EDSIM—Event based discrete simulation using general purpose languages such as FORTRAN

A. Parkin and R. B. Coats

School of Mathematics, Computing and Statistics, Leicester Polytechnic, The Newarke, Leicester LE1 9BH

A versatile method of simulation, which dispenses with the use of special simulation languages, is described. Subroutines are supplied to allow commonly needed simulation requirements to be handled in a host general purpose language. The method is modelled on Extended CSL and is estimated to be about as powerful; yet it is substantially faster in execution, uses less memory, requires no special compiler, interpreter or pre-processor and does not require the simulator to know any programming language other than the general purpose language.

(Received November 1975)

1. Background

Many simulation languages have been developed (Stock and Stock, 1973 list over 50) but none has proved to be universally popular. According to Reitman (1971), GPSS and SIMSCRIPT are the most widely used simulation languages. In the United Kingdom, a popular language is CSL (Buxton and Laski, 1962) and its enhanced version Extended CSL (Clementson, 1966). The simulation method described here, EDSIM, arose out of a desire to find an event based discrete simulation language which (*a*) is easily learnt and used by a person with some programming experience in FORTRAN, COBOL or PL/1, (*b*) could easily be made available to the person in (*a*), whatever make of computer he used, and (*c*) offered facilities approximately equivalent to those in Extended CSL (ECSL).

Since no simulation language met these criteria and since it was felt undesirable to invent any further simulation languages, the solution chosen uses a general purpose language with supplied subroutines emulating ECSL facilities. In this way, all three criteria are automatically met; the user does not have to learn a new language, since the statements he makes are in the general purpose language with which he is most at home; the general purpose languages are widely available; the extra facilities can be made available by adding subroutines to the object statement subroutine library in the usual way.

For brevity the description that follows is confined to the FORTRAN version and it is assumed the reader has an appreciation of ECSL and FORTRAN.

2. The method

The data which the subroutines manipulate are provided by the programmer in accordance with the conventions for EDSIM, mainly in the form of arrays. Although he does not need to know how EDSIM works internally, the programmer needs to be aware of the structure of the arrays. As in ECSL it is the programmer's responsibility to declare the entities of the simulation, the states to which these entities can belong, the time cells and any histograms required; he does this by means of the following declarations:

2.1 *Classes*

The term CLASS is used in EDSIM to refer to a group of entities, with the class name being the plural of the entity name. For example a class SHIPS comprising 48 of the entities called SHIP would be declared by:

```
INTEGER SHIPS, SHIP
CALL DCLASS (SHIPS, SHIP, P, 48, 4HSHIP)
```

Class names and entity names are represented in EDSIM as FORTRAN INTEGER variables. The call to the EDSIM subroutine DCLASS (Declare CLASS) defines SHIPS as being a class of 48 entities called SHIP (i.e. SHIP 1, SHIP 2, . . . SHIP 48).

2.2 *Sets*

The term SET refers to an array which is used to keep track of the entities in a particular state. For example the entities, SHIP 1, SHIP 3 and SHIP 27 may be in the state 'at sea', in which case the array ATSEA would contain the values 1, 3 and 27. The declaration would be

```
INTEGER ATSEA (50)
CALL DSET (ATSEA, 50, 5HATSEA)
```

Sets are represented as FORTRAN INTEGER arrays, the size of the array being the maximum number of entities the set may hold, plus 2. In this case 48 ships may be in the state 'at sea'. The call to the EDSIM subroutine DSET (Declare SET) defines ATSEA as being a set capable of holding 48 entities. It is not a requirement that the set size is the same as the class size (unless required by the logic of the simulation)

2.3 *Time cells*

A time cell array (ITIMES) is used in an EDSIM program to hold all the time cells needed for the simulation. The first cell of this array is reserved for the EDSIM clock ICLOCK; then follow N time cells as required in the simulation; finally there is the cell ITIMCT which holds the number of cells in the ITIMES array, i.e. $N + 2$.

An example declaration of time cells is shown in the EDSIM program contained in Appendix 3. The six machine time cells and the two server time cells, together with ICLOCK and ITIMCT are equivalenced to the array ITIMES. The COMMON statement is just a dummy to ensure contiguity of storage.

