,
      use that instance,
    else
      allocate a new remote graphic context and map it to
      the local graphic context;
    draw a line to the window instance with the graphic
    context instance;
end;
```

When the allocation of a display resource fails, HP SharedX uses several means to recover. In the case of pixmaps, HP SharedX continues operation of the application without the use of that image and notifies the user of this situation. In other cases, HP SharedX will substitute other display resources for the ones it cannot allocate. The next two sections provide detailed examples of two display resources HP SharedX will gracefully degrade: colors and fonts.

Managing Colors

The management of display resources such as graphic contexts and windows in HP SharedX is fairly straightforward compared to the management of colors. The goal of displaying identical images on all shared windows requires mapping colors from the sender's display to the receiver's display and degrading the colors gracefully if the exact colors are unavailable.

The term pixel refers to the number that represents a color value. The pixel can be one of two types: read-only, which represents a color that does not change and can therefore be accessed by multiple programs simultaneously, or read/write, which can be changed and is exclusively used by a single application.

Pixels are converted to colors either by using the pixel as an index into a color map (a look-up table) or by treating the pixel as a representation of the color itself. The interpretation of the pixel is based on the visual type,† of which there are six in the X window system. Three of these allow X programs to allocate read/write cells (grayscale, pseudo color, and direct color), while the other three provide a fixed set of colors (static gray, static color, and true color).

To divide the six visual types another way, four of them (static gray, grayscale, static color, and pseudo color) use a single color map to map a pixel into a gray value or an RGB triplet (Figs. 8a and 8b). These displays typically use between one and eight planes, so their color maps typically store between 2 and 256 color values. Another visual type, direct color, uses three color maps, one each for red, green, and blue. The pixel is decomposed into three indexes for look-up in the three color maps (see Fig. 8c). The last, true color, decomposes the pixel directly into red, green, and blue values (see Fig. 8d). True color and direct color typically use either 12 or 24 planes, yielding 4,096 or 16,777,216 colors.

