THE IMPLEMENTATION OF NLS ON A MINICOMPUTER

BY JAMES G. MITCHELL

09-73

XEROX

PALO ALTO RESEARCH CENTER

THE IMPLEMENTATION OF
NLS ON A MINICOMPUTER

BY JAMES G. MITCHELL

CSL 73-3   AUGUST 1973   NIC #18941

This technical report covers the research performed at Xerox Palo Alto Research Center (PARC) for the period June 30, 1972 to July 1, 1973 under Contract Number DAHC15 72 C 0223 with the Advanced Research Projects Agency, Information Processing Techniques Office.   The research covers initial studies and evaluation of transferring a large, display-oriented documentation system (the NLS system developed at Stanford Research Institute) to a minicomputer system and a protocol for accessing NLS over the ARPANET.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

XEROX

PALO ALTO RESEARCH CENTER
3180 PORTER DRIVE/PALO ALTO/CALIFORNIA 94304

Xerox Palo Alto Research Center (PARC):

PARC is a research facility of the Xerox Corporation. It is involved in research in the physical sciences, computer hardware and software systems and the computer sciences in general. These latter include computer architecture, interactive graphics, operating systems, programming research, and natural language understanding systems. The Computer Science Laboratory (CSL) of PARC currently numbers thirty technical and support staff.

INTRODUCTION

The ARPA contract reported herein concerns the development of an NLS-like [1] display-oriented text-editing system supporting multiple users on a minicomputer configuration. The research also aims to use this development as a source of suggestions and a gauge for a set of software engineering techniques which are intended for large and complex software systems.

The originally proposed (two year) research plan included the following major steps:

(1) A reimplementation (including redesign) of the TENEX-based NLS system in the Modular Programming System (MPS)--MPS is an ongoing, separate research program which has been developed collaboratively by PARC and SRI-ARC.

(2) Bootstrapping of MPS to a suitable minicomputer;

(3) Development of a mini operating system (mini-OS) to support multiple MPS programs, a file system and a virtual memory (probably with little or no assistance from any memory mapping hardware);

(4) Transfer of NLS (written in MPS) from a PDP-10 to a minicomputer by recompilation--with as little rewriting as possible.

(5) "Tuning" of the experimental mini NLS (xNLS) to support as many users as possible.

The first year of this plan has primarily dealt with the following specific tasks:

(a) initial evaluations of the problems involved in putting a large multi-user system like NLS on a small computer and specification of suitable minicomputer hardware,

(b) redesign of NLS to

   (1) take advantage of SRI-ARC's considerable experience with the system;
   (2) use and evaluate the features of MPS for large systems;
   (3) provide for an NLS which would be amenable to being distributed over the ARPANET (e.g., with display "front ends" at remote locations),

(c) development of an ARPANET protocol for providing access to NLS over the network from suitable display terminals.

Large Systems on Small Machines:

Although the exact forms which computer hardware may take in the future are hard to predict, there are a number of trends which are evident:

(1) processor speed on the order of 1 microsecond per instruction is a fact;

(2) the cost of digital logic has dropped by a factor of two every five years;

(3) memory determines the cost of a minicomputer to a much greater extent than its CPU.

These trends must interact with some observed facts about large, interactive systems such as NLS:

(1) The body of code for the system is large: NLS is composed of a number of well-integrated, sophisticated facilities each of which is in itself fairly complex;

(2) the large subsystems of NLS interact only infrequently (relative to the interactions within a subsystem) and over fairly narrow interfaces;

(3) when the user is specifying the next action to be performed by NLS, very little state information and processing are necessary to respond to him, but that response must be immediate. (Character echoing or command recognition and feedback are examples.)

These two sets of observations have led to some design decisions for xNLS. Since most minicomputers are about as fast at manipulating text as a PDP-10 (the machine on which NLS is currently implemented), the size of the xNLS software and the size of the memory on the minicomputer will be the hardest constraint on system performance. There are a number of means by which this constraint can be eased:

(1) Implement only the most frequently used parts of NLS on the mini (e.g. its text-editing facilities) and access other system functions such as hard-copy output formatting, automatic document distribution and classification by using the parent NLS over the ARPANET;

(2) Provide for a virtual memory underlying MPS but lock the portion of a user's process into memory which is required for quick

4

response during user input; this saves swapping overhead without wasting much primary memory since the state needed is usually very small (certainly less than 100 words). An underlying segmentation scheme for automatic overlaying of running MPS programs is described in [6].

(3)   Obtain minicomputer hardware which will help to implement a virtual memory: this includes some memory mapping hardware and a reasonably large, high-speed swapping device.

(4)   Since computing speed is much less a constraint than memory, a trade can be made: compile MPS programs into a very compact interpretive code and run as much of NLS as possible in this way; some recent work by Deutsh [2] indicates that such a technique can compact programs by a factor of 2 to 5. This should lower the swapping rate accordingly (unless the interpretive slowdown backs up the system and artificially causes thrashing).

