X Window Terminals

Björn Engberg and Thomas Porcher

Abstract

X window terminals occupy a niche between X window workstations and graphics terminals. The purpose of terminals in general is to provide low-cost user access to host computers or smaller dedicated systems. X window terminals further the advance in graphics terminals and provide new and interesting ways to utilize host systems. Ethernet cable provides for graphics performance previously not seen in terminals. The X Window System developed by MIT allows multiple applications to be displayed and controlled from the user's workstation. Now, with X window terminals, the same powerful user interface is available on host and other non-workstation computers. [The XTERMINALS paper starts here.]

In mid 1987, the Video, Image, and Print Systems (VIPS) Group began the design of Digital's first X window terminal, the VT1000 terminal and its code upgrade, the VT1200 terminal. Our goal was to design and implement an X window terminal that would allow the use of windowing capabilities on large computer systems. In 1989, Digital developed the VT1300 X terminal and in 1991 the VXT 2000 X terminal. The designs of these X window terminals are all quite different. Our design approach changed as the underlying technology changed.

This paper first compares host-system computing with applications run on workstations. It summarizes the significance of the X Window System developed by MIT and discusses the client-server model. The paper then presents the need for X window terminals and follows their development stages. It compares and contrasts Digital's different design strategies for the VT1000, VT1200, and VT1300 X terminals. The paper concludes with a summary of the recently announced VXT 2000 X terminal.

Background

Before the development of the X Window System, there was very little overlap in functionality between workstations and other kinds of computers. Workstations had stunning and fast graphics, and many powerful applications were available on them. Those applications were not available to users of basic 80-by-24 character-cell text display terminals connected to a host system located in a clean room. Graphics terminals, of course, allowed the use of ReGIS or another protocol for math and business graphics, but their performance was far below the expectations of a workstation user. Few people have the patience to run, for example, a computer-aided design application on a VT240 terminal, assuming such a version of the application is available.

Although a workstation offers fast graphics capabilities, its applications sometimes need more CPU power or more disk space to do calculations in a timely fashion. Graphics applications written for workstations could not run on faster host computers, which did not provide a display. Nor was there a standard way to get data from the host to display on a workstation. Each application required a unique solution to this problem.

Since the introduction of the new client-server model of computing and modern networks, many tasks can be divided into subtasks that can run on the most suitable processor. The X Window System uses the client-server approach, as shown in Figure 1. The application is viewed as an X client, and a workstation or a terminal can run an X server that controls the display. The X server also controls input from the keyboard and mouse or other pointing devices.

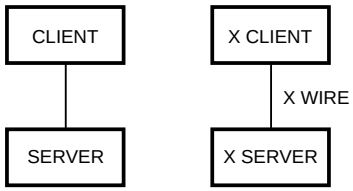

Figure 1 Client-server Model

An X client and an X server use an X wire to communicate, as shown in Figure 2. The X wire is simply a two-way error-free byte stream, which can be implemented in many different ways. The X Window System architecture does not stipulate how the X wire should be implemented, but several de facto standards have emerged. Manufacturers have designed X wires usually based on the data transport mechanisms that were available and convenient when the X Window System was implemented. The X wires use transmission control protocol/internet protocol (TCP/IP), DECnet, Local Area Transport (LAT), and other protocols, and even shared memory buffers as a transport to avoid protocol overhead. A single implementation often supports several transport mechanisms.

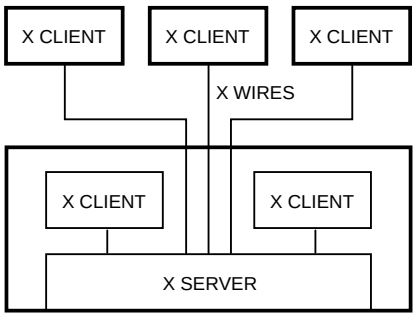

Figure 2 X Wires

The X server typically executes on a processor with display hardware. The X client can execute on almost any processor. It may execute on the same CPU as the X server, or it may execute on a host, another workstation, or a compute server. The X server can be connected to several X clients simultaneously, with any combination of local (running on the same CPU) or remote (running on another CPU) X clients. The X server treats local and remote clients equally.

