What You See Is What I Want: Experiences With The Virtual X Shared Window System

Ian Smith and Elizabeth Mynatt
The Multimedia Computing Group
Software Research Center
Georgia Tech College of Computing
Atlanta Georgia, 30332-0280
iansmith@cc.gatech.edu
beth@cc.gatech.edu

ABSTRACT

There has been considerable interest in the problems associated with real-time, multi-user interfaces for computer supported collaborative work. This paper describes our experiences in designing and implementing Virtual X, a real-time window sharing system based on the X network protocol. The Virtual X environment provides a mechanism for the collaborative use of unmodified X applications, even though these applications were originally designed to have only one user. We discuss how our approach provides a basis for future research into the human factors of shared window systems. We explore the issues that arise in implenting this system in a heterogenous environment. Finally, we explain our support for collaboration-aware software and the future of Virtual X.

KEYWORDS

Shared windows, multi-user interfaces, computer supported collaborative work, real-time groupware

INTRODUCTION

Before we begin to discuss the design and implementation of the Virtual X shared window system, we must define the term "shared window system." Informally, we will define a shared window system to be a special interface to existing applications which permits single-user applications to be used by multiple persons simultaneously. This new interface is employed without the application being aware of the collaboration.

A shared window system may also have support for applications that are designed for multi-user interaction so these applications can take advantage of the existance of this environment. In our shared window system only one user at a time may provide input to applications which are being run cooperatively, and this user is referred to as the user with "the floor." The shared window system determines exactly which user has the floor, and how/when the floor should change to a different user. Speaking broadly, the shared window system is only the facilitator of the multi-user interaction; it is generally passive during user activity.

More formally, a shared window system can be broken into four components [7]:

- Display Management: A shared window system must stay informed of the state of all displays and input devices in the current session. From this information, the shared window system determines how to display application output and how to respond to user input.
- Floor Control: The shared window system must stay aware of who the "active" user is, and enforce this determination when necessary.
- Participant Management: If users are allowed to enter and leave the system dynamically, policies must be in place to deal with these situations; this is the role of participant management. If dynamic conferences are not permitted, actions must be taken to insure that users do not leave the conference while it is in progress.
- Inter-user Communication: For a shared window system to be effective, some means must be avail-

able for users to communicate with each other. The type of communication system provided by the shared window system varies from system to system.

In this paper we will describe the all of the areas of this model in general terms, and we will be returning to this model of a shared window system in our discussion of the design and implementation of Virtual X.

AN EXAMPLE OF THE USE OF VIRTUAL X

Multi-user interfaces have many uses, and to motivate our discussion of Virtual X, it may be useful to give an example of Virtual X in use. Three people in a research center, Keith, Beth, and Ian, are putting together a document describing their center's latest project. (A snapshot of their session is shown in Figure 3.) They are using a multi-user text editor, which allows all of them to write simultaneously on their document; this is the only truly multi-user application that they are running, since it is the "collaboration-aware" application in this session. They are also using two collaboration-unaware applications, *idraw*, a WYSIWYG drawing program, and *xterm* a terminal emulator. They are using these programs to draw their figures for the paper, and to process their document with the command line interface to their typesetting program, respectively.[1]

They have decided to use a Floor Control Manager (a program that arbitrates access to the floor) that implements a "I take the floor whenever I want it" policy, since they are connected with a voice connection that allows them to arbitrate access to the floor easily. The Floor Control Manager is running on Beth's machine and is communicating with the Floor Control Clients (FCCs) on each workstation in the conference. Whenever any particpant wants to "grab" the floor, he or she invokes a function of the local FCC, which communicates his or her desire to the FCM. At it changes the floor of the session (by communicating with Virtual X), the FCM informs the inter-user communication system of this change. The communication system they are using sends the voice of one person in the session over a local ethernet to the other members of the conference. This system wants to keep the "speaker" of the communication channel the same as the user with the floor, and is using information provided by the FCM to accomplish this.