A Minicomputer Configuration for xNLS:

The above hardware and software considerations led to the selection of the following minicomputer configuration:

A 16-bit, 800 nanosecond CPU

A 64k 16-bit, 800 nanosecond memory

A simple mapping device for accessing memory in pages

A high speed, 1.25 Mword, fixed head disk (expandable to 2.5 Mwords); its transfer rate is 275,000 words/second and it operates over a DMA channel

The original hardware mapping device design was a simple addition to a Data General NOVA 800 with 4k memory boards and operated on 4k word pages. It was designed by E. McCreight of PARC and required about 15 standard integrated circuits. Since then, minicomputer manufacturers have provided their own memory mapping hardware and main memories as large as 128k words.

The fixed head disk, main memory, and memory mapping device are all attempts to overkill the memory constraint discussed earlier. Such a system can be purchased commercially for approximately $50k. If it could support ten NLS users, then the cost per user, including a reasonable terminal ($3000 to $5000) would be $8000 to $10000; if it could only support five users, this range would be raised to $13000 to $15000. The trends in hardware may very well indicate that it would be better to build

5

single user systems (one man, one computer) if the cost of the multi-user system is this large.

What problems would go away if the system were to serve only a single user? Hardly any. The size of NLS would diminish not at all, and although the entire bandwidth of the swapping channel would be available to a single user, the size of main memory would have to be smaller (because of the price ceiling of $10k to $15k) and the swapping rate would therefore be higher. Computing power, however, would be a freer resource than under a multi-user system. Other than this, all the problems of a large system still remain: a file system, memory management and a mini-OS would still be necessary, and the system would still have to be maintained and changed over time. Thus, the multi-user requirement for xNLS could be elided with almost no alteration to the remainder of the project: the results pertinent to large systems on small machines and to a redesign of NLS would not be affected.

File Storage for xNLS:

Long term file storage in xNLS can be accomplished in a number of ways without requiring a large disk system and its concomitant management overhead. Frequently accessed, small to medium sized files (0 to 20 page documents) can be kept on the swapping disk along with system programs and working storage without exhausting the storage. Larger files which are accessed often could be stored at some ARPANET site with a suitable file facility. Such a file could either be transmitted in its entirety to the xNLS site when needed or "paged" over the network. In the latter case, xNLS would request a given page at most once, but might transmit "dirtied" (altered) pages back to the file source many times. Lastly, files might be kept on removable media (such as tape cassettes, tape cartridges, or floppy disks) and transferred to the swapping disk when needed. This could be especially useful in a single-user system. The network "paging" technique can also be used with some removable media, both for speed and for minimizing work loss resulting from xNLS system failures.

The NLS Protocol:

The NLS Display is divided into a number of rectangular display areas
(DAs). A DA may contain a number of STRINGS at a selected set of (X,Y)
positions in the DA. A DA is named by a DAID, which is a 9-bit number.
The size of a DA is given by two parameters: NSTRS, the height (in units
of lines) and NCHARS, the maximum number of characters in a line in the
display area. Other attributes of a DA include the coordinates of its
lower left-hand corner, called (XBOT,YBOT), the separation between lines,
called DY (the units of DY, X and Y are a function of a particular display
device), and the width of a character, called HINC (also in the units of
the display device). A typical DA as used on an IMLAC PDS-1 display
computer [3] is illustrated in figure 1 with its DA parameters illustrated
(this NLS protocol has been implemented for IMLAC displays and is used at a
number of ARPA sites). Characters as used on the IMLAC are 7x9 points with
2 points of horizontal spacing between characters and 9 points between
lines. This DA may hold at most NSTRSxNCHARS=30x73=2190 characters.
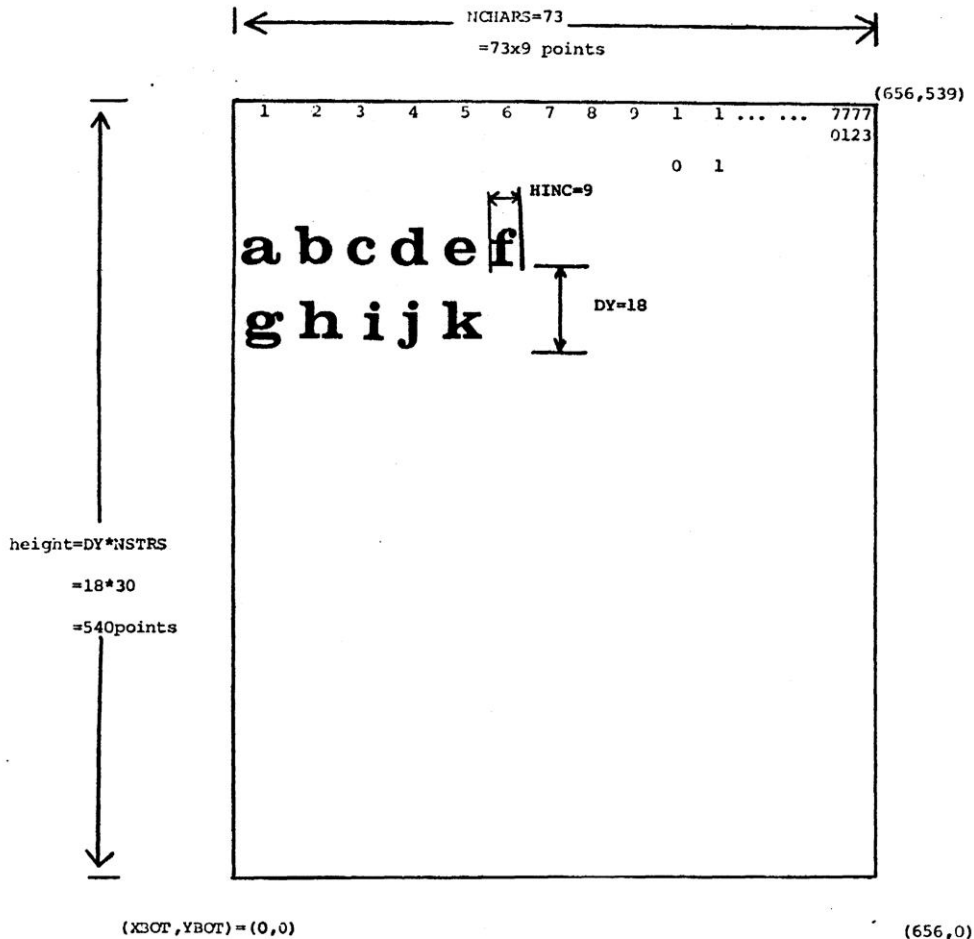

