SPECIFICAREA FORMALĂ A SISTEMELOR CU PROCESE SECVENŢIALE COMUNICANTE (II)

Scopul lucrării :

- Exemplificarea utilizării limbajului Estelle pentru modelarea unui sistem de timp real complex (alocator de resurse)

1. Descrierea generală a alocatorului de resurse

Se va modela un sistem de telecomunicaţii ilustrat în figura 1.

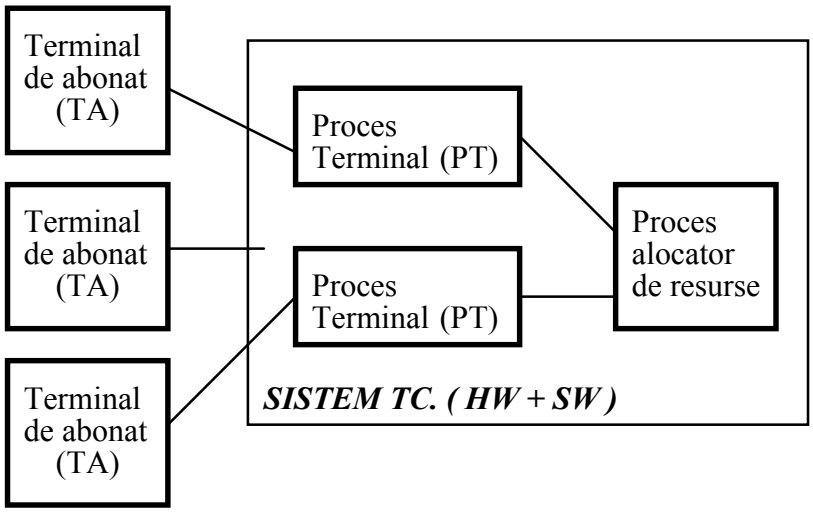

Figura 1. Arhitectura sistemului

Sistemul are în compunere N terminale de abonat (TA); fiecare terminal de abonat are alocat cîte un proces de tratare terminală (PT).

Printre alte prelucrări, aferente cererilor unui terminal TA(i), pentru procesul PT(i) permite accesul TA la o resursă comună dintr-un grup de M < N resurse. Alocarea unei resurse se face în urma unei cereri adresate de către un terminal TA către procesul aferent.

Procesul terminal, PT, la rîndul său face o cerere către un proces alocator de resurse (PAR) care gestionează alocarea resurselor astfel :

- **există resursă liberă** : PAR alocă o resursă dacă are vreuna disponibilă şî face cunoscută identitatea acesteia către PT apelant. Procesul terminal returnează un răspuns către TA, după care TA începe prin intermediul lui PT utilizarea resursei.
- **nu există resursă liberă** : PAR răspunde lui PT cu refuz.

Procesul terminal face K-1 reîncercări (noi cereri) după fiecare interval de timp T1. Dacă după K încercări nu s-a obţinut o resursă atunci PT răspunde lui TA cu refuz şi trece în repaus.

O resursă ocupată se va utiliza un timp TR(i) de către TA(i), după care este eliberată la cererea TA.

Timpul maxim de ocupare a unei resurse este TRMAX unităţi de timp. O resursă menţinută ocupată mai mult decît TRMAX va fi eliberată automat de către PAR, iar

procesul PT aferent este anunțat despre aceasta. La rîndul său PT va genera un mesaj de *reset* către TA.

Există relația TR(i) ≤ TRMAX .

2. Modelul Estelle al sistemului

Modelul Estelle al sistemului de telecomunicații este prezentat în figura 2.

