A Survey of X Protocol Multiplexors

John Eric Baldeschwieler, Thomas Gutekunst, Bernhard Plattner
<gutekunst@tik.ethz.ch>, <plattner@tik.ethz.ch>

Swiss Federal Institute of Technology
Computer Engineering and Networks Laboratory (TIK)
ETH-Zentrum, ETZ
CH-8092 Zürich, Switzerland

Abstract

An X multiplexor allows a single X Window System client to be displayed and interacted with on several X servers simultaneously. Such a service is necessary for the construction of a computer-supported cooperative work (CSCW) environment such as JVTOS (Joint Viewing and Tele-Operation Service) which is being implemented within RACE II project CIO[1]. This paper describes several existing X multiplexors and evaluates their usefulness for JVTOS.

Keywords

Application Sharing, Computer Conferencing, Computer-Supported Cooperative Work (CSCW), Distributed Systems, Joint Viewing and Tele-Operation Service (JVTOS), X Protocol Multiplexer, X Window System.

1 Introduction

A computer-supported cooperative work (CSCW) environment requires *joint viewing*. This allows multiple users, each on his own computer workstation, to view and interact with a single application. One solution to this problem is to build a new set of cooperation-aware applications which explicitly support this requirement. Such an approach has several problems. Perhaps the most critical of these is that users would be limited to the use of only special cooperation-aware applications. Given the diversity of computer applications available, this requirement appears very limiting.

An *X protocol multiplexor* (MUX) is another solution to the joint viewing problem. A MUX is a special program which exploits properties of the X Window System to allow joint viewing with unmodified X applications (clients). Such an approach is called *application sharing* and has several advantages. First, users are not required to use new applications, they can share their existing applications. Also, the joint viewing system does not need to be modified to incorporate new applications or changes to existing applications. And finally, the task of developing a CSCW environment will be greatly reduced. Instead of reimplementing many existing programs, the developers need only implement the MUX program.

The X Window System uses a client/server model. Each workstation is an X server which presents information to the user and receives his responses. Each application program which receives user input or generates output is an X client. Clients and servers communicate using the X protocol through a network connection. This allows a client to execute on a different machine than its user interface, which is presented by its server. The X protocol describes a list of requests, events, responses, and errors that can be sent over a client/server connection. A client sends a series of requests to its server. The server then performs the requested actions, such as reporting its status, or drawing windows and their contents. An X server sends a series of responses and events to each of its clients. Responses contain answers to client requests, and events allow the server to report user activity such as key presses and mouse movement. Because all user interface activity of an X client must pass through the client/server connection, it is possible to manipulate this stream with a MUX to provide joint viewing functionality.

A MUX must provide three services: *connection*, *multiplexing*, and *filtering*. Connection consists of intercepting the X client/server connection. A MUX appears to be a normal server to X clients. When a client connects to such a MUX, the MUX then connects to several servers. To each server the MUX appears to be a client. After the connection has been established, the MUX sends a copy of every client request to each connected server. This action is called multiplexing. Responses, events, and errors from all of the servers are then collected by the MUX and reported to the client as if they come from a single server. This collection must involve filtering of unwanted responses and events. For ex-

[1] Participation in CIO was financed by the Swiss Confederation under grant no. BBW-R2122.

ample, when an application requests information from its server it only expects one answer, not one for each server to which the MUX is connected. The result of this combination of connection, multiplexing, and filtering is a solution to the joint viewing problem. An unmodified X client is displayed and interacts with several users on separate workstations simultaneously.

Unfortunately, design and implementation of a MUX is not simple [Abdel-Wahab 92a]. Several of the design problems are inherent in the concept of the MUXes, such as the floor control problem described below. Other problems, such as the late connection problem (also detailed below), are caused by details of the current X protocol. Following is a list of major MUX design issues which we have observed during our survey.

A MUX's *floor control policy* is the technique it uses to determine which user actions will be passed to the client and which will be filtered out. There are both protocol and user interface issues to be considered when designing a MUX's floor control policy. The default answer of passing all user actions generated by any user to the client is potentially confusing to the users and also to the client. If three people type simultaneously to a word processing program, the resulting text is unlikely to be useful. If two users push their mouse button simultaneously, the result may violate the X protocol. As a result of this action, two button down events without a button up event would be reported. A normal X server would not generate such a sequence of events. One approach is a lecture-style policy in which one master user is allowed to interact with the application and the other users may only watch. This is clearly limiting. A more complicated policy is a token-based model, in which the floor is passed from one user to another. The current floor holder can then interact with the application and others are excluded. This approach requires a user interface to allow users to request and pass the floor to each other.