Figure 1 Parameters for a Typical DA

7

DAs may be created and destroyed dynamically. When one is created, it is given an integer "name" called a DAID by the host computer (i.e. not by the display processor which is obeying the protocol) which is used in all operations relevant to DAs. Those operations are:

    Create DA
    Destroy DA
    Suppress DA (stop displaying all strings therein)
    Suppress and clear DA
    Restore DA (cause strings in it to be displayed)

DAs may overlap, subject to some rules about which strings will show in the regions of overlap. A feasible layout of DAs on a display screen is shown in figure 2. Note that DAs may overlap to any extent, and that there is no requirement that the entire display screen be covered by DAs.
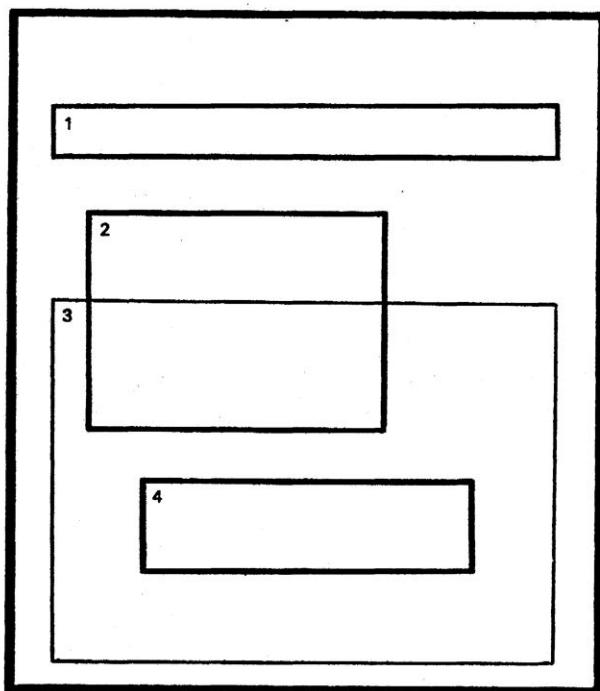

Figure 2 Sample Layout of DA's on a Display Screen

8

The only entities which may be put into a DA are strings: a string is a sequence of 7-bit ASCII bytes. A string has a name called an STRID (a number in the range [0,177B] -- "B" suffixed to a number indicates octal notation) which is local to the DA in which it lives. This name is assigned by the host computer when the string is initially transmitted.

The manipulations which may be done to strings are

(a) New string to position (X,Y) in DAID with given STRID
(b) Suppress display of a string in a DA
(c) Restore display of a string in a DA
(d) Suppress displaying of a string and delete it from the DA

Command (a) may also be used to move a string from one position to another without retransmitting the entire string: this is a great economizer of bandwidth.

If two DAs overlap, the strings in both will be displayed unless either one or the other DA is suppressed or strings in the overlap areas of the DAs are selectively suppressed. Overlapping DA's are used by NLS for displaying user-requested status information by selectively suppressing the strings in one DA in order to display the necessary information in an overlying DA. When the user no longer needs the information, the overlying DA is completely deleted and the suppressed strings in the underlying one are restored. This is especially important for fully utilizing the display surface area without permanently allocating DAs for infrequently required information.

