# E.S.P. The Elliott Simulator Package

*By* J. W. J. Williams

**This paper describes informally a method for programming simulations in ALGOL, using a standard package of procedures (E.S.P.) to control the simulation.**

**A novel feature is the means of specifying the sequence of events, which permits multiple occurrences of any event with independent sets of parameters. This involves the use of sophisticated sorting routines and dynamic storage allocation by the E.S.P. procedures.**

The behaviour of complicated systems, involving many interrelated processes, is often most conveniently studied by means of a model. The model is set up, and exposed to random demands, and statistics are collected which describe its response. This is the technique of simulation. The manipulation of the model usually involves only simple steps, but a great many steps are required if the results are to be statistically significant. For this reason it is usual to devise a numerical model, and employ a digital computer to manipulate it.

A number of programming systems have been devised to enable such models to be set up more readily. E.S.P., the Elliott Simulator Package, is such a system. It possesses some novel features, among them the fact that it comprises a set of ALGOL procedures.

One might question the need for special programming systems for simulation studies, when powerful general-purpose systems such as ALGOL are available. The answer is that the central difficulty of the problem is the control of the sequence in which the interdependent actions forming the model occur. If one attempts to write a simulation program using only a general-purpose language, one rapidly becomes enmeshed in the complexities of this sequencing control, which is not of great interest but nevertheless affords surprisingly fertile ground for minor errors. Moreover, mistakes here are liable to produce obscure effects, and are correspondingly difficult to eradicate.

The major task of a simulator programming system, then, is to deal with this sequencing problem. In addition, the user will need to program certain parts of his problem himself, and should be allowed to use some problem-oriented language. In the case of E.S.P. this language is of course ALGOL. Finally there are procedures which may be required by almost every user, such as random-number generators, and statistical routines, and these too should be built into the system.

### Structure of the model

When writing the program for a simulation study, the use of prefabricated units from any system necessarily imposes certain constraints. These constraints are best considered at the time when the model is designed, rather than at the detail programming stage. There is, in fact, a preferred type of model. In the case of E.S.P. it is thought that this type is general enough to allow any system of interest to be simulated, and that it is self-explanatory enough to be easily written and easily explained to others.

The system to be simulated may be regarded as a set of "actions" which manipulate a set of "objects." The objects might be the patients waiting in a doctor's surgery, or the medicines with which they depart. The actions in this case would be the arrival and departure of patients, and the beginnings and ends of consultations. In the model, all objects are represented by numbers—or possibly by lists of numbers—and all actions are represented by units of program which manipulate the numbers.

It is necessary to recognize actions as being of one of two possible kinds. There are actions which are pre-ordained to occur at a certain time—alarm clocks as it were—which are referred to as *delayed actions*. There are also actions which will occur as soon as some favourable combination of objects appears, and these are called *conditional actions*. It is possible to envisage hybrid actions which depend on several conditions, and also on time. Fortunately, hybrid actions can always be separated into a delayed action and a conditional action by making one of the conditions of the conditional action be that the delayed action has already occurred, and has set some suitable mark of its passing.

These concepts may be illustrated by an example. We may consider a doctor's surgery, shared by several doctors, any of whom are prepared to examine any patient. The objects of interest are free doctors, waiting patients, and examined patients.

The actions are:

(1) Whenever there is a free doctor and a waiting patient, both disappear.
(2) After some time, the doctor and an examined patient reappear.

Action number 1 is a conditional action, which occurs whenever the stated objects are present. Action number 2 is a delayed action, which occurs at a pre-ordained time. This time was, of course, pre-set by action 1, when the consultation commenced. The process by which a future occurrence of a delayed action is arranged will be referred to as "calling" that action.

In the program, it is the task of the E.S.P. to enter the various sections representing actions in the correct sequence, and correspondingly to adjust the value of a

variable representing time. Any section entered may call any other section, or itself. Provision is made for this to be done, and the subsequent behaviour of the sequencing routine will be modified in response to such calls. The details will be discussed below.

### Structure of the program

The program which is to perform the simulation will have a structure closely related to that of the model described above. The program will consist of several sections, each devoted to one action. It will be preceded by declarations of identifiers and arrays, needed to represent the objects to be manipulated, and further variables used to accumulate the records which are the results of the simulation. There will also be a section which sets up the initial state of the model. The whole will be embedded in an outer block containing the standard procedures of the E.S.P., so that these may be available when required—because of this the program requires an extra **end** which terminates this outer block.

The various sections concerned with actions must be obeyed in proper sequence. The sequencing routines must first examine a list of actions which have been called but have not yet been employed, select the earliest of these, and enter it. On completion of this action, it must examine the remaining called actions, and employ any which are called for the same time as the one just performed. After this, the conditional actions must be examined, and any which are possible performed, and the cycle then repeats.

Since only the writer of the simulation can decide what tests are needed in the conditional actions, these tests are his responsibility. The whole set of conditional actions is grouped together into a single section, in which each individual action is represented by a conditional statement. This grouping is possible, because the conditional actions are always examined together, so that there is never any need to break into the middle of the group.

