TYMES: A high-level language for process control and data manipulation in the behavior laboratory

TETSUO TAKIGAWA
Hokkaido University, Sapporo 060, Japan

and

TETSURO MINO
Kwansei Gakuin University, Nishinomiya 662, Japan

This paper describes outlines of the TYMES software system designed and developed to control complicated experiments in the behavior laboratory. A versatile programming language is adopted to perform a wide range of real-time data manipulations and interactive controls based on a PDP-11 small computer.

TYMES is a multiuser real-time monitoring system developed on a DEC PDP-11 computer to describe the sequential control flow in psychological experiments. The TYMES software system consists of the TYMES language compiler, the object program loader, and the run-time monitor (RTM). The compiler and loader are executed in background, and the RTM is executed as a foreground job under DEC's RT-11 operating system.

PROGRAMMING LANGUAGE

TYMES source programs are written in the generalized form of a state notation system, as in SKED (Snapper & Kadden, 1973) and ACT (Millenson, 1973, 1975), but they differ in the structure of programming. In general, a sequence of experimental controls may be described in the form of procedures concatenated by transitions. In each procedure, several statements, each representing a kind of operation, are placed and executed one by one in sequence. A cluster of procedures concatenated by transitions is called a block. TYMES allows up to 128 procedures in one block and up to five blocks in one program.

Procedure and Block Labels

At the beginning of the first statement in a procedure, a procedure label is placed for identification and the interprocedure transitions are performed in reference to that label. A procedure label is formed by up to six alphanumeric letters begun by an alphabetical letter and followed by a colon. At the beginning of the first procedure in a block, a block label is placed and is formed as a procedure label with a leading dollar mark. For example, LABEL1: is a procedure label; \$LABEL2: is a block label.

Number System and Variables

Decimal one-word (16-bit) integers (−32,768 to 32,767) and two-word floating-point real numbers (8 bits exponent, 23 bits fraction) are permissible. In addition, octal and binary notations of integer constants are allowed to represent bit patterns by adding a backslash character followed by a letter O or B. For example, 100 is a decimal notation, 100\O is an octal notation (64 in decimal), and 100\B is a binary notation (4 in decimal).

A variable in TYMES is a symbol representing an integer number, or a real number if declared in the REAL statement, and is formed by up to six alphanumeric letters, as in FORTRAN. The TYMES variables are common for all blocks so that the synchronized control across blocks is easily realizable.

STATEMENTS

A TYMES program is composed of lines of statements, each of which starts with an English word specifying the type of operation to be performed by the RTM. TYMES statements are divided into five categories according to their functions, as listed in Table 1.

Control-Oriented Statements

An experimental room controlled by TYMES is called a booth. It is assumed that a control panel on which 12- to 16-bit TTL-level input/output signal lines are terminated is installed in each booth.

The ON and OFF statements send control signals to the booth. The signal pattern of the ON statement is the logical OR of the operand and the bit pattern already in effect. The signal pattern of the OFF statement is the logical AND of the complement of the operand and the bit pattern already in effect.

Table 1
Summary of TYMES Statements

Control-oriented Statements:	
ON	[#b:] operand
OFF	[#b:] operand
INPUT	[#b:] variable
Transition-oriented Statements:	
GOTO	label
IF	operand 1 * operand 2 { $\begin{smallmatrix} \text{GOTO} \\ \text{THEN GOTO} \end{smallmatrix}$ } label
AFTER	operand [,] GOTO label
Terminal-oriented Statements:	
PRINT	[#t:] $\left\{ \begin{smallmatrix} \text{variable} \\ \text{('') characters ('')} \end{smallmatrix} \right\}$
READ	[#t:] variable
File-oriented Statements:	
ASSIGN	#f1: [, #f2: , ...] list of variables
OPEN	file name = f FOR $\begin{smallmatrix} \text{INPUT} \\ \text{OUTPUT} \end{smallmatrix}$
GET	#f
SAVE	#f
CLOSE	#f
Miscellaneous Statements:	
DIM	variable(size) [, variable(size) , ...]
REAL	variable [(size)] [, variable[(size)] , ...]
[LET]	variable = $\begin{smallmatrix} \text{function(operand)} \\ \text{operand 1 } \otimes \text{ operand 2} \end{smallmatrix}$
TIME	variable
RANDOM	variable
FOR	k variable = operand 1 TO operand 2 [STEP operand 3]
NEXT	k
GOSUB	label
RETURN	
IF	operand 1 * operand 2 THEN statement
STOP	
END	
{NOTE;	
\$END	

Note—In the notation, operand means either a variable or a constant, * is a relational operator, $\otimes$ is an arithmetic operator, [] is an optional specification, {} is a choice, b is a booth number, t is a terminal number, f is a logical file number, and k is a loop identification number.