Sequential Display Areas:

The NLS protocol allows the display processor to act as a simple teletype so that the normal typewriter communications methods in many systems may be used without change. A DA which can accept such communication is called a sequential display area (SDA), and there is assumed to be one distinguished SDA which simulates normal typewriter output for the system which supports NLS (e.g. the TENEX system used at SRI-ARC). This SDA, hereafter called the TTYSDA must be the initial, "current" DA when communication between the NLS host and the display is initiated.

The distinguishing features of an SDA (as contrasted with a DA) are as follows:

(a) The last character in a line may be deleted; in a normal DA the minimum entity which can be manipulated is an entire string; thus, to delete the last character in a line in a DA, the entire line minus its last character must be transmitted to replace itself.

(b)   The lines in an SDA are automatically scrolled up
       one line when the SDA is full and a new line is
       added at the bottom

(c)   An SDA is specified just like a DA. The display
       processor, however, interprets some characters
       whose destination is an SDA in a special way:
       carriage-return (015B) and line-feed (012B) cause
       actions comparable to those evoked on a typewriter
       terminal. Carriage-return means that the next
       character to the SDA will be "written" at the
       beginning of the current line; line-feed increments
       the pointer to the current line or causes the lines
       in the SDA to "scroll" up one line (in the case
       that the SDA is full when the line-feed character
       is received).

(d)   When an "unescorted character" is received by the
       display processor, it is always placed in the
       TTYSDA. It may also be written into one or two
       extra "default" SDAs if they have been so enabled.

There are a number of protocol commands which are applicable to the TTYSDA:

(a)   Enable TTYSDA and suppress the display of all other
       DAs. (the display is then said to be in "TTY
       mode")

(b)   Suppress TTYSDA and restore the display of any DAs
       that were suppressed by the last "Enable TTYSDA".
       (the display is then in "Display mode")

(c)   Unescorted characters always go to the TTYSDA and
       any associated default SDAs even if the TTYSDA is
       not enabled. Moreover, if the TTYSDA is
       suppressed, the default SDAs need not be. This is
       useful for providing a small "TTY window" even when
       the TTYSDA is suppressed.

Terminal Input:

The input devices for NLS include a typewriter keyboard, a pointing device
called a mouse (with three push buttons B2, B1 and B0 on it) and a device
called a keyset (or, more descriptively, a "five-finger chord keyset").
These are all described fully in [1]. The display processor must
"continuously" display a string on the screen at the position determined by
(x,y) coordinates read from the mouse. This string (usually a single
character) is called the cursor and "tracks" the mouse. The five-finger
keyset acts like a replacement for the keyboard, but in order to key more

than 31 possible characters, two of the three buttons on the mouse (B2 and B1) are used as "case-shift" buttons in conjunction with it. Appendix B gives the keyset/ASCII correspondences for (B2, B1)=(0,0), (0,1) and (1,0).

In TTY mode the display will send a single ASCII character per keystroke (keyboard or keyset "chord") to the NLS host; this is called "short character mode" (SC-mode).

The display processor can also operate in a "long character mode" (LC-mode): this provides mouse coordinates to the NLS host when necessary. Short, unescorted ASCII characters are still sent to the NLS host when the display is in LC-mode under the following circumstances:

(S1) the user input is a character from the keyboard and is not an ASCII control character (i.e. is not in the range [0,37B]);

(S2) the input is a single character "chord" from the five-finger keyset.

Some translation is done on keyset characters. If k is the 5-bit number corresponding to a keyset chord then the character sent to the NLS host is

(a) (k+140B) if k is in the range [1,32B] (i.e. lowercase alphabetic)

(b) otherwise, a chord in the range [33B,37B] is translated according to the following table:

k	Character Sent	Printed form
33B	54B	,
34B	56B	.
35B	73B	;
36B	77B	?
37B	40B	SPACE

Long characters are sent to the NLS host in the following circumstances:

(L1) the state of one or more mouse buttons changes (either because one or more are depressed or released);

(L2) an ASCII control character is typed on the keyboard.

The formats of long characters are

(a) in case (L1), a seven byte string is sent to the NLS host:

BMSG 45B (buttons+100B) XX YY

(Where BMSG = 034B).

The state of the buttons B2, B1 and B0 are encoded into a single octal digit with B2 as its most significant bit. XX and YY are each two bytes long; they represent 12-bit numbers (6 bits in each 8-bit byte) and are the coordinates of the mouse at the time the mouse buttonb changed.

(b)   case (L2) has two subcases. If the control character is one of the NLS special control characters (↑B [Center Dot], ↑D (Command Accept), or ↑X [Command Delete]),character sent is

      BMSG 45B (character+140B) XX YY

which is similar to that sent for case (L1). Otherwise, the long character has the form

      BMSG 41B (character+140B)

Thus, mouse coordinates only accompany mouse button changes or NLS control characters: their presence is indicated by the 45B following the ESC character.

