22. FORMAL DESCRIPTION OF SIMULA COMMON BASE RUNTIME SYSTEM

```
begin ref (driver) CD;
      ref (notice) nothead,POOL2TOP,POOL2BOTTOM;
      ref (object) POOL1FIRST,POOL1LAST;
      ref (driver) array DDISPLAY[1 : maxlevel];
      ref (object) array DISPLAY [1 : maxlevel];

      comment  the integer maxlevel is assumed to be
      defined when the RTS is written.  The variables
      and arrays above are "non-local" to the runtime
      routines.

      comment  "driver" and "eventnotice" are sub-
      classes of a class called "notice".

      class notice (obj); ref (object) obj;
            begin Boolean referenced; ref (notice) notc;
            end notice;

      notice class eventnotice (time); real time;
             begin ref (eventnotice) BL,RL,LL;
             end eventnotice;

      notice class driver (drp,pex,drex,acs,md,level);
             ref (program) pex; ref (driver) drex,drp;
             ref (object) acs;
             Boolean md; integer level;
             begin Boolean con,rp,pb,dot,ob;
                   ref (eventnotice) evp;
                   ref (driver) cdrp,drch;
             end driver;

      comment  all block instances are a subclass of
      "object".  This includes arrays and stored accumu-
      lator stacks which have special values of PP.
```

```
class object (PP); ref (prototype) PP;
      begin ref (driver) MDP; end object;

comment  a prototype is a description of a family of
block instances.  One prototype is generated by the
compiler for each procedure, class, subblock and pre-
fixed block in the program;

class prototype (lg,nvirt,nrp,plev,nrl,level);
      integer lg,nvirt,nrp,nrl,level,plev;
      begin ref (program) statements,inretur,endblk,declare;
            ref (prototype) array prefix[0:plev+1];
            integer array relad,kind,type[nvirt+1 :
                              nvirt+nrp+nrl];
            Boolean array valu[nvirt+1 : nvirt+nrp];
            ref array progaddr[1 : nvirt];
            Boolean pb,ob,local classes;
            integer vtype;
      end prototype;

comment  update display is used to update DISPLAY and
DDISPLAY to reflect the textual situation defined by CD;

procedure update display;
   begin integer i;
      DDISPLAY [CD.level] :- CD;
      for i := CD.level-1 step -1 until 1 do
          DDISPLAY [i] :- DDISPLAY [i+1].drp;
      for i := 1 step 1 until CD.level do
         DISPLAY [i] :- DDISPLAY [i].obj;
   end update display;

comment  deletenotice is used to put a driver or event-
notice in the list  of available notices;

procedure deletenotice (x); ref (notice) x;
   begin x.notc :- nothead; nothead :- x; end;
```

```
comment  SUBBLOCKS
         BB    - begin subblock
         EBL   - end subblock;

procedure BB(p); ref (prototype) p;
   begin
      CD :- new driver (new object (p),CD,none,CD,none,true,
      p.level);
      CD.obj.MDP :- CD;
      DISPLAY [p.level] :- CD.obj;
      DDISPLAY [p.level] :- CD;
      go to p.declare
   end BB;

procedure EBL;
   begin ref (driver) x;
      x :- CD.drp;
      deletenotice (CD);
      CD :- x;
    end EBL;
```

```
comment  PROCEDURES;

comment  procedure end;

universal procedure EPR (val);
   begin ref (driver) x; ref (program) out;
      x :- CD.drex;
      out :- CD.pex;
      restore (CD.acs);
      if CD.dot then deletenotice (CD.drp);
      deletenotice (CD);
      CD :- x;
      EPR := val;
      update display;
      go to out;
   end EPR;

comment  utility routine to enter declaration code
or in-line coding for parameter transmission;

procedure ENTPROC;
   begin ref (driver) y;
      CD.obj.MDP :- CD;
      if CD.obj.PP.nrp = 0 then
         begin update display;
            go to CD.obj.PP.declare end;
      y :- CD.drex;
      CD :- new driver (y.obj,y.drp,none,CD,none,
                false,y.level);
      CD.con := y.con;
      CD.cdrp :- y.cdrp;
      DDISPLAY[y.level] :- CD;
      go to CD.drex.pex;
   end ENTPROC;
```

comment utility routine to enter declaration code of procedure or class after parameter transmission.

