A Graphical User Interface Toolkit Approach to Thin-Client Computing

Simon Lok, Steven K. Feiner, William M. Chiong and Yoav J. Hirsch
Dept. of Computer Science, Columbia University
1214 Amsterdam Ave.
New York, NY 10027
{lok,feiner}@cs.columbia.edu, {wmc14,yjh9}@columbia.edu

ABSTRACT

Network and server-centric computing paradigms are quickly returning to being the dominant methods by which we use computers. Web applications are so prevalent that the role of a PC today has been largely reduced to a terminal for running a client or viewer such as a Web browser. Implementers of network-centric applications typically rely on the limited capabilities of HTML, employing proprietary "plug ins" or transmitting the binary image of an entire application that will be executed on the client. Alternatively, implementers can develop without regard for remote use, requiring users who wish to run such applications on a remote server to rely on a system that creates a virtual frame buffer on the server, and transmits a copy of its raster image to the local client.

We review some of the problems that these current approaches pose, and show how they can be solved by developing a distributed user interface toolkit. A distributed user interface toolkit applies techniques to the high level components of a toolkit that are similar to those used at a low level in the X Window System. As an example of this approach, we present RemoteJFC, a working distributed user interface toolkit that makes it possible to develop thin-client applications using a distributed version of the Java Foundation Classes.

Categories and Subject Descriptors

C.2.4 [**Networks**]: Distributed Systems; D.3.3 [**Programming Languages**]: Language Constructs and Features; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

Keywords

User interface toolkit, remote method invocation, client-server systems, network computing

1. INTRODUCTION

Few would argue that the explosive growth in the computer industry is not closely tied to the rise in popularity of networks and, in particular, the Internet. In 1990, an average person purchased a computer for running application software such as word processing, spreadsheets, and perhaps a drawing program. Today, that same person would undoubtedly purchase a computer to access the Internet. However, while computing is heading in radically new directions, the techniques used to present graphical user interfaces have remained the same.

The vast majority of user interfaces are written using visual components (often called *widgets* or *controls*) that are gathered together in libraries that are usually referred to as *user interface toolkits* [6]. The most popular toolkit is MFC (Microsoft Foundation Classes) [33], which, as its name implies, is used to construct user interfaces for the various flavors of Microsoft's Windows operating system. A more recent toolkit that is gaining popularity because of its ability to create cross-platform compatible graphical user interfaces is JFC (Java Foundation Classes) [23]. Complementing these are the more mature user interface toolkits built for the X11 Window System [43], such as Athena [32], Motif [35], and Tk [37].

A user interface toolkit provides an abstraction layer for the low-level drawing and interaction routines made available to programmers by the graphics subsystem that is usually bundled with the operating system. This abstraction allows programmers to quickly create commonly used visual components, such as buttons, scrollbars, menus, and text fields. End users also benefit, since most of the applications they run on a particular operating system will have roughly the same "look and feel" because the applications are all built out of components from the same user interface toolkit. However, despite these advantages, the tight binding of the user interface toolkit to the underlying graphics subsystem presents significant challenges when creating distributed applications in which the application logic execution and user interface presentation occur on different computers.

Many approaches have been researched academically and deployed commercially to support a distributed computing paradigm in which the network separates the presentation of the user interface from the application logic. Phrases such as "server-centric computing," [28] "network computing" [3, 14], "thin clients" [50, 44, 14], "distributed presentation" [14], and "remote presentation" [41] are prevalent in the literature. In the remainder of this paper, we first discuss in Section 2, two approaches to thin-client computing that are commonly used in both industry and academia: Web-based applications and graphics pipeline interception. We then describe an alternative, distributed user interface toolkits, in Section 3, and introduce in Section 4 our distributed user interface toolkit, RemoteJFC, describing how it addresses many of the issues that arise from using existing systems. Next, in Section 5, we present a performance comparison, and, in Section 6, conclude with some remarks about the directions in which our research is heading.

2. THIN-CLIENT COMPUTING

2.1 Web-Based Approaches

One of the most widely deployed approaches to thin-client computing uses HyperText Transfer Protocol (HTTP) [5] and Hyper-

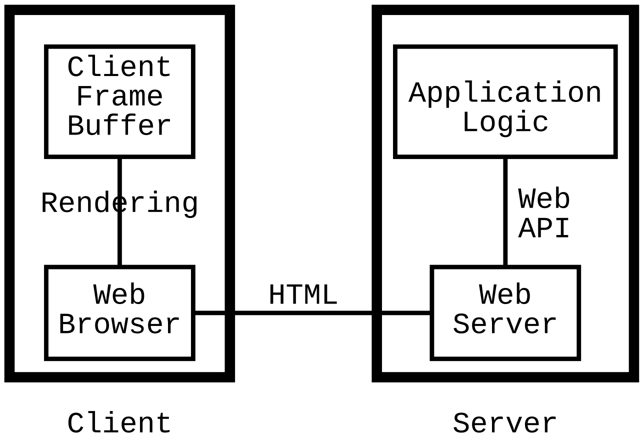


Figure 1: The architecture of a Web-based application. The application must be written with a special Web API (e.g., CGI [8], ISAPI [20], NSAPI [36], ASP [1], PHP [38], or JSP [24]) to communicate with a Web server. HTTP is used to negotiate the transfer of HTML data between the client Web browser and the Web server. The Web browser then renders the HTML onto the client frame buffer.

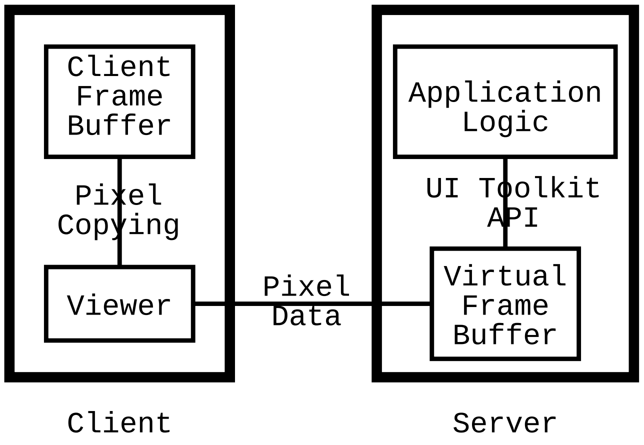


Figure 2: The architecture of a remote frame-buffer–based application. The application is typically written using a standard UI toolkit API (e.g., JFC [23], MFC [33], Tk [37], or Motif [35]) and renders onto a remote virtual frame buffer. The resulting pixel data is transported across the network, using a proprietary protocol (e.g., ICA [9], RFB [34], or RDP [40]), to the client viewer, where the image is then reconstructed and copied into the client frame buffer.