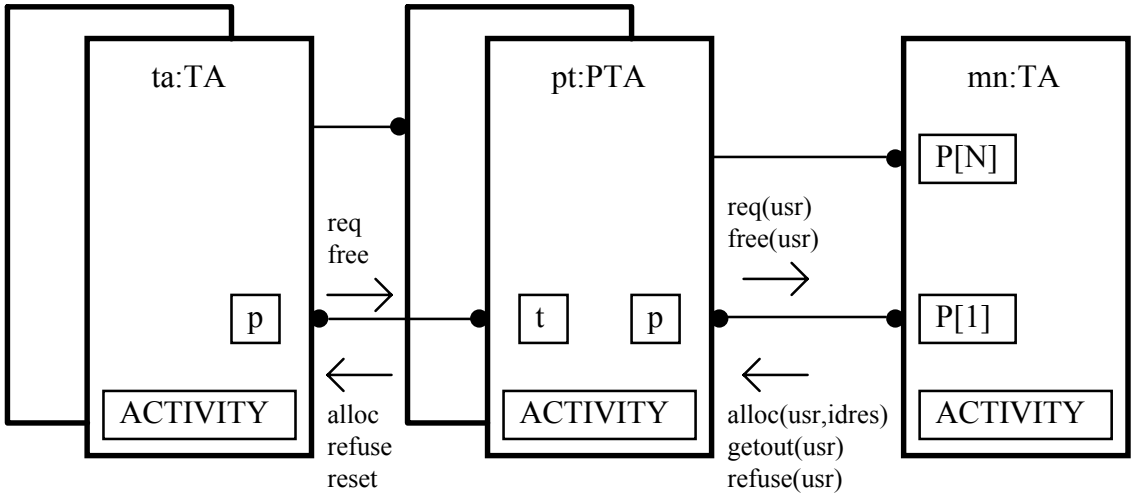

Figura 2. Modelul Estelle al sistemului

Există N module de tip TA (terminal de abonat), N module de tip PTA (proces de tratare abonat) şî un modul de tip MNG (management - pentru alocarea celor M resurse comune).

Între aceste module există următoarele interacțiuni :

- între modulele ta - pt :

req - cerere a unui terminal către procesul asociat pentru o resursă
free - eliberarea resursei de către terminal
alloc - alocarea unei resurse terminalului
refuse - refuzul alocării unei resurse
reset - resetarea unui terminal din cauza depăşirii timpului de utilizare a resursei

- între modulele pt - mng :

req(usr) - cerere de resursă către *mng*, de la procesul cu identitatea *usr*
free(usr) - eliberarea resursei folosite de procesul *usr*
alloc(usr,idres) - alocarea resursei cu identitatea *idres* pentru procesul *usr*
getout(usr) - comanda de eliberare forțată a resursei de către *mng* datorită depăşirii timpului

Graful simplificat pentru modulele modelul Estelle al sistemului este ilustrat în figura 3.