In order to provide a reasonable interface between the characters sent by the display and the NLS system, the current NLS host, TENEX, performs some transformations on the characters received by it before they are made available to NLS. These basically make coordinates available "on request" to NLS and integrate sequential mouse button changes (so that simultaneous actuations are not necessary) before presenting the changed state to NLS. Also, mouse button states and characters are only obtained by direct request from NLS (and therefore, not asynchronously).

This way of sending mouse coordinates to the NLS host may seem somewhat wasteful of bandwidth; however, it guarantees that the coordinates for a pointing operation by the user will be the ones he specified when he pressed button B0 (called the select button or "Command Accept") on the mouse. If this were not done, and NLS had to request mouse coordinates after deciding that it needed them, (random) delays in NLS's response time would cause it to acquire coordinates corresponding to where the user was pointing when NLS made the request rather than the ones available when he made the pointing selection; this would be a totally unacceptable situation. Short character or long character mode is set by the NLS host's sending a suitable command to the display; it is not automatically associated with TTY mode or display mode, which have to do with the messages sent to the display from NLS.

The cursor string which is displayed as the mouse is tracked is specified
by the NLS host and can be dynamically changed: NLS uses this feature to
indicate when the mouse can be used to select something on the screen (the
cursor is displayed as a "↑") and when it cannot (a "+" is displayed).

NLS allows two separate displays to share a common view so that two people
may view the same information even if they are remote from one another.
This includes displaying each person's mouse cursor on the other person's
screen so that they can each refer to objects on the other's display. To
accomplish this, the NLS host can request that the display processor send
mouse coordinates "continuously" so that they may be displayed on the
remote screen. Actually, the NLS host may only request that the mouse
coordinates be sent every DT milliseconds if either coordinate has changed
by an amount greater than DXY (the absolute minimum DT is 6/b where
b=bandwidth of transmission line in bytes/sec, the value "6" comes from the
fact that the coordinates are sent as the six byte stream

$$BMSG \ 44B \ X \ X \ Y \ Y,$$

which is analogous to long characters as described above. For the ARPANET,
this minimum is 20 milliseconds because all messages shorter than 1000 bits
use 1000 bits and the network (burst) bandwidth is 50000 bits/second).
Thus, DT=20msec and DXY=1, is the finest grain allowed for continuous
tracking of the remote cursor by the NLS host. The host can, of course,
also request that the continuous transmission of mouse coordinates be
turned off. The specific formats of the commands for doing this are given
in detail in Appendix A.

The NLS Redesign

A redesign of NLS has begun under the auspices of SRI-ARC with participation by Xerox PARC. The motivations for this effort have been outlined earlier in this report; they are also given in detail in "A Look at the MPS Conversion" [4] which describes the areas of NLS requiring redesign and the problems of the current NLS implementation. Thus far, the design has concentrated on the parts of NLS which are associated with the user interface: display control and feedback, user command specification and the command language facilities themselves.

The system components which deal with the hardware realities of a given display have been segregated from the "ideal" (or at least, normalized) display to which NLS interfaces. This has used the NLS protocol described in Section II as a starting point and the design adopts the view that the display is always accessed via an interface like the ARPANET protocol. Indeed, one of the design "meta-decisions" for the redesigned NLS (NLS*) has been that the system should be severable at a number of places in order to facilitate a distributed implementation, with as much (or as little) front-end processing done remotely as is desirable. This aims partly at the parent NLS to xNLS relationship discussed in Section I, and partly at cleanliness and modularity within NLS itself. The issue of distributive modularity has also influenced the control structures of MPS in that it demonstrated that some form of parallel control mechanism was essential if NLS* were to be implemented in this way. The capability for parallel control in turn suggested that "display correction" (the problem of guaranteeing that what is displayed reflects accurately changes which occur in the document being edited) might best be handled as a pseudo-parallel process which reacts to "alteration events" which may affect the validity of the display contents. Since the contents of an NLS display include such disparate items as the status of the current command specification, the time and date, various parameters indicating the "view" of the document as specified by the user and a portion of the document itself, this approach nicely eliminates the very complex scheduling of display alterations resulting from such "unconnected" events. A primary result of this view is that the rapidly changing command status indications, the displayed time, and the document display are all seen as display feedback and may be handled in one consistent manner.

Experience with NLS in the ARPANET community has spotlighted the need for a flexible command language facility for accommodating everyone from the occasional user to the hard-core sophisticate and for allowing users to tailor the system to their individual tastes and tasks. NLS* will replace the rather built-in command recognizer of NLS with a more flexible system facility consisting of a data structure which describes a command language

and a number of modules which use that specification. The modules perform the following duties:

(1) Interpretation of a complete command to invoke the facilities of NLS: separating this function out provides part of the framework for a user-programmable NLS.