Each person in this session is running local applications, Keith is running two and Beth and Ian are running one each, and these applications are not affected by the conference taking place. Although these applications are independant of the conference, the use of shared and non-shared applications can be freely intermixed. It should be noted that the "selection-space" (means of selecting an item in one application and transferring it to another) of Virtual X is normally separate from the selection-space of non-shared applications. Thus, "cut-and-paste" operations can occur in Virtual X independent of and in parallel with similar operations by any of the unshared applications. On each user's workstation, there is a large "virtual root window" which is the background window for all applications running cooperatively. Within that root window, "What You See Is What I See" is enforced on the collaboration-unaware applications, and the collaboration-aware text editor is allowed to present any interface it wishes to the users. The root window however, may be manipulated (as a whole) in any way the individual user wishes, including being moved, resized, or iconified without effect on other conferece sites.

SHARED WINDOWS AND X

The X Window System is a device-independent and network-based window system originally developed at MIT. X provides programmers with the mechanisms for the development of many types of user interfaces. The X window system has a client-server architechture, in which the X server controls all display-oriented hardware and resources, and X clients (applications) communicate with the X server, over a network, to manipulate any of these resources. This network transparency provides excellent functional separation of the client and server. For an complete overview of the X window system, see [20]. In the following section, we will provide an overview of shared window systems, and mention specific issues that relate to window sharing in X.

Display Management

Significant previous work has addressed the technical issues of real-time window sharing under X and how to provide multi-user interfaces to existing applications. These efforts explored the first element of our model of a shared window system mentioned before, the display management system. These systems in general exploit the fact that X at its lowest level

[1] This command line interface, "the shell", runs under the permissions that correspond to the user who started the shell. The issues associated with this and similar problems are discussed fully in [2].

is simply a network protocol, and therefore network transmissions can be intercepted and then retransmitted as the shared window system sees fit.

Considerable research has focused on the method of performing the interception and retransmission mentioned above. A classical distributed systems problem is whether the shared window system should have a replicated or centralized architechture. There have been several systems based on a centralized architecture including Rapport[3], XTV[1], Shared X[8], and Shadows[18]. All of these systems centralize control of the the shared window system, and allow the shared window "psuedoserver" to connect to each X server in the session, and permit one copy of each collaboration-unaware client to connect to the shared window system (see figure 1). By contrast, in the

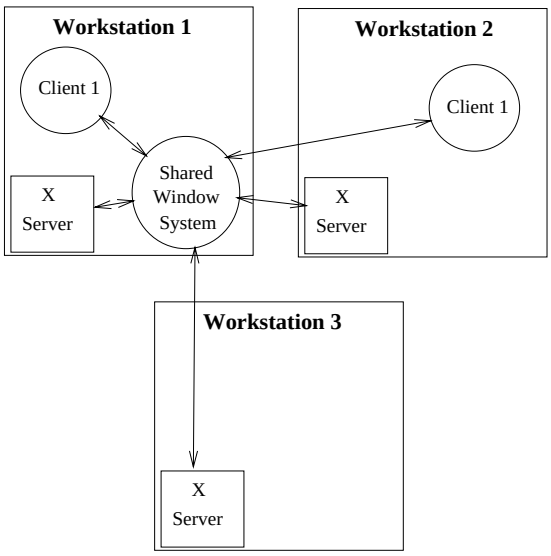

Figure 1: Centralized Model

replicated architechtures n copies of the application are run (one per worksation) and the shared window system keeps all copies of the program in similar states. Two systems of the systems based on this type of architecture are Dialgo[17] and MMConf[5].

Although these approaches are functionally similar to the user, they have radically different architectures, implementations, and problems. The centralized approach enjoys the benefit of being in complete control of the system, and therefore is guaranteed to know the states of all parts of the system. Further only one copy of each application is running, so applications that affect their external environment (such as saving a file) perform such operations only once and thus function normally. The replicated architecture distributes the computation over the workstations, and therefore does not consume network band-

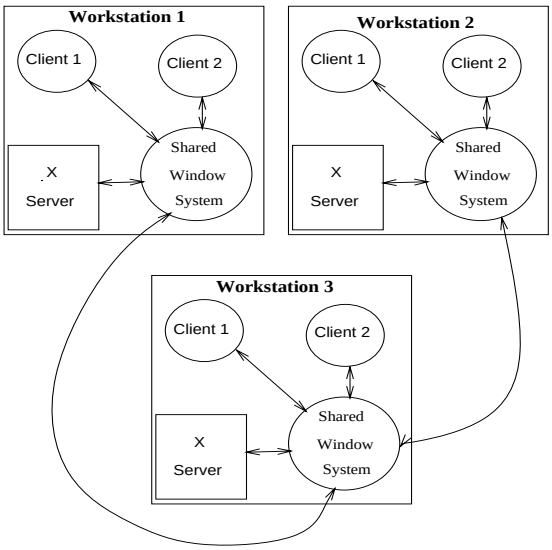

Figure 2: Replicated Model

width transmitting results, provided the each replicant application can be kept in "synch." Disscussion of the environmental, synchronization, and serialization problems associated with replicated architectures can be found in [11, 10].

