Graphical Interfaces for Network Operations and Management

James P. Cunningham
C. Douglas Blewett
J. Scott Anderson

This paper describes the advantages to the network manager of using graphical user interface systems, and ways to alleviate the software development challenges they present. We discuss some of the graphical techniques and tools that allow developers to efficiently summarize, manipulate, and navigate complex network data. The paper encourages development organizations to consider an iterative refinement development style for graphics applications. Applications built using this style are constructed from small, reusable processes that can be incrementally specified by members of the application team.

Introduction

Operating and managing a large modern communication network is a complex data-intensive task. This includes identifying and solving problems via network monitoring and control, rearranging and adding network resources, tracking and allocating the costs associated with providing communications resources, and planning for the network's evolution.[1] A network can generate thousands of pieces of information per hour that must be presented in a form that lets network managers perform their jobs efficiently.

Increasingly, computer systems used for network operations and management have graphical user interfaces (GUIs) that let managers manipulate menus, windows, and other graphic objects, rather than typing commands. In a typical GUI, information is presented in multiple windows on the display screen, and graphical presentations, such as diagrams, pictures, or maps, are common. A pointing device, such as a mouse, performs many functions, including selecting from menus and moving objects on the display.

The following are examples of AT&T GUI-based network management products, services, or systems:

- The Network Operations Center for AT&T's Worldwide Intelligent Network. This center has graphical workstations that access various monitoring and control systems. Its video wall displays network information on 75 rear-projection screens.
- PERCEVAL. This application verifies the reliability and quality of the AT&T network by providing convenient graphical access to network data and algorithms.
- Accumaster® Services Workstation. Large business customers use this workstation to monitor and control various aspects of services, such as AT&T 800 and AT&T Software Defined Network Service.
- Definity® Manager IV. This system gives PBX customers a multi-window GUI for administrative functions.
- TRANSVU II. This operations system lets telephone companies centrally support transmission network administration, operations, and management. The system's workstation provides a multi-window GUI.

This paper describes the advantages to the network manager of using GUI systems, and ways to alleviate the software development challenges they present.

Value of GUIs for Network Management

Graphical user interfaces are becoming common parts of network management systems because a GUI's style of interaction is particularly appropriate for the tasks performed by network managers. Properly designed, a GUI's flexible approach to the organization of tasks through multiple windows and graphics can combine to make a network management system easy to use.

Windows. A graphical user interface allows multiple windows of information to be

Panel 1. Acronyms and Terms Used in This Paper

Birdseye filter	Graphic technique that permits a user to highlight in one screen that portion of a display to be shown in more detail in another screen.
GUI	Graphical user interface
IPC	Interprocess communication
Parent/child	Defines the relationship of widgets to each other.
Widget	An interactive object, such as a button, icon, or text.
`xtent`	A specification language, developed at AT&T Bell Labatories, that allows quick creation of X Toolkit widget applications.
X Window System	A graphic system developed at MIT.

simultaneously displayed on the screen. Figure 1 illustrates a typical screen display.

A network manager can display multiple applications on a single screen, with each application in its own window or set of windows. This allows a network manager to switch easily between tasks, and to share information between them. For example, a network manager can monitor network status in one window while tracking progress on resolving trouble tickets in another window.

Multiple windows also allow a user to simultaneously access data, from several individual applications, through a window that integrates information from those applications. This concept is used in the AT&T Accumaster Services Workstation, which provides access to applications, such as Routing Manager, Customer Traffic Manager, and On-Line Call Detail Manager. Each application has its own window or set of windows for user interaction. In addition, a "status" window alerts the user to any important events that occur in any application, providing a central place to look for important new information while the manager does other tasks. The window at the top of the screen in Figure 1 is an example of a window that—by integrating information from several applications—provides a summary of current alarms, tests, trouble tickets, and electronic mail.

Windows can offer access to an almost limitless amount of display space on a physical screen display of limited size. An application can be designed to display text or graphics on a screen space of any size, without the designer needing to know the physical dimensions of the screen or window that will be used, and without regard to the amount of data to be displayed. A window can be used to display just a portion of the data, and the user can change the displayed portion by, for example, manipulating scroll bars or zooming in and out.

Another advantage of windows is that an application can have a main window and multiple secondary (i.e., pop-up) windows that can display information when and where it is needed, and then be removed from the screen. This allows new information to be added to a display without losing the context of the currently-displayed information. For example, a network manager can request on-line help about a particular screen object, and the system can be triggered by the help message to open a new, secondary "help" window. The manager can move this help window to any portion of the screen, consult it as needed, then remove it after it is no longer needed. Or a secondary pop-up window can be used to warn the user against a potentially destructive action, then let the user confirm the choice. Popping up the new window draws the user's attention to the warning, and, if necessary, the system can prevent other actions until the user responds to the warning.

Graphics. In a GUI, information is presented whenever possible by using diagrams, pictures, or maps, and the user can work directly with graphical representations of objects, such as scroll bars, files, printers, or network elements. When a user manipulates an object on the display, a corresponding action takes place in the system. For example, dragging a picture (or "icon") of a document to a picture of a printer can direct the system to print a copy of the open file.

