OETPN

(Object Enhanced Time Petri Nets)

Rețele Petri înzestrate de timp cu obiecte

1. Justificare
2. Definiții
3. Proprietăți
4. Exemple de modele OETPN
5. Utilitatea și utilizarea modelelor OETPN
6. Calitatea modelelor OETPN și calitatea programelor

5. Utilitatea și utilizarea modelelor OETPN

Modelele OETPN pot fi folosite în toate etapele de dezvoltarea ale aplcațiilor:

1. Specificare
2. Analiza și verificarea specificațiilor
3. Sinteza modelelor și algoritmilor (destinați domeniului aplicațiilor reactive)
4. Proiectarea software-ului
5. Analiza și verificarea proiectului software
6. Implementarea software-ului
7. Analiza și verificarea implementării
8. Testarea software-ului
9. Integrarea componentelor software cu modele OETPN în aplicații compuse cu componente similare
10. Mentenanța (întreținerea și updatarea aplicațiilor)

Modelele OETPN sunt înzestrate cu diferite trăsături și sunt folosite diferite proprietăți ale lor în funcție de etapa în care sunt utilizate.

În cazul sistemelor mari este necesară structurarea lor în componente care includ modele OETPN.

Modelele OETPN pot fi concepute să fie mai aproape de gândirea umană (adică de reprezentarea oamenilor a entităților sau sistemelor pe care le concep), sau mai aproape de programele caluculatoarelor care le vor implementa.

Modelele OETPN pot fi utilizate la rafinarea și detalierea dezvoltării aplicațiilor.

5.1 Specificarea și verificarea specificațiilor

Specificațiile pot fi:

- formale și
- informale (uzual sub formă textuală).

Specificațiile pot conține informații:

- funcționale și
- ne-funcționale.

Se consideră o aplicație reactivă compusă din:

- Instalație (numită pe scurt Plant) ← Sistem controlat – la care reacționează software-ul
- Mediu (numit pe scurt Environment) și
- Sistem de control (numită pe scurt Controller).

Plant, Controller și Environment pot fi concentrate sau distribuite (incluzând ierarhizate).

Specificațiile textuale ale aplicației conțin:

- Declarații și descrieri privind Plant
  - Structura adică elementele componente ale lui Plant și relațiile dintre ele
  - Comportamentul elementelor
  - Comportamentul lui Plant
  - Canalele de interacțiune cu Plant (intrări și ieșiri)
    - Evenimente
    - Fluxuri de date
    - Active
    - Pasive
- Declarații și descrieri privind mediul în care Plant este inclusă și interacțiunea instalației cu mediul
- Declarații și descrieri ale comportamentului operatorilor umani (dacă există) care pot interacționa cu cu Plant
  - Comportament
  - Mod de interacțiune (continuă, sporadică etc.)
- Declarațiile și descrieri privind cerințele de comportament ale instalației
  - Performanțe
  - Informații nonparametrice

Specificarea continuă cu realizarea (sinteza) modelelor OETPN ale lui Plant, Environment și operatori.

Scopurile creării acestora sunt:

- Obținerea unei descrieri formale (exacte, precise și clare) care trebuie să fie
  - Consistentă
  - Completă
- Posibilitatea verificării specificațiilor
  - Verificarea structurii
  - Verificarea comportamentelor elementelor și comportamentului global al sistemului
  - Verificarea interacțiunilor dintre Plant, Controller, Environment și operatori
  - Verificarea structurii informațiilor schimbate între elementele componente
  - Verificarea logicii modelelor
  - Verificarea compatibilității dintre relațiile de evoluție și conținuturile jetoanelor
  - Verificarea disponibilității informațiilor la momentele de timp când sunt necesare
- Posibilitatea simulării modelelor OETPN obținute pentru
  - înțelegerea mai bună a sistemului
  - utilizarea lor la sinteza Controllerului și verificarea comportamentului sistemului rezultat

5.2 Proiectarea

Aplicația este parțiționată în:

- Plant,
- Controller
- Environment.