To summarize, there is one section of program for each delayed action, and one section for the group of conditional actions. These sections are entered in an appropriate sequence under control of E.S.P., corresponding to the calls for actions.

The starting point of each section is given a label, and all these labels must be declared in a switch, with the label corresponding to the conditional actions section first, followed by the other labels in any order. The switch itself must be given a name, and the name *act* is suggested and will be used in the following description. Thus if there are three sorts of delayed actions, labelled *first*, *second* and *third*, and the conditional happenings are labelled *maybe*, the switch declaration will be

**switch** *act* := *maybe*, *first*, *second*, *third*;

Calls for actions are made in the following way. There is a procedure *call* (*i*, *t*) provided in E.S.P. This procedure is associated with the switch *act*, in the following way. The parameter *i* specifies which delayed action is to be called, and this is the *i*th delayed action in the switch list, which is to say, the section of program of which the label is the $(i + 1)$th in the list. Thus *call* (2, *t*) is a call for the delayed action with the label *second* and so on. The parameter *t* specifies the amount of 'time' which is to elapse before the action actually occurs.

When a section of program has been performed, it is necessary to progress to the section representing the appropriate next action, that is to jump to one of the labels of the switch *act*. It is the task of E.S.P. to decide which action is now required, and a procedure *next* is provided for this purpose. This is an integer procedure and is used to select one of the labels from the switch *act*. Each section of the simulation program ends with the statement

**go to** *act* [*next*];

This activates the procedure *next*, which scans a list maintained inside E.S.P. and selects the next action required. In this list, of course, actions are known by the parameter *i* from the call procedure. Consequently *next* may take the appropriate value to cause entry to the section of program required. The procedure *next* also causes entry to be made to the group of tests for conditional actions, at appropriate times. That is, *next* takes the value 1 at these times. This occurs whenever a delayed action has just been completed, and the first subsequent delayed action is due to occur at a later time.

Thus, to the simulation programmer, the statement **go to** *act* [*next*] is effectively the statement that the current action is complete and another section of program is to be entered, and the statement *call* (*i*, *t*) is the means of pre-arranging future actions. Also, the conditional actions will in general be obeyed as soon as the appropriate tests are satisfied.

### Parameters

It may be that, at the time when an activity is called, certain data is available which will be useful when the activity is eventually performed. For example, suppose that the doctors of our previous example are to be distinguishable for some reason. Then when a doctor and patient vanish, at the beginning of a consultation, we know which doctor it is, and when the doctor reappears later, it must be the same doctor. This might be arranged by having a large number of activities, effectively "doctor 1 reappears," "doctor 2 reappears," and so forth. This is rather clumsy. Instead we prefer a single activity "doctor *i* reappears" with a parameter *i*.

The manipulation of such procedures is accomplished with the aid of two integer arrays, *send* [0 : 15] and *get* [0 : 15], which are declared as part of E.S.P. If, just prior to a call, values are assigned to elements of *send*, then, just prior to entry into the program representing the called action, these values will be assigned by

E.S.P. to the corresponding elements of *get*. More precisely, *send* [0] must be assigned a value *n* representing the number of parameters to be transmitted, and *send* [0] to *send* [*n*] inclusive will be so transmitted. A side-effect of the procedure *call* is to restore *send* [0] to zero.

Note that in the example above, several consultations may be in progress concurrently, each with an associated call for "doctor *i* reappears." The various times and parameters *i* will be so manipulated by E.S.P. that entry occurs to the activity at the appropriate times, and at each entry the correct associated parameter *i* will be present. That is, the several calls do not interfere with each other in any way.

### Example

The basic ideas of E.S.P. have now been sufficiently explained to allow an example program to be given. In order to avoid complication the bare bones of the simulation are considered; only trivial statistics will be collected, and no random numbers will be used. Facilities for such purposes are of course available in E.S.P., but a description of them will be postponed to a later Section.

The system to be simulated is a clinic, employing four doctors. The doors of the clinic are initially open, and remain open for three hours, during which period patients arrive at a rate of one every seven minutes. After the doors are closed, any further patients are turned away. Any doctor may deal with any patient. Consultations last for nine minutes. It is desired to find the total number of patients seen by each doctor, assuming that the doctors are numbered one to four, and that if more than one doctor is free for consultation then the lowest-numbered doctor accepts the next patient to arrive.

The required program is given in full on the opposite page.

### Additional facilities

E.S.P. also offers facilities for the generation of random numbers and the construction of histograms.

All the random number routines require an integer variable which is used to store the latest value used by the basic random number generator. This is usually given the identifier *u*. This technique allows independent streams of random numbers to be produced, each using its own identifier in place of *u*.

*Random* (*a*, *b*, *u*) is an integer procedure, which takes a random value *x* so that $a \leqslant x \leqslant b$ and *x* is uniformly distributed.