Graphics allow a user interface to employ representations and interaction techniques with which a network manager should already be familiar. The color of elements on a network map can be used to represent a status—e.g., green for normal, yellow for a warning, and red for an alarm. As shown in Figure 2, symbols or icons representing functions can be made to resemble their real-world analogs—e.g., a mailbox representing "mail," a trash can representing "delete," and a printer representing "print."

The network manager can use a mouse to manipulate on-screen controls, much as the manager's hand would manipulate real-world controls. For example,

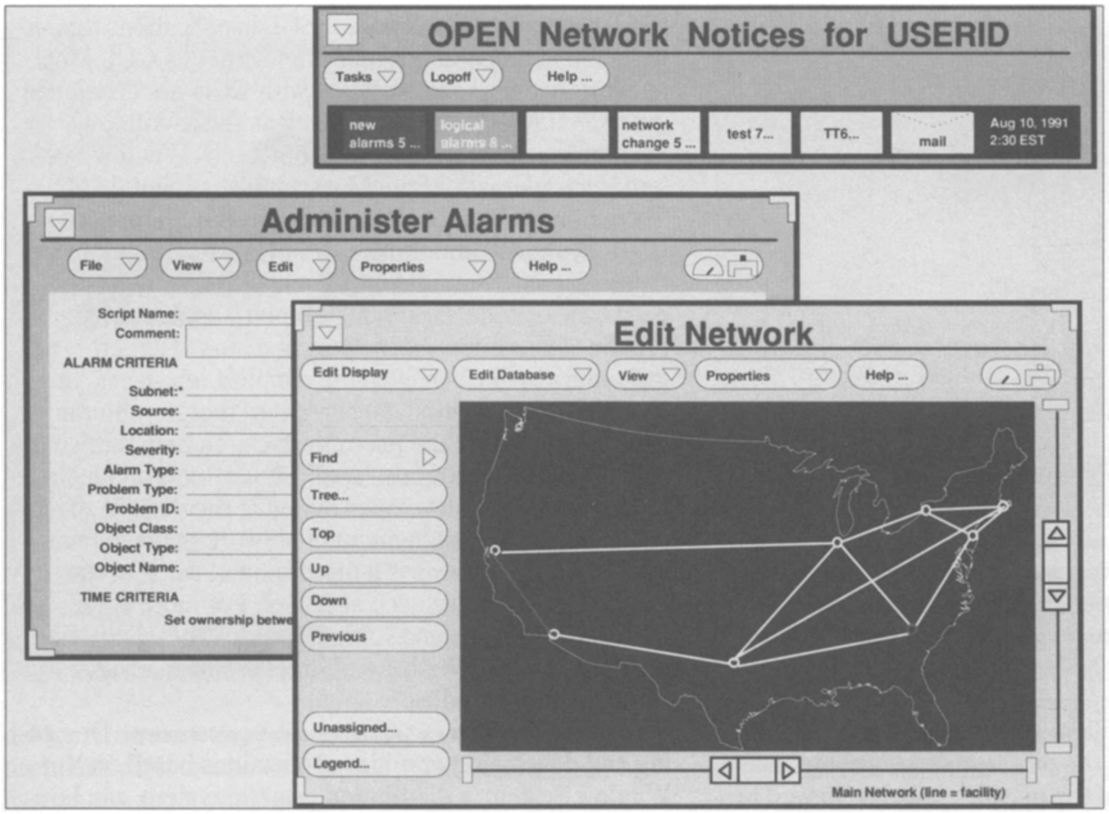


Figure 1. The windowing capability of a graphical user interface (GUI) permits the user to display multiple applications on a screen. The user can easily move from one task to another, and share information among tasks.

values for an alarm threshold can be changed by using an on-screen slider to adjust the scale.

Graphics also can be used to efficiently summarize complex network data and highlight the most important information, thereby helping network managers examine and manipulate large amounts of complex information. For example, a single symbol can be used to simultaneously represent the location, identity, and status of a network element.[2] On the right in Figure 3 is a table with a textual representation of information describing a network fragment: network element type, identifier, location, connectivity, status. On the left of the figure is a graphic summary of the information. In many applications, a network manager can see and use the information in the graphic representation more quickly than the information in the textual representation. The advantage of graphic representations multiplies as the size and complexity of the network increases beyond the small fragment shown in the figure.

A GUI can make multiple levels of detail available to the network manager. For example, an initial display can present the toplevel summary of network information, such as a geographic map showing locations and their interconnectivity. The network manager can select a location to display the next level of information, such as the equipment configuration at the selected location. The manager can then select a piece of equipment and display detailed information about it. The ability to summarize network information, then let the user unfold additional levels of detail as needed, helps a network manager quickly and easily focus on the precise information needed in varying situations.

