ShareIt: An Application Sharing System using Window Capturing and Multicast under Heterogeneous Window Systems

Jin H. Jung, Hyun. J. Park, and Hyun S. Yang
Dept. of Computer Science / Center for Artificial Intelligence Research,
KAIST, 373-1 Kusong-dong Yusung-gu,
Taejon, Korea 305-701
email : hsyang@paradise.kaist.ac.kr

Abstract

Application sharing is the ability to use existing applications, such as Excel or MS-Word, during a group session without modification. In this paper, we present the design and implementation of an application sharing system, called ShareIt, which enable users to share arbitrary MS-Windows applications under the Win 3.1/95/NT and X window system, and evaluation of the system performance. To share an application, the image of the application window is captured and transmitted to other sites. With the use of the window capturing method, ShareIt allows any MS-Windows application to be shared regardless of not only the window systems but also the version-up of window systems.

Keywords : application sharing, groupware, multicast

1 Introduction

With recent advances in computer and network technology, a new research field, Computer Supported Cooperative Work(CSCW), has been created to examine how computer and communications technology can support and improve group productivity[2]. While many CSCW systems have been developed, they are not widely used for the following reasons: 1) one must learn how to use these new applications; and 2) the insufficient functionality of the whiteboard used in the shared workspace for group work[15]. Actually, it is either impossible to make CSCW applications that have all the functions users need or hard to use them.

The solution is application sharing. Application sharing enables users to share a program running on one computer with other people in group work. Since users can use everyday applications, there is no need to learn how to use them, and group work can be progress more effectively. Because it allows a seamless change from personal work to group work, CSCW can be spread out more quickly[14].

Many application sharing systems target the same window system. Computing environment of an office, however, is mixed with different window systems such as MS-Windows, OS/2, X window system or Machintosh, while MS-Windows is dominant among these. So application sharing under heterogeneous window systems supports group work more conveniently and naturally.

In this paper, we present the design and implementation of an application sharing system, called ShareIt, which enable users to share arbitrary MS-Windows applications under the Win 3.1/95/NT and X window system, and evaluation of the system performance. To share an application, the image of the application window is captured and transmitted to other sites, so that only one copy of the application is enough.

2 Application Sharing

The idea of shared applications in the computing environment has been around for a long time. Application sharing is the ability to use existing applications such as Excel or MS-Word during a group session without modification. Currently, many research and application sharing products are targets of the X window system. The network transparency of the X window system allows efficient application sharing[12, 4].

In the case of MS-Windows, however, it is not easy to extract information about the graphic output, since the display server is tightly coupled with the application through the Graphic Device Interface(GDI). Furthermore, to enhance graphic output speed, some hardwares control the output device directly. So it is hard to provide application sharing under MS-Windows[5].

2.1 Sharing Environment

There are many windows systems depending on the hardware environments. The popular ones are MS-Windows, OS/2, and X window system. The previous application sharing environment is

within the same windows system. XpleXer[14], ShowMe SharedApp[11] are application sharing products under X window system, whereas ProShare[9], NetMeeting[7], and pcANYWHERE[5] run on MS-Windows.

In contrast, some research has been done on different windows system. WinDD[13] is a commercial product which provides sharing MS-Windows applications under MS-Windows or X-Terminal. In WinDD, the GDI on the MS-Windows are changed to X protocol to share the output image under X-Terminals. WinDD, however, is not application sharing but an emulator which enables multiple users to use Windows NT. Another interesting research concerns application sharing between X window system and QuickDraw [15]. To share any application under both windows systems, a converter between QuickDraw and X protocol has been developed. Research to develop converter modules for MS-Windows continues.

2.2 Implementation Methods

There are two ways to implement application sharing; intercepting the graphic primitives and capturing the application windows.

2.2.1 Intercepting Graphic Primitives

Intercepting graphic primitives is a popular method under the X window system. One approach is to modify the X window system itself to support application sharing[3]. In this approach, one can easily make a shared application using new APIs. Another approach is to make a virtual X-Display server[14, 15]. In this approach, graphical output requests of an X client are forwarded by the virtual X-Display server to several X servers and events from these X servers are collected and sent back to the X client.

A product which uses this method under MS-Windows is WinDD[13]. WinDD sends the graphics for Windows 95, Windows NT, and Windows 3.1 applications over the network to any X11 device such as an X-terminal or a workstation and Intel-based PCs.