Text Markup Language (HTML) [4, 10] to interact with the server and display document database, commonly known as the World Wide Web. The popularity and wide availability of browsers that provide the front-end to view HTTP/HTML content has brought about the rapid development and deployment of applications that are accessible only through this format. The architecture of an application developed using a Web-based methodology is depicted in Figure 1.

One severe limitation of a Web-based application that relies solely on HTTP/HTML is the "pull-only" data transfer methodology. Such applications are prevented from generating events and thus cannot provide a rich user experience. For example, when a user executes a search on a Web search engine, the engine must ideally complete the search in its entirety within a few seconds of when the request was made because the user is expecting an immediate response and the engine cannot notify the user that better results have been found after the initial page has been displayed. A second problem is that HTTP is stateless, which makes it difficult for programmers to create even a simplistic notion of persistence between page accesses. In addition, the user interface toolkit (HTML forms [4]) is also extremely rudimentary, providing only a handful of the most commonly used components.

Many attempts have been made to address these problems, including sending entire applications over HTTP (e.g., Java applets [22]), designing browser "plug-ins" that interpret their own language to provide a richer user experience (e.g., Macromedia Flash and Shockwave [31]), creating a 3D world in which the user can navigate (e.g., VRML [49]), and providing an application programmer interface (API) for storing persistent session identification data (e.g., cookies [27]). All these attempts to address the problems with HTTP/HTML give rise to a host of new problems.

Java applets raise numerous security concerns because HTTP is used to transport executable code to the client. Although the byte codes transmitted across the network are in compiled form, Java decompilers are readily available that will allow any user to have access to the source code of the application. In addition, the use of Java applets typically violates the thin-client principle of not running any application logic on the client. Flash and VRML define richer languages that have been built with user interactivity in mind, but suffer from the problem that mature browsers for anything other than the Microsoft Windows desktop operating systems are generally not available. HTTP cookies raise numerous security concerns because they permit the server program to write data to the permanent storage device on the client. In addition, HTTP cookies have been the target of severe criticism due to a recent surge in public awareness regarding privacy concerns when using the Internet. These issues make HTTP cookies an unattractive method for programmers to add server-side state to the HTTP protocol.

2.2 Graphics Pipeline Interception

A second approach to delivering applications in a thin-client environment that has been gaining popularity involves intercepting rendering commands sent to the graphics pipeline. One simple way to implement this is to create a virtual frame buffer in the RAM of the server on which the application can render it's GUI and then transporting the resulting raster image to the client. A more complicated implementation would try to send higher level commands (e.g. draw a line from x_i, y_i to x_j, y_j) whenever possible in order to reduce network traffice.

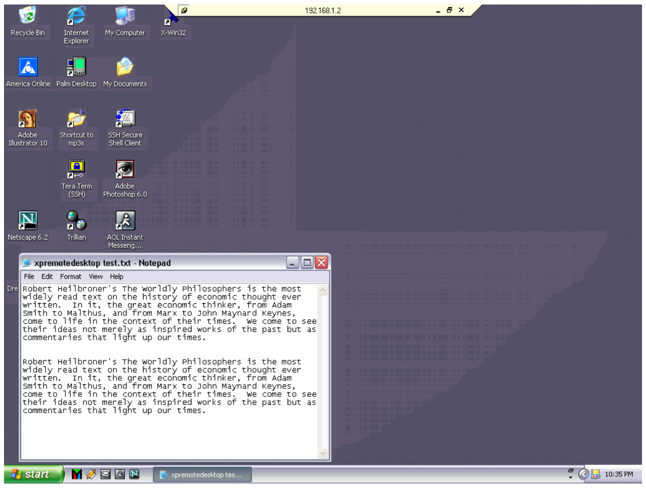
In essence, this approach attempts to bring the server's desktop to the user and thereby permits a full range of user interactivity. Products such as Citrix MetaFrame [9], Insignia NTrigue [19], SCO Tarantella [46], Graphon RapidX [16] and Symantec PC-Anywhere [47] are among those that have been providing this type of functionality for many years as an extension to the underlying operating system. A recent explosion in the popularity of this approach occurred when AT&T released their cross-platform VNC [29] system to the public free of charge. Microsoft has now made this capability a standard part of their Windows 2000 and XP operating systems [34]. The architecture of an application that employs the remote frame buffer approach for presentation on a thin-client is displayed in Figure 2.

Although the approach of intercepting the graphics pipeline addresses many of the problems with a Web-based approach that uses HTTP/HTML, it also introduces a number of other problems. While the Web-based approach using HTTP/HTML is capable of operating reasonably well over relatively low-speed modem network links, the graphics pipeline interception approach demands high-bandwidth connections. This is because transporting the virtual frame buffer from the server to the client is essentially sending a video stream of computer-generated graphics. Although the use of advanced lossy video compression algorithms (e.g. MPEG [21]) has been proposed [40], none of the existing systems employ such techniques. This is because real-time encoding of MPEG streams usually requires special hardware that can only handle one or two streams at a time, thereby eliminating the possibility of using the remote frame buffer approach on a current shared server. In addition, the use of lossy compression techniques would introduce unwanted compression artifacts into the display, reducing the system's usability, particularly when working with text and detailed graphics.

Some systems (e.g. RDP-based Citrix Metaframe and Microsoft Windows Terminal Services) attempt to reduce bandwidth consumption by trying to intercept high level drawing commands. However, this doesn't necessarily result into better performance. For example, if the GUI employs many image labels for buttons, the bandwidth consumed by transferring the images every time the display needs to be redrawn dwarfs everything else. In addition, these systems often employ other optimizations such as not transferring display updates that are thought to be "unimportant" (e.g. animated cursors). Although this technique appears to reduce the the bandwidth consumed independent of all possible factors, it almost always causes the display on the client to not be updated properly, ultimately resulting in user frustration. Users that experience this will often start moving the mouse or windows around in order to force the system to update the display. This can actually result in more bandwidth being consumed than if the system had simply sent the updates using the more naive approach. Finally, these systems suffer from the overhead of bringing the entire user's desktop from the server to the client rather than just the application.