A "birdseye filter" shown in Figure 4 is another technique to display a large network containing detailed information. With this filter, the main display shows a detailed representation of a small part of a network, and a smaller (birdseye) display simultaneously shows a less-detailed overview of the entire network. The birdseye display shows a dotted box around the area that is visible in the main display. The network manager can use a pointing device to drag the dotted box to a new location in the birdseye window, thereby changing the main

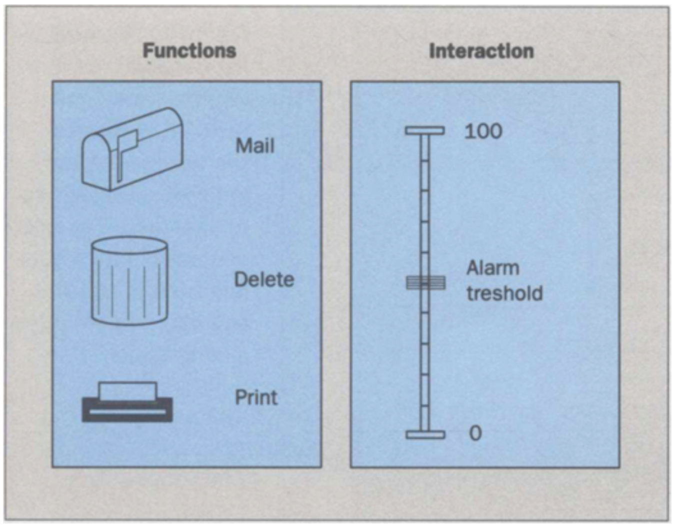


Figure 2. Graphics lets users scan a screen quickly to determine the status of a system, the location of specific functions, or to use "mechanical" aids, such as scroll bars, to define parameters.

display view. Birdseye filters are especially useful for large networks because the birdseye display allows users to easily see what part of the network is being viewed in the main display, and the filter provides a convenient way to instantly navigate different parts of the network in the main display without losing orientation.

Flexible Task Orientation. GUI systems give the user flexibility in sequencing when actions will be performed. Traditional character- or text-based interfaces were designed as a series of "screens" a user would go through in sequence. However, network managers perform *tasks*, for which no single sequence of screens will be best for all users at all times. A GUI can be designed so users can structure their activities to best suit their current tasks. Multiple windows can be simultaneously displayed, users can switch between them, and information can be shared among windows. With a flexible design, the network manager can focus on getting the job done efficiently, rather than on navigating through a rigidly predefined sequence of steps.

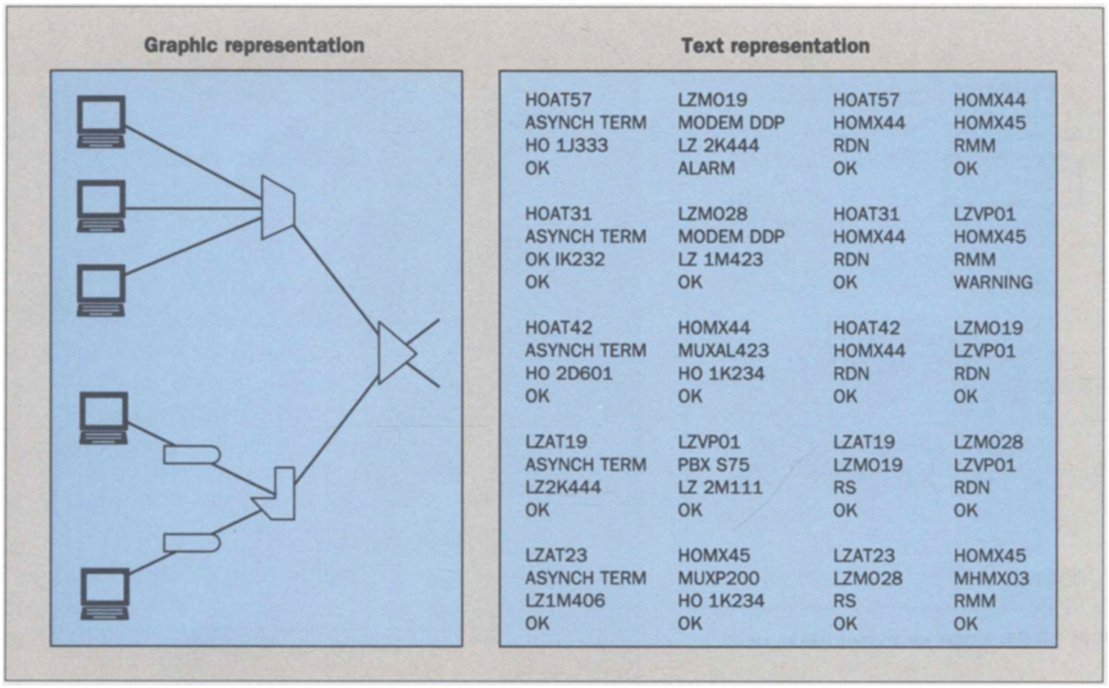
The Software Development Challenge

While GUIs are particularly appropriate for network management systems, they are costly to develop. Traditional software development techniques make it harder to write the software for GUI applications than to write for a comparable application without a GUI. Most network management systems with GUIs are developed using technology standards, such as the X Window System™[3], Open Look®[4], or Motif™[5]. (X Window System is a trademark of the Massachusetts Institute of Technology. Open Look is a registered trademark of UNIX Systems Laboratories. Motif is a trademark of the Open Software Foundation.) Even with this huge standard base of code, producing graphics applications remains labor-intensive, requiring teams of expert programmers to develop even the simplest interfaces. In a typical GUI application, 50 percent of the programming effort is devoted to the user interface. To help reduce the effort in building graphics applications, tools and techniques have been developed aimed at the design, specification, and rapid deployment of network management systems. What follows is a discussion of some of the tools and techniques used at AT&T. For more details see Blewett, Anderson, and Kilduff's work,[6] their USENIX presentation with Wish,[7] and Rotella, Bowman, and Wittman's GUI studies.[8]

Current State of Applications Development. Designing and developing graphics applications based on X Window System, a distributed graphic system, can be a costly process, where most of the effort goes into developing and maintaining the code. Typically, a software project begins with teams of systems and human factors engineers consulting with the end users to generate system requirements and specifications. Next, the software developers take over to build the application. Human factors engineers contribute to the GUI design, but the final responsibility for the working system usually is the programmer's.

