Scalable and Resilient Application Sharing System for Internet Collaboration

Agustín José González
Electronic Engineering Department
Universidad Técnica Federico Santa María
Valparaíso, Chile
agv@elo.utfsm.cl

Hussein Abdel-Wahab and J. Christian Wild
Computer Science Department
Old Dominion University
Norfolk, VA, USA.
wahab@cs.odu.edu

Abstract

Increased desktop processing power and network bandwidth have made feasible distributed multimedia collaborative systems. Such systems are characterized by video, audio and data exchanges. While there has been much research and development of protocols and services for video and audio transmission, less work has been focused on data sharing particularly when the data is being generated by an application in real time and in turns controlled by one or more of the participants in the collaborative session. While sharing the view of an application can be thought of as a kind of video stream, the unique characteristics of these dynamic images require new algorithms and transmission protocols to achieve legibility and size dynamic. This paper describes a protocol and its implementation for sharing desktop applications in a distributed collaborative session. Key objectives of this protocol are scalability and resilience to dropped packages and to participants joining and leaving a session. We describe design decisions and give results demonstrating the effect two different compression algorithms and protocol parameters. Finally, Odust a tool sharing system built on the implementation of shared application views is described.

1. Introduction

The growth of the Internet and the increased performance of desktop computers have made feasible large-scale multimedia applications over communications networks. This work focuses on synchronous collaborative multimedia applications. Synchronous collaboration takes place when the participants involved in common tasks are seated simultaneously at their workplaces. This is the case of virtual-classroom systems for distance learning and multimedia conferencing systems. In this work we describe a distributed tool for sharing applications to be used in conjunction with video and audio in these scenarios.

Synchronous multimedia applications are based on three basic components: audio, video, and shared data. The transmission of audio and video information over digital networks has been studied for years and the field is relatively mature. However work on the transmission of data from interactive applications to a large group is still a relatively new area of investigation. The information shared in a collaborative session is often one of the main foci of the session. Rather than sending hard copies or faxing the material to remote participants, today's collaboration systems use the network to distribute this information on the fly. Many specialized systems have been developed for that purpose, such as co-browsers [3] [7], and sharing tool engines [1] [21]. In other cases, the collaboration application includes a module for data sharing such as in [16] [14] [17]. Although all these systems provide a number of features, the major contribution of them to a collaborative session is the ability of distributing data information on-line by emulating a virtual projection screen. One approach for tool sharing enables one to share existent unmodified single-user applications. Examples of such systems are X Teleconferencing and Viewing (XTV) [1], Virtual Network Computing (VNC) [21], and Java Collaborative Environment (JCE) [2]. Another technique for tool sharing is to control the execution of multiple synchronized instances of the same application; an example is Habanero [9]. Our work follows a similar approach to that taken by VNC, which achieves tool sharing by distributing the desktop as an active image. This means an image which users can interact with in a similar way they do with their local desktop. This technique allows one to share visual component of all application on the screen. Rather than sending the entire desktop, we propose to send the images of the active windows of the applications running on any of the participants' screens. Unlike the other approaches our transmission protocol relies on IP multicasting to achieve scalability and a video-like scheme to overcome packet losses. In this work, we describe a protocol for sending dynamic images and Odust, a distributed sharing tool application based on this protocol. Odust, users at the receiving sites cannot only view transmitted images of the shared application, but they can also request a floor and control the shared tool remotely. Likewise, any participant can share any local machine-dependent application. Our goal in designing this Odust was to provide a tool for interactive distance learning systems and large group conferences on the Internet where it should work in conjunction with distributed audio and video tools. In such scenarios, we used Java and a minimum of unavoidable architectures dependencies to achieve portability. Also, the robustness of our distributed protocols provide graceful degradations in face of users or communications failures.

The rest of this paper is organized as follows. The next section describes the Dynamic Image Transmission Protocol giving results of experiments that were used in developing and tuning its implementation. Section 3 describes a tool sharing service which uses the dynamic image transmission protocol. This is followed by a section giving related work and a conclusion.

2. Dynamic Image Transmission Protocol

The protocol for transmitting dynamic images presented here enables data sharing by disseminating images generated by shared interactive applications. From the communication point of view, the two main features of this protocol are resiliency and scalability. Resiliency refers to the ability of the protocol to overcome transmission losses and to accommodate the leaving and joining of participants. Scalability is achieved by eliminating the need for acknowledgements or any feedback from the receivers. Dynamic images, like video, contain spatial and temporal redundancy that the protocol removes. In this respect, the set of views of a running application can be thought of as a video stream of dynamic images and can utilize existing video compression technology. Our protocol tiles the image in square blocks, and then it encodes each block using a standard image coding to remove spatial redundancy. Only blocks that change between two image samples are encoded, thus some temporal redundancy is also removed. There are several advantages to dividing the image into tiles that relate to error recovery, late joins, and transmission efficiency. Temporal redundancy refers to those parts of the image that remain unchanged from one instant of time to the next. To remove this type of redundancy motion-compensated prediction [6] has been used in video encoding. It assumes that pixels within the current picture can be modeled as a translation of those within a previous picture. Motion prediction is computationally intensive and has limited utility for the synthetic video streams generated by computer applications when the sample rate is low, therefore we have chosen to only use motion prediction with null motion vector. Thus blocks that remain unchanged from one sample to another are detected and not resent. In contrast to video, these images tend to be of higher and variable resolution than traditional video images and present lower degrees of motion. In addition, dynamic images often contain information which requires high fidelity rapidly becoming illegible as image quality is reduced. We call this property *legibility*. This is especially true for images containing text. We estimate that a sampling rate of around 2 samples per seconds fulfills the requirements of most types of data sharing. It also takes into consideration the computation power utilized by multimedia applications; so that given a bounded CPU allocation for data sharing, the bigger picture processing can only be achieved by reducing the processing cycle rate. Knowing the expected sampling rates, let's us revisit our decision about motion prediction and better justify our argument. We believe that in low sampling rate, i.e. around 2 Hz, motion prediction loses effectiveness because at this frequency the motion vector is likely to be out of the reach of the search window of motion-compensated prediction techniques. For example, in H.263 the search window for motion prediction is such that motion of at most 16 pixels horizontally and/or vertically can be predicted.

