Teleporting - Mobile X Sessions

Tristan Richardson

Abstract

This paper examines issues involved in making an X session mobile. A *mobile X session* is one which is not fixed to a particular X display, but can be materialised on demand at any suitable display. The *Teleporting System* developed at Olivetti Research Laboratory (ORL) is a tool for experiencing mobile X sessions. It provides a familiar, personalised way of making temporary use of X displays as the user moves from place to place. When linked to location facilities such as those provided by the Olivetti Active Badge[1] System the traditional log-in process can be almost entirely eliminated, allowing the nomadic user to easily make use of computing resources which are to hand.

Introduction

Traditional interaction with character-based terminals has taken the form of a session where the user "logs in" by entering a user-id and password, interacts with applications one at a time from the command line, and eventually "logs out" to end the session. Workstations using a windowing system such as X have extended this traditional session to allow the user to interact with multiple applications at the same time, each being displayed in a different window. However, the process of logging in and out has remained largely the same.

When the user runs a number of applications there is a considerable amount of state involved in a session. It is unrealistic to expect a user to log out every time the workstation is left temporarily. Users tend to stay logged in to a workstation for long periods of time, often for a whole day and sometimes for several days or weeks.

[1] Active Badge is a registered trademark of Ing. C. Olivetti & C., S.p.A.

Tristan Richardson (trichardson@cam-orl.co.uk) is a Research Engineer at Olivetti Research Laboratory, Cambridge, England.

Making temporary use of a workstation where someone else is logged in can be a problem. It can be considered antisocial to interfere with another user's session, and interacting with another's session is likely to be frustrating since the applications will be running as the wrong user-id and will be configured in a way which is unfamiliar. Logging the original user out before logging oneself in is likely to be considered antisocial as well, because it loses the state inherent in the original user's session.

In an environment where people are mobile, portable computers are one way of accessing computing resources, but they will always be restricted due to size, weight and power considerations. A relatively unexplored alternative is to facilitate the temporary use of workstations and other equipment which is to hand. In X, one obvious way of making this possible is via a mobile X session, which is not fixed to a particular X display, but can be materialised on demand at any suitable display.

Related work

One way of appearing to move an X session from one display to another is to shut down the session on the original display, and start up a session with the same clients in the same state on a new display. This could be achieved in X with the help of a special client known as a "session manager".

The purpose of the session manager is to record the state inherent in a session in such a way that the clients can be restarted in the same state. Using a special protocol it tells clients when the session is about to be shut down so that they can save their state. Previously this has been attempted using the "WM_SAVE_YOURSELF" protocol of the ICCCM. Release 6 of X has introduced a new "Session Management Protocol" as an attempt to rectify the deficiencies in the old protocol. However, the vast majority of existing clients use neither of these protocols. Furthermore, even if clients are written which participate in the session management protocol, there may be aspects of a client's state which are difficult or impossible to save. For example there is no way of "saving" the state of a unix process which has pipes or sockets connected to other processes.

In general, the only way to avoid losing the state of a client when moving an X session is not to shut down the client at all. Instead the client must be able to close its connection with one server, connect to the new server, create all its windows and other server resources as before, and so continue where it left off on the original server.

There have been attempts to develop such "mobile applications" in X. Trestle [Manasse93] and XTk [Jacobi92] are both toolkits designed for the development of applications whose windows are redirectable between displays. Interesting new applications can be developed using such a toolkit, but clearly this approach cannot be used to add mobility to existing, unaltered applications.

In order to make existing applications mobile without alteration, a different approach is required. This is to put a level of indirection between the client and the real server, called a *proxy server* (also called a pseudoserver).

Proxy servers are not a new idea. Many have been written in order to duplicate an application's output on to multiple displays [Abdel-Wahab91] [Altenhofen90]. However, these are

not directly applicable to client mobility. They all use one X server as a "master" server to which the clients must remain connected throughout their lifetime. There is no way in which this connection for a client can be closed without shutting down the client.

