SPECIFICAREA FORMALĂ A SISTEMELOR CU PROCESE SECVENŢIALE COMUNICANTE (I)

Scopul lucrării :

- Descrierea limbajului ESTELLE şi a mediului de dezvoltare XEDT (Estelle Development Tools)
- Exemplificarea utilizării XEDT pentru descrierea formală a proceselor secvențiale comunicante

1. Descrierea generală a limbajului Estelle

Limbajul *Estelle* este un limbaj de descriere formală a proceselor secvențiale comunicante.

Estelle reprezintă o extensie a modelului clasic de automat cu stări şi permite, în plus, modelarea timpului, paralelismului, şi a comunicației între procese.

Limbajul *Estelle* constituie una dintre tehnicile de specificare formală (independentă de implementarea finală) a sistemelor modelabile ca maşini comunicante, abstracte, cu stări (sisteme de telecomunicații, în timp real, distribuite). Scopul specificației formale este verificarea corectitudinii funcționării sistemului şi validarea (concordanța funcționării cu specificațiile inițiale), înainte de a face implementarea acestuia.

1.1. Realizarea comunicaţiei

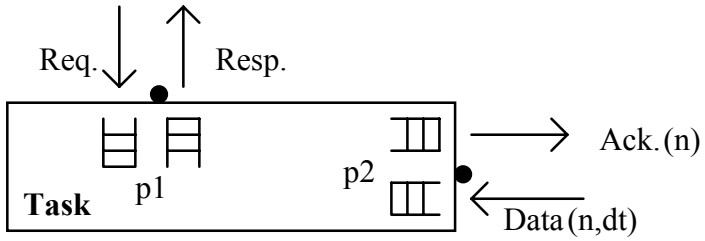
O specificație *Estelle* descrie comportamentul unei colecții de componente , numite *module* , ce comunică între ele. Un *modul* defineşte un task (ca tip de componentă) al sistemului. Un exemplu particular de task se numeşte *instanțiere (instanță) de modul*.

Un modul poate avea descendenți (module incluse), rezultînd astfel o *ierarhie de module*. Corespunzător ierarhiei de module vom avea şi ierarhii de instanțieri ale acestora.

Fiecare modul poate avea *puncte de interacțiune (IP)* cu alte module sau cu mediul.

Este posibilă o comunicare bidirecțională între task-uri (module) prin *canale* ce leagă punctele de interacțiune. Pe canale circulă *mesaje* numite *interacțiuni* ce pot avea parametii.

Reprezentarea grafică a unui modul este prezentată în figura 1.



p1, p2 - puncte de interactiune

Request (Req.), Response (Resp.), Acknowledge (Ack.), Data - interactiuni (mesaje)

Figura 1. Reprezentarea grafică a unui modul

Modulele sînt caracterizate prin interfața cu mediul înconjurător (realizată prin puncte de interacțiune, canale şi mesaje sau prin variabile exportate de la un modul ierarhic inferior la un modul ierarhic superior) şi prin *comportamentul intern.*

În textul specificației interfața şi comportamentul intern vor fi specificate respectiv prin *antetul* şi *corpul* modulului.

Comunicația între module presupune un transfer asincron (emisia în orice moment) de mesaje pe canale între 2 puncte de interacțiune. Fiecare punct de interacțiune are la intrare o coadă FIFO asociată , de lungime infinită. Cozile FIFO pot fi individuale sau asociate mai multor puncte de interacțiune (opțiune).

Legăturile de interconectare prin canale sînt specificate prin operațiile de atașare (pentru module fiu - părinte) sau conectare (pentru module de acelaşi rang). Nu sînt permise legături multipunct.

Un modul poate să îşi modifice în timpul funcționării structura internă prin creare / distrugerea de submodule şi a legăturilor (canalelor) de comunicație asociate.

Comportamentul intern este determinat de structura internă a modulului. Structura internă este alcătuită din : declarații de submodule şi variabile de stare şi descrierea comportamentului modulului (ca automat extins cu număr finit de stări); se indică setul de tranziții posibile şi prelucrările asociate acestor tranziții.

1.2. Modelarea paralelismului

Paralelismul sistemelor reale se modelează prin *atribute* asociate modulelor. Un modul activ (care are cel puțin o tranziție) trebuie să aibă atribut.

Un modul sincronizează activitatea submodulelor sale , conform atributului său, doar el le poate crea / distruge, iar tranzițiile sale au prioritate la execuție față de cele ale submodulelor.

Atributele ce definesc paralelismul intern al unui modul sînt :

PROCESS - submodulele sale evoluează paralel şi sincron (se execută în paralel toate tranzițiile execuatbile propuse de submodule)

ACTIVITY - submodulele sale evoluează paralel şi asincron prin intercalarea aleatoare a tranzițiilor.