Workstation Environment

Figure 3 compares a traditional non-X windowing workstation with an X windowing workstation. In both workstations the application must use a graphics library to communicate with the display hardware and software.

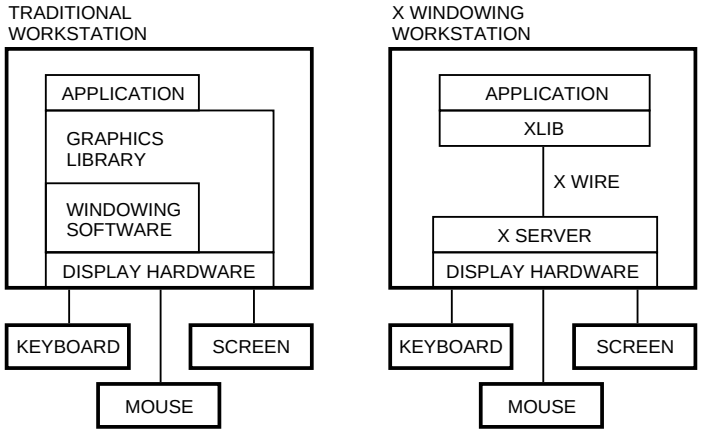

Figure 3 Inside the Workstation

In an X windowing client environment, the library of routines is called Xlib. An application designer can choose from a wide variety of toolkits, which are essentially a level of additional library routines between the application and Xlib. The use of a toolkit can significantly reduce the amount of work an application programmer has to do. The application software, Xlib, optional toolkit, and other libraries compose the X client, as shown in Figure 4.

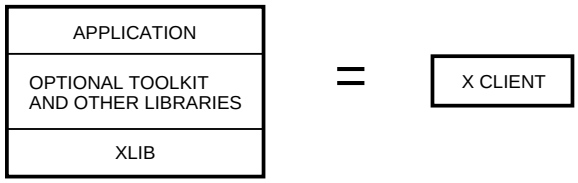

Figure 4 The X Client

With few exceptions, the X server comes with the display hardware and input devices (keyboard and pointer) indicated in Figure 5.

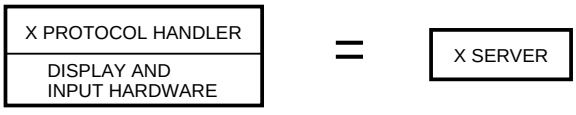

Figure 5 The X Server

The X Window System with its flexibility neatly solves the problems of CPU power and disk space versus display availability. Applications written for X can execute on a wide variety of computers, and the results can be displayed on any of a multitude of devices, even on a workstation that would not have the capacity to run the application locally. Figure 6 shows how the X Window System fits into a network environment.

