Accommodating Latecomers in Shared Window Systems

Goopeel Chung, Kevin Jeffay
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175
{chungg,jeffay}@cs.unc.edu

Hussein Abdel-Wahab
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162
wahab@cs.odu.edu

Abstract

A viable concurrent engineering environment should support synchronous collaboration among distributed users. Shared window systems provide this support by allowing multiple users to create workstation-based conferences wherein conferees simultaneously view and manipulate shared files using existing, familiar applications. To be effective, a shared window system must be flexible and robust: one should be able to dynamically create conferences around applications that are already in use, new participants should be able to join conferences already in progress, and participants who become disconnected from a conference should be able to rejoin. We describe an efficient solution to the problem of making shared window conferencing flexible and robust. This general problem is phrased in terms of allowing a latecomer to join a conference. The essence of our approach is a protocol filtering and archival process that is tapped into the communication channel between a conference application and a user's display manager. This filter process interprets, in real-time, the application/display manager protocol stream and records the minimal information necessary to generate an exact replica of the application's interface on a remote display when a latecomer joins this conference. This solution has been implemented in XTV: a public domain shared window system based on the X Window System.

Keywords: Concurrent engineering, CSCW, shared window systems, workstation-based conferencing.

In *IEEE Computer*, Vol. 26, No. 1, (January 1993) pp. 72-74.

Accommodating Latecomers in Shared Window Systems

Goopeel Chung, Kevin Jeffay
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175
{chungg,jeffay}@cs.unc.edu

Hussein Abdel-Wahab
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162
wahab@cs.odu.edu

A central component of a concurrent-engineering environment is a facility for synchronous collaboration such as a distributed, workstation-based conferencing facility. The ultimate goal of such a conferencing facility is to allow geographically separated engineers to simultaneously view and manipulate shared images, documents, or programs while they communicate via audio and possibly video links.

There are two basic approaches to supporting such conferences. The first involves the development of so-called "collaboration aware" applications — special purpose applications such as editors and electronic whiteboards, that directly support multiple simultaneously active users. Alternatively, one can leverage the large base of existing single-user applications by augmenting a window system, so that it supports the sharing of application interfaces (*e.g.*, windows) across multiple distributed users. The resulting window system is typically called a "shared window system."

We have developed, and placed in the public domain, a shared window system, called XTV (X Teleconferencing and Viewing [1]). XTV is based on the X Window System. It lets users create a conference around one or more arbitrary X applications. Conferees have the same view of shared applications and by following a simple floor passing protocol, they can control the shared applications.

To be effective, conferences must be flexible and robust. Users should be able to create conferences dynamically around applications that are already in use, and new participants should be able to join conferences already in progress. Moreover, since

conferences are distributed, a conferencing system should be fault tolerant. Participants who become disconnected from a conference should be able to rejoin — transparently to other conferees.

Here we describe an efficient solution to the problem of making shared window conferencing flexible and robust. We phrase this general problem in specific terms of allowing a latecomer to join a conference. (In our view, a user initiating an application is logically initiating a conference with one conferee.) The fundamental problems in accommodating latecomers are (1) to establish communication paths from each application in the conference to the latecomer's display manager, and (2) to alter the state of the latecomer's display manager so that it both displays the current image of a shared application and correctly processes future commands from the shared application.

We have reported our solution to the first problem elsewhere [1]. The essence of our approach to the second problem is a protocol-filtering and -archival process that taps into the communication channel between a conference application and the initial user's display manager. This filter process interprets, in real-time, the application/display manager protocol stream and records the minimal information necessary to generate an exact replica of the application's interface on a remote display when a latecomer joins this conference.

This approach is analogous to log-based recovery schemes in database management systems. We based our protocol-filtering and -archival scheme on a careful analysis of the application/window system communication protocol. In essence, the scheme allows a shared window system to keep minimal length logs. The infrastructure we have developed for protocol filtering and recording is part of a more comprehensive environment for concurrent engineering and group work [3].