Image size changes are also transmitted by the protocol. The size of an image might change from one sample to another. In computer application the main cause of images change in size is window resizing. We observe that window resizing usually preserves the upper left content of the view regardless the side or corner used for resizing. Therefore, while comparing blocks between an image and its resized version, the protocol assumes that both samples share a common upper left region. Consequently, receivers initialize the new version of the image with the upper left content of the previous instance of the image.

The use of unreliable transport protocol forces our us to take some precautions to overcome packet losses. We decided against selective retransmission of lost data because of its difficulties in getting feedback from an undetermined number of receivers [19]. Instead, we send new data or control the retransmission of the same data to eventually repair the original lost. As introduced by the principle of Application Level Framing (ALF) [5], we define the protocol data unit (PDU) such a way that each PDU can be processed out of order with respect to other PDUs. As a result, each PDU conveys at least a tile, its coordinates within the image, a tile-based sequence number, and timestamp. In principle, each altered tile needs to be sent once; however, we schedule its retransmission after random times taken from (0, *update_tile*] sampling periods. The protocol accommodates late comers by sending a refresh for each PDU after a random time taken from the interval (*update_tile*, *refresh_time*]. This ensures a full image retransmission takes place at most every *refresh_tile* sampling periods. This type of refresh also strengthens protocol resiliency and enables the detection of removed or closed images as we discuss below.

Image creation is simple and signaled by the reception of the first PDU; however image removal is a little more involved since there is no guarantee that any explicit close image message will reach all the receivers. Close image messages are used in conjunction with a refresh image timeout to determine that the dynamic image was closed.

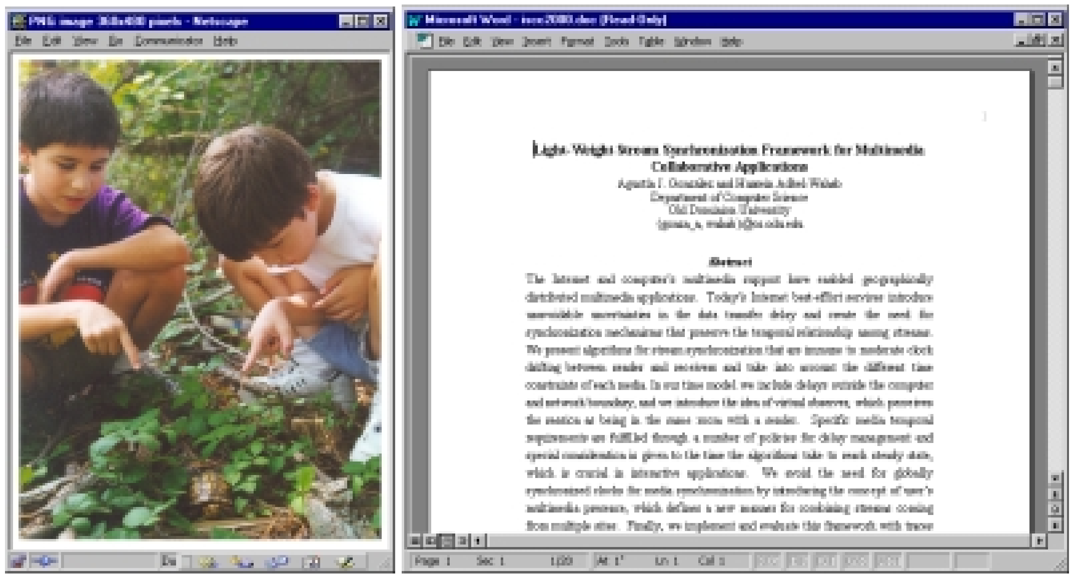
In the following sections we discuss the parameters of the protocols and their impacts on performance. First, we analyze the effect of two common compression standards for still image encoding that we tested for tile compression, and we discuss the tradeoffs in selecting the tile size. Then, we measure the protocol processing times and used them to estimate the sampling rate. Finally, we extend these ideas for the transmission of multiple possible overlapped images. To evaluate our protocol, we implemented it on Java 2 SDKv1.2.2 and employed Java Advanced Imaging 1.0.2 for tile compression [22].

2.1. Tile Compression Format Study

We considered and compared Joint Photographic Experts Group (JPEG) [24] and Portable Network Graphics (PNG) [4] for tile coding. The criteria for selecting the compression technique were compression time, compression ratio, and legibility. These factors were evaluated as a function of the tile size around 32x32 pixels (the selection of tile size is covered in the next subsection). We compared two image types shown in Figure 1. While Figure *1a* is a real world picture, Figure *1b* shows a synthetic image containing text generated by a word processing program. These two types of images were picked as representative of the differences between natural and synthetic images. As shown in Table 1., there are significant compression ratio variations between these two images. For the natural world image, JPEG compression outperforms PNG by a factor of almost 10. For the synthetic image, PNG outperforms JPEG by a factor around 4.5. Another factor in consideration is the lossy and lossless nature of JPEG and PNG respectively. Due to the lossy nature of JPEG, a quality factor needs to be provided for compression. While quality values of around 50% are normally acceptable for pictures, higher values are required for legible text images. PNG, on the other hand, is lossless and offers good compression rates for text and line type of images, but it does not compress well the information of real-world scenes.

TABLE 1. PNG/JPEG comparison for two types of images.

