The journal of Apple technology.

Volume Number: 19 (2003)
Issue Number: 3
Column Tag: Section 7

X11: A C of Window Systems

or possibly a Fortran?

by Rich Morin

Taken on its own terms, X11 is a stunning success. As www.x.org/X11.htm proclaims:

> The X Window System, more simply 'X' or 'X11', is judged worldwide to be one of the most successful open source, collaborative technologies developed to date. It is the de facto standard graphical engine for the UNIX and Linux operating systems and provides the only common windowing environment bridging the heterogeneous platforms in today's enterprise computing. The inherent independence of the X Window System from operating system and hardware, its network-transparency, and its support for a wide range of popular desktops are responsible for its continuing and growing popularity.
>
> All major hardware vendors support the X Window System. Many third parties provide technologies for integrating X Window System applications into network or personal computer environments under DOS, Windows, Windows 9x, and Windows NT, while thousands of independent software developers provide X Window System applications. The worldwide community of users of the X Window System currently exceeds 30 million.

X11 is such a dominant technology in Unix circles that shipping a Unix system without an integral copy of X11 would be like omitting a C compiler or, erm, hiding the command line. Nonetheless, Apple's initial position was fairly plausible: "X11 support is an opportunity for third-party developers".

Several developers, indeed, accepted the challenge. Tenon Intersystems (www.tenon.com) has produced a well-regarded proprietary X11 implementation for OSX. OroborOSX (oroborosx.sf.net), XDarwin (www.xdarwin.org), and XonX (www.mrcla.com/XonX) provide free distributions of X11 for OSX.

Nonetheless, Apple's release of an "official" implementation of X11 is certain to interest many users, including some who have been wary of trying X11. And, indeed, Apple is in a position to make X11 interoperate with Aqua in ways that a third-party vendor might not have been able to duplicate. I would expect, however, to see the third-party developers adopt any enabling technologies that Apple makes available.

The Apple implementation includes "standard X11 display server software, client libraries, and developer toolkits". The intention, as confirmed by an Apple spokesman, is to place X11 at about the same level of OS support as, say, Carbon. That is, better integrated than Classic, but possibly missing some Cocoa functionality.

So, we can expect to see X11 apps appearing commonly on OSX. Despite the best efforts of all concerned, however, it is not clear that the fit will be all that good: the goals, designs, and philosophy of the two systems are wildly different. Let's look at each of these, using OSX as our point of reference.

X VS. X

The goals of X11, as noted above, are "network transparency" and "bridging heterogeneous platforms". An X11 user can access an X11-based application on either the local machine or on any machine which is reachable over the network.

Apple Remote Desktop, SSH, Timbuktu, and VNC all allow remote access, but none of these provides the level of integration and performance that X11 does. In short, if you want network-transparent, cross-platform support for GUI-based apps, X11 is your answer.

X11's design differences reflect its differing goals. An X11 "client" application sets up callback functions and issues UI requests, just as a Cocoa or Carbon app would. In X11, however, everything is translated into a serial stream of data, suitable for sending over the network.

In place of Apple's Cocoa and Carbon APIs, X11 has "widget sets" (e.g., GTK+, Motif, Qt, Xaw). In place of the Finder, X11 has "window managers" (e.g., fvwm, kvm, mwm, sawfish, twm) and "desktop environments" (e.g., CDE, Gnome, KDE).

Making all of this play nicely with Aqua (e.g., supporting drag and drop) isn't going to be an easy job. In fact, some things will simply fail to translate. If X11 has a widget that Aqua doesn't (or vice versa), there isn't much that a "glue layer" can do about it.

X11's basic philosophy is likely to cause the worst dislocations, however. Apple has worked for years to develop, document, and promote the Macintosh "Human Interface Guidelines". As a result, Mac users feel comfortable starting up a new app, knowing that the command keys and menus will all be familiar. No such guidelines are present in X11:

> "It is important to keep in mind that the protocol is intended to provide mechanism, not policy."
>
> Robert W. Scheifler, June 1987
>
> http://www.ietf.org/rfc/rfc1013.txt

With no central authority specifying policy, no consistent user interface developed. Each of X11's widget sets, window managers, desktop environments, and applications was free to go off in its own direction. The resulting inconsistency will grate on seasoned Mac users.

Porting X11 apps

Now that Apple is providing an X11 development system, porting an X11 app shouldn't be much harder than porting any other Unix-based app. Nonetheless, I'd like to see Apple provide an X11 and Unix analog to the "Carbon Dater" (an analysis tool for Carbonizing Classic apps); this could ease the task of identifying the OSX equivalents to unavailable include files, library calls, system calls, etc.

While I'm fantasizing, I'd also like a way for Aqua apps to be accessible via the X11 protocol and an X11 toolkit that implements the Aqua look and feel. Both technical and legal issues surround these pipe dreams, however, so I'd best get back to reality.