Plant și Environment au fost determinate în faza de specificație.

A rămas determinarea unui Controller care satisface cerințele precizate anterior.

Proiectarea începe cu realizarea unei arhitecturi (concentrate sau distribuite) care permite conectarea Controllerului la Plant și Environment.

Determinarea canalellor de intrare/ieșire împreună cu tipurile informațiilor pe care le transferă dintr-o parte în alta.

Dacă aplicația este mare și complexă ➔ partiționarea Controllerului

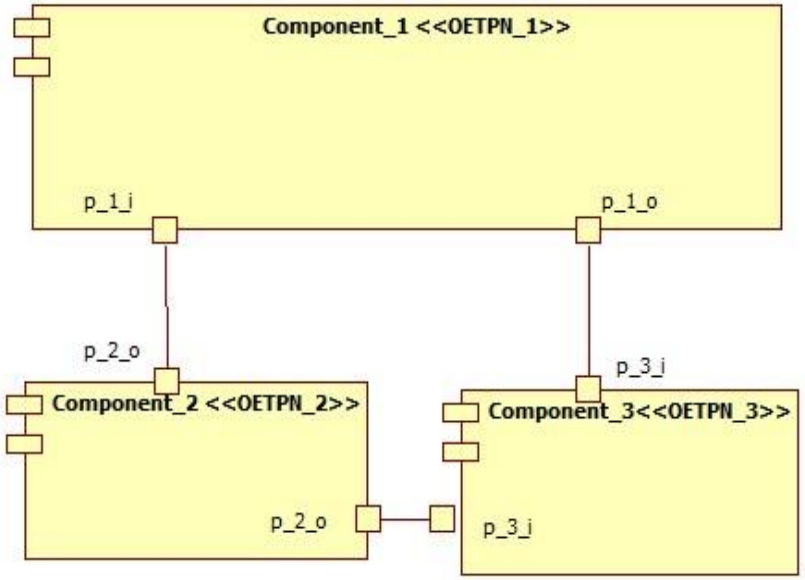
Proiectarea software-ului:

- alegerea mediului de implementare conținând un sisteme de operarea sau executive și pachete cu clase sau proceduri care susțin implementarea
- stabilirea interfețelor cu mediul (Plant si Environment) în care va fi inclusă aplicația și determinarea driverilor care realizează operațiile intrare/ieșire
- partiționarea Controllerului în componente care vor implementa modele OETPN sau algoritmi
- stabilirea claselor care pot servi la instanțierea obiectelor care vor stoca informațiile (adică structura jetoanelor și implicit tipurile locațiilor)
- stabilirea interacțiunilor dintre componente adică a canalelor intrare/ieșire
- stabilirea cerințelor care vor fi realizate de fiecare componentă.

Luând în considerare capabilitățile modelelor OETPN, proiectarea poate fi realizată cu componente care interacționează și care pot include la rândul lor alte componente. De aici apare posibilitate de structurare pe orizontală sau pe verticală (adică ierarhizată) a aplicației și posibilitate de dezvoltare a proiectului prin detalieri și rafinări succsive.

Componente cu modele OETPN

Fig. 25. Diagramă de componente



Ca toate rețelele Petri, modelele OETPN crează neplăceri din cauza dimensiunii atunci când trebuie să descrie aplicații complexe. Evitarea acestui inconvenient se bazează pe partiționarea aplicației în componente care includ modele OETPN.