Image	PNG size [Kbyte]	JPEG size [Kbyte]	PNG/JPEG Ratio
388x566 Real-wold	340	36 (Q=50%)	9.4
680x580 Text	21	94 (Q=75%)	0.22


PNG image 388x400 pixels - Netscape
File Edit View Go Communicator Help
Microsoft Word - ieee2000.doc [Read Only]
File Edit View Insert Format Tools Table Window Help
Page 1 Sec 1 1/20 At 1" Ln 1 Col 1

Light-Weight Stream Synchronization Framework for Multimedia Collaborative Applications
Agustín J. González and Hussein Abdel-Wahab
Department of Computer Science
Old Dominion University
{gonza_a, wahab}@cs.odu.edu

Abstract
The Internet and computer's multimedia support have enabled geographically distributed multimedia applications. Today's Internet best-effort services introduce unavoidable uncertainties in the data transfer delay and create the need for synchronization mechanisms that preserve the temporal relationship among streams. We present algorithms for stream synchronization that are immune to moderate clock drifting between sender and receivers and take into account the different time constraints of each media. In our time model, we include delays outside the computer and network boundary, and we introduce the idea of virtual observer, which perceives the session as being in the same room with a sender. Specific media temporal requirements are fulfilled through a number of policies for delay management and special consideration is given to the time the algorithms take to reach steady state, which is crucial in interactive applications. We avoid the need for globally synchronized clocks for media synchronization by introducing the concept of user's multimedia presence, which defines a new manner for combining streams coming from multiple sites. Finally, we implement and evaluate this framework with trace

a) b)

Fig. 1. Test images a) 388x566-pixel real-world image and b) 680x580-pixel Text image.

In addition to comparing these two formats for full-size images, we also compared them for tile-size images including two representative backgrounds typical on many synthetic images. JPEG (at 50% quality) has an overhead of around 600 bytes while PNG overhead is near 100 bytes; however, JPEG image size grows with a lower slope compared with PNG's as image size grows. By overhead we mean the size of the compressed image as its size approaches zero.

As a result, there is lesser variation of compression ratio with JPEG, but it imposes a higher overhead in each tile. This defines the compression limit. As for these results, it appears that tiling decreases the performance; nonetheless, the counter argument is that smaller tiles enable more temporal redundancy removal. In addition, network packet fragmentation also plays a role in determining an optimal tile size. We discuss these tradeoffs in the following section.

2.2. Selecting Tile Size

The definition of the tile size has a crucial effect on performance. As stressed by the principle of Application Level Framing (ALF) [5], loss of data unit fragments prevent data unit reconstruction and cause bandwidth misusage due to the reception of data that cannot be processed. We measured packet size after compression using PNG and JPEG coding formats.

For PNG encoding, only 16x16-pixel tile size leads to a single network frame per packet for all tiles on Ethernet, and fragmentation is unavoidable for any other size on real-word images. For text-like images, on the other hand, PNG does a very good job in producing a single fragment even for 64x64-pixel tiles. In contrast, the average and maximum packet sizes do not vary much with the image content in JPEG. As a result, we selected JPEG compression and 40x40-pixel tile by being the biggest tile that does not lead to fragmentation on Ethernet. Fragmentation imposes a penalty not only on bandwidth but also in transmission processing time as we elaborate in the next section.

2.3. Protocol Processing Time

We measured the processing time using our prototype implementation. On the sending side, processing time is divided into the following steps:

1) Time to capture the dynamic image; this is independent of the compression algorithm.
2) Time for comparing new and old tile images for temporal redundancy removal (also independent of compression algorithm.
3) Time to compress all changed tiles.
4) Total transmission time.

The two images shown in Fig. 1 were used is this comparison study. We experimented with the two compression techniques and different tile size. The main results are shown in TABLE 2. In our tests compression was performed by the JAVA Advanced Imaging package [22] and was the dominant processing time. Notice these measures represent an upper bound for the two regions since we assumed a sequence of two identical images for comparison which forces the algorithm to touch every pixel, and two distinct images for compression which forces the protocol to compress every tile. In practical cases, we have a fraction of this comparison cost and a fraction of this compression time. Thus depending on the updates from one sample to another, the total processing time goes from 407 ms to 1.3 s for the real-world image.

TABLE 2. Sender's processing time using JPEG and 40x40 pixel tiles.

Step	Real-world Image (Fig. 1a)	Text Image (Fig. 1b)
Capture	100 ms (6%)	170 ms (6%)
Comparison	307 ms (19%)	556 ms (20%)
Compression	1132 ms (70%)	1889 ms (69%)
Transmission	78 ms (5%)	143 ms (5%)

Overall, JPEG encoding ended up being faster for computing the complete processing cycle of this application mainly due to its library speedup over PNG. This result shows that small updates can be sent at a rate of 2 Hz for this image size, and it takes up to around 2 seconds to send an entire new image. These lower and upper bounds are directly proportional to the image size.

At the receiving site, processing time is divided into

1) Time to decompress the tile images
2) Time to draw the image
3) Time to display

TABLE 3. Receiver's processing time using JPEG and 40x40-pixel tiles.

Step	Real-world Image (Fig. 1[a])		Text Image (Fig. 1b)	
	WinNT	Solaris	WinNT	Solaris
Decompression	677 ms (86%)	483 ms (67%)	795 ms (79%)	826 ms (64%)
Updating Image	38 ms (5%)	46 ms (6%)	135 ms (13%)	85 ms (7%)
Displaying	75 ms (9%)	191 ms (27 %)	78 ms (8%)	364 ms (29%)

In contrast to the sender part of the sharing tool application that depends on native calls for image capture, the receiver part is fully coded in Java and run on WinNT or UNIX machines (Pentium II 266 MHz, 64 MB and Sun Untra 10 respectively). The results for both platforms are shown in TABLE 3. Like de sender's processing times, these are upper bounds because we expect a fraction of the tiles to remain unchanged and therefore an update will take a fraction of these total times. The dominant cost is decompression and the total time shows that the bottleneck is the sender side of the protocol.

