Networking with X Windows

X WINDOWS IS FAST BECOMING THE DE FACTO STANDARD FOR WORKSTATIONS. IN THIS ARTICLE, MIKE CHARNLEY-FISHER LOOKS AT THE FEATURES OF X AND EXAMINES SOME OF THE PITFALLS ASSOCIATED WITH USING X IN A NETWORK OF WORKSTATIONS.

MELVYN EVANS

Over the last 12 months, X Windows has become the buzzword in terms of windowing environments for computers generally, but especially for workstations. The reason is simply that 75 per cent of the world's computer manufacturers have decided to standardise on 'X' for their graphics devices. This, in turn is largely due to the fact that X is public domain, is free and provides all the necessary window functionality in a device-independent manner. The distinguishing factor that separates X Windows from other windowing environments is that it is totally network-transparent and offers more device independence than any other windowing system.

This article looks at each of the main goals of X and how they have been addressed. These are: device independence; network transparency; the ability to support concurrent display of multiple applications; freedom from 'policy'; and, finally, extensibility.

Device independence

X achieves device independence by providing a complete virtual 'display' in software. A 'display' to X includes the keyboard, mouse (and/or other pointing devices), screen or screens, and the controlling processor. This side of X, known as the 'server' is the main device-dependent part which has to be ported to any new hardware wishing to support X. This software emulation is a lot more extensive than most other windowing systems and includes device-independence routines for the keyboard, mouse/pointer and screen.

While most keyboards generate ASCII codes for the primary alpha-numeric characters, almost every manufacturer differs in the codes 'extra' keys generate when pressed. Coupled with the variations in things like the EBDIC character set, and some of the far eastern alphabets, this results in a vast range of possible codes to be recognised by truly portable software. In the past this has meant that in order to run IBM-compatible software you had to have an IBM-compatible keyboard, for example. X changes this by imposing a virtual keyboard between the physical device and the application software. As far as the application is concerned it handles

fixed key symbols known in X as 'Keysyms'. It is up to the Server to map the key codes to the 'Keysym'. In the same way, keys such as Shift, Alt, Ctrl, and so on, are known as 'modifier keys' and the status of each of these is available to the application.

Looking at the advantages: it is obviously device independent; the user is able to change the 'Keysym' to 'Key code' mapping to suit their preferences; applications can do the same; it can handle the large alphabets common in the Far East; the availability of the 'Modifier key' status saves a decode operation on the application side and key up and downs are available to the application (if the hardware can handle it). The disadvantages are that it is less efficient (in terms of speed and memory requirements); the ability to change keyboard mapping can upset other applications using the same display if those other applications are not written correctly and changing the keyboard mapping can confuse the user.

On the whole X does rely on having a pointing device of some sort but this could be emulated by the Server using a keyboard. As far as an application is concerned all it receives are 'pointer' movements and 'button' presses and/or releases. Up to five logical 'buttons' are supported and these may be mapped in any way the application chooses. Likewise the sensitivity of this device can be controlled, both in terms of acceleration and threshold, if the Server is fully implemented.

The advantages of this include: device independence; freedom to control operating characteristics of the device; most pointer configurations are supported; the ability to modify the button mapping allows for example, a one-button mouse to emulate a three-button mouse (by use of button modifiers through the keyboard). The disadvantages are that it is less efficient; there is possible confusion (if button mapping is changed); only one device is really supported at any one time unless the application explicitly switches devices (perhaps by use of button modifiers).

Most windowing systems provide a level of abstraction at the window level but not at the screen level. Typically they identify a pre-determined range of device standards (such as CGA, EGA, VGA, and so on in the PC world) and adjust to suit these – they do not operate on the raw characteristics of the screen. In geometrical terms the X Server is responsible for: mapping 'logical' pixels to 'screen' pixels; storing the size of the screen in pixels (width and height); storing the physical size of the screen (inches or millimetres); and, storing the aspect ratio for the pixels associated with the screen. Doing the above for all the 'screens' attached to the 'display' (a 'display' in X can consist of multiple screens as long as they are controlled by the one keyboard/pointer).

More significantly, perhaps, this software emulation extends to colour. Devices have a multitude of ways for representing colour. The X Server is responsible for logically mapping these colours to the hardware capability of colour hardware.