Floor Control

In our system during a cooperative session only one person can be giving input to applications that are designed for a single user. (An application written specifically to take advantage of the collaborative system may allow all users to provide input simultaneously.) Exactly how the floor should change during a session is a policy decision and certainly a matter of debate.

Previous human factors research has established that many issues come into play when people work cooperatively. Some of the factors to consider when determining a control policy for collaborative work include (but are not limited to):

- The familiarity of the participants with each other
- The type of task being worked on collaboratively
- The number of persons in the group
- The bandwidth of any communication channel(s) between them particpants during the session

We feel that more work is needed in the area of floor control strategies and therefore Virtual X has been designed with the purpose of supporting future

research on this topic.[2] Several ideas have been implemented by other systems as the basis for floor control policy decisions [7]. Some of these are:

- Ring Passing: The particpant with the floor must explicitly pass it to another person.
- Request: A queue of users requesting the floor is kept, and as one user voluntarily releases the floor, the next user in the queue is given it.
- Pre-emptive: Anyone may grab the floor at anytime.
- Moderated: One user is designated to arbitrate floor-control and this user gives (and takes away) the floor in any way he wishes.
- Time slicing: Each user is given a specific amount of time with the floor, and is then preempted by another user. To whom the floor is given next is another floor control issue, and therefore this method is usually used in conjunction with some other approach.

Referring back to our example of the group constructing a document, the users in that session were well acquianted, small in number, and had a voice channel for communicating their ideas to each other. They felt a pre-emptive floor control management strategy would be best for them. In contrast, a teacher instructing 40 students in how to use a new piece of software might have radically different needs. In such a situation, a moderated floor control strategy might be the easiest way for the the teacher to control the situation while still permitting students to experiment with their newly acquired skills.

Particpant Management

Past work has suggested that participant management presents difficult problems, no matter what type of collaborative system is being employed.[7, 10] The main goal of participant management is to keep ongoing conferences functioning smoothly as old participants leave and new participants join the session. This usually requires insuring that the new particpant's workstation (or the shared window system managing it) can be put in a state similar to that of participants who are already in the conference. Once this is done, processing can continue in a normal fashion, without regard for the changing make-up of the conference.

[2] For an excellent overview of previous work done in performance measurements of computer supported cooperative work based on different types of groups, task, and other variables see [13].

Shared window systems based on X have an additional problem in that all "state information" about resources related to applications is kept inside the X server. (The "resources" of an X server are data structures associated with the physical display of output, such as windows, cursors, fonts, etc.) Thus when client ouput requests are multiplexed to each server, all resources on all servers must be in consistant states. When a new server is added to a session, its resources will not be in a state consistant with other servers. This presents a problem since it is unlikely that collaboration-unaware clients will be capable of recreating all previous states required to get the new server's resources synchronized with servers already in the conference. Previous systems which supported dynamic conferences have depended on journalling to bring new particpants up to date by simply replaying all previous interactions in the session.

Inter-user Communication System

Several types of media have been used in the past as the basis for interaction between users running a multi-user interface, including audio, video, text, telepointers[22], drawing surfaces[4, 12], and combinations of these[3, 16, 23]. The amount of communication support given by the multi-user interface can vary greatly, and this support certainly drastically affects how effective the system is in practice. However, the amount of information the communication channel provides the users with is directly related to the amount of bandwidth consumed in the transport of the media. For example, video is an excellent way for users to communicate large amounts of useful information while employing some type of multi-user software. However, video also puts a significant load on the underlying transport agents of a communication channel and frequently users do not have the bandwidth capabilities to support this type of media. In a system requiring the use of video, special hardware must be purchased to support the multi-user interaction, or the interface must be scrapped. It seems clear also that the audience of a shared window system is somewhat determined by the communication system of the shared window system, since potential users may be unable to support the communication media, or may not wish to use the shared window system unless it supports certain types of media.