3. Protocol Extension for Transmission of Multiple Related Windows

A scheme for sharing multiple window images of a single application can be extended from the protocol described in Section 2. In addition to image dimension, now the position of each image must be sent in each tile data unit. Also, when the shared application spawns multiple windows, overlapped regions can be identified to reduce traffic and coding processing.

The problem of partitioning a rectilinear polygon into a minimum number of non-overlapping rectangles appears in many applications besides our imaging application. These include two-dimensional data organization [13], optimal automated VLSI mask fabrication [20], and image compression [18]. The problem is illustrated in Fig. 2. In our application, a simple and straightforward approach would capture and transmit each window. The result is that the overlapped regions (in dark) would be processed and transmitted twice.

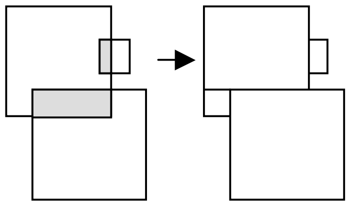


Fig. 2. Overlapping regions in Related Windows.

The minimum partitioning problem was optimally solved by Lispki in [13] and Ohtsuki in [20]. Ohtsuki's algorithm runs in $O(n^{5/2})$ time in the worst case. Later, in [10] Imai and Asano proposed an algorithm that requires $O(n^{3/2}\log n)$ time. Liou *et al.* proposed in [12] an optimal $O(n\log\log n)$-time algorithm for partitioning rectilinear polygon without holes. Despite the optimality of the previous algorithms, their complexity has precluded their usage in applications that require fast encoding operations [18].

We opted for a sub-optimal solution that could be easily integrated with the tiling technique for image transmission. Our algorithm progressively receives the rectangles to be transmitted and returns for each tile the already sent rectangle that fully contains it, as shown in Fig. 3.

```
Initial Condition:
        $\boldsymbol{R} = \phi$;                // set of already sent rectangles.
```

```
Before transmission of tile x:
    for each rectangle r in R:
        if ( x is fully contained in r )
            return r;
    return null;
After transmission of image within rectangle r:
    $R = R \cup \{r\}$;
```

Fig. 3. Algorithm to suppress overlapped region retransmission.

The protocol for sending dynamic images slightly changes to integrate the algorithm for overlapping suppression. If a tile is already at the receiving site, a copy message is transmitted for the receiver to take the tile from the already received image. Obviously the algorithm is not optimal except for 1x1-pixel tiles since a tile that only falls partially into an already sent rectangle is transmitted anyway.

4. Odust

Odust (Old Dominion University Sharing Tool) is a sharing tool engine based on dynamic image protocol for transmitting images of application windows. We mention only WinNT even though we also mean Win95, and Win98. We have successfully tested receivers on WinNT and Solaris 2.6; however, recipients could be on any machine that runs Java and the Java Advanced Imaging package (JAI) [22]. Indeed, only Java is strictly required since JAI can run over pure Java code with some loss in performance.

4.1. Description

Odust is a distributed cross-platform application that enables data sharing in synchronous multimedia collaboration. An owner of the shared application operates the real instance of it on the screen, while the other participants see and operate images, which are generated by Odust and are in many ways indistinguishable from the real application. Sharing is done with *process granularity* meaning that all the windows belonging to a process are shared atomically. A floor control service, described in [8], allows any receiver to request the control of the shared application by preempting it from the current holder. Although one receiver can have the floor at a time, the shared tool owner running the real version of it can also operate it at any time. A drawback of this technique is the interference of the floor holder input events, i.e. keyboard and mouse, with the same input devices at the application owner's machine. Due to the lightweight nature of the protocol for sending images, any participant can leave the collaboration session at any moment. Likewise, anybody can join the session at any time. These two situations have virtually no effect on the other participants. Users joining the session late reach a synchronic view within a bounded time, which is a parameter in Odust. Multiple participants can simultaneously share their applications as long as only one tool is shared per site. Each shared tool is displayed in a separate window at the receiving user's desktop.

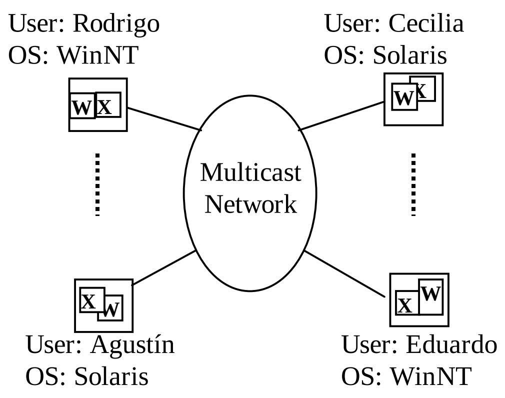
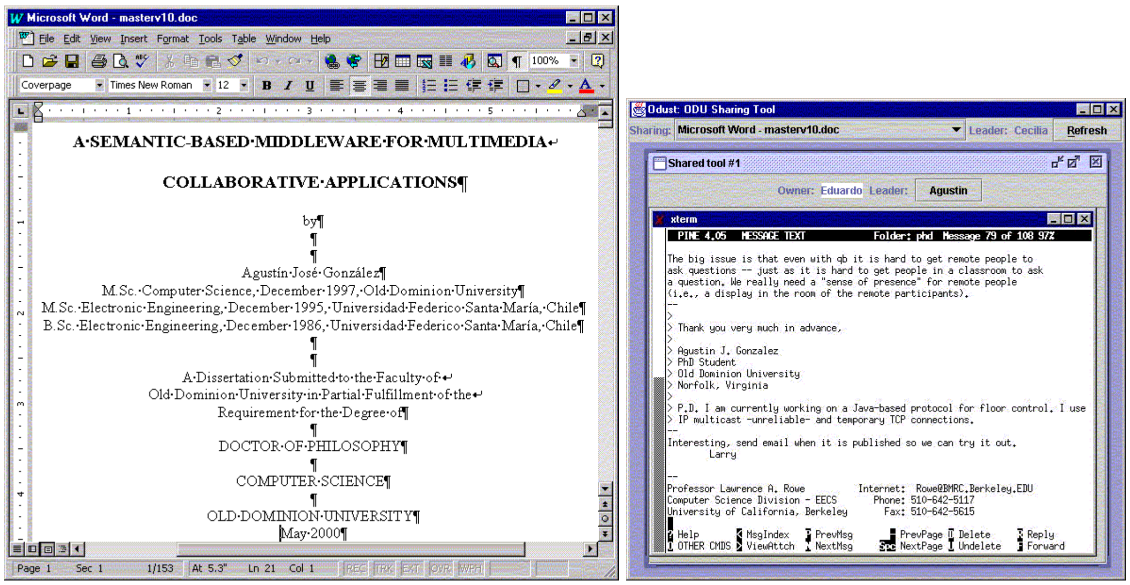


