Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

SIMULATING WITH ACTIVITIES USING C.A.P.S./E.C.S.L.

(THE BRITISH APPROACH TO DISCRETE-EVENT SIMULATION)

Alan T. Clementson
Department of Engineering Production,
University of Birmingham,
England, B15 2TT

ABSTRACT

This paper is designed to illustrate the British Approach to Simulation and how it would be used in conjunction with the CAPS/ECSL package. There are several features of this approach which are significantly different to the usual American approach, most notable of which is the use of the "Activity Structure" rather than the Event or Process Bases. It was from this structure that many of the recent discrete simulation developments originated, for example program generators and visual interactive modelling. The approach of the paper is to describe the steps which a typical user would follow in developing a model to solve a problem using simulation. The CAPS/ECSL system is designed to assist as many stages of this process as possible, claiming to be more complete than competing systems. It is hoped that two new developments will be described in the verbal presentation.

1. INTRODUCTION

This paper is designed to introduce the British Approach to discrete simulation as it would be followed when using the CAPS\ECSL package. It is presented in the form of a guide to a potential new user of the system, except that, since it is being presented at a conference of simulation experts, a knowledge of simulation is assumed.

The steps are:-

a) Formulation of the model with an Activity Cycle Diagram.
b) The analysis of the diagram to discover which are the critical parts of the model
c) The analysis of data to be used as input to the model.
d) The construction of a program.
e) The validation of the model.
f) Analysis of a pilot run and the design of an experiment.
g) Running the experiment.
h) Analysing the experiment.
i) Selling the results.

2. ACTIVITY CYCLE DIAGRAMS

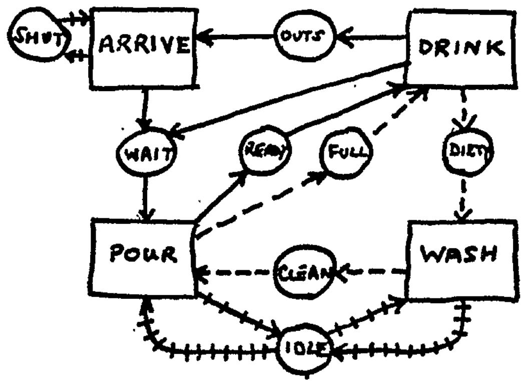
Activity cycle diagrams are essentially diagrams showing the progression of states through which entities pass as time passes. They are usually drawn with each entity represented by a different colour. Two different types of state must be distinguished. These are called "activities" and "queues". An activity is represented by a rectangle drawn in a neutral colour. A queue is represented by a circle drawn in the colour assigned to the entity type concerned. The states for one entity type are connected by arrows (i.e. lines with a direction indicator) drawn in the colour assigned to the entity type. Each state is given a name which is written in the rectangle or circle.

The following four definitions are the basis of the method:-

a) An entity is an "indivisible" element of the system.
b) Entities are organised into groups such that all members of the group obey the same <u>rules</u> of behaviour. Such a group is called a "class" (and represented by a colour).
c) An "activity" is a state for which the duration can be calculated at the beginning of the state. (Note: the formula for this calculation can be as complex as required and often involves taking a random sample.)
d) A "queue" is a state for which the duration is not able to be calculated at the start of the state, typically because it depends on the state of other entities which are also involved in the next activity. Thus, queues are delays between activities.

From these definitions we see that as the lines for a class are followed by any legal route, they must always pass alternately through queues and activities.

By convention, the diagram for one class, which is called a "cycle", is closed. Where this is not naturally the case, it is achieved by adding an artificial queue



containing all entities which are currently not in the system. The class size must be chosen to be larger than the largest possible number of entities in the system at one time.

The diagram above is a very trivial example, that of a bar. This is used since it is not possible to adequately represent a realistic diagram without the use of colour. This printed version of the paper is based round this trivial example, whereas the verbal presentation will be based around a "small but realistic" flexible manufacturing system.

It should be noted that the Activity Cycle Diagram has been the basis of simulation in the UK since the late fifties, but that it can be viewed as an extension of the "Petri-Net". The prime virtue of the activity approach is that it "decomposes" the system complexity in a way that simplifies the modelling of that system.

3. ACTIVITY CYCLE DIAGRAM ANALYSIS

Enitities travel through the sequence of states represented by their cycles. Suppose, for the moment that each cycle consists of a pure cycle - i.e. it has no branches. Then, suppose that the total of the average durations of each of the activities in the entity's cycle is obtained. If this type of entity were never delayed by any other type of entity, this total is the average time it would take for an entity to travel round the cycle - this we call the "cycle time". The cycle time divided by the number of entities of this type, which is called the "cog-time", is the average undelayed time between repeats of any activity in the cycle.