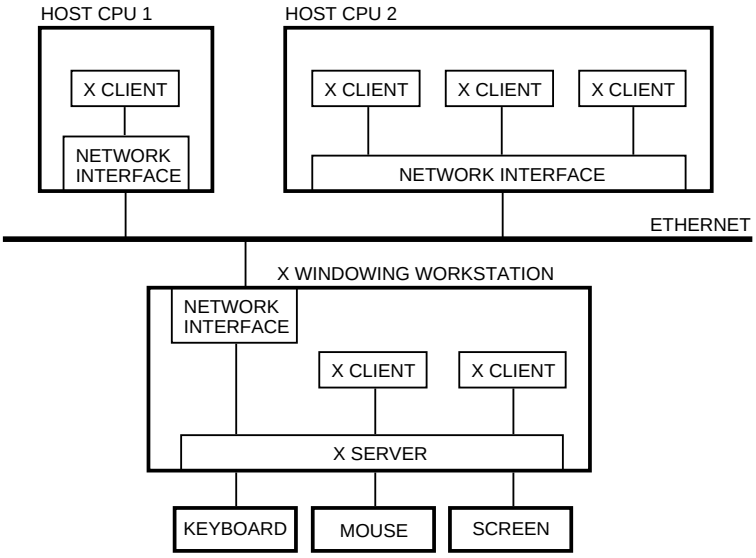

Figure 6 X Window Network Environment

The X Window System has already generated many useful applications, and its widespread popularity ensures that many more applications will be made available in the future.

Need for X Terminals

In a study to determine how workstations are used, the VIPS Group found that many users did not take advantage of the full potential of their workstations. In a software development or document editing environment, the users often set up their workstations as terminals. They usually created a few terminal emulation windows and used SET HOST or RLOGIN commands to connect to a host system on which they stored their working environment and files. Only two features of a workstation were frequently used. Users kept several terminal emulators on their screens at the same time, and set the terminal emulator windows to be larger than 80 by 24 characters. Only rarely did the average workstation user take advantage of the full power of graphics applications.

The results of our study indicated a need for a cost-effective alternative to a workstation that would provide the features desired by a large number of users. We envisioned a new kind of terminal, one that would allow people to have multiple windows of arbitrary size, to connect with multiple hosts, and, since the X architecture allowed it, to be able to use the same kind of graphics as a workstation.

From an X architecture standpoint, X terminals and X workstations are quite similar. They can in fact use the same hardware. For example, Digital's VT1300 terminal runs on the same hardware as the VAXstation 3100 workstation. X terminal software can also be made to run well on hardware platforms that are not suitable for workstations.

The main architectural difference between X terminal and X workstation software is that X terminals are closed systems that do not support local user applications. Although this may seem to be an unnecessary restriction, it does allow X terminals to be made for less money. An open system that allows any user application to run locally must have

an established CPU architecture, a supported operating system, such as the VMS, UNIX, or ULTRIX system, and, subsequently, sufficient memory and /or disk space to support such an environment. A closed system, on the other hand, can be designed with simpler hardware, a smaller operating system, less memory, and thus lower cost. The absence of the ability to run user applications locally does not impact usability significantly since the user can run any desired application on another CPU. Digital's VT1000 and VT1200 X terminals were designed based on this approach.

X Terminal Environment

X terminals often have local applications, but they must be built into the terminal by the designers. The VT1200 terminal has a video terminal emulator (VTE), a window manager, and a terminal manager as the local applications. The VTE allows the VT1200 terminal to make American National Standards Institute (ANSI) character-cell connections to a host, via the Ethernet or the serial lines. This capability makes the VT1200 terminal useful in an environment that does not have X window support.

