On the varieties of computer experience

STEVEN J. STADLER, *GRASON-STADLER COMPANY, INC., West Concord, Massachusetts 01781*

An on-line, real-time, high-level experiment control language is described. It is written to run on a PDP/8I with 8K memory, automatic multiply-divide. PTO8B and RO-33 Teletype, and a interface hardware system.

In what I am about to say, I shall proceed on the assumption that all of you are at the very least positively inclined toward the use of computers in the laboratory. If we can assume that common interest, I would like to discuss with you a series of problems which have, until recently, stood in the way of a satisfactory interactive relationship between experimenter and computer for all but a small group of pioneering spirits.

The following list of problems will not include everything that may have discouraged one or another of you at one time or another, but is meant to be illustrative of the general categories of malaise. Once these have been stated, I shall try to show how recent developments have helped to alleviate or solve some of the nastier ones.

I suppose that, inevitably, the categories of problems fall into two broad groups: software and interface hardware. I think we can assume that the computer hardware itself is no longer a problem except that we would perhaps like it to be more reliable than, at certain critical moments, it has sometimes proven to be. But, we are all, in one way or another, loyal DEC users, so let's talk about the remaining categories.

HISTORICAL OBSTACLES

The Tedium of Machine Language Programming

This includes such factors as the length of time, in months, it takes to become proficient at writing machine language programs, time that could be much more productively used doing research.

The Inappropriateness of Machine Language

Machine language is a stylized way of telling the computer what to do. It has absolutely nothing to do with describing an experiment. Just as FORTRAN effectively describes what happens in mathematical computation, a language is needed that is descriptive of an experiment in the real-world and in real-time, and that is user oriented rather than machine oriented.

The Unavailability of a Generalized, Experiment-Control Interface Package

Although computer manufacturers and other suppliers of digital logic modules sell a variety of devices that can be used to construct special-purpose interface, there has been little help available to the novice. Once again, as with machine-language programming, it takes a good deal of time and ingenuity to design, build, debug, and use a tailor-made hardware interface, not to mention the additional risk of obsolescence when it turns out that a new and promising digression was not anticipated.

These three obstacles–two relating to software, one to hardware–stand in the way of the typical experimental scientist pursuing really productive interaction with a computer.

GENERAL SPECIFICATIONS

I would now like to draw up some specifications for a hypothetical system that circumvents most, if not all, of these obstacles. (1) The programming language should be a high-level, user-oriented, easily-learned language specifically designed for experiment process control. It should be capable of operating in an on-line, real-time, multi-user environment. It should allow the experimenter to interact with the experiment in a conversational way, letting him examine the parameters of the experiment and letting him modify them when he feels so inclined. (2) It should be a generalized, sophisticated, professional language that derives its power, in part, from the hardware interface and computer for which it was written; that is, it should be designed for the hardware it manipulates and vice versa. (3) The various elements of the interface hardware should be modular and expandable so that new and complex experiment requirements can be met by addition *to* the interface rather than rebuilding *of* the interface. It should do a great deal of useful timing, storing, and device-identifying work so that the computer itself can devote most of its time to what it does best, namely the manipulation of numbers. It should be constructed of high-quality, modern logic devices and use the highest standards of assembly to assure long-term reliability of operation. It should be capable of being connected to virtually any input or output device, digital or analog in function, that is likely to be found in the laboratory.

THE SCAT SYSTEM

Finally, let's see how the general problems are circumvented and the general specifications are met by a real-life system that is now available for experiment process control. The system is called SCAT, an acronym for State Change Algorithm Terminology. The basic concept around which the SCAT language is constructed is that of the STATE. A state is characterized by two primary attributes: (1) The initiation of a set of physical events–stimuli of any sort that can be controlled by the interface hardware, as well as such control conditions as D-A conversions, attenuator settings, etc. (2) The continuance of this set of events until a condition has been met–at which time a transition will be made to a new state. Conditions may be the completion of a time interval, the arrival of an nth response, the termination of a D-A conversion, the satisfaction of a logical or arithmetic expression, etc.

SCAT software offers the following multiprogram, multiple-station capabilities: (1) All operator inputs to the system occur by conversation through the teletype. (2) Any station can be loaded, started, stopped, or deactivated while any or all other stations are running. When a station is deactivated, its core memory allocation is released and made available for other stations and/or programs. (3) Experiment parameters for any station can be examined and modified while the program is loaded and/or running. (4) Data can be outputted in a variety of formats on a separate report Teletype whose operation does not interfere with query and control through the main Teletype. (5) Experiments are written in a high-level, easily learned experiment control language designed expressly for multiple experiment environments.

