Supporting Input Multiplexing in a Heterogeneous Environment

A. Krantz, I. Rhee, C. Breuker, S. Chodrow, and V. Sunderam
Dept. of Math & CS, Emory University, Atlanta GA 30322, USA
email: {atk, rhee, carla, sal, vss}@mathcs.emory.edu *

December 9, 1996

Abstract

For effective collaboration using a computer, it is often useful for the collaborators to share applications. Two common methods of implementing application sharing are either to share a single application among the participants or to replicate an application, one per participant and keep all copies of the application synchronized. The process of synchronizing multiple copies of an X11 application through the multiplexing of X11 input events, which we refer to as imuxing, is further complicated when the environment is heterogeneous. In this paper, we describe a partial solution for imuxing that supports a heterogeneous environment.

1 Introduction

In collaborative computing, where many participants wish to collaborate through the aid of networked computers, it is beneficial to allow all the participants to share existing graphical applications. Ideally the system should provide identical views of the application-generated images on all participants local computers while allowing all participants to share the use of the application. The client-server model implemented by X11 makes application sharing feasible through the aid of a pseduo server, a program that mimics the X11 server to the application but does not actually render images. However, there are perceived performance problems due to limited network bandwidth and latency when a single application is shared by multiplexing the program's X11 requests.

Traditionally, in the UNIX workstation environment, such multiplexing has been done by redirecting a single client's X11 request to a pseduo server that then multiplexes those requests to a number of different X11 servers[4, 1] as shown in Fig. 1. We refer to the technique of sharing a single application by multiplexing the application's X11 requests as X multiplexed or X multiplexing. Efforts to achieve the same effect through application replication and imuxing have been undertaken in [6, 2, 3] using an architecture similar to the one shown in Fig. 2. However, this approach suffers from a number of technical complications that makes it difficult to keep the programs synchronized. These complications include: asynchronous computations where the replicated applications take a different amount of time to reach the same state; the ordering of multiplexed input events where multiplexed events from different sources have to be sent in a sequence that is acceptable to the client [6]; and the processing of a stream of input events in which time of reception is a critical factor. The problem of program synchronization is further complicated when the system is heterogeneous because the local environment can cause an instance of the replicated program to diverge in execution flow from other instances of the program. In particular, an individual program's flow of execution is heavily influenced by the local X11 server since individual program flow is dependent on the server's supported visual and generated keycodes. In this paper, we describe our solutions for dealing with a heterogeneous environment and accommodating for non-deterministic processing of input events. Specifically, we describe how we handle a system consisting of heterogeneous display types; heterogeneous keycodes; and non-deterministic delays between input events.

*Research supported by NSF Grant No. ASC-9527186

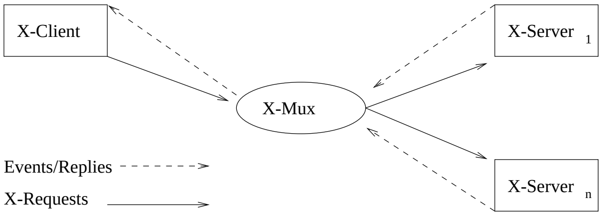

Figure 1: Architecture for an X11-multiplexor.

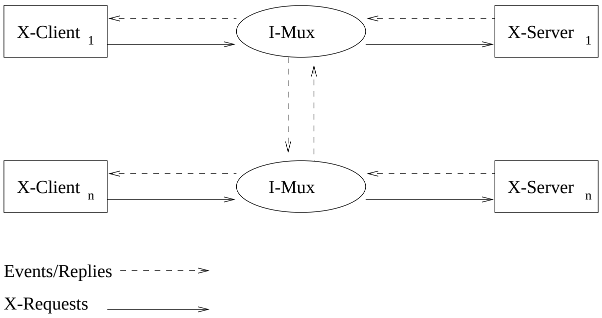

Figure 2: Architecture for an input-multiplexor.

2 Masking a Heterogeneous X11 Environment

In conjunction with the CCF project[9] we have developed the collaborative computing session manager (CCSM)[5] to present a "what you see is what I see" (WYSIWIS) environment for collaborating members who are using both X11 multiplexed and input multiplexed programs. The CCSM is used as a frontend for the input multiplexed programs (Fig. 3) in order to create the view of a homogeneous X11 server for all the replicas even though the actual X11 servers may differ.