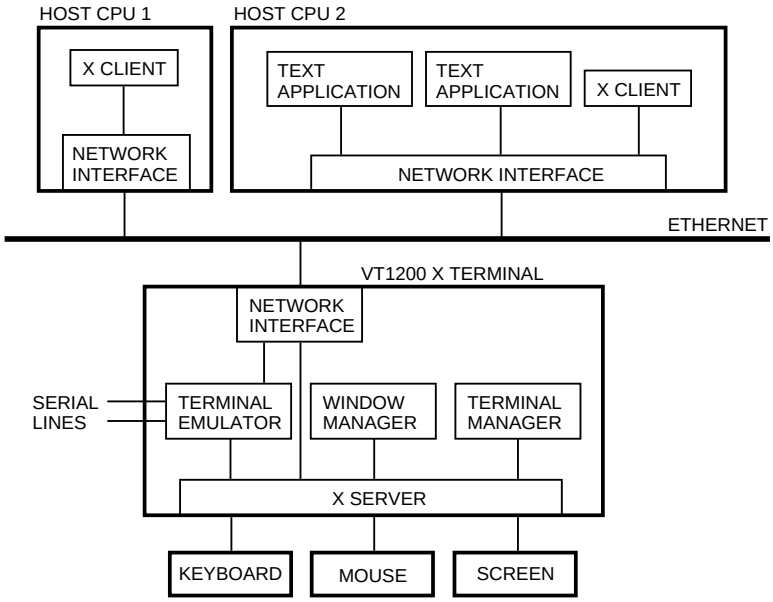

Figure 7 The X Terminal Environment

Although any X server can run windows software, it does not provide a user interface. To manipulate the windows, the user needs a window manager. The window manager creates window frames that allow the user to invoke functions to move windows, resize windows, change stacking order, and use icons. This capability also makes the VT1200 terminal useful when no host is available to run a remote window manager. A terminal with a local window manager generates less network traffic, and window management is not slowed by host congestion or network round-trip delays. The VT1200 X terminal allows use of a remote window manager, if the user prefers a different style of window management.

The local terminal manager provides the user interface to initiate connections to host systems. It is also responsible for the terminal customization interface.

All clients communicate with the X server using standard X wire commands only. Any window manager, remote or local, can manage all the windows on the screen, regardless of whether the clients are remote or local.

Development of X Windor Terminals

The development process of the VT1000 and VT1200 X terminals has important lessons to teach us. The knowledge we gained in 1987 has helped us develop future generations of X terminals.

When we designed the VT1000 X terminal and its code upgrade, the VT1200, we held many discus-

sions within the group and with people from other groups. We planned many iterations before we arrived at the final architecture. It was by no means the only way to design an X terminal, and in 1989 we tried a different approach with the design of the VT1300 terminal. We knew that the best decision at a particular time might be very different from the best decision one year later, since the technical and marketing environment is constantly changing. New tools, standards, and practices enter the field while others become obsolete. Newer products must always have new features to meet changing technology requirements.

Hardware Platform

Our first step was to discuss the hardware platform and select the kind of CPU to use, memory size, I/O considerations, type of display, etc. We studied many different CPUs to determine which one would provide the most capabilities for the lowest cost. A VAX chip was rejected because, at the time, it was far too expensive for the required price range of the VT1000 terminal. The Motorola 68000 series CPUs are quite powerful, but we had to consider other factors such as availability of software and hardware tools, cross compilers and linkers that could run on VMS, and hardware debugging facilities of sufficient power. We finally selected Texas Instruments' TMS34010 microprocessor with video support and several built-in graphics instructions that made it a cost-effective solution. It also came with VMS development tools, a C compiler, an assembler and linker, a single-step, hardware trace buffer with disassembler, and a powerful in-circuit emulator that made it possible to control execution in detail, inspect registers and memory, and set break points and hardware watch points (for example, break when writing value *x* into location *y*).

We further discussed the kind of I/O to use. A sample implementation of the MIT X server on a VAXstation 2000 workstation and a primitive serial line protocol showed, as expected, that serial lines were clearly insufficient to carry the X wire protocol without some compression of the wire protocol itself. We had to build Digital's first X terminal with an Ethernet interface.

We needed to determine if this hardware platform could give us sufficient performance. We made several performance estimates, based on what little we knew then about the X server and other software components. We went through each step in as much detail as we could (before anything was built). We calculated how many instructions were necessary to perform each task in the chain of receiving a command and displaying it on the screen. By knowing the speed of the CPU, we could estimate performance in characters or vectors per second. Our estimates showed that the VT1000 X terminal would not be exceedingly fast, but the performance would most probably be sufficient, definitely faster than a VAXstation 2000 in most cases.

In retrospect, actual performance of the VT1000 terminal and the later software upgrade, the VT1200, was close to our estimates, but it took several passes of code optimization to achieve such performance.

