A software environment for problem-oriented systems

ANDRES C. POLIT

Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

The command language concept, which is used in some large time-sharing systems, has been adapted to construct single-user systems with real-time capabilities. These systems share a common core, whose structure is sketched by a description of three implemented systems. The modularity and generality of much of the core makes it possible to use the same code in systems with widely differing objectives.

A look at the history of the use of computers in behavioral, psychological, and physiological research will show a convergent evolution of software and hardware concepts, with recent developments favoring the systems approach in which the flexibility of the computer and the requirements of the experiment are molded into a problem-oriented system (Ossanna, 1972; Reigll, Faber, & Fisher, 1972; Stacy & Waxman, 1965). In a university environment, where a small computer might be shared by several Es, the constraints of time and money on any program of automation are sufficient in themselves to motivate attempts to characterize the software environment that would give a system flexibility without compromising performance.

In the case where a computer is shared by many Es, what constitutes a system is somewhat arbitrary. Consider Fig. 1, which is a schematic view of a hardware environment. It consists of a core set of hardware and experiment-specific hardware. For our purposes, a hardware system is defined by a subset of the core hardware and the experiment-specific hardware. Figure 2 is a schematic of our hardware environment, which is typical for data acquisition and control applications.

Similarly, a software system consists of a subset of a core, which is composed of the manufacturer-supplied monitor with an added software "front end," problem-specific routines, and data structures. A problem-oriented system is the union of the hardware and software systems.

From a computer scientist's point of view, one of the main things that a problem-oriented system allows a user to do is to manipulate data within a prestructured data base. Given the objectives of the system, good design will then suggest a structure for the data base which permits economical use of the system's resources without restricting the user. In designing our systems, we have noticed that there is an interaction between the data base structure and the design of the routines comprising the system. From another point of view, a software environment, in conjunction with system objectives, helps define "natural" data structures. This reduces the amount of time required to implement a system.

THE COMMAND CONCEPT

In recent years, the concept of having languages that are tailor-made to solve a particular class of problems has gained wide acceptance as a means of increasing the accessibility of the computer to scientists. Generally, these languages are designed for off-line use or else require a compilation step, rather than being on-line and interpretive. Some large time-sharing systems make the resources of the system available to the user via commands. In the Multics system, for example, commands can be entered in on a console or read from a file (Corbató & Vyssotysky, 1965). By allowing several commands to be typed on a line and allowing labels and branch commands, the MULTICS commands form a

Fig. 1. A hardware environment with three systems. Each system is made up of a subset of a core plus experiment-specific hardware.

Fig. 2. The makeup of a hardware core for real-time data acquisition and control systems.

simple language. This command language concept has been used to construct single-user systems.

To the user, each of our systems appears to have an on-line interpretive language especially designed for his application. The language is composed of a set of instructions, or commands, with a very simple syntax. As in Multics, commands are executed on a line-by-line basis with the possibility of typing several commands on a line. There are two forms of the commands:

$\langle n \rangle, \langle m \rangle \langle op \rangle (\langle const \rangle) (\langle comm \rangle) \langle term \rangle$

and

$(\langle n \rangle) \langle op \rangle (\langle const \rangle) (\langle comm \rangle) \langle term \rangle$

Items in parentheses are optional. $\langle n \rangle$ and $\langle m \rangle$ are integers in the range of 1 to 8 or the letter A. $\langle const \rangle$ is a constant in the range ± 32767 ; the value is assumed to be positive unless a minus sign follows the constant. $\langle comm \rangle$ is a comment consisting of nonnumeric characters. $\langle term \rangle$ is either a semicolon or a carriage return; the use of semicolon for a terminator allows several commands to be typed on one line. $\langle op \rangle$ is a one-letter command or operation; it falls into one of the following general categories: display, input/output, mathematical, structure, control, real time, or miscellaneous.

Syntax errors cause the entire line to be ignored, and an error message given. Each system has its own set of commands, but there is a certain amount of overlap between systems.

Each command is translated into three words. The first word is the address of the command routine, the second word contains $\langle n \rangle$ and $\langle m \rangle$ in separate bytes, and the third word contains the constant. When not in the "file" mode, the ASCII representation of the carriage return is

placed at the end of the translated command list. The executive recognizes this to be the end of the list and resets the appropriate pointers. In the "file" mode, each line is transferred to an output buffer which is output as it is filled. Only when the second "file" command is found is there a "return" inserted into the list.