The INPUT statement waits for input signals from the booth and stores the signal pattern into the variable, and execution continues to the next statement. When the temporal transition by the preceding AFTER statement occurs, the data input is canceled. Therefore, the order of statements is important.

Transition-Oriented Statements

The transition-oriented statements are used to control the interprocedure transitions. The GOTO statement transfers the control of a program unconditionally from the current procedure to that specified. The IF-GOTO or IF-THEN statement transfers the control of a program to a specified procedure or executes a specified statement if the designated condition is satisfied at the execution moment. If not, execution continues to the next statement.

The relational operators to be tested are: = (equals), <= (= <) (less than or equals), >= (= >) (greater than or equals), < (less than), > (greater than), and >< (> <) (does not equal).

The AFTER statement transfers the control of a program to a specified procedure after the duration set by the operand. The value of the operand designates the units to be counted. The counting is in system clock-tick units if the operand is followed by a double quotation mark, and it can be in seconds if the operand is followed by an apostrophe. The floating real value of the operand is converted to an integer. At the execution of the AFTER statement, the RTM starts the internal user's timer and continues execution to the next statement if placed. At the end of a specified timing interval, it causes transition and cancels the following input-waiting statement (INPUT or READ). In contrast, the temporal transition is canceled if any other following statement is still being executed or the transition is already being performed by any subsequent GOTO or IF-GOTO statement.

Terminal-Oriented Statements

TYMES supports up to eight terminals on which PRINT and READ statements are performed independently of the RT-11 operating system. The PRINT statement is used to print either a value of the variable or a message on the terminal. The RTM refrains from executing the next statement until the printing is completed. When the PRINT statement is used without the operand, it causes a carriage return and a line feed.

The READ statement is used when an ASCII value is to be input from the terminal keyboard. In execution of the READ statement, the RTM waits for a key strike and stores the value of the key code into the variable. The occurrence of the temporal transition by the AFTER statement cancels the key input. In the multiterminal environment, the terminal on which the READ or the PRINT statement is performed is specified by a # mark and a number followed by a colon.

File-Oriented Statements

The file-oriented statements, including ASSIGN, OPEN, GET, SAVE, and CLOSE, are concerned with the read/write operations on file-structured devices. The TYMES data files can be made to correspond with the variables defined in the program.

The ASSIGN statement associates the variables with the logical file number to specify the contents of the

data file. The ASSIGN statement is not a run-time operation, but a directive to the compiler. The OPEN statement is used to open a data file for input or output and associates it with the logical file number specified in the ASSIGN statement. Once opened, an input file may be read by the GET statement, and an output file may be written by SAVE. The GET statement transfers the contents of the data file to the variables associated by the ASSIGN statement. The SAVE statement transfers the contents of the variables to the data file. The CLOSE statement makes the opened and saved data file permanent. When a file is closed, the number for the file can be reused to open another data file.

Miscellaneous Statements

The miscellaneous statements, consisting of DIM, REAL, LET, TIME, RANDOM, FOR, NEXT, GOSUB, RETURN, STOP, END, and NOTE, enable the programmer to control complicated experiments precisely and conveniently.

The DIM (dimension) statement declares and reserves memory space for one-dimensional arrays of integer variables used by the program. The first element is assumed to have a subscript of zero. The REAL statement declares the data type of the variables to be two-word floating real numbers. The REAL statement is also used to declare one-dimensional arrays. These statements must precede the first block label.

The LET statement assigns a value to the specified variable. The word LET is optional. TYMES allows only two types of the LET statements to facilitate functions in the multiuser environment. The first is a single-operand type that assigns a value to the variable using one of the following internal functions of the form LET variable = function (operand), where the operand is a constant or a variable: + (direct assignment, where + is optional), - (negated assignment), ABS() (absolute value), and INT() (truncated value). The following functions may be defined optionally according to the user's selections at the TYMES system generation: SQR() (square root), SIN() (sine in radians), COS() (cosine in radians), ATN() (arctangent), LOG() (natural logarithm), LOG10() (base 10 logarithm), EXP() (exponentiation), ADC() (performs A/D conversion and assigns the value, where the value of the operand specifies the channel number), and DAC() (performs D/A conversion, where the operand specifies the channel number, and the value of the variable is the data to be converted).

The second is a double-operand type that assigns to the variable a value of an expression of the form LET variable = operand $\otimes$ operand, where $\otimes$ represents one of the following operators: + (addition), - (subtraction), * (multiplication), / (division), & (logical AND), and ! (logical OR).

