14. <u>System classes</u>

Two additional system-defined classes are available:

<u>class</u> SIMSET; .....;

and

SIMSET <u>class</u> SIMULATION; .....;

The class SIMSET introduces list processing facilities corresponding to the "set" concept of SIMULA I [2]. The class SIMULATION further defines facilities analogous to the "process" concept and sequencing facilities of SIMULA I.

The two classes are available for prefixing or block prefixing at any block level of a program. Such a prefix or block prefix will act as if an appropriate declaration of the system class were part of the block head of the smallest block enclosing the first textual occurrence of the class. An implementation may restrict the number of block levels at which such prefixes or block prefixes may occur in any one program.

In the following definitions, identifiers in capital letters, except "SIMSET" and "SIMULATION", represent quantities not accessible to the user. A series of dots is used to indicate that the actual coding is found in another section.

14.1 The class "SIMSET"

The class "SIMSET" contains facilities for the manipulation of circular two-way lists, called "sets".

14.1.1 General structure

14.1.1.1 Definition

```
class SIMSET;
   begin class linkage; .......;
      linkage class head ; ......;
      linkage class link; ......;
   end SIMSET;
```

14.1.1.2 Semantics

The reference variables and procedures necessary for set handling are introduced in standard classes declared within the class "SIMSET". Using these classes as prefixes, their relevant data and other properties are made parts of the objects themselves.

Both sets and objects which may acquire set membership have references to a successor and a predecessor. Consequently they are made subclasses of the "linkage" class.

The sets are represented by objects belonging to a subclass "head" of "linkage". Objects which may be set members belong to subclasses of "link" which is itself another subclass of "linkage".

14.1.2 The class "linkage"

14.1.2.1 Definition

```
class linkage;
   begin ref (linkage) SUC, PRED;

         ref (link) procedure suc;
             suc :- if SUC in link then SUC
                                   else none;

         ref (link) procedure pred;
             pred :- if PRED in link then PRED
                                     else none;
    end linkage;
```

14.1.2.2 Semantics

The class "linkage" is the common denominator for "set heads" and "set members".

"SUC" is a reference to the successor of this linkage object in the set, "PRED" is a reference to the predecessor.

The value of "SUC" and "PRED" may be obtained through the procedures "suc" and "pred". These procedures will give the value "none" if the designated object is not a "set" member, i.e. of class "link" or a subclass of "link".

The attributes "SUC" and "PRED" may only be modified through the use of procedures defined within "link" and "head". This protects the user against certain kinds of programming errors.

14.1.3 The class "link"

14.1.3.1 Definition

```
linkage class link;
   begin procedure out;
         if SUC =/= none then
            begin SUC.PRED :- PRED;
                  PRED.SUC :- SUC;
                  SUC :- PRED :- none
            end out;

         procedure follow(X); ref (linkage)X;
         begin out;
               if X =/= none then
                  begin if X.SUC =/= none then
                     begin PRED :- X;
                           SUC :- X.SUC;
                           SUC.PRED :- X.SUC :-
                               this linkage
                     end
                  end
         end follow;

         procedure precede(X); ref (linkage)X;
            begin out;
               if X =/= none then
                  begin if X.SUC =/= none then
                        begin SUC :- X;
                              PRED :- X.PRED;
                              PRED.SUC :- X.PRED :-
                                   this linkage
                        end
                  end
            end precede;
```

```
procedure into(S); ref (head)S;
            precede (S);
end link;
```

14.1.3.2 Semantics

Objects belonging to subclasses of the class "link" may acquire set membership. An object may only be a member of one set at a given instant.

In addition to the procedures "suc" and "pred", there are four procedures associated with each "link" object: "out", "follow", "precede" and "into".

The procedure "out" will remove the object from the set (if any) of which it is a member. The procedure call will have no effect if the object has no set membership.

The procedures "follow" and "precede" will remove the object from the set (if any) of which it is a member and insert it in a set at a given position. The set and the position are indicated by a parameter which is inner to "linkage". The procedure call will have the same effect as "out" (except for possible side effects from evaluation of the parameter) if the parameter is "none" or if it has no set membership and is not a set head. Otherwise the object will be inserted immediately after ("follow") or before ("precede") the "linkage" object designated by the parameter.

The procedure "into" will remove the object from the set (if any) of which it is a member and insert it as the last member of the set designated by the parameter. The procedure call will have the same effect as "out" if the parameter has the value "none" (except for possible side effects from evaluation of the actual parameter).