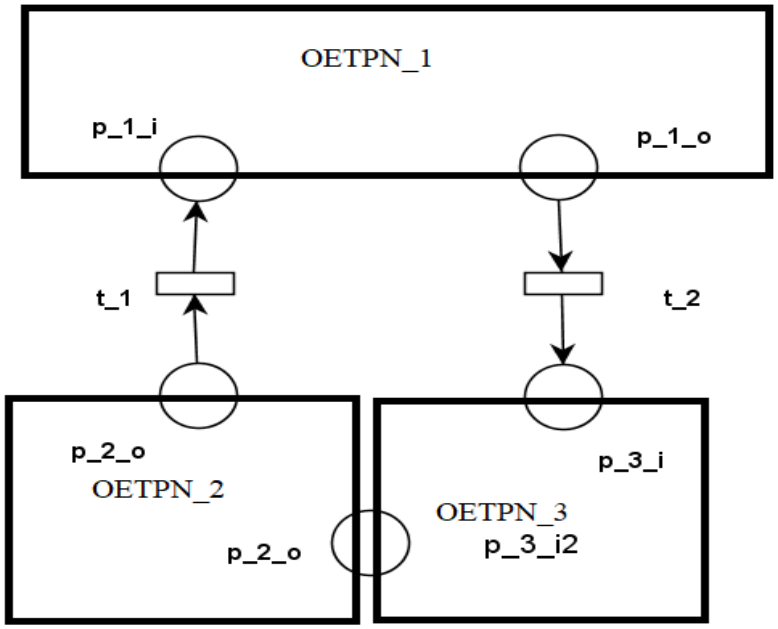
Diagrama alăturată conține 3 componente care includ fiecare câte un model OETPN și care interacționează prin intermediul porturilor notate cu p_1_i, p_1_o, p_2_o, p_2_o, p_3_i și p_3_i..

Diagrama din Fig. 26 reprezintă aceeași aplicație unde au fost reprezentate cu dreptunghiuri rețelele OETPN și s-au reprezentat pe frontiera lor locațiile de intrare și iessire. Trebuie precizat că identificatoarele porturilor sunt și identificatoarele locațiilor (modelelor OETPN) reprezentând canale de intrare și respectiv de ieșire. Locațiile de pe frontiera unui model compun interfața lui cu alte modele, iar în cazul de față coincid cu porturile componentei.

Fig. 26. Exemplu de concepere distribuită

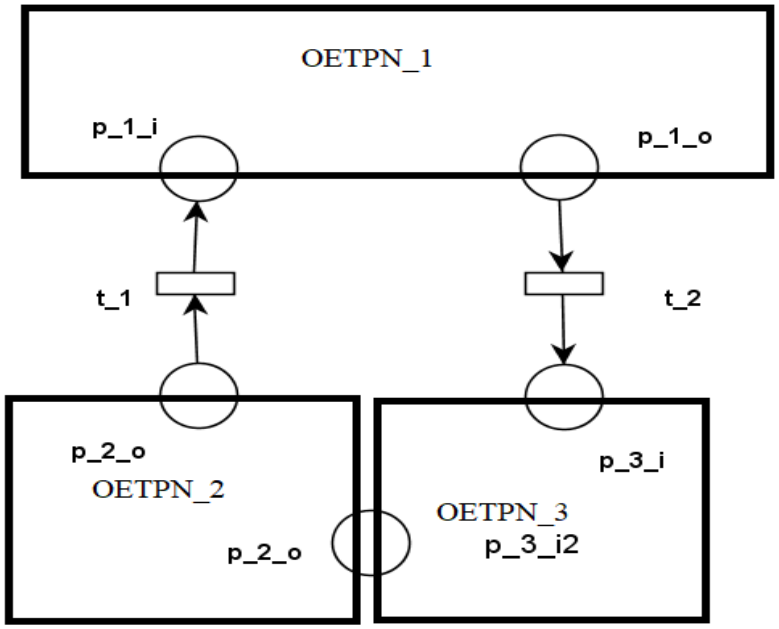
Realizarea comunicației între modele implementate în componente se poate realiza în următoarele moduri:

- prin locații comune dacă ambele OETPN-uri sunt executate de fire diferite, dar care aparțin aceluiași proces;
- prin intermediul mediului (de exemplu, sistemul de operare) care implementează canale de comunicație (de exemplu, conducte sau transmisii cu protocoalele TCP/IP, UDP etc.).



