XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration

Hussein M. Abdel-Wahab
Mark A. Feit
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

email: wahab@cs.odu.edu

Abstract

XTV is a distributed system for sharing X Window applications synchronously among a group of remotely-located users at workstations running X and interconnected by the Internet. The major components of the system are designed and implemented in such a way that make them reusable in other collaborative systems and applications. This paper describes the fine technical details and knowledge required to understand and replicate the work which went into developing XTV. The following concepts are discussed: interception, distribution and translation of traffic between X clients and display servers; regulation of access to tools using a token passing mechanism and reverse-translation of server traffic; and accommodation of systems with differing architectures which may have different byte orders for integer representation.

1 INTRODUCTION

With the proliferation of high-bandwidth computer networks and the increasing popularity and affordability of powerful workstations, it is now feasible to provide the general user community with the environment and tools to work together synchronously in real-time and each one can see what other collaborators can see. This mode of collaboration is known in the CSCW (Computer Supported Cooperative Work [Greif 88]) field as WYSIWIS (What You See Is What I See [Stefik 86]). We started our research in synchronous real-time group collaboration by building the Remote Shared Workspaces (RSW) system as a research vehicle [Abdel-Wahab 88]. RSW provides the large community of UNIX users linked by Internet [Comer 90] with a general-purpose facility that effectively converts a single-user software tool into one that can be used for real-time collaboration by a group of remote users. The RSW system is restricted in that textual applications (vi and emacs, for example) could operate with it. This meant that output could contain no graphics and input could only be taken from the keyboard. RSW's major advantage, however, is that a user equipped only with a dumb terminal can participate and collaborate with others.

Our current system, called X Terminal View (XTV), is an extension of RSW using MIT's X window system [Scheifler 86] where individual X clients (also called applications or tools) can be shared. X is gaining ground as a standard windowing system and is currently supported by most leading workstation manufacturers including Sun, DEC, IBM, HP, Apple, AT&T. Because it is designed as a network-transparent, device-independent windowing and graphics system, X completely achieves its device independence by splitting the job of drawing windows into two parts using the increasingly familiar client/server model. One or more clients, connected through a network, communicate with the server by sending packets of instructions and data (called requests) conforming to the X protocol [Nye 89]. X can be used for truly distributed systems since a client running on one workstation can open windows on remote workstations which may have been made by different manufacturers. It is these aspects of X that make it an appealing and valuable system for supporting group collaboration.

XTV has been designed in such a way that most of its components can be reused in building other collaborative tools. For example, the core routines which handle the translation of X protocol requests and replies can be easily extracted and implanted in other programs which might require this capability. XTV and all of the methods developed during its implementation are in the public domain, and all attempts have been made to reveal as many of the its operational details so that others may replicate our work or use it in other contexts.

Section 2 explains how XTV collaborative sessions are started and how X clients are invoked during the session. This section also describes the user interface to the system and the relationship between the distributed processes created during a session and the commu-
nication paths between them. Section 3 describes the process of intercepting traffic between clients and servers and the methods used to examine this traffic and route it to the proper destinations. Also outlined are the procedures required for overcoming the differences in the operational characteristics used by different X servers. Section 4 details the translation required to make request packets intended for one server indicate the correct actions to another. The reverse process, translating events and replies from multiple servers to provide the right information to a single client, is shown next. Also described here are the methods used to overcome the problem of communicating with processes on different hosts that may use different byte orders to represent integer quantities. Finally, Section 5 outlines the conclusions drawn from the development of XTV and the plans for its future extension.

2 SYSTEM DESCRIPTION

2.1 Starting a New Session

Figure 1 shows the overall structure, relationship and communication paths between different processes involved in a given collaborative session. To start a session, one user, designated the “chairman,” creates a “Session Manager” process. This process is invoked on the local host or any remote host able to connect to the local host’s display server. The Session Manager listens for connections from “Participant Agent” processes at a TCP/IP port which is well-known to all other copies of XTV. A participant joins an ongoing session by creating a Participant Agent process and specifying as arguments the host and session ID number (referred to hereafter as SessionNumber) of the Session Manager.

