SESSION XIII

DATA GENERAL USER'S GROUP MEETING AND CONTRIBUTED PAPERS

JOHN F. KING, *Illinois State University, President*

EMPP: An extensible multiprogramming system for experimental psychology

KENNETH J. BURKHARDT, JR.

University of Washington, Seattle, Washington 98195

EMPP is a problem-oriented language developed to simplify programming of psychological experiments on a minicomputer. An important feature of EMPP is its ability to be modified as the need for more complex hardware or software arises. The basic EMPP language is summarized and exemplified in a typical experiment.

EMPP, an extensible multiprogramming system for experimental psychology, is a problem-oriented language (POL) developed to simplify programming of psychological experiments. The system is capable of simultaneously controlling a number of experimental stations (eight, presently) where the same experiment can be performed synchronously or asynchronously at each of the stations or where stations may be totally independent in the sense that as many as eight unrelated experiments can be performed simultaneously. The EMPP language is not directly related to conventional (data processing) programming languages, but was developed primarily for the class of applications encountered in the psychological testing laboratory at the University of Washington.

Although there have been many attempts to develop specific POLs for use in computer-controlled psychological testing laboratories (Getty, 1975; Pinkus & Gregg, 1973; Pitz, 1975), no one language has gained universal acceptance. This lack of a standard language is a function of economics, differences in applications, and differences in physical hardware between laboratories. Rather than attempt to modify one of the existing languages to operate in the environment at the University of Washington, a decision was made to develop a language that could make use of our specific

Development of the EMPP language was supported by the National Institute of Mental Health, Grant MH21795, Earl Hunt, principal investigator, and the National Institute of Education, Grant NIE-6-74-0104, Clifford Lunneborg, principal investigator, to the University of Washington. Special thanks are due John Palmer, who was instrumental in the implementation of the EMPP operating system.

hardware capabilities to simplify programming of experiments. The language, however, is sufficiently modular to be implemented on other similar systems with a minimum of difficulty.

This paper summarizes the important features of the EMPP system by (1) describing the hardware facilities of the University of Washington computer-controlled psychological testing laboratory, (2) outlining the major design goals that led to the implementation of the system, and (3) exemplifying the EMPP language through a sample experiment.

HARDWARE FACILITIES

Figure 1 is a block diagram of the hardware facilities of the psychological testing laboratory at the University of Washington. The laboratory system is controlled by a Data General NOVA 820 minicomputer equipped with 16K words of 16-bit 800-nsec core memory. The NOVA has a primitive priority interrupt system, high- and low-speed direct memory access facilities, a programmed real-time clock, and integer multiply and divide hardware. Programs and data are stored on either one of two 2.5-megabyte moving arm disks or on one of two LINC tape drives.

Communication with individual experimental subjects is through an intelligent experimental station controller. The station controller oversees the operation of eight independent experimental stations. Each station contains a display and a console interface. The display consists of a Tektronix 604 monitor scope equipped with a high-speed P11 phosphor and a 2K x 16-bit MOS

Figure 1. Hardware facilities at the University of Washington Psychological Testing Laboratory.

character memory. The character memory is loaded by the station controller under command from the NOVA. The station controller converts ASCII (or special graphic) characters to the required x,y display coordinates and stores the coordinates in the character memory. Once a display command is issued, there is no further burden on the NOVA. Displays are refreshed at either 50 or 100 Hz under control of the station controller. Refresh operations at a particular station are independent of operations in both the NOVA and in other experimental stations.

The console interface allows the controller to recognize subject responses from special devices such as voice keys or telegraph keys or from one of eight keys located on the console interface enclosure. Eight digital outputs are available at each console for latching relays or solid state switches to perform functions such as controlling random access slide projectors or turning on tape recorders.

A serial interface connects the NOVA to an IMPLAC PDS1 graphic display, a 165-cps Centronics 301 line printer, and a Teletype equipped with a low-speed paper-tape punch and reader.

SUPERVISORY SOFTWARE

During program development, the laboratory computer system is controlled by the Ball Computer Corporation disk operating system (DDOS 1.6). This vendor-supplied software allows users to edit programs or data, to compile EMPP programs with the EMPP compiler, and to analyze experimental results using analysis programs written in FORTRAN or BASIC. DDOS 1.6 also contains a powerful diagnostic program (the "debugger") which has proved quite useful for system program checkout.

