What is Emacs?

UniPress Emacs began life as a text editor with multi-window and extension capabilities, and over the years has grown into a complete working environment with a window manager, process control, and a simple database facility. Even in its intermediate stages, Emacs served very well as a consistent front end for line-oriented applications, such as shells and mail readers. For example, Emacs is able to act as a window manager on character terminals, allowing a shell to be run in separate portions of the screen so that the user can cut and paste between the shell windows. So, Emacs has always been able to act as a portable user interface that can be easily customized by its users. It is continuing to grow towards a complete application development and delivery environment.

UNIX Emacs Version 2.20

Version 2.20, expected for Summer 1988 release by UniPress, continues this evolution by extending the set of Emacs user-interface tools. These tools are available both to Emacs and to extension-language programs written by users. Thus, users' own programs can take advantage of the device-independent interface facilities that we are creating. We believe that these facilities will greatly ease the software developers' efforts to make their applications communicate with users.

In addition, Version 2.20 makes use of the facilities of window systems, such as NeWS ™, SunView ™, and X-Windows, so that Emacs now resembles more closely the other applications that run on those systems. Most importantly, any extension language programs can also use the window-system facilities portably. For example, you can write a user interface in Emacs using menus and dialog boxes, and it will run on any of the supported window systems without modification.

Multiple Frames

Version 2.20 has multiple frame capabilities on window-system environments. All previous versions of Emacs have maintained multiple "Emacs windows," known as *panes*, within just one window-system window. With SunTools™, for example, Emacs V2.15 runs in a Sun window. The user can re-size the window, etc., but all Emacs output is directed within that one Sun window. Emacs Version 2.20 can maintain one or more window-system windows, known as frames, which present views of Emacs buffers. Each frame resembles a traditional Emacs window, consisting of a set of horizontal panes, each presenting a view into a buffer, separated by modelines. Thus, a single Emacs session can display its output in multiple window-system windows. Users can navigate as easily between frames as between panes. This new multiple frame capability will give Emacs users the full benefits of their window system. We expect that users will typically display things related to the same task in the same frame. For instance, one frame might be used for compilation, with one pane displaying the compilation output log and the other pane displaying the site of the next error message found in the log. At the same time, another frame might contain a single pane running a shell window, while another frame might be running the Info documentation reader. Another use would be pop-up help windows occupying their own window space on the screen so that they do not take up any space from the main Emacs text window. Of course, users are free to use the multi-frame capability any way they wish.

Summary

Emacs V2.20 may be thought of as "regular Emacs" with connections to the user-interface toolboxes provided by the window systems themselves (menus, buttons, sliders, etc.). Thus, Emacs becomes an application development environment which includes a powerful text editor. In addition to the user-interface tools provided by the window system, Emacs adds some tools of its own.

Emacs 2.20 provides both text editor windows and listener windows. Text editor windows allow an application to present a piece of text that the user can peruse or edit using the full power of the Emacs text editor, including keyboard macros, abbreviations, and cut/paste. User may choose the style of text editor commands that they prefer, such as vi (e.g., j moves down a line) or traditional Emacs (^N moves down a line). Listener windows allow users to interact with a running program. A listener window familiar to most Emacs users is a shell window. These two specialized window types are of general value to most applications, and can be put to great advantage by developers for their own applications.

Emacs Internals in V2.20 and Future Plans

Emacs has been internally reorganized as a set of libraries that may be used in various combinations for different tasks. In V2.20, these libraries reside in a single program. Future releases will allow the libraries to be distributed among different processes, possibly running on different processors. This distribution is accomplished using the SoftWire™ network-extensible library interface, described in the next section.

Distributing the pieces of the Emacs session among different processors allows the editing of, say, a text file, to be done on the processor that has direct access to that file, while the displaying of the text and the management of user interface objects (menus, dialog boxes, etc.) is done by the processor connected to (or inside) the user's terminal. This scheme is further discussed in the sections on SoftWire and the User Interface Library.

Future releases of Emacs will be able to edit objects other than text using commands that parallel the commands used for editing text. Different views of the same object (including text) may be seen in different frames. A "view" is a graphical interpretation of the object. Emacs has always displayed a simple view of the text in the buffer, wherein newline characters separate the lines of text and tab characters move to the next tab column, and all other characters are displayed as themselves. The reorganization of the editor allows different types of views to be presented of the contents of the buffer. The buffer may still contain printable text, though it may be displayed in a special way. Fig. 1 shows how views project a representation of an object onto a display pane.