Since applications under MS-Windows are tightly coupled with a display configuration, it is not easy to provide intercepting-based application sharing. Moreover, due to some programs that directly control graphic hardwares to enhance graphic output performance using a special display driver, GDI messages are not sufficient to support application sharing.

The intercepting method provides fast response time since the data needed to share an application is graphic primitives. This method, however, has numerous critical disadvantages when used under MS-Windows.

- It is difficult to implement since all GDI messages used under MS-Windows must be intercepted.
- Continuous version-up is needed to handle new GDI messages when the MS-Windows version is changed.
- Due to different graphic primitives depending on the windows system, protocol converters between different windows systems are needed.
- It is hard to accommodate latecomers into a group session because the current state of the shared application must be updated by playing back the intercepted messages of the whole session.

2.2.2 Capturing Application Windows

In order to share applications within MS-Windows, capturing application windows is mainly used due to its simple mechanism. In capturing application windows, captured images of an application on the desktop are periodically broadcast to other sites. Many commercial products, for example, NetMeeting[7] and ProShare[9], use this method. One of the disadvantages of this approach is the slow response time, since the size of the captured bitmap is large. On the other hand, the advantages are the following:

- It is easy to implement.
- It is possible to use without modification even if Microsoft releases a new version of MS-Windows.
- As for special messages, such as MCI messages for multimedia, they are easy to share at least as far as the images shown on the screen themselves are concerned.
- It is easy to apply to different window systems.

2.3 The Chosen Items of This System

We select heterogeneous window systems, including MS-Windows and X Window system as the sharing environment. To provide application sharing, a window capturing method is selected. As a result, ShareIt allows not only sharing video data which uses MCI commands but also application sharing between different window systems without protocol conversions.

3 Design and Implementation

3.1 Overview

To share arbitrary MS-Windows applications under both the MS-Windows 3.1/95/NT and the X window

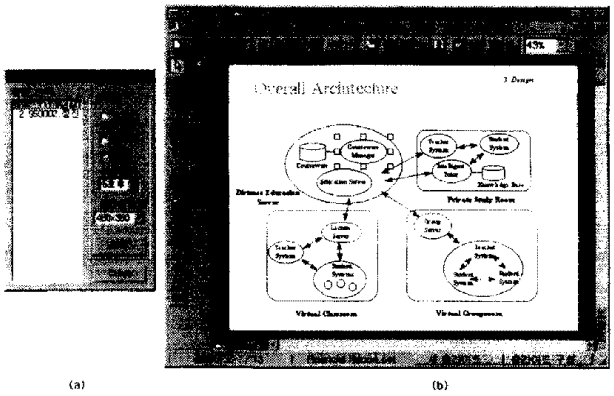

Figure 1: Powerpoint executed on the server

system, ShareIt provides application viewers for MS-Windows and X window system. The hardware platform is SUN Sparc10 and PC Pentium 160Mhz with 32M memory. A 10 Mbit/s Ethernet is used to transmit a captured bitmap and control information. For efficient group communication IP Multicast is used.

Fig. 1 shows that group discussion is performed under Win95 and the X window system using Powerpoint. Fig. 1(a) is the monitor enabling the local user to control the application sharing session. The participants on the group session are listed on the left side. Fig. 1(b) is a snapshot of the shared Powerpoint. Fig. 2 and Fig. 3 show the application viewer for Win 95 and X window system, respectively.

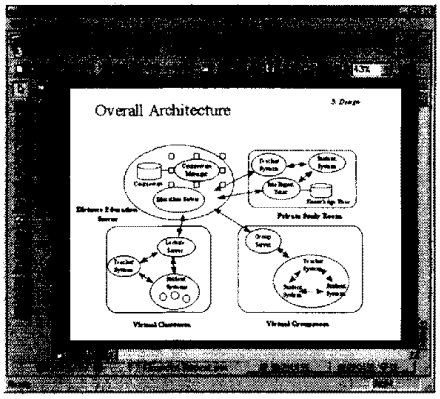

Figure 2: The Application Viewer for Win95

In order to share an application among users, the control of user input is essential. In this system, one user at a given time can manipulate the application, since one copy of the application is executed. That is, only the user who has input right, called *token*, can control the shared application. When a remote user wants to use the application, he can control it after

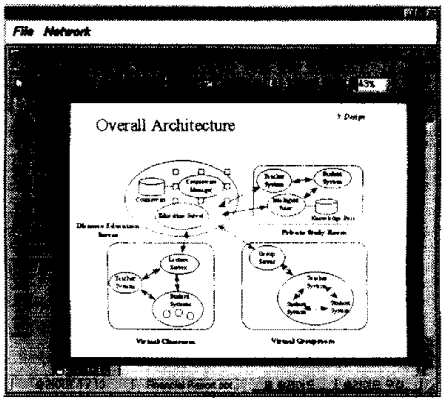

Figure 3: The Application Viewer for X window system

receiving the token from the AppShare Server.