More recently, new proxy servers have been developed specifically for the purpose of application mobility. These include our "teleporting" proxy X server, *tpproxy* [Richardson94], and *xmove* [Solomita94]. These two proxy servers are in many ways very similar, but there are some important design goals in xmove which make it unsuitable as a basis for making X sessions mobile.

Xmove is designed specifically for moving clients on an individual basis. Although one instance of xmove can cope with several clients, it deals with each client separately, not as a coherent session with a root window and a window manager.

Much effort has gone into making a client running through xmove appear on the real X server just as it would without xmove. Private protocols for communication between clients running through xmove and clients running directly on the real X server (needed for features like drag-and-drop) must be understood and translated by xmove. This means that each time new inter-client protocols are developed, an extension to xmove must be written which understands the protocol. This is unnecessary when one considers a mobile X session as a whole.

The Olivetti Teleporting System

The teleporting system's main component is *tpproxy*, a proxy X server for making X sessions mobile. A mobile X session running through tpproxy is called a *teleport session*. Tpproxy acts more like a real X server than other proxy servers. It has its own root window, on which the windows of its clients can be placed under the control of a window manager.

Normally, tpproxy behaves as a filter, much like other proxy servers. It takes requests from its clients, translates them and sends them on to the real server, and in the reverse direction takes replies, events, and error packets from the server and sends them on to the appropriate client. While doing this, tpproxy also records all changes to the state inside the server relevant to its clients. When told to materialise the teleport session on a new server, tpproxy can generate requests to the new server to bring up to date the state of each of its clients as seen by the new server. Most of the issues involved in this have been covered elsewhere [Chung91] [Richardson94] [Solomita94].

When told to dematerialise, tpproxy disconnects from the real server. It continues to process requests from its clients, and for most requests, simply records any state changes before discarding the request. For some requests, however, the client needs a response from the real server. A client which issues such a request while the teleport session is dematerialised will be suspended, and no more requests from that client will be processed until the session is rematerialised on a real X server.

Security

Authorisation for connection by a client to tpproxy is just the same as for any X server, using the normal authorisation mechanisms, the "host list", and the "magic cookie" scheme.

Clients which are in tpproxy's host list, or which provide the correct value of the magic cookie are allowed to connect to tpproxy, while any others are rejected.

A separate and more difficult problem is how tpproxy is given authorisation to access real X servers. Within ORL this is achieved by running tpproxy as a special privileged user-id which has access to the magic cookies for all real X displays at ORL. This scheme clearly does not scale outside a single administrative domain. For teleporting to displays at other sites a different scheme is needed whereby the cookie for the real display is passed to tpproxy just before it connects to the display. Further research is needed to find a secure way in which permission to connect to an X display can be given temporarily to visitors from other sites.

Integration with the underlying X session

The teleport session's root window appears as a normal client window on the real X server, decorated in the usual way by the window manager in the underlying X session. Each of tpproxy's clients' windows appear as subwindows of this "pseudo-root window", controlled by the window manager running in the teleport session. This is much like the Virtual Screen program [Lin92].

If no window manager is running on the real X server, the teleport session's root window simply covers the whole screen. This most commonly occurs when no-one is logged in and the XDM login window is showing. In this case tpproxy must also unmap the XDM login window to release XDM's keyboard grab.

Another aspect of integration with the underlying X session is how inter-client communication is affected. Because tpproxy deals with an X session as a whole, it is usually only necessary to consider inter-client communication between clients of the same proxy X server. Any private protocols between these clients need not be understood by the proxy server in the same way that they need not be understood by a real X server.

In general, inter-client communication will not work between a client in the underlying session and a client in the teleport session. One reason for this is that the space of identifiers for server resources as seen by a client of the proxy server is different from that seen by a direct client of the real server. Any private protocol in which these clients exchange resource identifiers is likely to fail. One important exception to this rule is the normal selection mechanism, which is part of the X protocol. Thus cutting and pasting of text between a client in a teleport session and a client in the underlying session works in the normal way.