2.4 *Histograms*

In an EDSIM program histograms are stored as INTEGER arrays. For example a histogram with 50 frequency cells would be declared as

```
INTEGER WAITS (55)
CALL DHIST (WAITS, 50, 0, 60)
```

The size of the array is the number of frequency cells in the histogram, plus 5. The call to the EDSIM subroutine DHIST (Declare HISTogram) defines WAITS as being a histogram with 50 frequency cells, the variate value associated with the first cell being 0, and the variate values associated with subsequent frequency cells being 60 more than the previous variate value.

Information about the data structures of classes, sets and histograms is given in Appendix 4.

3. An EDSIM program

An EDSIM program is made up of a declarations section, a transitions section and a termination section, corresponding respectively to the Initialisation, Activities and Finalisation of an ECSL program. The class, set, time cell and histogram definitions described above are normally contained in the Declarations section.

EDSIM provides ECSL functions by means of FORTRAN function subprograms; ECSL Set tests by FORTRAN logical function subprograms; and other ECSL facilities by called subroutines. In the case of ECSL Set manipulation verbs with implicit set tests (e.g. FROM) a logical variable DONE, contained in a labelled COMMON block, is set FALSE by the subroutine if the implicit test fails. Appendix 1 is a list of EDSIM subroutines—the ECSL equivalent is obvious in most cases. Appendix 2 contains notes on the programming of ECSL facilities not specifically catered for in EDSIM. Appendix 3 shows the same simple simulation program written in ECSL and FORTRAN.

4. Modes of running EDSIM programs

EDSIM programs can be run in one of two modes—CHECK mode or NOCHECK mode. The former is used during the development stage of a simulation program, when the programmer may need to know intermediate contents of specified sets, time cell values and so on, when checking or debugging his program. The EDSIM subroutines to facilitate this checking are listed in Appendix 1 under the heading 'checking'.

During the development stage there is a greater chance of one part of a program corrupting another part to produce erroneous results, for example by an array overflowing (on a computer which does not perform array bound checking). Corruption of classes or sets could lead to formidable problems for the programmer to solve, when the routines which manipulate these data areas are not his own, but are provided from the EDSIM library.

Consequently, in CHECK mode, classes, sets and histograms are checked for errors of this sort. Examples of internal checking performed by EDSIM are:

1. The specified entity is valid (e.g. SHIP 51 would be invalid)
2. The class and the set are compatible (e.g. an attempt to put PERSON 21 into ATSEA would be invalid)
3. The current members of a set are valid (e.g. set ATSEA may have been corrupted, and now contains SHIP 483)

When the programmer has validated his model, and is satisfied the program is performing correctly, he will use the model for experimenting. This stage often consumes the most computer time. During this stage the program will normally be run in NOCHECK mode, which bypasses the checking facilities and results in substantially shorter execution times (a factor of 2 or 3, depending on the actual simulation).

5. Comparison between EDSIM and ECSL

The following simulation programs were written in both ECSL and FORTRAN, and run on a Burroughs B6700 computer.

Program I: a simulation of a beercan factory (Clementson, 1976)

Program II: a simulation of a port (Clementson, 1976)

Program III: a simulation of a hospital operating theatre suite (Kwak *et al*, 1975).

5.1 *CPU times and core usage*

Fig. 1 shows how the two methods compared with regard to central processor times and average core usage.

The simulation program written in ECSL requires about 3 times as much core, and takes 8-25 times as long to run as the same program written in EDSIM. The original program of

Program	*Core usage (kwords)*		*Cpu time (seconds)*	
	EDSIM	*ECSL*	*EDSIM*	*ECSL*
I	3·28	9·76	3·88	82·6
II	4·10	12·13	12·96	107·1
III	3·93	11·51	4·14	104·3

Fig. 1 Comparison of EDSIM (NOCHECK mode) and ECSL for three programs

Kwak, *et al* simulated a hospital operating theatre suite over a period of 37 days; in order to compare ECSL and EDSIM the period was reduced to 3 days. Simulating the original period of 37 days takes 28 seconds of cpu time, for EDSIM, and would take in excess of 15 minutes of cpu time for ECSL.

