The terminal for the Military Message Experiment

by ROBERT STOTZ, PAUL RAVELING and JEFF ROTHENBERG

USC/Information Sciences Institute
Marina del Rey, California

INTRODUCTION

One of the technical challenges faced in the Military Message Experiment (MME) is providing a system that is easy to learn and operate by the typical action officers who are the users of the message service. These people have no computer background and have neither the time nor the inclination to master a complex system in order to accomplish a simple task such as reading their message traffic, which they already do effectively. The system must offer some new capabilities to make it attractive, but above all it must be comfortable and natural to use. A most critical ingredient of the user's interface to the system is the terminal.

To provide the desired naturalness and ease of use, we want to make available the same sorts of facilities that the user has at his desk, where he is able to scan quickly through large amounts of data, see whole pages of text at once, and easily change his attention back and forth between pages of different documents. He is able to view several documents at the same time and edit or annotate a message at any spot by simply writing there. With paper and pencil these abilities are effortless and natural. We would like to be able to give him the same capabilities in our on-line system with the same simplicity of use.

Unfortunately we are constrained by such practical considerations as the cost of the terminal hardware, its maintainability, the amount of desk space that it occupies, etc. Like most such devices, the MME terminal is a compromise between conflicting goals.

In the early days of the MME program it was envisioned that users would be distributed all over the island of Oahu and that the host computer would be on the mainland. The long delays involved (a satellite hop and an unknown number of network nodes) argue for providing buffering and processing power in the terminal itself in order to guarantee responsiveness to the user (an essential attribute of a natural interface). Economics restricts the speed and memory size of the terminal processors. The terminal that has resulted is different in concept than any currently on the market.

OPERATING WITH THE MME TERMINAL

The style of interaction we have attempted to achieve is to have the user feel he is talking directly to the application program, with the terminal transparent to him. Simple but powerful two-dimensional editing functions are available in the terminal with a one-to-one relation between a keystroke and the execution of a function. The editing operations automatically format the screen dynamically so at all times the user sees a well ordered presentation of his data. Because of instant response to these operations, the user feels he is editing the document directly (rather than executing abstract commands to a computer which makes the changes for him, which is really what is happening).

To further this impression, the terminal masks the user from its memory limitations. The user can scroll through a full document without having to break it into pieces that will fit on screen or in memory. To achieve this the terminal has considerably more display buffer memory than screen space. The terminal gives the user the impression that it holds the entire document locally, even though it is really bringing new data in from the host as needed.

The naturalness of the interface rests on the principle that "what you see is what you get." The user merely edits a text image on screen to cause the system to make the semantic changes to the database that those editing changes imply. This editing often has the side effect of controlling some operation; for example, the addressees for a message are thought of as simply names in the "To" field of the message, which is filled in by editing the screen along with all the other message fields. The user is not forced to think of these names as active arguments to the message sending process, which the application program must parse, check for validity, and perhaps correct, before accepting them.

To allow flexibility in the data presentation made to the user, the terminal has a variety of highlighting facilities, such as inverse video, underlining, and half-intensity. Basic editing and dynamic formatting are done in the terminal, but the application program has control over how text is highlighted, what format constraints apply and where editing is allowed to occur, with granularity down to a single character. Thus the data sent to the terminal contains editing, highlighting and formatting attributes, as well as the text itself.

To give a user the ability to view several objects simultaneously or quickly switch to views of different objects, the terminal provides a facility called "windows." Each window can hold a different data object and can be independently scrolled, giving the user the impression that he is...
working directly on the entire document. Windows may be established by the application program to occupy only part of the screen, so several can be displayed at the same time. They may be moved on or off screen almost instantaneously, making it easy for the user to shift his attention between objects.

The terminal also supports a variety of command language styles. The terminal has function keys whose actions are assigned by the application program, but it also allows typed command input, menu selection (using the cursor to point), and an ability to refer to arbitrary characters on screen as arguments to commands.

IMPLICATIONS FOR THE APPLICATION PROGRAM

The terminal's style of interaction has profound effects on the application program. First it implies a rich data structure. A sequential file does not lend itself to arbitrary editing conveniently. Whatever representation the application program has of the documents it deals with, it must be able to extract and send to the terminal the controls for editing, highlighting and formatting along with the text. It also must be able to change its database to match the changes reported from the terminal, and make correct semantic interpretation of these changes. We call this application resident model of the data in the terminal the "Virtual Terminal" (VT). There must be a VT for each active window in the terminal.