The TIME statement stores the system clock-tick counts into the variable. The system clock counter

consists of 24 bits and is incremented by 1 at the system clock interruption. If an integer variable is specified, the lower 15 bits' value of the counter is stored.

The RANDOM statement stores a floating real uniform random number between 0 and 1 into a real variable.

The FOR and NEXT statements define the beginning and the end of a program loop. A loop is a set of statements or procedures that are repeated until the value of the variable reaches the final value, each time being modified by the STEP value. If the STEP is omitted, +1 is assumed. There must be one NEXT statement for each FOR statement indexed by the same loop identification number.

The GOSUB statement causes execution of a sequence of procedures called a subroutine. The RETURN statement causes program control to return to the statement following the GOSUB. The TYMES subroutines are identical to other procedures but end with the procedure including a RETURN statement.

The STOP statement terminates the execution of the concerned block, and the END statement terminates the entire program. Both may be placed anywhere in the program. When the END statement is executed, all opened files are closed.

The NOTE statement, or alternatively simply a semicolon (;), is used to insert comments into a program without these comments affecting execution. A semicolon is especially useful to insert a comment at the end of a statement in the same line.

In addition, the ending statement \$END must be placed at the end of a program. This is a directive to the compiler, as is the ASSIGN statement.

Programming Example

Program development in TYMES goes through the following steps: (1) creating a source program file with the aid of the text editor, (2) compiling the source program and creating a loadable object program by using the TYMES compiler, (3) loading the object program by using the TYMES loader, and (4) executing the program using the run-time commands to the RTM.

The compiler accepts source programs and translates them into the internal format with error checking. The RTM interprets and executes the loaded program one statement at a time in a round robin fashion. As there is often a wait for data transfers or for some clock ticks in execution of ordinary experiments, idle times of the RTM are available for a non-time-critical background program such as the compiler, the loader, or any other utility program.

A TYMES program example is shown in Figure 1.

KEYBOARD COMMANDS

To communicate with the RTM, the run-time one-letter commands are used. The RTM responds to a com-

```

NOTE  TYMES PROGRAM EXAMPLE: RECORDS RESPONSE TIME DATA
NOTE                                     IN "DRILL20/LR36" SCHEDULE
REAL  RTIME(500)                       ;RESPONSE TIME DATA
ASSIGN #1:RTIME(500)                   ;ASSIGN FILE NO. FOR RTIME

SHAIN:
INIT:  GO=0                           ;POSTPONE THE EXECUTION
      HOUSE=1                         ;HOUSE LAMP SIGNAL BIT
      CUELAP=2                       ;CUE LAMP SIGNAL BIT
      FEED=4                          ;FEEDING SIGNAL BIT
      ALLBIT=7                       ;HOUSE/CUELAP/FEED
      OPEN DY1:DATA=1 FOR OUTPUT;OPEN DATA FILE FOR RTIME
      FOR 1 N=0 TO 500
      RTIME(N)=0.0                   ;CLEAR VARIABLES
      NEXT 1
      GOTO START

START:  N=0                           ;RESPONSE COUNTER
      ON  HOUSE                      ;TURN ON THE HOUSE LAMP
      TIME RTIME(0)                  ;SAVE THE STARTING TIME
      GO=999                         ;START THE EXPERIMENT
      AFTER 1800' GOTO FIN           ;30 MIN. SESSION
FIN:    OFF ALLBIT                   ;ALL SIGNALS OFF
      SAVE #1
      CLOSE #1
      END

SEXEC:  BEGIN: IF GO=999 GOTO LOOP    ;WAIT FOR THE FLAG
      GOTO BEGIN
LOOP:   INPUT LEVER                  ;WAIT FOR THE LEVER PRESS
      N=N+1
      TIME RTIME(N)
      OFF CUELAP                    ;TURN OFF THE CUE LAMP
      GOTO ITI
ITI:    OFF FEED                    ;TURN OFF THE FEEDER BIT
      AFTER 5' GOTO CUE             ;WAIT FOR 5 SECONDS
      INPUT LEVER                   ;HE CAN'T
      N=N+1
      TIME RTIME(N)
      GOTO ITI
CUE:    ON CUELAP                   ;TURN ON THE CUE LAMP
      AFTER 20' GOTO RANGE          ;WAIT FOR 20 SECONDS
      INPUT LEVER                   ;HE CAN'T
      N=N+1
      TIME RTIME(N)
      OFF CUELAP
      GOTO ITI
RANGE:  AFTER 36' GOTO LOOP          ;THIS IS THE CHANCE
      INPUT LEVER                   ;HE GOT IT
      N=N+1
      TIME RTIME(N)
      RTIME(N) = -RTIME(N)          ;MINUS MEANS REINFORCED
      OFF CUELAP
      ON FEED                       ;DINNER IS HERE
      AFTER 2' GOTO ITI            ;40MS. DURATION IS REQUIRED
      SEND

```

Figure 1. TYMES program example.

mand typed on the console terminal by printing the entire command name; it then asks for the user's two-letter identification name (ID) and then executes the command. The user programs are distinguished only by the user ID. The run-time commands and their meanings are listed in Table 2.