The execution time of simulation programs written in ECSL can be reduced by about 30% by utilising the AFTER feature (Clementson, 1976). Programs written in EDSIM using the AFTER feature take comparable times to execute as their equivalents written in EDSIM without the AFTER feature. (A description of the EDSIM implementation of the AFTER feature is not included in this paper.)

5.2 *Size of source program*

More statements are required for a simulation program written using EDSIM than for its ECSL equivalent; in particular more explanatory comment is required if a clarity of purpose equivalent to that of ECSL is to be preserved. Typically EDSIM programs require about 60-80% more statements, depending on the particular program. Most of the extra statements occur in the declarations section; there is a near one-to-one correspondence between EDSIM and ECSL in the transition/activities sections. Using the proposed 1976 FORTRAN standard reduces the excess statements through the use of IMPLICIT typing of integers, and by taking advantage of the non-execution of a loop if the control variable exceeds the maximum value initially.

5.3 *Diagnostics*

In principle EDSIM should not itself give run-time diagnostic messages as good as ECSL's, because variable names are not retained by the system; how serious a disadvantage this is will depend largely on the diagnostic facilities provided by the FORTRAN system being used. EDSIM does provide run-time diagnostic messages which identify offending classes and sets by name, by utilising the literal arguments passed to the subroutines DCLASS, DSET and DTEMP. In addition the CHECK and TRACE facilities described in Appendix 1 can be used to advantage in debugging programs. In practice little difficulty in diagnosing problems has been encountered on the Burroughs B6700 system.

6. Conclusions

EDSIM provides most of the facilities of ECSL. It would appear that ECSL facilities not provided by EDSIM can be reasonably handled in the ordinary statements of FORTRAN.

EDSIM is faster in execution and requires less core than ECSL (or more precisely ECSL as implemented on a Burroughs B6700).

EDSIM promises to be valuable educationally; students learning event based discrete simulation methods can learn and use the concepts without having to master the syntax of a new language at the same time.

EDSIM supports any device or processing mode (e.g. remote terminal, interactive) that is supported by the host language.

More statements need to be written using EDSIM than with ECSL. Amending the number of entities in a class normally also entails amendment to the number of cells in sets to which the entity may belong, and corresponding amendments to the number of time cells for the entity, and the declarator for the ITIMES array. Although this difficulty could be overcome in host languages other than standard FORTRAN, it is felt that portability of the method is more desirable.

EDSIM bears some resemblance to the FORTRAN based simulation language GASP II (Pritsker and Kiviat, 1969). The major differences are EDSIM's ECSL orientation; EDSIM data is generally declared while GASP data generally comes from eight different types of card (prepared from a worksheet); the EDSIM programmer writes his program as a single integral unit (GASP requires a main program and a subroutine).

Appendix 1 EDSIM subroutines and functions

Key:

E =	Entity Name, e.g. PERSON, SHIP. This is the singular name specified in the DCLASS call.
I =	Index, an integer variable or constant identifying a particular entity in a class
S =	Set name, e.g. ATSEA
V =	Integer variable or constant, e.g. ICLOCK
D =	Distribution (Histogram) name, e.g. WAITS
literal =	A Hollerith constant of the form nH followed by *n* characters, the last of which is a full stop, e.g. 5HFREQ. The maximum value of *n* is 30
DONE =	The logical variable contained in the first location of the EDSIM Common Block EDSIMZ. It is set by the EDSIM subroutines HEAD, INTO and FROM

The EDSIM subroutines and functions are grouped according to their purpose, and are defined in the column labelled TYPE as being one of the following.