The system development process often involves several iterations of specification and software changes before the finished product is delivered. These applications tend to be quite large, with process sizes ranging from five to 10 megabytes or more. Most systems with GUIs are difficult to develop on schedule, are maintenance nightmares, and have user interface designs that are far from optimal.

Rapid Prototyping. Rapid prototyping is seen by some as a "silver bullet" to fix many of the ills mentioned above. The idea is to quickly generate a working prototype that can demonstrate, in a limited way, the proposed application's intended capabilities and user interface



HOAT57	LZM019	HOAT57	HOMX44
ASYNCH TERM	MODEM DDP	HOMX44	HOMX45
HO 1J333	LZ 2K444	RDN	RMM
OK	ALARM	OK	OK
HOAT31	LZM028	HOAT31	LZVP01
ASYNCH TERM	MODEM DDP	HOMX44	HOMX45
OK IK232	LZ 1M423	RDN	RMM
OK	OK	OK	WARNING
HOAT42	HOMX44	HOAT42	LZM019
ASYNCH TERM	MUXAL423	HOMX44	LZVP01
HO 2D601	HO 1K234	RDN	RDN
OK	OK	OK	OK
LZAT19	LZVP01	LZAT19	LZM028
ASYNCH TERM	PBX S75	LZM019	LZVP01
LZ2K444	LZ 2M111	RS	RDN
OK	OK	OK	OK
LZAT23	HOMX45	LZAT23	HOMX45
ASYNCH TERM	MUXP200	LZM028	MHMX03
LZ1M406	HO 1K234	RS	RMM
OK	OK	OK	OK

Figure 3. Graphic elements make network management easier to troubleshoot. On the left is a high-level overview schematic of a portion of a network, which can use color codes to define the status of elements. On the right is a textual description of the same network, providing the user more data on the elements.

design. However, experience suggests that a prototype is useful only if it can be developed quickly and if it can be used in the production-quality product.

Several factors affect prototyping:

- Development organizations do not have the resources to develop two of *anything*. When management sees something working, they will want the product delivered. This can make the resource situation worse for developers who try prototyping.
- Developers hate to throw out anything that works and has taken a significant amount of time to produce. Either the developers do a poor job of prototyping, or they do not want to throw out the prototype, and write the application using production-quality techniques.
- Prototype development often has been the sole domain of expert programmers. This means that the prototype often is simply a scaled-down version of the application, but still has most of the problems associated with the development of the "real thing."

Prototyping tools emerging in the marketplace allow a non-programmer to design the GUI of an application, and also generate the code. These tools provide a rapid, easy to use, WYSIWYG (What You See Is What You Get) interface for developing prototypes and applications. Unfortunately, most also encourage using monolithic architectures that can't be easily decomposed, thus, the code is hard to maintain and update.

On the positive side, a "software tools" approach represents a different approach to prototyping and developing graphics applications. With it, applications are produced from collections of reusable processes that tend to be small and modular. Applications can be built using short time schedules, usually under two months. The short schedule keeps developers from investing large amounts of resources, and from getting absorbed in optimization and complexity. The resulting code is production quality, can be used in the real product, and can be reused in future products.

The `xtent` Specification Language. A specification language called `xtent` has been developed at AT&T Bell Laboratories to facilitate the quick creation of X Toolkit (Xt) widget-based applications.[9] In the X Window System environment, a widget is an interactive graphic object, such as a button, icon, or text. An interprocess communication (IPC) package has been developed to allow communication between processes in our applications. And another software tool, an application builder, has been developed to simplify the creation and maintenance of applications. These tools and techniques have been used by both programmers and non-programmers in AT&T