† Color map capabilities (see glossary on page 26).

Our color mapping solution addresses two questions. Given the sender's and receiver's visual types, and the sizes of their color maps (number of different colors available):

- How are the available colors used to best advantage?
- How is color mapping maintained?

Using the Best Colors Available. The key to color matching is always to have a color that is close enough if the exact color is unavailable. If the receiver's color display system is static, supporting only read-only colors, the solution is simple: map each sender color into the closest receiver color. For receivers with dynamic color systems that support read/write colors, a color ramp is created. The color ramp contains a set of colors evenly distributed throughout the RGB color space, a three-dimensional space defined by axes of red, green, and blue values (see Fig. 9). The benefit of the ramp is that any color matches a ramp color within some maximum color difference in the color space. However, the maximum color difference may still be larger than is desirable, so the ramp is a fallback if an exact color match cannot be allocated.

A color ramp array and the values for the receiver's color map are created when the display connection is made to the receiver for sharing. Fig. 10 shows an example of a color

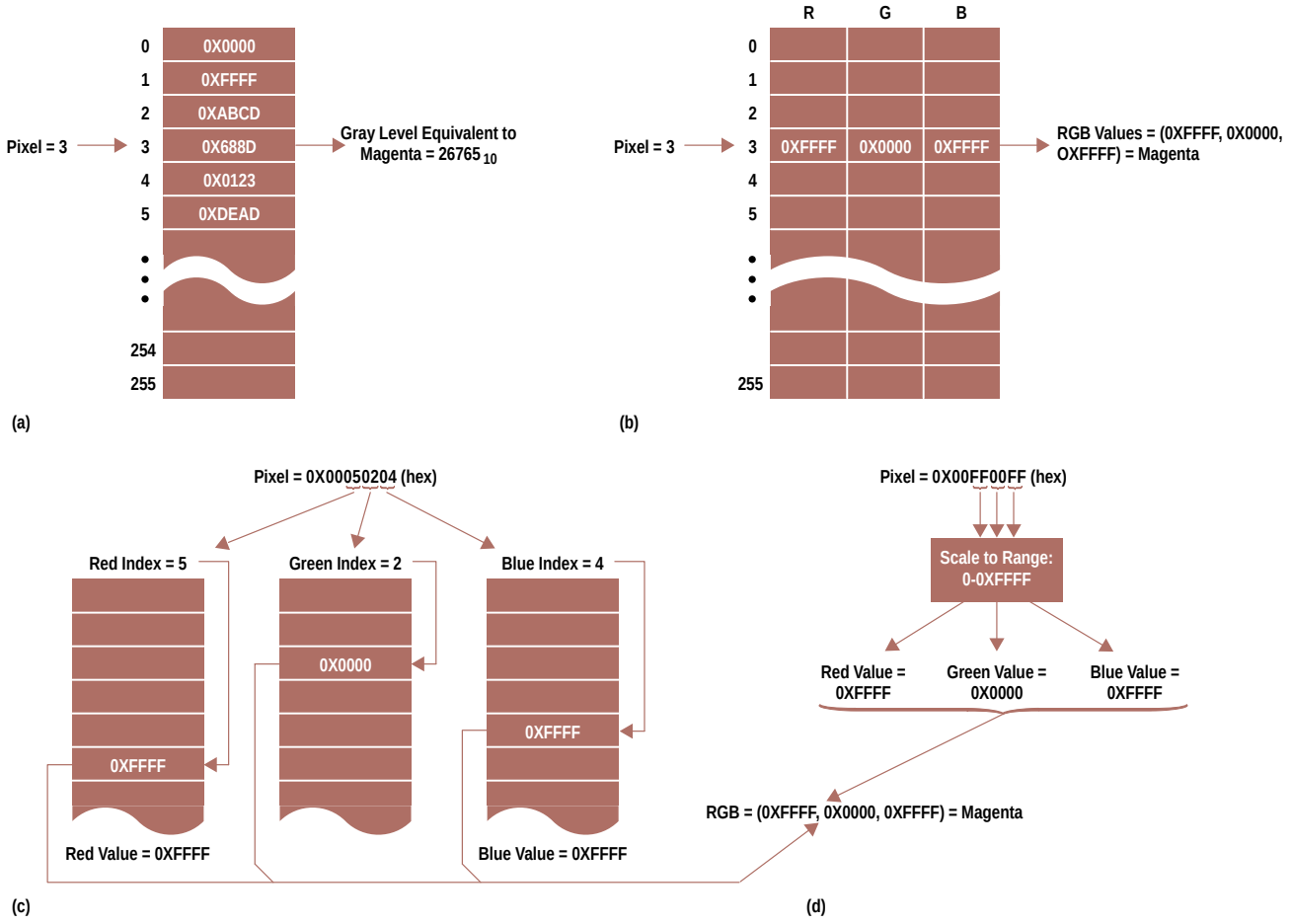

Fig. 8. Four representations for the color magenta. (a) The read-only gray level (static gray or grayscale) equivalents for magenta. (b) Read-only static color or pseudo color representations. (c) Direct color representation of magenta. The pixel is split into indexes into the color map. (d) True color representation. The pixel value is split into parts, and each part is scaled to create the values for the different shades of red, green, and blue for magenta.

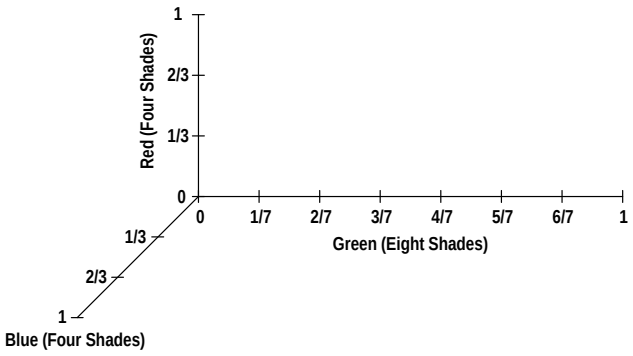

Fig. 9. A three-dimensional representation of the color ramp for an 8-bit color display with four shades of blue, four shades of red, and eight shades of green.