SUB:	SUBROUTINE
LF:	Logical Function
IF:	Integer Function

Group	*Name*	*Type*	*Purpose*
Checking	CHKON	SUB	Enables CHECK mode
	CHKOFF	SUB	Enables NOCHECK mode. No EDSIM checking of any kind is performed
	CHECKD (literal, D)	SUB	The literal and the cells of the named Histogram are printed if in CHECK mode
	CHECKS (literal, S)	SUB	The literal and the members of the named set are printed if in CHECK mode
	CHECKT (literal, ITIMES, M)	SUB	The literal and the contents of the first M cells of the ITIMES array are printed if in CHECK mode
	CHECKV (literal, V)	SUB	The literal and the value of the integer variable are printed if in CHECK mode
	PRINT	SUB	Allow the printing of output from the checking subroutines CHECKD, CHECKS, CHECKT and CHECKV
	NPRINT	SUB	Suppress the printing of output from the checking subroutines CHECKD, CHECKS, CHECKT and CHECKV
	TRACE	SUB	Print the names of EDSIM routines in the sequence they were executed. The names and values of entities, indexes and sets passed as arguments to the routines, the truth or falsity of DONE, and the values returned by functions are also provided
	NTRACE	SUB	Suppress the printing of trace information
Distributions	CLEAR (D)	SUB	The cells of D are set to zero
	TALLY (D, V)	SUB	The tally in the cell whose variate value is closest to V is incremented by 1
	SAMPLE (D, V)	IF	The function takes the variate value of a random sample drawn from D. V is used as a seed for the random number generator
	FREQU (D, V)	IF	The function takes the value of the contents of the cell whose variate is closest to V
	PRHIST (D, literal-1, literal-2)	SUB	A visual representation of D is produced on the lineprinter. The literals are used to label the axes. Literal-1 refers to what is normally the frequency axis, whereas literal-2 describes the variate axis
	HSTATS (D, NTOTAL, XMEAN, SDEV)	SUB	Returns the number (NTOTAL), the mean (XMEAN) and the standard deviation (SDEV) of the entries in D. XMEAN and SDEV are REAL variables. This information is printed automatically by PRHIST
Class	LOAD (E, J, S)	SUB	The set is initialised to contain the entities from 1 to J
	HEAD (E, I, S)	SUB	The set is searched for Entity I. If it is found, DONE is set to .FALSE. and no further action is taken. If it is not found, DONE is set to .TRUE. and Entity I is inserted before the first member of S
	INTO (E, I, S)	SUB	The set is searched for Entity I. If it is found, DONE is set to .FALSE. and no further action is taken. If it is not found, DONE is set to .TRUE. and Entity I is inserted after the last member of S

Group	Name	Type	Purpose
	FROM (E, I, S)	SUB	The set is searched for Entity I. If it is not found, DONE is set to .FALSE. and no further action is taken. If it is found, DONE is set to .TRUE. and Entity I is removed from the set
	IN (E, I, S)	LF	The function takes the value .TRUE. if Entity I is a member of S, otherwise it is set to .FALSE.
	NOTIN (E, I, S)	LF	The function takes the value .TRUE. if Entity I is not a member of S, otherwise it is set to .FALSE.
Set	ZERO (S)	SUB	All members of the set are removed
	FIRST (S)	IF	The function takes the value of the index of the first member of the set, or 0 if the set is empty
	LAST (S)	IF	The function takes the value of the index of the last member of the set, or 0 if the set is empty
	ANY (S, V)	IF	The function takes the value of the index of a member of the set chosen at random, or 0 if the set is empty. V is used as a seed for the random number generator
	COUNT (S)	IF	The function takes the value of the number of members in the set, or 0 if the set is empty
	EMPTY (S)	LF	The function takes the value .TRUE. if the set is empty, otherwise it is set to .FALSE.
Sets	GAINS (S_1, S_2)	SUB	Those members of S_2 which are not already members of S_1 are added to S_1 after the last member
	LOSES (S_1, S_2)	SUB	Those members of S_2 which are also members of S_1 are removed from S_1
	EQUALS (S_1, S_2)	LF	The function takes the value .TRUE. if both sets contain the same members (ordering is not significant), or if both sets are empty. Otherwise it takes the value .FALSE.
	WITHIN (S_1, S_2)	LF	The function takes the value .TRUE. if the members of S_1 are a subset of the members of S_2 (ordering is not significant), or if S_1 is empty. Otherwise it takes the value .FALSE.
	APART (S_1, S_2)	LF	The function takes the value .TRUE. if S_1 has no members in common with S_2 (ordering is not significant). Otherwise it takes the value .FALSE. The case of S_1 being empty returns the value .FALSE.
Statistical	RANDUM (V_1, V_2)	IF	The function takes the value of a random number in the range 1 to V_1. V_2 is used as a seed for the random number generator
	NEGEXP (V_1, V_2)	IF	The function takes the value of a random sample drawn from a negative exponential distribution of mean V_1. V_2 is used as a seed for the random number generator
	NORMAL (V_1, V_2, V_3)	IF	The function takes the value of a random sample drawn from a normal distribution of mean V_1 and standard deviation V_2. V_3 is used as a seed for the random number generator
TIME	ITIMOK (ITIMES, ITIMCT)	SUB	Checks the time cell declarations for consistency, and initialises all time cells to zero
	TIMADV (ITIMES, ITIMCT)	SUB	Advances the clock to the next event by finding the smallest positive value in the time cells ITIMES (2) to ITIMES (ITIMCT-1). This value is then subtracted from all the cells in this range, and added to ITIMES (1), i.e. ICLOCK
	TIMCLR (ITIMES, ITIMCT)	SUB	clears all time cells to zero.