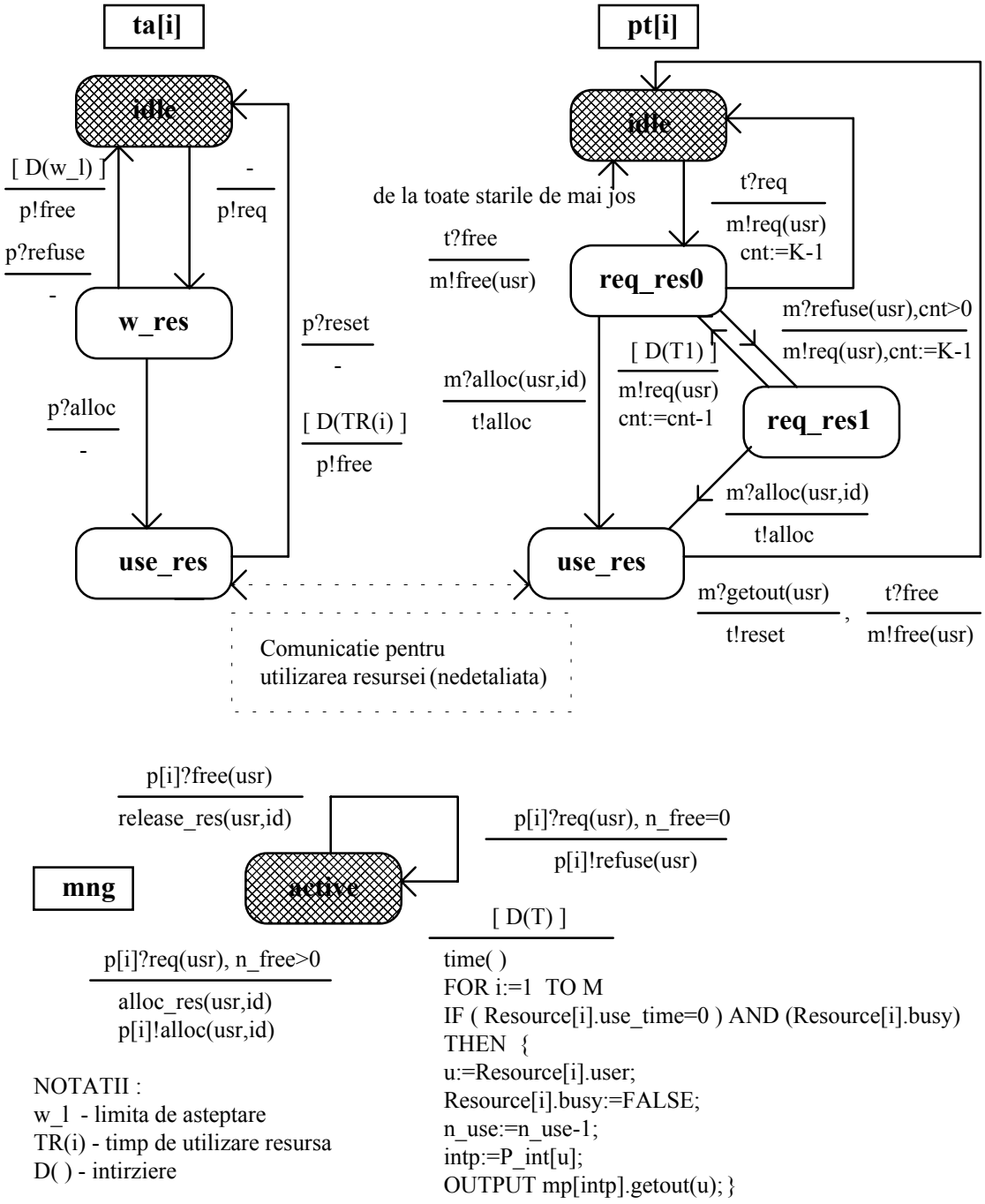

Figura 3. Graful simplificat al proceselor

Modulul **ta** are următoarele stări principale : **idle** - repaus, **w_res** - aşteaptă alocarea unei resurse, **use_res** - utilizarea resursei. Cererile pentru resursă pot fi generate periodic cu o întîrziere *req_delay* > *t_use_res* (timpul de utilizare a resursei) sau aleator - cu o probabilitate dată de *p_req* (dacă *random_req_en* = TRUE). Cererile aleatore se pot genera la un interval de timp fixat sau aleator.

Resursa cerută este aşteptată un interval de timp dat de *wait_limit*, dacă *wait_limit_en* = TRUE.

Modulul **pt** are stările principale : **idle** - repaus, **req_res0, req_res1** - aşteptarea alocării unei resurse de către *mng*, **use_res** - utilizarea resursei de către proces.

Modulul **mng** are o singură stare (activă) şi utilizează următoarele proceduri :

- **alloc_res**(*usr*,*id*) - alocă resursa *id* procesului *usr*, returnează *id*.
- **release_res**(*usr*,*id*) - eliberează resursa *id* ocupată anterior de procesul *usr*.
- **time()** - decrementează toți contorii care măsoară timpul de utilizare a resursei.

Variabila *n_use* conține numărul de resurse utilizate, iar variabila mean_usage conține utilizarea medie a resurselor calculată astfel :

$$M_k = \frac{x_1 + x_2 + \ldots + x_k}{k} = \frac{x_1 + x_2 + \ldots + x_k}{k} \cdot \frac{k-1}{k-1} =$$

$$= \frac{x_1 + x_2 + \ldots + x_{k-1}}{k-1} \cdot \frac{k-1}{k} + \frac{x_k}{k} = M_{k-1} \cdot \frac{k-1}{k} + \frac{x_k}{k}$$

Fiecare resursă are asociată o structură de date de tip înregistrare cu următoarele cîmpuri :

busy : BOOLEAN	{ indică dacă resursa e ocupată sau nu}
user	{ indică procesul ce utilizează resursa}
use_time	{indică timpul de utilizare}