Fig. 4. Tool sharing scenario with Odust.

Fig. 4 shows one of the multiple scenarios where Odust can be used. In this scenario, four participants are sharing two computer applications. One application is an "xterm" (marked with an "X" in Fig. 4) which is running on user machine Eduardo. The second application is MS-WORD (marked with a "W" in the figure) which is running on user machine Rodrigo. Scalability is gained mainly due to the use of IP multicasting, which is a network requirement for Odust to work in more than 2 participant sessions. It also works over unicast network for 2-party sessions. This feature is basically inherited from the unified unicast-multicast API described in [8]. The following figure illustrates the view that user Rodrigo of the Fig. 4 sees on the screen. The other views are similar, although each user can arrange the screen differently if desired (as suggested in Fig. 4).


Microsoft Word - masterv10.doc
File Edit View Insert Format Tools Table Window Help
100%
Coverpage
Times New Roman
12
Odust: ODU Sharing Tool
Sharing: Microsoft Word - masterv10.doc
Leader: Cecilia
Refresh
Shared tool #1
Owner: Eduardo Leader: Agustin
xterm

```
A·SEMANTIC-BASED·MIDDLEWARE·FOR·MULTIMEDIA↵

COLLABORATIVE·APPLICATIONS¶

by¶
¶
¶
Agustín·José·González¶
M.Sc.·Computer·Science,·December·1997,·Old·Dominion·University¶
M.Sc.·Electronic·Engineering,·December·1995,·Universidad·Federico·Santa·María,·Chile¶
B.Sc.·Electronic·Engineering,·December·1986,·Universidad·Federico·Santa·María,·Chile¶
¶
¶
A·Dissertation·Submitted·to·the·Faculty·of·↵
Old·Dominion·University·in·Partial·Fulfillment·of·the↵
Requirement·for·the·Degree·of¶
¶
DOCTOR·OF·PHILOSOPHY¶
¶
COMPUTER·SCIENCE¶
¶
OLD·DOMINION·UNIVERSITY¶
May·2000¶

Page 1   Sec 1   1/153   At 5.3"   Ln 21   Col 1   REC TRK EXT OVR WPH
```

```
  PINE 4.05   MESSAGE TEXT               Folder: phd  Message 79 of 108 97%

The big issue is that even with qb it is hard to get remote people to
ask questions -- just as it is hard to get people in a classroom to ask
a question. We really need a "sense of presence" for remote people
(i.e., a display in the room of the remote participants).
--
>
> Thank you very much in advance,
>
> Agustin J. Gonzalez
> PhD Student
> Old Dominion University
> Norfolk, Virginia
>
> P.D. I am currently working on a Java-based protocol for floor control. I use
> IP multicast -unreliable- and temporary TCP connections.
--
Interesting, send email when it is published so we can try it out.
        Larry


--
Professor Lawrence A. Rowe          Internet:  Rowe@BMRC.Berkeley.EDU
Computer Science Division - EECS       Phone: 510-642-5117
University of California, Berkeley       Fax: 510-642-5615

? Help        < MsgIndex   P PrevMsg      - PrevPage D Delete      R Reply
O OTHER CMDS  > ViewAttch  N NextMsg    Spc NextPage U Undelete    F Forward
```

Fig. 5. The real MS-word application and Odust interface viewed by Rodrigo.

Rodrigo shares an MS-Word application, as shown in Fig. 5. MS-Word runs outside Odust the same way it does any application on his machine. In addition, he receives the *xterm* being shared by Eduardo (owner label) but controlled by Agustín (leader label). Even though the *xterm* here is a UNIX application, it runs via an X Window-server on WinNT. Rodrigo selects what to share from the upper menu of Odust. On this widget, he also learns who has the floor of the tool he shares, Cecilia at this time.

If other WinNT participants started sharing more applications, each participant would receive them in separate windows within Odust. This is the case of UNIX users in this scenario. They receive Rodrigo's MS-Word and Eduardo's xterm in different windows.

Finally, floor control is done on per shared tool bases. This feature enables collaboration at a level it cannot be reached even in face-to-face encounters when two people sit in front of the same computer. We could have this type of view on a single computer screen; nevertheless, we cannot use the computer's keyboard and mouse to simultaneously operate both applications.

4.2. Overall Architecture

Odust's architecture reflects the three main external features of it, application view dissemination, floor control, and remote tool interaction. A distributed object architecture implements the protocol for transmission of dynamic compound images. Another set of distributed object implements a lightweight floor control framework for centralized resources, which is described in [8]. Finally, two objects work in a client-service architecture to support the interaction with the shared application from remote sites. Odust depends on a single multicast group that is provided as command line argument. Now, in order to support multiple shared applications at a time, Odust multiplexes the multicast group in up to 256 channels. A distributed multiplexer-demultiplexer object dynamically manages channel allocation as new applications are shared. Each of the basic components of Odust, compound image transmission, floor control, and user's input events is made of two related objects. One centralized object resides on the machine sharing a tool and the others are replicated at every shared tool receiver. Fig. 6 illustrates a situation where multiple applications are shared. Although a machine that shares a tool can also receive others coming from other sites, we have logically divided Odust in a sender and a receiver component for description purpose.