Appendix 2 Notes on the programming of ECSL facilities not specifically catered for in EDSIM

The host language facilities are used which are nearest in concept to ECSL's FLOAT, GLOBAL, BOOLEAN, STRING, arithmetic statements, implicit functions, incremental statement, arithmetic tests, Boolean expressions, UNLESS, GO TO, SWITCH, FOR, REPEAT/DUMMY/CONTINUE, PRINT, READ, PROCEDURE.

The following ECSL facilities are not offered in EDSIM:

1. IS
2. Operations on multiple sets

```
        ECSL CAR 8 LOAD ROAD, GOING
       EDSIM CALL LOAD (CAR, 8, ROAD)
             CALL LOAD (CAR, 8, GOING)
```

3. Operations on multiple classes

```
  ECSL CHECK T. PERSON J, T. SERVER X
 EDSIM CALL CHECKV (7HPERSON., TPERSON(J))
       CALL CHECKV (7HSERVER., TSERVR(X))
```

4. Compound statements

```
        ECSL GOING GAINS ROAD LOSES CRASHED
```

```
EDSIM CALL GAINS (GOING, ROAD)
        CALL LOSES (GOING, CRASHD)
```

5. Test chains

```
ECSL CHAIN
        T.CAR 1 EQ 0
        CAR 1 IN ROAD
        CAR 1 FROM ROAD INTO CRASHED
EDSIM IF (TCAR(1).NE.0 .OR. NOTIN(CAR,1,ROAD))
      2  GOTO 20
        CALL FROM (CAR,1,ROAD)
        CALL INTO (CAR,1,CRASHD)
    20 ......
```

6. Loop control with a set name

```
            ECSL FOR I ROAD
         EDSIM JLAST = COUNT (ROAD)
                IF (JLAST.EQ.0) GOTO 20
                DO 10 J = 1, JLAST
                I = ROAD(J)
                  etc
              10 CONTINUE
              20 ......
```

7. Test chains with FIND and ANY

```
ECSL FIND M GOING ANY (1999)
        T.CAR M LT 10
        OR CAR M IN ROAD
        T.CAR M EQ 0
EDSIM CALL ZERO (TEMPST)
        DO 10 M = 1, CARS
        IF (NOTIN (CAR, M, GOING)) GOTO 10
        IF (TCAR(M).LT.10 .OR.
           (IN(CAR,M,ROAD) .AND. TCAR(M).EQ.0))
           . CALL INTO (CAR,M,TEMPST)
     10 CONTINUE
        IF (EMPTY(TEMPST)) GOTO 20
        M = ANY(TEMPST, 1999)
```

This type of construction is handled by first forming a temporary subset of members who satisfy the conditions, then applying the test. A similar approach can be used for ALL, EXISTS and UNIQUE test chains. All subroutines which manipulate sets recognise a special type of set known as a temporary set; it differs from a normal set in that it may contain entities from *any* class. A temporary set of size 100 is declared by a call to the EDSIM subroutine DTEMP (Declare TEMPorary set) e.g.

```
INTEGER TEMPST (102)
CALL DTEMP (TEMPST, 102, 6HTEMPST)
```

