The Future is Coming: Where the X Window System Should Go

Jim Gettys
Cambridge Research Laboratory, Compaq Computer Corporation.
Jim.Gettys@Compaq.com

Abstract

The X Window System was developed as a desktop window system, in a large (for its time) campus scale network environment. In the last few years, it has escaped the desktop and appeared in laptop, handheld and other mobile network devices. X from its inception has been a network transparent window system, and should thrive in this environment. Mobility forces a set of issues to surface that were only partially foreseen in X's design. For one reason or other, the hopes for the design were not entirely realized.

Our original view of X's use included highly mobile individuals (students moving between classes), and a hope, never generally realized for X, was the migration of applications between X servers. Toolkit implementers typically did not understand and share this poorly enunciated vision and were primarily driven by pressing immediate needs, and X's design and implementation made migration or replication difficult to implement as an afterthought. As a result, migration (and replication) was seldom implemented, and early toolkits such as Xt made it very difficult. Emacs is about the only widespread application capable of both migration and replication, and it avoided using any toolkit.

You should be able to travel between work and home or between systems running X at work and retrieve your running applications (with suitable authentication and authorization). You should be able to log out and "park" your applications somewhere until you retrieve them later, either on the same display, or somewhere else. You should be able to migrate your application's display from a handheld to a wall projector (for example, your presentation), and back again. Applications should be able to easily survive the loss of the X server (most commonly caused by the loss of the underlying TCP connection, when running remotely).

There are challenges not fully foreseen: applications must be able to adapt between highly variable display architectures. Changes to the X infrastructure in recent work make this retrofit into modern toolkits appear feasible, enabling a much more dynamic view of applications. Also, applications must be able to adapt between very different resolution displays (more than an order of magnitude) and differing pointing devices.

I cover the changes and infrastructure required to realize this vision, and hope to demonstrate a compelling part of this vision in action. This vision provides a much more compelling vision of what it means for applications to work in your network. With the advent of high speed metropolitan and wide area networks, and PDA's with high speed wireless networks, this vision will provide a key element of the coming pervasive computing system.

1 Introduction

The X Window System [SG92] architecture lay fallow for at least five years, roughly the period from 1995 to 2000, though many of the extensions defined after the base X11 architecture were questionable at best [Get00]. The good news is that the original X architecture enabled the development of desktop systems such as Gnome [dI99] and KDE [Dal99]. But to again match and then exceed other systems such as Microsoft Windows and Apple's Aqua [App] in all areas, serious further work is necessary. This paper attempts to outline areas where base window system work is required to again make X competitive with other current commercial systems, specifically in the current large scale network environment.

I believe that migration and replication of running X applications is key to the near term future environment. A number of issues made replication and/or migration difficult. These include (and some of which are discussed in detail in [GP01]):

- Fonts have been server side objects, and there is no guarantee the fonts available on one server are available on the other (slightly mitigated by the existence of font servers).

- Graphics operations in X can be made to drawables (windows or off screen memory, called "pixmaps"). The X protocol only guarantees 1 bit deep drawables. Displays of different depths might share no other depth. Futhermore, early toolkits typically did not handle image rendering at all, exposing this disparity of displays completely to applications.

- Resource ID's (for windows, pixmaps, fonts, etc) and Atoms are per server identifiers, and have to be remapped. These were typically exposed directly to applications, though sometimes with some abstraction (for example, partially hidden in a widget data structure).

- Pseudocolor displays again presented toolkits, applications, or pseudoservers major headaches: there would be no guarantee the color resources would be available when needed. It took longer for pseudocolor to become unimportant than expected due to the not fully understood economics that drove displays down market at the same functionality. Thankfully, monochrome (1 bit) and pseudocolor displays are now a thing of the past, for all intents and purposes.

2 Client Side Fonts

While server side fonts clearly provide major headaches for migration and replication (since there is no guarantee that the fonts the application needs are available on the X servers being used), client side fonts are needed on general architectural grounds. This section lays out why.

2.1 Large Scale Distributed Systems Deployment

In large scale commercial systems, applications are often run remotely from the X server to a desktop system or X terminal, and therefore applications won't be deployable until both ends of the communications have been updated. The X Window System, due to its network transparency, presents the problems of large scale distributed systems. It is infeasible to update all X servers to support new font technology.

The X extension mechanism is at best only a partial solution, since if an application relies on an extension's existence, it cannot be deployed until the X server has been upgraded. Many older systems can never be updated with new X extensions. Application developers won't use a new feature if it badly restricts the market for their software.

