Global Teleporting with Java: Towards ubiquitous personalised computing

Kenneth R. Wood*, Tristan Richardson*, Frazer Bennett*,
Andy Harter*, Andy Hopper*†

*The Olivetti & Oracle Research Laboratory
Old Addenbrooke's Site
24a Trumpington Street
Cambridge
CB2 1QA
United Kingdom

†University of Cambridge
Computer Laboratory
Pembroke Street
Cambridge
CB2 3QG
United Kingdom

Abstract

Previous work has described *teleporting*, an approach to mobile computing in which it is the user's personal application environment which is mobile rather than the hardware on which the applications run. In this paper we describe a new teleporting system which makes the user's environment available on any machine in the world running a Java-compliant web browser. We present some preliminary experimental results together with discussions of security and performance issues.

1 Introduction

The essence of mobile computing is having one's personal computing environment available wherever he or she happens to be. Traditionally this is achieved by physically carrying a computing device (say, a laptop or PDA) which may have some form of intermittent network connectivity, either wireless or tethered.

However, in [6] another form of mobility was introduced in which it is the user's *applications* which are mobile. The user does not carry any computing platform but instead is able to bring up his or her applications on any nearby machine exactly as they appeared when last brought up in this way, there or elsewhere. This form of mobility is called *teleporting* and has been used continuously and fruitfully by many members of our laboratory for the last three years.

Clearly, the machines to which one can teleport in this way must be attached to a network and must provide a common interface at some level. In our case the network is our local area network (Ethernet and ATM) and the common interface is the X Window System[1] [8]. When we teleport, our personal X

[1] The X Window System is a trademark of The X Consortium.

session with all of its associated applications in their latest collective state is transferred from one host's display to another within the lab. This allows us, for example, to walk into someone else's office and immediately call up and interact with our personal working environment on their machine, alongside any other working environments currently displayed there.

In our current work we are attempting to extend this idea from our local area network to the entire internet using Java[2] as the common interface. It is still our personal X sessions which are made mobile, but now they can appear within any browser which can execute Java applets, anywhere on the internet.

Although in theory the original form of teleporting could be used across the internet, it would be restricted to hosts running an X server, and, even more problematically, would contravene the X security policy implemented by most system administrators. Perhaps most importantly, though, our approach to teleporting across the internet is intended to take advantage of the rapid global proliferation of the World Wide Web. Web browsers are available in a dramatically growing range of locations, including corporate, personal, and even public-access sites. Thus, the ability to call up one's personal computing environment on any such browser will enable nomadic computing on a truly global scale[3].

2 Teleporting

The teleporting system offers a means of redirecting the user interface of applications which run under the X window system. In X, a display is controlled by an *X server* and applications are clients of the server, communicating with the server using the *X protocol.* This protocol allows applications to create windows on the screen and receive input from the keyboard and mouse.

The teleporting system introduces a level of indirection between applications and the display. This is done using a special X server, known as a *proxy server*. (See Figure 1.) Applications are made mobile by running them as clients of the proxy server, within a *teleport session*, rather than within a traditional *X session* under a real X server.

Unlike a real X server, the proxy server does not have a screen, keyboard and mouse (a *display*) of its own. Instead, it is able to make use of the display of some real X server. To the real X server, the proxy server appears just like an ordinary set of clients. In this way, the output of the proxy server's clients will be sent to the screen of the real X server, and their input will come from its input devices.

The proxy server makes its clients mobile because it is able, upon request, to break down its connections with the real X server and, if desired, re-build them with another. This occurs without the clients' needing to be aware of this activity. The result is that the teleport session with all the clients' windows can disappear from one screen and (possibly much later) re-materialise on another.

[2] Java is a trademark of Sun Microsystems.

[3] Note that the next release of X, codenamed "Broadway", will also address some of these issues.