14.1.4 The class "head"

14.1.4.1 Definition

```
linkage class head;
   begin ref (link) procedure first; first :- suc;

         ref (link) procedure last; last :- pred;

         Boolean procedure empty;
                 empty := SUC == this linkage;

         integer procedure cardinal;
                 begin integer I; ref (link~~age~~)X;
                       X :- ~~this linkage;~~ FIRST;
                       ~~for X := X.suc~~ while X =/= none do
                       begin I := I+1; X:- X.SUC end;
                       cardinal := I
                 end cardinal;

         procedure clear;
                 begin ref (link)X; While first =/=none do first.out;
                       ~~for X :- first while X =/= none do X.out.~~
                 end clear;
         SUC :- PRED :- this linkage
   end head;
```

14.1.4.2 <u>Semantics</u>

An object of the class "head", or a subclass of "head" is used to represent a set. "head" objects may not acquire set membership. Thus, a unique "head" is defined for each set.

The procedure "first" may be used to obtain a reference to the first member of the set, while the procedure "last" may be used to obtain a reference to the last member.

The <u>Boolean</u> procedure "empty" will give the value <u>true</u> only if the set has no members.

The <u>integer</u> procedure "cardinal" may be used to count the number of members in a set.

The procedure "clear" may be used to remove all members from the set.

The references "SUC" and "PRED" will initially point to the "head" itself, which thereby represents an empty set.

14.2 <u>The class "SIMULATION"</u>

The system class "SIMULATION" may be considered an "application package" oriented towards simulation problems. It has the class "SIMSET" as prefix, and set-handling facilities are thus immediately available.

The definition of "SIMULATION" which follows is only one of many possible schemes of organization of the class. An implementation may choose any other scheme which is equivalent from the point of view of any user's program.

In the following sections the concepts defined in SIMULATION are explained with respect to a prefixed block, whose prefix part is an instance of the body of SIMULATION or of a subclass. The prefixed block will act as the main program of a quasi-parallel system which may represent a "discrete-event" simulation model.

14.2.1 General structure

14.2.1.1 Definition

```
SIMSET class SIMULATION;
begin link class EVENT NOTICE(EVTIME,PROC);
                 real EVTIME; ref(process)PROC;
      begin ref(EVENT NOTICE) procedure suc;
                suc :- if SUC is EVENT NOTICE then SUC
                                                 else none;

            ref(EVENT NOTICE) procedure pred;
                pred :- PRED;

            procedure RANK(BEFORE); Boolean BEFORE;
            begin ref (EVENT NOTICE)P;
                  P :- SQS.last;
                  ~~for P := P~~ while P.EVTIME > EVTIME do
                             P :- P.pred;
                  if BEFORE then ~~begin~~
                  ~~for P := P~~ while P.EVTIME = EVTIME do
                              P :- P.pred~~end;~~
                  follow(P)
            end RANK;
      end EVENT NOTICE;
      link class process;
      begin ref (EVENT NOTICE)EVENT; ...... end process;
      ref (head) SQS;
```

```
          ref (EVENT NOTICE) procedure FIRSTEV;
              FIRSTEV :- SQS.first;

          ref (process) procedure current;
              current :- FIRSTEV.PROC;

          real procedure time; time := FIRSTEV.EVTIME;

          procedure hold .....;
          procedure passivate .....;
          procedure wait .......;
          procedure cancel .......;
          procedure ACTIVATE .....;
          procedure accum .....;
          process class MAIN PROGRAM .....;
          ref (MAIN PROGRAM) main;

          SQS :- new head;
          main :- new MAIN PROGRAM;
          main.EVENT :- new EVENT NOTICE (0,main);
          main.EVENT.into(SQS)
     end SIMULATION;
```

14.2.1.2 Semantics

When used as a prefix to a block or a class, "SIMULATION" introduces simulation-oriented features through the class "process" and associated procedures.

The variable "SQS" refers to a "set" which is called the "sequencing set", and serves to represent the system time axis. The members of the sequencing set are event notices ranked according to increasing values of the attribute "EVTIME". An event notice refers through its attribute "PROC" to a "process" object, and represents an event which is the next active phase of that object, scheduled to take place at system time EVTIME. There may be at most one event notice referencing any given process object.

The event notice at the "lower" end of the sequencing set refers to the currently active process object. The object can be referenced through the procedure "current". The value of EVTIME for this event notice is identified as the current value of system time. It may be accessed through the procedure "time".

14.2.2 The class "process"

14.2.2.1 Definition

```
link class process;
begin ref (EVENT NOTICE)EVENT;
      Boolean TERMINATED;

      Boolean procedure idle; idle := EVENT == none;

      Boolean procedure terminated;
         terminated := TERMINATED;

      real procedure evtime;
         if idle then ERROR
                 else evtime := EVENT.EVTIME;

      ref (process) procedure nextev;
         nextev :- if idle then none else
                   if EVENT.suc == none then none
                   else EVENT.suc.PROC;
```

```
          detach;
          inner;
          TERMINATED := true;
          passivate;
          ERROR
     end process;
```

14.2.2.2 Semantics

An object of a class prefixed by "process" will be called a process object. A process object has the properties of "link" and, in addition, the capability to be represented in the sequencing set and to be manipulated by certain sequencing statements which may modify its "process state". The possible process states are: active, suspended, passive and terminated.