As an example, suppose we have a buffer that contains several pairs of graph coordinates expressed as decimal numbers in ASCII, and we want to display them on a coordinate plane. We could create a view that projects

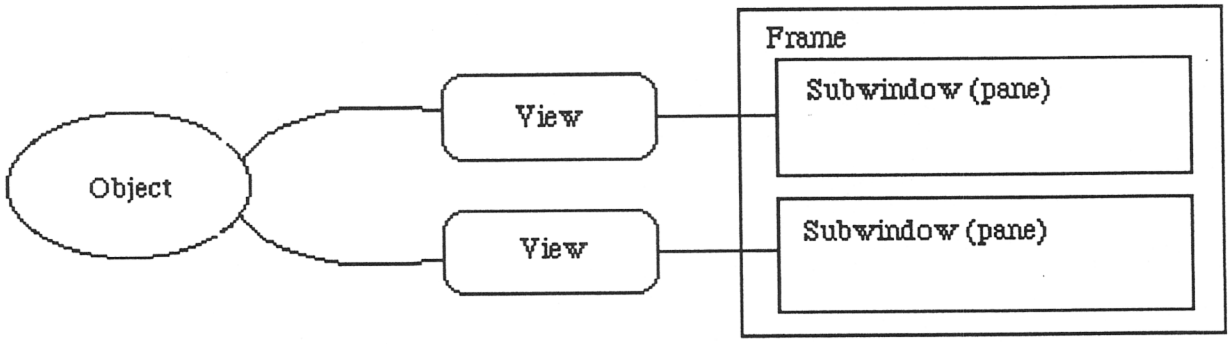

Fig. 1. A view projects a representation of an object being edited onto a display pane in a frame.

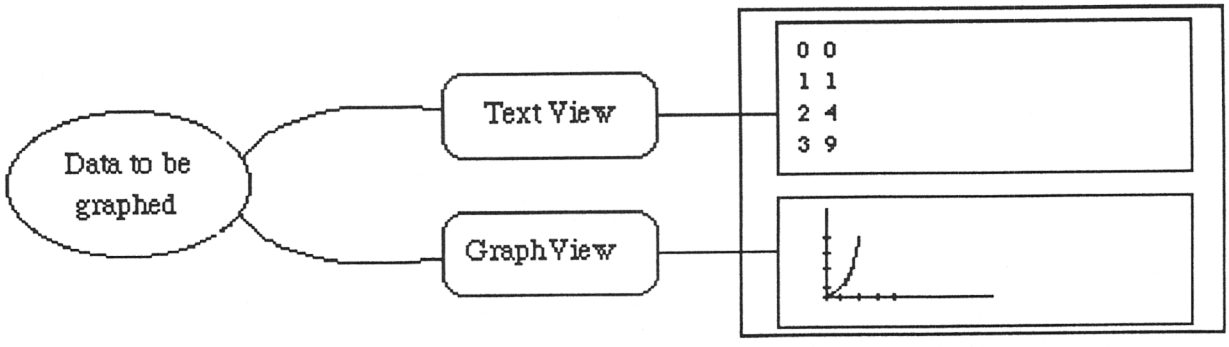

Fig. 2. The simple text view of the graph data appears as pairs of decimal coordinates. The graph view renders it on a coordinate plane. Both views may appear at the same time in different panes.

Emacs Internals (Cont'd)

draws and labels the axes appropriately and plots the points. Fig. 2 shows such a buffer being displayed as simple text in one frame and as a graph in another frame.

Future releases of Emacs will provide more sophisticated views for text to accommodate formatting (typeface changes, placement, etc), and will allow new views to be defined by the user.

SoftWire

SoftWire is a set of tools that allows a user-supplied library to be used by a program over a network. It can loosely be described as "NeWS without graphics," but while NeWS provides only a PostScript graphics library, SoftWire can act as an interpretive interface for any library. A SoftWire *node* is a program that consists of the SoftWire libraries, which provide a portable interprocess communication package and a protocol language interpreter, together with whatever libraries are to be made available to client programs. Linking a library into a SoftWire node creates a network-accessible interpreter for that library. For example, linking in the UNIX *curses* library produces a *curses* interpreter, and linking in SunView libraries yields a SunView interpreter.

SoftWire nodes communicate with each other by using a dynamic protocol language called NetScript™. This language closely resembles Adobe Systems' PostScript® language. It may be thought of as a dynamic remote procedure call interface that allows work to be done where ever it may be done most efficiently. Network traffic may be minimized by having the client load reusable procedures into the server. This technique allows the client to send the data in some compressed form, after having sent a procedure that the server can use to uncompress it. The server need not know ahead of time how the client intends to compress the data. This is the way NeWS operates, except that NeWS is primarily a display server, whereas SoftWire can act as a server for any library.