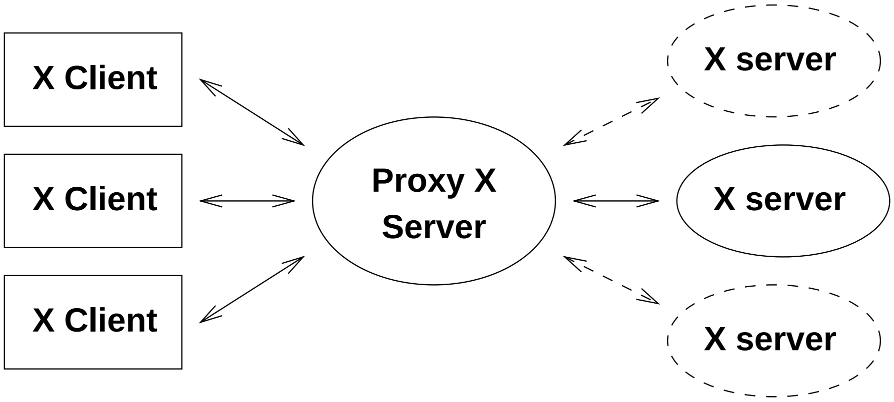

Figure 1: Proxy Server

In our lab we have made extensive use of the teleporting system in our everyday work, and we have found that the ability to move our working sessions on the fly from office to office, office to meeting room, office to kitchen, office to home, etc, is extremely useful, especially for the sort of peripatetic collaboration which tends to go on in a typical research lab. After having used the teleporting system for our primary work environment, most of us would find it difficult to go back to a static login session.

3 Teleporting in Java: The Concept

Given how useful we have found teleporting to be, it is only natural to want to extend its range beyond the immediate environment of our lab and homes. In order to do this, of course, we need a network and common interface widely available in places over which we have no control. The World Wide Web and Java provide just such an infrastructure.

Web browsers are now available almost everywhere a networked computer can be found, and Java is emerging as the dominant technology for enabling programs to be downloaded and executed within a browser. Thus, we decided that an initial attempt at global teleporting should be based on the idea that a working session is identified with a web page containing a Java applet. Simply by pointing any Java-capable browser at this page, we cause the corresponding working session to appear within the browser where we interact with it in the natural way.

This is, in fact, exactly what we have done. We call the implementation *VNC* (for *Virtual Network Computer*[4]) and Figure 2 shows a typical VNC session which has been brought up in Netscape.

Having pointed Netscape to the web page corresponding to the session shown, we can use the mouse and keyboard to manipulate windows and graphical applications, edit files, and so on, just as if we were logged in to the session in the normal way. We can also browse other pages, returning to the VNC page

[4]We originally used the name *JavaTel* (for *Tel*eporting in *Java*) but changed to VNC to avoid confusion with Java Telephony applications.

whenever we want to do some work there. Furthermore, we can go to another physical location and point a different browser at the VNC page, whereupon the session will appear in the new browser and vanish from the old one. (We also provide the capability to disconnect a VNC session from one browser without having it appear in another. It can then be called up from the same or another browser at any later time.)