(2) Recognition and feedback during user command specification: this helps accommodate various classes of users; beginners can get maximal feedback about commands and their options and even be protected from "accidentally encountering" overly-complex commands; sophisticated users can choose a recognition algorithm which minimizes keypushes and still specify a feedback regime which gives adequate information about their interactions with the system.

(3) Language definition and alteration: new commands can be added easily or a user can tailor a command language to his requirements by macro definition or by altering the data structure which defines the language.

The need for many of these facilities has been so keenly felt that they are being added to the current NLS; they are described in [5].

The primary data domain for NLS is text; simple graphics, mixed text and graphics and other structures would be extremely useful. In order to facilitate the addition of new types of data to the NLS world, NLS* will provide a single interfacing method for all data types. A datum which is to be manipulated by NLS will carry a tag specifying its type. This tag is used to select a specific action from an action class for a given operation. For instance, if the datum is to be displayed, the display action for that data type is invoked with a suitable handle on the datum and information about a display area in which it should be formatted. If the datum has the type tag "mixed text and graphics" (MTG) then the MTG display routine is invoked for it. Because of this approach, this routine can act as overlord for the text and graphics display routines and subdivide the virtual display area given to him among the components of the MTG datum. This approach appears very promising for combining primitive data types into more complex structures without duplicating the facilities for each subtype simply because they are being used in combination.

Appendix A: NLS Protocol Command Formats

This Appendix gives the protocol specifications in detail. Each command is
headed by a phrase which identifies it, and the command is defined in a
BNF-like notation; square brackets denote optional items, slashes divide
alternatives, and parentheses are used to group alternatives. A "$"
preceding an element of a rule means "zero or more occurrences of".
Numbers which are suffixed with a "B" represent octal numbers and occupy no
more than seven bits, i.e. they fit in one transmitted byte. Any variables
which can only be 0 or 1 occupy one bit and are usually packed together
into a single byte.

Each command arrives as a string of characters at the display, which must
interpret the first two characters of the command in real time. The
remaining characters in a message may be buffered until the entire message
is received and then interpreted. There is, however, no requirement in the
protocol that the entire message may not be interpreted as it arrives.
There is no checksum or other error check as part of the command stream.

```
    command ::=     UnescortedChar/ESC CNT CommandTail;
        ESC ::= 33B; -- the ESC or ALTMODE (ASCII) character identifies   a
                      command
    UnescortedChar ::=   -- any 7-bit character except ESC --;
```

Unescorted characters are placed sequentially in the TTYSDA and in any
default SDAs.

```
        CNT ::=   --  a number in the range [40B, 177B] --
    CNT is the (count of the number of bytes which are to follow in the
    message)+40B; this offset notation guarantees that the count cannot
    look like 33B (ESC) or any control character; hence commands and
    unescorted characters are readily distinguishable.
```

Command tail ::=

ada /	--	Allocate a Display Area
dda /	--	Deallocate a Display Area
sdda /	--	Suppress Display of a Display Area
rdda /	--	Restore Display of a Display Area
strda/	--	STRing to a DA
ssda /	--	Suppress String in DA
rsda /	--	Restore String in DA
apsda/	--	Append String to Sequential DA (SDA)
echda/	--	Specify SDA(s) for Unescorted Characters
tsnda/	--	Teletype Simulation ON
tsfda/	--	Teletype Simulation OFF

16

```
scm    /     --    Short Character Mode
lcm    /     --    Long Character Mode

scsr   /     --    Specify Cursor String
ccnda/       --    Continuous Coordinate Transmission ON
ccfda;       --    Continuous Coordinate Transmission OFF
```

Allocate a Display Area:

```
    ada ::= 16B DAID NSTRS NCHARS ATXY DY CSIZE HINC FONT;
    DAID ::= LONGNUMBER; -- a DAID is a 9-bit number which names a DA.
```
A LONGNUMBER is sent as two 8 bit bytes; if B1 and B0 are the high and low order bytes (they arrive in the order B1, B0) then the value v of a LONGNUMBER is

$$v= 2\uparrow6 * ((B1 \bmod 2\uparrow6)-40B) + ((B0 \bmod 2\uparrow6)-40B)$$

thus, a LONGNUMBER is limited to the range [0,4095].

```
    NSTRS ::= SHORTNUMBER;
```

A short number occupies the low order 7 bits of a bit and is therefore restricted to the range [0, 177B]. NSTRS defines the number of lines in the display area (i.e. its height).

```
    NCHARS ::= SHORTNUMBER; -- lines sent to this DA which contain more
                               than NCHARS characters are to be broken into
                               multiple lines.
    ATXY ::= XBOT YBOT; -- the coordinates of the lower left-hand corner of
                           the DA.
    XBOT ::= LONGNUMBER;
    YBOT ::= LONGNUMBER;
    DY ::= LONGNUMBER; -- this is the height of a line in terms of basic
                          display increments, and includes any blank space
                          between lines.
    CSIZE ::= 0/1/2/3;
```