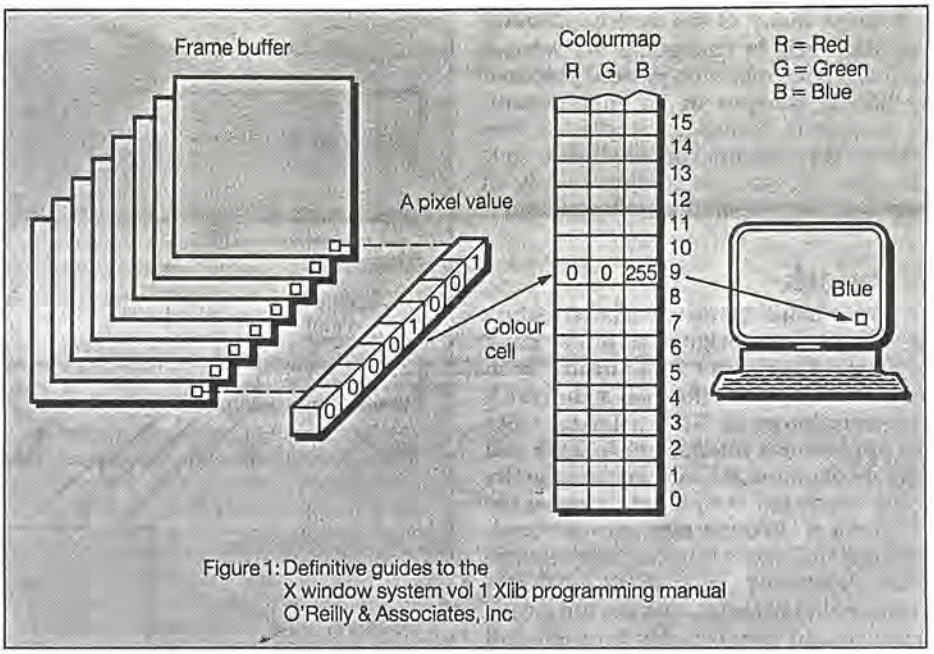

Figure 1: Definitive guides to the X window system vol 1 Xlib programming manual O'Reilly & Associates, Inc

In a bit-mapped display, which is what X is intended to support, the number of colours available on-screen at any one time is dependent on the amount of memory used as the 'frame' buffer for each pixel. A one-bit deep pixel allows the colour to be on or off (monochrome), a four-bit deep pixel allows 16 colours (or shades of grey), eight-bits 256, and so on. In order to display these colours on the screen most devices use what are called 'digital to analogue converters', 'DACs' for short. These convert bit values into intensity levels which in turn are sent to each gun. On many modern workstations each gun is controlled by an eight-bit DAC giving the screen the ability to display 16.7 million colours. Obviously some form of conversion is required between the two. The method used is to employ CLUTs (Colour LookUp Tables) with each value in the frame buffer acting as an index to this table. Each entry in the table has the three entries associated with each DAC and it is these that are sent to the screen. It is often possible to change the contents of these tables, giving the 256 colours out of a palette of 16.7 million often quoted by manufacturers. Figure 1 illustrates this concept graphically.

The point of all this is that X controls colour through software CLUTs. But, instead of using numerical indices, text descriptions (available from a database provided by the Server) are used. For each colour the database also holds values, in the range 0-65535, representing the intensity of each primary gun (red, green and blue). In the latter case the Server will hunt through the CLUT until it finds the closest match the Server can support (it does not fail, however, if it cannot find an exact match).

Since it is possible to view the frame buffer in different ways, X actually provides 'visuals' which hold details of the depth of each pixel and the associated colour map. Each 'screen' may have more than one 'visual'. Software can select colours, reserve colours, change the logical mapping of colours, and, if the hardware permits, change the physical mapping of colours.

The advantages of this approach include true hardware independence, and software that does not need prior knowledge of screen characteristics. Monochrome and colour screens are handled in exactly the same way; colours may be selected with total freedom; an application can reserve colour entries for its own use; and, each application can have its own colour map. However, the disadvantages are that it is a lot less efficient; the software has to find out the device attributes; and the ability to change and reserve colours can become very 'antisocial'. Each application can change the colours for all the other applications running on the same device.

The X Server is also responsible for supporting the usual fonts, lines, circles, arcs, and so on, found in other windowing systems. In addition, the Server provides bitmap (or 'Pixmap') handlers and the ability to manipulate screen 'Images' off screen. It also provides block copy/move facilities which allow manufacturers to make use of 'blitter' or 'raster op' technology rather than the low-level, pixel-based operations. Each primitive in 'X' also goes through what IXI terms the 'graphics pipeline'. This means that lines can be made out of a pattern, for example, rather than being just one colour. This again adds to processing overheads. Because it is intended that no application should have a prior knowledge of device properties, the 'X' Server has numerous facilities for providing information about the device.