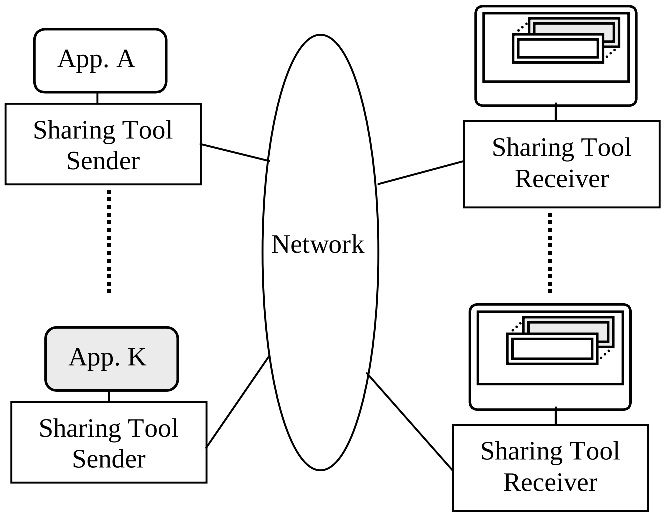


Fig. 6. Odust distributed logic modules.

While Fig. 6 shows the interactions between multiple senders and receivers, Fig. 7 focuses on the internal architecture of one sender and one receiver. All the objects of the sender are instantiated at execution time; however, only the demultiplexer remains up all the time at receiving sites. The demultiplexer listens for messages coming on any channel. Multiplexer (Mx in Fig. 7) and demultiplexer (Dx in Fig. 7) are actually two Java interfaces implemenmted by the same class. Thus, each multiplexer can keep track of the channel in use and can randomly allocate a new unused channel when the local sender requests one to start transmitting a new shared tool to the session. As soon as its counterparts receive an Application data Unit (ADU) from an unallocated channel, each receiver creates new *application receiver* object to process subsequent ADUs.

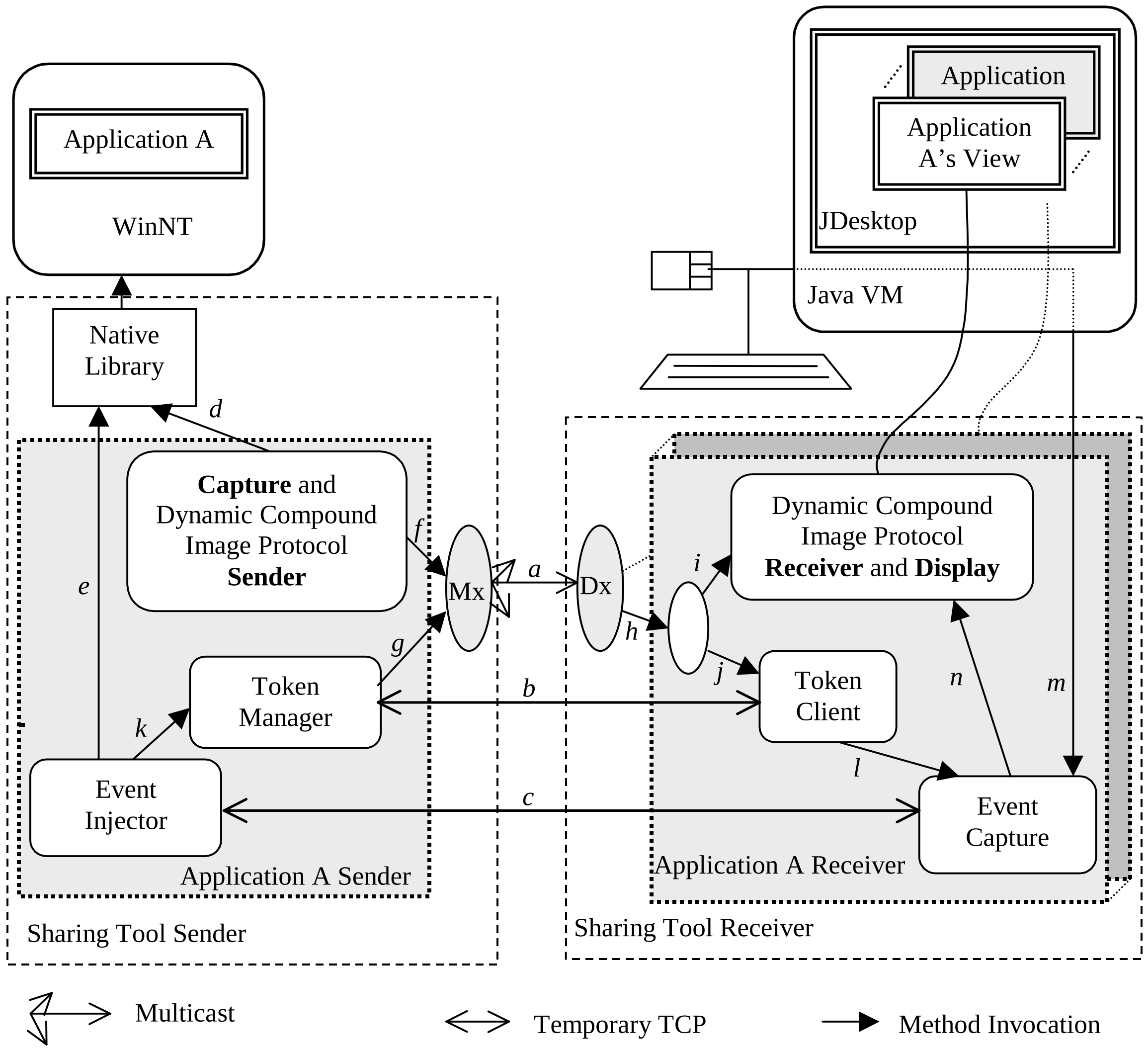


Fig. 7. Odust sender/receiver overall architecture.

