Galadriel: A Display List-Based Window Manager

Bob Lewis

Graphics Workstation Division
Information Display Group
P.O. Box 1000, MS 61-277
Tektronix, Inc.
Wilsonville, OR 97070
tektronix!tekecs!bobl

ABSTRACT

This paper presents an architectural description of Galadriel, a window management system that provides both text and graphics services to client processes. Unlike most other window managers, Galadriel runs under UNIX† on a hosted, display list terminal instead of a bitmapped workstation. It discusses the advantages and disadvantages of this approach as well as areas for further development.

1. Introduction

Window managers are gaining increasing acceptance as a part of many engineering environments. Usually, however, they use a display connected directly to a single-user workstation over a high-bandwidth communications line such as a memory bus.

This paper discusses a window manager called ''Galadriel''[1] which shares many characteristics with previous window managers, but runs on a substantially different hardware configuration.

Galadriel runs under the UNIX operating system residing on a host computer and interfaces to several different terminal models over an RS-232 communications line. All of the models supported are display list-based; in additional to conventional ASCII text (''alphatext''), output to them may include encoded graphics primitives such as polylines, polymarkers, filled polygons, and graphic text (''graphtext'') which can be sent directly to the display processor or put in terminal RAM as a ''segment''. The host can then change various attributes of the segment, redisplay it, and replicate it without retransmitting the primitives.

Section 2 describes the target hardware and software constraints. Subsequent sections deal with the effects of these constraints on traditional window management capabilities. Section 3 covers the ways Galadriel organizes windows and related objects from the application's and the user's points of view. Section 4 briefly discusses the interfaces to these objects that Galadriel presents. Section 5 goes into more detail on the internal operation of the window manager process itself. Finally, Section 6 discusses possible directions for future development.

2. Hardware and Software Constraints

Galadriel was originally implemented on a Tektronix 4115B terminal connected over an RS-232 line to a VAX† (780) running an internal Tek version of (relatively standard) 4.2bsd UNIX. So far, it's been ported

———————

0 Current address: University of British Columbia; Department of Computer Science; 2366 Main Mall; Vancouver, BC V6T 1Z4; CANADA; Internet: bobl@cs.ubc.ca

† UNIX is a trademark of AT&T Bell Laboratories.

1 From J. R. R. Tolkien's *Lord of the Rings*, ''owner of magic mirror'' is about as close as we could come to a mythological ''window manager''.

† VAX and VT100 are trademarks of Digital Equipment Corporation.

to a different 4.2bsd variant (Tektronix's UTek[2‡]), two other host compute engines (Tek 6030 and 6130), and four other terminals (Tek 4111, 4125, 4128, and 4129).

2.1. Hardware Characteristics

Table 1 gives the relevant differences between the various terminals that affected system design.

model(s)	4111	4115B, 4125, 4128, 4129[3]
display size (pixels)	1024 x 768	1280 x 1024
character size (pixels)	8 x 16	8 x 16 or 16 x 30
#rows x #columns	48 x 132[4]	64 x 160 or 34 x 80
number of bit planes	4	2[5], 4, 6, or 8
display list RAM	1.2MB	768KB

Table 1 -- Display Characteristics

All terminals[6] accept the same set of over 200 commands. Other design-relevant features common to all of the terminals are:

- 64 viewports (i.e., clipping rectangles)
- 64 dialog areas (independent alphatext output areas, each emulating most of a VT100†)
- signed 32-bit integer display list coordinate space
- raster operations between screen, display list RAM, and host
- a 24-bit color map, indexed by the value of each pixel
- baud rates of up to 19.2 kbaud
- the ability to make any segment the cursor
- mouse or puck/tablet

2.2. Software Considerations

By far the most serious constraints on the design and implementation of Galadriel arose from two factors that set it apart from most other window managers now available. The first is that communication between the host and the terminal must take place over a serial RS-232 line. The second is that Galadriel's *modus operandi* must be acceptable on a host shared with several other users.