3. Specificaţia Estelle a alocatorului de resurse

{res_mng.stl} - specificaţia Estelle

```
SPECIFICATION  Resource_Manager SYSTEMPROCESS;

DEFAULT INDIVIDUAL QUEUE;

TIMESCALE SECONDS;

CONST
   N  = 8;      { nr. terminale}
   M  = 4;      { nr. resurse}

    K = 8;      { nr. Max. de reincercari de cerere de catre procesul de tratare}
   T = 1;               {cuanta de timp a ceasului local al managerului }

{ Estelle transitions priorities }
  High = 0;
  Medium = 1;
  Low = 2;

TYPE
       User_type = 1..N;
       Id_res_type = 1..M;

VAR
```

```
        PROC_ADDR: ARRAY[User_type] OF INTEGER;


PROCEDURE init_proc_adr; {aloca adrese proceselor}
    VAR i: INTEGER;
        BEGIN
          FOR i:= 1 TO N DO   PROC_ADDR [i]:= i;
{val. arbitrara, modificabila prin comenzi Edb, pentru fiecare terminal in parte}
        END;

CHANNEL term_proc_ch (term, user_proc);
 BY term: req; free;
 BY user_proc: alloc; refuse; reset;
{-------------------------------------------}
CHANNEL  proc_mng_ch(user_proc, manag);
 BY  user_proc:
        req(usr:User_type);
        free(usr:User_type);
 BY manag:
        alloc(usr:User_type; id:Id_res_type);
        refuse(usr:User_type);
        getout(usr:User_type);
{-------------------------------------------}

{---------------------------------}

FUNCTION  proba : REAL;          PRIMITIVE;
PROCEDURE init_proba(v: INTEGER); PRIMITIVE;
FUNCTION  local_time:REAL;       PRIMITIVE;
FUNCTION  rtrunc(v:REAL):INTEGER; PRIMITIVE;

 (***** Modul  terminal *****)

MODULE Terminal ACTIVITY;
 IP tp :  term_proc_ch (term);
END;
{-------------------------------------------------------------------}
BODY terminal_body  FOR terminal;

   STATE
        idle, w_res0, w_res, use_res;
  { STATESET}

   VAR del_req_en:BOOLEAN;       {validare intarziere a urmatoarei cereri}
      req_delay0:INTEGER;     {interval constant de generare a cererilor}
      req_delay:INTEGER;     {interval variabil de generare a cererilor}

        random_time_req_en:BOOLEAN;{valid.cereri la intervale req_delay variabil}
        random_req_en:BOOLEAN;  {valid gener. de cereri in mod}
                                { aleator cu prob. p_req la fiecare incercare}
```

```
        p_req:REAL;             {prob. de efectuare  a unei cereri}


        wait_limit_en:BOOLEAN;  {validare a asteptarii limitate in timp}
        wait_limit:INTEGER;     {limita de timp de asteptare}

        t_use_res:INTEGER;{timpul de utilizare a resursei de catre terminal}
        mainstat:INTEGER;   {indica starea principala a modulului}
        {mainstat = 0/1/2 --> idle/w_res/use_res}
        u:REAL;                 {var auxiliara}
        time_stamp: REAL;

{---------------------------------}

   PROCEDURE InitVar;
        { valorile marcate cu # sunt arbitrare, modificabile prin comenzi Edb}
      BEGIN
          req_delay0:= 6;           { # dar trebuie ca req_delay > t_use_res}
          req_delay:=req_delay0;
          random_time_req_en:=FALSE;   {# FALSE => cereri periodice}

        random_req_en:= FALSE;      {#  cerere la fiecare interval}
        p_req:= 0.7;                {#}

          wait_limit_en:=FALSE;     {# FALSE => invalid. limita de timp de ast}
          wait_limit:=3;            {#}

          t_use_res:=4;             {#}
          mainstat:=0;
          time_stamp:=local_time;
      END;
{---------------------------------------}
 INITIALIZE
   TO idle
    BEGIN
        InitVar;
    END;

{---------------------------------------}

 TRANS
  FROM idle
   TO SAME
     PROVIDED del_req_en
        DELAY(req_delay) {urmatoarea cerere posibila - peste req_delay unit.
                    de timp}
 {0}   NAME delay_up_to_next_possible_req:
          BEGIN
           del_req_en:= FALSE;
          END;
{---------------------------------}
```

SPECIFICAREA FORMALĂ A SISTEMELOR CU PROCESE SECVENȚIALE COMUNICANTE (II)

```
   TO w_res0
    PROVIDED (NOT del_req_en)

 {1}    NAME prepare_user_request:
          BEGIN
            u:= proba;    {val. aleatoare in gama [0.0, 1.0]}
          END;
FROM w_res0
  TO w_res

         PROVIDED (u < p_req) OR (NOT random_req_en)
 {2}   NAME user_generates_request:
         BEGIN
            OUTPUT tp.req;
            mainstat:=1;
         END;
   TO  idle
         PROVIDED u >= p_req
 {3}   NAME user_abort:
       BEGIN
        del_req_en:= TRUE; { valideaza intarz. pana la urm. cerere}
       END;
{---------------------------------}
TRANS

  FROM w_res To idle
     PRIORITY High
     PROVIDED wait_limit_en
     DELAY(wait_limit){ abandon al cererii inainte de servire}
 {4}  NAME user_aborts_request:
           BEGIN
              OUTPUT tp.free;
              del_req_en:= TRUE; {decide sa mai astepte in starea idle}
                 mainstat:=0;
           END;
{---------------------------------}
TRANS
   FROM w_res TO idle
        PRIORITY high
        WHEN tp.refuse
{5}   NAME service_refused:
           BEGIN
         del_req_en:= FALSE;
         mainstat:=0;
       END;
{---------------------------------}
TRANS
  FROM w_res TO use_res
     WHEN tp.alloc
 {6}   NAME user_receives_access_to_resource:
       BEGIN mainstat:=2;   END;
```

```
{---------------------------------}

TRANS
  FROM use_res TO idle
    PRIORITY high
    WHEN tp.reset
 {7}  NAME user_is_forced_out_by_mng:
    BEGIN  mainstat:=0;  END;
{---------------------------------}
TRANS

FROM use_res TO idle
    DELAY(t_use_res)
        VAR x:REAL;
{8}   NAME user_ends_using_resource:
    BEGIN
       OUTPUT tp.free;
       del_req_en:= TRUE;
          mainstat:=0;
        IF  random_time_req_en  THEN
          BEGIN
            x:=proba;
            req_delay := rtrunc(x * req_delay0);
          END
        ELSE  req_delay:=req_delay0;
    END;
{---------------------------------}

{INTERACTIUNI IGNORATE - nerelevante in starea respectiva}
TRANS
  FROM idle
         WHEN tp.reset          BEGIN  END;
      WHEN tp.alloc   BEGIN  END;
         WHEN tp.refuse        BEGIN  END;

TRANS
  FROM w_res
    WHEN tp.reset     BEGIN  END;
    WHEN tp.refuse   BEGIN  END;

TRANS
  FROM use_res
    WHEN tp.alloc     BEGIN  END;
        WHEN tp.refuse        BEGIN  END;

END; {terminal body}
{-------------------------------------------------}

 (***** Modul  Call_process*****)

 MODULE Call_process ACTIVITY;
```