When a process object is generated it immediately becomes detached, its LSC positioned in front of the first statement of its user-defined operation rule. The process object remains detached throughout its dynamic scope.

The procedure "idle" has the value true if the process object is not currently represented in the sequencing set. It is said to be in the passive or terminated state depending on the value of the procedure "terminated". An idle process object is passive if its LSC is at a user defined prefix level. When the LSC passes through the final end of the user-defined part of the body, it proceeds to the final operations at the prefix level of the class "process", and the value of the procedure "terminated" becomes true. (Although the process state "terminated" is not strictly equivalent to the corresponding basic concept defined in section 9, an implementation may treat a terminated process object as terminated in the strict sense). A process object currently represented in the sequencing set is said to be "suspended", except if it is represented by the event notice at the lower end of the sequencing set. In the latter case it is active. A suspended process is scheduled to become active at the system time indicated by the attribute EVTIME of its event notice. This time value may be accessed through the procedure "evtime". The procedure "nextev" will reference the process object, if any, represented by the next event notice in the sequencing set.

14.2.3 Activation statements

14.2.3.1 Syntax

```
<activator> ::= activate|
                reactivate
<activation clause> ::= <activator><object expression>
<simple timing clause> ::=
                at <arithmetic expression>|
                delay <arithmetic expression>
<timing clause> ::= <simple timing clause>|
                    <simple timing clause> prior
<scheduling clause> ::= <empty>|
                        <timing clause>|
                        before <object expression>|
                        after <object expression>
<activation statement> ::= <activation clause>
                           <scheduling clause>
```

14.2.3.2 Semantics

An activation statement is only valid within an object of a class included in SIMULATION, or within a prefixed block whose prefix part is such an object.

The effect of an activation statement is defined as being that of a call on the sequencing procedure "ACTIVATE" local to SIMULATION.

```
procedure ACTIVATE(REAC,X,CODE,T,Y,PRIOR);
    value CODE; ref(process)X,Y; Boolean REAC,PRIOR;
    text CODE; real T;
```

The actual parameter list is determined from the form of the activation statement, by the following rules.

1. The actual parameter corresponding to "REAC" is true if the activator is reactivate, false otherwise.

2. The actual parameter corresponding to "X" is the object expression of the activation clause.

3. The actual parameter corresponding to "T" is the arithmetic expression of the simple timing clause if present, otherwise it is zero.

4. The actual parameter corresponding to "PRIOR" is true if prior is in the timing clause false if it is not used or there is no timing clause.

5. The actual parameter corresponding to "Y" is the object expression of the scheduling clause if present, otherwise it is none.

6. The actual parameter corresponding to "CODE" is defined from the scheduling clause as follows:

scheduling clause	actual parameter
empty	"direct"
at arithmetic expression	"at"
delay arithmetic expression	"delay"
before object expression	"before"
after object expression	"after"

14.2.4 Sequencing procedures

14.2.4.1 Definitions

```
procedure hold(T); real T;
   inspect FIRSTEV do
   begin if T > 0 then EVTIME := EVTIME + T;
         if suc =/= none then
         begin if suc. EVTIME ≤ EVTIME then
               begin out; RANK (false);
                     resume(current)
               end
         end
   end hold;

procedure passivate;
   begin inspect current do
         begin EVENT.out; EVENT :- none end;
         if SQS.empty then ERROR
                      else resume (current)
   end passivate;

procedure wait(S); ref (head)S;
   begin current.into(S); passivate end wait;

procedure cancel(X); ref (process)X;
   if X == current then passivate else
   inspect X do if EVENT =/= none then
      begin EVENT.out; EVENT :- none end cancel;
```

```
procedure ACTIVATE(REAC,X,CODE,T,Y,PRIOR); value CODE;
   ref (process)X,Y; Boolean REAC,PRIOR; text CODE;
                                         real T;
   inspect X do if ¬ TERMINATED then
   begin ref (process)Z; ref (EVENT NOTICE)EV;
         if REAC then EV :- EVENT
         else if EVENT =/= none then go to exit;
         Z :- current;
         if CODE = "direct" then
  direct: begin EVENT :- new EVENT NOTICE (time,X);
                EVENT.precede(FIRSTEV)
          end direct
          else if CODE = "delay" then
          begin T := T + time; go to at end delay
          else if CODE = "at" then
  at:     begin if T < time then T := time;
                if T = time ∧ PRIOR then go to direct;
          EVENT :- new EVENT NOTICE(T,X);
          EVENT.RANK(PRIOR)
          end at
  else if (if Y == none then true else Y.EVENT == none)
  then EVENT :- none else
  begin if X == Y then go to exit;
        comment reactivate X before/after X;
        EVENT :- new EVENT NOTICE (Y.EVENT.EVTIME,X);
        if CODE = "before" then EVENT.precede(Y.EVENT)
                           else EVENT.follow(Y.EVENT)
  end before or after;
  if EV =/= none then
  begin EV.out; if SQS.empty then ERROR end;
  if Z =/= current then resume (current);
exit: end ACTIVATE;
```