The RS-232 communications bottleneck means that the design must use whatever resources it can on both host and terminal ends to minimize the number of bytes that need to be transferred to get things done. Having a display list with definable segments is a big help with this, because once graphic primitives are put into a segment, that segment can be transformed and redisplayed as a unit – the primitives never have to be sent again.

The shared host requirement limits the desirability of tight interaction loops. For such a loop, the nature of the interface would require the window manager to poll the terminal continually for the cursor position. At 9600 baud, the transit time for the 20 or so characters involved in the poll is < 0.02 second, so the communications bottleneck is not the culprit here. The problem is that the window manager would be almost constantly writing to and reading from the terminal. Because input from both of them goes in though the tty driver, the operating system has no way to distinguish responses to polling from (asynchronous) user input.

---

2 The only change needed in the source code was caused by UTek's allowing 64 open file descriptors (cf. 4.3bsd) instead of the usual 20.

‡ UTek is a trademark of Tektronix, Inc.

3 The 4125 is an upgrade of the 4115B. The 4128 and 4129 include three-dimensional capabilities that Galadriel does not currently use.

4 Galadriel only uses the 128 columns that have pixels under them.

5 The 4115B has a minimum of 4 bit planes.

6 We'll use the somewhat inexact term ''412x'' hereafter to refer generically to all of the terminals in Table 1.

The window manager would look like a highly interactive process, even though there might be no input from the user taking place. The effect is that a process in a polling loop hogs as much system time as several conventional interactive processes. Galadriel should therefore avoid polling as much as possible.

The window manager process *windman* was designed to run as a normal user process and require no modifications to the 4.2bsd kernel. It has met these goals, although some speedups in the pseudo-tty driver have improved performance.

Existing programs should work without recompilation. This included not only ‘‘glass tty’’ programs like *ls*(1) and *ed*(1), but screen-oriented ones like *vi*(1), *emacs*(1), and programs using *curses*(3t)[7].

2.3. System Performance Goals

Galadriel was designed to support CAD systems. Its original clients included VLSI layout and schematic capture editors. As usual for such systems, performance is a critical requirement.

Galadriel maximizes performance by several strategies:

- Minimize the number of bytes sent to the terminal.
- Reduce the number of times scalars are encoded for transmission (most of the time, they are now encoded only once).
- Reduce the number of byte copies.
- Use 412x features (e.g., macros) to speed things up.
- Use efficient programming techniques (loop unrolling, etc.) in critical areas.

Benchmarks of the layout editor (Tek’s Leia) on a Tek 6130, comparing a version running under Galadriel with one that outputs directly to the terminal using a compatibility library, show less than 5% overhead (clock time) of the window managed over the non-window managed version.

3. Window Organization

The window manager process exists as a separate process servicing requests from you, sitting at the terminal, and one or more ‘‘client’’ processes (see Figure 1).

3.1. Windows and Window Ttys

Each of these clients owns one ‘‘window tty’’ (‘‘wintty’’) and one or more possibly overlapping, rectangular windows. Each window has a title bar, which displays the name of its wintty, which is of the form ‘‘/dev/tty??’’.

Internally, each wintty is implemented as a pseudo-tty (*pty*(4)), so its name is that of the slave end of the pseudo-tty whose master end Galadriel is watching.

3.2. Panes

Clients may subdivide windows into ‘‘panes’’, rectangular areas contained within windows which display output. Most of a window’s area is devoted to panes. Each window starts out with a single pane, called its ‘‘primary’’ pane.

The terminal hardware has no notion of windows or panes. The software treats windows as groups of panes that get created, deleted, reframed, etc. together, associated with a title bar. It then maps each pane to a list of ‘‘virtual’’ viewports.

This list may have no members, one member, or several members depending on whether the pane is completely obscured or collapsed, completely unobscured, or partially obscured, respectively. Actions affecting this pane, such as making a new segment visible in it, require a traversal of the list.