As should be fairly obvious by now, X does not implement the graphics as a series of low-level processor interrupts or function calls. The whole of X is written in 'C'. In almost all of the above cases device-independence and software flexibility have led to less efficiency. Many of the options provided by X are not handled well by traditional processors and specialist hardware is usually required to give good performance – particularly on high-resolution colour systems. X also uses a large amount of memory (again, more so in colour). X does not come cheap.

Because many of the device attributes can effectively be changed in software, a well-written application must be prepared to handle changes in its environment. Extra code is needed to handle the case where the colours or keyboard gets changed, for example. X relies heavily on applications co-operating with each other.

Network

The other major hardware-related feature of X is that it is designed to run across a network. The Server has already been discussed fully; the other half of the core X system is known as 'XLib'. XLib provides the applications interface to X. XLib and Server communicate with one another by using 'messages' in a format known as the 'X' protocol'. Because messages are used, XLib and the Server do not have to be on the same proccessor. In practice what this means is that an application can run on one type of machine and display its results on a totally different type of machine across a network. In fact, multiple applications can be running on multiple machines and all the output sent through one X workstation. Hence the use of the term Server above. It is a 'Server' in that multiple users (in this case applications) all share a common output device in much the same way as computer users share a printer 'Server' across a similar network. Of course, the Server and XLib can run on the same machine but the same message passing mechanism is used – hence 'network transparency'. This model of computation is known as the 'Client-Server' model – where a 'Client' usually refers to the application. (In fact I shall use Client from now on to refer to the application).

Graphics are typically processor-intensive and take time. On most windowing systems running on a single machine, the application would have to wait for the graphics to finish. Not so with X. As soon as the Client has sent the message it can start doing other things. X also makes every attempt to optimise the use of the network.

A Client need not hang around waiting for a line to be drawn; it can be doing other things at the same time. Everything to X is an 'Event', a 'Request', or an 'Error' message. In other words the sequence is in the order in which these 'events' *actually* occur not how the logic dictates they *should* occur. In order to work in this way a Client has to be 'Event Driven'. In practice this means that a true X application is written to respond to these 'Events' rather than dictate the flow via other means. An 'Event Gathering Loop' forms the top level of the application and this branches according to the type of 'Event' received.

The advantages of this approach are that the Server and XLib do not have to be in synchronisation – XLib can indeed do other things while waiting for that error message; there is more freedom to the user – the user dictates the flow through the program; and the code produced tends to be significantly more modular. However, there are disadvantages: a different method of programming has to be employed; the programmer has to keep track of things at the same time; it can be difficult to find out which Request caused an Error (fortunately X does provide switches which force synchronisation for use in debugging – but this slows everything down) and events may not come in the desired order (although X does provide the means to 'Peek' through the 'Event Queue' or look ahead, in other words). The 'event' handling is complicated still further by the fact that X usually buffers these messages. Events, Requests and Errors, do not usually get transmitted immediately. Normally these messages are queued up until the message buffer is full, or the network is quiet, or the buffer is 'Flushed' (explicitly or implicitly). The advantages are that the network has less overhead (overhead is usually proportional to the number of messages rather than the size) and neither the Client nor the Server need wait for the network to be ready.

But the disadvantages are that graphics are not drawn as the functions are called – they are drawn after the buffer is flushed which can result in poor user feedback. For example, the user draws a line, it fails to appear on the screen, he/she reacts by drawing the line again – both lines then appear! In the same way 'Events' do not arrive until the buffers are flushed, magnifying the above debugging problems.

As you can imagine a line can have several properties: width, colour, style, length, and so on. Likewise, windows all have their own properties, position, width, height, background colour and background pattern. To pass all of this information backwards and forwards across the network would rapidly saturate it. Instead, the Server maintains the information and, once created, returns a 'Resource Id' to the Client. The Client then uses this 'Id' in all subsequent messages. In the case of graphics this 'Id' is known as the 'Graphic Context' or 'GC' in X.

This significantly reduces network traffic and makes it easier to share 'Resources' or communicate between windows. In addition, the burden of maintaining this information in the client is reduced.

However, the scope for abusing another client's resources is increased. (In practice, well-written X applications are written to co-operate with each other). Certain information retained by the Server is also usually needed by the Client with the Client duplicating this information (the alternative is to increase network traffic by sending 'Round Trip Requests' to the Server).