We also discussed alternate hardware designs for performance improvements. One solution proposed two CPUs, the TMS34010 microprocessor to handle the display and a 68000 microprocessor to handle I/O and other tasks. Unfortunately, we found no easy way to balance the workload between the two CPUs. We estimated that the different software components would have the following relative CPU demands:

Interrupts, 5 percent
Communications, 10 percent
Operating system, 5 percent
X server (minus display routines), 60 percent
Display routines, 20 percent

To equalize the load between the CPUs, we would have had to split the X server in two, a solution that was not feasible. Any other split of tasks would cause one CPU to spend most of its time waiting for the other, and the overall performance gain would be minimal. Communication between multiple CPUs is complex and is very difficult to debug. Therefore, we decided that two CPUs were not worth the trouble or the cost. The best way to double performance is to install a single CPU that is twice as fast. At that time, the TMS34020 was already being mentioned as a follow-up microprocessor. Since its software would be compatible with the TMS34010, we decided to keep it in mind for possible use in a future terminal.

Code Selection

The use of read-only memory (ROM)-based code versus downloaded code has been debated for some time. ROM-based code starts up faster and incurs less network traffic at startup time (especially on a site with many X terminals), but is not flexible when software is upgraded. On the other hand, downloaded code can be easily distributed. An entire site can be upgraded with one or a few installations by a system manager as opposed to changing ROMs in a large number of terminals. (With the VT1200 X terminal, customers can change ROM boards.) From the point of view of terminal business, it made sense to use ROM-based code in 1987.

We reasoned that not all sites would have Ethernet, but with ROMs the X terminal would still be useful as a multiwindow terminal emulator. We realized that such concerns would change with time, and on the whole, downloaded code would become the better approach. The only exceptions would be in the home or small office markets where a boot host or an Ethernet might not be available. Subsequent X terminals are being made in both downloaded (for example, in the VT1300 terminal) and ROM versions.

Operating System Selection

Next we considered which operating system to use. We looked at other vendors' operating systems, but found they were either too complex and big or inadequate. One of our coworkers had written a very compact operating system for a VAX system used on another project. We used it in our prototype and then adapted it for the TMS34010 processor. We implemented additional functions to run the rest of the software with minimum changes.

There are many advantages to working with "your own" operating system. It is easy to make changes, to work around tricky problems, and to make special enhancements. But operating system code is difficult to debug. Timing is very critical, and throughout the project, we found strange bugs in code that had initially appeared to be all right to everyone involved. We found bugs under heavy load conditions after a rare sequence of events uncovered little timing windows and race conditions that had not been handled properly. Even with in-circuit emulators, such bugs could take weeks to track down.

In the VT1300 we decided to use the VAXELN operating system. We wanted to avoid the possibility of time wasted on finding and patching holes in the design of a new operating system.

Local Terminal Manager

The VT1000 X terminal is self-starting at power-up, but without a host system, it needs a local user interface. We decided that this interface should resemble a workstation session manager and thus called it the local terminal manager. Although it covers a different set of functions, we wanted the local terminal manager to implement a similar set of objects and operations (the "look and feel" or style) of a workstation session manager. The style of the DECwindows session manager was chosen to make it easier for a user to switch between an X terminal and a DECwindows workstation. We wrote a subset toolkit for all the "customize" screens and ensured that the VTE could use the same subset toolkit for its "customize" screens. As DECwindows has progressed, subsequent X terminals have adapted the new user interface preferences, in this case Motif.

Local Terminal Emulator

We considered a local terminal emulator to be an important component. We knew that X-based terminal emulators could run on the host, but in 1987 hosts with X windowing support were rare. Since we were in the terminal group, a terminal that could not manipulate ordinary text by itself was considered unsellable. We wanted the ability to access both X and non-X hosts and we wanted to support multiple text windows. Therefore we defined the terminal emulator as an X client so that text windows could coexist with X client windows. This feature has proved to be exceptionally popular. A large number of users use nothing but video terminal emulator windows. They are not interested in X windowing graphics, but do want multiple and/or larger text windows on a large screen.