Galadriel maps the 64 most recently used virtual viewports to the 64 viewports that the 412x supports in hardware, so the window manager must maintain a relation between panes and segments to be able to actualize any virtual viewport at any time. For bookkeeping and memory reasons, however, each application is

7 Including, of course, *rogue*(6).

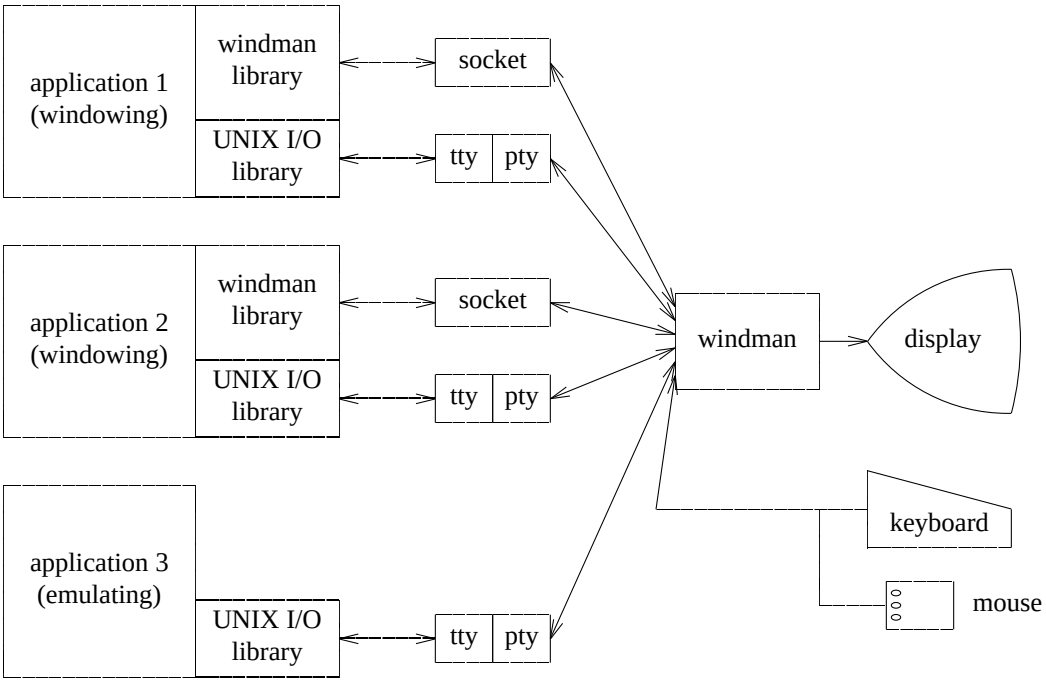

Figure 1 -- The Window Manager Process

arbitrarily limited to 128 panes.

The management of viewports is the counterpart to the management of on-display and off-display bitmaps that the Blit (see [Pike84]) performs.

3.3. The Client Models

The window manager looks at clients in one of two ways; ‘‘emulating’’ or ‘‘windowing’’, depending on the functionality the client requires.

3.3.1. Emulating Clients

Galadriel supports terminal emulation for conventional UNIX programs such as *mail*(1), *vi*(1), *ls*(1), etc. Emulating clients do not require recompilation to run under the window manager. Galadriel treats new clients as emulating by default.

By using pseudo-ttys along with the 412x’s dialog areas, each emulating client gets its own VT100-like screen. *Pty*(4) lets clients believe that they’re talking to a *tty*(4)-like device. Hardware support allows fast scrolling of even partially obscured windows and an emulating client never has to regenerate a window as a result of a window manager command (e.g. **Move**).

3.3.2. Windowing Clients

As Figure 1 shows, windowing clients are compiled with an additional library of window management functions. Section 4.2 will discuss these.

Only windowing clients can own more than one window or more than one pane. They can specify that one or more of those panes emulate a VT100 as above. They then select one of them to be the ‘‘text pane’’.