The existence of server-side state and asynchronous event generation by the server permits the graphics pipeline interception approach to provide a rich level of user interactivity that a Web-based approach using HTTP/HTML cannot. However, there is a practical limitation caused by network latency. Figure 4 shows a typical client "viewer" (the graphics pipeline interception analogue of the Web browser) that displays two mouse pointers. One mouse pointer represents where the cursor should be pointing, and is tied to the local mouse. A second mouse pointer, which typically lags behind the first, displays where the mouse position is on the server. When a simple remote frame buffer system is run on anything other than a high-speed LAN connection, there is always a noticeable difference in position between the client (virtual) and server (real) mouse positions. More advanced graphics pipeline interception implementations (e.g. RDP systems) generally do not have the same mouse pointer lag issue, but still stuffer from a similar problem when a window is dragged. On a slow modem link, this makes highly interactive user interfaces difficult to control, and, in extreme cases, may even make the system unusable.

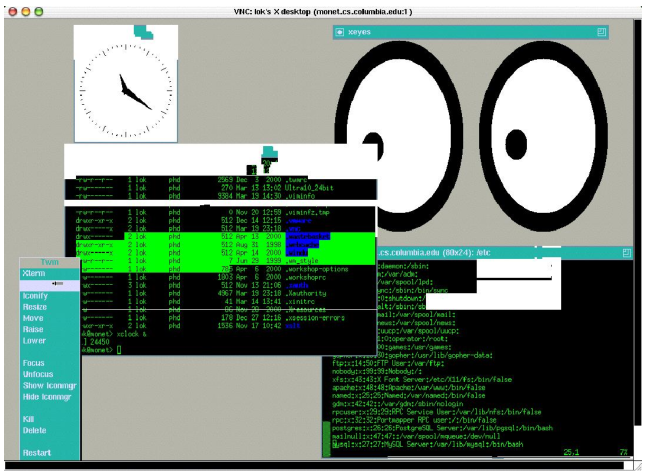
One important advantage of graphics pipeline interception systems are that they tend to be binary-compatible with a large set of existing software packages intended for use with desktop computers. Many industrial and academic institutions employ graphics pipeline interception systems in production environments to provide users with thin-client access to some subset of the enterprise or campus computing infrastructure. By intercepting graphics routine at the operating system level, little or no programming effort is involved in deploying the system.



```
Robert Heilbroner's The Worldly Philosophers is the most
widely read text on the history of economic thought ever
written.  In it, the great economic thinker, from Adam
Smith to Malthus, and from Marx to John Maynard Keynes,
come to life in the context of their times.  We come to see
their ideas not merely as inspired works of the past but as
commentaries that light up our times.


Robert Heilbroner's The Worldly Philosophers is the most
widely read text on the history of economic thought ever
written.  In it, the great economic thinker, from Adam
Smith to Malthus, and from Marx to John Maynard Keynes,
come to life in the context of their times.  We come to see
their ideas not merely as inspired works of the past but as
commentaries that light up our times.
```

(a)



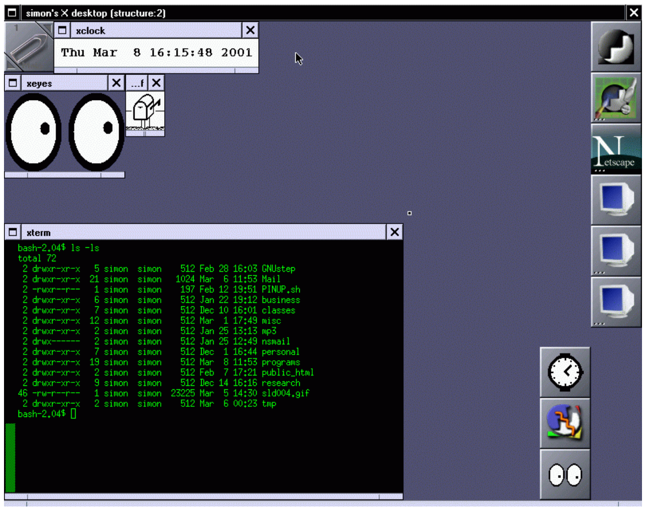
```
-rw-r--r--    1 lok      phd          2569 Dec  3  2000 .twmrc
-rw-------    1 lok      phd           270 Mar 13 13:02 Ultra10_24bit
-rw-------    1 lok      phd          9384 Mar 19 14:30 .viminfo

-rw-r--r--    1 lok      phd             0 Nov 20 12:59 .viminfz.tmp
drwxr-xr-x    2 lok      phd           512 Dec 14 12:15 .vmware
drwx-----x    2 lok      phd           512 Mar 19 23:18 .vnc
drwx------    2 lok      phd           512 Apr 13  2000 .wastebasket
drwxr-xr-x    2 lok      phd           512 Aug 31  1998 .webcache
drwx-----x    2 lok      phd           512 Apr 14  2000 [illegible]
rw-r--r--     1 lok      phd             7 Jun 29  1999 .wm_style
w-------      1 lok      phd           785 Apr  6  2000 .workshop-options
w-------      1 lok      phd          1803 Apr  6  2000 .workshoprc
wx------      3 lok      phd           512 Nov 13 21:06 .xauth
w-------      1 lok      phd          4967 Mar 19 23:18 .Xauthority
w-------      1 lok      phd            41 Mar 14 13:41 .xinitrc
w             1 lok      phd            06 Nov 20  2000 .Xresources
w-------      1 lok      phd           178 Dec 27 12:16 .xsession-errors
wxr-xr-x      2 lok      phd          1536 Nov 17 10:42 xslt
k@monet> xclock &
] 24450
k@monet> 
```

```
:daemon:/sbin:
m:/var/adm:
/var/spool/lpd:
ync:/sbin:/bin/sync
:0:shutdown:/
alt:/sbin:/sb
mail:/var/spool/mail:
news:/var/spool/news:
:uucp:/var/spool/uucp:
1:0:operator:/root:
00:games:/usr/games:
30:gopher:/usr/lib/gopher-data:
ftp:x:14:50:FTP User:/var/ftp:
nobody:x:99:99:Nobody:/:
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false
apache:x:48:48:Apache:/var/www:/bin/false
named:x:25:25:Named:/var/named:/bin/false
gdm:x:42:42::/var/gdm:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/bin/false
rpc:x:32:32:Portmapper RPC user:/:/bin/false
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
mailnull:x:47:47::/var/spool/mqueue:/dev/null
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
                                                  25,1          7%
```