Software

The software consists of three parts: (1) *The SCAT Editor/Compiler.* This program allows the entry, editing, and compilation of SCAT language programs. Error checking and diagnostics are provided. The output of this program is a tape suitable for input to the SCAT assembler. (2) *The SCAT assembler.* This program accepts output tapes from the SCAT Compiler or hand-prepared SCAT assembly language tapes and produces SCAT binary tapes which are acceptable to the SCAT Operating System. (3) *The SCAT Operating System.* This program controls the interpretation of user programs, communicates with the operator, loads, starts, stops, kills, and/or queries the SCAT programs which control the stations.

The SCAT Language

In order to illustrate how the SCAT language meets the specification of user orientation, let me show how simply one would write a fixed-ratio schedule in SCAT.

```
C1:   S1 ON;   (15R1)→(2)'
C2:   S2 ON;   (3SEC)→(1)'
```

Control State 1 (C1) turns on the stimulus (S1) and after 15 responses (15R1) of Type 1 have occurred, a transition is made to State 2 (C2). State 2 reinforces (S2) the S, and after 3 sec, a transition to State 1 occurs and the cycle repeats.

The program is admittedly a trivial one, but it does illustrate dramatically the simplicity of the language, the ease of using it, and its appropriateness, because it does describe the experiment precisely in the way the E would if he were engaged in a conversation with a colleague. In this instance, the colleague happens to be a computer.

The SCAT language offers a series of features designed to make it useful and powerful. The first two of these features are SCAT's arithmetic and relational operators.

Arithmetic Operators

An expression in SCAT consists of an element or of two elements and an arithmetic operator surrounded by parentheses. The arithmetic operators are "+," "–," "*," "/," which stand for add, subtract, multiply, and divide. An element may be a literal (e.g., "1," "17," "30.5"), a constant, a counter, a clock, or a list element (e.g., "CON1," "CTR2," "CLK1," "LIST (17)"). An element may also be another expression, e.g., "(CTR2 + CON1)," ((LIST (2) + 4)/CON2)."

Relational Operators

SCAT also provides a variety of relational operators:

.E.	equal
.GE.	equal or greater
.LE.	less or equal
.NE.	not equal
.L.	less
.G.	greater

The third feature is SCAT's functionals that develop numbers with certain characteristics and that can be used whenever an expression is allowed.

RAND	Generates a uniformly distributed random number in the range 0 to 1.
NORM (x,y)	Generates a random variable from a normal distribution about x, with a standard deviation y.
RANSET (x)	Initializes the pseudorandom number sequence from the value of the expression x.

For an illustration of the conversational, interactive character of SCAT, we should discuss query mode. These commands permit the E to interrogate and modify the parameters of an experiment schedule either while that schedule is resident in core (and inactive) or while it is actually running an experiment. They look as follows:

"CTRn"	Causes the current contents of counter n to be typed out. User may modify contents by typing new number on Teletype.
"CLKn"	Causes the current contents of clock n to be typed out. Contents can be changed as above.
"CONn"	Causes constant n to be typed out. Contents can be modified as above.
"HISTn"	Causes current contents of histogram n to be typed out. No modification is allowed.

Perhaps I have confused you because I did not first define the data elements, such as COUNTER, CLOCK, CONSTANT, etc. In general, data manipulated by the SCAT software is in the form of a two-word floating-point number with sign, five significant digits, and a range of 10^{11} to 10^{-7}. Except for histograms, the data elements listed below all manipulate numbers in this form.

COUNTERS (10 per station)	Referenced in SCAT language as "CTR1" .... "CTRØ."
CLOCKS (10 per station)	Referenced as "CLK1" .... "CLKØ." Each will be automatically incremented by the 10-msec hardware clock associated with each subject station.
CONSTANTS (1C per station)	Referenced as "CON1" .... "CONØ."
HISTOGRAMS (10 per station)	Up to 10 independent histograms are possible for any station, although their number and size may be limited by the amount of storage available. Referenced as "HIST1".... "HISTφ."
LIST (1 per station)	A set of individual cells, each of which can contain a standard floating point number. Number of cells limited only by available space.

Please note that CONSTANT cannot be modified by the program itself, but only by the E, and, conversely, that HISTOGRAM can be modified by the program but not by the E.

Finally, here is a brief description of the output facilities of SCAT, which allow a variety of output commands that facilitate titling and formatting the information contained in counters, clocks, list, etc., when these are printed out on the report Teletype. In addition, it allows the following two commands.

CYCLE	Tells the SCAT Operating System to save the data elements associated with the experiment just run, to assign new space for data elements, to restart the experiment, and then to proceed with the output.
RESTART	Tells the Operating System that the experiment should be run again. Restart saves memory, but makes the experiment wait for output to terminate before reassigning data access and continuing.

I might note that literal character strings are allowed so that titling and data identification are easily accomplished.

CONCLUSION

You have now heard and seen the specifications of the SCAT language. We should go on and discuss the interface hardware of the SCAT system because many interesting ideas are contained in it. But I thought that, given the limited time available, you would like to hear first about the language. If you are interested, we can talk hardware during the discussion period.