If an app is written for certain X11 toolkits (e.g., GTK+, Qt, Tk), it may be possible to link it to Carbon or Cocoa. This could provide access to useful Cocoa features and/or improvement in appearance and/or performance. Look around for interface libraries, ala:

If the app only needs to run on a single machine, a port to native Cocoa may be appropriate. If the app has a clean separation between its Model and View code, the rework shouldn't be all that hard. Unfortunately, the real problems

may not be obvious at the beginning of the effort.

X11, Layer by Layer

I have made some hand-waving claims about the differences between the OSX model and the X11 model. For those that may be interested, here is a (somewhat) detailed discussion of X11's structure.

At the bottom layer, X11 is a communication protocol. The units of discourse are user events (e.g., mouse and keyboard actions) and displayable items (e.g., characters and pixel rectangles, also known as "pixrects"). An X11 "server" runs on the user's desktop, making the screen and input devices available to the "client" application.

The low level of the X protocol limits the amount of processing power that the server can usefully provide. Once items display instantly, extra computing power is irrelevant. Compare this to Aqua, Display PostScript, Java, Javascript, or Sun's ill-fated NeWS, all of which can offload rendering and other processing to a separate processor.

In addition, X11 is not well suited to displaying shapes other than rectangles. Although any shape can be rendered as a set of rectangles, a PDF-based description of an arbitrary shape is likely to be a lot shorter than the pixrect equivalent.

At the next level, programming interfaces, X11 provides only the bare essentials. You can draw characters and pixrects, as implied above, but doing anything else requires "widgets". The good news is that there are a lot of widgets to choose from; the bad news is that there are a lot of widget sets to choose from. The sets all do roughly the same things, but the look, feel, and APIs vary.

Similarly, window management can be performed by any of a few dozen "window managers" (really, special-purpose X11 clients). These are similar to "skins" and "themes", but their influence extends into the feel, as well as the look, of the apps they are presenting.

Moving up to the application level, we see even more diversity. Historically, X11 app designers picked their own "hot keys", menu layouts, and other UI options. As a result, there was little consistency at this level.

Recently, some X11-based "desktop environments" have begun to promote more consistency in widgets, window management, and applications. The good news is that these environments are becoming popular; the bad news is that there are several desktop environments to choose from. Gnome and KDE are the most popular in Open Source circles. CDE is found on many proprietary Unix systems, but Sun intends to transition Solaris and its users to Gnome.

In Summary

The X mantra of "mechanism, not policy" has resulted in a prolific, but anarchic collection of APIs, window managers, and look and feel decisions. Like the many variations of the C programming language (e.g., C, C++, Objective-C), X11 has evolved into a "sea" of window systems, each with its own vagaries.

Apple's frameworks may be appallingly complex, but they are internally consistent, logical, and produce consistent results on the desktop. Where Microsoft Windows is simple (nay, simplistic), consistent, and ugly, X11 is powerful, inconsistent, and ugly.

This inconsistency, appearing at all levels of X11, means that users cannot build reliable habits. Apple's developer community has spent a great deal of time and effort on making sure that OSX apps act in a consistent manner; the X11 developer community has not, and it shows.

Co-existence?

Many OSX users and programmers will be inclined to ignore X11 entirely, but there are reasons why you might wish to consider a co-existence strategy. If you need a particular X11 app or some of the capabilities that X11 provides, you'll have to allow X11 into your world. Another reason might be the plethora of free (both libre and gratis) X11 apps.

Some institutions will find X11 unavoidable, because critically necessary apps are only available under it. The engineering and scientific communities have been using Unix and X11 for decades; OSX and Aqua are still (largely unproven) newcomers. Until the apps you need get ported to Cocoa, there really isn't any alternative to using X11.

There is also the possibility that you will need to create an OS-independent and/or network-capable application. As noted above, Apple has no real solution in this area; Apple Remote Desktop is fine for sharing a remote screen, but it isn't capable of supporting generic, distributed applications.

Even if your work doesn't require a specialized app, however, you may be interested in trying out some X11 packages. You can pick these up off the Internet (e.g., via Fink or the GNU-Darwin project), but you'll probably have less trouble with a pre-packaged collection.

BSDMall (www.bsdmall.com) has produced a set of X11-based "Office Applications for Mac OS X". The set includes word processors, spreadsheets, graphics tools, games, and more. You may recognize tools such as AbiWord, Calc, Chimera, Draw, Gaim, Gimp, Impress, Math, and OpenOffice. If you're interested in trying X11, this might be a good starting point.

More generally, for a good list of X11 resources, try Kenton Lee's "Technical X Window System and Motif WWW Sites" page, http://www.rahul.net/kenton/xsites.html. He has done a really fine job of collecting and organizing X11-related links!

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.