SIMULATION class COUNTERACT;
  cgi widget index; read simulation;
class station;
  station class store;
  station class service;
  station class multi;
process class clerk;
  clerk class singleclerk;
process class customer;
process class arrival;
  end COUNTERACT;
SIMULATION class COUNTERACT;

begin
    integer index; real service period;

class station;

virtual:
    ref (client) procedure kind; procedure cloneParam;
    real procedure serviceTime;

begin
    ref (client) server Friday;
    procedure cloneParam(); end station;

station class store;

begin
    ref (head) q; q := new head & end;

station class single (molecules);

begin
    ref (head) q, office;
    ref (head) is irreplaceable;
    ref (head) procedure kind;

    kind := new common client (then q);
    q := new head; office := new head;

for index := 1 step 1 until molecules do

    begin
        server := kind;
        service := get service;
        activate server;
        server := end;

end.
station class multiq (channels); integer channels
begin
integer mngers;
ref (head) array q[ ];
ref (clerk) array c[ ];
integer procedure mng;
begin
ref (clerk) procedure serve;
for k := 1 step 1 until channels do
begin
q[k] := new (c[k], 0, 0);
extend
serve := -new (c[serve], 0, 0);
activate serve;
serve := next serve;
end
end

procedure class clerk (location, q); ref (station) location; ref (head) q;
begin
procedure wait; ref (customer) procedure choice;
read procedure servtime;
begin
ref (customer) is (selected); real a, b;
procedure wait; private;
ref (customer) procedure choice; choice := q, first;
read procedure servtime; servtime := location, servtime;
repeat ; if q, empty then proceed wait;
selected := choice, selected, next, cancel (selected);
location, servtime, serve := --next serve;
activate selected after current;
selected := next; go to repeat;
end
end
proc clerk clerk customer;
    3.1 procedure wait;
        begin into (location(1), office); pprocess; out end;

    end clerk;

proc clerk service;
    begin real service time, deptime, init, min, min;
        procedure into (location, station, location);
        begin set (head) 1, select; set (clerk) 1, select;
            suspect location when single;
            select location when single;
                begin q, select; - 1, c, select; - office, first end
                begin q, select; - 1, c, select; - office, new end
            when single do:
                begin q, select; - 1, c, select; - office, new end
        end proc.
    end proc.

proc machine (T); real T;
    begin reactivate current at T; if idle then out end; end customer;
COUNTERACT begin

station

station : check 1, check 2, bank, kiosk, control, drop lobby, exit lobby;

arrival class charter;

begin hold (meqep (3));
activate new charter pass end;

arrival class international;

begin hold (meqep (2));
activate new it pass end;

arrival class domestic;

begin hold (meqep (2));
activate new dom pass end;

multi class station ;

begin procedure check param ;
begin server a := INPUT ; server b := INPUT end;

real procedure service time ;

server time := normal (a, b);

real procedure service time ;

service time := normal (server a, server b);
end procedure multi class station ;
multiq class SpecCounter;
    begin
    end serviceTime;
        serviceTime := server.selected.serviceTime;
        end SpecCounter;

singleq class ChartCounter;
    begin
    real a, b, a := E1.093; b := E1.09;
        real procedure serviceTime;
        serviceTime := normal(a, b);
        end ChartCounter;

singleq class BookCounter;
    begin
        real procedure serviceTime;
        serviceTime := server.selected.serviceTime;
        end BookCounter;

customer class ChartPass;
    begin
        real depletion := time + normal(30, 5);
        if (depletion > 10) then
draw(0.2); then
        begin into (bank); maximize (depletion - 3) end;
        into (control); maximize;
        into (not lobby); maximize;
    end ;
customer class uicide;

begin
depthtime := time + normal (25, 5);

into (check in 2); passivate;

if (depthtime - time) < 5 then
    begin
        into (bank); maxtime(depthtime) end;
    into (control); passivate;
    into (ticket lobby); passivate
end;

end;

customer class compass;

begin
depthtime := time + normal (25, 10);

into (check in 2); passivate;

if (time < depthtime) then
    begin
        service time := normal (5, 1);
        into (bank); maxtime(depthtime) end;
    if (time < depthtime) then
        begin
            service time := normal (5, 5);
            into (ticket lobby); maxtime(depthtime) end;
    into (don't lobby) passivate
end;

end;

// Comment: MAIN PROGRAM

simperiod := 1 E 39900;
checkin 1 := new check counter (2); checkin 2 := new check counter (5);
bank := new spec counter (2); bank := new ticket counter (2);
control := new new counter (1); dom lobby := new store;
don't lobby := new store; activate new counter;
activate new international; activate new domestic;
hold (simperiod + 500); REPORT
end;
block instances and relate them to each other in various types of structures. Hence powerful but processing facilities are required.

Another useful property of a language would be to allow a hierarchical classification of concepts: to start with some general concepts (like the classes of machines and orders in a factory simulation) and gradually split these into sub-concepts (like "machine class lathe", "machine class drill", "order class batchsize", etc).

SIMULA-67 represents one solution to the requirements stated above.

The class declaration

The format of the SIMULA-67 class declaration

(prefix) class (identifier) (formal parameter list) ; (specifications)

(begin) (declarations) (statements) end

The various components will be discussed below, and to first a simplified version simpler special case will be treated.

class (identifier) (formal parameter list) ; (specifications)

begin (declarations) (statements) end
Associated with the class declaration is a "generating expression":

```
new <class identifier> (<actual parameter list>)
```

When the PSC encounters an expression of this kind, a new block instance conforming to the pattern of the given class declaration is created. This block instance is attached to the block containing the generating statement. Hence a "class block instance" is initially in an attached state.

The binding rules are in this simple case the same as for procedures, but with two deviations:

- the "name" parameter mechanism is not available,
- a "reference" parameter mechanism is introduced (and also made available in the necessary extension of the procedure concept).

Block instances generated from clan declarations are called "objects."