Atributul **SYSTEM** defineşte în plus pentru modulul în cauză un comportament independent şi asincron față de **alte module sistem** . Rezultă mulțimea de atribute : **SYSTEMPROCESS (SP)**, **SYSTEMACTIVITY (SA)**, **PROCESS (P), ACTIVITY (A).**

Un modul sistem nu poate fi inclus într-un modul cu atribut. Rezultă că structura de sisteme este *statică.* În schimb un modul PROCESS/ACTIVITY este fiu al unui modul sistem.

Sînt permise doar următoarele descendențe (pentru consistența modelului) :

SP -	P	SA -	A	P -	P	A -	A
-	A	-	A	-	A	-	A

Modulul *anvelopă* , în general este chiar modulul specivicație. El poate avea sau nu atribut. Acest modul asigură mediul de lucru şi inițializarea submodulelor sale.

1.3. Comportarea internă a modulelor

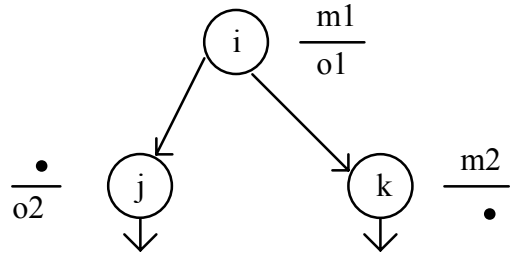
Un modul *Estelle* reprezintă un model de automat extins cu număr finit de stări, nondeterminist.

Starea extinsă conține : variabila de stare propriu-zisă , valorile varialbilelor modulului, conținutul cozilor FIFO asociate punctelor de interacțiune, structura de submodule şi modul lor de interconectare.

Există o mulțime de stări inițiale şi o relație de trecere dintr-o stare în alta (reguli de efectuare a tranzițiilor).

O tranziție include: condiții de execuție (mesaj de intrare, condiție booleană, expirare de timp) şi acțiuni de executat (prelucrări interne, generare de mesaje). Reprezentarea grafică a tranzițiilor este ilustrată în figura 2.

Execuția unei tranziții este neinteruptibilă (atomică).



m1 / o1 — mesaj de intrare (interactiune) / mesaj de iesire (interactiune)

- (la numarator) - nu se asteapta mesaj (tranzitie spontana)
- (la numitor) - nu se genereaza mesaj de iesire

m1,m2 - mesaje (intrari)
o1,o2 - iesiri (semnale generate la iesire)

Figura 2. Reprezentarea grafică a tranzițiilor

1.4. Modelarea timpului

Deoarece duratele de execuție ale tranzițiilor vor depinde de implementare, *Estelle* caută să modeleze sistemele independent de aceaste durate, considerîndu-le necunoscute. O soluție este următoarea :

- tranzițiile se consideră a avea implicit durata de execuție nulă
- nondeterministul evoluției sistemelor reale datorat duratelor de execuție necunoscute (de exemplu o întîrziere printr-o rețea a unui mesaj) sau variabile este modelat prin faptul că simulatorul poate alege aleator orice evoluție posibilă (selecție aleatoare între tranzițiile executabile)
- timpul se modelează cu ajutorul tranzițiilor cu clauză de întîrziere (DELAY) element ce dă posibilitatea unor ordini *relative* a evenimentelor. Se consideră că există un proces abstract, în afara specificației Estelle care măsoară timpul în unități convenționale.

În timpul simulării ceasul sistemului avansează doar în momentele în care s-au executat toate tranzițiile executabile curent şi care au clauză DELAY. Noua valoare a timpului este corespunzătoare întîrzierii specificate de prima tranziție (cu clauză DELAY) executabilă. Clauza de întîrziere specifică o întîrziere de forma :

DELAY(E1) = DELAY(E1,E1) - execuția tranziției se întîrzie cu E1 unități de timp
DELAY(E1,E2) - execuția tranziției se întîrzie cu o valoare ce aparține intervalului [E1,E2] .

În acest fel contează în simulare doar relațiile de timp relative între evenimente.

În plus pentru modelerea comportării sistemelor în timp real există posibilitatea în simulatoarele Estelle de alocare de valori de timp nenule (măsurate în unități de timp convenționale) pentru execuția tranzițiilor şi/sau pentru fazele de management (analiză a tranzițiilor executabile) ale sistemului.

Tranzițiilor li se pot aloca (opțional) priorități pentru execuție.