Each participant is presented with a “Session Panel” similar to that shown in Figure 2. The title section contains information about the participant (e.g, the name is wahab, connected to session number 2 at 9:15am from host seth.cs.odu.edu). On the session panel are three buttons:

- New Tool - Brings up a panel used to start a new X client (tool) to be shared among participants in the session.

- Leave Session - Leaves the session without any disruption to other participants.

- End Session - Brings the session to an end by terminating all tool and agent processes, destroying windows at all hosts associated with it and closing all network connections.

The Participants area of the panel contains a list of all of the session’s participants. The user can open a window containing pertinent information on a particular participant by clicking on his or her name. By clicking on the Talk button, a private, one-to-one conversation can be carried on with that participant.
Figure 1: XTV Processes and Communication Paths
Figure 2: User Interface to XTV
The **Tools** area contains the names of all tools being used in the session. Clicking on the name of a tool opens a "tool panel" window with some of the tool's operating information and mechanisms for manipulating its associated "token." By using a first-come, first-served method for handling the token, regulation of which participant may provide input to each tool is achieved. Several buttons allow participants to control a tool and its token:

- **Get Token** - Places the participant at the end of the queue to get the token. If no participant possesses the token, it is passed immediately.

- **Urgently Get Token** - Bypasses the token queue and passes the tool's token directly to the participant. At present, this function is restricted only to the session chairman.

- **Drop Token** - Passes the token on to the next participant in line for the token. If the token queue is empty at that point, none of the participants may provide input until someone clicks the **Get Token** button.

- **Leave Tool** - Disconnects the participant from the tool. When this happens, the participant's windows associated with the tool are closed while others continue to use the tool.

- **Join Tool** - Allows a participant to join a tool already in progress. This function is not currently supported, as a great deal of state information maintained by the server would need to be kept by XTV as well. The ability to acquire the information stored by the server will become practical only when a future version of the X protocol supports it.

- **End Tool** - Terminates the tool and all associated processes, windows and connections to all participating hosts.

The **Token Queue** area of the tool panel indicates which users are participating in this tool, who possesses the token and who is queued for it. Note that it is not necessary for all session participants to join a given tool, some of them may elect not to participate in some of the invoked tools.

### 2.2 Starting a New Tool

To start a new tool, one of the participants clicks on the **New Tool** button of the session panel. Through a pop-up dialog box, the name of the tool, its command line arguments and place of execution are specified. XTV executes tools at the chairman's location or on systems accessible through the Internet. Starting a new tool creates the following processes (See Figure 1):
• **Packet Switcher** - Responsible for handling the initial connection from the X client to the chairman’s X server and to all of the participants’ packet translators. Once those connections are established, the packet switcher handles the distribution of all traffic from the client to the server and the other packet translators as well as multiplexing input from all servers back to the client. This process interacts with the token manager (below) to determine the disposition of incoming traffic from the servers.

• **Packet Translator** - Converts client-to-server and server-to-client traffic from one server’s format to that of another. Any packet received from the Packet Switcher is “forward” translated and sent to its local server. If the associated participant has the token, then any event or reply received from its local X server will be “reverse” translated and sent to the Packet Switcher. (The translation processes are described in Section 4).

• **Token Manager** - Accepts input from all token agent processes (below) and determines the disposition of the token, which can be accomplished using any desired token management protocol [Abdel-Wahab 88] and informs the packet switchers of its new state.

• **Token Agent** - Maintains the token panel and implements all functions associated with the panel buttons by interacting with the token manager.

### 3 CONNECTION SETUP

#### 3.1 Intercepting traffic between X clients and servers

IN X, the specification of the display server that a client is to use takes the format `hostname:display,screen`. The `hostname` is a standard Internet host name such as `seth.cs.odu.edu` or `128.82.7.4`. Generally, `display` and `screen` are both zero, indicating that logical screen number zero of physical display number zero is to be used. This information is either placed in the `DISPLAY` environment variable or specified using arguments supplied on the command line.