In most alphanumeric applications, changes to the database are made only as a result of the execution of some command through a command interpreter. The semantic content of the command is extracted, the change is made to the database, and then the appropriate display information is generated from the new data and sent out to the terminal. SIGMA, the application program for the MME2, operates in this manner on many of its commands.

SIGMA, however, also allows changes to the database through screen editing. In this case the changes to the text that are sent by the terminal go to the virtual terminal, where their semantic impact is interpreted, and the database updated. In this way the meaning of the database can be changed as a side effect of editing, rather than by direct command execution. For example, while filling in a message form the user enters the contents of a message address field. So far as the user or the terminal is concerned this is just text, like any other text in the message. But since it is an address field, SIGMA parses the text, extracts user names and builds an address structure in the database, one element per addressee. If the text or format of the address field is changed as a result of SIGMA's semantic interpretation (e.g., addressees' names might be corrected), the necessary modifications are sent out to the terminal.

Since the terminal does its own formatting, the application program considers text of a paragraph as a continuous stream. Only where the structure of the text dictates (e.g., paragraph beginnings) does the application force format controls. The usual ASCII control characters CR (carriage return) and LF (line feed) do not appear in SIGMA's representation of text.

THE USER'S VIEW

The MME terminal is a Hewlett-Packard 2649A CRT Terminal (an OEM version of the HP2645A terminal3), with microcode supplied by ISI. The CRT holds 24 lines of text. SIGMA permanently assigns the top three lines as Status lines, where system and user status is continuously reported and error messages appear. The most frequently used SIGMA commands are assigned to 30 function keys. Other commands must be typed in the Instruction window, which occupies the next two screen lines below the status lines. The remaining 19 screen lines make up the working space; they may contain a View window for reference only, a Display window for editing, or a split between Display and View to allow referring to one object while working on another.

Two keypads next to the standard keyboard keys control the local terminal operations, including cursor movement (by character, word, line or window), character insertion (any of the normal typewriter printing keys), deletion (by character, word or line), scrolling of independent windows, and a special function called HERE. The terminal maintains the screen format, automatically breaking a long line at a word boundary and wrapping the remainder onto the next line. The carriage return key forces subsequent text to start on a new line.

FUNCTIONAL DESCRIPTION OF THE MME TERMINAL

Since the 2649 is microprogrammable, the functional operation is entirely defined by the microcode. Communication between the application program in the host computer and the terminal is done in blocks of data, representing a complete command from the application to the terminal (dispatch) or a complete report of some new condition in the terminal to the application (notice).

The terminal is basically a half-duplex device, in that it is either in input state (keyboard active) or output state (host computer active). During input state the user has at his disposal the full screen-editing capability. The terminal switches to output state whenever a function key is pushed (e.g., the EXECUTE key, which causes SIGMA to interpret and execute the contents of the Instruction window, is a function key). During output state the keyboard is disabled. The terminal is returned to input state when the host sends a special "Continue" dispatch. Strictly speaking, the system is not half-duplex because in output state the terminal does send certain control notices required to maintain consistency between the terminal's database and the host's model.

Communications between the terminal and the host is really one computer talking to another. Each transmission must be error-free; otherwise the computer's model and the terminal's model may not match. To insure the needed reliability of data across potentially noisy lines, a fully synchronized block retransmission protocol is used.
Windows

The MME terminal is designed to hold up to seven separate items of text in areas called "windows." Windows are allocated and deallocated by the host (never by the terminal). They are of arbitrary size so long as the total contents of all the allocated windows does not exceed the memory capacity of the terminal. Although the host fills the windows by sending data to them, the terminal does its own memory management and decides what data to keep when it nears its memory limits.

Windows may be thought of as numbered text buffer areas. A window may be assigned to occupy any contiguous portion (full horizontal lines) of the screen, such as lines 15 through 23, with an operation called "map." Normally a window contains more lines of text than will fit on the mapped area (it may also contain less). "Map" places that portion of the data from the window that will fit onto the screen, while any excess data beyond the screen area is stored in "margins," areas logically considered to be above and below the window's mapped screen area. Data scrolls on-screen from these margins.

Several windows may be mapped on different areas of the screen at the same time, which allows a user to view several text objects simultaneously. A window may also be unmapped, which means it remains in the terminal memory, but is not visible to the user. Figure 1 illustrates the terminal with four windows, three of which are mapped. The host may switch the contents of the screen from one text item to another very quickly by "unmapping" the displayed object and mapping the new object.