Applications that will work (even if somewhat slowly) always trump applications that won't run at all, from the point of view of an ISV (who wants to sell product), or an open source developer who wants their application used widely.

2.2 Historical Observation about Opaque Server side fonts

From the beginning, X's minimalist font abstraction has presented challenges to applications interested in serious typography. Due to X's inadequacies, serious applications from the very beginning of X's history have had to work around X's fonts, and coordinating X's fonts with those required for printing has been a 15 year long migraine headache.

Applications need sufficient information for either X's use, or for printing use (something we ignored in the design of the X protocol). Therefoer, to be useful to applications on systems, any access to fonts must be able to access *any* information in the font files, since they will continue to evolve.

There have already been four generations of font formats over X's history:

- Bitmap 1983-1990
- Speedo 1991
- Type1 1992
- TrueType 1997

OpenType, in the short term future, represents a fifth format. Approximately every 5 years, there is likely to be a new generation of and format for font files, with increasing quality (along some dimension, whether it be typographic, or character set, . . .). Any server side font solution, therefore, will always be difficult to deploy in this distributed environment and will end up chasing a moving target.

As each new font technology has deployed, applications have needed to access the new data contained in the font files for the new information. X itself has not evolved to do more than rendering the bits on the screen, and has made it difficult or impossible for applications to share this information.

Over this period, fonts themselves have become much larger. When X was designed, fonts typically had fewer than 128 glyphs (ASCII), and the number of fonts were small, and, since the fonts were bitmap fonts, the metrics were static. The number of round trips to retrieve font metrics were small, and the size of the metrics themselves were small. This situation contrasts greatly with the present.

Many modern fonts have thousands of glyphs and common desktop systems may have hundreds or even thousands of fonts. Searching among these fonts for correct style and code point coverage requires applications to retrieve (typically at startup) very large amounts of metric data, and, to add insult to injury, it is insufficient for most printing applications. Due to this growth in the number and size of fonts, transporting even X's minimal font metrics from the X server to clients now usually swamps transporting the glyph images actually used to the X server from applications. Furthermore, for X to use a scalable font (now the dominant form of fonts used), it must render the font at the specified size, often incurring large startup delays for many applications, before font metrics can be returned to applications, even if few or no glyphs are used (if the font is unsuitable).

The new fontconfig [Pac02] library provids a mechanism for finding the fonts required to fulfill an application's needs, which in concert with Freetype [TT00] allows access directly to the font files the application needs to find the glyphs it needs to render. Round trips to the X server usually dominate performance. Xft [Pac01b] uses the Render extension [Pac01a] when present to cache and render glyphs from the fonts at the X server, avoiding round trips. The Xft2 library now uses only core protocol image transfers if Render is not available, making client-side antialiased fonts (with subpixel decimation) deployable universally. This allows application writers to rely on the new font rendering technology immediately.

Client side fonts therefore have many benefits:

- lower total bandwidth required for modern applications with modern fonts,
- many fewer server round trips,
- incremental behavior: no long startup delays for font rendering,
- as a result, much faster startup time,
- applications can directly access font files to enable printing,
- future font formats can be added without client/server deployment interdependencies,
- an application is guaranteed that the fonts it needs will be usable on any X server in the network, easing migration and replication of applications.

2.3 Remaining work

Some data from modern applications show these benefits [Pac00a] at current screen resolutions (typically 100DPI); more data on a wider selection of applications is needed to confirm this early data.

Worth further investigation is the scaling of glyph data as a function of screen resolution. While sending outlines is certainly possible, this suffers from the deployment problems noted above as this information has changed in each font generation. Naïvely, you might think that the glyph image data would scale as the square of the display resolution, but this is not the case. Even a simple LZW compression algorithm reduces the glyph data to less than linear as a function of resolution; investigation of two dimensional compression techniques is still needed [Clo02].

Further work will be needed therefore to choose a compression technique to ensure that as screen resolution increases, the glyph transport remains efficient. Use of a stripped down PNG [Bea97] like algorithm (to remove redundant header information) is a likely candidate.

Sharing fonts using standard network file systems is much simpler than the current font server situation. This approach can take advantage of the full file sharing / replication / authentication / administration infrastructure being developed for distributed file systems. Using existing infrastructure to enable central administration of fonts is easier than building extensions to the current custom X font service protocol. Applications can get the font information they want and need, and it leverages the full caching/replication protocol work going on in network file system design.