Note: It is assumed that prefix [0] of the prototype for a procedure points to the procedure itself;

```
procedure ENTER;
   begin ref (driver) y;
      y :- CD;
      CD :- CD.drex;
      CD.pex :- exit;
      deletenotice (y);
      update display;
      go to CD.obj.PP.prefix [0].declare;
   end ENTER;
```

```
comment  NON-FORMAL, NON-VIRTUAL PROCEDURES;

comment  call on normal procedure (local or non-local);

procedure CPR (p,acs);
   ref (prototype) p; ref (object) acs;
   begin
      CD :- new driver (new object (p), DDISPLAY [p.level-1],
          exit,CD,acs,true,p.level);
      ENTPROC;
   end CPR;

comment  call on connected procedure (cfr. CONNECTION);

procedure CCP(p,c,acs);
   ref (prototype) p; ref(object) acs;
   ref (driver) c;
   begin ref (driver) x;
      x :- new driver (c.obj,c.cdrp,none,none,none,false,
          p.level-1);
      x.con := true;
      CD :- new driver (new object(p),x,exit,CD,acs,
          true,p.level);
      CD.dot := true;
      ENTPROC
   end CCP;
```

```
comment  call on remote procedure;

procedure CDP (p,c,slc,acs);
   ref (prototype) p; ref (object) c,acs;
   ref (driver) slc;
   begin ref (driver) x;
      x :- new driver (c,slc,none,none,none,false,p.level-1);
      x.con := true;
      CD :- new driver (new object(p),x,exit,CD,acs,true,
          p.level);
      CD.dot := true;
      ENTPROC;
   end CDP;
```

```
comment  VIRTUAL PROCEDURES;

comment  ENTVIRT utility procedure for virtuals corresponding
to ENTPROC.  The routine must
1.  check actual vs. formal number of parameters
2.  check type/kind of parameters
3.  perform calls by value;

procedure entvirt (p,dr,dot,acs);
   ref (prototype) p; ref (driver) dr;
   ref (object) acs; Boolean dot;
   begin
      CD  :- new driver (new object(p),dr,exit,CD,acs,true,
          p.level);
      CD.dot := dot;
      CD.obj.MDP :- CD;
      store descriptors for name parameters;
      store value parameters;
      update display;
      go to p.declare
   end entvirt;

comment  call on normal virtual procedure;

procedure CVP (cl,index,acs);
   integer index;
   ref (object) acs; ref (driver) cl;
   begin ref (prototype) p,q;
      p :- cl.obj.PP;
      q :- p.progaddr(index) qua prototype;
      if q == none then error ("cvp",1);
      entvirt (q,cl,false,acs)
   end CDVP;
```

```
comment  call on connected virtual procedure;

procedure CCVP (c,index,acs);
   integer index;
   ref (object) acs; ref (driver) c;
   begin ref (prototype) p,q;
      p :- c.obj.PP;
      q :- p.progaddr(index) qua prototype;
      if q == none then error ("CCVP",1);
      c :- new driver (c.obj,c.cdrp,none,none,none,
         false,p.level);
      c.con := true;
      entvirt (q,c,true,acs)
   end CCVP;

procedure CDVP(c,index,slc,acs);
    integer index;
    ref (object) c,acs; ref (driver) slc;
    begin ref (prototype) p;
       p :- c.PP.progaddr (index) qua prototype;
       if p == none then error ("CDVP",1);
       slc :- new driver (c,slc,none,none,none,
           false,p.level-1);
       slc.con := true;
       entvirt (p,slc,true,acs)
    end CDVP;
```

```
comment  CLASSES;

comment  GENERATING REFERENCE;

comment  begin class, enters declaration code if no
parameters, otherwise continue with in-line parameter
evaluation and return is later through ENTER.