Senders blindly transmit ADUs with no feedback from recipients. Both the image transmission and Token Manager objects share the same multicast channel. While the former transmits image protocol related messages, the latter periodically sends a heartbeat with the floor status (mainly floor holder), local host names, and Token Manager service port, so that clients can dynamically connect to the Token Manager (link *b* in Fig. 7). The native library implements three functions required by the Java capture object.

All the images of the shared application are sampled and transmitted using the protocol for compound image transmission described in an earlier section. At the receiving site, the demultiplexer dispatches the ADUs to the corresponding *application receiver* according to the setting it saves when the *application receiver* is created upon receiving the first ADU (method call *h* in Fig. 7). Then, the *application receiver* dispatches the message to either the compound image receiver (method call *i*) or the Token Client (method call *j*).

The Token Manager and Token Client have graphics interfaces. Upon user's floor request, the Token Client connects to the manager and obtains the service access point of the Event Injector in the Grant message (*b* connection in Fig. 7). The Token Client forwards this information to the Event Capture object (method call *l*) and updates its interface. Finally, connection *c* is established and the mouse and keyboard events of the new floor holder are sent to the application sender.

Connection *b* and *c* are only kept while the corresponding receiver holds the floor. The Event Capture object listens for input events within the application widget at receiving sites (method call *m*). When an input event is fired by the Java virtual machine, Event Capture forwards the event to its peer Event Injector as long as the event took place within one of the shared application images in the widget. This confirmation is done by a call to the compound image receiver object (method call *n*). This check suppresses events that do not fall into any image even though they are detected within the display widget. The compound image receiver detects when all the windows of the application are destroyed or no tile refresh has taken place after a timeout. It releases all the allocated resources by unbinding the *application receiver* from the channel demultiplexer and locally removing any graphics object of that application.

The Native Library is the only non-Java code. It implements 5 native methods that need to be ported to other platforms in order to share applications running on them; however, receiver's code has the same portability as Java code.

Even though the traffic due to the floor holder only affects two machines per floor in the session, we use mouse event filtering to reduce the number of events fired by mouse moves. Mouse movements are only sent to the application if they are far part in position or time. Two parameters govern the granularity of the filter.

5. Related Work

Important sharing tool applications like XTV [1], JCE [2] and VNC [21] use TCP as transport protocol. In contrast, our protocol works on top of unreliable multicast transport layer. This makes a crucial difference that lets our protocol be considerably more scalable than the other proposals. Habanero [9] is a Java-based framework for synchronous and asynchronous collaboration. This framework facilitates the construction of software for synchronous and asynchronous communication over the Inernet. It also provides methods that developers can use to convert existing Java applications into collaborative applications. The system employs a centralized architecture and utilizes TCP connections between each client and the central server. Platform independence is gained by using Java. Odust shares with Habanero its object-oriented approach and programming language; nevertheless, Odust supports a much general model for application sharing and higher scalability level.

The idea of sharing data by sharing images has been explored in the VNC project [21] at Cambridge University. While Virtual Network Computing proposes image distribution over reliable transport protocol, specifically TCP, our protocol also works over unreliable channels. We believe our protocol can handles larger groups and provides better responsiveness than VNC. VNC's unit of transmission is, like our protocol, the distribution of rectangle of pixels at a given position. It uses raw-encoding or copy-rectangle encoding. In the first one, the pixel data for a rectangle is simply sent in left-to-right scan-line order. In contrast, we use still image compression for tiles. VNC avoids compression time but demands more transmission bandwidth than our protocol. Copy-rectangle encoding allows receivers to copy rectangles of data that are already locally accessible and can be used for motion prediction on synthetic images (e.g. scrolling). We decided against this type of primitive because of the high processing cost in determining tile motion. While VNC shares an entire common desktop, we propose an application granularity for sharing which allows a participant to show a selected application and keep the rest private. Odust also avoid configuring and need of more licenses when one have to install applications on a centralized server like in VNC. Instead, Odust allows any user to share her local platform dependent application.

Video Conferencing tool has also been used for data sharing by transmitting dynamic images as video frames. Its main advantage is the access to highly refined and tuned libraries for video streaming that reach higher frame rate than image processing. In fact, there is experience in its use in the MBone [15]. Lawrence Rowe, at University of California at Berkeley, has been using video technology to deliver data information in the Berkeley Multimedia, Interfaces, and Graphics Seminar (MIG). There, they either use a scan converter to translate the computer screen signal into standard video format or employ a stand camera to capture hard-copy slides. While a first video stream is reserved to the presenter's video, the second one sends the computer screen from the converter and using H.261 format [11]. Another experience in sending data contents through video streams is found in *vic* version 2.8 [23] from University College London (UCL). One of the featured added at UCL allows the sender to select a region of the screen for frame capture as opposed to video frames. The video approach fulfills reasonably well the need for data distribution in many cases, especially under the lack of general-purpose alternative; nonetheless, this technique suffers from a number of shortcomings. First of all, video compression limits the video dimensions to a few sizes. This restricts its application when the information to be shared does not fit a predefined video size on the screen. On the other hand, the use of converters for sending the entire display view forces the sender to make the complete screen public. In addition, it inevitably reduces the resolution to, for example, 352x288 pixels for CIF (**C**ommon **I**ntermediate **F**orm) size video. The inevitable electronic thermal noise introduces fictitious changes in the captured digital image and, therefore, leads to more data traffic. In addition, such conversion leads to loss of legibility which is a critical shortcoming for many types of synthetic images. Our protocol for transmitting dynamic images overcomes these shortcomings.