Updating the shared libraries (e.g. Xt, Xaw, Motif, etc) to support AA text is as simple a way to make antialiased text universal than updating the X server and is planned for later in 2002. Modern toolkits such as GTK and Qt have already been updated to support client side fonts, and deployment will be widespread by the end of 2002 on Linux systems.

3 Non-Uniformity of X servers

X servers have varied greatly. The core protocol only guarantees one bit per pixel in addition to the native depth of the screen. This provides a great challenge to applications that might want to migrate or replicate between displays. Further complicating this issue is that historically, toolkits have not provided facilities for rendering of images, relying on applications that need to render images to adapt to the screen correctly (a poor assumption). This means that the visual resources of X servers typically using Xt based toolkits become embedded directly into the applications code, not even abstracted by a toolkit. Pseudocolor displays further complicated the challenge: the color resources required to render an image might or might not be available when needed on an X server

A number of developments over the last four years mitigate this situation.

- The most common X server, XFree86, now uses a new frame buffer package FB, so that all depths are available on all X servers.
- GTK 2.0, and Qt, fully abstract screen resources in ways that completely hide the details of the screen from applications, and provide image abstractions to applications. Applications have to go out of their way to discover information that would make them dependent on the details of rendering of the screen.
- Pseudocolor displays are becoming rare, at long last. So this source of allocation headaches between screens is becoming moot.
- The RandR extension potentially provides the ability to render different depth displays onto a frame buffer, while allowing toolkits to select accelerated visual types and rerender when appropriate. Further work to complete the implementation of RandR is needed, however.

4 Non-Uniformity of Screen size

Screens now vary greatly in size and capability. On the lowest end, the IBM watch provides a 100 by 100 pixel screen. PDA screens are now commonly 320x240 resolution, of small physical size. More conventionally desktop displays are in the 1400x1000 pixel size, with flat panel resolutions increasing. On the high end, within the next two years, some organizations are building large display walls or “caves” with up to 8000x8000 resolution. This very wide disparity of displays provide serious challenges to applications.

Previous attempts to provide migration and replication were often implemented using X servers [GWY94a] that would talk to additional X servers and provide migration and replication services. This model clearly failes across such disparity of screen sizes.

A significant fraction of applications, however, are useful at several display sizes. Personal information management applications are a good example: you want to use these on PDA screens, your desktop, and potentially in a conference room environment on a very large screen. It is unlikely that a single user interface can span this range of display sizes and uses, or deal well with the diversity of pointing devices. GUI builders are a solution for many applications.

The early generation of GUI builders for X were all proprietary, and not universally available, and depended on Motif, which has also not been universally available. Few applications were built using these builders as a result.

The Familiar project has had significant success using Glade [Cha01] and GTK, for example, to provide user interfaces tuned for either portrait or landscape mode on the iPAQ PDA running Familiar Linux. I believe this approach is most suitable to enable many/most applications to be usable across this diversity of screen capabilities. In our examples, the user interface is defined using an XML description built using Glade, and reloaded at run time when the characteristics of the screen changes (in this case, when the screen rotation is changed). I believe this approach is most likely to succeed in avoiding always having to develop applications from scratch for different screen sizes.

Scalable fonts are now widespread, though we must continue to promote their use in existing applications. Some thought, however, needs to be completed on what the abstract size of fonts actually means. X provides the physical dimensions of the screen, and the number of pixels; it does not, however, provide the typical viewing distance required to allow applications to “do the right thing” with the size of objects to be displayed. In practice, most people’s gut feeling is that a 10 point font is a small, but readable size independent of viewing distance and pixel size; most applications don’t care about the true physical size of the display. We need some convention here to match users’ intuitions.

Notification of the exact characteristics of different X servers will be provided using the RandR extension. A convention to notify applications to migrate or replicate themselves is needed.

5 Resource ID’s

Resource IDs in X are values that only have meaning to a given X server, and are used to identify resources being stored at the X server, such as windows, pixmaps, server side fonts, etc.

Resource IDs are visible in old toolkits, but are generally buried in widgets. In principle, the process of unmapping/mapping/realizing of widgets provides a mechanism for migration. Unfortunately, almost any sort of drawing beyond basic text was left to applications in Xt; therefore real Xt based applications will likely be difficult to adapt to server migration. Modern toolkits no longer expose resource IDs, visual types or other X resources directly to applications, and since they provide image abstractions, very few applications now have any need to access server dependent X resources directly. This is paying off: patches now exist to allow GTK 2 applications to migrate between dissimilar screens on the same X server; this is much of the work required for migration between X servers. Completion of migration support in GTK and/or Qt is pressing.