procedure BC (x,slx,acs);
   ref (prototype) x; ref (object) slx;
   ref (object) acs;
   begin ref (driver) y; ref (prototype) q;
      y :- new driver (new object(x),slx.MDP,exit,CD,acs,
          true,x.level);
      y.ob := true; y.obj.MDP :- y;
      if x.nrp ≠ 0 then
         begin y :- new driver (CD.obj,CD.drp,none,y,none,
                  false,CD.level);
               y.con := CD.con;
               y.cdrp :- CD.cdrp;
               CD :- y;
               DDISPLAY[CD.level] :- CD;
               go to exit
         end else CD :- y;
      update display;
      go to CD.obj.PP.prefix[0].declare
   end BC;

comment  begin class return, signifies the end of declaration
code in a class.

procedure BCR(q); integer q;
   begin ref (prototype) x,y;
      x :- CD.obj.PP;
      y :- x.prefix[q+1];
      if  y =/= none then go to y.declare;
      go to x.prefix[0].statements
   end BCR;
```

```
comment  procedure corresponding to the statement inner;

procedure CINNER (lev); integer lev;
   begin ref (prototype) p;
      p :- CD.obj.PP.prefix[lev+1];
      if p =/= none then go to p.statements
   end CINNER;

comment  end class body;

ref (object) procedure ECB(p); ref (prototype) p;
   begin ref (program) out; ref (driver) x;
      procedure delete;
         begin x.rp := false;
            if x.obj.PP.local classes then
            begin
                  x.drex :- x.drp;
                  x.pex :- none;
                  x.acs :- none;
            end
            else begin x.obj.MDP :- none;
                       deletenotice(x)
                 end
         end delete;
             if p.plev ≠ 0 then go to p.prefix [p.plev-1]
                                                  .inretur;
             x :- CD;
             if CD.rp and not CD.pb then
             begin CD :- CD.drp; delete; go to L2;
             L1:   CD :- CD.drp;
             L2:   while not CD.rp do CD :- CD.drex;
                   if not CD.pb then go to L1;
                   x :- CD.drp;
                   while not x.rp do x :- x.drex;
                   x.drex :- CD;
             L3:   if CD.pex =/= none then go to L4;
                   CD :- CD.drex;
                   go to L3;
             L4:   out :- CD.pex;
             end else
```

```
    begin
        if x.pb then
        begin CD :- x.drp;
              out :- x.obj.PP.endblk;
              deletenotice (x);
              go to ud
        end else
        begin out :- x.pex;
              CD :- CD.drex;
              ECB :- x.obj;
              restore (x.acs);
              delete
        end
    end;
    update display;
    ud : go to out
end ECB;
```

```
comment  PREFIXED BLOCK;

comment  begin prefixed block, enter declarations or
evaluate parameters.  If latter case, return through EPBPAR;

procedure BPB (x); ref (prototype) x;
   begin ref (driver) a,y,z;
      z :- new driver (new object(x),CD,none,none,none,true,
          x.level)
      z.rp := z.ob := z.pb := true;
      z.obj.MDP :- z;
      a :- CD;
      while not a.rp do a :- a.drex;
      a.pex :- none;
      a.drex :- z;
      if x.nrp ≠ 0 then
         begin
            y :- new driver (CD.obj,CD.drp,none,z,none,false,
                CD.level);
            y.con := CD.con;
            y.cdrp :- CD.cdrp;
            CD :- y;
            go to exit;
         end else CD :- z;
      DISPLAY [x.level]:- CD.obj;
      DDISPLAY [x.level]:- CD;
      go to x.prefix[0].declare
    end BPB;

comment  end prefixed block parameters;

procedure  EPBPAR;
   begin ref (driver) y;
       y :- CD;
       CD :- CD.drex;
       deletenotice(y);
       DISPLAY [CD.level] :- CD.obj;
       DDISPLAY [CD.level] :- CD;
       go to CD.obj.PP.prefix[0].declare
   end EPBPAR;
```

```
comment  begin prefixed block return, end declaration
code in prefixed block;

procedure  BPBR;
   go to CD.obj.PP.prefix[0].statements;

comment  end prefixed block;