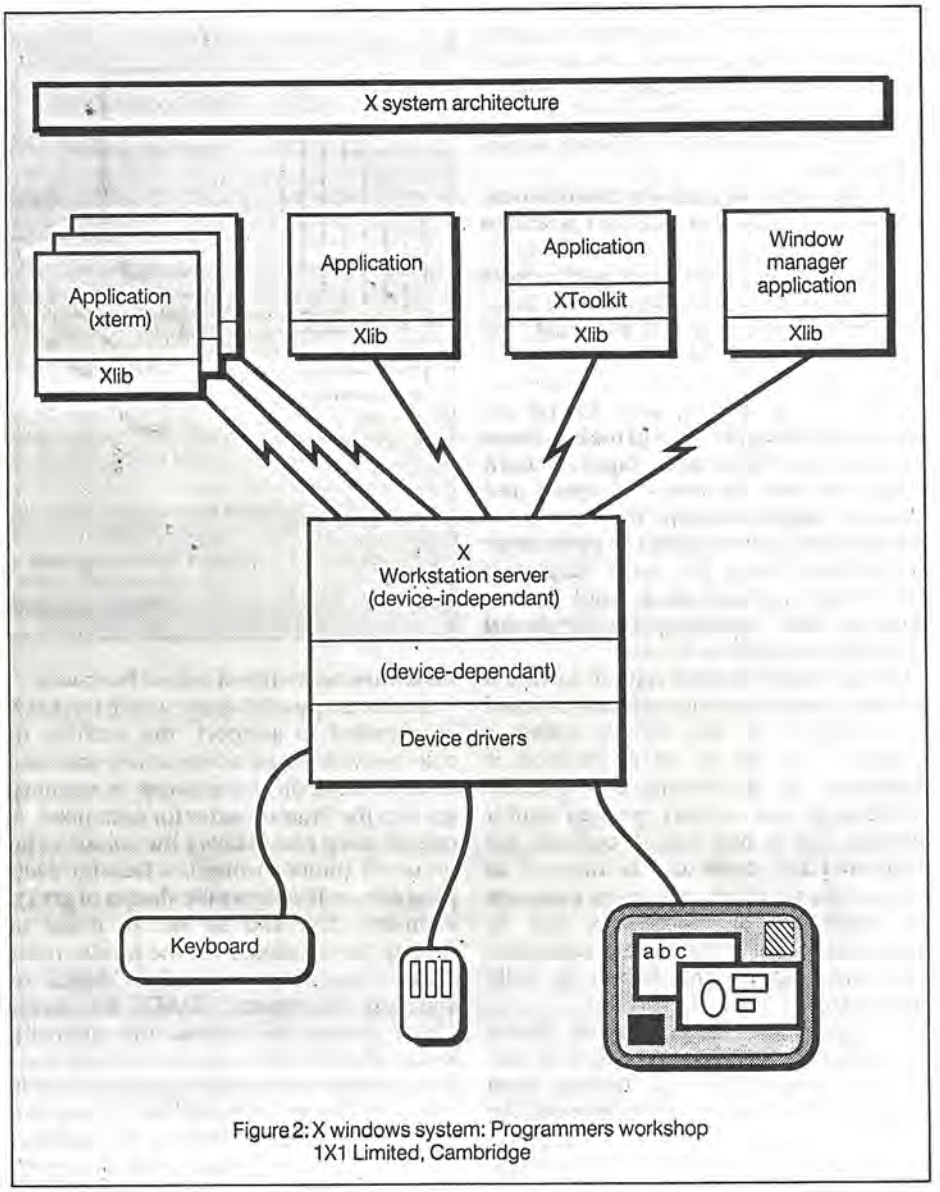

Figure 2: X windows system: Programmers workshop
1X1 Limited, Cambridge

Multiple concurrent applications

It is clear, therefore, that X supports multiple Clients all running at the same time on the same display. In addition, whereas most systems have a limit on the number of windows, X is only limited by the available memory. In fact, windows are so plentiful that they are used to create scroll bars, screen icons, and anything else for which it is useful to get keyboard or mouse input. In fact it is this latter characteristic which really dictates whether a window or some other primitive should be used.

Windows under X operate on the one parent multiple child basis. Child windows are only able to display those parts which lie within its parent's boundaries but apart from that there is relative freedom about the placement of windows (children of the same parent at the same level may overlap each other). The top window, known as the root window is created by X at start-up time.

Windows act as focus points for 'Events'. Events are usually selected on the basis of a window id. There are, however, exceptions in that Clients are able to 'grab' the keyboard and/or mouse for their own dedicated use. Normally Events also propagate up the ancestral tree if they are not dealt with. However, this can be blocked at any level.

The major benefit of the Client-Server model is that resources may be shared between multiple clients. Unlike traditional systems which have to maintain their own copies of resources, only one is maintained by the Server. Figure 2 summarises the overall concept of 'X'.