6 Connection Failure

TCP connections on wireless networks are more fragile than wired networks, due to signal strength, multipath, and interference, not to mention roaming of the users between networks.

Roaming can be handled via MobileIP [Per95], and X does not pose any special issues here. IPv6, however, will require some additions in a few areas. While the X core protocol does not have any length dependencies on addresses, some of the ancillary protocols built on top were not built with such foresight and will require some tweaks.

Failure of TCP connections, however, are much more common in the wireless environment. Ideally, applications should be able to survive such losses, and work in toolkits described above should provide most of the required mechanisms. The X library itself, however, needs a small amount of work in this area, to inform toolkits that their connection to the X server has been lost, and allow reclaim of resources. Toolkits could then reconnect themselves to a pseudo X server to allow later retrieval by the user.

There is one possibly significant issue to resolve: in the X protocol, while it is perfectly acceptable (and common) for the X server to send Expose events to clients to request redraw of windows, the core protocol has no such mechanism for pixmaps. In the migration case without connection failure, pixmaps can be retrieved and migrated to the new server. But if the connection has failed, those pixmaps have almost certainly been destroyed, and the client would have to regenerate them. I do not know if this poses an issue to current toolkits or not, since they have taken a higher level of abstraction than Xt based toolkits did, and they may not have problems similar to Xt. Experimentation here is in order, though there have been successful application-specific implementations done in the past [Pac].

7 Replication

Replication poses issues to toolkits, particularly if you would like an application to present different user interfaces on different size screens. This may be hopeless with Xt based toolkits, but may work with modern toolkits.

Pseudo server approaches are likely to work well so long as screen sizes are roughly comparable, given the lack of pseudocolor, the new uniformity of X servers, and the RandR extension. I believe approaches such as HP’s SharedX server [GWY94b] will be useful where toolkit retrofitting is not feasible. Migration will allow sharing via a pseudoserver without a priori arrangement, as applications can migrate to a sharing server. Again, first hand experience is needed.

8 Transparency

Apple's Aqua system for OS X provides full use of alpha blended transparency. The Render extension for X provides part of the equivalent capability for X, by providing alpha blended text and graphics. It does not, however, provide for alpha blended windows, and work here is needed [Pac00b]. That will require significant reimplementation of the device independent part of the X server (which, thankfully, is only a few 10s of thousands of lines of code).

This work is also needed for full implementation of the RandR protocol, to enable rendering to windows that are a different depth than the frame buffer.

9 Authentication and Authorization

Migrating applications presents a serious security problem. You would not want an attacker to be able to hijack your applications, running as you, on your machine. Strong authentication will therefore be required for migration or replication to become widespread.

Support for Kerberos 5 is already in the X library; and SSH provides public key facilities. I do not yet know which may be most appropriate. SSH also provides encryption, which is less urgent (though sometimes vital) than strong authentication. Either or both may be appropriate, and work here is clearly needed, though clearly there are existing facilities than can be exploited to provide the required authentication.

Authorization is another piece of the puzzle, and thought is needed here to understand what is required.

10 Putting the Pieces together

So far, we've discussed the solutions to the individual problems. Here is a sketch of how this might work in practice, to pull the pieces together.

1. A user interface of some sort (whether it be via accelerometer, as discussed in the RandR paper, or something more conventional), would connect to the X server where the application is running, authenticated as you.
2. This application would send a message to the appropriate client(s) to migrate to a destination.
3. The application's toolkit would receive this message, and authenticate that it is from the user.
4. The toolkit opens a new connection to the new X server. If it fails, it would indicate so via some sort of ICCCM message to the user interface, to provide feedback to the user the migration was not possible.
5. The toolkit copies any pixmaps it needs from the original X server to the new one, or creates and rerenders those pixmaps from data it has on hand (probably better, if at all possible). Since all depths are available, the toolkit is guaranteed to be able to copy the bits if needed. This copy phase should occur first, to minimize the likelihood of failure due to inadequate server resources.
6. The toolkit unrealizes itself from the original screen, and realizes itself on the destination screen. If the size of the destination screen is significantly different than the origin screen, the toolkit might reload its user interface definition at this time to provide a more usable user interface. The RandR extension can be used both to provide information on which visual types have acceleration (and a clever toolkit might reconfigure itself to use one of those accelerated visuals). RandR's full implementation also enables rendering to a depth screen different than the actual framebuffer, if the toolkit cannot adapt.
7. At completion, the toolkit would signal its completion on the origin server of the migration it has migrated successfully. A similar sequence should work for connection failures, taking the loss of the TCP connection as a signal the application should attempt to migrate to a home server for later retrieval. Clearly some timeout should be imposed to reap applications after frequent failures. As noted above, there may be some unforeseen problems found in handling connection failure.