We add two features to the CCSM to support input multiplexing in a heterogeneous environment. First, the CCSM must respond to the connection sequence, that describes the X11 server to the client. Second, the CCSM must intercept and respond to all X11 requests that are server dependent and when necessary translate these requests to the local server environment before sending them to the X11 server. The majority of the X11 server dependent requests can be grouped into two classes: those which are used to manipulate the visual; and those that are needed for manipulating the keyboard.

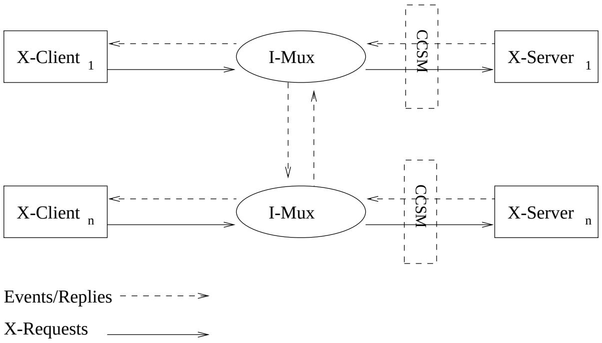

Figure 3: IMUX's architecture with a CSSM frontend.

2.1 Heterogeneous Visuals

The X11 server uses the concept of visual[8] to abstract the different types of display hardware available. For example, two common classes of visual are PseudoColor and TrueColor. In PseudoColor mode an array (typically 256 elements in 8 bit mode) is used to hold a list of 24 bit RGB specification. Then for each pixel on the screen, an index into this array is assigned. Therefore, while 2^{24} different colors can be created, only 256 colors can be used at any time. In TrueColor mode, each pixel is assigned an RGB value which is 24 bits (in 24 bit mode).

The visual available to the application is determined when the application establishes a connection with the X11 server. Since, in our system, the CCSM[5] provides this information, we dictate to the application what visuals are available. In our system we restrict the program to using TrueColor since X requests based on this visual are easier to translate to other visuals. Because the CCSM might be utilizing an X11 server which does not support 24 bit TrueColor mode, the CCSM must intercept and respond to the following X11 requests: `AllocColorCells`, `LookupColor`, `QueryColors`, `AllocColor`, `AllocNamedColor` and `CreateColormap`. The exact method in which these requests are translated is described in [5].

So, for example, if a collaborator is using an X11 server that only supports 8 bit PseduColor mode, then the CCSM must translate references to 24 bit TrueColor pixels, which are RGB values, from the application to an appropriate 8 bit pixel value suitable to the local X11 server before passing the X request to the X11 server. This is accomplished by having the

CCSM do a best fit from the requested RGB value into the color map in use by the CCSM which has the property of a color cube[8]. While we have only described the translation from 24 bit TrueColor to 8 bit PseudoColor, other systems such as monochrome and 8 bit GreyScale can be supported in a similar fashion.

2.2 Heterogeneous Keycodes

In processing keystrokes, the X11 server sends hardware specific scan codes (keycodes) to the application. In order to allow programs to function in a machine independent fashion, routines in the Xlib[7] are supplied for obtaining and performing the conversion between a server specific mapping between keycodes and X11 symbols (keysyms). In addition, the range of legal keycodes is supplied to the program as part of the connection sequence.

To ensure that all clients have the same representation of the keyboard, the range of valid keycodes is set by the CCSM when a client first connects. The CCSM then responds the `GetKeyboardMapping` and `GetModifierMapping` requests so that the client has a server independent representation of the keyboard. Finally, the CCSM translates the keycodes in keyboard events from the system dependent keycodes to the CCSM's system independent keycodes before the events are passed to the client. While this can create some problems if the application is machine specific and expects a certain key to generate a specific keycode, in general these applications would not perform as expected in a foreign environment and therefore this behavior is acceptable (e.g., a Sun specific application would behave oddly if it received keycodes from an SGI).