SPECIFICAREA FORMALĂ A SISTEMELOR CU PROCESE SECVENȚIALE COMUNICANTE (II)

```
   IP  pt:  term_proc_ch (user_proc);
      pm: proc_mng_ch(user_proc);
   EXPORT proc_id:INTEGER;  { user terminal  process identity}
 END;
{---------------------------------}

 BODY Call_process_body FOR  Call_process ;

VAR
        cnt:INTEGER;            {contor de reincercari de cerere}
        errcode:INTEGER;  {cod de eroare}
        mainstat:INTEGER; {val. a var. principale de stare}
      T1:INTEGER;       {interval de timp pentru reincercarea cererii}

   STATE
        idle, req_res0, req_res1, use_res, error;

   STATESET
        User_release = [req_res0, req_res1, use_res];
        Req_ignore = [ req_res0, req_res1, use_res, error ];
        Getout_ignore = [ idle, req_res0, req_res1];
{---------------------------------}

INITIALIZE

  TO idle
   BEGIN
        mainstat:=0;
        T1:=1;
   END;
{---------------------------------}
TRANS
   FROM idle TO req_res0
        WHEN pt.req
{0}     NAME  user_request_received:
          BEGIN
              OUTPUT pm.req(proc_id);
              cnt:= K-1;
                 mainstat:=1;
           END;
{---------------------------------}

TRANS
   FROM req_res0 TO req_res1

        WHEN pm.refuse(usr)
        PROVIDED (cnt > 0)
{1}     NAME req_refused_by_mng_and_decide_to_retry:
          BEGIN
            OUTPUT pm.req(proc_id);
            cnt:= K-1;
```

```
            mainstat:=2;
          END;
{---------------------------------}
TRANS
  FROM req_res0 TO idle
       PROVIDED (cnt = 0)
{2}    NAME fail_after_K_retries:
         BEGIN
           OUTPUT pt.refuse;
           mainstat:=0;
         END;
{---------------------------------}
TRANS
  FROM User_release TO idle
       WHEN pt.free
{3}    NAME user_release_request:
         BEGIN
           OUTPUT pm.free(proc_id);
           mainstat:=0;
         END;
{---------------------------------}
TRANS
  FROM req_res1 TO  req_res0
       DELAY(T1)

{4}    NAME delay_to_the_next_retry:
         BEGIN
           OUTPUT pm.req(proc_id);
           cnt:=cnt-1;
           mainstat:=1;
         END;
{---------------------------------}
TRANS

   FROM req_res0, req_res1 TO  use_res
     WHEN pm.alloc(usr, id)
       PROVIDED usr= proc_id
{5}    NAME request_serviced:
         BEGIN
           OUTPUT pt.alloc;
           mainstat:=3;
         END;
{---------------------------------}
TRANS
  FROM req_res1 TO  error
     WHEN pm.alloc(usr, id)
       PROVIDED usr<> proc_id
{6}    NAME request_serviced_addr_error:
         BEGIN
           errcode:= 1; {eroare de adresa }
           OUTPUT pt.reset;
```

```
            mainstat:=4;
          END;
{---------------------------------}
TRANS

FROM use_res TO  idle
      WHEN pm.getout(usr)
        PROVIDED usr= proc_id
{7}     NAME mng_forces_getout:
          BEGIN
            OUTPUT pt.reset;
            mainstat:=0;
          END;
{---------------------------------}
TRANS
   FROM use_res TO  error
      WHEN pm.getout(usr)
        PROVIDED usr<> proc_id
{8}     NAME mng_forces_getout_addr_error:
          BEGIN
            errcode:= 1; {eroare de adresa }
            OUTPUT pt.reset;
            mainstat:=4;
          END;
{---------------------------------}
TRANS
   FROM use_res TO  idle
      WHEN pt.free
{9}     NAME release_request_from_user:
          BEGIN
             OUTPUT pm.free(proc_id);
          END;
{---------------------------------}
{INTERACTIUNI IGNORATE - nerelevante sau nepermise in starea respectiva}
TRANS
   FROM idle
        WHEN pm.alloc(usr,id)       BEGIN  END;
        WHEN pm.refuse(usr)         BEGIN  END;
        WHEN pt.free                BEGIN  END;
  FROM Req_ignore
        WHEN pt.req           BEGIN  END;

  FROM Getout_ignore
        WHEN pm.getout(usr)         BEGIN  END;
  FROM use_res
        WHEN pm.alloc(usr,id)       BEGIN  END;
        WHEN pm.refuse(usr)         BEGIN  END;
  END; { Call_process_body }
{-----------------------------------------------------}
 (***** Modul  Manager*****)
```

```
 MODULE Manager ACTIVITY;
   IP  mp: ARRAY [User_type] OF proc_mng_ch(manag);
 END;

 BODY Manager_body FOR  Manager;
TYPE
        Res_type=RECORD
                busy:BOOLEAN;
                user: User_type;
                use_time: INTEGER;
                END;

VAR     TRMAX:INTEGER;   {interval maxim de timp permis pentru utilizarea unei}
{resurse}
        n_use:INTEGER;   { numar de resurse alocate}

        Resource: ARRAY [Id_res_type ] OF Res_type;
        P_int : ARRAY[User_type] OF User_type; {vector de corespondente}
                        { proces  terminal  -> punct de interactiune asociat }
        nr_release_err:INTEGER; {contor erori eliberare}
        nr_alloc_err:INTEGER; {contor erori alocare}
        id_resource:Id_res_type;
        mean_usage:REAL; {utilizare medie a resurselor}
        count:INTEGER;
        irnd:INTEGER;