Local Window Manager

We debated whether or not to implement a local window manager. The DECwindows window manager was under development and was constantly changing. The DECwindows window manager contained far too many VMS dependencies to be ported easily. Also the X terminal did not have enough memory to run the DECwindows toolkit locally. We could have ported other window managers, but they lacked the essential characteristics of the DECwindows window manager. For a while we considered letting the local clients have a primitive way to manage their own windows, until a full-featured window manager could be started on a host. Again, this alternative lacked the DECwindows system's qualities. We eventually decided to write a window manager based only on Xlib and our subset toolkit calls. It has the essential characteristics of the DECwindows product. Also, since the DECwindows window manager of necessity would keep changing, we wrote the local window manager in such a way that it could relinquish control to a remote window manager. This solution gave us the most flexibility for this hardware platform. The recently announced VXT 2000 X terminal has been designed with virtual memory to accommodate a well-established unmodified window manager, the Motif Window Manager.

X Server

We also needed to choose an X server. We could have based our code on the distribution tape from MIT, but at the time the X Window System was not yet a mature product. Every implementor had to spend considerable time stabilizing the implementation enough to yield a product and improve performance. Since the VMS DECwindows Group had been writing code for the server, we decided to use DECwindows code. Once the porting effort started, we found that most of the performance had been improved by VAX MACRO code. Consequently, we had to re-engineer all the modules or adapt new ones from the MIT tape. As we kept porting and enhancing performance, our code changed more and more until it became extremely difficult to track bug fixes made by the DECwindows Group. The MIT patches were also nearly impossible to use because of code changes and because our starting code was one step removed from the tape.

Today the MIT X server is a mature product; patches and bug fixes are readily available from MIT and the X community. In our current X terminals, the high degree of portability of the MIT X server allows us to keep most of the MIT X server source code almost unchanged so patches are easily applied.

Communications Protocol

Many communications protocols were available, but our choice was dictated by market pressures rather than technical reasons. The market demanded TCP/IP. DECnet would have been acceptable, but it was running out of available addresses, at least within Digital. DECnet address space supports only 64,000 nodes and requires manual address and name assignments. After waiting weeks to get addresses for a few workstations, we realized that adding thousands of X terminals into Digital's internal network would not be possible. DECnet Phase V software has solved this problem.

Next we looked at the LAT protocol used by Digital terminal servers and found that it had several advantages. First, the VMS operating system supports the LAT protocol. LAT uses unique 48-bit Ethernet addresses to identify each node, which allows a large node address space. LAT also does not require any system management to add another terminal. A user can connect a terminal to a power source, and the terminal automatically becomes part of the network. Our performance evaluations found that the LAT interface on the host could be written to incur less host overhead than DECnet, which is important when many X terminals are connected to hosts.

Changes were needed in the VMS LAT driver to accommodate X wire and font service connections. The VMS Software Engineering Group worked with us to ensure that we would have those changes on schedule and in the appropriate VMS releases. As a result, we chose the LAT protocol for the VMS community and TCP/IP for users of ULTRIX and UNIX systems.

Font File System

Storing fonts and changing font file formats were major problems. Since the VT1000 X terminal did not have a local file system, some fonts had to be stored in ROM to allow the VT1000 terminal to function in standalone mode. A quick review of the available DECwindows fonts showed that not all of them fit in the ROM space allowed for the terminal. Furthermore, customer-designed fonts or new font releases could not be accommodated. The solution was to be able to read fonts from a host system. This approach provided a font service on VMS, and enabled font files to be read over the Internet. We designed a process called the font daemon to run on the VMS operating system. This process could deliver font data on request to one or several VT1000 terminals. The VMS font daemon uses the LAT protocol to deliver the fonts and protects somewhat against font file format changes. In many ways, the design of the font daemon makes it a precursor to a general font server, and it is very similar to the X Font Server being delivered by MIT in the latest release of the X Windows System.