It is now possible to view the complete diagram as analogous to a train of gears, the activities represent places where the cogs of two (or more) cycles interlock. At such an interlock, both "wheels" must move at the same speed, measured in cogs per unit time. Thus, the system cog-time is the maximum of the cog-times of the various entity types. This represents the rate at which the system would work if all activities in the system were deterministic. (Although not described here, it is easy to remove the restriction that the cycles be without branches.) The entity type with the maximum cog-time is the "bottleneck". By dividing the cogtimes of each entity type by the system cog-time we get the "criticality" of each entity type.

It is well known that the effect of randomness in queueing systems is always to slow the system down, but the reduced speed cannot be easily estimated without performing the simulation. However, it is clear that the higher the criticality of the non-bottleneck entity types, the greater will be the effect of the randomness.

Suppose the above calculation were now repeated with the mean durations for those activities which are not in the cycle for the bottleneck entity replaced by their maximum durations. It is clear that, only those entities whose cogtimes are now greater than the original cogtime of the bottleneck entity can ever be even temporarily critical. By experimenting with different numbers of entities of the various types, it is possible to narrow down the "design" of the system.

The value of this analysis is two-fold. It gives upper (using the means) and lower (using the maxima) bounds for the system speed. This will restrict the range of the experiment that has to be performed. Secondly, it is clear that the accurate representation of those entities which cannot even become temporarily critical is not necessary. In particular, this will enable the costs of data collection to be trimmed. For critical activities a complete accurate probability distribution of duration is required. For semi-critical activities perhaps it is only necessary to have mean and variance correct, while for non-critical activities the use of a fixed mean may be sufficient.

The ECSL system contains an element designed to assist in the above analysis with a "spread-sheet" like tool.

4. INPUT DATA ANALYSIS

Most simulation languages allow the user to use histograms for the sampling of input random number distributions. In the case (at least) of activity durations, it is to be expected that these distributions ought to be capable of being represented by one of the theoretical distributions. ECSL contains a routine for the fitting of theoretical distributions to collected data. The output from the procedure is the formula to be used. It is well understood by statisticians, that the use of the proper underlying distribution is an important aid in variance reduction.

Secondly, it is important that the different points in a model which use random numbers should be properly independent of each other. ECSL contains a help facility to analyse the proposed set of "seeds" to ensure that they are not "neighbours" in the sequences.

5. CONSTRUCTION OF THE PROGRAM

CAPS is an acronym standing for "Computer Aided Programming of Simulation". The program called CAPS operates in a user friendly conversational mode with a user who supplies the problem definition by his responses. At the end of a session (typically from 15 minutes for a simple problem up to an hour or more for a large problem), CAPS generates an ECSL program that contains the logic as conveyed by the user during the session. This program will almost never fail to compile and execute. This, however, only means that a valid program has been generated - not that it is the correct one. Thus the user should never become complacent: an unrealistic logic might have been entered as part of the CAPS session. Alternatively the user may have omitted some details of the problem with a view to enhancing the generated program using the computer's normal editing facilities.

Before approaching the computer, the user will need to have prepared for the conversation. The following is a check list of the things that should have been prepared:-

a) A complete detailed activity cycle diagram. It is recommended that this is drawn using different colours to represent each entity type. Each state, activity or queue, must be given a different name.

b) The maximum number of each type (or class) of entities to be used in the simulation. This number must not exceed 999. In some cases the "theoretical number is infinite - such as the CUSTOMers in a bar. In these cases the user must guess a safe upper bound to the number that might be in the system at any given time. (Note that the number chosen is NOT critical, the program has a built in check that it was sufficient and it is easy to change it later.)

c) Optionally, the queue disciplines for any queues which are not FIFO (First-In-First-Out). Particular attention should be paid to queues where the choice of entity is restricted by the need to satisfy rules of matching between the entities involved in an activity. (Note that the system provides a wide variety of queue disciplines which are capable of handling quite complex scheduling ideas.)

d) The formula for the duration of each activity. If it is not constant, then it is likely to be a sample from a given probability distribution.

e) An idea of which class sizes are likely to be the subject of experiment and the range of values which may be required. (This, of course, is obtained from above)

f) A decision as to which queues are to be recorded and in which way (length of queue, delay time, utilisation). In selecting this data, it is desirable in the first instance to produce a time series of the queue length of a couple of significant queues. This output should then be analysed as discussed below to determine the appropriate run-in period and simulation duration for the main experiment.

g) The chosen length of the simulation including any run-in period. (In the first instance this is probably zero, to be modified after the analysis mentioned above has been performed.

h) An initial state, expressed as activities in progress and entities in queues. In selecting this it is often appropriate to choose a state in which as many activities are in progress as possible, since this is likely to be near to the equilibrium condition. (CAPS does contain a procedure which will do this for you which will be appropriate in many cases.)

The CAPS session will take the user through thirteen stages of discussion. These are listed below in groups representing major phases of the work.