{---------------------------------}

   PROCEDURE alloc_res(usr: User_type; VAR id_res:Id_res_type;VAR ok:BOOLEAN);
        VAR i:INTEGER;
        BEGIN
         i:= 1;
        WHILE (i <= M ) AND Resource[i].busy DO i:=i+1;
        IF i < ( M+1) THEN
          BEGIN
           Resource[i].busy:= TRUE;
           Resource[i].user:= usr;
           Resource[i].use_time:=TRMAX;
           n_use:= n_use + 1;
           id_res:= i; ok:= TRUE;

          END
        ELSE ok:=FALSE;
END;
{---------------------------------}

 PROCEDURE  id_res_to_be_released(usr:User_type; VAR id_res:Id_res_type; VAR
ok:BOOLEAN);
        VAR i:INTEGER;
        BEGIN
          i:= 1;
```

```
        WHILE (i <= M ) AND ((Resource[i].user <> usr) OR ( NOT Resource[i].busy))

          DO i:=i+1;
      IF i < ( M+1)
            THEN BEGIN  id_res:= i; ok:=TRUE; END
              ELSE  ok:=FALSE; {error}
    END;

PROCEDURE release_res(i:Id_res_type );
        BEGIN
          Resource[i].busy:= FALSE;
          Resource[i].user:= 1;  {val. arbitrara}
          Resource[i].use_time:=0;
          n_use:= n_use - 1;
        END;

 FUNCTION  in_use(usr : User_type):BOOLEAN;
        VAR i:INTEGER;
        BEGIN
          i:= 1; in_use:=FALSE;
         WHILE (i <= M ) AND ((Resource[i].user <> usr) OR ( NOT Resource[i].busy))
          DO i:=i+1;
      IF i < ( M+1)     THEN in_use:=TRUE;
    END;



{---------------------------------}
  PROCEDURE time(VAR mean_us:REAL; VAR cnt:INTEGER);
        VAR i:INTEGER;
        BEGIN
        FOR i:= 1 TO M DO
        BEGIN
              IF   (Resource[i].busy) AND (Resource[i].use_time > 0) THEN
                    Resource[i].use_time:= Resource[i].use_time - 1;
        END;
         cnt:= cnt + 1; {calcul medie de utilizare}
         IF cnt > 1 THEN
         mean_us:= (mean_us*(cnt - 1) + n_use)/cnt
         ELSE mean_us:= n_use/cnt;
         END;
{---------------------------------}
STATE active;

INITIALIZE TO active
VAR i:INTEGER;
  BEGIN
        TRMAX:=15;

     nr_release_err:= 0;
        nr_alloc_err:= 0;
```

```
      n_use:=M;
         FOR i:= 1 TO M DO release_res(i);
         FOR i:= 1 TO N DO P_int[i]:=i; {nr. punctului de interactiune mp
                  asociat procesului i. in particular aici sunt identice}
         irnd:=3243;     { Valoare initiala a generatorului <- Edb }
      init_proba(irnd);
         mean_usage:=0;
         count:=0;


   END;
{---------------------------------}
TRANS
   FROM active
      ANY n:User_type DO
         WHEN mp[n].req(usr)
         PROVIDED   (n_use < M) AND (NOT in_use(usr))
         PRIORITY high
         VAR
            rs: Id_res_type;
            ok:BOOLEAN;
{0-7}   NAME user_request_and_res_available:
          BEGIN
             alloc_res(usr, rs, ok);
             IF ok THEN OUTPUT mp[n].alloc(usr,rs)
             ELSE
                  BEGIN
             nr_alloc_err:= nr_alloc_err +1;
               OUTPUT mp[n].refuse(usr);
                  END;
       END;

{---------------------------------}

         PROVIDED   (n_use = M) AND (NOT in_use(usr))
{8-15} NAME user_request_and_all_res_busy:
          BEGIN

             OUTPUT mp[n].refuse(usr);
          END;
{---------------------------------}
         PROVIDED   in_use(usr)
{16-24}          NAME user_in_service_so_ignore_request:
          BEGIN
          END;

{---------------------------------}
TRANS
   FROM active TO SAME
      ANY n:User_type DO
         WHEN mp[n].free(usr)
```

```
        VAR
           idrs:Id_res_type;
           ok:BOOLEAN;
{25-32}         NAME user_release_resource:
         BEGIN

           id_res_to_be_released(usr, idrs, ok);
           IF ok  THEN  release_res(idrs)
           ELSE  nr_release_err:= nr_release_err +1;
          END;

TRANS
  FROM active TO SAME
        DELAY(T)
        VAR i:INTEGER;
           u:User_type;
           intp:User_type;
{33}    NAME timed_actions:

 BEGIN
  time(mean_usage, count);
  FOR i:= 1 TO M DO
        IF (Resource[i].use_time =0) AND (Resource[i].busy)
          THEN BEGIN
          u:= Resource[i].user;
          Resource[i].busy:= FALSE;
          n_use:= n_use - 1;
          intp:= P_int[u]; {nr.punctului de interactiune asociat}
          OUTPUT mp[intp].getout(u);
          END;
  END;

END; {Manager_body }
{-----------------------------------------------------}
(***** Initializare sistem  *****)

 MODVAR
  term :  ARRAY[1..N] OF terminal;
  call_pr : ARRAY[1..N] OF Call_process;
  mng: Manager;

 INITIALIZE
 VAR i:INTEGER;
  BEGIN
  init_proc_adr;
  INIT mng WITH Manager_body;
  FOR i:= 1 TO N   DO
    BEGIN

        INIT term[i] WITH terminal_body  ;
        INIT  call_pr[i]  WITH  Call_process_body ;
```

```
        call_pr[i].proc_id:= PROC_ADDR[i];

       {alocare de identitati proceselor de tratare }
        CONNECT term[i].tp TO call_pr[i].pt;
     CONNECT call_pr[i].pm TO mng.mp[i];
    END;
 END;
END. { SPECIFICATION }
```