2. Sintaxa limbajului Estelle

2.1. Definirea canalelor de comunicaţie

În limbajul Estelle canalele specifică seturi de interacțiuni (mesaje).
Definirea unui canal se face astfel:

```
channel Identificator_canal (capăt_A, capăt_B);
        by capăt_A :  m1;
                      m2;
                      .
                      .
                      mN;
        by capăt_B :  n1;
                      n2;
                      .
                      .
                      nK;
        by capăt_A,capăt_B :        k1;
                                    k2;
                                    .
                                    .
                                    kP;
```

unde m1,m2, ..., mN,n1,n2, ..., nK,k1,k2,...,kP sînt tipuri de interacțiuni (mesaje)

Un tip de interacțiune constă dintr-un nume şi posibili parametrii, de exemplu :

```
m1(x:integer ; y:boolean )
```

2.2. Definirea punctelor de interacţiune

Punctele de interacțiune se declară după modelul următor:

```
p1: Identificator_canal (capăt_A)
```

sau

```
p2: Identificator_canal (capăt_B)
```

În primul caz setul de interacțiuni ce pot fi transmise prin punctul de interacțiune p1 conține toate mesajele specificate după " *by capăt_A* " şi " *by capăt_A, capăt_B* "din definiția canalului *Identificator_canal* (adică m1,m2,..mN,k1,k2,...,kP), iar setul de interacțiuni ce pot fi recepționate conține toate mesajele specificate după " *by capăt_B* " şi

" *by capăt_A, capăt_B* "din definiția canalului *Identificator_canal* (adică n1,n2,..nK,k1,k2,...,kP).

Interacțiunile specificate pentru ambele capete pot fi transmise sau/şi recepționate.

În cel de-al doilea caz punctul de interacțiune p2 are un rol opus punctului de interacțiune p1.

Cozile FIFO asociate punctelor de interacțiune se pot declara astfel:

p1: Identificator_canal (capăt_A) **common queue**

sau

p1: Identificator_canal (capăt_A) **individual queue**

În primul caz coada FIFO va fi comună pentru mai multe puncte de interacțiune.

2.3. Definirea modulelor

Un modul este compus din două părți : un antet (*header*) şi un corp (*body*).

Antetul unui modul specifică *tipul modulului*, *punctele de interacțiune* şi *variabilele exporte* :

```
module Nume_modul atribut (parametrii);
        ip      p : T(S) individual queue;
                q : W(K) common queue;
        export X,Y : integer;
end;
```

unde **atribut** poate fi : SISTEMACTIVITY, SYSTEMPROCESS, ACTIVITY sau PROCESS, **ip** desemnează punctele de interacțiune, iar **export** variabilele exportate modulele părinte. Un modul poate avea parametrii cu tip.

Corpul unui modul cuprinde : o parte de declarații, o parte de inițializare şi o parte de tranziții. Se defineşte astfel :

```
body Nume_corp for Nume_modul;
{definirea corpului - vezi secțiunile următoare}
end;
```

sau

```
body Nume_corp for Nume_modul; external;
```

2.3.1. Partea de declarații

Partea de declarații a corpului cuprinde declarații uzuale în Pascal (constante, tipuri, variabile, proceduri şi funcții) şi declarații specifice limbajului Estelle : canale, module (antet şi corp), variabile modul, stări, puncte de interacțiune.

Variabilele modul sînt utilizate pentru crearea şi referirea instanțelor de modul la nivelul specificației Estelle .

De exemplu declarația :

```
modvar X,Y,Z : Nume_modul;
```

indică faptul că X,Y şi Z sînt variabile de tip modul cu antetul Nume_modul.

Stările se pot declara astfel :

state s0,s1,s2,s3;
S-au definit stările s0,s1,s2 şi s3.

Se pot defini *seturi de stări* (**stateset**) care cuprind mai multe stări cu caracteristici comune (de exemplu pentru toate stările din set se efectuază aceeaşi tranziție):

stateset
nume_set_stări = [s0,s3]; { stările s0 şi s3 formează setul *nume_set_stări* }

2.3.2. Partea de inițializare

Partea de inițializare, din structura corpului unui modul, specifică starea inițială şi eventualele valori inițiale ale variabilelor declarate în partea de declarații ; de exemplu dacă starea inițială este s0 şi s-a declarat o variabilă **var** x:**integer** cu valoarea inițială 0 ; se va scrie în partea de inițializare a corpului modulului :

initialize to s0 **begin** x:=0; **end**;

În fiecare modul ce are submodule trebuie introdusă o parte de inițializare ce specifică pentru fiecare variabilă modul tipul de corp asociat şi modul de conectare între module (vezi instrucțiunile **init** şi **connect**) astfel :

```
initialize
        begin
        init Nume_variabilă_Modul1 with Nume_Corp_A;
        init Nume_variabilă_Modul2 with Nume_Corp_B;
        connect     Nume_variabilă_Modul1.punct_interacțiune_a   to
                    Nume_variabilă_Modul2.punct_interacțiune_b

        end;
```

Punctele de interacțiune trebuie să fie capetele unui canal declarat anterior.