3.2 System Architecture

As shown in Fig. 4, ShareIt consists of four components: the AppShare Server, the AppShare Viewer, the Input Recorder and the Input Player.

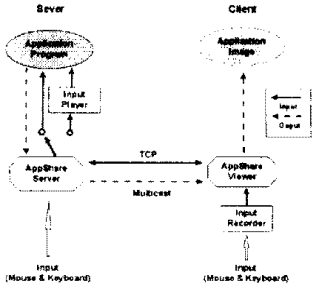

Figure 4: System Architecture

The AppShare Server resides on the server part on which an application is executed and takes charge of capturing the application window as well as the input control of the application. Periodically, the AppShare Server captures the shared application window and broadcasts it to clients. Another rule of the AppShare Server is input control. The AppShare Server selects input direction for the application. By default, the user residing on the server has input right. Based on input permission, the AppShare Server arbitrates between the local and remote user's input messages. If a remote user has input permission, the received input messages from the client are sent to the application.

The AppShare Viewer displays a bitmap image from the AppShare Server and shows a virtual mouse cursor which shows the same position of the mouse on the shared application. When a user on the client gets input permission, the input messages which were saved

by the the Input Recorder are sent to the AppShare Server.

The Input Recorder records the client input. The recorded information is the state of the mouse button, the mouse location, the state of the keyboard, the type of pressed key, and the time of the occurrence of the event.

The Input Player acts as a virtual input device such as a keyboard or mouse. Recorded input messages are changed to real input messages and then given to the application as if the local user generated them on the local mouse or keyboard.

3.3 Input Translation

In order to enable a remote user to operate the shared application on the server, it is necessary to consider the remote input messages as local input messages. In the Microsoft Windows graphical environment, a hook is a mechanism by which a function can intercept events before they reach an application[6]. To support a hook mechanism MS provides two special functions: JournalPlaybackProc and JournalRecordProc.

The application viewer for the X window system has several special functionalities. Since the X window system has no a hook mechanism, the AppShare Viewer for the X window system acts as an input recorder to record input messages. Another rule concerns the format conversion of messages. The format for input messages of the X window system is different from that of MS-Windows. The AppShare Viewer converts the X events to events used on MS-Windows. In terms of the AppShare Server, there is no need to distinguish the input messages from which types of application viewers.

4 Network Consideration

The amount of data to be transmitted increased proportional to the number of users. To make an application sharing system feasible, one must address not only an image compression scheme to reduce the data to be transmitted, but also an efficient group communication mechanism to send data from one source to group destination.

4.1 Image Compression

Recently, many standards such as JPEG, MPEG and H.261, have developed depending on the image to be compressed[1]. These standards give a high compression ratio with lossy compression. In application sharing, however, the following three demands arise: First, there must be lossless compression. Since a captured application image is computer-generated, users notice a difference even when there is a small loss of information. Second, to allow fast response time on the currently available networks such as Ethernet, high compression is needed. Lastly, fast compression and decompression should be possible. There must be a simple way to compress/decompress in real time without specialized hardware support.

In this paper, to satisfy the above requirements, we select Run-Length Encoding(RLE) for the difference image between the current image and the previous image for the following two reasons. First, RLE is lossless data compression[10] and allows compression/decompression in one path. Second, the updated region of an application is generally limited to a part of the screen. At this time, the difference between the current image and the previous image may be mainly zero except for the updated region which has a difference value. By applying RLE to the difference image, we can get a high compression ratio.

4.2 IP-Multicast

Distributing data to multiple participants using connection-oriented protocols, such as TCP, can be inefficient because the data must be transmitted over the network multiple times, once to each target. The solution is multicasting. Multicasting is a networking mechanism which enables the large scale delivery of information from one sender to many recipients or from many senders to many recipients. The main objective of multicast protocols for transporting real-time data is to guarantee the quality of service by bounding end-to-end delay at the cost of reliability[8]. In this paper, we propose an efficient retransmission strategy for minizing ene-to-end delay.