Most windowing clients, however, are more concerned with graphics output. Galadriel does not use pseudo-ttys for this, but sockets. This allows greater speed[8] and reduces the need for Galadriel to

8 On a 6130, for large (4K byte) block sizes, pseudo-tty’s are typically a factor of 10 slower than sockets, which makes the latter preferable for large segment definitions.

demultiplex emulated text from window manager commands.

The graphics that windowing clients can perform is modelled after the Graphical Kernel System (GKS). Unlike text output, this allows for simultaneous graphics output to multiple panes.

3.4. The Active Process

In the case of mouse[9] input, the cursor presence in a particular window implies that the recipient of the input should be that window's owning process, but keyboard and function key input has no such implied destination. Galadriel therefore designates one process as ''active'' and takes it to be the current focus of the user's attention. The active process gets keyboard and function key input.

There are also certain terminal-wide resources that can have only one controlling process. These include:

- rubberbanding
- the current color map[10]
- the cursor segment

It's convenient to let the active process also take charge of these, so that whenever a process is activated, Galadriel takes care of restoring these resources.

4. Window Manager Interface Implementation

Galadriel provides most of the features commonly found in window managers. This section will concentrate on how it does so within its design constraints. The window manager has three interfaces: user, shell, and programmer. The shell interface, however, doesn't use anything that the user and programmer interfaces don't use as well, so it won't be discussed here.

4.1. User Interface

4.1.1. Mouse Input and the Cursor

The 412x provides the ability to define a particular segment to be the cursor, tracking mouse movement without host intervention (i.e., polling). Whenever the user presses a mouse button, the terminal sends a GIN (graphic input) report to the host. It is up to Galadriel to decide what to do with the report, depending on what the cursor was pointing at as well as its own internal state.

4.1.2. Pop-Up Menus

One of the buttons on the mouse is designated the ''window manager button''. Pressing it brings up the system pop-up menu. With it, you can perform all of the basic window manager commands: **Activate**, **Bury**, **Collapse**, **Create**, **Delete**, **Expand**, **Move**, **Reframe**, and **Uncover**. **Reframe** is the only command that requires polling, since the 412x doesn't have a ''rubber box'' cursor to show the new frame of the window.

All of these commands are ''postfix'' or ''object-verb''. This means that you first point at the window you want to act on, request the system pop-up menu, choose a command, and the window manager acts[11].

Of course, application programs can define pop-up menus as well.

Pop-up menus must appear and disappear quickly. They could be implemented as titleless windows, i.e., viewports, but this causes problems when you pop-up a menu over a window containing many segments. Even though the window would be refreshed at the rate of tens of thousands of vectors per second, a few seconds is an unacceptable time for a menu to go away.

---

9 Galadriel works with either a mouse or a puck/tablet. We'll use ''mouse'' hereafter to refer to both.

10 Galadriel gives each process control of all $2^{\#\text{ of planes}}$ colors, although restraint is recommended.

11 One exception to this: because it's so final, the **Delete** command requires you to confirm the window you want to delete.

An alternative is to use the 412x pixel save and restore commands. These copy an area of the screen to and from display list RAM. Galadriel saves what's underneath a menu until the selection is made and then restores it before further output takes place. There are two drawbacks here:

- There can be no output to the terminal while the menu is up, since, unlike the Blit [Pike84], no output can go to the saved rectangle.
- The size of the menu is restricted, since there are 1.2MB of frame buffer and available display list RAM is usually much smaller than that.

The first drawback turns out to not be particularly annoying, and Galadriel satisfies the second by limiting the use of this technique to pop-up menus no larger than 32 characters wide by 16 characters high. It treats any pop-up larger than that with the previous window creation approach.

4.1.3. Scroll Bars

Clients can specify that any or all of their panes have scroll bars associated with them. Scroll bars on a pane allow a consistent and convenient way to browse its design space, assuming that that design space is larger than you can see all at once. Depending on the application, you can pan and zoom both horizontally and vertically[12]. There is also an ''overview button'' which toggles a view of the entire design space with the previous view.