14.2.4.2 <u>Semantics</u>

The sequencing procedures serve to organize the quasi-parallel operation of process objects in a simulation model. Explicit use of the basic sequencing facilities (detach,resume) should be avoided within SIMULATION blocks.

The statement "hold(T)", where T is a real number greater than or equal to zero, will halt the active phase of the currently active process object, and schedule its next active phase at the system time "time + T". The statement thus represents an inactive period of duration T. During the inactive period the LSC stays within the "hold" statement. The process object becomes suspended.

The statement "passivate" will stop the active phase of the currently active process object and delete its event notice. The process object becomes passive. Its next active phase must be scheduled from outside the process object. The statement thus represents an inactive period of indefinite duration. The LSC of the process object remains within the "passivate" statement.

The procedure "wait" will include the currently active process object in a referenced set, and then call the procedure "passivate".

The statement "cancel(X)", where X is a reference to a process object, will delete the corresponding event notice, if any. If the process object is currently active or suspended, it becomes passive. Otherwise the statement has no effect. The statement "cancel(current)" is equivalent to "passivate".

The procedure "ACTIVATE" represents an activation statement, as described in section 14.2.3. The effects of a call on the procedure are described in terms of the corresponding activation statement. The purpose of an activation statement is to schedule an active phase of a process object.

Let X be the value of the object expression of the activation clause. If the activator is activate the statement will have no effect (beyond that of evaluating its constituent expressions) unless the X is a passive process object. If the activator is reactivate and X is a suspended or active process object, the corresponding event notice is deleted (after the subsequent scheduling operation) and, in the latter case, the current active phase is terminated. The statement otherwise operates as an activate statement.

The scheduling takes place by generating an event notice for X and inserting it in the sequencing set. The type of scheduling is determined by the scheduling clause.

An empty scheduling clause indicates direct activation, whereby an active phase of X is initiated immediately. The event notice is inserted in front of the one currently at the lower end of the sequencing set and X becomes active. The system time remains unchanged. The formerly active process object becomes suspended.

A timing clause may be used to specify the system time of the scheduled active phase. The clause "delay T", where T is an arithmetic expression, is equivalent to "at time + T". The event notice is inserted into the sequencing set using the specified system time as ranking criterion. It is normally inserted after any event notice with the same system time; the symbol "prior" may, however, be used to specify insertion in front of any event notice with the same system time.

Let Y be a reference to an active or suspended process object. Then the clause "before Y" or "after Y" may be used to insert the event notice in a position defined relation to (before or after) the event notice of Y. The generated event notice is given the same system time as that of Y. If Y is not an active or suspended process object, no scheduling will take place.

Example:

The statements

```
activate X
activate X before current
activate X delay 0 prior
activate X at time prior
```

are equivalent. They all specify direct activation.

The statement

```
reactivate current delay T
```

is equivalent to "hold(T)".

14.2.5 The main program

14.2.5.1 Definition

process class MAIN PROGRAM; WHILE TRUE do detach;
~~begin L: detach; go to L end MAIN PROGRAM;~~

14.2.5.2 Semantics

It is desirable that the main program of a simulation model, i.e. the SIMULATION block instance, should respond to the sequencing procedures of section 14.2.4 as if it were itself a process object. This is accomplished by having a process object of the class "MAIN PROGRAM" as a permanent component of the quasi-parallel system.

The process object will represent the main program with respect to the sequencing procedures. Whenever it becomes operative, the PSC (and OSC) will immediately enter the main program as a result of the "detach" statement (cf. section 9.2.1). The procedure "current" will reference this process object whenever the main program is active.

A simulation model is initialized by generating the MAIN PROGRAM object and scheduling an active phase for it at system time zero. Then the PSC proceeds to the first user-defined statement of the SIMULATION block.

14.2.6 Utility procedures

14.2.6.1 Definition

```
procedure accum (a,b,c,d); name a,b,c;
          real a,b,c,d;
begin a := a + c  × (time - b);
      b := time; c := c + d
end accum;
```

14.2.6.2 Semantics

A statement of the form "accum (A,B,C,D)" may be used to accumulate the "system time integral" of the variable C, interpreted as a step function of system time. The integral is accumulated in the variable A. The variable B contains the system time at which the variables were last updated. The value of D is the current increment of the step function.