THE DESIGN OF VIRTUAL X

Virtual X was designed along the lines of the formal model of a shared window system we described before.

The Virtual X system proper handles the display and participant management duties and delegates responsibility for the floor control and inter-user communication to external programs. Referring back to our example of Keith, Beth, and Ian preparing a document, their session as depicted in Figure 3 contains all the major components of the Virtual X environment: Virtual X itself, a Floor Control Manager, Floor Control Clients, an inter-user communication system, and both types of cooperative applications, collaboration-aware and collaboration-unaware.

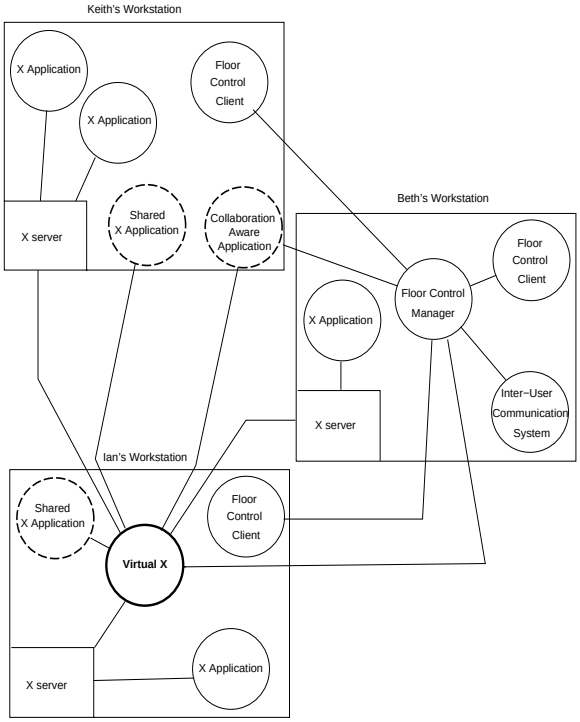

Figure 3: The Virtual X Environment

The following were considered objectives in the design of Virtual X:

1. Provide a "virtual workstation" on which unmodified X applications can be run. This "workstation" in reality has its display shown on several physical workstations, and any of the physical workstations can provide input to the virtual workstation.

2. Provide a foundation for human factors research into multi-user interfaces.

3. Provide support for applications that wish to be aware of collaboration, and allow these applications to effectively utilize Virtual X's existing facilities.

4. Allow other collaborative tools and conferencing systems to coexist with Virtual X and, when possible, utilize information about the state of Virtual X conferences.

5. Provide all of the above in a heterogenous environment of workstations.

The display management system is the first design choice that must be made in designing any shared window system. We examined both the centralized and replicated architectures and chose to implement our system with the centralized approach for three main reasons:

1. Reliability. We intended Virtual X to be used by normal X users on an everyday basis. Therefore, we felt that a more robust system was required for everyday use.

2. Heterogeneity. Implicit in the replicated architechture is the assumption that all users have a copy of the software and their machine can run it. In a heterogenous environment, this assumption may not hold.

3. Data Collection. One of the goals of this project was to support research into the human factors of multi-user interfaces. A centralized architecture leads to much easier data collection and analysis.

As we examined the issue of floor control, we felt that since no one floor control method is appropriate in all situations, Virtual X should not make any policy decision on this issue. Virtual X instead designates a special program, called the "Floor Control Manager" (FCM), to make this policy decision and allows the FCM program to make such policy decisions in any way it desires. We assume that the users in a Virtual X session will select an FCM that is appropriate to the nature of their collaborative work. This separation also allows the shared window system to avoid the extra overhead associated with managing the floor and to support other programs related to floor control. Further, by permitting the FCM to control all access to the floor, floor control data collection can be easily centralized in the FCM, and experiments can be conducted on different floor control stratagies without modification to Virtual X itself.

The Virtual X system allows users to add and leave conferences as they wish, provided at least one user is always in the conference. Above we detailed the issues associated with this problem, and since we allow users to join sessions in progress, we must provide mechanisms (transparent to the application) for bringing the new user's workstation "up to date."