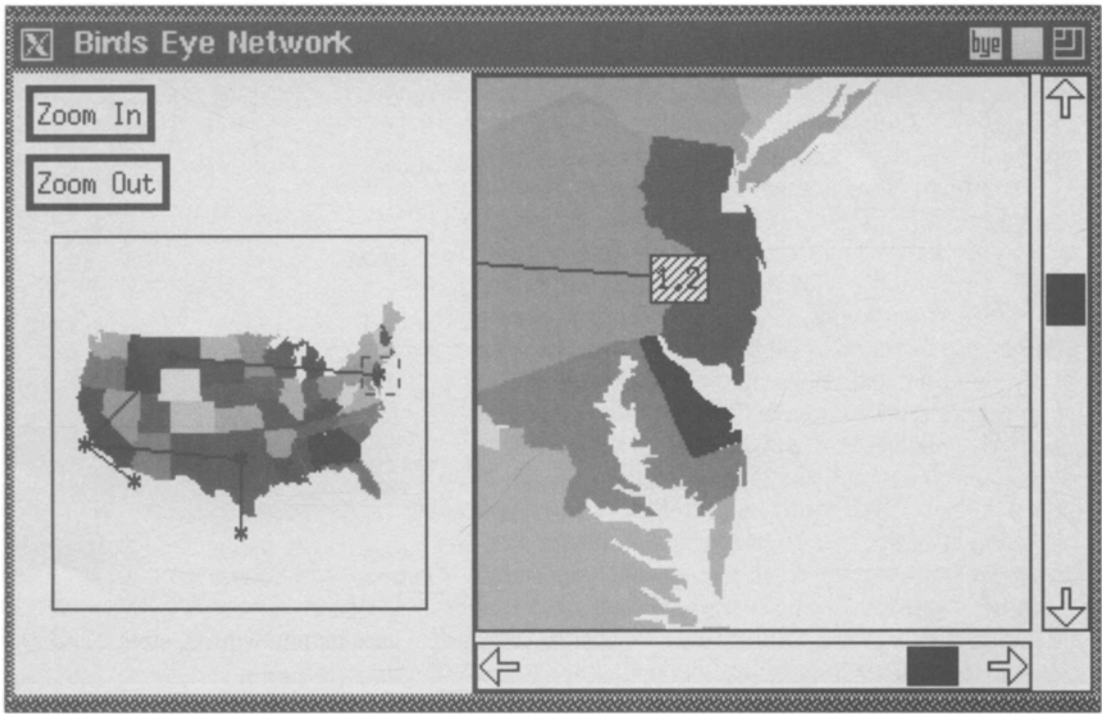


Figure 4. Graphic representations of a network can employ a "birdseye filter" to provide additional detail on any portion of the network. The map on the left of the screen provides an overview (birdseye view) of the network, and the dotted box can be moved from location to location to provide a closeup view of the network in the screen on the right. The "Zoom In-Zoom Out" capability can provide even more detail.

research and development groups. The range of applications produced using the system has been quite large, in both size and scope. The `xtent` language is fully Xt-compatible, so any program that can be written using Xt and widget libraries can be specified using `xtent`. It is a macro interpreter based on the X resource file format. Resource values in the X resource database, and widget resource values can be manipulated as strings.

The following line shows how a value is assigned to a resource variable:

```
*font: 6x13B
```

Setting parameters in this manner is inherently declarative, in that this line will result in the X font `6x13B` being used wherever a font is required by the application.

The X resource syntax has a simple object-oriented style that can be extended for programming. Consider the following line:

```
xtent.allowShellResize: True
```

This line allows the toplevel window (i.e., the shell widget) to be resized when the application requires it. The sense of line above is to set a variable, `allowShellResize`, associated with the application `xtent`, to `True`. This implies that the string `True` will be converted to the appropriate type.

In object-oriented terms, one could also say that the `allowShellResize` procedure associated with the object `xtent` is called with or sent the message `True`. This object/message notion also can be written in a style more closely matching that of C++:

```
object.procedure: message (or argument)
```

Using this syntax and semantics, all the procedures required to do Xt-level specifications have been added. `XtCreateManagedWidget()`, one of the functions used to create widgets, might be used in applications as follows:

```
xtent.Hello.XtCreateManagedWidget: Button
```

The line causes a widget with the name "Hello" to be created. The parent widget is the toplevel widget in the widget hierarchy for the `xtent` application. The widget class is `Button`. The `C` language interface to the X toolkit uses the following function call to create the widget:

```
Hello = XtCreateManagedWidget (''Hello'',
        ButtonWidgetClass,
        parent,
        args, arg_count);
```

The C-based `XtCreateManagedWidget()` interface has four additional parameters. Using the `xtent` format, the name is given on the left hand, or object, side of the expression. Because widget names are hierarchical, the parent name is included in the widget name, so it does not have to be provided. In the C language version, the `args` array is for setting widget resources. The standard X widget resource mechanism handles that. Note that the widgets in Xt applications form a tree-like data structure. The toplevel, parent widgets, also called shell widgets, have child widgets, which may also have children. A pushbutton widget, within a form widget, within the toplevel `xtent` widget, might be specified by the following string:

```
xtent.form.button.
```

The one-line `xtent` entry above is enough to specify a complete (one widget) Xt program. The same program written directly in C, using the X toolkit, takes more than a page of code.[10] Using X alone takes five or six pages. An application in which the widget has been created by the "Hello" line above, runs at the same speed as applications written using the X toolkit or directly in X. This is because Xt programs run from a data structure, which is similar to a display list.

Because creating widgets with `xtent` specifications is much easier than writing C code, non-programmers can easily build their own user interfaces. Also, because `xtent` is an interpreter, there is no compilation or link step. This means that changing widget attributes, or even redesigning the screen, is relatively painless. The ASCII `xtent` specifications also can be used, without compilation or modification, on all machines running the X Window System.

`Xtent` as an IPC Protocol. The `xtent` scheme for connecting graphics processes involves a client/server model similar to the model upon which X is based. The reusable graphics processes are clients of a server that coordinates their activity. A small IPC library has been constructed, and messages are transported between processes as arbitrary-length, asynchronous datagrams.

