```
SIMULATION class COUNTERACT ;
   begin integer index ; real simperiod ;
    class station ;
   station class store ;
   station class singleq ;
   Station class multiq ;
 process class clerk ;
 clerk class commonqclerk
 process class customer
 process class arrival
   end COUNTERACT ;
```

station

ref(clerk)server → store

ref(head)q

U v: real proc servicetime

U v: ref(clerk)proc kind

U v: proc ~~cparam~~ clerkparam

lf proc clerk,param e

singleq(nclerks)

ref(head) q, office

~~procedure~~

~~ref(clerk)proc kind~~

ref(clerk) proc kind

<initialization>

~~proc clerkparam~~;

multiq(channels) integer channels

ref(head) array q[1: channels]

ref(clerk) array c[1: channels]

~~pr~~ integer minqnr;

integer procedure minq

~~ref~~ ref(clerk) proc kind

<initialization>

~~proc~~

clerk(location, q)

real a, b

v: procedure wait,

ref(customer) proc choice

real proc servicetime

ref(customer) proc choice

procedure wait

real proc servicetime

ref(customer) selected

customer

real servicetime, deptime;

procedure into(location) ...

procedure maxtime(T)

arrival

commonqclerk

proc wait

Hor brukes egentlig kind til?

```
SIMULATION class COUNTERACT;

 begin integer index; real simperiod;

      class station;
   virtual: ref(clerk) procedure kind, procedure clerkparam;
            real procedure servicetime;
        begin ref(clerk) server ~~[illegible]~~;
              procedure clerkparam;; end station;
  station class store;
        begin ref(head) q; q :- new head; end;


  station class singleq(nrclerks); integer nrclerks;
    begin ref(head) q, office;  ~~q :- new head~~;
      ref(~~head~~ clerk) procedure kind;
        kind :- new commingclerk(this ~~[illegible]~~ station, q);
      q :- new head; office :- new head;
    for index := 1 step 1 until nrclerks do
      begin server :- kind; ~~server.visitor~~
            ~~get clerkparam;~~ ~~c.param;~~ clerkparam
            activate server; server :- none end;
  end
```

```
Station class multiq (channels); integer channels
    begin integer minqno;
        ref(head) array q[1:channels];
        ref(clerk) array c[1:channels];
        integer procedure minq;
          begin . , - - - -  - - - - - - - end;
        ref(clerk) procedure kind;
          ~~begin~~ kind:-new clerk (this station, q[index]);
      for index := 1 step 1 until channels do
      begin q[index] :-new head; server:-kind;
          clerkparam; ~~act~~ c[index]:-server;
          activate server; server:-none end;
    end;

process class clerk (location, q); ref(station)location; ref(head)q;
    virtual: procedure wait; ref(customer) procedure choice;
              real procedure servicetime;
    begin ref(customer) ~~[illegible]~~ selected; real a, b;
        procedure wait; passivate;
        ref(customer)procedure choice; choice:-q.first;
        real procedure servicetime; ~~servicetime :=~~
              servicetime := location.servicetime;
repeat : if q.empty then ~~passivate~~ wait;
    selected:-choice; selected.out; cancel(selected);
    location.server :- this clerk;
    hold (servicetime); activate selected after current;
    selected :- none; goto repeat
  end clerk;
```