Shared Window Systems

Figure 1 shows the architecture of a typical shared window system. It illustrates a conference involving one application and three conferees on three separate machines. A

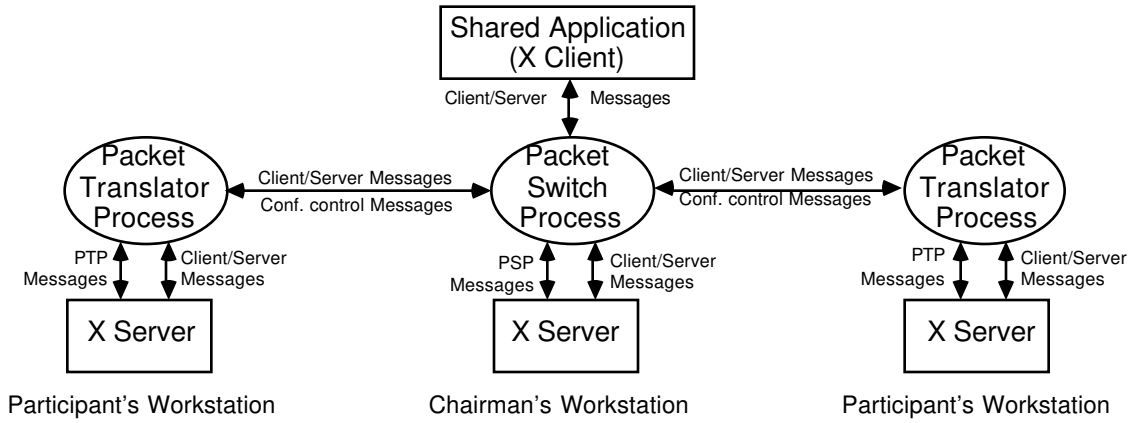

Figure 1: Architecture of a shared X Window System.

central conference agent exports an X-server interface to the shared application and an X-client interface to the X server, and a per-remote-conferee agent that exports an X-client interface to each conferee's X server.

The central conference agent, called the packet-switch process in XTV, has three functions. First, it distributes network packets to the X servers of remote conferees. These packets contain messages sent from the shared application to the initial user's X server. Second, the packet-switch process provides an input path from the X servers of remote conferees (and hence from the remote conferees themselves) to the shared application. Lastly, it logs messages sent from the shared application to the initial user's X server that have an effect on the internal state of the X server. These messages are logged for later playback to a latecomer's X server.

Messages sent from a shared application to an X server cannot be blindly redistributed to other X servers (and vice versa). Such messages must be parsed for data fields containing objects called X resource IDs, and the values of these IDs must be translated to values that other X servers can understand. The X application programming model presents six basic abstractions: window, cursor, graphics context, pixmap, colormap, and font. Instances of these abstractions are referred to as *resources* in X. Resources also have *attributes* such as color.

An X server creates resources , manipulats their attributes , and destroys these resources in response to requests by applications. Resources are referred to by unique identifiers called resource IDs. Applications choose these IDs from a range of acceptable values that are specific to an individual X server. For messages distributed by the packet-switch process to be intelligible at a remote conferee's X server, the resource IDs must be translated into a subrange assigned by that X server.

In XTV, each remote conferee has a local conference agent, called the packet-translator process, that is responsible for translating resource IDs.

Accommodating Latecomers

When a latecomer joins a conference, the latecomer's X server does not have any of the resources created by the shared applications in the conference. Hence, the packet-switch process must create these resources — with their current attribute values — on the latecomer's X server before the server can interpret messages forwarded to it by the packet-switch process.

The problem is to identify the minimal set of resources that must be created. For example, an application may create resources, such as a window, for a temporary object such as a pop-up menu or dialog box. The application destroys these resources when a user completes the interaction with the object. The resources corresponding to the window should not be created on the latecomer's X server if the window is no longer part of the application's interface. For this reason, a simple approach to accommodating latecomers such as archiving all messages sent by a shared application to its X server is undesirable. The storage required to record all messages becomes prohibitive as the conference progresses and the time required for a latecomer to join a conference will be a function of the conference's duration.