ramp array that contains pointers to the color map on the receiver. The values in the color map are determined from the three-dimensional color ramp mentioned above. As each entry is allocated in the receiver's color map, the index to that entry is sent back to the sender and placed into the ramp array table, creating a table of indexes into the receiver's color map.

The size of the color ramp depends on the size and type of the destination visual type. For grayscale visual types, a grayscale ramp of up to 16 values is allocated. For direct color visuals, up to 16 levels of red, green, and blue for a total of 4,096 distinct color values are allocated. In both of these visual types, 16 levels provide enough resolution to ensure a good color match without using excessive color map entries. The ramp used for eight-plane pseudo color consists of all combinations of four evenly spaced values of red, eight values of green, and four values of blue, for a total of 4 by 8 by 4 = 128 colors. The ramp takes up half of the available colors in an eight-plane color map.

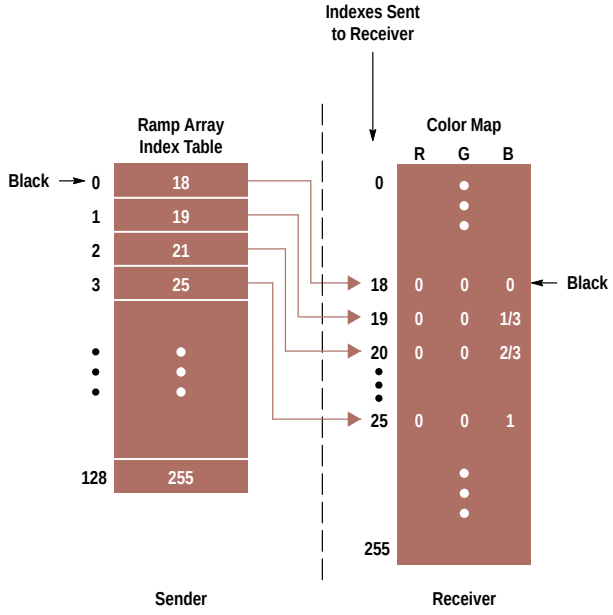

Fig. 10. The color ramp array containing indexes into the receiver's color map. Both of these items are created when the connection between sender and receiver is first established.

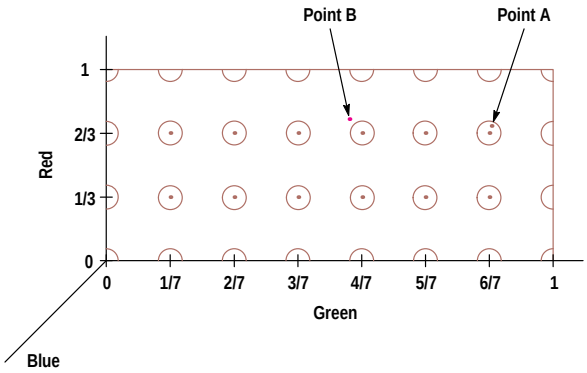

Fig. 11. A two-dimensional representation of the color space used by the color-matching algorithm.

Eight-plane pseudo color visual types are the most common and most difficult type to deal with since not enough colors exist to cover the color space adequately with a ramp. To compensate for large color errors, HP SharedX judiciously uses the remaining color cells available in addition to the ramp. For any given color, the following color-matching algorithm is used to map the sender's colors onto colors on the receiver:

1. If the X application on the sender allocated the color as read/write, then attempt to allocate a matching read/write color on the receiver. A read/write color is needed since the application may change the color of the allocated pixel at some later time and, to maintain color accuracy, the receiver's matching color must be changed as well.

2. If the application allocated a read-only color or if the above allocation failed, find the closest color from the ramp or a previously allocated read-only color. If that color is close enough to the desired color, use it as the match.

3. If the closest read-only color is not close enough, try to allocate a new read-only color. If this succeeds, add the color to the list of allocated read-only colors and return the new pixel as the match.

4. If the read-only color could not be allocated, use the closest pixel from step 2 since it is the best available match. Notify the user that the colors do not match exactly.