Another design issue is the *late connection problem*, caused by adding a new server to the list of servers displaying an already executing client. In principle this should not be difficult to implement, but the current X protocol lacks support for this operation. The problem is that a client must create and then reference X resources, such as windows, on its servers in order to generate its user interface. The X protocol provides no way for the MUX to later request enough information from either the client or the server to create the resources on the new server. Without these resources, the new server cannot correctly interpret client requests.

Heterogeneous servers are also a great source of problems. A simple example of this is byte ordering. If a client is MUXed to servers on two different machine architectures then the MUX must provide byte order translation. A more complicated problem is heterogeneous displays. How should a MUX react if its servers have different bitmap depths or formats?

This paper reports the results of our evaluation of currently available MUXes. We plan to incorporate X application sharing functionality into a so-called *Joint Viewing and Tele-Operation Service* (JVTOS) [Dermler 92], [Gutekunst 93]. JVTOS is issue of work package 4.2 within RACE II project CIO (R2060) [Bauerfeld 92]. For this reason, we are concerned not only with the current functionality of the products, but also with the ease with which these products can be incorporated into a larger development project. We have acquired and tested the following MUX products: *X Terminal View (XTV)* [Abdel-Wahab 91], *SharedX (ShX)* [Altenhofen 90], *Xmux* [McFarlane 91] and *XMX*. Our evaluation consisted of first conducting a set of black box functionality tests of the various products and then considering issues such as source code availability and product architecture. Section 2 of this paper introduces the tested MUXes. Section 3 describes the test suites we used for evaluation. Section 4 discusses our more detailed review of the products. Section 5 summarizes our results and concludes this paper.

2 The Multiplexors

At their core, each of the MUXes has a *pseudo server*, a program which presents an X server interface to clients shared over the MUX. Clients connect to this application in exactly the same fashion as they connect to a normal X server, by specifying a target machine and a display number. By choosing pseudo server display numbers which do not conflict with a machine's normal server, a MUX may be run on any host computer. This pseudo server is then responsible for transmitting multiplexed protocol streams to the actual X servers among which an application is to be shared. To have an application shared, a user must simply specify the MUX as the server. For example: After starting a MUX on the computer 'myhost' with display number two, a jointly viewed xterm could be started with the following command line:

```
xterm -display myhost:2
```

Comparing the currently available MUXes is complicated by their dissimilarity. The varied target environments of the MUXes have effected their designs and have resulted in different strengths and weaknesses. Before presenting our test results we will introduce the MUXes and describe their origin and major features.

2.1 XMX

XMX was developed by John Bazik at Brown University to support presentation of computer programs to students in a class room where each student has a computer on his desk. Several simplifying assumptions were made in XMX's design. The floor control policy is lecture mode, all servers are assumed to be identical to the lecturer's server, and only black and white displays are handled. The source code of this product is provided. It is small and comprehensible. Brown University holds the source code copyright, but allows free non-profit use.

XMX is very simple to use. The XMX pseudo server is launched with a command line which specifies a list of X servers to be used as displays. The first server specified is the *master server*. Once the pseudo server is running, applications connecting to it are automatically multiplexed to all of the servers on the list. The master server is the lecturer and will always hold the floor of all MUXed clients.

2.2 Xmux

Xmux [McFarlane 91] was developed by the Australian company OTC Ltd. It has proven to be both relatively robust and full featured. Unfortunately it is a commercial product, source code is not available, and our copy was lent to us on a short-term trial basis. All of the other MUXes we have reviewed have been freeware, with source code and the right to copy and modify granted. This seriously limits Xmux's value in a research environment.

Xmux can be launched in the same manner as XMX. However the floor control policy can be either lecture or anarchy in which all users can act at once. In anarchy mode, mouse event filtering is provided to avoid protocol errors. Floor control commands can also be provided through a command socket interface to the pseudo server. We did not test this interface, all testing was done with a lecture or anarchy policy.