4.2.1 NACK Processing

When sending the first packet of a frame, the server sets the total number of packets for the given frame to an *Offset* field. Packet loss is normally detected by finding a gap in the packet number. However, since it is possible to estimate the arrival time of the next packet based on the relation between the received packet number and the total number of packets for the given frame, we can determine a packet loss without receiving the next packet. After receiving the first packet of the frame, each client estimates the next packet's arrival time, T_{exp}, based on the distance[1] from the server, and waits T_{exp} time. If the next packet arrives before the request timer expires, the client adjusts T_{exp} using an exponentially-conveying delay estimator,

$$T_{exp} = \alpha * T_{trs} + (1 - \alpha) T_{exp},$$

[1] When a client joins an application sharing session, the distance from the server to the client is calculated

where the value of T_{trs} is the time delay between the previous and current packet arrival time and $\alpha = 1/4$ in our system.

4.2.2 Retransmission Strategy

When receiving NACKs from clients, the server inserts the NACK packet to the *retransmission queue* while incrementing a counter as well as the client address. According to the status of the current sending frame, a different retransmission strategy is used. If the current sending packet and the NACK packet are both the part of *frame i*, the server delays retransmission until the remain packets of the given frames are sent out. We can reduce the number of retransmission packets by delaying the retransmission as long as possible. After sending out the last packet of the frame, the server looks up the *retransmission queue* and multicasts NACK packets if the counter is more than one-third of the participants. Otherwise NACK packets are retransmitted to the client via a TCP connection. If the current sending packet is a part of *frame i+1* or more, the server retransmits the NACK packets to the clients immediately via a TCP connection. As a result, we achieve minimizing end-to-end delay at the expense of the retransmission cost.

5 Performance Evaluation

In this section, the performance of ShareIt is evaluated. To measure response time for the keystrokes, we selected MS Word that mainly used for word processing. For response time of mouse movements Powerpoint was used. According to the resolution of the applications, average transfer time and rates as well as response time were evaluated.

5.1 Transfer Time

Each processing stage in the sending and receiving terminals introduces delay. The time needed to transfer an application bitmap image from the server to clients can be divided into the following four stages: capturing a window image from the frame buffer, calculating the difference image between the current image and the previous image, processing the difference image to runlength data, and transmitting the runlength data to client. It also takes time to decode a received image and display the bitmap on the client screen. Since the time taken by the client is constant depending on the resolution of the shared application, we omit it.

We evaluate how much time is taken for each stage using MS Word and Powerpoint during five minutes. Table 1 summarizes the processing time for each stage depending on the application resolutions. It took more time to process Powerpoint than MS Word for each resolution. Since Powerpoint is used to design a presentation by graphic data, the updated region was wider than MS Word in handling text data.

Many frames are not needed to send bitmaps to the client. The rate of no difference between the current image and the previous one was approximately 57%. The time to capture a window, make the difference image and process runlength encoding was linear in the size of the shared application. These stages can be improved by using a faster CPU or a graphic accelerator.

The amount of RLE data during the above experiments is shown in Table 2. Images which have no difference from the previous image were omitted. An interesting point is that in the case of 800x600 resolution of MS Word, runlength encoded data were smaller than the 640x484 resolution. This shows that the size of the runlength encoded data is determined from the updated region rather than from the resolution of the application. Powerpoint showed more runlength data than did MS Word. That is, the updated region of MS Word, due to key strokes, is relatively small. In contrast, the updated region of Powerpoint was wider and the update rates were faster. A similar result was shown in the average compression ratio. In the case of MS Word, ShareIt showed a more than 10000:1 compression. Powerpoint's average compression ratio was 3000:1, one third of MS Word's.