Keyboard personalisation

In a normal X session, users can personalise the keyboard according to their preference by altering the keyboard (and modifier) mapping[2]. This facility should also be provided in a mobile X session. Tpproxy achieves this by keeping two different keyboard mappings, one for the teleport session, and one for the underlying session running on the real server. When the input focus enters the teleport session's root window, tpproxy saves the mapping for the underlying session and installs the mapping for the teleport session. Similarly, the teleport session's mapping is saved and the underlying session's mapping installed when the input

[2]Users can also customise the pointer mapping, key-click, auto-repeat, etc, but no users of the teleporting system have found these to be important.

focus leaves the teleport session's root window.

Because a keyboard mapping is specific to a physical X display, it makes no sense to copy the keyboard mapping when the teleport session is moved to a new real X server. What tpproxy does upon materialisation is to copy the default keyboard mapping for the real X server to be the starting point for the teleport session's mapping. The user can then modify the teleport session's mapping as desired, completely isolated from the mapping in the underlying session. A mechanism is provided by which these modifications to the keyboard mapping can be automatically applied upon materialisation, so that for a given display, the keyboard is always personalised to the user's preference.

Display heterogeneity

X servers have a wide variety of characteristics such as screen size, depth and framebuffer format, which are exposed to the client. Tpproxy has to present a view to all its clients of the kind of X server it is, and maintain this consistently, even when the underlying real X server has substantially different characteristics. We review some of the problems this causes and the solutions which tpproxy adopts, some of which are only partial solutions.

Screen size

Screen width and height vary greatly between different X servers. When the teleport session is moved from a large display to a smaller one, obviously not all of the root window can be visible at once[3]. The teleporting system provides two solutions to this, both of which treat the window on the real X server as a "viewport" on to tpproxy's root window. This viewport is similar to the facilities offered by "virtual desktop" window managers such as *tvtwm* (available in the contributed software section of R5 and R6).

The first solution involves running a special window manager called *tpwm* which is a version of tvtwm, modified to be aware of the possible resizing of the viewport. The disadvantage of this solution is that it forces the user to adopt a particular window manager. Since most users at ORL use a twm-like window manager (tvtwm is based on twm), this has been found to be the best solution.

The second solution allows the use of any window manager, and effectively adds virtual desktop functionality by the use of a special client called *tpviewport*. The disadvantage of this approach is that there is a potential conflict between the window manager and tpviewport, especially if the window manager already has a virtual desktop feature.

Colour handling

Colour handling is one area where the X protocol exposes a lot of detail about the server to the client. There is a wide variety of possible combinations of framebuffer depth and mapping of pixel values to colours on the screen. Tpproxy's solution to these problems is a compromise which we have found to cope satisfactorily with nearly all clients. However there will always be clients which cause problems when teleporting to monochrome servers and servers which do not allow writable colormaps.

[3]The same problem occurs when tpproxy's root window is managed by the underlying window manager and the window is resized.

The current version of tpproxy normally tells its clients it is an 8-bit "StaticColor" server, meaning that it has a read-only colormap. If the real server also has a read-only colormap, then all pixel values must be translated between the client and the server. This means that functions such as exclusive-or on pixel values may not work as the client expects. In addition, image data has to be translated a pixel at a time, which can be somewhat time-consuming.

If the real server allows writable colormaps then tpproxy will normally install its own colormap so that functions such as exclusive-or on pixel values may be freely performed. If the server happens to also be an 8-bit server then image data can be passed through untranslated, virtually removing any performance problem.

Controlling The Teleporting System

There are two issues in controlling where a mobile X session is displayed. Only the correct user must be able to control the teleport session, so some sort of authentication is needed. Secondly, the user must somehow specify which X display the teleport session should appear on. The proxy server offers a textual command-line interface which meets these requirements. In addition, its integration with the location facilities developed at ORL provides a simpler and richer interface.

The command-line interface

The command-line interface to the proxy server offers one single command, `teleport`. This command is used to initialize a teleport session, as well as control its materialisation to, and de-materialisation from, any X display within the building. The `teleport` command can be executed from within another user's session by quoting a user name and password. However, finding a suitable terminal window in the other user's session may not always be trivial. A preferable approach to controlling the teleport session is not to have to interfere with the underlying session at all.