11 Plea for help

There are all sorts of projects of various sizes associated with the vision laid out here. I believe that most or all of this vision is achievable. I would like to take this opportunity to solicit others to help realize this vision, which I believe is compelling and should be a lot of fun.

Acknowledgements

The author would like to thank Keith Packard for conspiring on this architecture over the last several years. Additional thanks go to Chris Demetriou for his help in the publication of this manuscript.

References

[Ano01] Anonymous. Errata: Izzet Agoren's Kernel Corner, May 2001, Mitch Chapman's "Create User Interfaces with Glade" (July 2001). *Linux Journal*, 89:6–6, September 2001. See [Cha01].

[App] Apple Computer, Inc. AQUA - the mac OS X user experience. http://www.apple.com/-macosx/technologies/aqua.html.

[Bea97] T. Boutell et al. RFC2083: PNG (portable network graphics) specification. March 1997. http://www.ietf.org/rfc/rfc2083.txt.

[Cha01] Mitch Chapman. Create user interfaces with Glade. *Linux Journal*, 87:88, 90–92, 94, July 2001. See erratum [Ano01].

[Clo02] James H. Cloos, Jr. Glyph image compression techniques. Technical report, XFree86, 2002.

[Dal99] Kalle Dalheimer. KDE: The highway ahead. *Linux Journal*, 58:??–??, February 1999.

[dI99] Miguel de Icaza. The GNOME project. *Linux Journal*, 58:??–??, February 1999.

[Get00] James Gettys. Lessons learned about open source. In *Usenix Annual Technical Conference*, 2000. http://-www.usenix.org/publications/library/-proceedings/usenix2000/invitedtalks/-gettys_html/Talk.htm.

[GP01] Jim Gettys and Keith Packard. The X Resize And Rotate Extension - RandR. In *FREENIX Track, 2001 Usenix Annual Technical Conference*, pages 235–244, Boston, MA, June 2001. USENIX.

[GWY94a] Daniel Garfinkel, Bruce C. Welti, and Thomas W. Yip. HP SharedX: A tool for real-time collaboration. *Hewlett-Packard Journal: technical information from the laboratories of Hewlett-Packard Company*, 45(2):23–36, April 1994.

[GWY94b] Daniel Garfinkel, Bruce C. Welti, and Thomas W. Yip. HP SharedX: A Tool for Real-Time Collaboration. *HP Journal*, April 1994.

[Pac] Keith Packard. NCD's WinCenterPro: Networking NT applications using X. http://www.xfree86.org/ keithp/talks/-wincen.html.

[Pac00a] Keith Packard. An LBX postmortem. Technical report, XFree86 Core Team and SuSE, Inc, 2000.

[Pac00b] Keith Packard. Translucent windows in x. In *Fourth Annual Linux Showcase & Conference*, pages 39–46, Atlanta, GA, October 2000. USENIX.

[Pac01a] Keith Packard. Design and Implementation of the X Rendering Extension. In *FREENIX Track, 2001 Usenix Annual Technical Conference*, pages 213–224, Boston, MA, June 2001. USENIX.

[Pac01b] Keith Packard. The xft font library: Architecture and users guide. In *Conference Proceedings*. XFree86 Technical Conference, November 2001.

[Pac02] Keith Packard. Fontconfig - a shared font configuration mechanism. In *Conference Proceedings*. GNOME Users And Developer European Conference, April 2002.

[Per95] Charles Perkins. IP mobility support. Technical Report Internet Draft, IETF Mobile IP Group, January 1995. ftp://-software.watson.ibm.com/pub/mobile-ip/-draft-ietf-mobileip-protocol-08.txt.

[SG92] Robert W. Scheifler and James Gettys. *X Window System*. Digital Press, third edition, 1992.

[TT00] David Turner and The FreeType Development Team. The design of FreeType 2, 2000. http://www.freetype.org/freetype2/-docs/design/.