Describing the Activity Cycle Diagram
1) Input and Editing of Cycles
2) Batch Activities
3) Analysis of the diagram- bound activities and parallel realisations

Priorities
4) Queue disciplines and matching rules
5) Activity priorities

Arithmetic
6) Formulae for activity durations
7) Calculations of attributes and other variables

Experimental Design
8) Set of variable class sizes and ranges

Recording
9) Recording of queues
10) Recording of attributes
11) Output Control

Initial State
12) Activities in progess, initial queue lengths

Rationalisation
13) Keyword and Function name check. Aggregation
14) Program Generation

At the end of each stage the user is given the opportunity to go back to any earlier section if it is thought necessary to correct information has been entered.

Since this step of the process is one in which the value of a good system is greatest, we illustrate it in detail. The following is an anotated complete run of CAPS for the trivial problem illustrated in the diagram above.

The input (the answers to questions from the computer) always follows a question mark as is printed in upper case. In reality, the output is neatly presented as a number of distinct screens, but for economy of space it is presented here as a continuous output with the screen headings etc omitted. Line lengths have also been adjusted to fit. Some discussion has been inserted between sections to help the reader.

```
Problem name?PUB
Are you going to use implicit queue mode?No
```

The principle type of entity is called CUSTOMER. The behaviour of these entities is now described.

```
Type name of one kind of entity?CUSTOMER
How many?10
Type a list of the states through which these
entities pass.  Precede queues by Q and
activities by A. Blank ends the list.
?QOUTSIDE,AARRIVE,QWAITING,APOUR,QREADY
?ADRINK,QWAITING,CNEED>0,QOUTSIDE
?
Is this cycle correct?Yes
```

The blank line of input indicates the end of the list and after extensive checking the computer indicates that it was acceptable by the question. Then we go on to the next type of entity.

```
At present the model has the following entity
types:-
CUSTOM

Your model includes the following activity
names:-
ARRIVE,POUR,DRINK

Type name of one kind of entity?GLASS
How many?12
Type list of states as above
?QCLEAN,APOUR,QFULL,ADRINK,QDIRTY,AWASH
?QCLEAN
?
Is this cycle correct?Yes
```

The reader will have noticed that the two cycles which have so far been entered are similar in type. They both consist of a series of states which must occur in a fixed sequence. The behaviour of the "barman" is different. This consists of two activities which can occur in any order dependent on the circumstances and a single "idle" state. Cycles of this type, often called facilities, are described to the computer as a list of activities without the intervening queue. Cycles of this type can have as many activities in the list as required.

```
At present the model has the following entity
types:-
CUSTOM,GLASS

Your model includes the following activity
names:-
ARRIVE,POUR,DRINK,WASH

Type name of one kind of entity?BARMAN
How many?1
Type list of states as above
?AWASH,APOUR,

Is this cycle correct?Yes

At present the model has the following entity
types:-
CUSTOM,GLASS,BARMAN

Your model includes the following activity
names:-
ARRIVE,POUR,DRINK,WASH

Type name of one kind of entity?
Do you wish to modify any of the data given
so far?N
```

The first question above was answered by a blank to indicate that there are no more cycles to be described. The second will occur at the end of each section of the discussion, thereby allowing the user the chance to correct any errors.

```
Are there any (other) activities which use
more than one entity of a particular type?Yes
Which activity?WASH
Which entity type?GLASS
How many entities of this type are used by
this activity?3
Which activity?

Do you wish to modify any of the data given
so far?No
```

On occasions some activities "process" a batch of entities at once. Here it is supposed that washing up involves three GLASSES at a time.

The computer is now able to analyse the total diagram and to check that it makes sense. As part of that checking the following analysis is displayed and questions are asked about it.

```
From what you have said so far, the following
are the maximum number of simultaneous
realisations of the activities.
Activity Number
ARRIVE  10
POUR     1     Limited by the number of BARMAN
DRINK   10     Limited by the number of CUSTOM
WASH     1     Limited by the number of BARMAN

Do you wish to apply any limits which are
lower than these?Yes
Which activity?ARRIVE
What is the limit?1
Which activity?
```

The above can often, as in this case, point out that some entity, whose availability might be a limitation on one or more of the activities, has been forgotten. In this case the entity DOOR was omitted. The added limitation on ARRIVE will achieve the required result.

```
Activity DRINK appears to be bound to POUR
I.E. The following queues are dummies
READY,FULL
Do you agree?Yes
Do you wish to see a summary of the cycles?Y
```

When an activity seems to be bound, this may represent reality or, again, it may indicate that something has been omitted or specified incorrectly. For example, in this problem it might be required that a customer obtain a chair before beginning drinking. If this were so, then the entity chair would have to be added and this addition would prevent the activities being bound.