Fig. 11 shows a two-dimensional representation of the color space given in Fig. 9. Each circle encompasses colors that are close enough to a particular color ramp value (represented by the point in the middle of each circle). Each of the points represents a read-only color that has already been allocated on the receiver's X server.

The color-matching algorithm first checks to see if a color is close enough to an already allocated color. Point A in Fig. 11 is close enough so another cell is not allocated (step 2 in algorithm). Thus, for point A, 2/3 red, 6/7 green, and some value for blue is used.

Point B in Fig. 11 is not close enough to any allocated color, so a color is allocated and point B is added to the list of available colors (step 3 of the algorithm). If B cannot be allocated in the color map, the closest color is selected that has already been allocated, although technically it is not close enough (step 4 of algorithm).

The difference between the desired color and the candidate color is measured as the sum of squares:

$$\text{diff} = (\text{desired\_red} - \text{actual\_red})^2 + (\text{desired\_green} - \text{actual\_green})^2 + (\text{desired\_blue} - \text{actual\_blue})^2$$

The color with the minimum difference is the closest color. A close enough threshold was determined empirically to optimize the accuracy of color images while minimizing the number of receiver color cells used. Since images tend to have color themes (i.e., the colors are clustered in a few regions of the color space), a fairly high degree of color accuracy can be obtained without excessive demand for color cells.

Private Color Maps. An X display has a default, shared color map that most applications use. If an application needs more colors than are available in the default color map, it can create and use a private color map over which it has complete control. When an application using a private color map has its window focused, that application's color map is installed (copied into the display hardware) by the X window manager. On displays that support only one color map in hardware (most of the low-end displays), everything on the entire screen is displayed using the installed color map. When a private color map is installed, all applications using the default color map take on random colors. As soon as an application using the default color map gains the focus, the default color map is reinstalled, and the application with the private color map will have random colors. As the current color map switches back and forth from default to private, the user sees color flashing. The user typically prefers to display shared windows with the default color map to avoid this irritation.

When a window is shared to receivers with display types that have read/write color maps, a color ramp is created on the receiver's X display. The ramp is created in the default color map to reduce the probability of color flashing. If the desired ramp cannot fit in the default color map, it is placed in a private color map.

The system user interface is usually the first process to allocate pixels, and the pixels with the smallest indexes are allocated first. When a private color map is used, HP SharedX copies some of the lower pixel values from the default color map into the private color map. This way, when X switches between the two color maps, the colors used by the system user interface do not flash. The number of pixels copied is optimized to reduce color flashing while leaving color cells for later allocation.

Keeping Track of Pixel Mapping. To minimize the number of times the color matching algorithm is executed, a mapping of previously matched pixels is maintained. Different mapping methods are used depending on the range of possible sender color values. For small ranges (up to 256 different colors) a simple array is used, in which:

```
receiver_pixel = map [sender_pixel]
```

When the sender is a 24-plane display, this approach would require 48 megabytes of memory, so a more memory-efficient mapping scheme is needed. Since the goal is to produce close colors for the receiver rather than exact color matches, resolution of the sender color is reduced before applying the color mapping. Red, green, and blue values of the color are reduced to four bits rather than eight bits for each plane, which results in 16 (2^4) levels of each. Thus, the total number of possible color values is 4096 (16^3), a reasonable size for a mapping array. If color() is a function that returns the RGB values of a pixel, and crunch() is a function that converts an RGB triplet to a number in the range 0…4095 (four bits per pixel), then the mapping is as follows:

```
receiver_pixel = map [crunch (color (sender_pixel))]
```

The result of this mapping is that all colors close to a mapped color are mapped with that color. This method, called color-zone mapping, gives fast performance and adequate color matching, while keeping memory use fairly small.

Matching Fonts

The fonts in which applications display text can be specified by the user in the X Window System. From the set of fonts supported by the X server, the user selects a font that is aesthetically pleasing. Since there are no standard fonts in X, each X server can support a different set of fonts.