To use the font service, the terminal user must specify a font path in the VT1200 local terminal manager. Specifying a host name is sufficient to access the default font path, although users with their own font files can optionally search other directories. At startup, the VT1200 terminal makes a font connection to the host's font service and delivers the font path specification to the font service. The font service sends font names and other basic font information about all the fonts in the selected path. When the VT1200 X server needs a font, the VT1200 first searches the ROM-based fonts; if it is not there, a request to read the font is sent to the font daemon. The daemon sends the required information to the VT1200, and the X server can display characters from that font. Since memory is limited, the VT1200 has font caching, a mechanism to discard fonts no longer used or to discard the least used fonts. Our current X terminals increase the robustness of the font mechanism; for example, they provide recovery should the font service or its host become unavailable.

The special LAT code that we used on VMS systems for the font service was not available on UNIX and ULTRIX operating systems. Since internet protocol (IP) was available, we could use the trivial file transfer protocol (TFTP) to read a file from a host system, if the system manager set the proper protections. We chose TFTP for its ease of implementation and its wide availability on UNIX and ULTRIX systems. The TFTP font path in a VT1200 terminal specifies a host IP address and a complete path to a file (usually named font.paths) that contains the complete path to all the font files that the VT1200 can use. The terminal can then access all those font files, again through TFTP, to obtain font names and other basic information about each font. When a client wishes to use a font, the proper font file can be read again, this time to load the complete font. Since this process is time-consuming, the font path pointing to the file has an alternate format in which the font name follows the complete path to each file. Using this alternate format, the VT1200 terminal does not have to open and read the font file until a client actually intends to use it.

Comparison of X Terminals

The VT1200 and VT1300 X window terminals were built using different approaches to solve the problems encountered during development. The X terminal is a new and flexible concept; there is no single "best" design. Table 1 compares the most important differences between the two terminals. We also include the specifics for the VXT 2000 X terminal.

Table 1
Comparison of X Window Terminals

VT1200	VT1300	VXT 2000
Monochrome only	Color only	Monochrome and color
1 bit plane	4 or 8 bit planes	1 or 8 bit planes
Code in ROM	Code downloaded	Code downloaded
No virtual memory	No virtual memory	Virtual memory
2 to 4MB RAM	8 to 32MB RAM	4 to 16MB RAM
TMS34010 CPU	VAX CPU	VAX CPU
Special operating system	VAXELN operating system	Special operating system
Local clients: Terminal manager Window manager Video terminal emulator	No local clients	Local clients: Terminal manager Motif window manager DECterm terminal emulator
Local customization	Customized on host just as a workstation	Local customization Centralized customization
Choice of host (LAT only)	Automatic X window login to boot host	Choice of host(LAT and TCP/IP using XDMCP)
LAT protocol	DECnet protocol	LAT protocol
TCP/IP protocol	TCP/IP protocol	TCP/IP protocol
Special hardware	Available on several workstation platforms	Uses standard hardware

The VT1200 is ROM-based; all its software is permanently resident in the terminal. The VT1300 software is downloaded, so a host or bootserver on the same network must supply the terminal with a load image at power-up.

Since downloaded terminals are dependent on the existence of at least one working host system, the user interface can be designed differently. While the VT1200 X terminal has a built-in user interface, the VT1300 does not need it. The VT1300 terminal automatically makes an X connection to a host at power-up, and the user is presented with the same DECwindows login box as on a workstation. The VT1300 has no local clients; all clients run on the host system.

The VT1200 terminal uses the LAT protocol for its ease of use and minimal network management demands. The VT1300 terminal uses the DECnet software already implemented in the VAXELN operating system used internally. Both terminals support

TCP/IP.