Commands can be structured into programs which are then stored on disk. To create a program, one gives the "file" command to go into input mode. The argument of this command defines the file name. Subsequent commands are then translated and stored in this file. When another "file" command is found, the disk file is closed and the system goes back into the command mode. One can then call this file and execute it. One program can call another, but there is no return command to return to the calling program. A program is somewhat analogous to an overlay in some FORTRAN systems.

Another type of structure that is implemented is the "program section." This is a set of commands that starts and ends with a "label." The program section is named by the starting label and can be executed a number of times before returning to the command after the call. One program section can call another, and the return parameters are saved. A program section is then a cross between a subroutine and a DO loop. A label command in our languages is effectively a no-op with an argument. The argument is used internally as the real label.

THE SOFTWARE ENVIRONMENT

In this section, the main features of a general-purpose environment for use in psychological systems are examined by referring to three systems that have been implemented using the command language approach.

Figure 3 shows a block diagram of the data analysis system, DATAN (Morasso, 1972). This system was

Fig. 3. Schematic representation of the DATAN system showing the structure of the "background" loop.

designed to allow a neurophysiologist to input and analyze data recorded during experiments on eye-head coordination in monkeys. It is basically composed of four parts: the main monitor, a monitor-user interface, command routines, and a data base. Command routines are basically independent, but can interact by virtue of their effects on the data base.

DATAN is an off-line system which uses data stored in DECTapes. The system allows from one to eight channels to be displayed on a storage scope. The channels can then be manipulated independently or together. For example, signals from eye muscle electrodes can be converted to degrees by a linearization procedure that involves building up a calibration table and then using it to convert voltages to degrees. A similar procedure can be performed on the head position signal. These two channels can then be added, and the resultant function, the direction of gaze, displayed. In order to facilitate manipulation of data, two kinds of buffers are used. "Display" buffers act as accumulators in which the operations can be performed. All input/output functions are also performed on these buffers. A second set of buffers, the "keep" buffers, allow a display buffer to be saved and then restored at a later time. The display and keep buffers are 256 words long. This figure was chosen because it corresponds to the length of a block of data on DECTape, and because it makes for an effective display.

Since each buffer has to be scaled differently in order to fit in its block on the screen, a way of recovering the original data was devised. This is simply to store the scale factor plus an offset at the start of the buffer. In addition, the X and Y coordinates of the origin have to be stored for each block to allow blocks to be moved around on the screen. This facility is needed in order to generate three-dimensional displays, to measure the

simultaneity of events in different blocks, or to compare two blocks for identity without having to subtract one from the other. Table 1 is a complete list of the commands available in this system.

Table 1
DATAN Commands

A	Average block
B	Blow up around cursor
C	Calibrate block
D	Take derivative of block
E	Erase scope
F	Filter block
G	Graphic (plotter) output
I	Integrate block
J	Justify block
K	Keep block
L	Linearize block
M	Measure position of cursor
N	Normalize block
P	Execute program section
Q	Squeeze block
R	Restore block
S	Standard deviation of block
T	Truncate above or below
U	Unnormalize block
V	Event definition
W	Write to tape
X	Execute program file
←	Move block left/right
↑	Move block up/down
!	Change scope mode
?	Exit to monitor
=	Restore origins of all blocks
>	Move tape forward and read
<	Move tape backward and read
*	Multiply block by constant
/	Divide block by constant
+	Add constant to block or block to block
-	Subtract constant from block or block from block
:	Define label
.	Open/close program file

Fig. 4. Relation of routines to data bases in the UNITS system. A, keyboard routines; B, scope routines; C, mathematical routines; D, control routines; E, interrupt handling routines.

An extension of the data analysis program was made so that it could handle real-time inputs and control certain aspects of a neurophysiological experiment in which recordings are made from single units in the visual cortex. These extensions are the basis of the single units system, UNITS. This system allows one to create poststimulus time histograms, response histograms, and combinations of these. An important design specification was that the system be able to vary stimulus parameters randomly.

Basically, the UNITS system is the same as DATAN, except that some commands make use of the real-time clock and the class of inputs and outputs has been extended to include digital I/O. The data structure and the relation of the command routines to this structure are what have changed the most. In the DATAN system, almost all commands worked on the display buffers, but now there are routines that work on raw data and running calculation buffers.