Experiments are run under control of the stand-alone EMPP operating system (EOS). EOS is an interpreter-based multiprogramming system that supports a maximum of eight subjects operating independently. Subject scheduling, input/output operations, and hardware and software diagnostics necessary to insure proper

system operation are all transparent to the user. User programs issue requests to EOS, which physically performs the desired operations. The structure of the EMPP operating system and its relationship to user programs are described in greater detail below.

SYSTEM DESIGN CRITERIA

Prior to the development of the EMPP operating system (early 1975), experimental control programs were written strictly in assembler language. Several input/output drivers were available, but they were quite primitive and fairly difficult to use. Program development was inordinately slow, requiring skilled programmers to implement even simple experiments. Furthermore, due to a lack of scheduling capabilities, the only experiments that could be programmed were experiments that required all subjects to perform basically the same operations at the same time. Under this system, if an individual subject required a particularly long time to respond to a stimulus, other subjects would encounter corresponding delays in the presentation of new stimuli.

The EMPP system was designed in an attempt to alleviate these problems. The primary design goal was to produce a system with the following characteristics:

(1) Simplicity. Psychologists with minimal programming experience should be capable of writing experimental programs in a short period of time.

(2) Hardware control capability. It should be possible to display data on CRTs, record subjects' responses at keyboards, and make use of other system hardware facilities.

(3) Asynchronous operations. It should be possible to operate experimental stations independently.

(4) Multiplicity of experiments. The system should be capable of simultaneously performing more than one experiment.

(5) Extensibility. It should be relatively easy to add new software and hardware facilities to an existing system.

Extensibility is perhaps the most important feature of this system. It is difficult (if not impossible) to predict all operations required by a general set of psychological experiments. In conventional programming systems, situations often arise where a desired function cannot be directly performed in the language, e.g., permuting a list or performing a string search in FORTRAN. The programmer is then forced to manipulate the language in an attempt to implement the desired function. In some instances, this is time consuming; in others, the function simply cannot be implemented. In the EMPP system, if there is a desire to perform a "new" function that cannot be easily implemented in the existing language, the function is coded in machine language and added to the basic set of EMPP commands available to the programmer. A

secondary benefit of extensibility is that once a technique is devised for performing a particular function, that function is readily available to all system users. Rediscovery of algorithmic solutions is avoided to a large extent.

SYSTEM OVERVIEW

EMPP was designed in a top-down fashion. An attempt was made to determine what types of experiments would be performed in the laboratory and then a set of tools was designed for implementing these experiments. Recognizing that it would be impossible to completely specify all experiments to be run in the laboratory, an attempt was made to modularize EMPP to the extent that any portion of the system could be modified without affecting the remainder. (On the simplest level, a modification might involve adding a new input/output device or modifying a scheduling algorithm. On the most complex level, modification might be implementing EMPP on a different computer.)

The EMPP system should be viewed as evolutionary. EMPP started with a basic set of primitive commands and has grown with the need for more complex operations. This ability to grow makes it difficult to rigidly define the EMPP system. Even though the system may change, however, the basic system concepts will remain the same.

Structure of the EMPP System

The EMPP system consists of five extensible program types. (1) Subject programs consist of a series of commands written in the EMPP language that control the execution of individual experiments. An individual subject program can control one or more experimental stations. (2) The compiler is a FORTRAN program that converts subject programs to the meta language recognized by the EMPP operating system. (3) The master system scheduling and control program schedules the execution of subject program commands and passes control to the experiment macros. (4) The experiment macros are a series of macro subroutines (approximately 25) written in NOVA machine language that perform operations requested by the subject programs. (5) The system subroutines are a series of general purpose subroutines used to simplify the coding of experiment macros. System subroutines typically perform operations required by several experiment macros, e.g., argument address calculations, list processing functions, etc.

In a sense, the EMPP software system can be viewed as a pseudocomputer, where the instruction set of the pseudomachine is a higher level language defined by the experiment macros.

The primary emphasis in this paper is on the EMPP language as used in the subject programs. While the size and operation of subject programs vary significantly

between experiments, the scheduler, experiment macros, and system subroutines are the same for all experiments. These three program types comprise the resident EMPP operating system (EOS). The relationship of the EOS modules to subject programs is depicted graphically in Figure 2. Each subject has a unique data storage area for recording experimental results. Data is recorded in these areas via commands issued by the subject program. The actual data transfers are physically performed by the experiment macros or the system subroutines. When a subject program issues a command, the scheduler causes control to be passed from the subject program to the appropriate experiment macro. The experiment macro either performs the requested operation or transfers control to a system subroutine. On completion of the requested operation, control is returned to the subject program.

