A brief description of EXTENDED SCAT*

PETER G. POLSON†

University of Colorado, Boulder, Colorado 80302

State Change Algorithm Translator (SCAT) is a programming language specifically designed for experimental process control in the psychological laboratory. The first SCAT system, based on a Digital Equipment Corp. PDP-8/I, was developed and marketed by the Grason-Stadler Company (Stadler, 1969). EXTENDED SCAT is a dialect of the SCAT language that includes all of the basic features of the language plus additional capabilities that facilitate the programming of a wider range of experimental procedures. This paper briefly describes the possible applications of the SCAT language, the design criteria used in the development of EXTENDED SCAT, and the additions made to the SCAT language. It is assumed in this last section that the reader is familiar with the SCAT System developed by Grason-Stadler. Space limitations prevent me from describing the SCAT language in any detail.¹

APPLICATIONS

The SCAT System is a hardware-software system specifically designed for data logging, limited on-line data analysis, and the real-time control of psychological experiments. The language has been designed to simplify the conceptual and technical problems involved in writing computer programs that run in a multiuser, real-time environment. The E can easily describe his procedure in the SCAT language. The SCAT language and operating system enable the E to ignore the fact that his program will run concurrently with programs controlling other laboratories. Finally, SCAT is an easily mastered programming language, and it can be used to teach real-time programming to students with minimal backgrounds in computer science.

SCAT was originally designed for use in experiments involving operant paradigms. However, the language is far more general than that. The SCAT language can describe any experimental procedure that can be characterized abstractly as involving

contingencies between discrete stimuli and a limited number of response alternatives. Examples are the Sternberg (1966) memory search paradigm and human learning paradigms in which the stimuli can be presented by a random-access slide projector and the responses are buttonpresses. The SCAT language is not intended for use in experimental paradigms involving high rates of analog input/output (e.g., many biopsychological experiments) or in situations requiring the recording and processing of long typewritten responses (e.g., many CAI applications).

DESIGN GOALS AND IMPLEMENTATION

Given the decision to develop a high-level language for real-time computing applications, a compiler and operating system were written for an already developed language, SCAT. The design criteria for EXTENDED SCAT were as follows: EXTENDED SCAT and the version of SCAT developed by Grason-Stadler are compatible at the programming language level. Programs written in the Grason-Stadler dialect of SCAT can be compiled on and will run on the EXTENDED SCAT System. All additions to the SCAT language are consistent with the design philosophy of the original SCAT language. The initial additions to the language are intended to make it easier to program various classes of discrete trial procedures where several measures are taken on each trial and recorded for later analysis. In summary, EXTENDED SCAT includes all of the features of the original SCAT language plus additional capabilities that facilitate the programming of a wider range of experimental procedures and that exploit the potential of larger real-time systems like the XDS Sigma 3, IBM 1800, and the PDP-15.

The major elements of the EXTENDED SCAT System's software are the compiler and operating system. The compiler takes as input SCAT statements and outputs basic FORTRAN IV statements. The vendor's FORTRAN compiler is then used to generate object code. A majority of the SCAT commands compile into subroutine calls, and the functions specified by these commands are implemented by subroutines that are part of the operating system. The EXTENDED

SCAT compiler is written in basic FORTRAN IV, and the two-stage compilation process (SCAT to FORTRAN, FORTRAN to object code) is practically system-independent. The SCAT operating system implements the on-line, real-time functions of SCAT, and it is dependent on the machine and particular system configuration.

EXTENSIONS TO THE SCAT LANGUAGE

The following descriptions of the features of EXTENDED SCAT assume that the reader is familiar with the SCAT language as developed by Grason-Stadler.

The Grason-Stadler dialect of SCAT requires that arithmetic expressions be fully parenthesized, and exponentiation is not defined. In EXTENDED SCAT, the rules for expression formation and evaluation are identical to those for basic FORTRAN IV, with the following two exceptions: (1) legal variables are limited to the predefined SCAT variables; (2) any legal arithmetic expression can be a subscript. Integer variables are defined in EXTENDED SCAT. Corresponding to each of the predefined, real SCAT variables are integer COUNTERS, CONSTANTS, and an integer array ILIST.

An additional stimulus descriptor has been implemented in EXTENDED SCAT to enable an E to present material on character-oriented CRT displays like the IBM 2260, the UNIVAC Uniscope 100, and other similar displays. The DISPLAY command causes a character string to be presented on the CRT at specified coordinates. The character strings are defined in the Declaration Section.

Implicit in the SCAT language is the restriction that a program monitors the behavior of a single S. EXTENDED SCAT removes this restriction by enabling an E to coordinate the operation of several programs that are running concurrently and that jointly control an experiment. The TRIGGER command enables an E to transmit limited amounts of information between programs in real time. Associated with each program is a set of logical variables whose states can be manipulated by other SCAT programs using the TRIGGER command. The states of these logical variables are tested using a special form of the State Change Phrase. The purpose of TRIGGER command is to enable EXTENDED SCAT to run experiments in which several Ss respond to the same stimulus display or to run social interaction experiments like two-person games.

EXTENDED SCAT permits

*EXTENDED SCAT has been developed with the cooperation of the Grason-Stadler Company, and their assistance is gratefully acknowledged. This work was supported in part by National Science Foundation Grant GJ 453.