Table 1: Comparison of processing time depending on the resolution of the application (msec)

Application		Capture	Diff.	RLE	Trans.	Total
MS WORD	400x300	12.78	4.50	13.86	21.09	52.23
	640x484	31.62	25.79	35.02	36.72	129.15
Power Point	400x300	13.08	17.72	14.37	45.82	90.99
	640x484	37.57	54.86	36.45	67.39	196.27

Table 2: Comparison of RLE data depending on the resolution of the application (bytes)

Application (Original)	400x300 (120000)	640x484 (309760)	800x600 (480000)	Avg	Comp. Ratio
MS Word	339.16	1113.10	682.19	711.33	10236:1
Powerpoint	1159.43	1186.84	4113.84	2153.33	3380:1

5.2 Response Time

When the response from the keystrokes or mouse movements is slower than expected, users feel inconvenienced. Response time is the time that elapses after a remote user invokes input messages by keyboard or

mouse before the result is shown on the screen. A user on the server can get a response for his input immediately. In contrast, client users get the result image of an application from the server. So the response time for remote users is inversely proportional to the frame rates.

Fig. 5 shows the client response time when a 680x484 resolution of MS Word is shared. The average response time was around 136 msec. That speed is the same as the input 7 characters per second(cps). In expert terms, 7 cps is somewhat slow. Application sharing, however, is used when groups of people work together, so intensive word processing seldom occurs. To improve keystroke response time, a focus technique can be used. For example, as long as the content in the application window is not scrolled, the updated region is restricted within a small area. So there is no need to capture the whole application window.

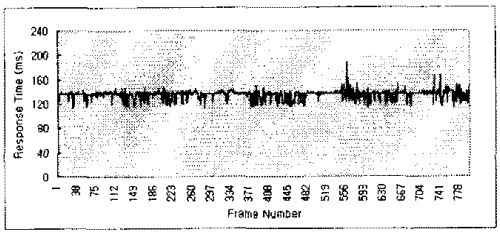

Figure 5: Response time for 640x484 MS Word

One possible way to enhance the mouse movement response time is by applying GDI message-intercepting to extract the updated region of the application window.

6 Conclusion and Future Research

In this paper, we have presented the design and implementation of ShareIt which allows users to share arbitrary MS-Windows applications under both Win 3.1/95/NT and the X window system using a captured window bitmap, and evaluated the system performance. To achieve fast response time, we use run-length encoding for the difference image between the current image and the previous image and multicasting with an efficient retransmission strategy. ShareIt can be used in a shared workspace of CSCW systems such as desktop conference systems and on-line software training.

Future research remains to be done. We are interested in an application viewer using JAVA. Since JAVA can be executed on platforms-independent environments, there is no need to make application viewers for each window system. Lastly, the notion of making application sharing private and secure needs to be addressed.

References

[1] Andleigh PK, Thakrar K (1996) Multimedia Systems Design. Prentice Hall

[2] Ellis C, Gibbs S, Rein G (1991) Groupware Some Issues and Experiences. Commun ACM 34: 38-58

[3] Garfinkel D, Welti BC, Yip TW (1994) HP SharedX: A Tool for real-time coolaboration. HP Journal 45:23-36

[4] Gutekunst T, Bauer D, Hansan GC, Plattner B (1995) A distributed and Policy-Free General-Purpose Shared Window System. IEEE/ACM Trans on Networking 3: 51-62

[5] Lori M (1996) Dialing in to Win 95. PC Week 2

[6] Marsh K (1992) Microsoft Windows Hooks. Microsoft Development Library

[7] NetMeeting http://www.microsoft.com/ie/ie3/netmtg.htm

[8] Pejhan S, Scjwartz M, Anastassiou D (1996) Error Control Using Retransmission Schemes in Multicast Transport Protocols for Real-Time Media. IEEE/ACM Trans on Networking 4: 413-427

[9] ProShare http://www.intel.com/comm-net/proshare/techinfo/index.htm

[10] Rabbani M, Jones W (1991) Digital Image Compression Techniques. SPIE Press

[11] ShowMe SharedApp http://www.eu.sun.com/products-n-solutions/sw/ShowMe/Products/ShowMe_SharedApp.html

[12] Wahab HA, Jeffay, K (1992) Issues, Problems and Solutions in Sharing X Clients on Multiple Display. TR92-043, Univ. of North Carolina

[13] WinDD User Information http://www.tek.com/Network_Displays/Support/windd/winddusr/main.htm

[14] Wladimir M (1995) The Application Sharing Technology. http://landru.unx.com/DD/ advisor/docs/jun95/jun95.minenko.shtml

[15] Wolf KH, Froitzheim K, Schulthess P (1995) Multimedia Application Sharing in a Heterogeneous Environment. In: ACM Multimed 95 San Francisco, California