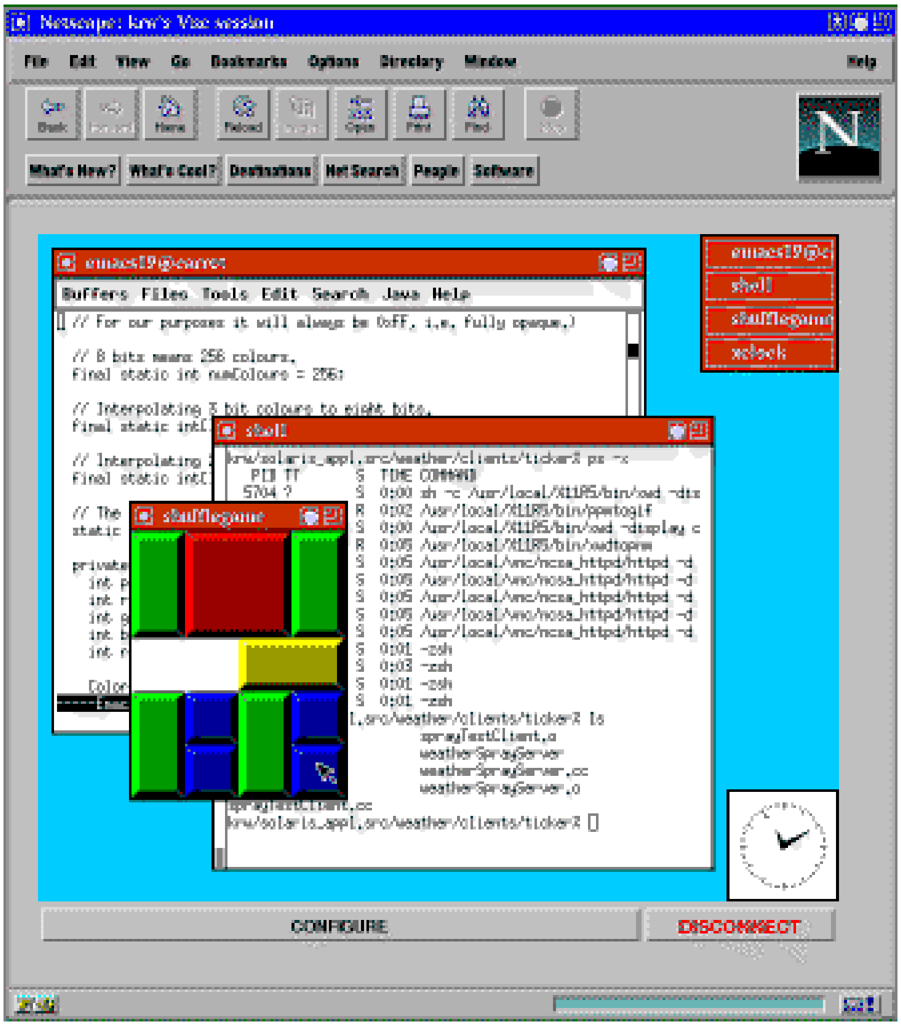

Figure 2: A sample VNC session

4 Teleporting in Java: The Mechanics

In order to move the concept of teleporting to the wider arena of the internet, we make use of another sort of proxy which we call a *remote frame buffer* (or RFB) service. In our case, the RFB service is provided by an *RFB X server* which is just a standard X server to which we have added two simple features:

1. The RFB X server provides a TCP socket interface on which it will accept mouse and keyboard events using a simple protocol. When any such event is received, it is processed just as if it had been generated by a mouse or keyboard connected directly to the X server.

2. The RFB X server provides another TCP socket interface on which it will accept requests for information about the state of the screen display. In reply to such a request, the RFB X server sends details of those regions of the screen which have changed since the last such request. These details are sent as a set of bitmapped rectangles which represent the changes to the screen.

   For example, if there were a word-processor running in the X session and a lower-case letter '`l`' had been typed into it since the last request, then in response to the next request the RFB X server would send the following data:

   - the (x,y) pixel coordinates indicating the screen position of the top-left corner of a rectangle containing the '`l`'.
   - the width and height of the rectangle containing the '`l`', in pixels.
   - a block of width*height pixel values which represent the rows of pixels which make up the rectangle containing the '`l`'. For example, if the rectangle were 5 pixels wide and 11 pixels high, and the values 255 and 0 represented white and black respectively, then the block of pixel values might look like Figure 3.

   The changes to the screen might, of course, be much more extensive than the addition of an '`l`'(for instance, they might include the appearance of a set of complex images) and in this case the RFB X server would send as many rectangle specifications as are required to describe the changes.

Together the protocols used on the two socket interfaces described above comprise the *RFB protocol*. By connecting to these socket interfaces, an *RFB client* running on a remote (non-X-aware) device can provide seamless interaction between a user and the X server. The RFB client need only understand the RFB protocol, i.e. how to send input events (mouse and keyboard) and how to receive and render screen-change rectangles. The complete RFB protocol is somewhat (though not a great deal) more complex than that illustrated here, as it allows different synchronization modes and also provides for compression of the screen rectangles when this is deemed necessary.