(b)

Figure 3: A screen shot that shows (a) the Windows XP RDP implementation and (b) the AT&T reference VNC implementation not refreshing the screen properly. This can sometimes prevent the user from actually being able to do accomplish the task that they set out to do. Users will often try to randomly move their mouse or windows around in an attempt to force the screen to update, resulting in a large amount of bandwidth being consumed.



```
bash-2.04$ ls -ls
total 72
 2 drwxr-xr-x   5 simon  simon    512 Feb 28 16:03 GNUstep
 2 drwxr-xr-x  21 simon  simon   1024 Mar  6 11:53 Mail
 2 -rwxr--r--   1 simon  simon    197 Feb 12 19:51 PINUP.sh
 2 drwxr-xr-x   6 simon  simon    512 Jan 22 19:12 business
 2 drwxr-xr-x   7 simon  simon    512 Dec 10 16:01 classes
 2 drwxr-xr-x  12 simon  simon    512 Mar  1 17:49 misc
 2 drwxr-xr-x   2 simon  simon    512 Jan 25 13:13 mp3
 2 drwx------   2 simon  simon    512 Jan 25 12:49 nsmail
 2 drwxr-xr-x   7 simon  simon    512 Dec  1 16:44 personal
 2 drwxr-xr-x  19 simon  simon    512 Mar  8 11:53 programs
 2 drwxr-xr-x   2 simon  simon    512 Feb  7 17:21 public_html
 2 drwxr-xr-x   9 simon  simon    512 Dec 14 16:16 research
46 -rw-r--r--   1 simon  simon  23225 Mar  5 14:30 sld004.gif
 2 drwxr-xr-x   2 simon  simon    512 Mar  6 00:23 tmp
bash-2.04$ []
```

Figure 4: A screen shot depicting the latency between the actual position of the local mouse pointer (the small black square with a white outline near the center of the image, just above and to the right of the "xterm" window) and the position of the virtual mouse on the server (the arrow icon to the right of the "xclock" window) that is typically experienced in a remote frame buffer thin-client system. This screen shot was taken using the AT&T VNC [40] system with a 56K modem connection between the client and server.

3. DISTRIBUTED UI TOOLKITS

Distributed user interface toolkits address the issues that arise when employing Web-based HTTP/HTML and remote frame buffer approaches by allowing a server to manipulate user interface toolkit components directly on the client. The server can create, modify, and delete any of the components available in the distributed toolkit as if it were working with a local application. One might think of this as an implementation of a remote frame buffer with an extremely efficient, lossless compression algorithm. Instead of sending pixel data rendered on the server across the network, the distributed user interface toolkit sends the semantics necessary to render that pixel data on the client. In addition, since the mouse is handled locally on the client, there is no additional perceived latency beyond that caused by the processing that is necessary to service users requests when the application is running locally.

The concept of creating architectures and toolkits that support the development of distributed applications is not new. For example, the X Window System [43] and the Network extensible Window System (NeWS) [15] were built with the network in mind, although they transport low-level drawing commands. If a high-level user interface toolkit is used with X or NeWS, it appears as if the high-level commands are being transported across the network, although this is not what is happening. Under X, the high-level user interface toolkit commands (e.g., draw button) are actually translated into low-level commands (e.g., lines and rectangles) before being transmitted across the network. In addition, the X Window System stores state on the computer that is presenting the output (unfortunately called the server). Consequently, it is very difficult to "share" X Window System sessions between multiple users, and if the X Window System server (running on the client computer) fails, the user session is lost. It is for these reasons that a remote virtual frame buffer system, such as VNC, is often employed to transport an X Window System desktop from a UNIX server to an

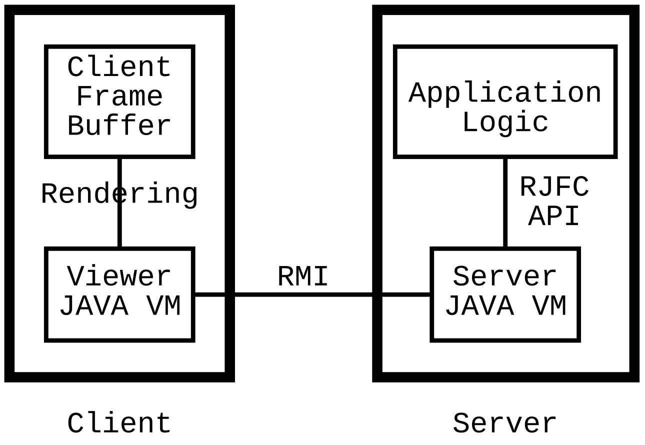


Figure 5: The architecture of an application written using the RemoteJFC distributed user interface toolkit. The RemoteJFC (RJFC) API closely parallels the standard JFC user interface toolkit API. RJFC commands are interpreted on the server Java VM and transported across the network using RMI. The client Java VM then translates the RJFC RMI command sequences into standard JFC calls for rendering on the client.

X Window System viewer running on a UNIX workstation, rather than relying on the built-in networking facilities of X.

Many research efforts have attempted to make systems more network aware. For example, the mobile computing community has looked into modifying the operating system to support network-aware applications [25, 26]. The collaborative computing community has also contributed numerous architectures and toolkits [12, 17, 39, 45]. In addition, the user interface community has been exploring how the network can enhance the functionality of user interface toolkits [18, 2], while the virtual reality research community has developed distributed user interface toolkits for 3D graphics [42, 30]. These architectures and toolkits are all designed with the premise that most, if not all, of the application logic executes on the client.

4. THE REMOTEJFC TOOLKIT

To determine how a distributed user interface could be constructed, and to explore the advantages that it could make possible, we have developed RemoteJFC (RJFC). Our primary goal for RJFC was to create an API that tracks the design pattern and functionality of the standard JFC API closely as possible, with the exception that the presentation displays on a remote client, rather than the local frame buffer. To accomplish this, we needed to create a well-defined API and corresponding software development kit (SDK), establish a protocol for client-server communication, and develop a Java-based viewer that provides a graphical context on the client. Figure 5 shows the architecture of a RJFC application.