*Negexp* (*t*, *u*) is an integer procedure which takes a value binomially distributed with mean *t*.

*Normal* (*m*, *v*, *u*) is an integer procedure which takes a value normally distributed about mean *m* with variance *v*.

Histograms are compiled in single dimensioned arrays. Certain elements of the array are used to contain the number of cells, cell width, etc. This avoids the need to re-specify these parameters when referring to the histogram. Typical routines are:

| | |
|---|---|
| *Hstset* (*A*, *n*, *l*, *w*) | which prepares array *A* to receive a histogram of *n* cells, lower bound of first cell *l*, cell width *w*. |
| *Hsput* (*A*, *x*) | which inserts the variate *x* into the histogram array *A*. |
| *Hstprint* (*a*) | which prints *A* in standard form. |
| *Hstsamp* (*A*, *u*) | an integer procedure which takes a value *s*, so that the probability that *s* lies in any cell of the histogram is proportioned to the count in that cell. |

In addition to the above built-in facilities, algorithms have been devised for manipulation of queues, which can be included if they are needed. This avoids the wastage of storage space involved in maintaining all possible routines of interest in the standard package.

### Implementation

A version of E.S.P. has been coded and is operational on the Elliott 503 and 803 computers. Work is currently in progress to improve the routines.

### Acknowledgements

The programs described were devised after study of various other simulation systems, as mentioned in the references below.

My thanks are also due to Mr. C. A. R. Hoare, of Elliott Brothers, for valuable assistance with the ALGOL technique employed.

This article is published by kind permission of Elliott Brothers (London) Limited.

### References

BUXTON, J. N., and LASKI, J. G. (1962). "Control and Simulation Language," *The Computer Journal*, Vol. 5, p. 194.

KELLEY, D. H., and BUXTON, J. N. (1962). "Montecode—an Interpretive Program for Monte-Carlo Simulations," *The Computer Journal*, Vol. 5, p. 88.

TOCHER, K. D. (1960). *Handbook of the General Simulation Program.* United Steel Companies Dept. of Operational Research Report Number 77/0RC 3/Tech.

```
begin   boolean open; comment open is true for 180 minutes, then false;
        integer patients; comment this is a count of the number of patients waiting;
        boolean array docfree [1 : 4]; comment docfree [i] is true if doctor No. i is idle;
        integer array seen [1 : 4]; comment seen [i] is a count of patients seen by doctor No. i;
        switch act := consult, arrival, finish, shutdoor, results;
        begin integer i;
              comment this is the entry point of the program, and this section sets initial conditions. Prepare is an
                      E.S.P. procedure which sets initial states inside E.S.P.;
              prepare;
              open := true;
              patients := 0;
              for i    := 1 step 1 until 4 do
                  begin docfree [i] := true;
                        seen    [i] := 0;
                  end;
              call (1, 2)     ; comment the first patient will arrive in 2 minutes;
              call (3,180)    ; comment shut door in 180 minutes;
              call (4, 1000) ; comment output results after 1,000 minutes, i.e. when all other activity has ceased;
              go to act [next]
        end   this completes the initialization. The procedure next will cause the appropriate section of program to
              be entered. In fact, on this occasion, this will be the section labelled arrive, in response to call (1, 2);
arrive: begin comment this section simulates the arrival of a patient. It also calls itself, so that a further patient
        will arrive in due course. The process stops when open becomes false;
        if open then begin patients := patients + 1;
                           call (1, 7)
                     end;
        go to act [next];
        end on this first occasion next will select the conditional action, consult;
consult: begin comment this is the set of conditional actions, which starts consultation whenever a doctor and patient
        are both available;
        integer i;
        for i    := 1 step 1 until 4 do
                    if docfree [i]  patients ≠ 0 then
                    begin docfree [i] := false;
                          patients := patients − 1;
                          comment the doctor and patient have disappeared, and we must now engineer the doctor's
                                  reappearance;
                          send [0] := 1;
                          send [1] := i;
                          call (2, 9);
                          comment we needed to transmit the doctor's number i, to enable us to make the correct
                                  doctor reappear;
                          seen [i] := seen [i] + 1
                    end;
        go to act [next]
        end the consultation thus started will be terminated by the action labelled finish in due course;
finish: docfree [get[1]] := true;
        comment this action restores the doctor after the consultation started in consult. The number of the doctor
                is get [1], corresponding to the send [1] of the action consult;
        go to act [next];
shutdoor: open := false;
        comment this action prevents the entry of further patients, since open is tested by the action arrive;
        go to act [next];
results: begin integer i
              for i := 1 step 1 until 4 do print seen [i];
              comment this is one form of output statement in Elliott ALGOL. More powerful output statements
                      are also available;
              go to act [next]
         end  In fact there will not be any more called events at this stage, and the E.S.P. will output a message to
              this effect, and then stop.
end   of example program which must be followed by an extra
end   to close the surrounding block containing the E.S.P. procedures.
```