Several different protocols that sit above the basic message-passing software level have been tried, but the scheme that seems to work best is to send `xtent` code. The advantages of `xtent` code over fixed-protocol techniques are that it:

- Requires no architecture-specific mechanisms to interpret the protocol. Most other schemes require what is called byte-order hacking techniques to handle message protocol conversions to multiple architectures.
- Is Xt-complete. That is, when `xtent` is used as the protocol converter, anything that can be specified in an application by script also can be specified remotely, via messages.
- Allows complete functions to be downloaded into an application, thus reducing the interaction required between system elements.

Network Widget. Because many applications involve network data or operations on networks, a common element of a network management GUI is a graphic network display (Figures 1 and 3). The *network widget* has been developed to support network displays. Like a pushbutton or textual widget, this is a discrete object, and communicates through its resources.

The data to be visualized is represented as nodes and links. A node can be a graphic object (e.g., a pixmap) or a text string in any font. The links are represented by various line widths, styles, and colors, and also may show direction. All items displayed with the widget may be selected and moved. The network displays also may be panned horizontally or vertically, and zoomed to magnify or reduce the network. The network widget can be used to display most any graph, but it has been used primarily within AT&T to display and manipulate large communication networks, some with as many as 5,000 nodes and 15,000 links.

Application Builder. The application builder is a tool to create and maintain widget-based applications. The builder itself is constructed from a collection of `xtent`-based processes. The elements of the application builder are a group of separate editors that manipulate and maintain X widget-based applications:

- *Layout*—An editor for manipulating the visual layout of a widget-based application. This editor is analogous to the WYSIWYG interface provided by other GUI development tools.
- *Resource*—An editor that allows manipulating the state of a widget. This can be both a learning tool and a debugging aid.
- *Connection*—An editor to set the connections between widgets and widget states. When a button is pushed, an application may, for example, display a menu. This sort of inter-widget communication can be described and maintained with the connection editor.
- *Widget tree*—An editor to display and manipulate the parent-child relationships between widgets. This is

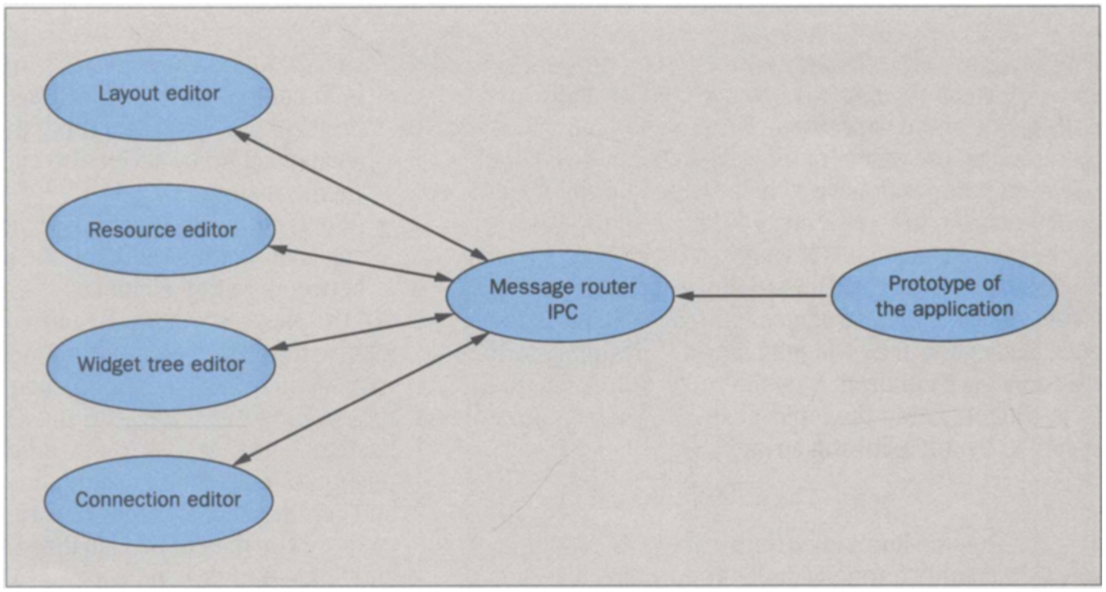


Figure 5. The `xtent` application builder provides four software editors, the layout, resource, widget tree and connection editors, to manipulate and maintain X widget-based applications. The application builder also creates a prototype `xtent` process. Messages between the editors and the prototype are handled by the interprocess communication (IPC) package.

also useful as a navigational aid for selecting and inspecting the values of parameters associated with the widgets.

The application builder creates a prototype `xtent` process, a separate process controlled via messages. The four editors comprising the application builder send messages to the router process that result in changes being sent to the prototype. The four editors have no notion of the specific process acted on, or the existence of the other editors. The relationships among the processes are illustrated in Figure 5.

This clean separation of components allows the editors to be easily modified or reused.

Figure 6 shows the application builder running in a typical construction session. The upper and lower windows are from the layout editor process. They allow the end user to create, destroy, and otherwise manage widget placement within the prototype. The middle window is the widget tree editor. It shows the widget structure of the prototype running in the window on the right. The widget tree editor allows the end user to quickly navigate the widget tree and select specific widgets. The application being manipulated in this example is an X to PostScript® language conversion program. (PostScript is a registered trademark of Adobe Systems, Inc.)