Galadriel performs pans and zooms by sending a short sequence of commands to the terminal to change the pane's viewport transformation(s) and redraw the viewport(s). It does not need to resend primitives contained in segments. Unless the application has requested notification (see Section 4.2.2), it won't know that the pan or zoom has taken place.

Unlike those of other window managers (cf. Smalltalk-80† [Goldberg84]) Galadriel scroll bars are static. This means that a pane is created (optionally) with scroll bars and the bars stay with the pane until it is deleted. This has the disadvantage that the scroll bars, if present, always take up screen area, but this is outweighed by the advantage gained by not requiring the host to poll for cursor presence.

4.2. The Programmer Interface

This interface allows applications to control windows and output to them, as well as receive input from you. It has three ''layers'': UNIX I/O, window (WL), and graphics (GL). Within this interface, a program can only control the windows and panes that it owns, but it has a greater measure of control over those.

4.2.1. UNIX I/O Layer

This is the only layer available to emulating clients. As we've said before, output is directed to panes. A pane is created with an attribute that says whether or not it can contain alphatext, i.e., whether or not it needs to be able to emulate a VT100. The reason for this is to conserve the 412x's 64 ''dialog areas''. Each pane that can contain alphatext takes up one dialog area, and, unlike viewports, dialog areas cannot be virtualized.

To an application, such a ''text pane'' appears as a virtual terminal. The most commonly used ANSI X3.64 commands work in it, and under UNIX it has a TERMCAP entry describing these commands. Additionally, for emulating clients the TERMCAP includes the proper window dimensions[13].

The **Reframe** command causes problems, though. Although the window manager sets the TERMCAP environment variable to a temporary file containing an up-to-date *termcap*(5)-like description, there is no standard way to inform screen editors and the like that this has happened while they're running[14].

Input is slightly more complicated. Following GKS, GL considers keyboard input to be part of graphic input (GIN). There are two keyboard input models corresponding to emulating and windowing clients. Emulating clients have a control tty (/dev/tty) that behaves like a normal tty. They can call *ioctl*(2) to

---

12 Zooming maintains aspect ratio, so a vertical zoom also causes a horizontal zoom, and vice versa.

† Smalltalk-80 is a trademark of Xerox Corporation.

13 This also applies to windowing clients that don't split their primary panes.

14 The SIGWINCH and *tty*(4) enhancements made to 4.3bsd will be helpful here.

control echoing, interrupt characters, line editing, and other *tty*(4) features.

Windowing clients get all of their input from the GL GIN routines, and any attempt to read from their control ttys will block because GIN doesn't travel that way[15].

4.2.2. The Window Layer

There are 24 functions in this layer. WL duplicates much of the functionality of the user and shell interfaces for the programmer. In addition, the programmer has greater control over panes, including the ability to create and delete individual panes. The application can create non-primary (see above) panes by ''splitting'' existing panes.

A significant departure here from most window managers is the association of two boolean flags with each of several window manager and user actions. One flag is for permission and the other is for notification. The permission flag allows the associated action to be carried out on the entity to which it is bound. If the notification flag is set, after the action is performed (or attempted, if the permission flag is not set), the owning process learns about it in the form of a detailed GIN report.

Permission/notification flags are bound to both windows and panes. For windows, they control the actions the user attempts from the system pop-up menu. For panes, they determine the presence and function of scroll bars.

4.2.3. The Graphics Layer

GL is a set of 102 graphics routines. Although not entirely GKS-compatible (usually for performance reasons), about 90% of these routines map into a Level 2b GKS implementation with enhancements.

5. The *windman* Process Architecture

Figure 3 shows the flow of data within *windman*, the window manager process.