2.3.4. Definirea tranzițiilor

Declarația unei tranziții cuprinde : clauzele de execuție a tranziției şi corpul (blocul) tranziției.
Clauzele de execuție a tranziției sînt :

FROM *stare_de_plecare* - indică starea inițială de unde se va efextua tranziția
WHEN *punct_interacțiune.interacțiune* - tranziția se execută dacă există mesajul *interacțiune* în capul cozii FIFO asociată punctului de interacțiune *punct_interacțiune.*
PROVIDED *expersie logică* (sau **PROVIDED OTHERWISE**) - tranziția se execută dacă expresia logică este adevărată .
PRIORITY *p* - indică prioritatea tranziției, *p*

DELAY (*e1,e2*) - se întîrzie execuția tranziției cu o valoare egală cu *e1* sau aparținînd intervalului (*e1,e2*).Valorile *e1* şi *e2* sînt exprimate în număr de unități de timp convenționale. Tranziția se execută dacă celelalte condiții sînt adevărate după expirarea timpului .

TO *stare_de _sosire* - indică starea în care s-a efectuat tranziția.

ANY *domeniu* **DO** - domeniu indică o listă de variabile locale.

Clauza **WHEN** precede clauza **PROVIDED**.

Dacă tranziția se efectuează în aceeaşi stare se pot omite clauza **TO** şi starea de sosire.

O tranziție este validată (într-o stare oarecare) dacă clauzele condiționale (FROM, WHEN, PROVIDED) sînt îndeplinite.

Tranziția fără DELAY validată este executabilă dacă are prioritate maximă. În plus, pentru o tranziție cu clauză DELAY, aceasta este executabilă dacă ea rămîne validată pe toată durata E1 indicată în clauza DELAY şi dacă are prioritate maximă.

Tranzițiile fără WHEN se numesc *spontane*.

Clauzele WHEN şi DELAY nu pot apărea în aceeaşi tranziție (sînt în mod natural mutual exclusive).

Un exemplu de definire a tranzițiilor este prezentat în continuare :

```
trans
        from s0 to s1
        priority 1
        when mesaj_1
                begin
                {prelucrări_0, generează mesaj out0}
                end;

        from s1 to s2
        provided (var_2 > 0 ) and (var_2 < 3)
        priority 0
        begin
        {prelucrări_1}
        end;

        from s2 to s0
        provided(var_3 = 0 )
        delay(6)
        begin
        {prelucrări_2}
        end;
```

Pentru tranzițiile definite mai sus se poate reprezenta graful din figura 3 :

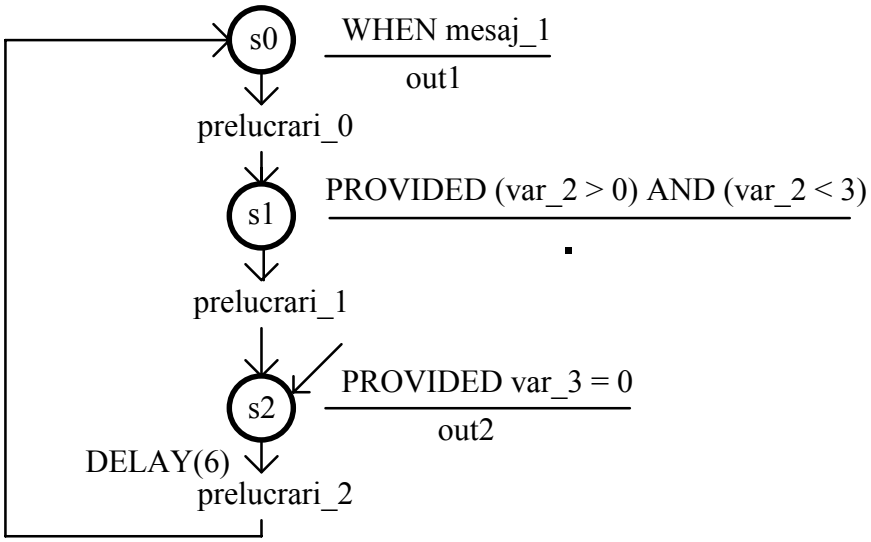


Figura 3. Graful tranzițiilor (exemplu)

2.4. Tipuri de instrucţiuni

Instrucțiunea **INIT** inițializează o variabilă modul cu un corp de modul (ca în secțiunea 2.3.2) astfel :

init Nume_variabilă_Modul1 **with** Nume_Corp_A;

Se crează o nouă instanțiere de modul *Nume_variabilă_Modul1* avînd corpul *Nume_Corp_A*.