From the example, one can see that the prototype receives messages from the router to update itself. Updating may include creating or deleting widgets or updating the state of existing widgets. Messages are also sent to query the state of the prototype. The widget tree editor requires a list of the widgets to be displayed. This list is obtained by sending a message requesting that the return of the widget names be in another message. The message traffic to a prototype `xtent` process, or to a `C`-based Xt process, is handled without application intervention. Applications are unaware of message traffic. This feature allows applications to be easily reused by weaving them into the IPC fabric of another application.

Application Development in this Environment. The software tools and techniques described in this article have made a positive impact on software projects within AT&T, improving both the management and daily functioning of application teams. One proposed project was estimated to take two years and 75 people to complete, using traditional software development techniques. But by using these tools and techniques, the project was completed in nine months, and with only 20 people.

Several other projects using this technology involved small teams creating prototypes of potential applications used to determine a project's feasibility. This is a reversal of the usual process of producing a specification and then the prototype. This, of course, is only possible in a system that allows engineers to quickly and succinctly express their ideas. Instead of starting with written requirements gathered to develop the application, these tools allow a quick evaluation prototype to be

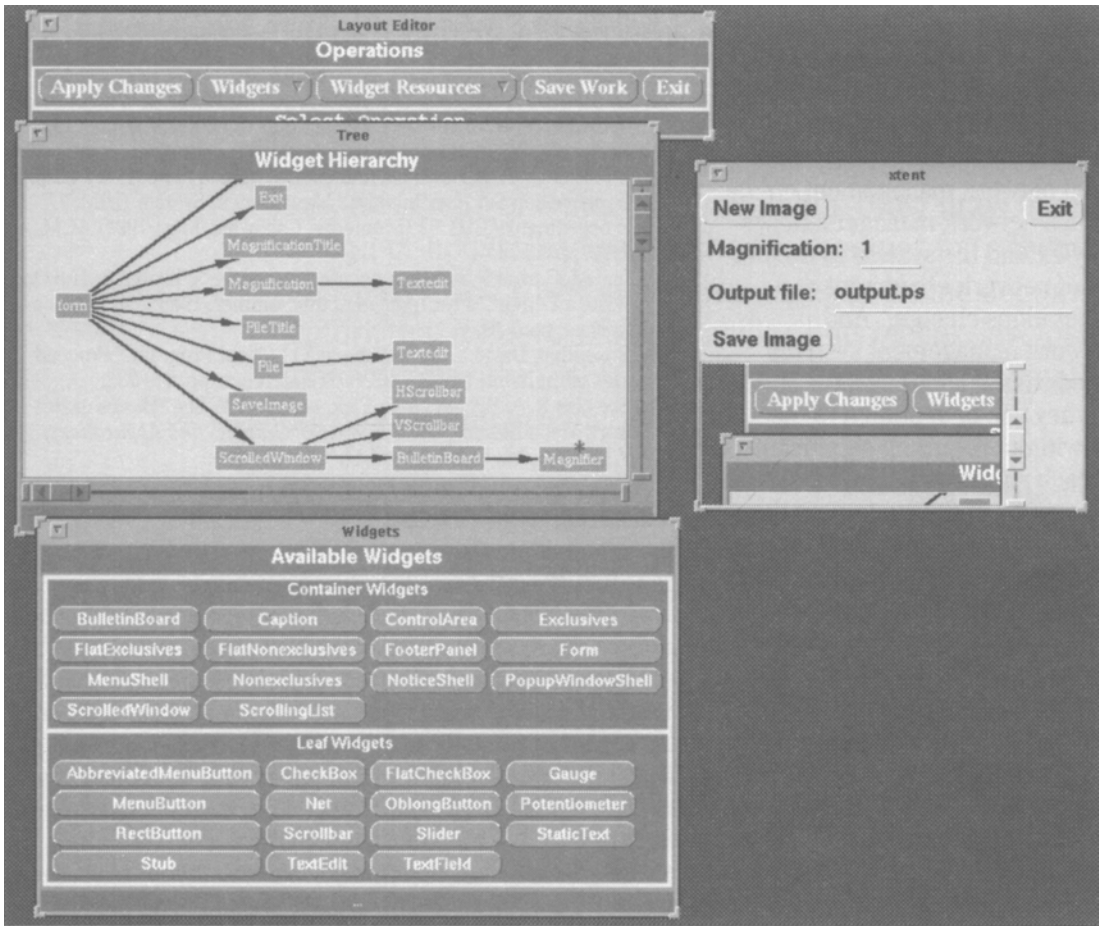


Figure 6. Here the application builder is running in a typical construction session. The upper and lower windows are from the layout-editor process. They allow the end user to create, destroy, and otherwise manage widget placement within the prototype. The middle window is the widget tree editor. It shows the widget structure of the prototype running in the window on the right. The widget tree editor allows the end user to quickly navigate the widget tree and select specific widgets.

created and shown to the customers. Their comments can be folded back into the prototype in minutes or hours, and the reworked prototype can be shown again.

Using prototyping in this fashion not only improves the requirements for the application, but also produces a skeleton or template from which the final product can be developed. Where in the past customers often did not "see" the application until it was nearly complete, the iterative approach brings them into the development process from the start.