Finally, the user can check for himself by requesting a summary of all that has now been agreed. This summary may be divided into sections if it will not fit onto the screen of the terminal being used. This principle applies throughout the system. Consideration of the summary shows it to be correct. Thus, the next section may be entered.

```
CUSTOM,10  ,QOUTSID,AARRIVE,QWAITIN,APOUR,Q
           ,ADRINK,QWAITIN,CNEED>0,QOUTSID
GLASS ,12  ,QCLEAN,APOUR,Q,ADRINK,QDIRTY
           ,AWASH*3,QCLEAN
BARMAN, 1  ,AWASH,APOUR
ZZARRI, 1  ,AARRIVE

ARRIVE uses 1 CUSTOM 1 ZZARRI
POUR   uses 1 CUSTOM 1 GLASS  1 BARMAN
DRINK  uses 1 CUSTOM 1 GLASS
WASH   uses 3 GLASS  1 BARMAN
```

Do you wish to modify any of the data given so far?No

Although the complete diagram has now been entered, definition of the model is nowhere near complete. The next two sections of the discussion are both concerned with priorities. Firstly, queue disciplines and then the relative priority of activities.
Are there any queues whose discipline is not F-I-F-O?No

Do you wish to preview the ECSL coding for the test head of any activity?
Do you wish to modify any of the data given so far?No

Most simulations require most of the queues to be processed first in first out (F-I-F-O). However, there may be some queues which require different treatment. In this case there are no such exceptions.

The only question of priority that remains to be settled, is the order of priority of the activities. In this case the BARMAN is able to WASH or POUR, and would normally give preference to an instance of POUR. WASHing will, therefore, only occur when for some reason POUR is not possible.

The following are bound activities:-
DRINK

The order of the following activities is unimportant:-
ARRIVE

I propose ordering the remaining activities as follows:-
WASH,POUR
Do you wish to raise the priority of any activity?Yes
Which activity?POUR
Which activity?

The revised order of the activities is
POUR,WASH
Do you wish to raise the priority of any activity?No
Do you wish to modify any of the data given so far?No

From now on the course of the conversation is more actively controlled by the computer. Essentially, the remainder of the discussion is concerned with filling in the details not yet given.

After each activity name, type formula for its duration, if any duration might be zero, type 0+...
```
POUR  ?NORMAL(6,1,S)
WASH  ?5
ARRIVE?0+NEGEXP(20,S)
DRINK ?5+RANDOM(5,S)
```
Do you wish to modify any of the data given so far?No

All of the formulae which are used in the system must be integer expressions which would be valid in E.C.S.L. (except that the subscripts of entities must be omitted). In the above examples, NORMAL, NEGEXP and RANDOM are pseudo-random functions which generate samples from the normal, negative exponential and uniform distributions. The variable S is a random number sequence. Many other theoretical distributions are available, together with those specified by the user as histograms. Here, as above, variables can be used without prior definition. The computer will fill in the details later. The "0+" is necessary to allow for the "coincidence" of two simultaneous arrivals.

In which activity is NEED of CUSTOM evaluated?ARRIVE,POUR
Please give the formula for its evaluation during ARRIVE
?1+RANDOM(3,S)
Please give the formula for its evaluation during POUR
?NEED-1
S is a random number "stream".
Please give an initial (Odd) random number?13
Does any attribute have further evaluation points?No
Do you wish to define any more attributes for entities?No
Do you wish to preview the ECSL coding for the evaluation section of any activity?No
Do you wish to modify any of the data given so far?No

Note: It is not necessary to say "need of customer", since the fact that need is an attribute of the customers is already known. The next section is ignored in this example.

Do you wish to set up an experimental design?No
Do you wish to modify any of the data given so far?No

The model itself is now complete. It is always necessary to record, in some way, what happens when the model is used. This is, therefore, the subject of the next section of the discussion.

Do you wish to see the instructions for this section?No

```
WAITIN?5      (This requests queue length and
OUTSID?0      and waiting time as histograms)
DIRTY ?0
CLEAN ?0            (Zero requests no output)
```
Do you wish to modify any of the data given so far?No

Do you wish to record any attributes?No
Do you wish to modify any of the data given so far?No

What length of Run-In period is required? 50
Do you wish to have intermediate results?No
Indicate required output mode-
```
 1   All output to terminal
 2   Intermediate output to terminal, final
     output to printer.
 3   All output to printer.
```
Which mode do you require?1
Please give the total duration of the simulation?1000
Do you wish to modify any of the data given so far?No

A run-in period of 50 units is allowed to avoid the results being too dependent on the starting conditions.

Finally, we come to establishing the initial conditions of the model. A system would rarely get into the state where nothing was happening anywhere in the system. Thus, it is often necessary to establish a starting point which includes the representation of some activities in progress.

```
Are there any activities in progress?Yes
Do you wish me to set up the activities in
progress?No
Which activity?ARRIVE
Termination Time?1
Which activity?

Type how many entities should be in each
queue listed after the queue name.
CUSTOM   10 Entities.
  1 used by activities in progress.
WAITIN?0
OUTSID?9
GLASS    12 Entities.
DIRTY ?8
CLEAN ?4
Do you wish to modify any of the data given
so far?No
```

The final questions and answers now complete the process. In some cases the user may have used a reserved word for a name of an element of the model (the computer will request replacement of these) or used a function that is not in the library (the computer will query these).

```
Some entities are apparently suitable for
aggregation. I.e. they have no attribute, no
delay recording and use only FIFO Q-
discipline.
Do you wish me to aggregate GLASS?Yes
Do you wish to use a cell structure?No

Your program has been written into A:PUB.ECS
```