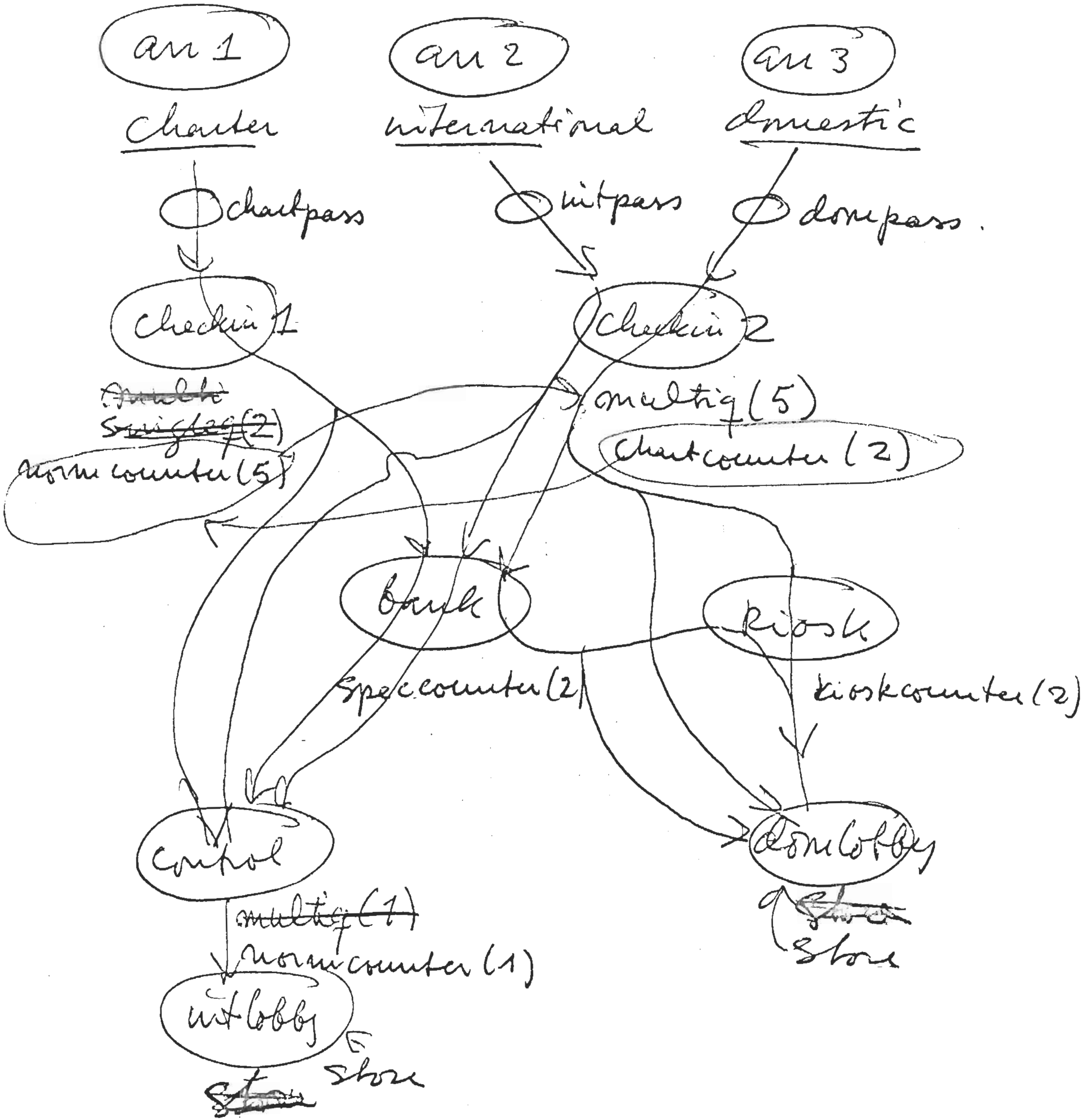
```
~~clerk~~ clerk class commonqclerk;
        begin procedure wait;
            begin into(location qua singleq . office);
                passivate; out end;
        end commonqclerk;
```

```
process class customer;
    begin real servicetime, deptime; integer minqnr
  procedure into(location); ref(station)location;
   begin ref(head)qselect; ref(clerk)cselect;
     ~~inspect location when singleq do~~
     inspect location
        when singleq do
                  begin qselect :- q; cselect :- office.first end
        when multiq do
                  begin ~~nrselect~~ minqnr := minq; qselect :- q[~~nrselect~~ minqnr];
                        cselect :- c[~~nrselect~~ minqnr] end ;
        when store do qselect :- q;
     this process. into(qselect); activate cselect after current
  end of into;

  procedure maxtime(T); real T;
   begin reactivate current at T; if ¬idle then out end;
    end customer;
                                        ?
```

```
process class arrival ;
    begin
    repeat: inner; if time < simperiod then go to repeat
    end arrival ;
end COUNTERACT;
```