Basically, *windman* is a command-driven finite state machine. Every command, including ANSI text output and escape codes, WL and GL requests, keystrokes, and mouse button presses, are awaited by a single *select*(2) call in the queue manager. Once a command is received, the queue manager decides which of the four parsers; ANSI, GL, WL, or GIN; to pass the command to. Apart from saving and restoring some context information, the queue manager is the only part of the *windman* process that knows (or cares) that there may be multiple clients.

The ANSI parser is a finite state machine itself. All pseudo-tty output from applications goes through this parser. It sees that only complete ANSI commands get sent to the terminal and prevents certain other sequences (such as RESET TERMINAL and non-ANSI commands) from getting there.

The GL parser is responsible for maintaining client context, so that each client can have the attributes it sets present while it's writing primitives and defining segments. This parser also queues segment definitions until the client closes the segment. Both of these features are necessary because the terminal permits only one set of attributes and one open segment, at most, at a time. It also manages terminal memory and sends responses to clients when needed.

The WL parser deals with the higher-level commands that make up WL and the shell interface as well. Most of the time, about all it does is decode the arguments and pass them on to the window controller, which does the real work.

Both WL and GL parsers converse with the window manager library in each application over sockets using a syntax that resembles a remote procedure call, but is buffered and otherwise optimized for Galadriel.

The main purpose of the GIN parser is to see who gets keyboard and mouse input. Under most circumstances, Galadriel permits GINahead – you can perform mouse or keyboard input before it is requested[16].

---

15 A way to permit clients to do this is under consideration.

16 Implementing this is not as easy as it sounds.

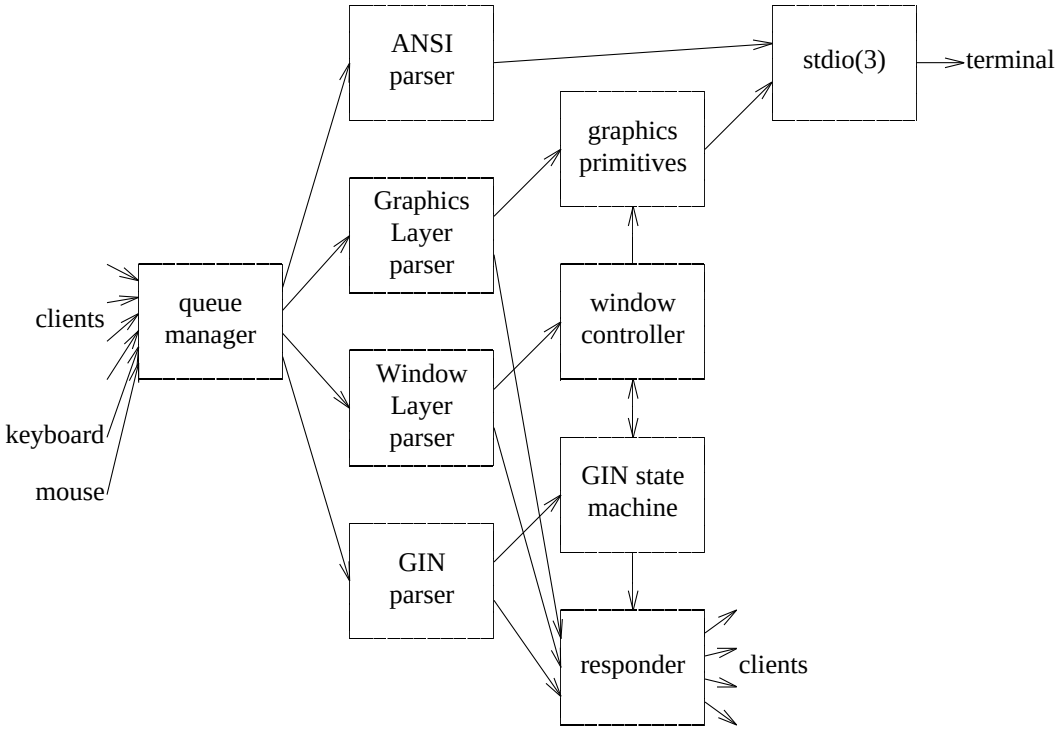

Figure 3 -- Dataflow Within *windman*

The GIN state machine handles what tight interaction loops Galadriel provides: rubberbanding, rubberboxing, and window moving. It also exerts some control over the queue manager. It can restrict the *select*(2) call to watch only the terminal when, for example, the user is making a selection from a pop-up menu.