In conclusion, I hope you will agree with me that the SCAT system is a significant jump forward on the path of experimenter computer interaction. The SCAT system has been designed to simplify enormously the problems enumerated at the beginning of this paper, or to eliminate them altogether. I look forward to your questions and comments.

REFERENCES

CIOFALO, V. B., TEDFORD, R. H., GOLDBERG, M. E., & DODGE, D. L. The use of a digital computer for control of behavioral research in animals. Behavioral Science, 1968, 13, 252-256.

HABER, R. N. An on-line computer in a visual perception laboratory. Behavior Research Methods & Instrumentation, 1968, 1, 86-93.

MARKOWITZ, J., & NICKERSON, R. S. On timing events in computer-controlled experiments. Behavior Research Methods & Instrumentation, 1968, 1, 82-86.

UTTAL, W. R. "Basic Black" in computer interfaces in psychological research. Behavior Research Methods & Instrumentation, 1968, 1, 35-40.

UTALL, W. R. *Real time computers–Technique and application in the psychological sciences.* New York: Harper & Row, 1968.

Can a small dedicated machine find happiness in a company that pioneered time-sharing?[1]

JOSEPH MARKOWITZ, *BOLT BERANEK AND NEWMAN, INC., Cambridge, Massachusetts 02138*[2]

In the context of an automated psychophysical laboratory, the adequacies and inadequacies of a small computer are discussed. Attention is given to the rationale for choosing the implementation in particular a dedicated small system as opposed to time-shared use of a larger system. The variables discussed include cost, flexibility, language level, and storage capabilities in addition to reliability. A compromise position that appears most viable for the future is also suggested.

Several years ago, we had the opportunity to build a new laboratory facility for psychological research. Specifically, the laboratory was to be used for basic psychophysical research with acoustic signals. In certain instances, the implementation choices may have been closely tied to the portended nature of our work. In general, we think they were not. Indeed, with small modification, the same facility now serves a much broader range of psychological research interests. Perhaps such versatility owes its due to the way in which the facility was, in fact, implemented. For various reasons, which are the very point of this paper, we decided to automate our laboratory. The medium we chose was a small, dedicated, general-purpose digital computer--the PDP-8.

Our need for automation arose, we felt, because of the drudgery that defines psychophysics. Our psychophysical experimentation is characterized by a simple, repetitive, short sequence of events where all the alternatives are drawn from a small closed set. It is characterized by a need for numerous observations in each of a few fixed conditions. The operations are routine and boring. Errors made by the E as to what he presents (or did present) are indistinguishable from errors on the part of the O as to what he observed. In this sense, errors are very costly.

Psychophysics, then, does not demand a computer of high intellectual ability or blinding speed. Its tasks are not that challenging. Rather, psychophysical research requires a tireless and accurate drone.

Quite simply, precision increases as E errors decrease, and as the number of observations increases. Unfortunately, the rate of an E's errors accelerates as the number of observations increases. On the other hand, a computer can keep its own error rate satisfactorily and constantly low.

In designing our laboratory facility, we were confronted with several options. Because all of these options are nominally open, even today it is worthwhile reviewing them.

The first option we examined was construction of a special-purpose facility out of logical building blocks. This was, of course, a proven approach. We, ourselves, had had experience with three generations of logical building blocks. We were intimately familiar with the clatter and electromagnetic splatter of relays, the shocking and geriatric nature of vacuum tube logic, and the unforgiving frailties of transistors. We knew, too, the joys and sorrows of a variety of connectors and programmable patch panels. Finally, we knew the disproportionate number of logic modules and readout devices required for even the simplest of data analyses. All of these lessons stood us in good stead, we felt, and we never regretted the experiences of the past. Yet, we were firm in our resolve to depart from the past, for the lure of computers was irresistible. The printout was on the wall.

Another way in which we could have accomplished the automation of our psychophysical laboratory was to use time-sharing facilities available to us. We could conveniently have run our experiments on one of several operating systems. Our colleagues interested in running manual-control experiments were setting out to do precisely that. Moreover, ruling out such an alternative would be particularly significant in view of our past history with time-sharing and with computerized psychological experimentation.

Bolt Beranek and Newman Inc. (BBN) has actively pursued time-sharing system research, development, and application for nearly a decade. Historically, we developed one of the first operational interactive time-sharing systems in 1962. That system was implemented on a DEC PDP-1 with the support of the National Institutes of Health.

A succession of hardware and software improvements enabled us to move from 4 to 64 simultaneous users and to increase the number and diversity of languages available to the individual user. In 1963, development was undertaken on a second PDP-1 interactive, time-sharing system. This system provided a number of operational programs in a specialized language at remote terminals in the Massachusetts General Hospital. In 1966, we implemented a real-time hybrid I/O interface to our time-shared SDS-940. This interface is called a Hybrid Processor, and is capable of accurately timing real-time data transfers between