NetScript provides basic control operators and data structures modeled after those found in PostScript. It also includes a lightweight process mechanism and an event distribution mechanism compatible with those found in Sun's NeWS window system. The NetScript language is compatible with PostScript, except that it lacks the graphics operators. When a user's library is linked into the SoftWire node, its functions become available to NetScript programs running in that node.

Note that many of the tools available for use under NeWS are also useful for SoftWire. For instance, the NeWS *cps* utility can also be used with SoftWire to allow C programs to call upon NetScript code running in a SoftWire node. Other tools will be provided as part of SoftWire, such as a C to PostScript compiler, and an automatic library interface compiler for generating SoftWire nodes from a native-language description of the library calling sequences. Most generally, a SoftWire application is organized as shown below in Fig. 3.

As an example, Emacs can be structured as a SoftWire application simply by linking it into a SoftWire node. In this case, the "user libraries" mentioned above are the libraries that comprise Emacs itself. Fig. 4 illustrates

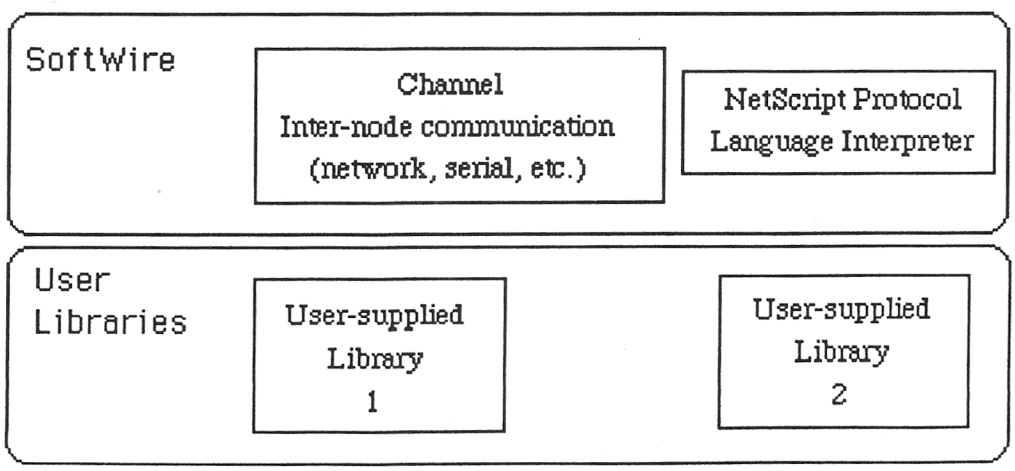

Fig. 3. A SoftWire node contains the SoftWire communication libraries and NetScript protocol language interpreter, together with libraries supplied by the user for network client

how Emacs fits into a SoftWire node. Such an Emacs session can communicate with other SoftWire nodes, such as database servers, and can be called upon by other nodes. An Emacs session might connect to another to provide a conferencing facility similar to the UNIX *talk* program.

We can distribute a single Emacs session over the network by dividing it into pieces that may each be placed in a separate SoftWire node. In this example, the display and low-level input is handled by a SoftWire node known as the **presentation engine**, while the rest of the Emacs session runs in a separate node. The presentation engine always runs on the machine connected to the user's display/input device, but the rest of the session may run anywhere on the network. Future versions of Emacs will be structured in this way. The diagram in Fig. 5 shows an Emacs node connected to a presentation engine for its display and input, and to a SoftWire database server node for the benefit of some Emacs subsystem that needs to store things in a database.

User interface library

The presentation engine isolates the Emacs session from the particular window system or display hardware being used, allowing the client to deal with the display in a device-independent language. Each window system, such as SunView or X-Windows, typically provides window management functions for maintaining overlapping display surfaces on the screen, and a set of graphics functions for drawing things on those surfaces. The presentation engine makes those (and other) functions available for use by client programs such as Emacs. NetScript programs running in the engine translate the system-independent protocol used by the client into calls to the window-system's functions. For example, to create a window on the screen, a client might send

```
/win framebuffer /new LiteWin send def
/reshapefromuser win send
/map win send
```

The NetScript program in the engine interprets this code in terms of the functions that create the window on that particular window system.

Note that NeWS does not require a separate presentation engine, because it talks to network clients in the same way that SoftWire does, and because it can execute NetScript code directly. Other systems require the presentation engine to make them resemble a NeWS server in those respects. Fig. 6 shows three presentation engines.