Appendix 3

```
                  E.C.S.L   PROGRAM   (HONEYWELL 200 VERSION)

        CLASS TIME MACHINE 6 SET DOWN INSERV
        CLASS TIME SERVER  2 SET IDLE
        HIST TIMDOWN(10,0,10)
        SERVER 2 LOAD IDLE
        STRA=1999
        STRB=1979
        FOR I MACHIN
            T.MACHINE I = NEGEXP(14,STRA) + 1

        ACTIVITIES 420

        BEGIN BREAKDOWN
        FOR I MACHINE
            T.MACHINE I EQ 0
            MACHINE I INTO DOWN
            OR CONTINUE

        BEGIN ENDSERV
        FOR Y SERVER
            T.SERVER Y EQ 0
            FIND K INSERV FIRST
            SERVER Y INTO IDLE
            MACHINE K FROM INSERV
            ADD -T.MACHINE K,TIMDOWN
            T.MACHINE K = NEGEXP(14,STRA) + 1
            OR CONTINUE

        BEGIN SERVICE
        FOR J MACHINE
            FIND X IDLE FIRST
            MACHINE J FROM DOWN INTO INSERV
            SERVER X FROM IDLE
            TOTIDL - T.SERVER X
            T.SERVER X = NEGEXP(14,STRB) + 1
            OR CONTINUE

        FINALISATION
        PRINT "END OF SIMULATION"/"TOTAL IDLE TIME" TOTIDL "MINS"
        PRINT "HISTOGRAM OF DOWNTIMES"/TIMDOWN

        END
```

```
C-
C-        EDSIM  PROGRAM
C-
C- *******              DECLARATIONS SCTION             **********
C-
      INTEGER FIRST
      LOGICAL DONE
      COMMON/EDSIMZ/DONE,IDUMMY(208)
C-
      INTEGER MACHNS,MACHIN
      CALL DCLASS(MACHNS,MACHIN,6,6HMACHIN)
      INTEGER DOWN(8),INSERV(8)
      CALL DSET(DOWN,8,4HDOWN)
      CALL DSET(INSERV,8,6HINSERV)
C-
      INTEGER SERVRS,SERVER
      CALL DCLASS(SERVRS,SERVER,2,6HSERVER)
      INTEGER IDLE(4)
      CALL DSET(IDLE,4,4HIDLE)
C-
      INTEGER TMACHN,TSRVR
      COMMON ICLOCK,TMACHN(6),TSRVR(2),ITIMCT
      ITIMCT=10
      DIMENSION ITIMES(10)
      EQUIVALENCE (ICLOCK,ITIMES(1))
      CALL ITIMOK(ITIMES,ITIMCT)
C-
      INTEGER TIMDWN(15)
      CALL DHIST(TIMDWN,10,0,10)
      CALL CLEAR(TIMDWN)
C-
      CALL LOAD(SERVER,2,IDLE)
      ISTRA=1999
      ISTRB=1979
      IDLTOT=0
      DO 10 I=1,MACHNS
   10 TMACHN(I)=NEGEXP(14,ISTRA)+1
C-
C-  ******              TRANSITIONS SECTION             **********
C-
   20 CALL TIMADV(ITIMES,ITIMCT)
C-
  100 CALL CHECKV(10HBREAKDOWN.,ICLOCK)
      DO 110 I=1,MACHNS
      IF(TMACHN(I).NE.0) GOTO 110
      CALL INTO(MACHIN,I,DOWN)
  110 CONTINUE
C-
  200 CALL CHECKV(8HENDSERV.,ICLOCK)
      DO 210 IY=1,SRVRS
      IF(TSRVR(IY).NE.0) GOTO 210
      K=FIRST(INSERV)
      IF(K.EQ.0) GOTO 210
      CALL INTO(SERVER,IY,IDLE)
      CALL FROM(MACHIN,K,INSERV)
      CALL TALLY(TIMDWN,-TMACHN(K))
      TMACHN(K)=NEGEXP(14,ISTRA)+1
  210 CONTINUE
C-
  300 CALL CHECKV(8HSERVICE.,ICLOCK)
      DO 310 J=1,MACHNS
      IX=FIRST(IDLE)
      IF(IX.EQ.0) GOTO 310
      CALL FROM(MACHIN,J,DOWN)
      IF(.NOT.DONE) GOTO 310
      CALL INTO(MACHIN,J,INSERV)
      CALL FROM(SERVER,IX,IDLE)
      IDLTOT=IDLTOT-TSRVR(IX)
      TSRVR(IX)=NEGEXP(14,ISTRA)+1
  310 CONTINUE
C-
  400 IF(ICLOCK.LT.420) GOTO 20
C-
C- *******              TERMINATION SECTION             **********
C-
      WRITE(6,500) IDLTOT
  500 FORMAT(18H END OF SIMULATION,/16H TOTAL IDLE TIME,I6,5H MINS)
      CALL PRHIST(TIMDWN,10HFREQUENCY.,23HHISTOGRAM OF DOWNTIMES.)
      END
```