Like XMX, Xmux does not address the late connection problem. All servers are specified at pseudo server start-up time. Xmux is the only MUX tested which handled the shape extensions to the X protocol correctly.

2.3 X Terminal View (XTV)

XTV [Abdel-Wahab 91] was developed by researchers located in Old Dominion University and the University of North Carolina at Chapel Hill. The software is copyrighted by the universities, which grant permission to use the source code. A version is also distributed in the contrib section of the X11R5 release of the X Window System. Unfortunately all versions we have tested have proven to be unstable and crash prone. We have tested both the first version of the XTV, and the recently distributed second version (denoted as XTV/2).

XTV was developed specifically as a CSCW tool and as such contains user interface features not present in the other MUXes. These include a graphical user interface to control client launching and floor control as well as chat and sketch windows to facilitate communication. XTV/2 also provides a virtual screen feature [Lin 92].

Unlike the other MUXes tested, XTV is started as a client on each machine. The users can join an XTV session by specifying a host and a session number at XTV start-up time. A solution to the late connection problem is provided to facilitate this joining process [Chung 91]. XTV/2 also provides a graphical user interface to session selection which is based on a conferencing information service for the internet [Abdel-Wahab 92b]. However, we did not test this.

The floor control policy supported is a chaired token passing model, in which users request the token and are queued. When the floor holder chooses to release the floor, the first user in the queue receives it. The creator of the session is the chairman and may forcefully take the floor at any time. This is the only MUX we reviewed which provided a user interface to such functionality.

2.4 SharedX (ShX)

SharedX [Altenhofen 90] was developed by Michael Altenhofen at the University of Karlsruhe with support from the Digital Equipment Corporation (DEC) to be used in a tutorial/classroom environment. The original version (ShX-1) was developed as a master thesis project and is no longer under development.

We have tested this original version and two more recent versions: ShX-UCL and ShX-DFKI. Both of these versions are the result of further work by other research groups. Both groups have improved SharedX's color support. ShX-UCL comes from the University College London (UCL) and is used in the CAR multimedia conferencing project at UCL. ShX-DFKI was developed by Siemens/DFKI Saarbrücken for integration into a JVTOS demonstration prototype.

SharedX is implemented as a set of modifications to the standard X library, xlib, yielding the new library shXlib. Applications linked with shXlib, instead of the standard xlib, can then be used for joint viewing. One application built

with this library is a more typical MUX pseudo server named *shXbridge*. We have tested shXbridge. The source code to both shXlib and shXbridge is copyrighted by DEC, which allows its free use.

ShXbridge is started in the same fashion as the other pseudo servers, but only one X server is specified. When clients connect to the shXbridge they are displayed on this X server. The user may at any point connect application windows to other servers, using the program *helper* (a solution to the late connection problem is used to allow this). Floor control policy is then token based, the helper program can again be used to pass the floor or to remove the client from selected applications. The user interface to this functionality is primitive, but the documentation explains how a more sophisticated interface can be built.

3 Test Suites

We used three test suites to evaluate the MUXes: a *functionality*, a *robustness* and a *heterogeneity* test suite. This section provides an overview of the three test suites and explains the motivation behind the various tests.

3.1 Functionality Test Suite

Our first test suite was designed simply to test functionality of the MUXes in a homogeneous environment. A pair of Sun3 servers were used as display servers and MUXes were executed on a single Sun Sparc2.

The following tests were used:

- *Clients:*
  Can a list of standard X clients run? The clients used for this test were: xterm, emacs, xtetris, xeyes, xfig and FrameMaker. All can be found in the MIT X11R5 distribution, with the exception of FrameMaker. FrameMaker is a commercial word processing/desktop publishing program.
- *Window expose:*
  Are MUXed windows correctly redrawn when they generate expose events? This tests a MUX's event filtering. XMX only does redraws for the master server!
- *Shapes:*
  Are the X shape extensions supported? Xeyes run as a test. If the shape extensions are not supported, then the eyes will be placed in a rectangular window. Xeyes is part of the MIT X11R5 distribution.
- *Resources:*
  Test handling of pixmaps and other resources. Xfig run as a test. It creates and uses a larger number of server resources. Xfig can be found in the contrib section of the MIT X11R5 distribution.
- *Cut&paste:*
  Can all users cut and paste to and from the MUXed windows? If not, can the floor holder do so?
- *Child window:*
  Are dialogue box style windows handled correctly? FrameMaker produces many small windows which are displayed at fixed locations without title bars when the application is not MUXed. How are they placed when it is MUXed?
- *Floor control:*
  How well does the floor control interface work?
- *Late connection:*
  Does the MUX allow a new server to connect to an already running client?