†Requests for reprints should be sent to: Peter G. Polson, CLIPR, 1511 University Ave., Room 4, University of Colorado, Boulder, Colorado 80302.

unformatted, real-time input/output from Control States. The SCAT commands TABLE and DATA enable the E to transmit to and retrieve information from bulk storage, respectively. Associated with each TABLE or DATA command is a declaration defining a list of variables whose values are to be transmitted or to be read from bulk storage. The functions of these commands is to enable the E to store trial-by-trial information for later analysis by a nonreal-time program and to read information necessary for the control of an experiment in real time.

EXTENDED SCAT enables the E to do a limited amount of data reduction and conditional outputting of formatted information in the Report Section. Replacement phrases can appear in Report statements, and the programmer can make conditional

state changes using State Change Phrases involving relational conditions.

Other additions to the SCAT language are in the process of being implemented or are under consideration. Examples are programmer-defined variable names, the segmentation of SCAT programs, and additional commands and functionals. The author solicits any suggestion concerning additions or modifications of the SCAT language.

REFERENCES

- STADLER, S. J. On the varieties of computer experience. *Behavior Research Methods & Instrumentation*, 1969, 1, 267-269.
- STERNBERG, S. High speed scanning in human memory. *Science*, 1966, 153, 652-654.

NOTE

1. A complete description of EXTENDED SCAT can be obtained from the author on request.

channels, and clock. As a complete package, however, I have found FOCAL-12 to be a failure. In the first place, for any experiment I want to run, there always seems to be a critical feature missing from FOCAL. Moreover, in their zeal to include as many features as possible into FOCAL-12, the DEC software specialists did not leave enough room in core for any but the simplest programs. The problem is aggravated to an intolerable level if one adds overlays to the basic FOCAL-12 in order to close relays or to accomplish some other necessary feature of one's program.

My approach has been to subdivide the psychology experiment into components and to use FOCAL only where it seems appropriate. Assembly language is reserved for those parts of the experiment where flexibility and speed of program execution are essential. Most experiments seem to divide nicely into three parts. In the first phase, one specifies the parameters to be used for a particular session and, on the basis of this, generates a sequence of trials, usually in random order. If one wishes to have a general program with many options, the initialization functions may require a large number of machine-language instructions. On the other hand, FOCAL contains all of the Teletype, binary-to-decimal, decimal-to-binary, and mathematical routines which make programming of this portion of the experiment a pleasure. Moreover, the relative slowness of an interpreted language is generally not a problem when one is inputting parameters.

The second phase of the experiment is concerned with the actual gathering of data, and here we have found Assembly language to be far and away the best choice. However, Assembly language is quite simple to use in Phase II since, for the most part, the operations involve turning machinery on and off and checking for responses. This is the part of the program where timing may be most critical, where particular user subroutines are most often required, and where it is desirable to have large segments of core devoted to data storage, not to a resident interpreter.

In the third phase of the experiment, the data collected in Phase II are analyzed and a printout of the results is usually obtained. Here, for obvious reasons, FOCAL can be used to best advantage.

To summarize, FOCAL seems to be preferable to Assembly language for inputting parameters, generating sequences of trials, and analyzing data. On the other hand, Assembly language wins out for the data-collection phase

Combining FOCAL and Assembly language

WILLIAM SIEGEL*

University of Western Ontario, London, Ontario, Canada

While programming of psychological experiments in Assembly language is most efficient from the machine's viewpoint, it is not without disadvantages for the user of the machine. Assembly language programming is tedious and time consuming and may be so negatively rewarding to the novice as to turn him forever back to his Digibits.

What are the possibilities if one wishes to use a higher-order programming language? One solution is to avoid the minicomputer market and to try a midi or maxi instead. A second is to write, borrow, or steal a software package which is specialized for the running of psychological experiments. I am aware of a number of such packages, but none of them seems to incorporate the particular features I need. I would write my own language, but I never seem to stick to one area of psychology long enough for a specialized package to do me any good. A third possibility is to use DEC's 4K or 8K FORTRAN package, but one is limited here by considerations of program length.

The solution that we have adopted at present is to employ FOCAL, an easy-to-use interpreted language, for

most of our programming chores. FOCAL is a FORTRAN-like language which makes programming, editing, and debugging easy to accomplish (in real time, no less), and one of its virtues is that it can be learned in a number of hours.

Given that FOCAL has been readily available for several years, you are probably asking yourself why few persons besides me seem to be using it for research. The answer is that, at least in its standard versions, FOCAL makes no provision for data input and storage on paper or magnetic tape. The result is that the only way that data can be accessed is on the Teletype, and so FOCAL seems to allow the minicomputer to be used as a sophisticated and expensive desk calculator and little else.

This problem has been circumvented with the new FOCAL which is now available for the PDP-12. FOCAL-12 allows the user to save and access both data and programs on magnetic tape. Data stored in tape files may be in integer, signed-fraction, or floating-point format.

DEC has attempted to make FOCAL-12 a complete system for the on-line control of experiments and has included instructions which access a number of peripheral devices, including the scope, analog input

*I would like to thank Keith Whittle and Jane Siegel for their help in developing the system described here.