In the single-terminal environment, the simultaneous sharing of the terminal with the background job is performed by typing CTRL/F (type F while holding down the control key) for the RTM commands and typing CTRL/B for the background job.

HARDWARE CONFIGURATION

A typical hardware system for TYMES is shown in Figure 2. TYMES requires a PDP-11 or an LSI-11 CPU, 16K words of memory, a mass storage device, a digital input/output interface, a line frequency clock, and a console terminal as a minimum configuration. In addition, the RTM supports up to eight terminals, A/D and D/A converters, the EIS (extended instruction set), and the FIS (floating-point instruction set). The memory management for the memory exceeding 28K words is not supported in the present version. Total memory required depends on the number of users, length of programs, internal functions included, and the background job to be executed concurrently. The type of CPU also restricts the number of time-critical tasks required.

System Clock

The line frequency clock (60 or 50 Hz) is used as the standard of a time base for the TIME and the AFTER

Figure 2. Typical hardware configuration for TYMES. The multiplexer and converters may be omitted.

Table 2
Keyboard Commands

Command	Response	Meaning
C	CLOSE FILE ID:	Close data file opened
D	DELETE ID:	Delete the loaded program
H	HALT ID:	Halt the program just running
P	PROCEED ID:	Proceed the program halted
R	REPORT	Report the status of loaded programs
S	START ID:	Start the loaded program
CTRL/T	\$	Return to the RTM command mode
CTRL/X	EXIT	Return to the RT-11 command mode

Figure 3. The input interfacing circuit. Intel's 8214 interrupt priority controller sends the interrupt signal to DR11. The 12-bit data from the booth are latched in 8212 and are fed into DR11.

Figure 4. The output interfacing circuit. The higher 4 bits of DR11 are fed into SN74159, which selects 1 of the 16 booths.

statements. The accuracy of the system clock is that of the frequency source. The line clock interruptions are processed first by the RTM and handed over to the RT-11 resident monitor. Any other clock rate up to 1 kHz may be obtained by utilizing DEC's KW11-P or any clock-interrupting hardware.

Booth Interfacing

TYMES does not require special hardware except the circuit for the multibooth interfacing. In TYMES, a 4-bit multiplexer is utilized to manage up to 16 booths. Therefore, only one 16-bit parallel interface (DEC's DR11 series) is employed, and the 16 bits are divided into 4-bit booth identification signals and 12-bit data signals. One of the inexpensive multiplexing circuits for input is shown in Figure 3. Figure 4 shows the output circuit. In the single-booth environment, these circuits are not required and all 16 signal lines are available in the ON, OFF, and INPUT statements.

THE SYSTEM GENERATION

The TYMES system source programs are written in the RT-11 MACRO assembler language. The user can create a customized version of TYMES by assembling each of the RTM, the compiler, and the loader. The system parameter source file defines the types of CPU and digital interface, the numbers of users, booths and

terminals, blocks, the system clock rate, the internal functions to be included, and allied items. This parameter file is assembled with other TYMES source program files.

Availability

At present, the authors are planning registration of TYMES with the DECUS program library being made for the purpose. Further information is available, upon request, from Tetsuo Takigawa, Department of Behavioral Science, Faculty of Letters, Hokkaido University, Sapporo 060, Japan.

REFERENCES

- MILLENSON, J. R. On-line sequential control of experiments by an automated contingency translator. In B. Weiss (Ed.), *Digital computers in the behavioral laboratory*. New York: Appleton-Century-Crofts, 1973.
- MILLENSON, J. R. System developments in the ACT language: Promises and problems. *Behavior Research Methods & Instrumentation*, 1975, 7, 165-173.
- SNAPPER, A. G., & KADDEN, R. M. Time-sharing in a small computer based on a behavioral notation system. In B. Weiss (Ed.), *Digital computers in the behavioral laboratory*. New York: Appleton-Century-Crofts, 1973.

(Received for publication November 16, 1979;
revision accepted October 6, 1981.)