Our approach selectively catalogs changes an X client can make to the internal state of an X server. Modifications to the server state are recorded by maintaining a log

of the messages used to create or modify the attributes of resources currently in use by the shared application. For attributes we need only record the most recent values. When a late comer joins the conference, the resources currently in use by the shared application are created on the latecomer's server by generating and sending a series of X-protocol messages to the new server. The messages are based on the contents of the packet switch's message log. Optimizations on this basic scheme are reported in Chung et al. [2].

Attributes of resources can themselves be resources; hence, dependency relations can exist between resources. These relations constrain the process of logging resource creation messages. When a shared application deletes a resource on an X server, the packet-switch process can not delete its representation of the resource if a second resource depends on its presence.

We represent dependency relations among resources with a directed graph: the nodes represent resources and the edges represent \ dependencies between pairs of nodes. We run a mark-and-sweep-style garbage-collection algorithm on the dependency graph when resources are deleted, deleting resources only when no other resources depend on their existence.

Performance

Two performance measures show XTV's capability to accommodate latecomers. These are the time and space required to maintain the per-shared-application resource log. These performance measures are largely a function of the extent to which a shared application creates resources. The following assumes a shared interactive drawing application — an application that extensively creates and manipulates resources.

The cost of recording shared-application resource information accounts for approximately 6 to 9 percent of the overhead in the packet-switch process for a conference with one conferee. For a conference with four conferees sharing a single

application, the packet distribution costs dominate and the overhead of maintaining resource information drops to approximately 2 percent of the execution time in the packet-switch process.

The resource log is kept in memory. The space required for the log will be a function of both the number of resources created by the application and the duration of the conference. The space required such applications as terminal emulators, which create few resources after drawing their initial interface, is between 3,000 and 5,000 bytes and grows at a near-zero rate as the conference progresses. The space required for the interactive drawing program after three minutes of use (in which time it generates over 300 Kbytes of X-protocol traffic to the packet-switch process) is approximately 50 Kbytes.

The time required for a latecomer to join a conference is a function of the length of the resource logs for the shared applications in the conference. For applications that generate long logs, the time required to join a conference is dominated by the performance of the latecomer's X server.

Summary

A viable concurrent-engineering environment should support synchronous collaboration among distributed users. Shared window systems provide this support by allowing multiple users to create workstation-based conferences wherein conferees simultaneously view and manipulate shared files using existing, familiar applications.

To be effective, a shared window system must solve the problem of allowing a latecomer to join a conference already in progress. We have constructed an X-based shared window system called XTV that accommodates latecomers by monitoring the communication channel between a conference application and the initial user's display manager and recording the X resources created by a shared application. XTV creates

copies of these resources on a latecomer's X server when he or she joins the conference. The recording scheme used in XTV keeps minimal length logs.

In practice, XTV's support for latecomers adds negligible overhead to its execution. The technology we have developed for accommodating latecomers has also been employed in a larger system for supporting concurrent engineering and group work [3].

References

[1] Abdel-Wahab, H., Feit, M. A. *XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration*, Proc., IEEE Conf. on Communications Software: Communications for Distributed Applications & Systems, Chapel Hill, NC, (April 1991), pp. 159-167.

[2] Chung, G., Jeffay, K., & Abdel-Wahab, H. *Dynamic Participation in Computer-based Conferencing Systems,* Computer Communications (to appear, 1993).

[3] Jeffay, K., Lin, J.-K., Menges, J., Smith, F.D., Smith, J.B. *Architecture of the Artifact Based Collaboration Matrix*, CSCW '92, Proc. of the Conf. on Computer-Supported Cooperative Work, Toronto, Canada, ACM Press, November 1992, pp. 195-202.