The X display server accepts connections from clients by listening to a TCP/IP socket at a port well known to all clients (defined by the protocol as `X_TCP_PORT`). Clients contact the server waiting at `X_TCP_PORT + DisplayNumber`. Thus if `DisplayNumber` is 0, the client contacts the X display server. XTV’s approach to intercepting the traffic between the client and server is based on providing false display information to the client (by way of `DISPLAY` environment variable). A TCP/IP socket at the port the client will attempt to connect with (using its “false” information) is opened by the XTV packet switcher, and thus the client connects to a packet switcher instead of the “real” X server. For example, if XTV is running on host `seth` and its `SessionNumber` is 2, XTV will open its port at `X_TCP_PORT + 2` and
set DISPLAY to seth:2.0. Using this example, XTV would, in effect, perform these csh commands to invoke xcalc:

```
% setenv DISPLAY seth.cs.odu.edu:2.0
% xcalc
```

### 3.2 Connection Request and Reply

Once a connection is established with a client, the packet switcher waits for a “connection request” packet, the first data sent by clients conforming to the X protocol. This packet contains the following information:

- **ByteOrder** - A one-byte value indicating how integer quantities are to be interpreted by the display server. This is to accommodate architectures which have varying methods of storage. The two alternatives are LSB first (called big endian and used by IBM, Sun workstations) and MSB first (called little endian and used by DEC workstations).

- **Protocol Version** - Indicates to the server which version of the X protocol it is expecting the server to use (e.g., Version 11, Release 4).

- **Authorization Data** - Designed for use in authentication of server connections but considered meaningless in the present version of the protocol.

The chairman’s packet switcher accepts this information and forwards a copy it to its X server (called the primary server) and to the packet translators of all participants. Each packet translator in turn forwards this packet to their own X servers (called local servers).

If an X server accepts a connection request, it replies by sending back a “connection acceptance” packet, from which the chairman’s packet switcher extracts and passes on the following information to the participants:

- **Resource ID Base**
- **Resource ID Mask**
- **Root Window ID**
- **Default Colormap**
- **Root Visual**
- **Visual ID List**
This information is used in translating requests and events from the local server’s format to that of others and back. (The details of this procedure are described in section 4)

When a participant’s packet translator receives a connection acceptance packet from its local server, the same information is extracted and stored for use in translation. Since there is no “real” client to forward this information to, the packet is then discarded.

4 PACKET TRANSLATION

4.1 Translating Resource IDs

X identifies server-stored abstractions (such as windows, colormaps and atoms) using a 32-bit identifier called a resource ID. Resource IDs are assigned by the client using a base value and bitmask provided by the server during the connection process. The bitmask indicates which bits may be used by the client in creating a resource ID which is unique. Once those bits have been filled in, the base value is overlaid and the completed ID is passed to the server.

By making use of the values and masks of the primary server and a local server, it is possible to translate resource IDs from one to the other without sacrificing its uniqueness. This is accomplished by removing the base value from the source ID, by ANDing it with the server’s bitmask and ORing the result with the target ID and overlaying the target server’s base value.

For example, assume the source server has provided a client with a resource ID base of 0xabc00000 and a bitmask of 0x000fffff and our target server has provided 0xdef00000 for the base value. Using the above steps, the resource ID 0xabc12345 would translate to 0xdef12345.

It is worth mentioning that not all resource IDs are translated in this manner. Certain items such as the root window, default colormap and visual IDs are assigned by the server and do not necessarily conform to a particular base value and bitmask. To accommodate these cases, the necessary values are extracted from the connection acceptance packet and stored for use by the translation routine. In the interest of speed, incoming resource IDs are first checked to see if there is a match between it and the source server’s base value. If there is, the procedure described above is used. Otherwise, the window, default colormap and visual IDs are checked manually and the corresponding value for the target server is returned.

4.2 Translating Client Requests

All requests submitted to the X server by clients begin with a header whose format is specified by the X protocol. In this header is the following information:
- **Major Opcode** - An unsigned short integer (one byte) indicating which server function is being requested. Values 0 to 127 represent the standard functions built into all servers. Those numbered 128 through 255 are reserved for implementation-specific extensions.