The restructuring of the data base is indicated in Fig. 4. While real-time data collection and control is taking place, the system operates in a multiprogrammed mode, with a foreground and a background process running concurrently. However, the more complicated problems of multiprogramming have been avoided, since the two processes do not interact significantly.

Table 2 lists the commands available in the UNITS system. A comparison with Table 1 will show that there is a marked decrease in the number of mathematical routines. This is because many of the calculations that are normally done on data of this kind, such as finding means and standard deviations of responses, peak responses, and tests of significance, are done automatically by the several variants of the output commands.

The final system to be considered, PERIM, was

designed for psychophysical experiments in which the stimuli are generated and controlled under complete computer control using a special-purpose display device. Although it is designed for this purpose only, it is a multiprogramming system, and it makes extensive use of

Table 2
UNITS Commands

A	Execute program file
B	Blow up block
C	Set clock
D	Define number of bins
E	Upper limit
F	Filter block
G	Go
H	Post stimulus time histogram
I	Input heading text
J	Input stimulus text
K	Keep block
L	Vary length
M	3D mode
N	Normalize block
O	Output to tape, plotter, or scope
P	Execute program section
Q	Squeeze block
R	Vary stimulus orientation
S	Erase scope
T	Tally sweeps
U	Unnormalize
V	Vary velocity of stimulus
W	Vary stimulus width
X	Vary X position of stimulus
Y	Vary Y position of stimulus
Z	Restore block
!	Set stimulus position
&	Wait for real time operation to finish
)	Compress to 1/2 number of bins
(	Fast/slow data mode
[	Clear buffers
]	Mark end of program or section

+, -, *, /, ., :, >, <, ?, =, ↑, and ← as in DATAN system

Fig. 5. Structure of PERIM showing the addition of a foreground loop to the background loop of the DATAN system.

the hardware resources (Korn, 1973; Mills, 1969; Moberg, 1967). This system has commands for defining stimuli and for specifying their dynamics. A typical use of this system is for the study of visual field defects in patients with brain injury. In this kind of experiment, one generates a small disk and causes it to move from the edge of the screen towards the center along different radii.

This system has a variable data base which depends both on the number of figures being displayed and the complexity of their dynamics. Figure 5 gives an overview of this system. The background loop is essentially the same as in the first two systems, but a foreground loop, designed to handle the real-time aspects, has been added.

The other significant change is the presence of routines and data structures stored on secondary storage. A queued priority system prevents deadlocks in accessing the secondary storage, which is a fixed-head disk (Saltzer, 1966). Disk resident routines are brought into a dynamic memory area or into an overlay area at the top of the stack. If it is to be brought into the stack, the entire stack is pushed down the length of the routine and the routine brought into core. When the overlay "returns," the stack is popped back to its original place.

A dynamic memory allocation scheme is used to manage free core in the system. Its functions include allocating and reclaiming buffers and concatenating adjacent buffers (garbage collecting). There is no

Fig. 6. Structure of the scheduler data base. Each priority level consists of a linked list of entries. Each entry has a name associated with it which can refer to a figure in the case of "dynamics" routines.

Fig. 7. Structure of the data base of a figure. Information regarding the form is centralized in a figure table, while the dynamics are specified in the figure dynamics table.

automatic invocation of the garbage collector. Normally, it is called after an unsuccessful attempt to allocate a buffer and on recalining a buffer for the system.

Commands that specify the dynamics of a figure normally use several routines to implement the actual moving of the figure. This is done by scheduling these routines at the proper rate, and by noting which routines a given figure is using. Since one can cancel or stop a figure at any time, it is important that the system know which routines a given figure is using. This feature is implemented by giving each routine a different name every time it is scheduled. The name is composed of two parts: one part identifies the command which caused it to be scheduled, the second part identifies the figure that it is associated with.

The essentials of the data structure of this system are outlined in Figs. 6 and 7. Figure 6 shows the scheduler's data base structure. It consists of a forward- and backward-linked list for each priority level. This makes it easy to add or delete entries, and to find a particular entry in the list. One of the elements of a scheduler entry is the name. Figure 7 shows the data structure associated with a figure. Information about the form of the figure is centralized in the figure table. Information about the dynamics is in the figure dynamics table. A template for building the figure is located in the figure descriptor. This table is used by a generalized building routine to construct the actual figure in the output buffer. Table 3 lists the commands for PERIM.