The SoftWire user interface library consists of NetScript code that runs in the presentation engine. It allows clients to present menus, dialog boxes, editor windows, and listener windows, in a system-independent language.

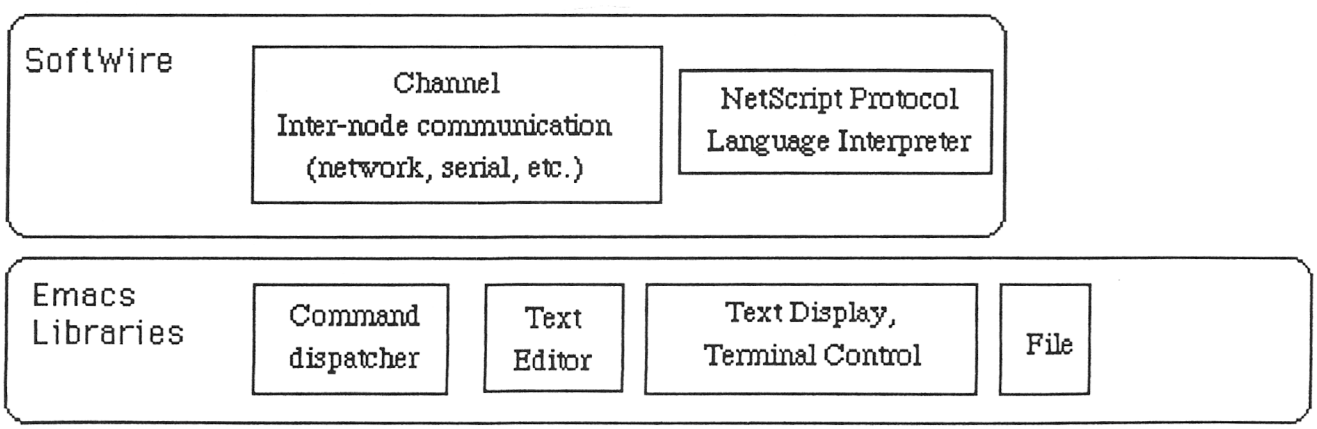

Fig. 4. Emacs can be embedded in a SoftWire node so that the Emacs session may communicate with other SoftWire nodes, which may in turn call upon it.