The remote device for which we originally developed the RFB service (and on which we are still using it) is a *video tile*, a pen-based ATM-connected display [5]. The RFB client running on the tile passes pen events as mouse events to the RFB X server and puts all screen changes it receives onto the tile display, thereby allowing interaction with X applications on the tile.

It soon became apparent that simply by writing an RFB client in Java we could use exactly the same approach to provide interaction with an X server from within a Java applet and hence from within a web browser. The RFB

```
255 255 255 255 255
255   0   0 255 255
255 255   0 255 255
255 255   0 255 255
255 255   0 255 255
255 255   0 255 255
255 255   0 255 255
255 255   0 255 255
255 255   0 255 255
255   0   0   0 255
255 255 255 255 255
```

Figure 3: A bitmapped rectangle containing the letter 'l'

client applet opens sockets to the mouse/keyboard and screen-change ports of the RFB X server. It then sends mouse and keyboard events from Java's AWT down the appropriate socket as they occur, and paints changed rectangles to the screen as they arrive from the RFB X server, again using the AWT. The result is the appearance of an X session within a web browser, as described above and depicted in Figure 2.

Figure 4 shows the structure of the VNC system. Note that the figure represents the state of a connected VNC session which presumes the following background steps:

- on the application host, at some earlier time the user will have initiated his VNC session by running a startup script. This script starts the RFB X server with its associated X session (which is completely user-configurable just as any other X session is) and creates an HTML file which encapsulates the corresponding RFB client applet. (Note that the X session is started on a virtual display, a feature of the current release of X, so as not to tie up a physical display.)
- on the browser host, the user will have pointed the browser at the URL of the generated HTML file, whereupon the RFB client applet will have been downloaded and run, and the connections shown in the figure established.

Because of current applet security restrictions, the HTML file must be served by an HTTP server running on the same machine as the RFB X server. When the level of Java security is adjustable, as is planned, this restriction may be removed if desired.

It is important to realize that although the effect of the VNC system is to provide teleporting across the internet, the method by which this is achieved is entirely separate from that employed by the original teleporting system. That is, the VNC system is a conceptual but not a technical extension of teleporting. The most significant technical difference between the VNC and teleporting systems is that the VNC client is much thinner than the teleporting client. In order to bring up a teleport session on a client machine, that client must be running a full X server, whereas to bring up a VNC session the client needs to run only a very small (about 20K) Java applet.