Previous shared window systems, in general, have used some type of journaling to keep all previous states of collaboration-unaware clients stored inside the shared window system for later retrieval. We felt that such solutions were unwieldy, and so we introduce a new method for handling this problem. Our solution is to provide a means for the new user's workstation to be brought into the same state as the other users incrementally. This is done by synchronizing resources on the new server only as they are used by currently running client applications.

The inter-user communication system for Virtual X has also been moved outside the shared window system itself. We felt that it was not the job of the shared window system itself to determine what types of communication should be used by collaborating users. We did feel that it was important, however, for programs that were providing this communication to have the capability to determine the status of the conference. For this reason, the Floor Control Manager Protocol includes the capability for external programs to query the FCM to determine status information about the session, especially regarding the user with the floor.

In our document preparation example above, we mentioned that Keith, Beth, and Ian were using a multi-user editor. Such an editor would be aware that it was being run in a collaborative environment, and would be able to handle input from several sources simultaneously. It also may present different displays to different workstations in the conference. Such a system would obviously require a radically new approach to the design of its interface and we are attempting to allow Virtual X to support applications of this type. Our design supports these applications by permitting them to interact with Virtual X in way very similar to the way they interact with normal X servers. The only difference being that extra information is exchanged so that both Virtual X and the multi-user application can stay aware of which server is the source of input and the target of the application's graphic displays.

At the present time, it is not clear what types of floor control stratagies are best suited to different types of multi-user interaction. Since the Floor Control Manager is an excellent place for both the collection of data about floor control policies and exploring new floor control policies, we are designing experiments to explore the possibilities for different floor management policies. Some of the questions that we find interesting are: At what number of particpants are different floor control policies effective or disfunctional? Can computer experience, familiarity, or some other quantifiable variable be used to predict which types of floor control are best for a given group of users? Can specific types of applications be pinpointed that operate well or poorly under given types of floor control algorithms? Is it possible to design and implement a floor control policy that dynamically adapts to the needs of the work group?

In summary, Virtual X provides a support layer for the use of both single-user and multi-user software in a cooperative environment. The Virtual X system is based around a centralized display management system, and allows external programs to control policies related to floor control and inter-user communication. Virtual X provides mechanisms to transparently allow users to join or leave sessions in progress, and does so without the need for expensive journalling. Virtual X permits external programs to synchronize to the state of Virtual X conferences by defining a method for Floor Control Mangers to respond to queries about the state of the conference. This allows inter-user communication systems and collaboration-aware software, for example, to coordinate their activities with Virtual X. The Virtual X concept of the Floor Control Manager also provides an excellent means for conducting experiments into the human factors of multi-user interfaces.

IMPLEMENTATION ISSUES

The technical issues associated with multiplexing and demultiplexing X protocol packets to support a shared window system have been detailed by others [1, 18]. Since they have addressed these issues, we will focus our attention on three other areas of interest:

- Heterogeneity: How to support a shared window system when the workstation hardware of the collaborating users is different.
- Participant Management: An approach to dynamic conferences.
- Collaboration Awareness: Support for clients that want to use Virtual X as a platform for true multi-user interfaces.

Heterogeneity

To accomplish the goal of running a heterogenous shared window system, one must first define a display model for interpreting the sematics of client requests, then translate the requests based on the model to the physical hardware present on a user's workstation. X defines certain abstractions of workstation display hardware called "visual classes" and these inform client applications of the type of display hardware present on a given workstation. A visual class

encompasses the physical hardware configuration of a workstation's display so that applications running on different types of display hardware will know what types of ouput may be rendered on the screen. We felt that the X visual class abstraction was a suitable model for our shared window system.

At the time a client connects to a X server (or, in our case, the shared window system), the client requests the visual class of the workstation. Virtual X responds to such a request with a visual class of "Static Gray", which corresponds to a workstation with monochrome-only display hardware. This informs the client application that only black and white output is supported by the shared window system. Choosing to support only monochrome shared applications has been deemed unacceptable by others [6]; we felt, however, that in supporting our first implementation of a *heterogenous* shared window system we should choose an abstraction that could be readily supported by all types of workstation hardware. In the future we will support applications that utilize color in a heterogenous environment[21].

In this section, we will address two distinct areas of heterogeniety support that Virtual X provides. We will discuss our support for running a conference on heterogenous output hardware simultaneously. This allows applications to be run on any combination of color and monochrome workstations simultaneously. We will also detail our work to support conferences in which all the users in the session do not have similar input hardware. This ability is needed so that applications, which are statically configured, can be used in the changing input environment of Virtual X.