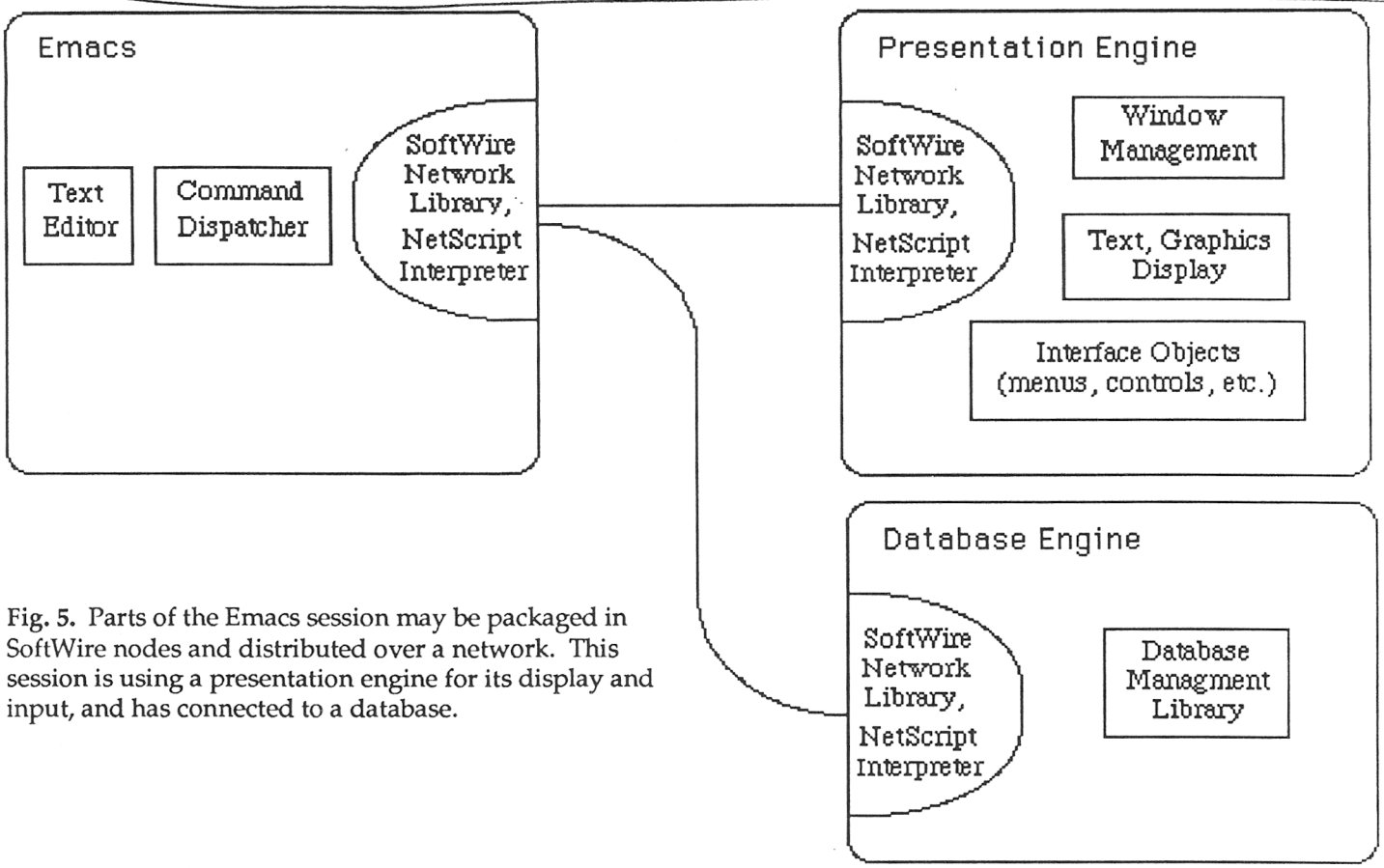

Fig. 5. Parts of the Emacs session may be packaged in SoftWire nodes and distributed over a network. This session is using a presentation engine for its display and input, and has connected to a database.

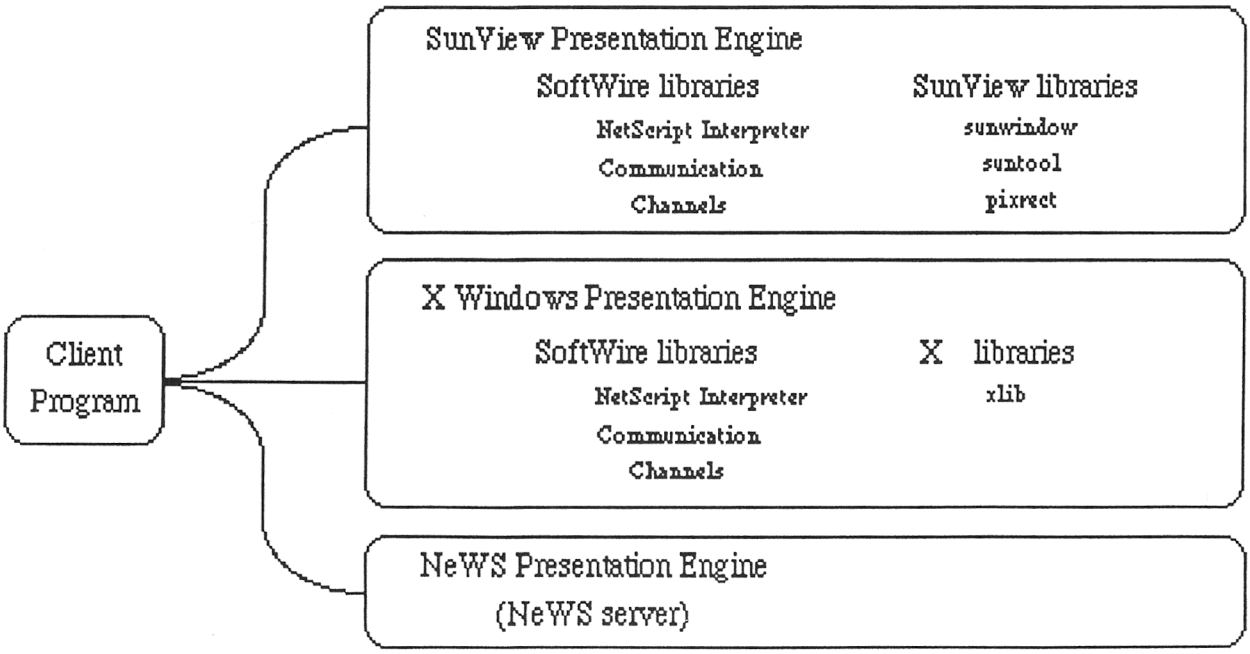

Fig. 6. The presentation engine makes the native window system's functions available tc SoftWire clients over the network. The SoftWire user interface library runs in the engine to provide a system-independent interface for clients.