Mapped windows scroll independently. The ROLL UP or ROLL DOWN keys cause scrolling in whatever window the cursor is in. This operation is done entirely within the terminal, without telling the host. Therefore, although the host always knows what text is in each window and what windows are mapped onto which screen lines, it usually does not know exactly what text is visible on screen at any time.

Domains

In the MME terminal all text is stored in "domains," the atomic unit for the communication of text between the host and the terminal. Each window is made up of a contiguous string of domains. Domains may be any length up to 100 characters. Any character stored in the terminal can be uniquely identified by its window, domain identifier within the window, and character position within the domain. Domains have format, highlight, and control attributes assigned when the domain is created. The user is not aware of the domain structure of text, except as domain attributes are apparent to him. Figure 2 illustrates the domain structure for part of a SIGMA display.

Normally each domain starts at the character to the right of the last character of the previous domain and may wrap around onto the next line. However, the application program may set a domain to be "formatted," which makes the domain start on the next line at the left margin of the screen, regardless of where the previous domain ends. In this case the blank space to the right of the previous domain is essentially undefined to the application program, since it cannot be identified by domain ID and character position. The terminal will not allow the cursor to move to an undefined location. If a user attempts to move his cursor into such an area, it will jump to the next enterable domain.

The 2649 allows any combination of blinking, underlining, inverse video and half brightness on a character-by-character basis. The MME terminal limits this highlighting to a domain basis; that is, all characters in a domain are highlighted the same. Character set selection is also done as a domain attribute.

Domains also have editing control attributes set by the host when domains are created. They control whether the cursor may enter the domain, whether the domain is editable (from the keyboard), whether characters within the domain may be marked with a HERE, and whether the domain will accept carriage returns. Space is left for assigning other attributes.

When a user edits text he is changing the contents of some domain. If, for instance, he inserts or deletes characters, the domain expands or contracts appropriately and the domain is recorded as "Changed." Nothing is sent to the host until the user begins to edit another domain, at which time the terminal will send the host the new contents of the previously edited domain (via a "Changed Domain" notice) and record this new domain as the Changed domain. Thus the host computer may be, at most, one domain change behind what is in the terminal. Eventually the user will push a function key, carriage return, or the HERE key. The first action the terminal takes on these keys is to lock the keyboard and send out the pending Changed Domain notice if there is one. It then processes the key pushed, which ensures that the host always has the up-to-date state of the terminal before performing any operation on the data (all SIGMA commands are initiated by a function key).

Most often the data displayed come from the application program, in which case the host computer creates the domains and sends them to the terminal. However, the terminal will generate domains in three special instances.

1. When user editing causes a domain to exceed 100 characters, the terminal will generate a new domain and tell the host its location, ID, and contents in the form of a special "Extraction" notice.
2. A carriage return creates a new "formatted" domain and a special "EOL" notice is sent to the host, reporting the location and the ID of the new domain.
3. The HERE key marks the character at the cursor position as a parameter for a subsequent command by splitting it off into a new, one-character domain with inverted video and non-editable attributes. This mark will stay associated with that character regardless of any editing the user might do thereafter before the command is executed. Merely reporting the character position in the domain and the domain ID is not sufficient. The "HERE" notice contains all the information needed to identify the position of the marked character,
how the old domain was split, and the ID of the new domains created.

Flash lines

The terminal has an eighth window called the Flash window. If it exists at all, it is assigned to the top of the visible screen, and can be set by the application to occupy from zero to 24 lines. It has no domain structure and has fixed attributes. It is always non-enterable, and has no highlight or formatting properties. SIGMA uses the Flash window for status and other output-only information, since it does not have to keep around a corresponding data structure for this window.

Cursor control

Although there is only one visible cursor, each window may have an implicit cursor position. When more than one
Figure 2—Domain structure.

From the collection of the Computer History Museum (www.computerhistory.org)
window is mapped on screen, the UP WINDOW or DOWN WINDOW key causes the cursor to move to the implicit cursor position of the adjacent window.

The host has two controls over the implicit cursor for a window: 1) On what character in the window it should be and 2) on what line on the screen (for mapped windows) it should be. For unmapped windows, the latter translates to "on what line on the screen it would be if the window were mapped." A separate dispatch is provided for each. This limited form of cursor control is the only way that the host can establish what data is shown on screen. Since the user can scroll the screen contents, this is transient control at best. The host also has a dispatch to put the actual visible cursor into the desired mapped window (at the implicit cursor position).