AIRPORT DEPARTURE



B

```
COUNTERACT begin

  ~~ref (station)~~
  ~~ref [illegible]~~
  ref (station) checkin 1, checkin 2, bank, kiosk, control,
      domlobby, intlobby;
  arrival class charter;
          begin hold (negexp (3));
                activate new chartpass ; end

  arrival class international;
          begin hold (negexp (2));
                activate new intpass end;

  arrival class domestic;
          begin hold (negexp (2));
                activate new dompass end;

multiq class ~~[illegible]~~ normcounter;
  begin procedure clerkparam;
        begin server.a := INPUT; server.b := INPUT end;
        ~~real procedure servicetime;~~
           ~~server servicetime := negexp~~
           ~~servicetime := normal (a, b)~~;
        real procedure servicetime;
         servicetime := normal (server.a, server.b);
  end ~~[illegible]~~ normcounter;
```

C

```
multiq class speccounter;
        begin ~~procedure~~
        real procedure servicetime;
            servicetime := server.selected.servicetime;
        end speccounter;

singleq class chartcounter
        begin real a, b; a := INPUT; b := INPUT;
        ~~procedure clientparam;~~
         ~~begin server.a := a; server.b := b end;~~
        real procedure servicetime;
          servicetime := normal(a, b);
        end chartcounter;

singleq class kioskcounter;
        begin
        real procedure servicetime;
           servicetime := server.selected.servicetime;
        end kioskcounter

customer class chartpass;
        begin ~~into(checkin1); passivate;~~
          ~~act~~ deptime := time + normal(30,5);
          into(checkin1); passivate;
          if (deptime - time > 10) ∧ draw(0.1) then
Servicetime := negexp(2);
          begin into(bank); maxtime (deptime - 3) end;
          into(control); passivate;
          into(intlobby); passivate
      end;
```

D

```
customer class intpass;
      bgin deptime := time + normal(25,5);
        into(checkin2); passivate;
        if (deptime-time < 5) ∧ draw(0.2) then
servicetime:=negexp(1);
        begin into(bank); maxtime(deptime) end;
        into(control); passivate;
        into(intlobby); passivate
      end;

customer class dompass;
      bgin deptime := time + normal(25,10);
             into(checkin2); passivate;
           if(time < deptime) ∧ draw(0.1) then
           begin servicetime := negexp(~~3~~1);
             into(bank); maxtime(deptime) end;
           if(time<deptime) ∧ draw(0,3) then
           bgin servicetime := negexp(0.5);
                into(kiosk); maxtime(deptime) end;
           into(domlobby) passivate
      end;

~~chk~~ comment MAIN PROGRAM;
  simperiod := ~~tt~~ INPUT;
  checkin1 :- new chartcounter(2); checkin2 :- new normcounter(5);
  bank :- new speccounter(2); kiosk :- new kioskcounter(2);
  control :- new normcounter(1); domlobby :- new store;
  intlobby :- new store; activate new charter;
  activate new international; activate new domestic;
  hold(simperiod + 1000); REPORT
  end;
```

block instances and relating them to each other in various types of structures. Hence powerful list processing facilities are required.

Another useful property ~~is~~ 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~~ then split these into sub-concepts (like "machine class lathe", "machine class drill", "order class batchorder" etc)

~~In~~ SIMULA-67 represents one solution to the requirements stated above

<u>The class declaration – I</u>

The format of the SIMULA-67 class declaration is

```
<prefix> class <identifier>(<formal parameter list>); <specifications>
              <virtual part>
              begin <declarations> <statements> end;
```

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

```
        class <identifier>(<formal parameter list>); <specifications>
          begin <declarations> <statements> end;
```

"Associated with ~~the~~ a class declaration is a "generating expression":

"~~new<identifier~~

" 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 proper 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 extensions of the procedure concept).

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

References

In order to be able to refer to objects, ~~a~~ new types of variables are introduced. To each class declaration corresponds a "reference type" declared by

ref (<class identifier>)<identifier list>,