Instrucțiunea **CONNECT** conectează două puncte de interacțiune :

connect Nume_variabilă_Modul1.punct_interacțiune_a **to** Nume_variabilă_Modul2.punct_interacțiune_b ;

Se conectează punctul de interacțiune *punct_interacțiune_a* din modulul *Nume_variabilă_Modul1* cu punctul de interacțiune *punct_interacțiune_b* din modulul *Nume_variabilă_Modul2*.

Instrucțiunea **OUTPUT** generează un mesaj de ieşire :

output *punct_interacțiune.identificator_mesaj(parametrii)* ;

Se generează un mesaj dat de *identificator_mesaj(parametrii)* prin punctul de interacțiune *punct_interacțiune*.

Instrucțiunea **DISCONNECT** deconectează o pereche de puncte de interacțiune (în instrucțiune se va specifica unul din capete) sau deconectează toate canalele asociate unei variabile modul (fără a distruge variabila modul) :

disconnect punct_iteracțiune ;

sau

disconnect Nume_variabilă_Modul ;

Instrucțiunile **RELEASE** şi **TERMINATE** distrug variabilele modul specificate în sintaxa instrucțiunii :

release Nume_variabilă_Modul ;

şi

terminate Nume_variabilă_Modul ;

Instrucțiunea **RELEASE** efectuează, înainte de distrugerea modului , toate deconectările canalelor conectate cu modulul respectiv (cu transferul mesajelor din cozi în capetele canalelor rămase) , spre deosebire de instrucțiunea **TERMINATE** care efectuaeză o distrugere abruptă, fără deconectare prealabilă.

3. Structura unei specificaţii Estelle (exemple)

În figura 4 este reprezentat modelul Estelle pentru un sistem cu două procese comunicante:

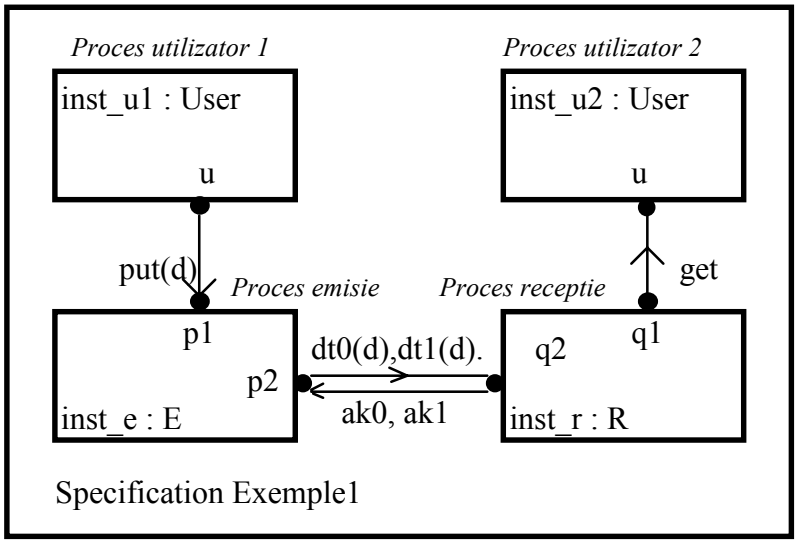


Figura 4. Ierarhia de module Estelle pentru modelarea unei comunicaţii unidirecționale E --> R unidirecționale

put	- mesaj de la procesul utilizator 1 ce conține informația pentru procesul utilizator 2
dt0, dt1	- unități de date numerotate (cu 0 şi 1) pentru a da posibilitatea distingerii unităților succesive la recepție
ak0, ak1	- confirmări pentru dt0 şi dt1
get	- mesaj livrat de receptor procesului utilizator 2
d	- bloc de date (informație de transferat)

Grafurile asociate modulelor de tip E şi I din modelul din figura 4 sînt ilustrate în figura 5 :