The program is written by the computer and placed into file and the details of this discussion could be preserved for later amendment if any other versions of the model might be required. It would now be usual to compile and execute the program (indeed, many versions of the system do this automatically). However for the readers benefit the program is listed below in its entirety (except that some adjustments have been removed so that the longer lines will fit on the printed page).

```
CLASS 10 CUSTOM SET WAITIN OUTSID WITH NEED
THERE ARE 1 BARMAN
FUNCTI PICTUR NORMAL RANDOM NEGEXP
ARRIVE+1FOR 1
NEED OF CUSTOM1 = 1 + RANDOM( 3 , S )
CUSTOM1 INTO WAITIN AFTER 1
TIME OF ZZARRI = 1
RECYCL
RUNINZ= 50 AND PREVCLOCK = RUNINZ
SWITCH ADD ON AFTER RUNINZ

ACTIVITIES 1000

RECORD
DURATION= CLOCK - PREVCLOCK
FOR BARMAN WITH TIME OF BARMAN LT 0
  ADD DURATION TO EZBARMAN
ADD ARRIVE*DURATION TO FZZZARRI
PREVCLOCK = CLOCK

BEGIN POUR
CLEAN GE 1 AND TIME OF BARMAN LE 0
FIND FIRST CUSTOM A IN WAITIN
DURATION= NORMAL( 6 , 1 , S )
POUR +1 FOR DURATION
NEED OF CUSTOM A- 1
AADURATION= DURATION+ 5 + RANDOM( 5 , S )
DRINK +1 AFTER DURATION
DRINK -1 AFTER AADURATI
CHAIN CUSTOM A
  NEED GT 0
  CUSTOM A FROM WAITIN FOR AADURATION
  OR CUSTOM A FROM WAITIN
  CUSTOM A INTO OUTSID AFTER AADURATION
CLEAN - 1 AND DIRTY + 1 AFTER AADURATION
TIME OF BARMAN = DURATION
ADD 1 TO AZPOUR

BEGIN WASH
DIRTY GE 3 AND TIME OF BARMAN LE 0
WASH +1 FOR DURATION
DIRTY - 3 AND CLEAN + 3 AFTER 5
TIME OF BARMAN = 5
ADD 1 TO BZWASH

BEGIN ARRIVE
ARRIVE LT 1
FIND FIRST CUSTOM A IN OUTSID
NEED OF CUSTOM A= 1 + RANDOM( 3 , S )
DURATION= NEGEXP( 20, S )
CHAIN
  DURATION GT 0
  OR RECYCL
ARRIVE+1 FOR DURATION
CUSTOM A FROM OUTSID
CUSTOM A INTO WAITIN AFTER DURATION
ADD 1 TO CZARRIVE

FINALISATION
TYPE **'Final report from simulation PUB '/
TYPE 'POUR   was started' AZPOUR ' times'
TYPE 'WASH   was started' BZWASH ' times'
TYPE 'ARRIVE was started' CZARRIVE ' times'
REAL ZZ
ZZ=CLOCK-RUNINZ
TYPE 'Utilization of BARMAN',(1.-EZBARMAN/ZZ)
TYPE 'Utilization of ARRIVE',(FZZZARRI/ZZ)

DATA
DIRTY 8
CLEAN 4
OUTSID2 TO *
END
```

6. ENHANCING THE SIMULATION PROGRAM

While CAPS can generate quite a sophisticated program, it cannot incorporate every feature of which ECSL is capable. However the analyst can easily edit the generated program so as to enhance it.

7. RUNNING ECSL

The procedure for running ECSL is:-

In response to the system prompt, type ECSL. When ECSL has completed its initalisation, it will prompt the user with ECSL>. Each time it occurs the user responds with a command. The usual order for first time users is:-

```
ECSL>COMPILE  modelname              [This
                        compiles the program.]

ECSL>EXECUTE  0               [This runs the
                        program. The zero is a
                        debugging switch which
                        cuases the program to
                        pause before clearing
                        the screen, so that
                        the user has time to
                        read the output. Other
                        switches are used
                        below.]

ECSL>STOP                [This terminates the
                        ECSL session.]
```

The following is the output from the program above.

```
Final output from simulation PUB

POUR   was started        13 times
WASH   was started         5 times
ARRIVE was started         7 times
Utilisation of BARMAN   .9100
Utilisation of ARRIVE  1.0000

Histogram of length of queue WAITIN
Cell  Frequency *=2
   0     36******************
   1     40********************
   2     20**********
   3      3**
   4      1*

Histogram of delays at WAITIN
Cell  Frequency
   1      6******
   3      3***
   5      2**
   7      4****
   9      3***
  11      0
  13      1*
```

8. VERIFYING AND VALIDATING THE MODEL

Some simulations have a "driving" activity. In the case of PUB, it is ARRIVE. Provided the distribution used for the duration of this activity does not have a large variance, then the number of times it occurs in a simulation run should be very close to the period recorded divided by the mean duration of this activity. The number of occurrences of each of the other activities should be capable of approximate estimation from the number of occurances of the "driving" activity. (In other cases the CAPSCOG program can be used to estimate these figures.) In the case of the PUB, the average number of DRINKs per customer is 4, so DRINK should occur four times as often as ARRIVE. Each WASHing up activity cleans 5 glasses, so the number of WASHings should be one fifth of the number of drinks. If the results are far from these figures then the model was not in equilibrium for a significant portion of the period of recording.