Depth Faults Let us motivate our discussion of heterogeniety with an example of an issue that occurs when running real applications. A client running under the Virtual X environment, after querying for the visual class of the "workstation", determines that the display hardware it is running on is monochrome-only. At this point, it may allocate resources based on this fact that will function correctly on a monochrome server. However, these resources may not be correct for other types of servers, and these servers may be part of the Virtual X conference. It is possible for the client application to manipulate these resources in a way correct for monochrome servers, that is *not* correct for other servers. If the session is to not end due to errors on non-monochrome servers, Virtual X must take action to insure that manipulations are carried out in a way that is permissable on all servers in the conference. To achieve this end, we introduce the concept of a "depth fault."

Before we discuss depth faults, some background is necessary on the X abstractions related to this issue. A property associated with every workstation that runs X Windows is its "depth." The depth of a workstation's display corresponds roughly to the number of bits of color information per pixel of display resolution. (For example, a monochrome workstation has depth 1.) Some systems allow emulation of depths other than the one that corresponds to the physical hardware. However in general we can only assume that a workstation supports one depth.

Two other relevant X abstractions, the "pixmap" and the "window" are important to window sharing heterogeneity. A window is an area of the physical screen of the workstation. This area has a depth associated with it, and this depth is usually the same as the workstation's display hardware. A pixmap is a region of "off-screen" memory that can be drawn into in the same way that an "on-screen" window can. Pixmaps can also be copied onto windows. This is usually done frequently, since it allows graphics to be drawn off-screen and copied onto the screen as one unit. There is a depth associated with every pixmap as well, and here lies a problem. All workstations running X support at least two depths for pixmaps: depth 1 and the depth of the workstation's display hardware. In the case of a monochrome workstation these are the same. A depth one pixmap is frequently referred to as a "bitmap."

As we mentioned in our example, clients may manipulate monochrome resources in a way that is permissable on a monochrome server that is not allowed on other servers. (Generally, the depths of resources used together in an operation must match, and all resources on a monochrome X server are depth 1.) Such a manipulation is, for example, copying a bitmap to a window; this is permitted on a monochrome server since the resources are the same depth, but it is not allowed on color servers, since the depths are not matched. If we blindly multiplex client requests for the use pixmaps, bitmaps, and windows, fatal errors will result.

Our solution to this problem is the depth fault, which is analagous to a page fault in a virtual memory system. In general terms, a depth fault occurs when a client attempts to use a pixmap in some operation, and on different servers currently in the conference the permissable depth for that pixmap is different. When a depth fault is generated, Virtual X will synthesize new requests to servers in the session, and these requests are placed in the request stream at a point directly in front of the client request causing the depth fault. Virtual X's requests take corrective action so that when the client request does arrive at the server, it will not generate errors.

Event Simulators It is likely in an environment run-

ning X that not all workstations that could be in an Virtual X session will have the same input hardware. It is clear that the Virtual X environment should provide support for collaborative work with applications that expect hardware that not all workstations in the session have. For example, some applications depend the user having a three button mouse, and some users in a Virtual X session may not have such hardware. Since the X protocol provides a means for simulating events for use by other client programs, the Virtual X environment provides several programs to synthesize events that a user's workstation cannot physically produce. Programs that synthesize these events transmit them to client applications that are being run cooperatively, and the synthetic event performs the same action as the physical event would. Programs that provide this service are generally referred to as "event simulators."

Participant Management

Previous work in shared window systems has suggested that journaling and possibly even dependance analysis of the resource usage of clients is necessary to dynamically add users to shared sessions [9, 18]. Others have suggested that the X protocol be extended to support the ability to dynamically add users [1]. We feel these approaches are limited, and we propose a new alternative to dynamic conferences.

To show the correctness of our approach, we need to introduce four facts about the X protocol that we are exploiting. (For a complete description version 11 of the X protocol see [19] and [15].)

1. Since the values of resources are kept by X servers, X applications have no way to access resources without generating a network request to a server. This insures that clients will inform the shared window system of all changes to a resource's value.

2. There is no facility in the X protocol for referencing past values of a single resource. (The value of a resource may depend on the value of other resources, however.) This means that the shared window system need only be concerned with the current value of a given resource, not its past states.