Specifies the default character size for the DA. CSIZE is a mapping value and all that is required is that characters of size i be greater than j-sized characters (i,j=0,1,2,3) for i>j.

```
    HINC ::= SHORTNUMBER; -- specifies the width of characters in terms of
                             basic display increments.  Hence, fixed pitch fonts
                             (all characters occupying equal width areas) are
                             assumed.  This includes the space between
                             characters.
```

```
    FONT ::= 0 0 FOV FUL FBO FIT FBL; -- this is one byte composed of the
```
one-bit switches given. FONT, CSIZE and HINC specify default values for strings displayed in the DA.

The switches for FONT are defined as

```
FOV ::= 0/1;  -- 1=> overline displayed characters
FUL ::= 0/1;  -- 1=> underline displayed characters
FBO ::= 0/1;  -- 1=> display characters in boldface type
FIT ::= 0/1;  -- 1=> display characters in italic type
FBL ::= 0/1;  -- 1=> blink displayed characters on and off
```

Deallocate a Display Area:

```
dda ::= 2 DAID;
```

The DA identified by DAID is deleted; it need not be empty when this command is sent: any strings in it will automatically be deleted first.

Suppress Display of a DA:

```
sdda ::= 6 DAID KILLFLAG;
```

The identified DA is turned off (none of its contents are displayed). Additionally, if KILLFLAG =1, its contents are destroyed.

```
KILLFLAG= 0/1;--1=> delete; 0=> don't delete.
```

Restore Display of a DA:

```
rdda ::= 7 DAID;
```

The contents of a previously suppressed DA are displayed (unless the previous sdda also deleted them, of course).

String Manipulation in a DA:

```
strda ::= 20B DAID STRID SLNGTH FRMT [ATXY] [HINC] [FONT] STRING;
```

This command is used to introduce a new string, replace an existing string, or move an existing string from one position in a DA to another.

```
STRID ::= SHORTNUMBER;
```

STRID identifies the string; if it is a new string, it is given the value of STRID as its name; otherwise STRID is used to identify which string in the DA is to be manipulated. The value of STRID is independent of the (X,Y) position of the string.

```
SLNGTH ::= SHORTNUMBER;
```

SLNGTH=0 means operate on the current string with name STRID; the actual string, STRING, is not present in the command. SLNGTH=1 means "pretend STRING is the null string"; STRING is not present in this case either. SLNGTH>1 means "make the string STRING which is appended to this command be the one with name STRID". If there already is a string with name STRID, it will be replaced by the STRING in the command.

```
FRMT ::= FORMATS + 40B; -- a single byte
FORMATS ::= FDS FDD IDS IDD SDS SDD XYDS;
```

FORMATS is a string of 7 bit switches which is sent in a single byte (offset by 40B).

```
FDS ::= 0/1; -- 1=> use the value of FONT switches from the previous
```
instance of STRID for the new instance;

```
FDD ::= 0/1; -- 1=> use the DA default value for FONT switch values.
```

FDS=FDD=0 means that FONT is present in the command.

```
IDS ::= 0/1;
IDD ::= 0/1;
```

IDS and IDD behave for HINC as FDS and FDD do for FONT

```
SDS ::= 0/1;
SDD ::= 0/1;
```

SDS and SDD behave for CSIZE as FDS and FDD do for FONT.

```
XYDS ::= 0/1; -- 1=> use STRID's current coordinates.
```

XYDS=0 means ATXY is present in the command.

STRING ::= $(SHORTNUMBER); -- 0 or more 7-bit bytes (ASCII) which are the contents of the string if SLNGTH>1.

Suppress String in DA:

```
ssda ::= 10B DAID STRID KILLFLAG;
```

Suppress the display of the named string in the given DA; if KILLFLAG=1, also delete the string from the DA. Altering the display status of individual strings is overridden by the display status of the DA; if the DA is suppressed, any displayed strings in it are suppressed also until the DA is again turned on: i.e., the display status of each string is remembered independently of whether the DA is suppressed.

Restore String in DA:

    rsda ::= 11B DAID STRID;

Restore the named string in the given DA; whenever the DA is displayed, the string will be displayed.

Append String to Sequential DA:

    apsda ::= 17B DAID STRING

The initial CNT field in the command header contains LENGTH(STRING)+3 to account for the characters 17B and the DAID (which is a LONGNUMBER). The string supplied is appended to the sequential display area. Carriage-return and linefeed must be simulated by the display to behave as they do for a teletype. Strings longer than the NCHARS parameter for the DA (see the "ada" command) are broken into an appropriate number of lines. A received ↑A (001B) or ↑H (010B) character causes the last character in the SDA to be deleted (it should disappear from the screen so that the next character sent to the SDA will occupy the same screen position as the character just deleted. ↑W (027B) must delete any invisible characters from the current SDA position to the left until a visible character is encountered and then all the visibles from there up to (but not including) the next invisible character. The set of invisible characters are carriage-return (↑M=015B), space (040B), and tab (↑I=011B); visible characters are all the normal, printable ASCII characters exclusive of these invisibles. ↑G (007B) should simulate the bell on a standard TTY; if there is no "aural" indicator, the display should produce a bell symbol (e.g. the word "BELL!") which is overwritten by the next character displayed.