În acest caz, OETPN-urile sunt implementate în procese diferite și pot fi chiar în calculatoare diferite.

Modelele OETPN_2 și OETPN_3 au o locație comună folosită ca ieșire (p_2_o) de primul și ca intrare (p_3_i2) de cel de-al doilea.



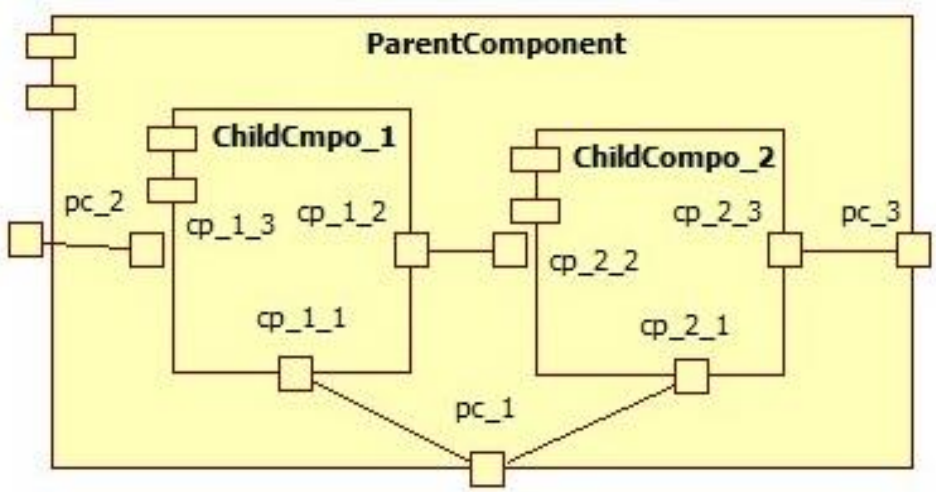
Modelele OETPN_1 și OETPN_2 comunică prin intermediul canalelor realizate de mediul de comunicație.
Logic, aceste canale pot fi considerate ca fiind implementate de o componentă care are locații comune cu componentele pe care le servește.

Un astfel de canal poate realiza o simplă transmisie (precum UDP) sau poate repeta comunicarea până se confirmă transmisia lui corectă. În acest ultim caz modelul de transmisie este echivalent cu două obiecte active care comunică printr-o locație comună.

Componente compuse

Se admite ca o componentă să includă la rândul ei alte componente.

Fig. 27. Componente compuse

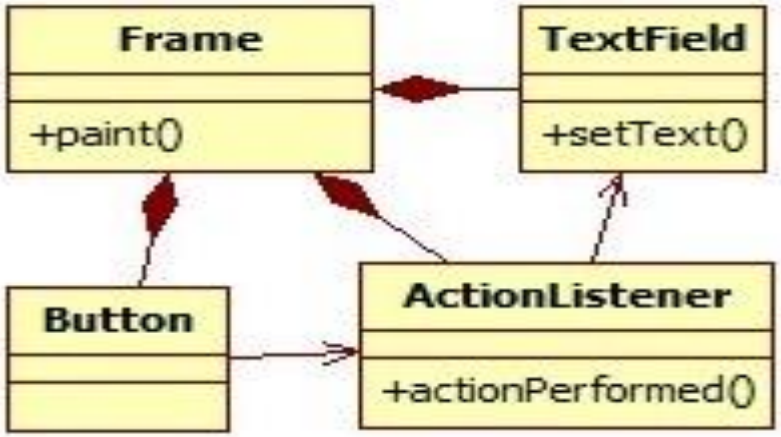


Comunicația dintre componenta (numită aici părinte) care include alte componente, numite copii, se poate realiza prin porturi care corespund la canale (locații comune) intrare/ieșire conectate (vezi in figura alăturată cp_1_1-pc_1-cp_2_1).