4.1 RJFC API

When building the RJFC system, we wanted to make sure that the system would be appealing to users. To accomplish that goal, we concluded that we would need to provide an API that was both familiar and rich in functionality. In other words, we need to emulate as much as possible of the JFC user interface toolkit provided by Sun. Like many contemporary user interface toolkits, the JFC API is extremely complex, with over 600 individual source files, each providing between 10 to 100 methods for the programmer to

```
public void registerDisplay(RJFrame d,
       RJFCFactory f) throws RemoteException {

  RJTextArea TheArea = f.getRJTextArea(20,20);
  TheArea.addKeyListener(new
                  TextAreaKeyListener());

  RJScrollPane Pane = f.getRJScrollPane();
  Pane.setViewportView(textArea);

  RJTextField StatusBar = f.getRJTextField();
  StatusBar.setEditable(false);

  RContainer rc = d.getContentPane();
  rc.setLayout(new BorderLayout());
  rc.add(StatusBar, BorderLayout.SOUTH);
  rc.add(CreateMenu(), BorderLayout.NORTH);
  rc.add(Pane, BorderLayout.CENTER);

}
```

(a)

```
public MyJFrame() extends JFrame {

  JTextArea TheArea = new JTextArea(20,20);
  TheArea.addKeyListener(new
                  TextAreaKeyListener());

  JScrollPane Pane = new JScrollPane();
  Pane.setViewportView(textArea);

  JTextField StatusBar = new TextField();
  StatusBar.setEditable(false);

  Container c = this.getContentPane();
  c.setLayout(new BorderLayout());
  c.add(StatusBar, BorderLayout.SOUTH);
  c.add(CreateMenu(), BorderLayout.NORTH);
  c.add(Pane, BorderLayout.CENTER);

}
```

(b)

Figure 6: A comparison of (a) the RemoteJFC API with (b) the Sun JFC API. The `registerDisplay` method executes on the server and creates a simple "notepad" application that is displayed on the client using the RemoteJFC API. The `MyJFrame` constructor creates the same notepad application in a desktop (non-network-aware) environment using the JFC API. Note the one-to-one substitution of RJFC components for JFC components. In addition, where the JFC code calls `new` to instantiate a component, the RJFC code makes a remote method invocation to an `RJFCFactory` object that resides in the viewer's memory space. The RemoteJFC API could have been designed with calls to `new`, but that would result in a performance degradation as components would need to be serialized and transported across the network. The syntax of RJFC and JFC is sufficiently similar that we believe a code generator could be written to convert any existing desktop JFC application to a RJFC thin-client implementation.

use. It would be a daunting task for a small research team to write wrappers by hand for all of this code. Since the source code to JFC is readily available, we created a code generator, using the Java Doclet API [11], that reads in the JFC source and produces RJFC for each JFC element. This approach allows us to generate different versions of our RJFC system for various implementations and releases of the Java SDK, making it possible to handle a broad range of supported JVMs.

An example of code written using the RJFC API is shown in Figure 6. We tailored our code generator so that the resulting RJFC API is sufficiently similar to JFC that a programmer fluent in creating applications with JFC need only know that a capital "R" must be prepended to the name of the toolkit component being referenced. Although we could have defined component creation using the "new" keyword, we use a RJFCFactory object for performance reasons (discussed in Section 4.2).

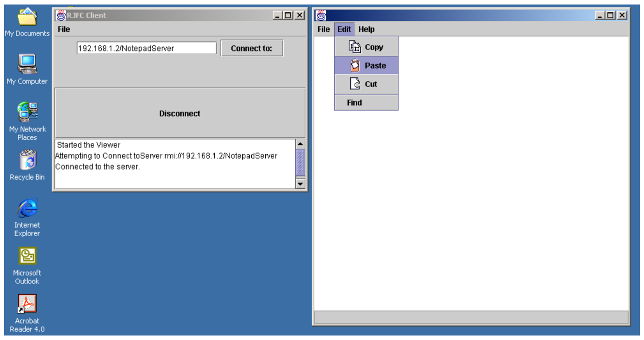
Manipulation of the RJFC components (e.g., changing the text of a label) and association of event handlers are syntactically identical to the JFC API. Although each RJFC component has an actual associated JFC component that lives in the viewer's memory space, the programmer interacts with the display solely by making calls on the RJFC components. The actual JFC components that are used to create the display on the viewer are hidden from the programmer. Since RJFC components track the JFC API and follow the Sun Java Beans standard, they may also be easily used in graphical user interface builders such as Sun Forte for Java [13], Borland JBuilder [7], and WebGain VisualCafe [48].

4.2 RJFC Protocol

When a RJFC component is instantiated, modified, or deleted on the server, the RJFC toolkit transparently informs the attached RJFC viewer of the event that has occurred using remote method invocation (RMI). The RJFC viewer then reacts to this by performing the exact same action on the viewer that would have occurred on the server if the JFC API were used. For example, if the server requests that a new RJButton be created, the RJFC toolkit would transmit that command to the viewer Java VM. The viewer Java VM then creates a JButton using the standard JFC API, thus causing the actual button to be rendered on the client. Similarly, when the RJFC server installs an event handler into a RJFC component, the server uses RMI to tell the viewer to install a proxy JFC event handler into the associated JFC component that is actually being displayed.

One key performance optimization in the RJFC Protocol is the use of a RJFCFactory to create JFC components in the viewer's memory space while returning a RJFC reference to the server. The RJFCFactory is a remotely accessible object (it extends UnicastRemoteObject and implements an interface that extends Remote) that lives in the viewer's memory space. When a viewer connects to a RJFC server, the viewer passes a reference to the RJFCFactory into the display registration method implemented on the RJFC server. Once the RJFC server has a reference to the RJFCFactory, the server can create JFC components that live in the viewer's memory space and receive a remote reference to the associated RJFC wrapper object rather than creating an object in the server's memory space, sending the serialized object to the client and then sending a remote reference to the wrapper object back to the server. Our measurements show that a RMI call consumes five Ethernet packets, whereas sending a serialized JButton consumes more than ten times that number.