procedure EPB;
   go to CD.obj.PP.prefix[CD.obj.PP.plev-1].inretur;
```

```
comment  QUASI-PARALLEL SEQUENCING;

comment  resume;

procedure resume (x);
   ref (object) x;
   begin
      ref (driver) y,z;
      Boolean b;
      if x =/= none then
      begin
         z :- x.MDP;
         if z == none then error ("resume",1);
         if not z.rp then error ("resume",2);
         y :- CD;
         while not y.rp do y :- y.drex;
         y.drex :- CD;
         y.pex :- exit;
         CD :- z;
         while CD.pex == none do CD :- CD.drex;
         exit :- CD.pex;
         CD :- CD.drex;
         y :- z;
```

```
L:      b := y.pb;
        y :- y.drp;
        while not y.rp do y :- y.drex;
        if not b then go to L;
        y.drex :- z;
        update display;
        go to exit
    end
    else error ("resume",3);
end resume;
```

```
comment detach;

ref (object) procedure detach;
   begin ref (driver) x,y; ref (program) out;
      x :- CD;
      if x.rp then
      begin while not x.pb do
                begin x :- x.drp;
                      while not x.rp do x :- x.drex;
                end;
                CD.pex :- exit;
                CD.drex :- CD;
                y :- x;
                x :- x.drp;
                while not x.rp do x :- x.drex;
                x.drex :- y;
                x.pex :- none;
                while y.pex == none do y :- y.drex;
                out :- y.pex;
                CD :- y.drex;
      end else
      begin
          out :- CD.pex;
          y :- CD.drex;
          CD.pex :- exit;
          CD.drex :- CD;
          CD.rp := true;
          detach :- CD.obj;
          restore (CD.acs);
          CD.acs :- none;
          CD :- y;
       end;
       update display;
       go to out
    end detach;
```

```
comment  call (not part of Common Base);

procedure call (x);
   ref (object) x;
      begin ref (driver) a,y,z; ref (program) next;
         if x =/= none then
            begin z :- x.MDP;
               if z == none then error ("call",1);
               if not z.rp then error ("call",2);
               y :- z;
               while y.pex == none do y :- y.drex;
               next :- y.pex;
               z.pex :- exit;
               a :- y.drex;
               z.drex :- CD;
               z.rp := false;
               CD :- a;
               update display;
               go to next
            end else error ("call",3)
      end call;
```

```
comment  PARAMETER EVALUATION (THUNK);

comment  call (enter) thunk;

procedure CTH(tha,thu,acs);
   ref (program) tha;
   ref (object) acs; ref (driver) thu;
   begin
      CD :- new driver (thu.obj,thu.drp,exit,CD,acs,false,
          thu.level);
      CD.con := thu.con;
      CD.cdrp :- thu.cdrp;
      update display;
      go to tha
   end CTH;

comment  end thunk;

ref procedure ETH(addr); ref addr;
   ETH :- epr(addr);
```

```
comment  CONNECTION;

procedure CONNECT (p,bl);
   ref (object) p; integer bl;
   begin
      CD :- new driver (p,CD,none,CD,none,false,CD.level+1);
      CD.con := true;
      CD.cdrp :- DISPLAY[bl].MDP;
      DDISPLAY [CD.level] :- CD;
      DISPLAY [CD.level] :- p
   end CONNECT;

comment   RELATIONS;

comment  check in;

Boolean procedure CIN(x,c);
   ref (object) x; ref (prototype) c;
   if x =/= none then
      begin if x.PP.plev ≥ c.plev then
               CIN := x.PP.prefix[c.plev] == c
      end;
```

```
comment  INSTANTANEOUS QUALIFICATION (QUA);

ref procedure CIQ (x,c);
   ref (object) x; ref (prototype) c;
      begin ref (prototype) d;
         d :- x.PP;
         if d.plev < c.plev then error ("CIQ",1);
         if d.prefix [c.plev] =/= c then error ("CIQ",2);
         CIQ  :- x
      end CIQ;
```

```
comment  GO TO STATEMENTS;

comment  utility procedure conddel;

procedure conddel (x); ref (driver) x;
   begin
      if x.md then
         begin if not x.obj.PP.local classes then