Componenta părinte este responsabilă de realizarea interfeței unui copil cu exteriorul componentei părinte (vezi pc_2-cp_1_3; cp_2_1-pc_3).

Modele OETPN și interfețe (componente) grafice

Fig. 28. Interfață grafică



O interfață grafică (precum cea din figura alăturată) este un obiect activ care are o imagine proiectată pe ecran împreună cu componente atomice (butoane, câmpuri de text, etichete etc.).

Unele componente (complexe, de exemplu *Frame*) pot integra alte componente precum butoane care pot genera (la intervenția utiizatorului) evenimente de intrare (externe calculatorului, vezi ActionListener) la care reacționează un fir *awtThread* (application window toolkit).
Acestea din urmă pot depune informații în elemente pentru stocarea informațiilor (de exemplu, liste, tablouri etc.).

Alte componente grafice (precum, *TextField*, *TextArea* etc.) pot afișa informații generate de aplicație. Acestea reprezintă canale de ieșire pentru modelele OETPN. Alte componente grafice mai complexe (precum Frame, Pannel) au metode care pot desena (*draw(), paint(), repaint()* etc.) pe suprafața vizibilă a componentei.

Analiza și verificarea proiectului

Analiza și verificarea proiectului software constă din:

- verificarea introducerii în proiect a tuturor cerințelor și specificațiilor
- verificarea respectării tuturor cerințelor (din specificații)
- verificarea corespondenței dintre structurile de obiecte (jetoanele din locații) și expresiile (grd și map, deci relațiile de evoluție) care vor accesa obiectele (adică jetoanele)
- verificarea relației dintre componente și algoritmi (modele OETPN în cazul de față)
- verificarea relației dintre componente
- verificarea comportamentului general al aplicației

5.3. Sinteza modelelor OETPN sau a algoritmilor

Modelele OETPN pot descrie algoritmii necesari aplicațiilor reactive. Dacă aplicația a fost partiționată în componente, urmează să se sintetizeze ce va realiza fiecare componentă. Sinteza (crearea) algoritmilor pe baza modelelor OETPN are în vedere determinarea:

- structurii modelului OETPN
- structurilor informațiilor stocate și procesate de OETPN (adică de algoritm)
- funcțiilor și condițiilor când acestea sunt activate
- interfețelor modelului cu mediul în care este înglobat și cu alte entități din sistem

Un alt aspect se referă la intervalele de timp când activitățile urmează să fie executate.

Condițiile *grd* și mappingurile *map* pot conține expresii matematice care pot fi parametrizate.

Ele pot accesa numai informații din jetoanele conținute în acel moment în locațiile de intrare și pot stoca informații numai în jetoanele din locațiile de ieșire ale tranziției.

Sinteza algoritmilor pe baza modelelor OETPN constă din determinarea tuturor elementelor conținute (declarate) în definiția OETPN:

- structură,
- condițiile de gardă,
- mappingurile și
- intervalele de timp pentru activarea tranzițiilor.

Sinteza algoritmilor urmărește îndeplinirea unor obiective clar specificate. Atingerea obiectivelor și evaluarea calităților algoritmilor (sau modelelor OETPN) se face pe baza unor criterii de evaluare privind:

- structura
- comportamentul

Metodele de evaluare pot să fie funcții simple sau algoritmi.

Metodele de sinteză pot fi:

- analitice, implicând oameni care adaugă sau elimină informații, proprietăți, atribute sau operații
- automate realizate de programe care se bazează pe simulări și algoritmi de îmbunătățire:
  - euristici
  - determiniști
- interactive implicând cooperarea dintre oameni și aplicații software care folosesc simulări și metode de evaluare
  - numerice
  - numerice și grafice