VXT 2000 X Terminal

One problem that has plagued all X terminals is limited memory space. Workstations usually have a virtual memory system, which provides large paging and swap areas on a disk, and applications and X servers can use more memory space than the hardware has. Until now X terminals have not had virtual memory systems. If too many applications made excessive demands, or if a client created large off-screen images (called "pixmaps" in the X Window System) the terminals quickly used all memory space. If the X server implementation was correct, an error was reported and a client might try a less demanding approach. In other cases, the terminal or client might simply crash. One alternative was to install more memory in the X terminal, although this can be costly and offers no guarantees.

In the next generation of Digital's X terminals, the VXT 2000, this problem has found a cost-effective solution. Based on the VAX architecture, the VXT 2000 terminal uses virtual memory and downloaded code. The Digital InfoServer, an Ethernet storage server, provides the load image, virtual memory paging space, fonts, and customization storage. The same InfoServer also solves another problem: now the X terminal has access to a file system. This allows more extensive customization, as well as centralized management of the customization of all X terminals on the network. Figure 8 shows the configuration for the VXT 2000 X terminal.

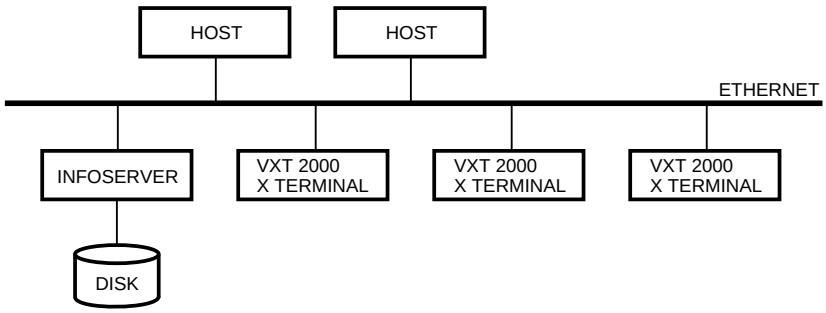

Figure 8 The VTX 2000 Network Environment

Conclusion

X terminals are not intended to replace workstations. Nor will workstations replace host systems or completely displace X terminals in the foreseeable future. It is likely that host computers will always be faster and have more memory and disk space than reasonably priced workstations of the same era. It is also likely that terminals can be built cheaper than workstations of reasonable performance for some time to come. As long as that is the case, there will be a market for X terminals and host systems. Future X terminals will be faster, and have more built-in functionality, more local applications, X extensions, and most likely, additional hardware features. X terminals will be the networked terminals of the 1990s.

Acknowledgements

We wish to thank the members of the VT1200 development team who worked many long hours on this project. Thanks to everyone inside and outside the Video, Image and Print Systems Group who contributed helpful suggestions, constructive criticism, and important hours using and testing the products. Thanks to the LAT and VMS Software Engineering Groups for incorporating the changes needed for the VT1200 X terminal to be useful. Thanks to the VIPS Quality Group for ensuring that as few bugs as possible remained in the product when shipped.

Björn Engberg As a principal software engineer in the Video, Image and Print Systems Group, Björn Engberg was the main architect and software project leader for the VT1000 and VT1200 X window terminals. He joined Digital in 1978 and worked as a development engineer at CSS in Sweden, where he modified Digital's terminals for the European market. He relocated to the United States in 1982 to work on the VT240, the VT320, the LJ250, and sev-

eral advanced development projects. Björn received an M.S.E.E. (honors) from the Royal Institute of Technology in Stockholm.

Thomas C. Porcher Principal engineer Tom Porcher is a member of the Video, Image and Print Systems Group. He provided technical leadership in the development of the VXT 2000 X terminal. Previously he was a technical leader for the VT240 terminal, VAX Session Support Utility, and the DECterm terminal emulator. Tom holds five patents for work on the VT240 terminal and on the multisession protocol used in the VT340 and VT400 series terminals. Tom received his B.S. in mathematics from Stevens Institute of Technology (1975). He is a member of the ACM.