6. Conclusions and Future Work

Along with audio and video, data sharing is a crucial component in multimedia collaboration. In order to achieve data sharing, we developed a protocol for image transmission and used it to implement Odust, a sharing tool application. This resilient and scalable protocol compresses a sequence of image samples by removing temporal and spatial redundancy. Tiling and changes detection achieve the former, and a standard image compression technique accomplishes spatial redundancy removal. Protocol data unit losses are overcome by randomly re-transmitting tiles. This technique also provides support for latecomers. We conducted an extensive study on the sensitivity of the dominant parameters of the protocol. These included tile compression format, tile size, sampling rate, and tile change detection technique.

This sharing tool application disseminates images of the shared application and accepts remote user input events as if they were coming from the local tool owner. It was tested on Win85, Win98, WinNT, and Solaris operating systems.

This work can be extended in two independent paths. One aims to reduce both processing time and bandwidth consumption of the protocol. The other approach is to adapt current video compression techniques to fulfill the requirements of data sharing. We are investigating the use of timeout in the protocol sample processing to reduce computation time and bandwidth. We are also considering H.263+ [6] video compression standard, which supports custom picture size. This feature removes one of the major drawbacks we have pointed out of video encoding and enables it for sharing images. Finally, we plan to port the sampling library to other platforms to not only receive but also transmit application's views from other platform.

References

[1] H. Abdel-Wahab and M. Feit, “XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration,” in *IEEE Tricomm '91: Communication for Distributed Applications & Systems*, Chapel Hill, NC, USA, 1991. IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 157-167, 1991.

[2] H. Abdel-Wahab, O. Kim, P. Kabore, and J.P. Favreau, “Java-based Multimedia Collaboration and Application Sharing Environment,” in Proceedings of the Colloque Francophone sur I’Ingenierie des Protocoles (CFIP ’99), Nancy, France, April 26-29, 1999.

[3] C.Bisdikian, S. Brady, Y.N. Doganata, D.A. Foulger, F. Marconcini, M. Mourad, H.L. Operowsky, G. Pacifici, and A.N. Tantawi, “Multimedia Digital Conferencing: A Web-enabled multimedia Teleconferencing system,” *IBM Journal of Research and Development*, vol. 42, no.2, pp. 281-298, March 1998.

[4] T. Boutell, “PNG (Portable Network Graphics) Specification: Version 1.0,” Request for Comments RFC 2083, January 1997.

[5] D.D., Clark and D. Tennenhouse, “Architectural considerations for a new generation of protocols,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, Philadelphia, Pennsylvania, IEEE, pp. 200-208, Sept. 1990.

[6] G. Côté, B. Erol, M. Gallant, and F. Kossentini, “H.263+: Video Coding al Low Bit Rate,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 849-866, November 1998.

[7] J.Z. Davis, K. Maly, and M. Zubair, "A Coordinated Browsing System”, *Technical Report TR-97-29*, Old Domimion University, Norfolk, VA, May 1997.

[8] A. González, "A Semantic-based Middleware for Multimedia Collaborative Applications," Old Dominion University, Norfolk, Virginia, Ph.D. dissertation, May 2000.

[9] HABANERO, On-line from: http://www.ncsa.uiuc.edu/SDG/Software/Habanero.

[10] H. Imai and T. Asano, "Efficient algorithm for geometric graph search problems," *SIAM Journal on Computing*, vol. 15, pp. 478-494, 1986.

[11] ITU Telecommunication Standardization sector of ITU, "Video codec for audiovisual services at p x 64 kbit/s," *ITU-R Recommendation H.261*, March 1993.

[12] W.T. Liou, J.J. Tan, and R.C.T. Lee, "Minimum Rectangular Partition Problem for Simple Rectilinear Polygons," IEEE Transaction on Computer-Aided Design, vol. 9 no. 7, pp. 720-733, 1990.

[13] W. Lipski, E. Lodi, F. Luccio, C. Mugnai, and L. Pagni, "On two-dimensional data organization II," *Fundamenta Informaticae*, vol.2, no. 3, pp. 245-260, 1979.

[14] K. Maly, H. Abdel-Wahab, C.M. Overstreet, C. Wild, A. Gupta, A. Youssef, E. Stoica, and E. Al-Shaer, "Distant Learning and Training over Intranets," *IEEE Internet Computing*, vol. 1, no.1, pp. 60-71, 1997.

[15] MBone http://www.mbone.com/.

[16] Microsoft's NetMeeting,http://www.microsoft.com/netmeeting.

[17] Microsoft's PowerPoint,http://www.microsoft.com/powerpoint.

[18] S.A. Mohamed and M.M. Fahmy, "Binary Image Compression Using Efficient Partitioning into Rectangular Regions," *IEEE Transactions on Communications*, vol. 43, no. 5, pp. 1888-1893, May 1995.

[19] J. Nonnenmacher and E.W. Biersack, "Scalable Feedback for Large Groups," *IEEE/ACM Transactions on Networking*, vol. 7 no. 3, June 1999.

[20] T. Ohtsuki, "Minimum dissection of rectilinear regions," In Proceedings *IEEE International Symposium on Circuits and Systems*, New York, USA, vol. 3, pp. 1210-1213, 1982.

[21] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper, "Virtual Network Computing," IEEE Internet Computing, vol. 2, no.1, pp. 33-38, Jan/Feb 1998.

[22] Sun Microsystems, Java language, http://java.sun.com/products.

[23] Videoconferencing Tool, The Networked Multimedia Research Group at University College London,

http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/.

[24] G.K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, vol. 34, no.4, pp. 30-44, April 1991.