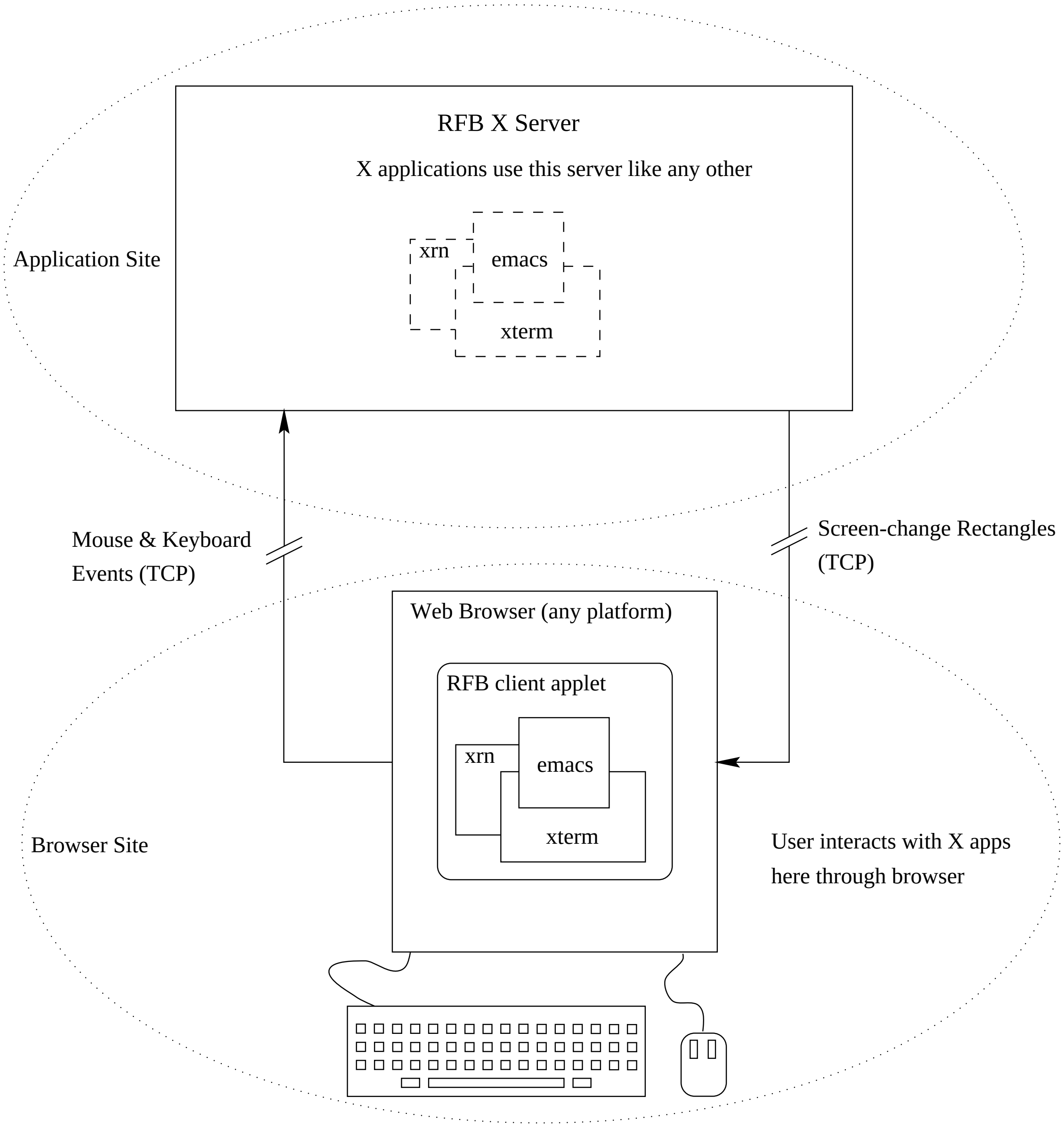

Figure 4: Structure of the VNC system

5 Experiments

The bulk of our experiments have been local to our lab. That is, the application host and the browser host have both been machines on our local network. However, we have also done a number of longer-distance experiments, many between Cambridge and Oxford, and a few between Cambridge and places further afield, including Munich, Princeton, and San Francisco. The application host has always been a UNIX[5] machine, but we have brought up the VNC session in browsers running on both UNIX and Windows[6] machines.

In general, the interactive performance is quite acceptable except where the Java AWT on the browser host machine is particularly slow. Because our early investigations revealed the performance bottleneck to be the Java image-drawing capabilities, we made a considerable effort to improve matters on that front by designing a compression scheme which puts the transmitted screen rectangles into a form particularly suited to fast rendering in Java.

For X sessions typical of work within our lab, our compression scheme reduces network traffic by a factor between 5 and 10. As we put more distance between the application and browser hosts, and as Java's image-drawing speed improves, interactive performance increasingly depends on the speed of network communication. Hence the ability of the compression scheme to reduce network traffic is becoming more important than its ability to reduce drawing time at the client, and we therefore intend to investigate other compression schemes as well.

Tables 1 to 5 give a rough indication of VNC interactive performance for three specific tasks in various situations. In all cases, the measurements are in seconds and represent an average of between 3 and 5 trials of the same task. The application host was always on our local network in Cambridge. The 'text' task comprised scrolling through 10 email messages using rmail in Emacs, the 'menu' task comprised pulling down a window manager menu 10 times, and the 'image' task comprised rendering a 27K JPEG image.