DISCUSSION

We have seen that despite the widely varying purposes of the three systems described above, certain parts of the software remain constant. These parts are located in the "background" loop. The components of this loop are a Teletype driver, a command string interpreter, an executive, and a scope driver that displays constantly

changing data. These elements, in conjunction with system objectives, help define a command set. Given these commands, a data structure is designed that will normally represent a compromise between generality, on the one hand, and speed of access and modification, on the other. Having a design that is modular in nature is of great help, since no system is ever designed and implemented without some change in design along the way. For example, we have noticed that some commands can be added with little overhead, and often

Table 3
PERIM Commands

A	Define arc length
B	Blank screen, erase 611 scope
C	Stimulus is a disk
D	Stimulus is a diamond
E	End of run, clear buffers
G	Go
H	Hold figure
I	Set figure intensity
J	Enter stimulus text
K	Enter experiment identification
L	Set background intensity level
M	Measure cursor position
N	Nullify last point
O	Output
P	Move stimulus along perimeter
Q	Stimulus is a square
R	Radial movement
S	Set starting angle
T	Set stimulus tilt
U	Usurp figure
V	Set figure velocity
W	Wait for response
X	Execute program file
Y	Execute program section
Z	Set response mode
!	Next commands refer to figure n
@	Stop figure
ctrl/c	Exit to monitor
#	Release figure

., ., >, <, =, &, ↑, ←, and ↓ as in UNITS system

these convenience commands have turned out to be very useful. Similarly, some commands have had to be dropped because they simply were not used.

The importance of having a flexible graphics capability cannot be overemphasized. In most experiments, there are too many parameters that have to be monitored to make the Teletype the system output device. An added benefit of having a graphical output built in is that publishable photographs can be taken routinely of the scope face rather than having to either draw or plot a function later. On the other hand, inexperienced users tend to trust the Teletype output more than output on a scope, and for this reason the amount of information displayed is limited. In our systems, all the information displayed on the scope can be output to either DECTape or a Teletype and then be redisplayed later. Figure 8 shows the format of the scope for the UNITS system.

Despite the fact that the commands are only one character long, users have had no trouble learning to use these systems. This is probably so for two reasons: Command names are chosen to have mnemonic meaning, and the number of commands that the user needs to know in order to accomplish something useful is small.

In summary, the main advantages to using the command language approach to problem-oriented systems are: (1) The systems are naturally modular, (2) uniformity of design makes documentation and modification easier, and (3) the systems are easy to use because each module represents a solution to an experimental problem.

REFERENCES

- Corbató, F. J., & Vysotsky, V. A. Introduction and overview of the Multics system. In *Proceedings of the American Federation of Information Processing Societies Fall Joint Computer Conference*, 1965, 27, 185-196.
- Korn, G. A. *Minicomputers for scientists and engineers*. New York: McGraw-Hill, 1973.
- Mills, D. L. Multiprogramming in a small-system environment. In *Second Symposium on Operating Systems Principles*, Association for Computing Machinery, 1969, 112-129.
- Moberg, L. V. An executive system for on-line programming on a small scale system. In *Proceedings of the American Federation of Information Processing Societies Fall Joint Computer Conference*, 1967, 31, 243-254.
- Morasso, P. DATAN: A data analysis system written in FOCAL-12. *DECUSCOPE*, 1972, 11, No. 3, 24.
- Ossanna, J. F. The current state of minicomputer software. In *Proceedings of the American Federation of Information Processing Societies Spring Joint Computer Conference*, 1972, 40, 111-118.
- Digital Equipment Corporation. PDP-11 disk operating system. DEC-11, SERA-D, Digital Equipment Corporation, Maynard, Massachusetts, 1971.
- Reigl, E. W., Faber, U., & Fisher, D. A. The interpreter—A microprogrammable building block system. In *Proceedings of the American Federation of Information Processing Societies Spring Joint Computer Conference*, 1972, 40, 705-723.
- Saltzer, J. H. Traffic control in a multiplexed computer system. Report MAC-TR-30, Project MAC, M.I.T., July 1966.
- Stacy, R. W., & Waxman, B. D. (Eds.) *Computers in biomedical research*. Vol. II. New York: Academic Press, 1965.

Fig. 8. Scope format for the UNITS system. Text items are changed via commands. Other information is: X and Y coordinates of cursor, active bin number, number of sweeps, current time, and data mode.