The RJFC protocol uses a similar approach to accomplish event handling. When an event handler is installed into a RJFC component on the server, RJFC uses RMI to send a simple message that



192.168.1.2/NotepadServer

```
Started the Viewer
Attempting to Connect toServer rmi://192.168.1.2/NotepadServer
Connected to the server.
```

Figure 7: A Microsoft Windows desktop with the RJFC viewer accessing a remote notepad application. The window on the left allows the user to enter a RMI URL to select a server and application to display in the viewer window on the right.

tells the viewer to install a proxy event handler in the associated JFC object. The proxy event handler makes a RMI call to the server whenever a new event is generated on the client side. The actual semantics of the event handler as defined by the application logic is executed on the server when the server receives the RMI call from the client. Server-generated events are supported by simply having the RJFC server retain the reference to the RJFC component returned by the RJFCFactory after the display initialization completed. With a remote reference to the RJFC component, the server is free to asynchronously generate events at will.

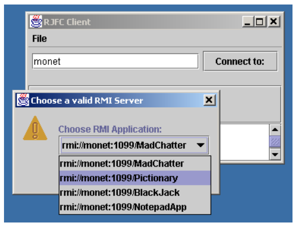
4.3 RJFC Viewer

The RJFC viewer, shown in Figure 7, provides a context in which the RJFC server application can manipulate the client frame buffer. The viewer is a hand-coded application that uses JFC and emulates the functionality found in a typical thin-client system. The user of the system invokes the viewer, at which point a JFrame window is created with a form that allows the user to connect to a server. Once a connection is established, another JFrame window is created for the server to manipulate remotely. The server may also request that additional windows be created by asking for dialogue boxes using the RJFC API. Figure 8 shows screen shots of several small applications being run in the RJFC viewer.

5. PERFORMANCE COMPARISON

The Web-based thin-client approach using HTTP/HTML consumes very little bandwidth because HTML represents a presentation's semantics at an extremely high level. While a relatively small amount of information is transported, this approach suffers from the problem that HTTP was not designed for implementing remote applications, but rather for sharing static data. In contrast, the remote frame buffer approach operates on the premise that compatibility with existing applications is paramount at the expense of network bandwidth. This is because many of the remote frame buffer implementations were designed for corporate or lab network environments whose administrators are trying to move users away from desktop computers to a thin-client subsystem with a lower total cost of ownership.

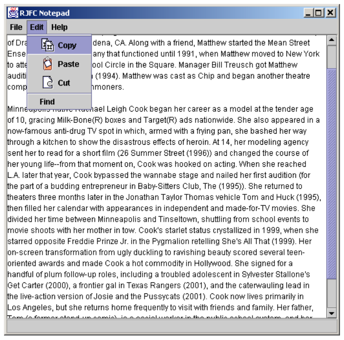
The RemoteJFC distributed user interface toolkit attempts to combine the benefits of both approaches without their performance and usability issues by transmitting the high-level semantics of a display using a standard toolkit API. Intuitively, one would expect the network bandwidth consumed by RJFC to be closer to that of the Web-based approach using HTTP/HTML than that of the remote



monet

rmi://monet:1099/MadChatter
rmi://monet:1099/MadChatter
rmi://monet:1099/Pictionary
rmi://monet:1099/BlackJack
rmi://monet:1099/NotepadApp

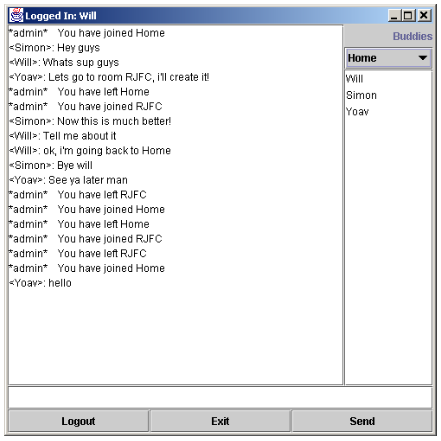
(a)



of Dra dena, CA. Along with a friend, Matthew started the Mean Street Ense any that functioned until 1991, when Matthew moved to New York to atte ool Circle in the Square. Manager Bill Treusch got Matthew auditi (1994). Matthew was cast as Chip and began another theatre comp nmoners.

Minne achael Leigh Cook began her career as a model at the tender age of 10, gracing Milk-Bone(R) boxes and Target(R) ads nationwide. She also appeared in a now-famous anti-drug TV spot in which, armed with a frying pan, she bashed her way through a kitchen to show the disastrous effects of heroin. At 14, her modeling agency sent her to read for a short film (26 Summer Street (1996)) and changed the course of her young life--from that moment on, Cook was hooked on acting. When she reached L.A. later that year, Cook bypassed the wannabe stage and nailed her first audition (for the part of a budding entrepreneur in Baby-Sitters Club, The (1995)). She returned to theaters three months later in the Jonathan Taylor Thomas vehicle Tom and Huck (1995), then filled her calendar with appearances in independent and made-for-TV movies. She divided her time between Minneapolis and Tinseltown, shuttling from school events to movie shoots with her mother in tow. Cook's starlet status crystallized in 1999, when she starred opposite Freddie Prinze Jr. in the Pygmalion retelling She's All That (1999). Her on-screen transformation from ugly duckling to ravishing beauty scored several teen-oriented awards and made Cook a hot commodity in Hollywood. She signed for a handful of plum follow-up roles, including a troubled adolescent in Sylvester Stallone's Get Carter (2000), a frontier gal in Texas Rangers (2001), and the caterwauling lead in the live-action version of Josie and the Pussycats (2001). Cook now lives primarily in Los Angeles, but she returns home frequently to visit with friends and family. Her father,

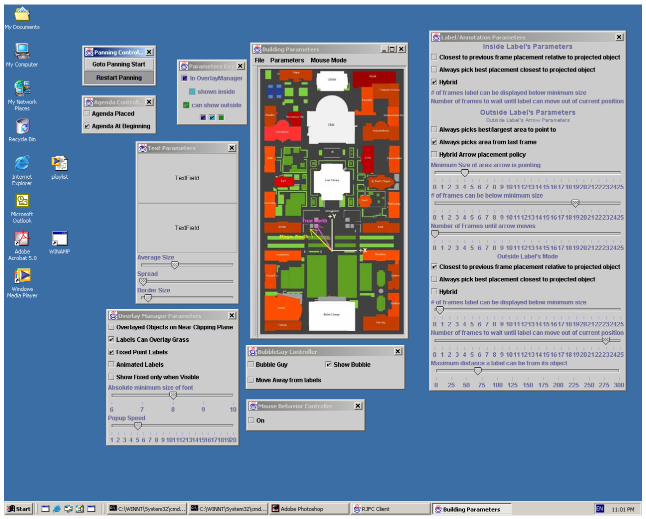
(b)



```
*admin*   You have joined Home
<Simon>: Hey guys
<Will>: Whats sup guys
<Yoav>: Lets go to room RJFC, i'll create it!
*admin*   You have left Home
*admin*   You have joined RJFC
<Simon>: Now this is much better!
<Will>: Tell me about it
<Will>: ok, i'm going back to Home
<Simon>: Bye will
<Yoav>: See ya later man
*admin*   You have left RJFC
*admin*   You have joined Home
*admin*   You have left Home
*admin*   You have joined RJFC
*admin*   You have left RJFC
*admin*   You have joined Home
<Yoav>: hello
```