The iterative approach also encourages early involvement of everyone on the project team. In the past, a project consisted of systems engineering, human factors, and programming teams, and the organizational or expertise boundaries that existed between all these factions. Allowing the user interface to be developed with specifications, rather than code alone, brings the human factors and systems engineers into the working realm of the programmers who, in their turn, can now concentrate more on the complexities of the application, and less on the color, size, and location of a pushbutton. This process allows the prototype to be designed and developed concurrently, resulting in faster overall system development.

Additional Challenges

Besides the challenge of developing applications with GUIs in a timely and efficient manner, GUI's face other challenges in network operations and management tasks:

Too Much Information. A network can generate thousands of events per hour, resulting in megabytes of information. Systems can filter and summarize the information, but at some point human beings have to process it. This paper has briefly discussed how graphics can be used to efficiently represent some network information.[11] However, new ways to graphically represent large volumes of network information still are needed.

Extensibility. Communication networks change and grow, and the required network management operations also change. How can a network management system be developed to easily extend the system as the network changes, new types of network equipment are installed, and company operations change? And, because customers of network management systems have their own special needs and operations, how can systems be developed so they can be modified to meet differing customer needs without requiring significant custom development for each customer? Can the system customization capability be given to the customers themselves? The "software tools" approach described in this paper is a step toward answering these questions, but more work is needed to make systems easily extensible.

Consistency. Often, network managers must use several different applications to get their job done. The design of user interfaces needs to be consistent, so the network manager can switch between applications without becoming confused, making errors, or being slowed down. Similar tasks should be performed in similar ways, a common vocabulary should be used, and symbols and colors should have consistent meanings. Achieving a common "look and feel" for user interfaces, when applications are developed by different organizations, is a great challenge. The current approach toward that end, within AT&T, includes using common design specifications and user interface development tools, user interface consistency reviews, and depends upon continuous cooperation among the development organizations.

References

1. R. S. Cohen, H. K. Kan, and R. J. Pennotti, "Unified Network Management from AT&T," *AT&T Technical Journal*, Vol. 67, No. 6, November/December 1988, pp. 121-136.
2. J. P. Cunningham, J. P. Rotella, C. L. Asplund, H. Kawano, T. Okazaki, and K. Mase, "Screen symbols for network operations and management," IEEE 1992 Network Operations and Management Symposium, Symposium Record, pp. 5.1.1-5.1.10.
3. R. W. Scheifler and J. Gettys, "The X Window System," *ACM Transactions on Graphics*, Vol. 5, No. 2, April 1986, pp. 79-109.
4. Sun Microsystems, *OPEN LOOK Graphical User Interface Functional Specification*, Addison-Wesley, 1989.
5. Open Software Foundation, *OSF/Motif Programmer's Reference 1.2*, Prentice-Hall, 1992.
6. D. Blewett, S. Anderson, and M. Kilduff, "Rapid Prototyping Tools for Telecommunications Applications", Proceedings of ACM/SIGAPP, March 1-3, 1992, pp. 46-52.
7. D. Blewett, S. Anderson, M. Kilduff, and M. Wish, "X Widget Based Software Tools for UNIX®", Proceedings of USENIX Winter 1992, January 20-24, 1992, pp. 111-123.
8. J. P. Rotella, A. L. Bowman, and C. A. Wittman, "The AT&T Display Construction Set User Interface Management System (UIMS)," Proceedings of CHI '92 (Monterey, California, May, 1992) ACM, 1992, pp. 73-74.
9. Joel McCormack and Paul Asente, "Using the X Toolkit or How to Write a Widget," Proceedings of the Summer, 1988 USENIX Conference, pp. 1-13.
10. Rosenthal, David S. H., "A Simple X11 Client Program," Proceedings of the Winter 1988 USENIX Conference, pp. 229-235.
11. See also R. A. Becker, S. G. Eick, and A. R. Wilks, "Basics of Network Visualization," *IEEE Computer Graphics and Applications*, V.11, No.3, May 1991, pp. 12-14.

(Manuscript approved July 1993)

James P. Cunningham *is a technical manager in the User Interface Architecture—Documentation and Systems Group at AT&T Bell Laboratories, Holmdel, New Jersey. He and his group are responsible for AT&T-wide technical planning in the areas of documentation, graphical user interfaces, and usability engineering methods. He joined AT&T in 1984 with a B.A. in psychology from Stanford University, Palo Alto, California; and a Ph.D. in mathematical psychology from the University of California at San Diego.*

C. Douglas Blewett *is a distinguished member of technical staff in the Software Techniques Research Department of AT&T Bell Laboratories, Murray Hill, New Jersey, where he is responsible for research in application development environments, with emphasis on interactive graphics and distributed computing. He was the AT&T technical representative to the MIT X Consortium during the early years of the Athena project. He joined AT&T in 1979 with a Ph.D. in vision and cognition from Rutgers - the State University of New Jersey.*

J. Scott Anderson *is a member of technical staff in the Network Management Operations Planning Department of AT&T Bell Laboratories, Middletown, New Jersey. He currently is engaged in graphics prototyping and application development, specifically graphical representation of AT&T network elements. He joined AT&T in 1985 with a B.S. in microbiology and an M.S. in computer science, both from Kansas State University, Manhattan, Kansas.*