The automated control interface

Integration with the Active Badge Location System [Want92] [Harter94] has introduced a level of automation to the teleporting procedure, making it much easier to make temporary use of X displays.

The Active Badge System provides a means of locating individuals and equipment within a building. Personnel and equipment wear a small infra-red transmitting *Active Badge*, which has a unique identity. With a networked infra-red receiver in each room, the Active Badge System creates a dynamic database describing the location of individuals and equipment within the building. The teleporting system uses this database to determine which X displays are co-located with a particular individual.

In addition, the Active Badge can be used as a control mechanism. One of two small buttons on the badge is used to select an appropriate X server within the room, and the other is used to confirm the selection and materialise the user's teleport session.

The badge acts as the required authentication mechanism, since it has a unique identity, and is generally worn by its owner at all times. An additional security feature is that when a

user leaves a room in which they have teleported, the teleporting system can automatically dematerialise their teleport session without any action by the user.

Conclusion

The original goal of the teleporting system, to add mobility to X sessions based on existing X applications, has been achieved. The system is in everyday use and for many users has completely replaced the standard log-in session.

The main feature of the teleporting system is that by using a proxy server it makes existing applications mobile, without the application being aware of any redirection. However, it has also provided a framework for experimenting with new applications which are aware of their own mobility. The first of these to be developed were tpwm and tpviewport, described earlier, which adapt to the screen size of the real display. Further ways of adapting to a smaller screen have been devised, including the ability to automatically iconify particular windows, and to reduce the font size for text windows.

Using the Active Badge database, mobile applications can be aware not only of the properties of the X display they are materialised on, but also of the surrounding environment. For example, applications can send documents to a "virtual" printer, which will automatically print the document on the user's nearest printer. Another simple example is a "display selector" tool which shows a menu of all the X displays in the same room and can be used to redirect the teleport session between them.

Teleporting is an important example of a nomadic style of working using an infrastructure of static resources. In a suitably-equipped environment, a mobile user need not be constrained by the limitations of equipment that can be carried. Instead, more powerful, networked computing devices can be used in a way that is familiar and personalised for the needs of each user.

References

[Abdel-Wahab91] Hussein M Abdel-Wahab and Mark A Feit. *XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration.* Proceedings of IEEE TriComm 91: Communications for Distributed Applications & Systems, Chapel Hill, North Carolina, April 1991.

[Altenhofen90] Michael P Altenhofen. *Erweiterung eines Fenstersystems für Tutoring-Funktionen.* Diploma Thesis, Universität Karlsruhe, Karlsruhe, Germany, 1990.

[Chung91] Goopeel Chung, Kevin Jeffay and Hussein M Abdel-Wahab. *Accommodating late-comers in a distributed system for synchronous collaboration.* Technical report TR91-038, Department of Computer Science, University of North Carolina at Chapel Hill, October 1991.

[Harter94] Andy Harter and Andy Hopper. *A Distributed Location System for the Active Office.* IEEE Network, Special Issue on Distributed Systems for Telecommunications, January 1994.

[Jacobi92] Christian P Jacobi. *Migrating Widgets.* Proceedings of the 6th Annual X Technical Conference, January 1992.

[Lin92] Jin-Kun Lin. *Virtual Screen: A Framework for Task Management.* Proceedings of the 6th Annual X Technical Conference, January 1992.

[Manasse93] Mark S Manasse. *The Trestle Toolkit.* Proceedings of the 7th Annual X Technical Conference, January 1993.

[Richardson94] Tristan Richardson, Frazer Bennett, Glenford Mapp and Andy Hopper. *Teleporting in an X Window System Environment.* IEEE Personal Communications Magazine, Third Quarter 1994.

[Solomita94] Ethan Solomita, James Kempf and Dan Duchamp. *XMOVE: A Pseudoserver For X Window Movement.* The X Resource, Issue 11, O'Reilly & Associates Inc, July 1994.

[Want92] Roy Want, Andy Hopper, Veronica Falcao and Jonathan Gibbons. *The Active Badge Location System.* ACM Transactions on Information Systems, January 1992.