3.2 Robustness Test Suite

The robustness suite tests exceptional cases which do not require handling heterogeneous servers.

The following tests were used:

- *Client kill:*
  Does the MUX handle a client terminating gracefully? This was tested by starting an xterm and then giving it the exit command. Correct behaviour is to simply remove the xterm from all servers.
- *Connection kill:*
  Does the MUX handle a server discarding a window gracefully? This was tested by using the twm (a standard X window manager) destroy window command. One can imagine many possible responses to this. The simplest

response is just to continue the application on the remaining servers. Crashing or freezing the application are not good responses!

3.3 Heterogeneity Test Suite

Unfortunately, due to our limited number of machines, we could not perform too many tests of heterogeneous environments. The only standard UNIX development environment we had available was on a Sun Sparc2. So all code was compiled and executed on one of our Sparcstations. However, this did not limit our choice of servers. We tested each MUX with a mix of Sparcs, Sun3s and Apple MacIIs (running MacX 1.2) as servers. This allowed us to test heterogeneous server and display mixes, but did not let us test code portability or byte order issues.

The following tests were used:

- *Server types:*
  Can the application MUX a client to a Sun3, a Sparc2 and a MacII?
- *Color:*
  Are color applications well supported? Xcolors was used along with color xfig and FrameMaker documents. Due to our limited selection of workstations this could only be tested between an 8bit-color Sparc2 and an 8bit-color MacII.
- *Color depths:*
  Can the MUX display a color application on an 8bit-color display and a monochrome display simultaneously?

3.4 Items Removed from the Test Suite Reports

The following test items were removed from the test suite reports because of uniform results or testing problems:

- *Menu location:*
  Ideally an application's pop-up menus should be drawn over window on every server, even if the window is not in the same location on every server. This was tested by bringing up xterm's and FrameMaker's menus. None of the tested MUXes did this correctly. In all cases the pop-up appeared in the same screen location on all screens, regardless of the application window's position.
- *Window resize:*
  What happens when a user resizes a multiplexed window? None of the MUXes handled this well. The solution commonly used was simply to allow the different instances of the window to become different sizes. Drawing commands are then cropped or fail to fill the full window on some server. A preferable solution would maintain a single common window size.
- *Key remap:*
  How does the MUX respond to miscellaneous server control commands such as those used to reconfigure the keyboard mappings? None of the MUXes handled this well. A common response was to report errors and apply the changes to one of the MUXes' servers.
- *Many clients:*
  Can the MUX support many clients simultaneously? Every MUX passed this test! In all cases the limit seemed to be the OS imposed limit of open connects a process may possess. The XTV floor passing interface requires the user to select the client one wishes to use from a scrolling list. This list does not scroll correctly, so that if more than six clients are created, not all can be used.
- *Many servers:*
  This robustness test involved multiplexing a client to several (five and more) servers simultaneously. All of the MUXes passed this test.

4 Test Results

This section presents our test results for each of the tested MUXes. They are shown in table 1, a dash indicating a failure, '+/-' a conditional pass, and 'X' a strong pass. The results are discussed on a MUX by MUX basis.

4.1 XMX

XMX is clearly the most limited of the MUXes tested. It was designed to be a lecture support tool, not a full function MUX. It was the only MUX tested that could not redraw windows that did not belong to the floor holder and it also lacks any support for heterogeneous servers.

One particularly strange aspect of its behaviour is that it appears to make only a single connection to each slave server, not a connection per client as the other MUXes do. This caused it to fail the connection kill test. Whenever a single window belonging to XMX is destroyed on a slave server, that server is disconnected from XMX, destroying all MUXed windows on that server. A related oddity is that all windows on the slaves appear as top-level untitled windows. This is particularly annoying when an application tries to create a dialogue box. Each slave running a twm-like window manager then freezes and waits for its user to position the dialogue box. The application's dialogues do not behave like this on the Master. The only MUX which did worse on the child window test was the UCL version of SharedX, which simply failed to display the window on all but the original server.