For tables 1 and 2, the VNC client applets were run in Netscape 3.01 on relatively slow Unix machines. This was the only type of machine available to us at the remote site in Oxford, and we ran the tests on a similar local machine for comparison purposes. We ran the VNC session in Oxford by telneting to the remote machine, and running Netscape there with the display set to our local machine. Thus, the Oxford figures include network round-trips which would not have occurred had we been physically located at the remote site.

From these tables, we can see that on slower browser host machines interactive performance is not really acceptable even when the machine is local. Moving to a remote slow machine (remembering that our figures include essentially doubled network traffic) reduces interactive performance further, but does leave it on the same order of magnitude.

Tables 3, 4, and 5 present figures for the three tasks on faster browser host machines on our local network. Here we see that the interactive performance is

[5] UNIX is a trademark of the Novell Corporation.
[6] Windows is a trademark of the Microsoft Corporation.

Task	Time (sec)
text	68
menu	35
image	25

Table 1: VNC times, Netscape 3.01, slow Unix browser host (Sun IPX), local (Cambridge)

Task	Time (sec)
text	142
menu	55
image	51

Table 2: VNC times, Netscape 3.01, slow Unix browser host (Sun IPX), remote (Oxford)

greatly improved, to the extent that it is almost comparable to native X server performance. These figures suggest that the VNC system provides a suitable method for accessing X applications from Java-based thin client machines on a local network, and our subjective experience within the lab has confirmed this.

Our subjective experience of using the VNC system on faster browser hosts from such remote sites as Munich, Princeton, and San Francisco suggests that the VNC system also provides a suitable method for the global teleporting which was the original aim of the work. In several cases, members of our lab have used their VNC sessions to do useful work while travelling, and they report that although interactive performance is by no means comparable to native X server performance on a local machine, it is certainly adequate for the occasional session of nomadic computing.

We plan to carry out more controlled experiments using fast browser hosts from remote sites, and at the same time to experiment with different techniques for improving VNC performance in these situations. Our experience to date makes us confident that the VNC approach is highly effective and that its efficiency for remote use will eventually approach its demonstrated efficiency for local use.

Task	Time (sec)
text	14
menu	13
image	11

Table 3: VNC times, Netscape 3.01, fast Unix browser host (Ultra Sparc), local

Finally, in an interesting (if somewhat confusing to imagine) experiment, we combined our original teleporting system with the VNC system. Since a VNC session provides remote access to a standard X server, we can teleport to a VNC session in exactly the same way that we teleport to any other X server, thereby

Task	Time (sec)
text	9
menu	7
image	5

Table 4: VNC times, Netscape 3.01, fast Windows browser host (Pentium 166), local

Task	Time (sec)
text	8
menu	7
image	4

Table 5: VNC times, Microsoft Internet Explorer 3.0, fast Windows browser host (Pentium 166), local

causing a teleport session to appear within a web browser. Thus, for members of our lab the VNC system can provide not only the capability of transporting a specially set-up X session around the internet, but also the capability of transporting their everyday working environment.

6 Security

There are clearly important security concerns about a system which makes full access to one's personal computing environment as easy as clicking on a web hyperlink. Since web browsers are designed to limit the damage an applet can do to the local environment, whereas no such damage limitation exists for an X session, the VNC user is more worried about others invading her computing environment than a browser-provider is worried about a VNC session invading his.

The minimal security requirement is therefore password protection for access to a VNC session, and we have implemented this level of security using a challenge-response authentication mechanism based on strong encryption. Better still would be ongoing verification of traffic integrity using a one-time session key, or even full encryption of the session traffic since, for example, one might want to read sensitive email in a VNC session. We are currently investigating various possible approaches to these additional levels of security, none of which presents any fundamental problem, since the appropriate technology is available now. Even with the strongest level of security, however, one must still trust the web browser and the host on which it runs.

A different approach would be to isolate the resources which are reachable from a VNC session such that unauthorised access to the session could not lead to any significant loss. This would, however, considerably limit the usefulness of the VNC system.