- **Auxiliary Field** - An unsigned short integer (one byte) used as a data field in requests which pass only one byte of data aside from the major opcode and request length. This eliminates the need for extra network traffic to complete short requests.

- **Request Length** - An unsigned integer (two bytes) indicating how many four-byte units of data the request contains, including the header, which is four bytes long.

Once a complete request arrives at the chairman’s packet switcher, copies are distributed unmodified to the primary server and the packet translators of all participants. At the participant’s end, the contents of the request header are examined and one of the following actions is taken to translate the packet from the primary server’s format to that of the local server:

- **No Translation** - Requests such as Bell (shown Figure 3 a in) which contain no resource IDs require no translation and are passed unmodified to the local server.

- **Simple replacement** - Requests such as DestroyWindow (shown in Figure 3 b) contain resource IDs in fixed locations which can be translated in a straightforward manner using the methods previously described.

- **Complex replacement** - Requests such as CreateWindow (shown in Figure 3 c) have a variable length which is determined by the contents of certain elements in a part of the packet which is known to be fixed. In these cases, a special function capable of handling that particular request is called. For example, the VALUE LIST of the CreateWindow packet shown in Figure 3 c requires special processing (the number of elements of the list is equal to the number of 1s in the value mask field).

For the sake of efficiency, the entire request translation operation is table-driven. When a packet arrives, its opcode is used to directly access a table containing information on what action should be taken. Each record in the table contains the following fields:

- **Packet Name** - An ASCII representation of the standard name of the request as assigned by MIT. The contents of this field are presently used for debugging purposes only.

- **Resource ID Offsets** - A zero-terminated list of the offsets (in units of four bytes) inside the request containing resource IDs, which are always at a four-byte boundary. Negative numbers indicate that a resource ID found is to be left alone if the value of the resource
<table>
<thead>
<tr>
<th>opcode</th>
<th>unused</th>
<th>length</th>
<th>WINDOW ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

(a) BELL request

<table>
<thead>
<tr>
<th>opcode</th>
<th>depth</th>
<th>length</th>
<th>PARENT WINDOW ID</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>8+n</td>
<td>width</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>height</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>border width</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>class</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WINDOW ID</th>
<th>VISUAL ID</th>
<th>VALUE LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4n bytes

(b) DESTROY WINDOW request

<table>
<thead>
<tr>
<th>opcode</th>
<th>width</th>
<th>height</th>
<th>border width</th>
<th>class</th>
<th>value mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4n bytes

(c) CREATE WINDOW request

Figure 3: Examples of X Protocol Request Packets
ID is 0 or 1, translated otherwise. If there are no replacements to be done, the first item in the list is a zero.

- **Special Function** - A pointer to a function designed for handling translation of a particular type of request. If no special translation is required, this pointer is **NULL**. In the case of requests which are to be eliminated entirely (such as those which would generate useless replies at a participant’s local server), a function called **ForceDropDead** is called. **ForceDropDead** replaces the request’s major opcode with 127 (the NoOperation request), and the packet is not transmitted to the participants.

For example, the entry numbered 001 in the table contains the following:

```
{ "CreateWindow", {1,2,-6,0}, XlateCreateWindow }
```

When a *CreateWindow* request arrives from the Packet Switcher, its record is retrieved from the table. The Resource IDs at offsets of 4, 8 and 24 bytes will be translated to match the local server, with the Resource ID at bytes 24-27 remaining untranslated if they are 0 or 1. Once the fixed translations are complete, the request is passed to a function called **XlateCreateWindow** which handles all translations on the variably-sized portion of the request.

### 4.3 Translating Server-Generated Events and Replies

Much like requests passed from the clients to the X server, events and replies generated by the local servers of the participants must be translated from the local format to that used by the primary server. The same methods used in translating requests are applied in reverse, the only significant difference being that a different table is used to determine the course of action.

Processing server-generated events and replies requires special consideration on the part of the participants’ packet translators. Events must be carefully filtered to avoid confusing the X client program. Generally, the one variable governing how events are filtered by a particular participant is whether or nor it has the token. When the token is not available, these types of events are discarded:

- Keyboard activity
- Pointer activity
- Configuration notification
To maintain complete, accurate displays of windows for all of the participants’ screens, exposures are always propagated.