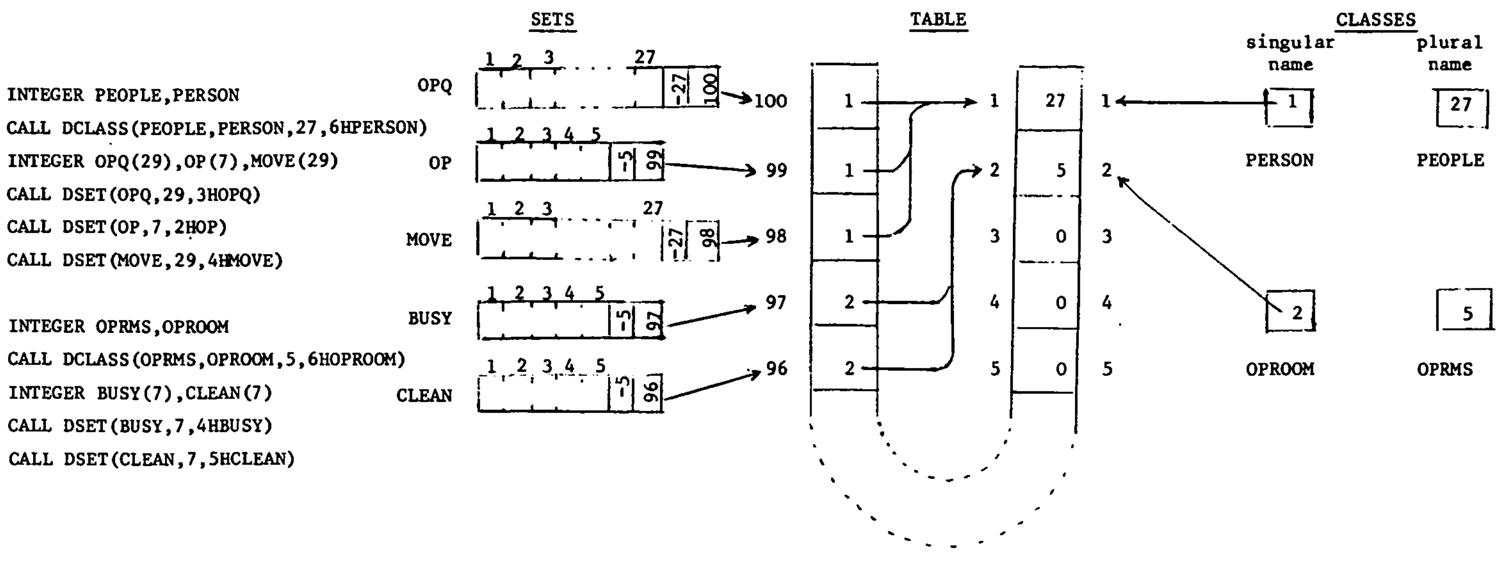
Appendix 4 Classes and sets

Every class is assigned a class identity number *N* by the subroutine DCLASS. This is an integer—the first class has an identity of 1, the second 2 and so on—which is held in the 'singular name' of the class. The 'plural name' holds the number of entities in the class.

Every set is assigned a set identity number by the subroutine DSET (or DTEMP). This is an integer—the first set has an identity of 100, the second 99, the third 98 and so on.

Every EDSIM set is terminated by two extra cells over and above those necessary for holding the entities which may be in the set. These are control cells used internally by EDSIM; the first contains the negated number of entities the set may hold, and the second holds the identity number of the set.