7 Reliance on X

One can interact with a VNC session on any platform running a Java-capable web browser including, for example, Windows-based PCs, Macs, and UNIX workstations. However, the applications within the session must use X as their display technology. It is possible, though, within an X session running on one machine to interact with a Microsoft Windows session running natively on a PC. This requires a multi-user version of Windows NT such as NTrigue, WinCenter, or WinDD, and we have successfully used WinCenter in this way to provide access to Windows applications within a VNC session, as can be seen in Figure 5.

Figure 5: A sample WinCenter VNC session

An alternative solution, which we are looking into now, would be to create an *RFB Windows server* comparable to the RFB X server which we have described above. The RFB Windows server would provide the same RFB protocol interface as the RFB X server, but would deliver mouse and keyboard events to, and retrieve screen changes from, a Windows session rather than an X session. This would immediately enable direct VNC-style interaction with Windows applications since the RFB client depends only on the RFB protocol and is therefore completely independent of the application host's windowing system. The recent emergence of ICA[7]-based systems for exporting Windows interfaces to web pages provides a model for this approach.

8 Conclusions and Related Work

The different facets of mobile computing and the related field of ubiquitous computing are explored in, for example, [1, 3, 4, 7]. The concept of ubiquitous personalisation extends these ideas to encompass the notion that wherever the user might be and using whatever device, the computing and/or control interface is in some way tailored to the user's specifications. The VNC system is a step in that direction insofar as it makes available on virtually any internet-connected machine a computing environment completely determined by the user.

Another X-based architecture for mobile applications on the web is presented in [2] together with a good overview and analysis of the central issues in designing such an architecture. The approach taken is similar to ours, although it relies on X on the browser side as well as the application side.

The next phase of our research aims to provide a means for the user's environment to adapt to radically different types of devices while still retaining some element of personalised behaviour. The devices we are considering here range from the most basic (say, mobile phones) through to desktop PCs and workstations. Perhaps this can be achieved by designing browsers which interpret web documents and applets in ways suitable to these different devices, as is apparently the thrust of Netscape's Navio Communications venture and, to some extent, Microsoft's Windows CE. However, on its own this will not be enough. There is a wide spectrum of possible distributions of both state and intelligence between the application and browser hosts, and the current VNC system represents just one extreme point on this spectrum. The success of this next research phase will depend on careful investigation and exploitation of other points along this spectrum, and this is where we intend to direct our research efforts in the first instance.

References

[1] Rajive Bagrodia, Wesley Chu, Leonard Kleinrock, and Gerald Popek. *Vision, issues, and architecture for nomadic computing. IEEE Personal Communications*, 2(6):14–27, December 1995.

[7] ICA is a trademark of Citrix Systems Inc.

[2] Daniel Dardailler. *Mobile GUI on the web. The X Advisor*, 1(2):19–28, July 1995. *The X Advisor* is at `http://www.unx.com/DD/advisor/`.

[3] Tomasz Imielinski and B.R. Badrinath. *Wireless computing. Communications of the ACM*, 37(10):19–28, October 1994.

[4] Arup Mukherjee and Daniel Siewiorek. *Mobility: A medium for computation, communication,* and *control.* In Luis-Felipe Cabrera and Mahadev Satyanarayanan, editors, *Workshop on Mobile Computing Systems and Applications*, pages 8–11. IEEE Computer Society Press, 1994.

[5] ORL web site. *Tile module.* `http://www.orl.co.uk/tile.html`.

[6] Tristan Richardson, Frazer Bennett, Glenford Mapp, and Andy Hopper. *Teleporting in an X window system environment. IEEE Personal Communications*, 1(3):6–12, Third Quarter 1994. Also available as ORL Technical Report 94.4, ORL Ltd, Trumpington Street, Cambridge CB2 1QA, England.

[7] Mark Weiser. *Some computer science issues in ubiquitous computing. CACM*, 36(7):137–143, 1993. See `http://www.ubiq.com/` for further information on this and related work.

[8] X Consortium web site. *X consortium welcome document.* `http://www.x.org/`.