The utilisation (which measures the proportion of time that an entity was busy) of the entity which controls the "driving" activity, here the DOOR, should always have the value 1.00. If this is not true, then the number of CUSTOMER entities was insufficient. In this case the results MUST BE DISREGARDED, the size of the class increased and the model re-run.

The summation of the frequencies in histograms of lengths of queues should be the same as the recording period. The summation of the frequencies in the histograms of delay time in queues should be the same as the number of times the subsequent activity(ies) started. If the cell size of any delay histogram has become very large or if the lowest cell printed does not include zero, the user should consider adjusting the origin and/or number of cells for the histogram. Note that in this case the MEAN estimated might be quite inaccurate because of the rounding effect cause by the wide intervals. The accuracy can often be estimated by observing that the inner product (i.e the sum of the products of corresponding elements) of the queue length and their frequencies and of the delay times and their frequencies should be equal if no inaccuracy has occurred.

The first concern of the user should be to get the model to run correctly. When the program has been written by CAPS, the principle reason for difficulty in achieving this is the initial conditions. The most frequent and obvious symptom of wrong (that is unrealistic, perhaps even "infeasible") initial conditions is that the activities have occurred a very much smaller number of times than expected from the static analysis. Three cases can be considered. First a model in which perhaps a few activties occur at the beginning (may be before the recording starts) and then no more action occurs. (For reference purposes we will call this "dead".) Secondly, a model in which the activities occur at the rate expected for quite some time and then, quite suddenly, things grind to a halt. (This we will call "stuck".) Thirdly, the case where the model runs consistently throughout the time period, but at a much lower rate of activity than expected. (This we will call "slow".) Even when the model has apparently run correctly, the user should check, using for example, a postmortem that there was no entity which has been stuck in one place for a very long time. (Look for a large negative TIME element.)

In all of these cases, the postmortem is the most valuable piece of evidence with which to seek out the "bug". The user should convert the state in which the model has terminated back to a state diagram using the activity cycle diagram. Place "counters" on the diagram to represent the state of each entity (as if a manual simulation was to be done). Now go through each activity in turn examining why it cannot happen. This consideration will usually show up the reason for the problem. The following is (a non-exhaustive) list of possible situations.

a) The entities needed by an activity are in the right queues but the value of an attribute prevents the activity occuring. This is a particularly frequent occurance if "complex" queue disciplines are used.

Correction: ensure that all attributes have values which correspond to their initial state. This will most frequently result in a "dead" model. This can be true when IDENtity or WIFE relationships are used and the initial conditions do not obey the "rules". (This can also result in an early failure of the program with the message "Invalid Index")

b) Where one queue is "parallel" to a sequence of states of some other entity. For example, in the diagram below, the number of entities in queue A should be equal to the total number of entities in states B, C and D. If this is not true then the model will act as if the "excess" entities (the amount by which B+C+D exceeds A), did not exist. Thus the model will appear "slow". This condition can be difficult to spot when the excess is a small proportion of the total number of that type of entity.

c) There are two activities, X and Y, which both need an A entity and a B entity. All the A's are available to X but none of the B's and all the B's are available to Y but none of the A's. This will result in a "stuck" model. (The circumstance is often called a "deadly embrace". This can be due to a bad starting condition, but is also a feature of many real systems if sufficient "control" is not exercised. This can particularly be caused when one of the entities concerned has a "branch from a queue" and the problem has occured because wrong decisions have been taken. Note that the problem cannot always (and even when it can it probably should not) be solved by increasing the number of entities or changing the priority at the branch. It may be nevcessary to explicitly limit the number of entities in each half of the cycle to being at least one less than the total number.

d) A particularly severe queue discipline can sometimes be the cause of problem. Insistance that the first entity in a queue be used, but applying a condition on its attribute may cause very long (even infinite) delays and result in a "slow" model (or even a "stuck" one.) The SUIT discipline should be used with great caution.