Will
Simon
Yoav

(c)



Panning Control
Goto Panning Start
Restart Panning
Agenda Control
Agenda Placed
Agenda At Beginning
Text Parameters
TextField
TextField
Average Size
Spread
Border Size
Building Parameters
File
Parameters
Mouse Mode
Overlay Manager Parameters
Overlayed Objects on Near Clipping Plane
Labels Can Overlay Grass
Fixed Point Labels
Animated Labels
Show Fixed only when Visible
Absolute minimum size of font
Popup Speed
BubbleGuy Controller
Bubble Guy
Show Bubble
Move Away from labels
Mouse Behavior Controller
On
Label/Annotation Parameters
Inside Label's Parameters
Closest to previous frame placement relative to projected object
Always pick best placement closest to projected object
Hybrid
# of frames label can be displayed below minimum size
Number of frames to wait until label can move out of current position
Outside Label's Parameters
Always picks best/largest area to point to
Always picks area from last frame
Hybrid Arrow placement policy
Minimum Size of area arrow is pointing
# of frames can be below minimum size
Number of Frames until arrow moves
Outside Label's Mode
Maximum distance a label can be from its object
Start
RJFC Client
Building Parameters
11:01 PM

(d)

Figure 8: Screen shots of (a) the RJFC viewer displaying (b) a notepad demo, (c) a chat room demo, and (d) a virtual environment control system. The RJFC system is capable of providing the rich user interaction that one would expect from a local application, including dialog boxes, mouse-over detection and server generated events.

Event	To Srv	To Client	To Srv	To Client	To Srv	To Client
Connect	724613	12553	12261	27343	69181	56623
Log In	0	0	82152	4687	0	0
Open Application	39607	9364	24613	4509	0	0
Idle (1 min, static mouse)	0	0	12660	6200	0	0
Idle (1 min, anim. mouse)	0	0	24810	9217	0	0
Idle (1 min, no mouse)	0	0	6390	2200	0	0
Idle (1 min, full screen)	0	0	1709	2960	0	0
Typing (1 min, 382 chars)	377159	135392	79617	74304	0	0
Cut Paragraph	125969	38786	79618	74304	0	0
Paste Paragraph	91811	29430	1437	1899	658	461
Copy Paragraph	153062	41224	2508	2979	295	460
Find in Paragraph	154306	40750	5157	2312	1390	1965
Save Document	187768	49384	10621	5684	1238	2004
New Document	60940	19498	1360	2123	689	875
Open Document	114686	25120	6590	3144	1423	1396
Resizing from full screen	741322	60056	180576	16185	0	0
Drag 1/4 size window	697433	64530	134016	212275	0	0
Drag mouse across screen	308324	99300	1471	3726	0	0
Tear down	9618	3006	1779	2097	1667	2210

Figure 9: A comparison of the number of bytes transmitted over the network by RJFC, RDP (as implemented in Windows 2000 Terminal Services) and VNC (the AT&T reference implementation) when simple operations were performed in a notepad application. We determined that with VNC, the vast majority of the bytes being transmitted were caused by mouse movement. Therefore we had an expert user perform the experiment to minimize the number of bytes sent. The VNC system was configured to use the standard hextile encoding method for transmitting raster data between the server and client. It should be noted that the VNC system would occasionally have difficulty knowing when to update the screen and would occasionally need to be "woken up" with mouse movements that increased the amount of bytes transferred. A Web-based method using HTTP/HTML would not be able to provide the same level of user interactivity and therefore was not included in the tests.

frame buffer approach, while permitting rich user interaction without artificially introduced latency. Figure 9 is a comparison of the bandwidth consumed by RemoteJFC and the AT&T VNC remote frame buffer system.

All thin-client systems need some kind of software browser or viewer that must reside in permanent storage on the client computer. Because the Web-based approach and the RemoteJFC approach both transmit high-level information across the network, the size of the client software package is much larger than that of the VNC viewer. The size of a typical Web browser download is about 25 megabytes, as compared to the VNC viewer, which can be about 110 kilobytes. The RemoteJFC viewer lies somewhere in between: the library adds 2.5 megabytes to a Java Runtime Environment, which can vary in size from 3 to 15 megabytes. In addition, the VNC viewer memory image when attached to an 800x600 desktop computer consumes 1.5 megabytes of RAM, whereas both the Web browser and RemoteJFC viewer require approximately ten times that amount. This also results in faster startup times for the VNC viewer than a Web browser or the RemoteJFC Viewer.

Overall, the remote frame buffer approach is much "thinner" than the Web-based and RemoteJFC approaches and is capable of running on less powerful hardware, but requires much more network bandwidth to operate effectively.

6. CONCLUSIONS AND FUTURE WORK

We believe that the distributed user interface toolkit approach, as embodied in RemoteJFC, represents a powerful competitor to the existing methods of delivering thin-client applications to the user. We are currently considering a number of possible directions in which to extend our work.

One attractive possibility is the implementation of another Doclet API code generator to automatically convert desktop JFC applications to thin-client RemoteJFC applications. We will also attempt to add new features to the RemoteJFC protocol. For example, since our protocol is client-side stateless, it is possible to create a collaborative groupware version of the RJFC toolkit. We would then be able to compare our system directly with the rich body of research in that area. In addition, we will consider augmenting the RemoteJFC toolkit to support a hybrid client capable of handling some events locally on the client while transmitting other events to the server.

Finally, we believe that exploring how to optimize the RJFC protocol could provide insight into the information complexity of a user interface. By knowing exactly how much information we are transmitting across the network, we can gain a better understanding of a user interface's efficiency and how to improve it.

7. ACKNOWLEDGMENTS

This research is supported in part by NSF Grant IIS-98-17434 and gifts from Microsoft and Intel. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the NSF or any other organization supporting this work.