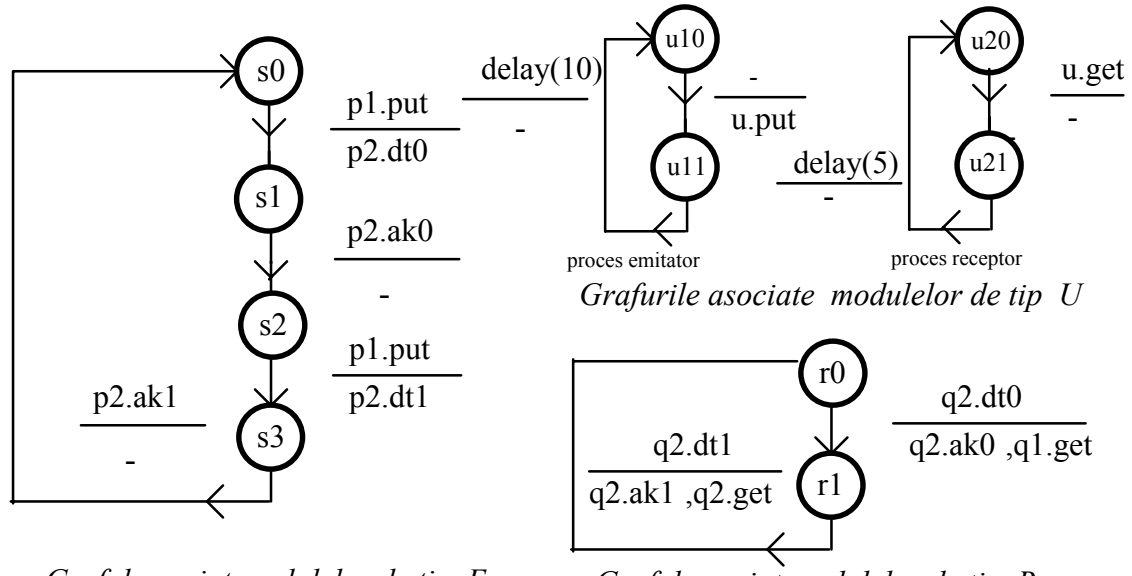


Figura 5. Grafurile asociate modulelor E ,R şi U din specificația Estelle Example1

Specificația Estelle (programul Estelle) pentru modelul din figura 4 este prezentată în continuare :

```
specification Example1;

        default individual queue;
        timescale ms;

        type data_type=integer;

                channel C(r1,r2);
                by r1:put(d:data_type);
                by r2:get(d:data_type);

                channel S(r1,r2);
                by r1:dt0(d:data_type);dt1(d:data_type);
                by r2:ak0;ak1;

        module E systemactivity;
                ip p1:C(r2); p2:S(r1);
        end;
        body E_body for E;
                state s0,s1,s2,s3;
                initialize to s0 begin end;
        trans   when p1.put(d)
                from s0 to s1
                        begin output p2.dt0(d) end;
                from s2 to s3
                        begin output p2.dt1(d) end;
                when p2.ak0
```

```
        from s1 to s2
                begin end;
        when p2.ak1
        from s3 to s0
                begin end;
end;

module R systemactivity;
        ip q1:C(r2); q2:S(r2);
end;
body R_body for R;
        state r0,r1;
        initialize to r0 begin end;
trans
        from r0 to r1
                when q2.dt0(d)
                begin output q2.ak0; output q1.get(d);end;
        from r1 to r0
                when q2.dt1(d)
                begin output q2.ak1; output q1.get(d); end;

end;

module User systemactivity;
        ip u:C(r1);
end;

body User_body1 for User;
var data:data_type;

        state u10,u11;
        initialize to u10 begin data:=0; end;
trans
        from u10 to u11 begin output u.put(data); data:=data+1; end;
        from u11 to u10 delay(10) begin end;
end;

body User_body2 for User;

        state u20,u21;
        initialize to u20 begin end;
trans

        from u20 to u21
                when u.get(d)
                begin end;
        from u21 to u20 delay(5) begin end;

end;
```

```
        modvar inst_e:E;
        modvar inst_r:R;
        modvar inst_u1,inst_u2 : User;

        initialize
        begin
        init inst_e with E_body;;
        init inst_r with R_body;
        init inst_u1 with User_body1;
        init inst_u2 with User_body2;

        connect inst_u1.u to inst_e.p1;
        connect inst_u2.u to inst_r.q1;
        connect inst_e.p2 to inst_r.q2;
        end;
end.
```

Pentru ilustrarea modelelor de paralelism se defineşte modeelul din figura 6.

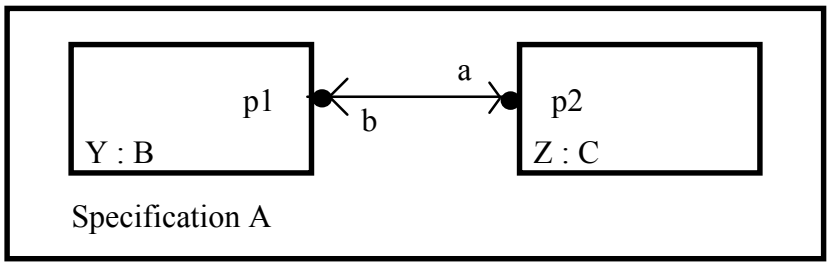


Figura 6. Ierarhia de module Estelle pentru ilustrarea modeleloe de paralelism

Grafurile asociate modulelor de tip B şi C din modelul din figura 6 sînt ilustrate în figura 7 :