If the above considerations fail to produce a correct program, then the user will have to resort to techniques for "watching" the model in operation. Two types of "debugging" facilities are available in ECSL. Firstly, the DISPLAY facilities can be used to produce a visual representation of the model. This can be used to watch the model to observe where, if at all, it goes wrong. (An activity cycle diagram type visualisation, while not the only or necessarily the best, is a useful one and it can easily be produced using the program DRAW (an optional extra with CAPS)

The alternative method is to produce a "trace" and "cycle dump" using execution switches 2 and 4. These can be produced by using the modified execute command

The DISPLAY facility can also be used to get a valid model in another way. It can be used interactively, like a simulator, so that someone who really knows the real system well can make some of the schedulling decisions while the simulation is runing. Analysis of these decision can be used to improve the performance of the model, particularly in its reaction to "extreme conditions".
It is important to note that these uses of the DISPLAY, which are by far the most important, are only possible because the DISPLAY is a live (as distinct from a post-processor) activity.

```
ECSL>EXECUTE 24
```

9. ANALYSIS OF A PILOT RUN

When the model apparently runs correctly, the next step is to determine the length of run-in and of run that are required. Typically, a simulation, once started, might oscillate quite markedly until it eventually settles down. It has then reached its "steady state" of "statistical equilibrium". Recording which occurs before this "point" will be biased. Thus there is a need to know the duration of this transient behaviour as it can be equated with the run-in period for which no recording takes place.

Unless some attempt is made to analyse the early behaviour of a simulation model (transient behaviour) there is no way of knowing whether the run-in period is too short, which may cause biased results, or too long, which wastes computer time and has unnecessarily reduced the simulation time for which results are recorded.

Two ways of determining the run-in period can be identitifed. Each requires that the program be run initially without a run-in period and then the output can be examined so as to determine a reasonable period to cover run-in. This is most easily done by recording two or three queues which do contain a significant average number of entities. Once the run-in period is determined, the portfolio of output data can be extended but the analyst should always consider carefully the reason(s) for recording any particular queue length and/or delay time.

a) Using the Analyse and Graph Commands

The selected queues are recorded using CAPS method three (queue length recorded as a time series) and then subjecting the results to the ECSL "Analyse" command. It is best if the selected queues are of different entities and widely scattered through the system. The names of the time series can easily be deduced as this is done. For example, suppose the queues are called WAIT, DELAY and VACANT. The ECSL commands may well be similar to the following.

```
ECSL>Load  Model          (This re-loads the
                          compiled version of the
                          program - there is no
                          need to recompile it if
                          it has not been changed)
```

```
ECSL>Execute            (To execute the model as
                        usual)

ECSL>Analyse            (Giving no parameters as
                        yet).
```

The model contains the following vectors:- ZCWAIT, ZEDELA, SIZE, SHAPE, ZHVACA
GIVE THE NAME OF THE VECTOR TO BE ANALYSED?
This list contains the three vectors required (their names have Z-prefixes attached by CAPS) viz ZCWAIT ZEDELAy and ZHVACAnt. One of these should be given in answer to the question and the other two should be analysed by further use of the analyse command when the first has been completed.

The analysis includes, among considerable other data, "the first point which is not an extreme of all subsequent points", "the first passage thtrough the mean of all subsequent points" and a "standardised time series" analysis. These give an indication of when the simulation begins to settle.

The Graph command could also be used to give a visual indication of the run-in time. The same procedure is used as above but Graph is substituted for Analyse.

b) Generating Intermediate Output

The output control section of CAPS allows the user to specify that intermediate output should be produced. The primary purpose of this output is to allow the user to determine if the model has reach equilibrium. If this is the case, the utilisations and histogram means produced should not show a trend. The number of times each activity has started should of course increase linearly with time.

If this is not true, then a graph drawn from the data should allow the run-in period to be determined by finding how much of the run should be ignored to make the above statements true.

Note however, that different points in the model may take a different time to settle down. Also note that run-in times can be very variable - different random number sequences can produce such different sequencies that they can give very different run-in times. Subject to these precautions, the run-in time now determined should be built into the model. It is a good idea to retain some data collection from which the stability of the model can be continually checked as the experiment procedes.

It is particularly important to realise that if the model has a cyclic behaviour, which is likely if there is not driving activity, then the time for which the recording takes place must be a whole number of cycles. The ANALYSE command also examine the time series for evidence of cyclic behaviour and provides data from which the "best" method of recording can be deduced.

10. SETTING UP AN EXPERIMENTAL DESIGN

Simulation can only properly be utilised for assisting in the decision making process if it is carried out correctly. The averaged queue lengths or delay times for a single simulation are of no use whatsoever for decision making purposes. What is required is a properly designed statistical experiment. This will contain two elements:- a selected set of values for the variable parameters of the system and replication. However, replication can be considered to be just an extension of the statistical experiment, by considering the random number "seeds" as parameter.

The experimental design phase of CAPS will have allowed the user to set up variability of the number of entities in the various classes of entities. These will be among the parameters. The size of each such class is controlled by a single value integer variable. The name of this variable was given by the user to CAPS. The default value is specified in the DATA segment.

Similarly, the variables holding random number seeds have names which, in ECSL, are the names of single valued integer variables. Again the default initial values are specified in the DATA segment.

If other parameters are required the user must also arrange that they are established by setting the value of single valued integer variables. The default value must be specified in the DATA segment. Don't forget that the effect of varying the initial conditions should be examined.