3. X resources have no circular or self-referential dependencies. This insures that there is a simple ordering on the definition of resources.

Given these three facts, we introduce the notion of the "resource fault", which is a generalization of the depth fault introduced earlier in this document. A resource fault is generated when a client application attempts to use a resource that currently does not have a defined value for a particular server. In a way similar to the depth fault, when a resource fault is detected, Virtual X takes corrective action and initializes the resource to a correct value. This value may be obtained from Virtual X itself or from another server in the session, depending on the type of resource.

When a resource fault is generated, Virtual X determines any dependancies for the value of the resource and generates resource faults for the depended-on resources if necessary. This recursive technique effectively creates a tree-structured dependance graph, and since the graph cannot have cycles (3), this process is guaranteed to terminate. Futher, this process will always generate correct results since clients must generate requests to change the display (1), only the current value of the resource is needed to complete any request (2), and Virtual X will always have the current resource value (1).

The resource fault approach does not require extensions to the X protocol and also has several advantages over journalling. First, the overhead of journalling is avoided, since only the current values of resources are kept. Second, significant amounts of recordkeeping are distributed to X servers in the session, which have to keep the contents of the resources anyway. Third, the dynamic nature of resource faults allows the cost of adding a new user to a session to be amortized over many requests, rather than forcing the session to wait significant amounts of time for the new user's workstation to be brought up to date.

Collaboration Awareness

Throughout this section, we have discussed our attempts to retrofit a multi-user interface onto existing single-user applications. We will now detail our work to support true multi-user interfaces, and how our efforts provide a suitable platform for the development of such software.

In attempting to provide a basis for multi-user interfaces, we faced two major decisions. First, should collaboration-aware programs use existing protocols in some new way for transmission of user interface data, or should new protocols be invented? Second, should the multi-user primitives be built into the shared window system itself, or should it be left to the client to determine how to effectively use a multi-user interface?

With regards to the first question, we felt that the existing X protocol was a suitable method for transmitting user interface data. We also felt the

abundance of tools and reference material for the development of X applications made this a reasonable choice, as any new protocol invented would need a large amount of supporting material to be useful. For these reasons, we chose to allow client applications to communicate a slightly "addended" form of the X protocol with Virtual X to support multi-user interactions.

Virtual X allows clients to send X network packets to a different network address if these packets contain the additional, multi-user information. This additional information consists of simply one integer, which is the target workstation of the X protocol packet. (The indexes of each workstation in the conference can be obtained from the Floor Control Manager of the conference.) Upon receipt of this packet, Virtual X transmits the normal portion of the packet to the X server specified as the target. This simple scheme allows clients to "do their own multicasting" of X protocol requests.

As to the input portion of the X protocol, when Virtual X receives any event from any server in the session that is destined for the collaboration-aware program, this event is passed on to this program. This transmission bypasses the floor control mechanisms, as it is assumed that the collaboration aware application will enforce its own access policy. Note that this behavior effectively multiplexes several client connections over one connection between Virtual X and the collaboration-aware client.

To allow clients to transmit and receive this addended form of the X protocol requires only simple modifications to the underlying transport layer of the X client libraries. The results is that programmers do not have to learn entire new libraries of routines to develop the interface portion of multi-user software. However, this decision to remain bound to the X protocol has a somewhat constraining effect on our second issue of how to support multi-user primitives in Virtual X.

Since we have chosen to not create new protocols for multi-user interactions, we are considerably limited in how we address the issue of the supporting multi-user interactions by Virtual X itself. Clients that have multi-user interfaces will need facilities such as locking and serialization, [14] but without modification to the X protocol, Virtual X cannot provide these. Therefore, under our system, clients have the responsibility for the management of the multi-user interaction. It is hoped that libraries can written to aid programmers in dealing with the concurrency issues faced by multi-user software.

STATUS

At the present time, the Virtual X system supports many unmodified X applications, and we are currently working towards supporting the entire core X release. Virtual X has been implemented on Sun Sparcstations running SunOS 4.1, and has been tested on monochrome and color Suns, color Hewlett-Packard workstations, and greyscale NeXT computers. A prototype implentation of Floor Control Managers and Floor Control Clients is complete, and complete implementation is expected soon. The inter-user communication system is operational, but as yet has not been coupled to the Floor Control Manager to provide synchronization with the Virtual X session. At the time of this writing, work on collaboration-aware programs and event simulators is just beginning.