**Scrolling**

The terminal’s basic heuristic for mapped windows is to keep extra lines of text in margins above and below the lines that are on-screen. This lets the user scroll in either direction without having to go to the host for more data. Whenever the terminal assesses that a margin is getting too small, it will send a “Vacancy” notice to the host asking for more data for that margin. The terminal calculates the number of lines of data to ask for based on the size of the on-screen area, the number of lines in the margin, and the amount of memory left in the terminal.

**Memory management**

The 12K bytes of display memory in the terminal are allocated as necessary for each dispatch. When the remaining memory is reduced to a prescribed limit, the terminal tries to reclaim memory from unmapped windows, and if that does not yield enough, from large margins of mapped windows. Memory is reclaimed by deleting domains and their contents from the edge of the margin and then telling the host through a “Scroll” notice. A Scroll notice identifies the last domain deleted and from which margin it came. It is important that the terminal be able to generate Scroll notices even when the keyboard is locked and the host is in control, for it is during this state that the host will send the data that reaches the memory limit. The terminal must be free to reclaim memory right away in order to have room for the next dispatch, which may already be in the terminal's input buffer.

It is possible for the terminal to try to reclaim memory from the same margin into which the host is writing. To prevent this, the host can set special “No Reclaim” controls for each margin of each window, and the terminal will not reclaim memory from a window margin so marked. The host must be careful not to leave these No Reclaim controls on, or the terminal will quickly run out of margins from which it can reclaim memory and the terminal memory will become full.

**CONCLUSIONS**

The MME Terminal is now operating with SIGMA at CINCPAC Headquarters. It appears to be successful in providing an interface that is quick to learn and natural to use. Editing is easy to master, since the operations provided are simple and their results immediate. The user deals with large messages or files as single continuous entities, which appear to be completely contained within the terminal. The user never has to consciously "send" an editing change to the host; he simply makes the change, just as he would on paper.

The terminal's most obvious limitation is the small size of the screen (24 lines of 80 characters), compared to the full page printed on paper that users are used to seeing. This is particularly noticeable when operating in split screen mode looking at two objects. This limitation, however, points out what we consider to be a significant technical contribution of the MME terminal.

By dealing in terms of windows and domains, and letting the terminal format the text and control the flow of data (via Vacancy and Scroll notices) we have achieved a division of labor which makes the application program almost completely independent of the terminal characteristics. The high-level protocol that achieves this also supports the natural user interface style we sought. The application program has no knowledge of, or concern for, the amount of memory in the terminal. Since the terminal manages its own memory and tells the host when and how much data to send, the application program never needs to consider whether the text “will fit.”

Furthermore, the application program assumes very little about the size or characteristics of the display screen. A small section of code in SIGMA, which controls the mapping of windows to screen lines, knows that the terminal has 24 lines. It has no idea how many characters fit on a line, or whether the terminal has proportional spacing, or what kind of display technology is employed.

The application program is also completely unconcerned about details of the terminal's editing facilities. For instance, SIGMA does not know the terminal does not have a "type-over" mode, or that it does have a "word delete" key. The terminal could scroll a line at a time or a page at a time. Adding a positioning device like a "mouse" or a joystick would not affect SIGMA at all. In theory, one could add a "replace string" function internal to the terminal if he wanted without affecting the application program (we chose not to because we feel such operators should be global to the entire document and therefore belong in the host).

The protocol isolates the physical and functional features of the terminal from the application program, which allows us to take advantage of new, more capable terminals with more memory, larger screens and more powerful editing features without having to rewrite the application software. At the same time we maintain the desired interactive coupling between the terminal and the host.

We believe a protocol which provides this independence is needed to foster the use of "intelligent" terminals for network applications. We are advocating the development
of a protocol to support communication with such powerful terminals, which we are calling the Network Virtual Processing Terminal protocol (NVPT). We do not propose that the MME terminal protocol in its current form is sufficient for NVPT, since it does not yet provide adequate format or editing controls. It needs to accommodate a variable number of display windows and visible screen lines, and issues such as backward compatibility with teletypes and down-load capabilities must be considered. We do feel, however, that the protocol in MME provides a good starting point for discussion of an NVPT and we invite constructive criticism directed toward achieving such a goal.

REFERENCES


This research was performed for the Advanced Research Projects Agency under Contract No. DAHC 15 72 C 0308, ARPA Order No. 2223. The views and conclusions expressed in this paper are not necessarily those of any person or organization except the author(s).