The advantages with this overall approach are that, once the initial overhead for X is accepted, adding new windows (and therefore new applications) to the display is relatively cheap. Also, because 'Resource Ids', including windows ids, may be queried by any client, it is easy for clients to communicate with each other. And, finally, any client can get any 'Event' simply by requesting it.

For a single application this is very expensive, and as we have seen there is scope for 'anti-social' behaviour. The most common occurrence of this is with colour as there is a danger of 'Thrashing'. Consider two applications, each with their own colour map. As each gets hold of the device each replaces the previous colour map with its own. The net result – a flashing screen! Programmers have to consider other Clients when designing applications.

Policy freedom

The above problems are handled in most multi-tasking or multi-proccessing environments by a 'Window Manager', although for the most part this function is tightly interwoven within the windowing system itself. The window manager dictates the policy about who gets access to a window, how the window is controlled, who can change colours, and so on. For example, the Apple Macintosh policy is that clicking at the top left corner of a window closes the window, double clicking on a menu item selects that item, clicking on a specific window brings that window to the front and only that window has control, and so on. One of the objectives behind X was that it should be 'policy free'. The reason is to allow any manufacturers to superimpose their own policy. And it has to be said that this is another reason why X has been adopted by so many. Apple, IBM, Microsoft, Hewlett-Packard, SUN, DEC, and many others have committed themselves to putting their policies on top of X in one way or another.

The lack of official policy does not mean that X comes without any 'toolkits'. The problem is that, until some standard is adopted, you may well end up having to maintain someone else's tookit in the event that the rest of the market goes in a different direction.

X does, however, provide methods for a Client to tell a window manager what its requirements are. It is up to the window manager as to how much notice it takes of these. This is another area where X differs from other windowing systems. Apart from the lack of a standard 'toolkit' to do many higher level functions (such as create scrolling windows, menus and so on) a programmer also has to consider co-operating with numerous different window managers. In programming terms this is achieved by giving the window manager hints, and by providing software which allows for the manager totally ignoring those hints. For example, a request for a top-level window (a child of the root) should be accompanied by a hint on where the Client would like that window to be placed and on what size it should be. It is quite likely that the window manager will put it somewhere totally different and that the window will be a totally different size. An application program must cater for this.

The lack of official policy does not mean that X comes without any 'toolkits'. Far from it, there are a number shipped with the distribution tape. The problem is that until some standard is adopted you may well end up having to maintain someone else's toolkit in the event that the rest of the market goes in a different direction. However, 'Open Look' should stabilise the market in this area.

The other item still to be finalised is how Clients communicate with each other. X does include an 'InterClient Communications Policy' but this is not officially part of X yet. However, the X documentation does infer that on the whole these conventions will be adopted and that it is reasonably safe to use them.

Extensible

Recognising that computer technology moves quickly, the authors of X have attempted to ensure that X can be extended without affecting compatibility with the core system. Aside from vendor specific extensions, current developments include:

- O extra 2D facilities (better curve handling)
- O 3D extensions (loosely based around the PHIGS standard)
- O display Postscript (the basis for Sun NEWS).
- O moves towards the 'Open Look' policy and standardisation of the official 'Toolkit'.

Despite the bureaucracy associated with what has become an international standard, X is moving with the times. By making it relatively easy to add extensions it should be possible for X to take advantage of new technological developments as they occur – without losing backward compatibility. This again differs from many previous systems which often have to change totally in order to incorporate new developments.

Conclusions

X introduces a slightly different method for a Client to interact with a display device, and obviously the cost is in terms of processor and memory requirements. Fortunately, this cost is largely met by the emerging 32-bit-based machines and X is set to provide a suitable standard on top of which many Clients may be ported. Hopefully, this standardisation should pave the way for machines from many different manufacturers to break into markets which were previously cornered by just one. X should also reduce developments costs and make the workstation market more attractive to more software developers. All of this is good news for the end user since it will reduce the cost of high-performance, high-resolution graphics solutions to problems which have so far been addressed by somewhat limited PC-based products.

Mike Charnley-Fisher may be contacted on (0494) 783594.

Further information

For those of you who wish to find out about X in more detail I would strongly recommend *The Definitive Guides to the X Window System* series published by O'Reilly & Associates, Inc – especially Volume One, the *Xlib Programming Manual*. Other sources of information include:

- O IXI of Cambridge do training courses in X and if you are not very good at reading books, then their course is well worth attending.
- O The *MIT X* manuals – for most purposes the O'Reilly books successfully replace these. However, if you want to use the X toolkits, or do slightly unusual things, they still remain useful.

There are a couple of other books now available covering X. Having browsed through them in the book-shops I have to say that I think the O'Reilly books appear to be the better buy.