Once these variables have been included in the program, and the program has been compiled, the user can then select sets of values for the parameters interactively, one by one, having seen the results of earlier runs. Thus the user will be able to conduct a properly designed "hill-climbing" statistical experiment.

This is done by using the "run-time" parameter facility. For example we will suppose that the BAR example has 2 parameters BMEN, the number of bartenders, and NGLASS, the number of glasses. Additionally there are two random number streams SA and SB. Then individual runs can be requested by commands like:-

```
ECSL>EXECUTE BMEN=3,NGLASS=10,SA=135,SB=157
```

Most textbooks on simulation will recommend the use of variance reduction techniques. Perhaps the simplest and most powerful of these is the use of antithetic random number streams. This consists of always running pairs of runs with the following property. In one run all the random number parameters have positive values. In the other, all the values of the initial random number parameters are given the same value but made negative. For example, the antithetic of the above is

```
ECSL>EXECUTE BMEN=3,NGLASS=10,SA=-135,SB=-157
```

The other main variance reduction technique is to make sure that the same random number parameters are used for each of the different sets of the none random number parameters.

11. RUNNING THE EXPERIMENT

If the experimental design used is of the fixed "classical" factorial type, the the ECSL command DESIGN should be used to create a batch file of execute command which will then be run automatically.

12. ANALYSING THE EXPERIMENT

The ECSL system contains a provision for recording the parameters and results obtained from the various executions in a file. This is simply done by use of the KEEP command, which is run before the series of executions is initiated. The parameters of the command are the names of the variables to be recorded. (The DESIGN command will also set this up, if desired)

At the end of the experimental run the RESULTS command will tabulate the "kept" data and the GRAPH command can be used to present the results in a graphical form. The file is in a form which can be directedly used by the MINITAB statistical package - thus any form of analysis, appropriate to the experimental design can be applied.

13. SELLING THE RESULTS

Remember that all results obtained from the experiment should be subject to proper statistical analysis. All differences observed should be tested for significance. Confidence limits should be obtained for all results.

Once properly validated results have been obtained from the model, it is often quite difficult to "sell" the conclusions to "management". In some cases the visual interactive display, discussed earlier, can be used to assist in this process by demonstrating the effects of possible decisions. This process should however be used with particular caution since the understanding of any stochastic model is a skilled (statistical) process - many results obtained are "counter-intuitive". Wrong impressions can be conveyed by viewing single samples and by "what if's" based on single short runs of a model. However, when used with the necessary professionalism, the visual display is a very powerful tool for selling the result of the study.

REFERENCES

Clementson, A.T., "Extended Control and Simulation Language", Computer Journal Vol 9, No 3, 1965.

Clementson, A.T., Computer Aided Programming of Simulations", User's Manual, Univeristy of Birmingham, 1972, (later versions published 1974, 1977, 1980)

Clementson, A.T., "Computer Aided Programming of Simulations", Proceedings 2nd National Conference of the Australian OR Soc, 1975.

Clementson, A.T., Extended Control and Simulation Language, Users' Manual, published by CLE.COM Ltd., 1982 and 1984. (Earlier versions 1972,1974,1977 and 1980 published by Univeristy of Birmingham)

Clementson, A.T., "Towards Modular Simulation", National Conference of the British OR Society, Durham, Sept 1985.

Clementson, A.T., and Hutchinson G.K., "Simulation of Flexible Manufacturing Systems", National Conference of the British OR Society, Durham, Sept 1985.

Hutchinson G.K. "An Introduction to Activity Cycles", Simuletter, October 1975

Hutchinson G.K. and Hughes J.J., "A Generalised Model of Flexible Manufacturing Systems", Proceedings- Workshop on Multi-Station, Digitally Controlled Manufacturing Systems, University of Wisconsin-Milwaukee,1977

Hutchinson G.K., "The Control of Flexible Manufacturing Systems: Required Information and Algorithm Structures", Proceedings of IFAC Symposium on Information Control Problems in Manufacturing Technology, Tokyo, Japan Oct1977

Hutchinson G.K. and Clementson A.T., "Manufacturing Control Systems: An Approach to Reducing Software Costs", International Conference on Manufacturing Science and Technology of the Future, MIT, Cambridge,MA, Oct 1984.

Peterson J.L. "Petrie Nets", Computing Surveys, Vol 9, No 3, Sept 1977.

Tocher K.D.,Aix-en-Provence, June 1956

AUTHOR'S BIOGRAPHY

ALAN CLEMENTSON is Head of the School of post-graduate studies in the Department of Engineering Production at the University of Birmingham England. He recieved a B.A. in Mathematics at Cambridge, a Diploma in Mathematical Statistics and a Doctor of Philosophy from Birmingham University. He was involved in the invention of C.S.L. at Esso Petroleum (UK) in the early sixties and has since the developed the E.C.S.L. system. In this work he was the first to propose and develop the current version of the activity structure and later the program generator.

Alan T. Clementson,
Department of Engineering Production,
University of Birmingham,
PO Box 363
Birmingham
England B15 2TT, UK.
(41)-21-472-1301