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.

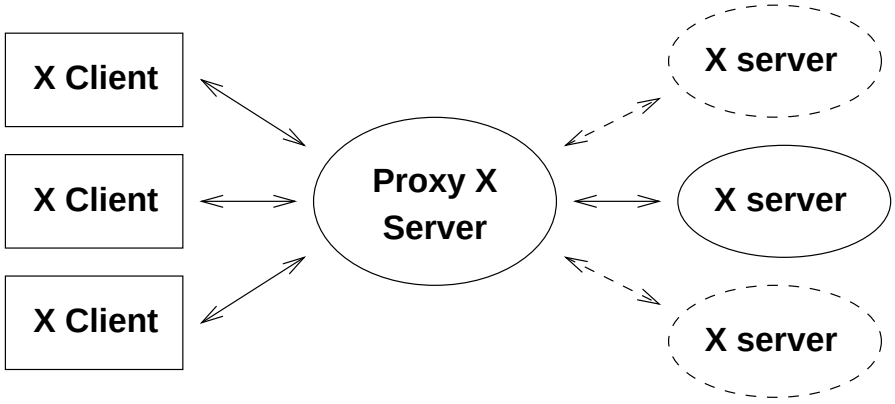

Figure 1: Proxy Server

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 *Tele*porting 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.)

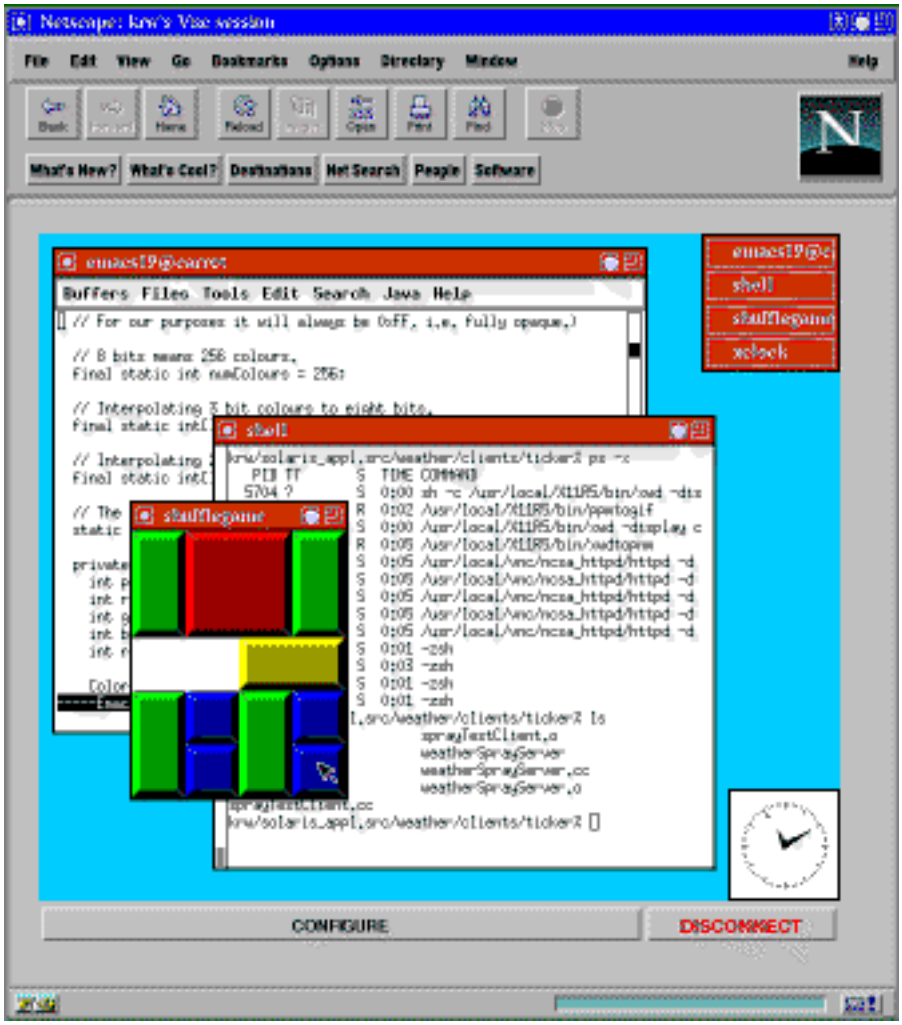

Figure 2: A sample VNC session

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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.