Although there are unique data areas for each subject, the system requires only one copy of the subject program to be located in memory.

THE EMPP LANGUAGE

EMPP subject programs consist of a series of requests to the EMPP operating system. The general format of a request is

Label: Command Argument List ;Comment

where the number of operands on the argument list is a function of the command. Labels, commands, and variables are alphanumeric strings that begin with a letter. Operands can be variables, address variables (a label followed by a period), or literals (described below).

There are six types of commands in EMPP: (1) data definition, (2) arithmetic operations, (3) program branch control, (4) input/output control, (5) real-time control, and (6) subject or parallelism control.

Data Definition

Subject programs define variable data types through the use of nonexecutable data definition commands. In

Figure 2. Overview of the EMPP system.

the EMPP language, variables may be dichotomized along three dimensions: global or local, scalar or vector, and integer or alphanumeric.

Global variables can be accessed by all subjects in an experiment. Local variables are unique to each subject. For example, a global variable might refer to the number of trials to be performed during an experiment. A local variable might be the reaction time of a subject to an experimental stimulus. Variables are considered global by default. The **LOCAL** command defines a variable as local and increases the subject storage accordingly, e.g., **LOCAL x,y**.

Subject programs can include simple variables (scalars) or singly dimensioned arrays (vectors). Both scalars and vectors can be either global or local. A variable is defined as a vector through the use of the **DIMENSION** command, e.g., **DIMENSION x(5), y(5)**.

All EMPP variables may be either numeric (single- or double-word integers) or alphanumeric (ASCII character strings). A variable is assumed to be numeric unless a **STRING** command specifies that the type is alphanumeric, e.g., **STRING x,y**.

EMPP also recognizes numeric and alphanumeric literals. A numeric literal is an integer constant (e.g., 123) and an alphanumeric literal is an ASCII character string (e.g., "This is an alpha literal").

If a variable is not defined by a data definition command, it is assumed to be a numeric global scalar.

Arithmetic and String Manipulation Commands

EMPP contains a full set of arithmetic and string manipulation instructions. Commands exist for performing conventional arithmetic operations (multiplication, division, subtraction, addition), logical operations (and, or, not), and special arithmetic operations such as permuting lists and generating random numbers. String manipulation facilities include commands for performing string searches, string replacements, concatenations, truncations, etc., for both variable- and fixed-length character strings.

Program Control Commands

Numeric compare (**CMP**), compare to zero (**CMPZ**), alphanumeric string compare (**ACMP**), decrement and test (**DECTST**), and increment and test (**INCTST**) commands are used for conditional branching and loop control. In conjunction with arithmetic instructions, these commands allow magnitude as well as logical and character string comparisons.

GOTO and **CGOTO** (computer **GOTO**) allow unconditional and vectored branching. **SUBJMP** and **SUBRET** allow for a single level of subroutine branching.

Input/Output Commands

One of the primary design goals of the EMPP system was to provide easy to use, yet powerful, input/output facilities. Although these two characteristics are not

necessarily compatible, the EMPP system has been fairly successful in achieving this goal through the use of formatted input/output statements.

Input/output requests have the format:

I/O command, format label, argument list

where the format statement referred to by the format label is a nonexecutable description of the desired operation. Input or output to any device connected to the system can be rigidly defined or relatively free of fixed format constraints. The system has the capability to process binary, octal, and decimal integers as well as fixed- and variable-length character strings. Input of string variables requires that an associated integer array be defined sufficiently large to contain the maximum number of characters expected for the variable. (Alphanumeric strings are stored two characters per word.) Commands are also available for positioning displays, loading and clearing display character memories, and for starting and stopping display operations.

Real-Time Control Commands

While many of the instructions encountered for previous command types are fairly typical of standard data processing languages, it is the ability to maintain real-time control over the computer and the experimental apparatus that separates real-time control languages such as EMPP from conventional data processing languages. EMPP has a set of commands that allows a subject to be suspended either as a function of time or as a function of an event external to the computer.

WAIT allows the suspension of a subject for a predetermined period of time (from 1 msec to 1 min).