The responder is called whenever it's necessary to send a GIN report or a command return to a client. The latter corresponds to the return mechanism from a remote procedure call, except that no ANSI and very few GL commands require a return.

All graphics output takes place through *stdio*(3) via a package of primitives which is the only part of Galadriel that knows 412x escape codes.

6. Future Developments

6.1. Faster Communications

As discussed in Section 2, the major bottleneck is the RS-232 communications line. Obvious possibilities are such things as Ethernet† or DMA communications between the host and the terminal. In fact, there already is a Unibus‡ interface card for the 412x, and the Galadriel project is looking at the capabilities it offers very closely.

As the communication speed increases, the distinction between host/terminal and workstation blurs. The terminal becomes more like a workstation with an independent display engine and a shared compute engine (the host).

---

† Ethernet is a trademark of Xerox Corporation.

‡ Unibus is a trademark of Digital Equipment Corporation.

6.2. Increased Hardware Support

The 412x was not originally designed for window management. Galadriel has indicated some likely functions to migrate from software to firmware and hardware. These include:

- the pane → virtual viewport → hardware viewport mapping
- output simultaneously to several panes
- better support for tight interaction loops
- context settings
- retain non-segment output during moves and occultations

6.3. Improved Pseudo-Tty Support

Under 4.2bsd, the number of pseudo-ttys is fixed in the kernal and must be created by /dev/MAKEDEV. This is rather artificial. It would be preferable to invoke *open*(2) on ‘‘/dev/pty’’ or whatever and have that create a new pseudo-tty automatically. An *ioctl*(2) call would return the actual names of the pair. The device should go away whenever the opener of the master end closed it, treating it as a hangup (i.e., SIGHUP) on the slave end.

Alternatively, Ritchie’s ‘‘streams’’ (see [Ritchie84]) would also provide a better way to implement pseudo-ttys.

6.4. Bitmaps

Although the 412x has several commands to copy between host, screen, and display list RAM, they are neither orthogonal nor functionally complete. These should be cleaned up, since as communications become improved, it will be feasible to treat the 412x as both a bitmapped and display list device, and let the clients choose accordingly.

6.5. Distributed Windows

Any emulating client can run *rlogin*(1), so this feature is already partially present. For windowing clients, intermachine sockets and a naming server (to get the name of the local *windman*’s socket to the remote client and vice versa) are needed.

6.6. More Segment Memory

Large applications can use up the 3/4 megabyte or so of display list RAM in the 412x with relative ease. Two ways to alleviate this are (1) more display list RAM (obviously) and (2) virtual storage in the terminal. If there were greater bandwidth between the terminal and the host, it might be feasible for the terminal to use the host’s virtual memory.

Acknowledgements

Paula Mossaides is Galadriel’s project manager. In addition to the author, designers, implementers, and maintainers are/were: Scott Hennes, Donna Nakano, Karen Palmer, Rob Reed, and Bob Toole. Evaluators were: Keith Koplitz, Larry Jones, and Max Miller.

References

[Goldberg84] Goldberg, A., *Smalltalk-80 The Interactive Programming Environment*, Reading, MA: Addison-Wesley, 1984.

[Pike84] Pike, R., ‘‘The Blit: A Multiplexed Graphics Terminal’’, AT&T Bell Lab. Tech. J., *63*, No. 8 (October, 1984).

[Ritchie84] Ritchie, D. M., ‘‘A Stream Input-Output System’’, AT&T Bell Lab. Tech. J., *63*, No. 8 (October, 1984).