Despite its obvious failings XMX is an interesting reference application. At about 7.000 lines of source code, it is much smaller than any of the other MUXes. The code is simply and clearly structured, and the MUX itself is fairly stable. When we tested unsupported features XMX reported errors where other MUXes simply crashed. So in conclusion, although XMX is not suitable for direct use in the JVTOS project, it may provide a useful example or base for future MUX development work.

	XMX	Xmux	XTV	XTV/2	ShX-1	ShX-UCL	ShX-DFKI
clients	X	X	+/-	+/-	+/-	X	X
window expose	-	X	X	X	X	X	X
shapes	-	X	-	-	-	-	-
resources	-	-	-	X	+/-	X	X
cut&paste	-	X	-	+/-	+/-	+/-	+/-
child window	-	X	X	X	X	-	X
floor control	-	X	X	X	X	X	X
late connection	-	-	X	X	X	X	X
client kill	X	X	-	X	X	X	X
connection kill	-	-	X	X	+/-	+/-	+/-
server types	-	X	+/-	+/-	X	X	X
color	-	X	-	-	-	+/-	+/-
color depths	-	X	-	-	-	+/-	+/-

Table 1: Test Results

4.2 Xmux

Interesting results for Xmux included having the only working solution to cut&paste and the shape extensions. It also had the best support for color. This was the only MUX that appeared to account for the available resources of all of its target servers when making color allocation decisions. The late connection problem is not addressed. The connection kill test left the effected client frozen on all but the connection-killing server. Other clients were not effected, so this was not fatal, but this is not an optimal solution.

Xmux tested very well. It proved to be stable and robust in comparison to the other MUXes. One major lacking is its failure to address the late connection problem. However, its largest disadvantage is the fact that it is a commercial product. Greg McFarlane and OTC Ltd. have been very helpful, but we have not been able to access source code to the system and have not been lead to believe that the research community will have free access to their product. For this reason we do not recommend its use in the JVTOS project.

4.3 XTV

In its first version, XTV performed very badly during testing. It crashed very frequently. It was the only MUX to fail the client kill test. It responded by crashing. It crashed when we tried to run it on MacX. It crashed while loading the xfig and FrameMaker program and during cut&paste testing. It crashed within every heterogeneity test suite test as well. In short, it crashed a lot.

Most of these bugs are fixed in XTV/2. However, FrameMaker still causes XTV/2 to crash! Another oddity of XTV/2 is that it is not independent of the window manager. We tried to use the same X display for the chairman as well as for a second participant intending to join the session. This failed when using the window manager which comes with OpenWindows 2.0 and caused the X server (!) to crash, whereas everything went fine when using the twm window manager.

In both versions, a problem with XTV's floor control interface emerged during the many clients test. The scrolling region used to select clients for floor passing and other actions does not scroll. This prevented floor passing and other actions for all but the first six clients on the list.

On the other hand, XTV outperformed all of the other MUXes in some areas. It was the only MUX to behave well in the connection kill test. It has the most advanced floor passing system, and it provides a solution to the late connection problem.

XTV's source code proved to be a mixed bag as well. At about 25.000 or 30.000 lines respectively, it is not very large. The architecture of the system is distributed and thus similar to the proposed architecture for JVTOS' application sharing service [Gutekunst 92]. As such it offers a lot of potential leverage and time savings. Unfortunately the code itself is not well documented. It seems to be fairly well organised, but lacks the summary documentation that should help explain its user interface code and multiple processes.

Our final evaluation of XTV is very mixed. It is the only MUX tested that was designed specifically for CSCW applications such as those we intend to develop. As such it offered many interesting features, such as late joining, good floor passing control, and a general session control user interface, that were not provided by the other MUXes. On the other hand XTV proved not to be a stable enough to pass the testing suites. For the present we cannot recommend incorporating XTV into the JVTOS project, although its approach warrants study, and some of its modules may prove directly usable.

4.4 SharedX (ShX)

All three versions of SharedX tested are covered within this section.

ShX-DFKI was the best performing of the freeware MUXes in the test suites. Both ShX-UCL and ShX-DFKI have better color support than their predecessor ShX-1. Neither handled the case of multiplexing to a second server with less color map entries available than the number already in use on the original server. This may not be easily solvable in the context of the late joining problem. ShX-UCL has introduced at least one new problem: Dialogue boxes and menus created by MUXed clients appeared only on the original server's screen. ShX-1 handled dialogue boxes well (child window test). The UCL version failed this test.