WAITK (wait for keyboard response) allows the suspension of a subject until a desired response occurs and records the time of the response.

WAITRT (wait for response or time) allows the suspension of a subject until either a desired response is encountered or a time limit is exceeded. **WAITRT** will take different returns as a function of subject response within the desired time frame.

Subject or Parallelism Control

EMPP experimental programs are written as if only a single subject could participate. At run time the EMPP operating system determines which stations have subjects and causes a copy of the subject program to be executed at each of the appropriate stations. The programmer can specify whether a program segment will be executed once for all subjects or once by each subject. The "global" mode of execution is used primarily for initialization of the data base and for experimenter communications. The experiment generally takes place in the "local" mode.

A subject program is put in the local mode by using the **FORK** instruction and is returned to the global

```

DIMENSION STIM(10),RT(2),GSTIM(10)                ;VARIABLE DEFINITIONS
LOCAL SEED,STIM,RT,TRIAL,DRESP,J,SIZE,I
STRING GSTIM,PROBE
      READTTY INFORM. GSTIM(0),GBUF(0)              ;READ IN STIMULUS LIST
      READTTY SEEDY. GSEED                          ;INPUT RANDOM SEED
      FORK                                           ;START SUBJECTS RUNNING
      ASSIGN 9. TO I
      ASSIGN GSEED TO SEED
INITIAL:  ASSIGN I TO STIM(I)                        ;SET UP LOCAL STIM. VECTORS
          DECTST I INITIAL.                          ;LOOP UNTIL STIM. SET INITIALIZED
          ASSIGN I TO STIM(I)
          ASSIGN 0 TO TRIAL                          ;INITIALIZE TRIAL COUNTER
          WRITDISK OPEN. "KB:TRIAL DATA"            ;OPEN OUTPUT FILE
START:    PERM STIM(0),10.,SEED                     ;PERMUTE STIMULUS SET
          RAND SEED,5,SIZE
          ASSIGN SIZE TO I
LEARN:    ASSIGN STIM(I) TO J                        ;LEARNING SEQUENCE
          WRITSCOPE DFORM. STIM(J)                  ;DISPLAY STIMULUS
          WAIT 1200.                                ;AND WAIT 1.2 SEC.
          WRITSCOPE ERASE.                          ;CLEAR SCREEN
          DECTST I,LEARN.
          RAND SEED,100.,X                          ;TEST SEQUENCE
INSET"    CMP X TO 50. INSET. NOTINSET. NOTINSET
          RAND SEED,SIZE,I                          ;SELECT PROBE INSET
          ASSIGN 1 TO DRESP
          GOTO PROBE.
NOTINSET: ADD 1 TO SIZE STORE I                      ;SELECT PROBE NOT IN SET
          ASSIGN 400. TO DRESP
PROBE:    WAIT 2000.                                ;DELAY 2 SEC.
          ASSIGN STIM(I) TO J
          WRITSCOPE DFORM. "PROBE" GSTIM(J)          ;DISPLAY "PROBE",SAMPLE PROBE
          WAITR 5000.,RT(0),RESP,TIMEOUT.,GDRSP.    ;ALLOW 5 SEC FOR ANS
GDRSP:    CMP DRESP TO RESP TIMEOUT. TIMEOUT. CORRECT.;SUBJECT RESP. CK.
CORRECT:  WRITSCOPE ERASE.                          ;GOOD RESPONSE, ISSUE FEEDBACK
          WRITSCOPE FEEDBK. "CORRECT",RT(0)
          GOTO DNTRIAL.
TIMEOUT:  WRITSCOPE ERASE.
          WRITSCOPE DFORM. "INCORRECT"
DNTRIAL:  PACK DRESP RESP IN DRESP                  ;SET RESP FLAG
          WRITDISK RECORD. RT(1),GSTIM(J),DRESP,SIZE
          INCREMENT TRIAL
          WAIT 2000.
          WRITSCOPE ERASE.
          CMP TRIAL TO 100. DONE. START. DONE.
DONE:    WRITSCOPE DFORM. "ALL DONE"
          JOIN
          EXIT                                       ;RETURN TO SYSTEM,ALL DONE
INFORM:   FORMAT(W"STIMULUS LIST"V<10.>A<0>)       ;FORMAT DEFINITIONS
SEEDY:    FORMAT(W"SEED" I<5>L<1>)
DFORM:    FORMAT(W"SEED" I<5>A<0>)
ERASE:    FORMAT(KP)
FEEDBK:   FORMAT(L<5>X<5>A<0>L<1>X<5>D<5>)
OPEN:     FORMAT(G<0>)
RECORD:   FORMAT(F<0>I<5>A<0>I<4>I<4>I<2>)
END