{prim.c} **- funcţii primitive (pentru lucrul cu numere aleatore)**

```
#define MAXINT                  0x7fffffff
/*-------------------------------------*/
extern double edb_time;

double local_time()
{
 return edb_time;
}
/*-------------------------------------*/

void init_proba(myseed)
    int myseed;
{
 srand(myseed);

}
/*-------------------------------------*/

double proba()
{
 double val;

 val = rand()%MAXINT + 1;
 return(val/MAXINT);
}
/*-------------------------------------*/

int rtrunc(v)
   double v;
{
  if ((v-(int)v) > 0.5) return (int)v+1;
  else return (int)v;
}
/*-------------------------------------*/
```

{resinit} **- fişier de iniţializare**

```
print ""
```

```
print ""
print "Macros for scenario generation: res_mng.stl"
print ""

$fs:=1000

\% CURBE CU EVOLUTIA VAR. PRINCIPALE DE STARE

\% ALE TERMINALELOR

tmcurve("$edbtime","3->mainstat", 1.0)
tmcurve("$edbtime","5->mainstat", 1.0)

tmcurve("$edbtime","7->mainstat", 1.0)
tmcurve("$edbtime","9->mainstat", 1.0)
tmcurve("$edbtime","11->mainstat", 1.0)
tmcurve("$edbtime","13->mainstat", 1.0)
tmcurve("$edbtime","15->mainstat", 1.0)
tmcurve("$edbtime","17->mainstat", 1.0)

\% CURBA CU EVOLUTIA NR. DE RESURSE IN UZ CURENT
  tmcurve("$edbtime","2->n_use", 1.0)
  tmcurve("$edbtime","2->mean_usage", 1.0)

\%ACTIVARE DE MACROURI SAU OBSERVATORI
\%====================================
init()
\%opp()
tmcurve_1()
tmcurve_2()
tmcurve_3()
tmcurve_4()

tmcurve_5()
tmcurve_6()
tmcurve_7()
tmcurve_8()
tmcurve_9()
tmcurve_10()
```