All SharedX versions had some problems with cut&paste. Although the floor holder could usually successfully cut&paste we experienced one crash during testing, the display was often not correctly updated and the shXbridge often reported errors. Paste behaviour of non floor-holding users was also erratic. Xfig (resource test) reported many color related errors and all versions had pixmap related complaints. Connection kill was also erratic, but never fatally so.

The SharedX design approach is unique. The shXbridge incorporates a modified version of the complete xlib library. For this reason, the source code is much larger than for the other MUXes (about 53.000 lines). This size is slightly deceptive, because most of the files in the modified xlib have only a few lines changed from their original forms. However, xlib itself has changed with every release of the X Window System and SharedX is now two releases behind (X11R3 versus X11R5). The fact that no one has updated the shXlib system to the X11R5 leads us to believe that this update will be difficult. We believe that SharedX's dependency on the internals of the xlib implementation is a serious design weakness. All of the other MUXes reviewed are only dependent on the X protocol specification, which is well documented and stable. This dependency problem alone would be enough to cause us to reject SharedX if any of the other freeware MUXes had proven to be both stable and full featured.

4.5 Other Comparative Results

Table 2 contains a mix of non-test data that influenced our final recommendation:

- *MUX freeware:*
  Is the MUX a freeware product, i.e. is the source code available, and can it be legally copied, modified, and redistributed for non-profit uses?
- *Source lines:*
  Number of lines of MUX source code. This is a somewhat useful measure of how difficult the source code will be to understand, modify and maintain.

- *Least crashes:*
  The number of times this MUXes crashed during its best run through the test suites.
- *Current product:*
  Is this MUX currently under development? Can the product's developer be contacted? Note that with the exception of Xmux, these products are all freeware. The developers have made no commitment to product support, they have simply made it available to the research community.
- *New version soon:*
  Have the developers of this product promised a new release of the MUX in the near future?

	XMX	Xmux	XTV	XTV/2	ShX-1	ShX-UCL	ShX-DFKI
MUX freeware	yes	no	yes	yes	yes	yes	yes
source lines	7.107	?	25.798	30.465	52.811	53.168	53.189
least crashes	0	0	6	4	2	1	1
current product	yes	yes	no	yes	no	yes	yes
update expected	yes	?	no	?	no	?	?

Table 2: Other Comparative Results

5 Summary and Conclusions

We find *XMX* to be an interesting teaching example. Its code is small and simple. However it is not nearly full featured enough to consider use as is. Its most serious lacking is that the servers supported are not equal. One server is the master and the others are simply slaves, displaying the actions of the master. A new version with a much larger feature set entered alpha test in January 1993. We plan to review it as soon as it has become stable.

Xmux seems to be a well designed and implemented product. It is stable and full featured, although it does not provide a solution to the late connection problem. However, unlike all of the other X wedges reviewed, it is a commercial product. Hence, no source code is available. For this reason we are not considering using it for JVTOS.

XTV has the most promising architecture and feature set of all of the MUXes reviewed. It provides solutions to the late connection problem and full floor control features. Its first version has proven to be very unstable. XTV/2 crashed less but is not stable enough yet. We do not believe that the code can be used as is for JVTOS. However some modules may prove directly usable.

We reviewed three versions of *SharedX*: ShX-1, ShX-UCL and ShX-DFKI. Both ShX-DFKI and ShX-UCL improved color support, but ShX-UCL has introduced a serious child window handling bug. For these reasons ShX-DFKI appears to be the best choice of the three. SharedX is full featured and well documented. Two related flaws are its large source code size and its grounding in the outdated release 3 of the X Window System. The fact that it has not been updated from this release in over two years of development cause us to believe that such a renovation has proven difficult.

Our survey has not yielded an obvious best choice MUX for inclusion in the JVTOS project. By process of elimination ShX-DFKI appears to be the best choice for as is inclusion into a development project. The factors influencing this recommendation are: First, ShX-DFKI is freeware. Secondly it is full featured, providing support for heterogeneous servers, color, the late connection problem, and has a token-based floor control policy. And finally, it has proven to be fairly reliable and crash resistant. However, ShX-DFKI has enough detrimental features that we shall be forced to consider either waiting for improvements in the existing MUXes or building a new MUX, perhaps incorporating technology from XMX and/or XTV.