To avoid irrational behavior on the part of XTV clients, participants never pass any of these events back to the client:

- Reparenting notification
- Keymap notification

As each event arrives, any fields containing resource IDs must be translated from the format used by the participant’s local server to that of the primary server. This is easily accomplished, as the participant already has copies of both formats on hand and can pass them to the resource ID translator in “reverse” order (i.e., passing the local server’s information to the resource as the “source” argument and the primary server’s information as a “target” argument). This effectively “back-translates” each of the resource IDs contained in the event. On a larger scale, the translation of event packets is handled in much the same way as requests. A separate translation tables for events and replies is maintained and like the first, it provides the same information about the locations of resource IDs within the packet and pointers to special functions required for processing particular events and replies.

4.4 Byte Order Interpretation

Under the rules of the X protocol, client processes may run on machines with differing byte orders for the representation of integers.

For Example, we may run an X client on a DECStation (MSB first), the chairman’s packet switcher on an IBM RISC/6000 (LSB first) and a participant’s packet translator on a Sun SPARCStaion (also LSB first). In such a heterogeneous environment, each process must correctly interpret the integer quantities it receives from the others.

At startup, the packet switcher finds the byte order used by the host on which it is running. This is accomplished by taking known integer value and examining the way it is stored as bytes.

XTV takes a hands-off approach to handling the byte order of each tool. Whenever the need arises to interpret one of the client’s integers, the value and the client’s byte order are first passed to one of SwapInt() or SwapLong() for processing before the value is placed elsewhere and then actually used. If there is a difference between the local and client byte orders, the contents of the integer are rearranged to properly express the integer locally.

In addition to dealing with each tool’s requests and events by using its byte order as a guide, XTV also makes use of that information to facilitate communication between the chairman’s packet switcher and the participants’ translators. Since situations requiring the
transfer of integers only arise when dealing with tools, all communications are done in the byte order of the particular tool in question. By doing this, the need for the chairman’s packet switcher to be aware of the byte orders of the participants and vice-versa is eliminated.

5 CONCLUSIONS AND FUTURE GOALS

This paper has presented a framework to allow a group of users to join in a collaborative session for viewing and interacting with X-based, single-user applications. In effect, XTV converts the tools we are all familiar and comfortable with from single- to multi-user without modifications to source code, libraries, application behavior, the operating system, or X servers.

The token and participant management policies are designed as independent components of the system which can be modified to suit the requirements of the group. The packet translation tables and routines were designed for efficiency and need not be changed as long as there is no change in the X protocol. However, they can be easily extended to include any extensions to the protocol, such as the addition of new request types.

XTV is by no means complete. In its present state, it is not much more than a working prototype which provides the features described here. The current system is vulnerable to a failure of the chairman’s packet switcher or token manager, either of which will bring the tool to a halt. One possible way of dealing with this problem is the implementation of a replicated tool architecture similar to the ones described in [Abdel-Wahab 88, Lantz 86].

XTV’s performance in its current implementation has been more than adequate on medium-speed local area networks. A test of the system over a FDDI network is planned for the future, and it appears that these higher-bandwidth channels will result in a better performance and help to eliminate the occasional time lag experienced.

A major function remaining to be implemented is to allow users to join a tool after it has already started. As discussed earlier, future extensions to the X protocol should make this facility not only feasible but practical as well.

ACKNOWLEDGEMENTS

We want to extend special thanks to Don Smith, Peter Calingaert and Kevin Jeffay of UNC for their invaluable assistance and encouragement at various stages of this project. We are grateful to Alan Blatecky and Tom Sandoski of MCNC for testing and experimenting with different versions of XTV. We also would like to thank Ashit Patel, John Menges,
Alok Ramsisaria, Jin-Kun Lin and Jie-Shan Lin of UNC, and Osman ZeinELDine, Mohamed ElToweissy and Rafat Shaheen of ODU for their contributions and comments on various aspects of XTV.

References