To maintain consistency between the sender and receiver of a shared application, it is important that text be displayed in a similar font on the receiver's display. HP SharedX employs a font matching algorithm to ensure a close font match.

Fonts in X have both a name and characteristics. A font can be loaded by specifying either its name or its characteristics using the X Logical Font Description (XLFD) format. HP SharedX first tries to match the font by name since it is unlikely that fonts with the same name differ. If the font with the same name is unavailable on the receiver's X server, fonts are matched by their characteristics.

The XLFD description defines 13 characteristics of a font such as its size, character set, weight, slant, and style. When matching fonts for the purpose of sharing an application, some of these characteristics are more important than others in choosing the font with the closest characteristics.

HP SharedX obtains a list of all XLFD type fonts from the receiver's X server at display connection time. When a font match is needed, the source font's characteristics are compared to those available on the receiver's X server. A weighted sum of differences of the characteristics is calculated and the font with the minimum difference is considered the closest match.

The weighting of characteristics was determined through a combination of intuition and observation. With the exception of character set, discussed below, it seems clear that size is the most important criterion, with width taking precedence over height. This is partly because many X applications write text in two different ways. One way is to pass a whole string of characters to the server and let it determine the location of each character based on its knowledge of the font. The other way is for the program to control the spacing and send one character at a time to the X server. The program must then know about the character widths for the font in use. In this method, the program has no knowledge that the receiver may be using a different font with different spacings, and characters may overlap or be widely spaced for their size. If the first method is used, the receiver's X server will correctly space the characters for the font in use, but the positions of characters within the window will be incorrect.

If both methods are used, as they are in terminal emulators, the receiver is faced with a messy mixture of incorrect spacings and characters in incorrect positions.

Another important factor with respect to spacing is whether a font is proportional or monospaced. Proportional fonts use different widths for different characters, while a monospaced font uses the same space for all characters. In most cases, it is better to match a monospaced font to another monospaced font, even if they are of slightly different size, than to match it with a proportional font of the same size. Likewise, matching a proportional font with another proportional font tends to give the best appearance, and if the typefaces are similar, the chance is high that specific character widths will be similar.

The issue of character sets is an easy one for an English speaking person to ignore. Almost all fonts have identical characters in the range of characters most often used (the ASCII characters) numbered 32 to 127. The differences lie mainly in the upper range, the characters numbered 128 to 255 and beyond. This is where characters with special accents, used in most European languages, are found. If accented characters are used, the difference in character sets is very important.

This issue becomes more important when character sets for pictograms or entirely different alphabets are used. If the character set does not match, the receiver is given meaningless garbage. But what constitutes a match? Does the name of the character set have to match exactly, or are there more or less equivalent character sets that have different names? These issues have not been addressed in the current font matching algorithm. For HP SharedX to work acceptably with these types of characters, it is best to have the same font on sender and receiver machines.

Performance

The performance of HP SharedX depends on the characteristics of the network connection, the performance of the workstations involved, the application being shared, and the operations performed with the application. There are three networking factors that affect performance: network speed or bandwidth, line delays, and network load. Increased load is roughly equivalent to decreased bandwidth. Performance increases directly with network bandwidth, up to a limit of about 500 kbits/s for an unloaded network. Beyond that, other factors limit performance. Line delays are of greater importance when sharing over a wide area network (WAN). Screen update times increase proportionally to line delays.

Given a network with reasonable bandwidth, HP SharedX performance will be greatly affected by the performance of the computers involved in the sharing, especially the sending machine. When users have a choice of who sends and who receives, the person with the faster machine should be the sender.

Applications vary widely in how they make use of X. For example, an application receives an expose event from the X server, indicating that some portion of a window has just become visible and therefore needs redrawing. Some applications redraw the entire window, while others are more efficient and only repaint the portion that is exposed. A word processing application may update the entire window every time the user types a letter. While this scheme works acceptably when not sharing, this application is nearly unusable when shared.

If application windows can be resized, the sender can improve performance by making windows smaller. Also, users often come to recognize slow operations. If resizing can be done before starting the sharing session, there will be less demand on the network. If the application is still too slow when shared, the sender can take snapshots of it with the Whiteboard and share the Whiteboard.