{init} **- macrodefiniţie pentru iniţializare variabile din program**

```
print "Terminal overriding initialization";\
\%
\%INTERVAL MAXIM DE TIMP INTRE DOUA TENTATIVE DE CERERE
3->req_delay0:=6;\
5->req_delay0:=6;\
7->req_delay0:=6;\
9->req_delay0:=6;\
11->req_delay0:=6;\
13->req_delay0:=6;\
```

```
15->req_delay0:=6;\
17->req_delay0:=6;\
\%
\% VALID. INTERVAL DE TIMP ALEATOR INTRE DOUA TENTATIVE DE
CERERE
  3->random_time_req_en:=true;\
  5->random_time_req_en:=true;\
  7->random_time_req_en:=true;\
  9->random_time_req_en:=true;\
 11->random_time_req_en:=true;\
 13->random_time_req_en:=true;\
 15->random_time_req_en:=true;\
 17->random_time_req_en:=true;\
\%
\% VALIDARE A GENERARII ALEATOARE DE CERERI
 3->random_req_en:=true;\
 5->random_req_en:=true;\
 7->random_req_en:=true;\
 9->random_req_en:=true;\

11->random_req_en:=true;\
 13->random_req_en:=true;\
15->random_req_en:=true;\

17->random_req_en:=true;\
\%
\% VALID. INTERVAL DE TIMP ALEATOR INTRE DOUA TENTATIVE DE
CERERE
  3->random_time_req_en:=true;\
  5->random_time_req_en:=true;\
  7->random_time_req_en:=true;\
  9->random_time_req_en:=true;\
 11->random_time_req_en:=true;\
 13->random_time_req_en:=true;\
 15->random_time_req_en:=true;\
 17->random_time_req_en:=true;\
\%
\% VALIDARE A GENERARII ALEATOARE DE CERERI
 3->random_req_en:=true;\
 5->random_req_en:=true;\
 7->random_req_en:=true;\
 9->random_req_en:=true;\
 11->random_req_en:=true;\

13->random_req_en:=true;\
15->random_req_en:=true;\
17->random_req_en:=true;\
\%
\% PROB DE GENERARE A UNEI CERERI LA O TENTATIVA
 3->p_req:=0.5;\
 5->p_req:=0.5;\
```

```
 7->p_req:=0.5;\
 9->p_req:=0.5;\
 11->p_req:=0.5;\
 13->p_req:=0.5;\
 15->p_req:=0.5;\
 17->p_req:=0.5;\
\%
\% LIMITA DE TIMP DE ASTEPTARE A SATISFACERII CERERII
\%
\% TRUE DACA SE DORESTE LIMITAREA ASTEPTARII
\%  3->wait_limit_en:=true;\
\%  5->wait_limit_en:=true;\
\%  7->wait_limit_en:=true;\
\%  9->wait_limit_en:=true;\
\% 11->wait_limit_en:=true;\
\% 13->wait_limit_en:=true;\
\% 15->wait_limit_en:=true;\
\% 17->wait_limit_en:=true;\
\%
\% VAL. TIMPULUI LIMITA DE ASTEPTARE
\%  3->wait_limit;=3;\
\%  5->wait_limit;=3;\
\%  7->wait_limit;=3;\
\%  9->wait_limit;=3;\
\% 11->wait_limit;=3;\
\% 13->wait_limit;=3;\
\% 15->wait_limit;=3;\
\% 17->wait_limit;=3;\
\%
\%
\% VAL. TIMPULUI DE UTILIZARE A RESURSEI

\%  3->t_use_res;=4;\
\%  5->t_use_res;=4;\
\%  7->t_use_res;=4;\
\%  9->t_use_res;=4;\

\% 11->t_use_res;=4;\
\% 13->t_use_res;=4;\
\% 15->t_use_res;=4;\
\% 17->t_use_res;=4;\
```

{opp} **- observator pas cu pas**

```
OBS opp

print "****************************************";\
 d $lins; d $ltrid; d $ltri;d $ltro;\
print "------------------";\
 print "(terminal1)";\
 d 3->mainstat;\
```

```
 d 3->u;\

 print "------------------";\
 print "(terminal2)";\
 d 5->mainstat;\
 d 5->u;\
 print "------------------";\
 print "(terminal3)";\
 d 7->mainstat;\
 d 7->u;\
 break;\
```

{pl} - comanda afişare curbe

```
d $h
d 2->Resource
d 2->mean_usage

plot tmcurve_1
plot tmcurve_2
plot tmcurve_3
plot tmcurve_4
plot tmcurve_5
plot tmcurve_6
plot tmcurve_7
plot tmcurve_8
plot tmcurve_9
plot tmcurve_10
```

4. Desfăşurarea lucrării

a) Se va studia descrierea generală şi modelul Estelle al alocatorului de resurse (secțiunile 1 şi 2).

b) Se va analiza specificația Estelle a sistemului (secțiunea 3) şi se va simula execuția utilizînd mediul de dezvoltare XEDT . Se va selecta , în simulator , opțiunea **- Lprim.o**.

Se va rula macrodefiniția init (comanda *init()*, se va executa comanda *resinit()* - la prompt-ul **edb** şi se va activa / dezactiva observatorul *opp*.

Se vor urmări următoarele aspecte :

- definirea observatorilor şi a macrodefinițiilor
- efectuarea tranzițiilor (pe modul pas cu pas sau continuu)

c) Se vor analiza curbele obținute în urma simulării diverse situații . Afişarea curbelor se va face cu comanda *pl()* la prompt-ul **edb**.