Checking the legality of requested set operations is achieved through the control cells and a table which is indexed by the



```
INTEGER PEOPLE,PERSON
CALL DCLASS(PEOPLE,PERSON,27,6HPERSON)
INTEGER OPQ(29),OP(7),MOVE(29)
CALL DSET(OPQ,29,3HOPQ)
CALL DSET(OP,7,2HOP)
CALL DSET(MOVE,29,4HMOVE)

INTEGER OPRMS,OPROOM
CALL DCLASS(OPRMS,OPROOM,5,6HOPROOM)
INTEGER BUSY(7),CLEAN(7)
CALL DSET(BUSY,7,4HBUSY)
CALL DSET(CLEAN,7,5HCLEAN)
```

Fig. 2 Data structures for an example EDSIM declaration

class identity number N passed to the called EDSIM subprogram, and the set identity numbers. This table is in EDSIM's labelled common area EDSIMZ. **Fig. 2** gives an example EDSIM declaration, together with the resulting values in TABLE and the set control cells established by EDSIM.

The following checks are carried out in CHECK mode:

Set checks

(*a*) no member's index is out of range

(*b*) no embedded empty cells among the members

(*c*) the *position* of the first control cell (containing the negated length of the set) corresponds to the *value* in the control cell.

(*d*) the ID of the set is within range of the table entries.

Class checks

(*a*) N is within range of the table entries

(*b*) the entity index I passed as an argument to the subprogram is in the range $1 \leqslant I \leqslant$ TABLE(N)

(*c*) N is equal to TABLE(ID of the set).

Set-set operations

The set checks described above are carried out for both sets. In addition a check is made to ensure that the sets share the same class ID.

Histograms

The first five cells of the array representing the histogram are used as control cells.

cell 1	: frequency cell count (K)
2	: variate value of first frequency cell
3	: variate value interval
4	: Σx
5	: Σx^2

cells 6 to K + 5 are the frequency cells of the histogram. Cells 4 and 5 are updated whenever the histogram is tallied, and they contain the sum and the sum of the squares respectively of the raw data. These cells are used by the EDSIM subroutine HSTATS to calculate the mean and the standard deviation of the histogram.

References

Buxton, J. N. and Laski, J. G. (1962). Control and Simulation Language, *The Computer Journal*, Vol. 5, No. 3, pp. 194-199.

Clementson, A. T. (1966). Extended Control and Simulation Language, *The Computer Journal*, Vol. 9, No. 3, pp. 215-220.

Clementson, A. T. (1976). Extended Control and Simulation Language, *Users manual*, University of Birmingham.

Kwak, N. K., Kuzdrall, P. J., and Schmitz, H. H. (1975). Simulating the use of space in a hospital surgical suite, *Simulation*, November 1975, pp. 147-152.

Pritsker, A., Alan, B. and Kiviat, P. J. (1969). *Simulation with GASP II*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey.

Reitman, J. (1971). *Computer Simulation Applications*, Wiley-Interscience.

Stock, M. and Stock, K. F. (1973). *Bibliographie der Programmiersprachen*, Verlag Dokumentation, Pullach/München.

Computer applications in chemical engineering

The 12th European symposium on computer applications in chemical engineering will be held at the Montreaux Palace Hotel, Switzerland from 8-11 April 1979. The general theme will be 'Computer applications in process development'. The symposium is sponsored by the European Federation of Chemical Engineering. Papers are invited in the areas of:

Integrating the development process through the use of computers
Advances in model building
The development, design and planning of batch processes
Education in the use of computers to aid engineering decisions
Process synthesis and new flowsheeting methods
Preliminary project evaluation—accounting for uncertainty, reliability, etc
Computer applications to safety and hazard analysis.

Titles and abstracts (250 words) of proposed papers are required by 30 June 1978. Full text of accepted papers (in English, French or German) must be submitted by 30 November 1978. Preprints will be provided for participants well in advance of the symposium.

Technical enquiries and suggestions may be made to

Dr Ing R. Luerau	Prof Dr D. W. T. Rippin
F. Hoffmann-la Roche & Co AG	Technisch-Chemisches Labor
Grenzacherstrasse 124	ETH-Zentrum
CH—4002 Basel	CH—8092 Zurich

All arrangements concerning accommodation, administration and proposals for papers should be sent to Conventus, 61 Avenue de Cour, CH—1007 Lausanne, Switzerland.