Lazy allocation of resources in HP SharedX affects performance in some interesting ways. For example, the Whiteboard uses two large pixmaps for the drawing area, one for the erasable layer and one for the unerasable layer. The erasable pixmap is allocated immediately when the Whiteboard is shared. The other, however, is not used until the user does an erase operation. The user may be typing some text into the drawing and then hit the backspace key. At that point, the Whiteboard and all other X programs on the sender's display "freeze" while the unerasable pixmap is created on the receiver's X server.

When more than one receiver is involved in a sharing session, the method of connection becomes important. If there is one receiver, and the parties involved want to add a second, either the sender or the receiver can share to the new person. The first receiver sharing to the second receiver is called daisychaining. The sender sharing to more than one receiver is called fanning out. With the right combination of daisychaining and fanning out, an application can be shared to a large number of people.

Determining an optimal configuration for one-to-many sharing is more complex when the computers vary in performance or when the network links between the parties vary. Some general rules for optimizing performance are:

- The fastest machines should be daisychaining, and slow machines should all be leaves in the sharing tree.
- When some of the receivers are connected by a LAN, while others can only be reached over a WAN, the number of WAN connections should be minimized.

For example, if a sender in Colorado is sharing a window to five receivers in New York and three receivers in California, it is most efficient for the sender to share the window to the fastest receiver in New York and the fastest receiver in California and have those receivers share to others on the same local network.

Conclusions

The implementation of HP SharedX presented several problems. First and foremost, since this type of technology is new, no foundation of past experience was available to build upon, especially when it came to designing the HP SharedX user interface. The user interface design challenges were solved by applying human factors design techniques, including user task analysis and human factors testing. The HP SharedX extension presented a totally different challenge, including handling input from multiple sources in a sane manner and applying X protocol customized for a machine of one type and mapping it to machines of very different types. The HP SharedX extension challenges were met by understanding the nature of the X window system and

gracefully degrading the display image when receiver display resources do not match those of the sender's display. Although there are no perfect answers to any of these challenges, the value of HP SharedX far outweighs its limitations.

Acknowledgments

Many individuals have contributed to HP SharedX, from its conception at HP Labs to its becoming a product. Steve Lowder and Phil Gust played a major role in initiating the HP venture into collaborative tools and helped with the initial HP SharedX prototype. Thanks also to Nancy Kedzerski and Dan Flickinger; HP SharedX would have never become a product without their support. Special thanks to Joe Gersch, whose vision carried HP SharedX from an HP Labs prototype to a commercial product. Thanks to Randy Branson in product marketing for his valuable input and support. Thanks also to the design and implementation team of John Byrnes, Fred Sprague, Rich Wildman, Susan Frontczak, Perry Wells, Jeff Wood, Ken Burgess, Steve Wolf, and Mary Jones. Thanks to Bob Benusa for information on HP SharedX performance. Finally, thanks to Jan Ryles for her insight into users and tasks.

References

1. F. E. Hall and J. B. Byers, "X: A Window System Standard for Distributed Computing Environments," *Hewlett-Packard Journal,* Vol. 39, no. 5, October 1988, pp. 46-50.

2. K. H. Bronstein, D.J. Sweetser, and W.R. Yoder, "System Design for Compatibility of a High-Performance Graphics Library and The X Window System," *Hewlett-Packard Journal,* Vol. 40, no. 6, December 1989, pp. 6-12.

3. J. C. Lauwers, "Collaboration Transparency in Desktop Teleconferencing Environments," *Computer Systems Laboratory Technical Report CSL-TR-90-435,* Stanford University, July 1990.

4. D. Garfinkel and R.J. Branson, " A Comparison of Application Sharing Architectures in the X Environment," *Proceedings of Xhibition '91,* June 1991.

5. J. R. Boyton, et al, "Sharing Access to Display Resources in the Starbase/X11 Merge System," *Hewlett-Packard Journal,* Vol. 40, no. 6, December 1989, pp. 22-23.