Sinteza unui controller pentru aplicații reactive

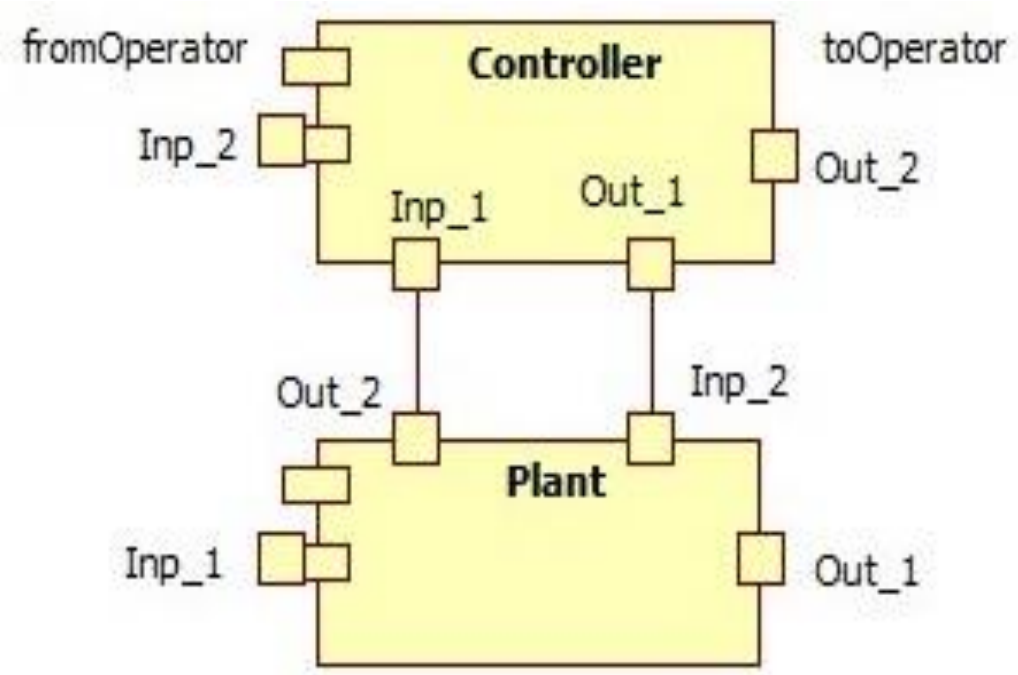
Problema sintezei

Din specificații se construiește modelul $OETPN_P$ al instalației (engl.: plant) termen folosit aici ca denumire generică.

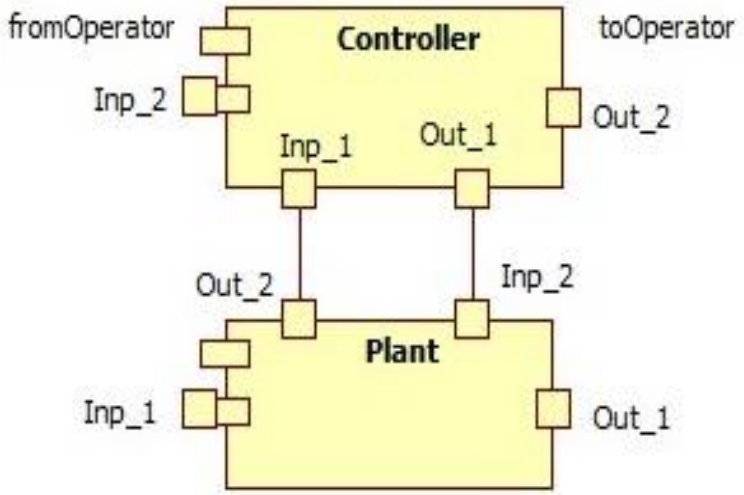
Plant poate fi: un vehicul, o unealtă, un dispozitiv, sau chiar o altă aplicație software.

Modelul $OETPN_P$ descrie comportamentul lui Plant aflat într-o stare inițială specificată și interacțiunile cu exteriorul instalației.