Specify DA(s) for Receiving Unescorted Characters:

    echda   ::= 3 FLAGS [DAID1] [DAID2];

    FLAGS   ::= 0 AD SD A1 S1 A2 S2; -- seven one-bit fields in a
    single byte.

    AD=1 means unescorted characters go to TTYSDA; if SD=1, they will not.

    A1=1 means unescorted characters should go to the DA identified by
    DAID1; S1 means they should not; A1=S1=0 means DAID1 is not present.

    A2 and S2 mean the same for DAID2 as A1 and S1 do for DAID1.

Unescorted characters, therefore, can go to three SDAs simultaneously if AD=A1=A2=1. Copies of the characters are made by the display and placed in each SDA.

Teletype Simulation On:

    tsnda ::= 12B;

Turn on the current TTYSDA and suppress any other DAs which are currently on.

Teletype Simulation Off:

    tsfda ::= 13B;

Suppress the current TTYSDA and restore the DAs suppressed by the most recent tsnda command.

Short Character Mode:

    scm ::= 15B

Send only short characters to the NLS host (characters like +X, +D, etc. will be sent just as they would from a teletype).

Long Character Mode:

    lcm ::= 14B;

Send long characters to the NLS host when appropriate. (See the description of long characters in Section 11 "The NLS Protocol".)

Specify Cursor String for Tracking the Mouse:

    scsr ::= 05B SLNGTH CSIZE HINC FONT STRING

The given string is to be displayed as the cursor which tracks the mouse (or other pointing device) on the screen. A detailed description of the parameters SLNGTH, CSIZE, HINC and FONT are given under the commands ada (Allocate Display Area) and strda (String to DA). For the scsr command, SLNGTH (=LENGTH(STRING)+1) must be greater than 1.

Continuous Coordinate Transmission On:

    ccnda ::= 21B DT DXY;

Mouse coordinates are to be sent to the NLS host every DT milliseconds, if either its x or its y coordinate has changed by more than DXY

```
DT  ::= LONGNUMBER;
DXY ::= LONGNUMBER;
```

Mouse coordinates, as sent to the NLS host, have the form

```
coord ::= BMSG 44B ATXY;
BMSG = 34B;
ATXY ::= XCOORD YCOORD;
XCOORD ::= LONGNUMBER;
YCOORD ::= LONGNUMBER;
```

Continuous Coordinate Transmission Off:

```
ccfda  ::= 22B;
```

Stop sending mouse coordinates continuously.

Appendix B: Keyset Chord/Character Correspondences

The mouse buttons are given as B2, B1, B0 and a
"1" indicates the corresponding button is depressed.

Mouse and Keyset, Codes and Cases

Mouse Buttons:	000	010	100
Keyset Chord			
00001	a	A	!
00010	b	B	"
00011	c	C	#
00100	d	D	$
00101	e	E	%
00110	f	F	&
00111	g	G	'
01000	h	H	(
01001	i	I	)
01010	j	J	@
01011	k	K	+
01100	l	L	-
01101	m	M	*
01110	n	N	/
01111	o	O	↑
10000	p	P	0
10001	q	Q	1
10010	r	R	2
10011	s	S	3
10100	t	T	4
10101	u	U	5
10110	v	V	6
10111	w	W	7
11000	x	X	8
11001	y	Y	9
11010	z	Z	=
11011	,	<	[
11100	.	>	]
11101	;	:	←
11110	?	\	ESC
11111	SP	TAB	CR

References

[1]   "Online Team Environment - Network Information Center and Computer
        Augmented Team Interactions", Augmentation Research Center,
        Stanford Research Institute, NIC #13041.

[2]   Deutsch, L. P., "A LISP Machine with Very Compact Programs", in
        Proceedings of the Third International Joint Conference on
        Artificial Intelligence, Aug. 1973, pp. 697-703.

[3]   "User's Reference Manual, IMLAC PDS-1 Programmable Display
        System", IMLAC Corp., 1970.

[4]   Dornbush, C. F., Irby, C. H., Mitchell, J. G., "A Look at the MPS
        Conversion", NIC #15376.

[5]   Irby, C. H., "Proposed Changes in the NLS Command Language",
        August, 1973, NIC #18408.

[6]   Mitchell, J.G., "MPS Segmentation System", MPS Group internal
        document, Jan. 1972, available currently as NIC #19734.