3 Non-Deterministic Input

Many interactive applications contain a code segment with a loop that checks for and then processes input events. As input events are generated, certain sets of input events are grouped together. The application typically groups these sequences by entering a loop (once the sequence has begun) and exiting the loop when a terminating event is received. However, if the number of iteration through the loop is dependent on the time from the first event to the last event of the sequence rather than on the number of events in the sequence, and if the execution of the loop is not idempotent, a different number of iterations can cause the replicated clients to reach different program states[6].

Related to this problem is the fact that some applications use only a sample from a series of consecutive events of the same type to reduce computational demand (e.g. the *motion notify* (MN) event). So, for example, if the application treats a *button press* event follow by a series of MN events terminated by a *button release* event as a sequence, the application might ignore some MN events to reduce calculations and graphical redisplays. Unfortunately, this sampling depends on when the events are received. Thus, even when given the same sequence of MN events, two replicated clients can generate significantly different displays if the time interval of the received events differ. The problems described above are exacerbated, particularly in heterogeneous environments with widely differing computing speed and network delay.

One solution to this problem, taken by the MM conf project [2], is to modify those applications that exhibit this behavior. Since one of our goals is to support applications without modification we find this solution unacceptable. Another solution to this problem is to pass each event at the same time to all the clients. Achieving this, if not impossible, would slow down the applications significantly due to synchronization delays.

However, one can emulate this synchronization by repackaging and passing all relevant events, including *button press*, *motion notify* and *button release*, in a single message to all the clients. Therefore, when the client receives a button press event, all the relevant events are already received and stored in the input queue, there is no inter-event delay among the MN events, forcing clients to select the same set of MN events. We have implemented this solution in IMUX and it correctly accommodates various applications, including Netscape and Rasmol[1], that suffer from this non-determinism. The events can be packaged either at the CCSM where the events originate or at the input multiplexor. In IMUX we incorporate the latter approach because translation of resource IDs referenced by the input event occur at the input multiplexor and therefore this operation can be performed concurrently with the succeeding MN events.

[1] An application for viewing molecule chains.

However, this solution has some undesirable effects since it reduces the responsiveness of the application to interactive users. We are therefore looking into a possible modification of this solution which selectively packages groups of the events giving some interactive response while preventing non-deterministic behavior.

4 Results and Future Work

Using the CCSM we have been able to multiplex some programs simultaneously on Sun Workstations containing 8 bit PseudoColor hardware, Sun Workstations containing 24 bit TrueColor hardware and Silicon Graphic Workstations containing 24 bit TrueColor hardware. Furthermore, when dealing with high bandwidth applications, such as Rasmol, we have found that IMUX shows a significant performance advantage over a X-multiplexor. However, many problems still remain that need to be addressed. Most notable is dealing with a heterogeneous file system - both with file availability and of the data contained in configuration and data files. These problems are being addressed by the next phase of the CCF project which will supply a homogeneous data space for the collaborators.

References

[1] Hussein Abdel-Wahab and Mark Feit. Xtv: A framework for sharing x window clients in remote synchronous collaboration. pages 159–167, April 1991.

[2] Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, and Raymond Tomlinson. Mmconf: An infrastructure for building shared multimedia applications. *Proceedings, CSCW 90*, pages 329–342, 1990.

[3] Gregg Stanley Foster. Collaborative systems and multi-user interfaces. *PhD Thesis, University of California - Berkeley*, 1986.

[4] Olive Jones. Multidisplay software in x: A survey of architectures. *The X Resource*, 1993.

[5] A. T. Krantz, Sarah Chodrow, Michael Hirsch, and V. S. Sunderam. Ccfx-ccsm a distributed x-multiplexer. 1996. In preparation.

[6] J. Chris Lauwers, Thomas A. Joseph, Keith A. Lantz, and Allyn L. Romanow. Replicated architectures for shared window systems: A critique. *Proceedings, IEEEE Conference on Office Information Systems*, pages 249–260, 1990.

[7] Adrian Nye. *Volume Two: Xlib Reference Manual for Version 11.* O'Reilly & Associates, Inc., Sebastopol, CA, 1988.

[8] Adrian Nye. *Volume One: Xlib Programming Manual for Version 11.* O'Reilly & Associates, Inc., Sebastopol, CA, 1993.

[9] V. S. Sunderam. Collaborative computing frameworks for natural science research. http://ccf.mathcs.emory.edu/ccf/overview.ps.