The process of elimination that yields ShX-DFKI results from the following ranked requirements:

(1) freeware; (2) full featured; (3) reliable; (4) simple, clean and maintainable code.

Requirement one is based on the fact that we anticipate needing to modify any system we use over time. This requires source code. We also expect to be able to freely distribute the result of our work. Hence our insistence on freeware. This eliminates Xmux from consideration.

Requirement two eliminates XMX and ShX-1. ShX-DFKI is a super-set of ShX-1's functionality. It also is currently under development, whereas ShX-1 is a finished thesis project.

This leaves only XTV and ShX-DFKI under consideration. The test suite has convincingly eliminated XTV. It has proven very unreliable. XTV crashed more often than all of the other MUXes combined.

So we are left with ShX-DFKI. Unfortunately the fourth requirement also brings ShX-DFKI under question. It is based on an outdated version of X and includes a very large amount of modified X11R3 library code. This is not very clean or reliable. Despite this, we see ShX-DFKI as the best as is MUX option.

Instead of a strong recommendation, we are forced to present two options for consideration. If it is decided that JVTOS development must not focus on MUX development, then, in our opinion, ShX-DFKI currently presents the best base. The second option is to develop a new MUX. Given that the reviewed MUXes provide plenty of source code examples and that code from them can be borrowed under the normal freeware conditions (i.e. credit the original authors), this option should also be considered.

6 Acknowledgements

The authors would like to thank the partners of CIO work package 4.2 for their discussions, advice and comments. Jürg Ehrbar, Michael Frass, Thomas Schmidt, Günter Schulze, and Wladimir Minenko have been of particular assistance.

7 References

[Abdel-Wahab 91] Hussein M. Abdel-Wahab, Mark A. Feit: "XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration". *Proceedings, IEEE Tricomm '91: Communications for Distributed Applications & Systems.* Chapel Hill, 1991.

[Abdel-Wahab 92a] Hussein Abdel-Wahab, Kevin Jeffay: "Issues, Problems and Solutions in Sharing X Clients on Multiple Displays". *Technical Report TR92-042, University of North Carolina.* Chapel Hill, 1991.

[Abdel-Wahab 92b] Hussein Abdel-Wahab: "Reliable Information Service for Internet Computer Conferencing". *Technical Report TR92-043, University of North Carolina.* Chapel Hill, 1991.

[Altenhofen 90] Michael Altenhofen: "Erweiterung eines Fenstersystems für Tutoring-Funktionen". *Diploma Thesis at Universität Karlsruhe.* Karlsruhe, 1990.

[Bauerfeld 92] Wulfdieter Bauerfeld: "RACE-Project CIO (R2060): Coordination, Implementation and Operation of Multimedia Tele-Services on Top of a Common Communication Platform". *IWACA '92 - International Workshop on Advanced Communications and Applications for High Speed Networks.* München, 1992.

[Chung 91] Goopeel Chung: "Accommodating Latecomers in a System for Synchronous Collaboration". *Master Thesis at University of North Carolina.* Chapel Hill, 1991.

[Dermler 92] Gabriel Dermler, Konrad Froitzheim: "JVTOS - A Reference Model for a New Multimedia Service". *4th IFIP Conference on High Performance Networking (hpn '92).* Liège, 1992.

[Gutekunst 92] Thomas Gutekunst: "Proposed Implementation Architecture for the Application Sharing Service". *Internal Report of RACE/CIO WP 4.2.* Zürich, 1992.

[Gutekunst 93] Thomas Gutekunst, Thomas Schmidt, Günter Schulze, Jean Schweitzer, Michael Weber: "A Distributed Multimedia Joint Viewing and Tele-Operation Service for Heterogeneous Workstation Environments". *GI/ITG Workshop on Distributed Multimedia Systems.* Stuttgart, 1993.

[McFarlane 91] Greg McFarlane: "Xmux - A system for computer supported collaborative work". *Proceedings, 1st Australian Multi-Media Communications, Applications & Technology Workshop.* Sydney, 1991.

[Lin 92] Jin-Kun Lin: "Virtual Screen: A Framework for Task Management". *The X Resource 1, Winter 1992, pp. 191 - 198.* 1992.