CONCLUSIONS AND FUTURE RESEARCH ISSUES

We feel that the Virtual X system provides users with an excellent environment for collaboration and multi-user software interaction. Further, we state that our approaches to heterogenous shared window systems and dynamic conferences are suitable for real-world computer supported collaborative work. In the future, we plan to extend the implementation of Virtual X to support color applications and would like to experiment with the possibilities of replicating instances of Virtual X to provide better performance over wide area networks. We are also designing human factors experiments that should provide insights into what types of floor control paradigms are appropriate for groups of different sizes and tasks.

References

[1] H. Abdel-Wahab and M. Feik. Xtv: A framework for sharing x window clients in remote synchronous collaboration. In *Proceedings of IEEE Conference on Communications Software*, pages 159–167. IEEE Communications Society, 1991.

[2] H. Abdel-Wahab, S. Guan, and J. Nievergelt. Shared workspaces for group collaboration: An experiment using internet and unix interprocess communications. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 1989.

[3] S. Ahuja, R. Ensor, and D. N. Horn. The rapport multimedia conferencing system. In *Proceedings*

of Conference on Office Information Systems, pages 1–8, 1988.

[4] S. Bly and S. Minneman. Commune: A shared drawing surface. In *Proceedings of the ACM SIG for Office Information Processing*, pages 184–192. SIGOIP, 1990.

[5] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson. Mmconf: An infrastructure for building shared multimedia applications. Technical report, Bolt Beranek and Newman Inc, 1990.

[6] D. Garfinkel, P. Gust, M. Lemon, and S. Lowder. *The Shared X Multi-User Interface User's Guide, Version 2.0.* Hewlett Packard Laboratories, 1989.

[7] S. Greenberg. Sharing views and interactions with single-user applications. In *Proceedings of the ACM SIG for Office Information Processing*, pages 227–237. SIGOIP, 1990.

[8] P. Gust. Sharedx: X in a distributed group work environment. Presentated at the 2nd Annual X Technical Conference, 1988.

[9] Keith A. Lantz. An expriment in integrate multimedia conferening. In *Proceedings of the ACM SIG for Office Information Processing*, 1986.

[10] J. Lauwers, T. Joseph, K. Lantz, and A. Romanow. Replicated architechtures for shared window systems: A critique. In *Proceedings of the ACM SIG for Office Information and Processing*, pages 249–260. SIGOIP, 1990.

[11] J. Lauwers and K. Lantz. Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems. Technical report, Olivetti Research Center, 1990.

[12] J. Lee. Xsketch: a multi-user sketching tool for x11. In *Proceedings of the ACM SIG for Office Information Processing*, pages 169–173, 1990.

[13] J. McGrath and A. Hollingshead. Interaction and performance in computer-assisted work groups. Presentated at the conference on Team Decision Making in Organizations, 1991.

[14] R. E. Newman-Wolfe, C. L. Ramirez, H. Pelimuhandiram, M. Montes, M. Webb, and D. L. Wilson. A brief overview of the dcs distributed conferencing system. In *Proceedings Summer 1991 Usenix Technical Conference*, 1991.

[15] A. Nye, editor. *X Protocol Reference Manual*, volume 0. O'Reilly and Associates, 1990.

[16] M. Ohkubo and H. Ishii. Design and implementation of a shared workspace by integrating individual workspace. Technical report, NTT Human Interface Laboratories, 1989.

[17] Olivetti Research Center. *User's Guide to Dialogo Shared Window System Release 1.0.*

[18] J. Patterson. The good, the bad, and the ugly of window sharing in x. In *Proceedings of the 4th Annual X Technical Conference*, 1990.

[19] R. Schiefler. X window system protocol. MIT X Consortium Standard, 1988.

[20] R. Schiefler and J. Gettys. The x window system. *Association for Computing Machinery Transactions on Graphics*, 1986.

[21] I. Smith. Color and heterogeniety under x11, 1991. In Preparation.

[22] M. Stefik, G. Foster, D.G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1):32–47, 1987.

[23] K. Watabe, S. Sakata, K. Maneo, H. Fukuoka, and T. Ohmori. Distributed multiparty desktop conferencing system: Mermaid. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pages 27–38, 1990.