```

Figure 3. Sternberg (1966) memory scanning experiment.

mode by the JOIN command. When the JOIN command is issued, the program will not continue operation until all subjects have completed the previous program segment. A subject program can reference only global variables when in the global mode, while either global or local variables can be accessed in the local mode.

Compiled Subject Programs

EMPP requests are generally compiled as run time commands to the EOS scheduler. The compiler translates the EMPP commands and generates a program data base recognized by the run time system. The scheduler selects the proper interpretive commands and

passes control to the appropriate experiment macros. It is possible, however, to imbed NOVA assembler language in EMPP programs by bracketing the desired assembler instructions with exclamation points. This escape to assembly mode is used primarily in situations where execution speed is extremely critical.

SAMPLE PROGRAM

Figure 3 is a listing of an EMPP program used to implement Sternberg's (1966) memory scanning experiment. Sternberg's paradigm was coded both as a means to compare EMPP to other psychological programming languages and as a means to exhibit some of the strengths and weaknesses of the EMPP language. The format of the experiment is similar to the APCOL implementation described by Pinkus and Gregg (1973).

Experimental input consists of a list of 10 alphanumeric characters that correspond to the experimental stimuli. The input data is read into an alpha vector GSTIM. Each subject has a local vector of indices which points to the global data base. At the start of each trial, the local vector is permuted to obtain a random ordering of the input data. A random number of stimuli are then selected and presented to the subjects as a learning set. After a short delay, a probe stimulus is selected and the subject's response, reaction time, and experimental conditions are recorded on disk.

Several of the advantages and disadvantages of the EMPP language should be obvious from examining the sample program. Although the program is not strictly readable in the natural language sense, it is certainly easier to follow than a machine language implementation. There are a few simple rules to learn, but once these have been mastered, programming is relatively simple.

The primary weakness of the EMPP language is that both loop control and arithmetic operations are awkward. Extending the compiler to recognize "IF" statements and "DO" type procedures would certainly simplify programming. Mathematical operations would also be simplified if the compiler had the capability to recognize algebraic ("FORTRAN-like") assignment statements. On the other hand, the present English language approach is more natural in situations where calculations are kept to a minimum. This Polish prefix style was selected as the easiest technique to use for

implementing the compiler, but in programs that perform a large number of arithmetic calculations, it is unwieldy. Present plans call for extending the EMPP compiler to recognize both natural language and algebraic statements.

SYSTEM EVALUATION

The EMPP system stands up quite well to its initial design goals. With the completion of the EMPP compiler, experiment writing is fairly simple even for nonprogrammers. Experiment programming time can now be expressed in terms of hours rather than days.

The EMPP language has simplified the task of experiment programming in several important ways. EMPP permits the convenient use of arrays and constants including alphanumeric variables. It allows the specification of variables that refer to the experiment as a whole as well as to individual subjects. From the standpoint of the programmer, multiple subjects can be run as easily as a single subject. System initialization (including specification of which stations have subjects) is performed by the system without any action by the programmer. Finally, EMPP is self-documenting in that procedures are expressed in natural terms rather than in terms of registers or storage locations.

The language has proved to be quite powerful, allowing the development of several fairly complex experiments that had previously been avoided due to programming complexity. Extensibility has proved to be a valuable tool in allowing both the basic software and hardware facilities of the EMPP system to grow from our initial system.

REFERENCES

- GETTY, D. J. The PEPL system for control of experiments by a PDP-8 computer. *Behavior Research Methods & Instrumentation*, 1975, 7, 131-136.
- PINKUS, A. L., & GREGG, L. W. APCOL: A programming system computer-controlled psychology laboratories. *Behavior Research Methods & Instrumentation*, 1973, 5, 165-172.
- PITZ, G. F. Building a programming language for a small computer: Reinventing the wheel. *Behavior Research Methods & Instrumentation*, 1975, 7, 42-46.
- STERNBERG, S. High-speed scanning in human memory. *Science*, 1966, 153, 652-654.