            begin if x.dot then deletenotice (x.drp);
            deletenotice (x); x.obj.MDP :- none; end
            else begin x.drex :- x.drp; x.pex :- none;
            x.acs :- none end
         end
         else if x.dot then
         begin deletenotice (x.drp);
               deletenotice (x)
         end else deletenotice (x);
   end conddel;

comment  go to normal label;

procedure GL (b,m); ref (object) b; ref (program) m;
   begin ref (driver) d; Boolean legal;
         while CD.obj =/= b or not CD.md do
         begin if CD.rp then
               begin d :- CD.drp;
                     if d == none then error ("GL",1);
                     legal := CD.pb;
               end else d :- CD.drex;
               conddel (CD);
               CD :- d;
         end;
         if not legal then error ("GL",2);
         go to m;
   end GL;
```

```
comment  go to virtual label;

procedure GVL (bl,index); integer bl,index;
   begin ref (program) k;
      k :- DISPLAY [bl].PP.progaddr (index) qua program;
      if k = none then error ("GVL",1);
      GL (DISPLAY [bl],k);
   end GVL;
```

```
procedure storecollapse(req);integer req;
   begin
      integer kin,typ,pnr;
      Boolean pa,va;
      ref (object) objch,x,y,na,mta,mfa,pobj;
      ref (driver) drchn,drv;
      ref (program) pexit;
      ref (notice) d,lm;
      ref z;

procedure chain (x); ref (object) x;
  if x =/= none then
      begin
         if (x.MDP==none or x.MDP is driver) then
            begin x.MDP :- objch; objch :- x end
      end chain;
```

```
procedure chain2(y); ref (driver) y;
   begin y.referenced := true;
   y.drch :- drchn; drchn :- y end chain2;

procedure map (x); ref (driver) x;
   if x =/= none then
      begin
      L0:  if not x.referenced then
           begin
              if x.drex == none then  go to L6;
              while not x.rp do x :- x.drex;
              while x.pex == none do x :- x.drex;
      L3:  x :- x.drex;
      L4:  x.referenced := true;
           if x.con then
           begin
           if x.cdrp =/= none then begin chain2(x);
           go to L5; end
           end else if x.dot then chain2 (x.drp);
           chain (x.acs);
           chain (x.obj);
      L5:  if not x.rp then go to L3;
           if x.pb then begin x :- x.drp;
           go to L4 end;
           x :- x.drp; go to L0;
      L6:  chain (x.obj);
           x :- x.drp;
           go to L0
      end
  end map;

procedure upd1(n); ref (notice) n;
   begin procedure upd(z); name z; ref (object) z;
      if z =/= none then z :- z.mdp;
      upd (n.obj);
      inspect n when driver do upd (acs);
   end upd1;
```

```
procedure upd2(n); name n; ref (notice) n;
   if n =/= none then
      begin
         if n < POOL2TOP then n :- n.notc
      end upd2;

procedure maptree(e); ref (eventnotice) e;
   begin if e == none then go to ret;
   L:    map(e.obj.MDP);
         e.referenced := true;
         if e.RL =/= none then
            begin e :- e.RL; go to L end;
         if e.LL =/= none then
            begin e :- e.LL; go to L end;

   M:    e :- e.BL;
         if e =/= none then
            begin if e.LL.referenced
                  then go to M
                  else begin e :- e.LL; go to L end;
            end;
   ret: end maptree;

ref (ref) procedure asgn(z,i); ref (prototype) z; integer i;
   inspect z when prototype do
      begin kin := kind[i]; typ := type[i];
      på := par[i]; va := valu[i];
               asgn :- pobj+relad[i]; i := i+1 end asgn;

ref (ref) procedure firstpointer(x,L); value L; ref (object) x;
label L;
   begin ref (prototype) z;
      z :- x.PP;
      if z.nrpointers = 0 then go to L;
      pnr := 1; pobj :- x; pexit :- L;
      firstpointer :- asgn (z,pnr)
   end firstpointer;
```

```
      ref (ref) procedure nextpointer;
         begin ref (prototype) z;
            z :- pobj.PP;
            if pnr > z.nrpointers then go to pexit;