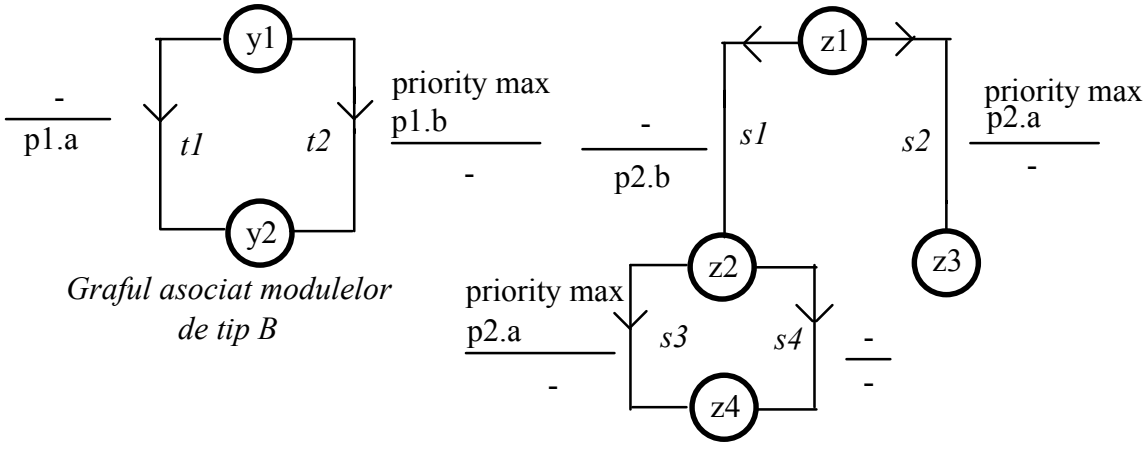


Figura 7. Grafurile asociate modulelor B şi C din specificația Estelle A

Specificația Estelle pentru modelul din figura 6 este prezentată în continuare :

```
specification A  systemprocess;

default individual queue;

const max=0;

channel BC(user1,User2);
        by User1:a;
        by User2:b;

module B process;
        ip p1:BC(User1);
end;

body BB for B;
        state Y1,Y2;
        initialize to Y1 begin end;
        trans from Y1 to Y2
                name t1: begin output p1.a end;
        trans from Y1 to Y2
                priority max
                when p1.b
                name t2:begin end;
        end;

module C process;
        ip p2:BC(User2);
end;

body CC for C;
        state Z1,Z2,Z3,Z4;
        initialize to Z1 begin end;
        trans from Z1 to Z2
                name s1:begin output p2.b end;
        trans from Z1 to Z3
                priority max
                when p2.a
                name s2: begin end;
        trans from Z2 to Z4
                priority max
                when p2.a
                name s3: begin end;
        trans from Z2 to Z4
                name s4: begin end;
        end;

modvar Y:B;Z:C;
initialize
        begin
```

```
        init Y with BB;
        init Z with CC;
        connect Y.p1 to Z.p2;
        end;

end.
```

Se pot studia 3 cazuri conform următorului tabel :

	Cazul 1	**Cazul 2**	**Cazul 3**
Specificaţia A	systemactivity	systemprocess	neatribuit
Modulul B	activity	process/activity	systemproces/ systemactivity
Modulul C	activity	process/activity	systemproces/ systemactivity

În cazul 1. specificația A reprezintă **un singur sistem**. Execuția se deşfăşoară în mod nedeterminist : din toate tranzițiile posibile, la un pas de calcul, se alege una, în mod aleator.

În acest caz setul de secvențe posibile de tranziții executate (admisibile) este :
<t1,s2>, <s1,t2,s4>, <s1,s4,t2> .

Fiecare tranziție reprezintă în acest caz un pas de calcul (analiza-selecție-execuție).

În cazul 2. specificația A reprezintă de asemenea **un singur sistem**. Execuția se deşfăşoară în mod sincron : la un pas de calcul se execută toate tranzițiile posibile (oferite de submodule), într-o ordine aleatoare.

În acest caz setul de secvențe de tranziții admisibile este :
<t1,s1, s3>, <s1,t1,s3> .

```
    |     |         |    |
    |   pas 2     |   pas2
  pas1           pas1
```

În cazul 3. specificația A reprezintă **două sisteme**. Execuția se deşfăşoară în mod independent şi asincron în cele două module: la un pas de calcul se execută toate tranzițiile posibile,în cele două sisteme, într-o ordine aleatoare.

În acest caz setul de secvențe de tranziții admisibile este :
<t1,s2>, <s1,s4,t2> ,<s1,t2, s4>, <t1,s1,s3>, <s1,s4, t1>, <s1,t1,s4>, <s1,t1,s3> .

4. Descrierea mediului de dezvoltare XEDT

Pentru lansarea în execuție a pachetului de programe XEDT sînt necesare urmatoarele comenzi (în sistemul de operare UNIX, pe o maşină locală identificată prin *id_maşina_locală*) :

- **startx** ---> se intră în modul de lucru cu ferestre grafice

- din mediul de lucru grafic se deschide o consolă XTERM; comenzile următoare se vor da din XTERM.