8. REFERENCES

[1] An ASP you can grasp: The ABCs of active server pages. http://msdn.microsoft.com/workshop/server/asp/ASPover.asp.

[2] D. Anderson. Experience with Flamingo: A distributed, object-oriented user interface system. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications*, pages 177–185, Oct 1986.

[3] A. Baratloo, M. Karaul, H. Karl, and Z. M. Kedem. An infrastructure for network computing with Java applets. *Concurrency: Practice and Experience*, 10(11-l3):1029–1041, Sep 1998. Special Issue: Java for High-performance Network Computing.

[4] T. Berners-Lee and D. Connolly. Hypertext markup language—2.0. RFC1866, 1995.

[5] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol—HTTP/1.0. RFC1945, 1996.

[6] C. Binding. The architecture of a user interface toolkit. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 56–65, Oct 1988.

[7] Borland JBuilder. http://www.borland.com/jbuilder.

[8] The Common Gateway Interface. http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[9] Citrix Metaframe. http://www.citrix.com/products/metaframe/.

[10] D. Conolly and L. Masinter. The text/html media type. RFC2854, 2000.

[11] Doclet Overview. http://java.sun.com/j2se/1.3/docs/tooldocs/javadoc/overview.html.

[12] W. Edwards, E. Mynatt, K. Peterson, M. Spreitzer, D. Terry, and M. Theimer. Designing and implementing asynchronous collaborative applications with Bayou. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pages 119–128, Oct 1997.

[13] Forte tools: Forte for Java. http://www.sun.com/forte/ffj.

[14] J. Golick. Network computing in the new thin-client age. *netWorker: The Craft of Network Comptuting*, 3(2):30–40, 1999.

[15] J. Gosling, D. S. H. Rosenthal, and M. Arden. *The NeWS Book: an introduction to the Network/extensible Window System*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1989.

[16] Graphon RapidX. http://www.graphon.com.

[17] R. Hill, T. Brinck, S. Rohall, J. Patterson, and W. Wilner. The Rendezvous architecture and language for constructing multiuser applications. *ACM Transactions on Computer-Human Interaction*, 1(2):81–125, 1994.

[18] S. Hudson and I. Smith. Supporting dynamic downloadable appearances in an extensible user interface toolkit. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pages 159–168, Oct 1997.

[19] Insignia Solutions Ntrigue. http://www.insignia.com.

[20] ISAPI Extensions Overview. http://msdn.microsoft.com/library/psdk/iisref/isgu9kqf.htm.

[21] ISO/IEC JTC1/SC2/WG11. MPEG. *ISO*, Sept. 1990.

[22] Java Applets. http://java.sun.com/applets/.

[23] Java Foundation Classes: Now and the Future. http://java.sun.com/products/jfc/whitepaper.html.

[24] JavaServer Pages: Dynamically Generated Web Content. http://java.sun.com/products/jsp.

[25] J. Jing, A. Helal, and A. Elmagarid. Client-server computing in mobile environments. *ACM Computing Surveys*, 31(2):117–157, 1999.

[26] A. Joseph, A. Lespinasse, J. Tauber, D. Gifford, and M. Kaashoek. Rover: A toolkit for mobile information access. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, pages 156–171, Dec 1995.

[27] D. Kristol and L. Monulli. HTTP state management mechanism. RFC2109, 1997.

[28] T. G. Lewis. Where is client/server software headed? *IEEE Computer*, 28(4):49–55, Apr 1995.

[29] S. Li, Q. Stafford-Fraser, and A. Hopper. Integrating synchronous and asychronous collaboration with VNC. *IEEE Internet Computing*, 4(3):26–33, May-Jun 2000.

[30] B. MacIntyre and S. Feiner. A distributed 3D graphics library. In *SIGGRAPH 98 Conference Proceedings*, pages 361–370. ACM SIGGRAPH, Addison Wesley, Jul 1998.

[31] Macromedia, Inc. http://www.macromedia.com/.

[32] J. McCormack, P. Asente, and R. Swick. *X Toolkit Intrinsics—C Language Interface*, Aug 1991.

[33] Microsoft Corp. *Microsoft Visual C++ MFC Library Reference*. Microsoft Press, Redmond, WA, 1997.

[34] Microsoft Windows 2000 Terminal Services. http://www.microsoft.com/windows2000/guide/server/features/terminalsvs.asp.

[35] Modular Toolkit Environment. IEEE 1295.

[36] NSAPI FAQ. http://developer.netscape.com/support/faqs/champions/nsapi.html.

[37] J. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[38] PHP: Hypertext Preprocessor. http://www.php.net.

[39] A. Prakash and H. Shim. DistView: Support for building efficient collaborative applications using replicated objects. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pages 153–164, Oct 1994.

[40] T. Richardson, Q. Stafford-Frasor, K. Woord, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, Jan-Feb 1998.

[41] J. A. Rody and A. Karmouch. A remote presentation agent for multimedia databases. In *International Conference on Multimedia Computing and Systems*. IEEE Computer Society, May 1995.

[42] K. Saar. VIRTUS: A collaborative multi-user platform. In *Proceedings of the 4th Symposium on VRML*, pages 141–152, Feb 1999.

[43] R. Scheifler and J. Gettys. The X window system. *ACM Trans. on Graphics*, 5(2):79–109, April 1986.

[44] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. The interactive performance of SLIM: a stateless, thin-client architecture. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 32–47, Dec. 1999.

[45] C. Schmuckmann, J. Kirchner, and J. Haake. Designing object-oriented synchronous groupware with COAST. In *Proceedings of the ACM 1996 Conference on on Computer Supported Cooperative Work*, pages 30–38, Nov. 1996.

[46] SCO Tarantella. http://www.tarantella.sco.com.

[47] Symantec PC Anywhere. http://www.symantec.com.

[48] Visual Cafe. http://www.webgain.com/products/visual_cafe.

[49] The Virtual Reality Modeling Language. http://www.web3d.org/technicalinfo/specifications/.

[50] D. J. Zukowski, A. Purakayastha, A. Mohindra, and M. Devarakonda. Metis: A thin-client application framework. In USENIX, editor, *The Third USENIX Conference on Object-Oriented Technologies and Systems (COOTS), June 16–19, 1997. Portland, Oregon*, pages 103–114, Berkeley, CA, USA, June 1997. USENIX.