            nextpointer :- asgn (z,pnr)
         end nextpointer;

      procedure move (x,y,i); ref x,y; integer i;;

      comment  pass 1;

      x :- CD;
      while not x.rp do x :- x.drex;
      x.pex :- exit;
      x.drex :- CD;
      POOL1LAST.val :- none;
      map (CD);

L:
      if objch =/= none then
         begin x :- objch; objch :- x.MDP; x.MDP :- x;
               z :- firstpointer (x,L).val;
         r:    inspect z when object do
                  begin if MDP == none then chain (z)
                     else if MDP is driver then map(MDP)
                  end
               when driver do map (this driver)
               when eventnotice do maptree (this eventnotice);
               z :- nextpointer.val;
               go to r
         end;
rep2:
      if drchn =/= none then
         begin drv :- drchn; drchn :- drv.drch;
            inspect drv when driver do
            inspect obj when object do
            begin if MDP == none then chain (obj)
            else if MDP is driver then map (MDP);
         end;
```

```
            if cdrp =/= none then map (cdrp); drv.referenced := true
         end;
         go to rep2
      end else if objch =/= none then go to L;

      comment  pass 2;

      y :- none;
      na :- x :- POOL1FIRST;
nextblock:
      if x.MDP =/= x then
          begin if y == none then
              begin y :- x; x.PP :- none end
          end
      else begin x.MDP :- na;
                 na :- na+x.PP.lg;
                 if y =/= none then begin y.MDP :- x; y :- none end
          end;
      x :- x+x.PP.lg;
      if x.PP =/= none then go to nextblock;
      if y =/= none then y.MDP :- none else x.MDP :- none;

      comment pass  3;

      d :- POOL2BOTTOM;
used:
      if d.referenced then
         begin updl(d);
            d.referenced := false;
            if d == POOL2TOP then go to pass3end;
            d :- d-drlg;
            go to used
         end;
```

```
unused:
     if not POOL2TOP.referenced then
        begin
             POOL2TOP :- POOL2TOP+drlg;
             if POOL2TOP == d then begin POOL2TOP :- POOL2TOP-drlg;
             go to pass3end end else go to
             unused
         end;

     move (POOL2TOP,d,drlg);
     POOL2TOP.notc :- d;
     d.referenced := false;
     updl(d);
     lm :- d;
     POOL2TOP :- POOL2TOP+drlg;
     d :- d-drlg;
     if d > POOL2TOP then go to used else
        if not (d == POOL2TOP and d.referenced) then POOL2TOP :-
        lm else
             begin updl(d); d.referenced := false end;
pass3end:

     comment  pass 4;

     x :- POOL1FIRST;
upnext:
     y :- firstpointer (x,m);
update:
     z :- y.val;
     inspect z when notice do
        if z  < POOL2TOP then y.val :- notc
     when object do y.val :- MDP;
     y :- nextpointer;
     go to update;

m:
     x :- x+x.PP.lg;
     if x.PP  =/= none then go to upnext;
     x :- x.MDP;
     if x =/= none then go to upnext;
```

```
        comment  pass 5;

        mta :- x :- POOL1FIRST;
        mfa :- x;
        go to entpass5;
   clear:
        x.MDP :- none;
        x :- x+x.PP.lg;
entpass5: if x.PP =/= none then go to clear;
        if mfa =/= mta then move (mfa,mta,x-mfa);
        mta :- mta+x-mfa;
        x :- x.MDP;
        mfa :- x;
        if x =/= none then go to clear:
        POOL1LAST :- mta;

        comment pass 6;

        d :- POOL2BOTTOM
    nextnotice:
        inspect d when eventnotice do
           begin upd2(BL); upd2(LL); upd2(RL) end
        when driver do
           begin
               upd2(drex); upd2(drp); upd2(evp);
               upd2(cdrp);
               if md then obj.MDP :- this driver
           end;
        if d =/= POOL2TOP then begin d :- d-drlg; go to nextnotice
        end;
        upd2(CD);
        update display;
        nothead :- none;
        if req > POOL2TOP-POOL1LAST then
                error ("storecollapse",1);
      end storecollapse
   end runtime system
```