- **xhost** *id_maşina_XEDT* ---> se indică faptul că se vor importa ferestre grafice de pe maşina Unix pe care este instalat pachetul de programe XEDT identificată prin *id_maşina_XEDT*
- **telnet** *id_maşina_XEDT* (conectare la distanţă pe maşina *id_maşina_XEDT*)

Următoarele comenzi se vor da de pe maşina distantă *id_maşina_XEDT :*

- **export DISPLAY =** *id_maşina_locală* : 0.0 ---> se indică destinaţia unde se vor transfera ferestrele grafice
- **cd** *dir_estelle* ---> se intră în subdirectorul de lucru identificat prin *dir_estelle*
- **xedt** ---> se lansează pachetul de programe XEDT

În mediul de dezvoltare XEDT există următoarele opțiuni :

- **Exit** **--->** pentru ieşire din program
- **Tools - Translator --->** transformă specificaţia Estelle într-o formă intermediară
  - **Generator --->** generează cod C, folosind forma intermediară
  - **Compilator --->** translator + generator
  - **Simulator** ---> simulează execuţia specificaţiei Estelle
  - Table Generator (nu se va utiliza)
  - **Viewer** ---> vizualizează specificaţia Estelle

Etapele de lucru în mediul de dezvoltare XEDT vor fi :

1. Se editează specificaţia Estelle (cu extensia *.stl)
2. Se lansează comenzile **Translator** şi **Generator** sau direct **Compilator**
3. Dacă există erori se corectează şi se revine la 2., altfel se sare la 4.
4. Se lansează comanda **Simulator**

În **Simulator** comenzile de lucru sînt :

a) Se selectează fişierul cu extensia *.stl care conţine specificaţia Estelle dorită pentru simulare.

b1) Editare observatori : se selectează, cu mouse-ul comanda de *Edit Observer* după care în cadrul acestei comenzi se editează observatorii. Se revine în simulator cu comanda *Close*.

Exemple de observatori :

```
d $ltri;\       \% afişează  interacţiunea de intrare care a produs ultima tranziţie
                \% "last transition input"
d $ltr0;\       \% afişează  interacţiunea de ieşire  generată la ultima tranziţie
                \% "last transition output"
d $ltrst;\      \% afişează stările iniţiale şi finale pentru ultima tranziţie
                \% "last transition state"
d $tfs;\        \% afişează  numărul total de tranziţii efectuate
                \% "total firing steps"
```

Observatorii activați (cu comenzile *Set* şi *Enable*) se execută în paralel cu specificația Estelle. Observatorii pot fi dezactivați cu comanda *Disable*.

b2) Editare macrodefiniții : se selectează, cu mouse-ul comanda de *Edit Macro* după care în cadrul acestei comenzi se editează macrodefiniția şi se activează cu comanda *Set*. Se revine în simulator cu comanda *Close*.

Exemple de macrodefiniții :

```
$fs:=100;\      \%numărul de tranziții (firing steps) este setat la 100
d $h;\          \%afişează ierarhia de module
n->var:=value;\         \%inițializează variabila var din instanța de modul n cu valoarea value
d $lins;\       \% afişează  ultima instanța de modul ("last instance")
d $trid;\       \% afişează  ultima tranziție efectuată ("last transition identifier")
print"Mesaj";\ \%tipăreşte mesajul Mesaj
```

O macrodefiniție se execută cu comanda *NUME_MACRO()*, unde *NUME_MACRO* este numele macrodefiniției, data la prompt-ul Edb.

c) În simulator se setează (dacă e cazul) parametrii de simulare : *Simulation Step* (implicit 1 tranziție) şi *Simulation Time* (implicit 0 - timp infinit de simulare)

d) Simularea propriu-zisă se efectuează cu comanda *Continue* urmată de *OK*. Se vor afişa tranzițiile efectuate (efectul observatorilor activați). Simularea poate fi reluată din punctul inițial cu comanda *Restart* urmată de *OK*.

Opțional se poate utiliza comanda **Viewer** pentru afişarea specificație Estelle simultan cu deşfăşurarea simulării.

6. Desfăşurarea lucrării

a) Se va studia descrierea generală şi sintaxa limbajului Estelle (secțiunile 1 şi 2).

b) Se vor analiza exemplele de specificații Estelle (din secțiunea 3) utilizînd mediul de dezvoltare XEDT (descris în secțiunea 4). Se vor urmări următoarele aspecte :

- definirea observatorilor (şi a macrodefiniților)
- efectuarea tranzițiilor (pe modul pas cu pas)
- modelarea timpului (instrucțiunea DELAY).
- efectuarea tranzițiilor conform modelului de paralelism specificat