Fig. 29. Diagrama de componente a unui sistem reactiv

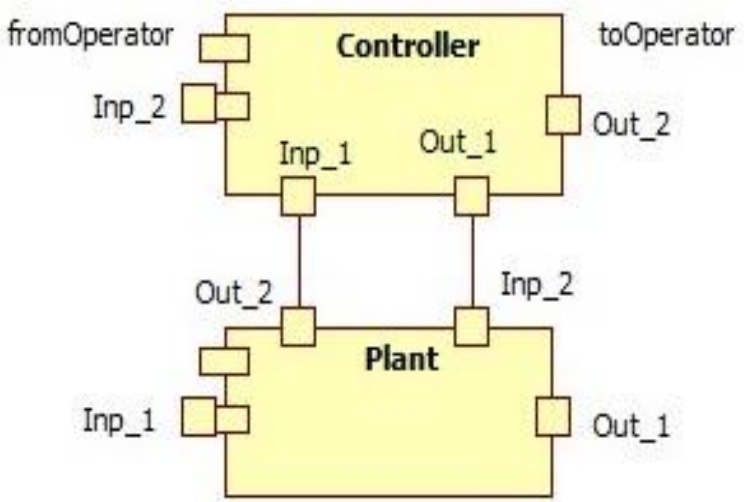


Se precizează pentru Plant:



- canalele de intrare Inp_1 care crează evenimente sau fluxuri de date necontrolate și necunoscute (de către Controller) în general; tipurile acestor informații trebuie să fie compatibile cu locațiile instalației; uneori operatori umani pot intra în interacțiune cu instalația prin unele dintre canale de intrare din setul Inp_1;
- Canalele de ieșire Out_1 care generează evenimente și fluxuri de date în exteriorul instalației, dar care nu sunt (în general) măsurate direct (sau cel puțin, nu toate) de către Controller; uneori operatori umani pot primi informații despre instalație prin intermediul unor canale din setul Out_1;
- Canalele de ieșire Out_2 furnizează evenimente si fluxuri de date care pot fi preluate de către Controller; tipurile acestora trebuie să fie compatibile cu tipurile locațiilor corespunzătoare din Controller;
- Canalele de intrare Inp_2 sunt furnizoare de informații (controlabile) către instalație; trebuie precizate dacă sunt de tip evenimente sau fluxuri de date și se specifică tipurile lor;

Prin *controller* se înțelege aici un program care interacționează cu Plant și îl determină pe acesta din urmă să aibă o evoluție cerută prin cerințele de specificare.

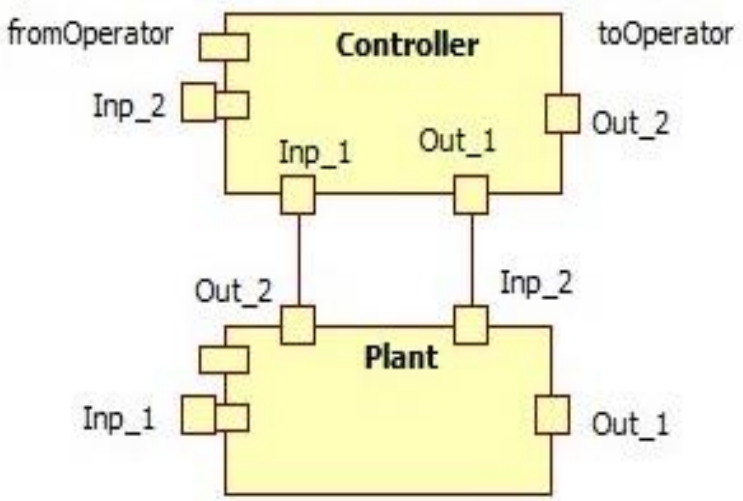


Cerințele de specificare (engl.: specification requirements) descriu cum trebuie să se comporte Plant.

Este posibil ca un operator uman să interacționeze cu Controller sau chiar și cu Plant în timpul evoluției.

Cerințele pot include texte furnizând informații într-o formă informală.

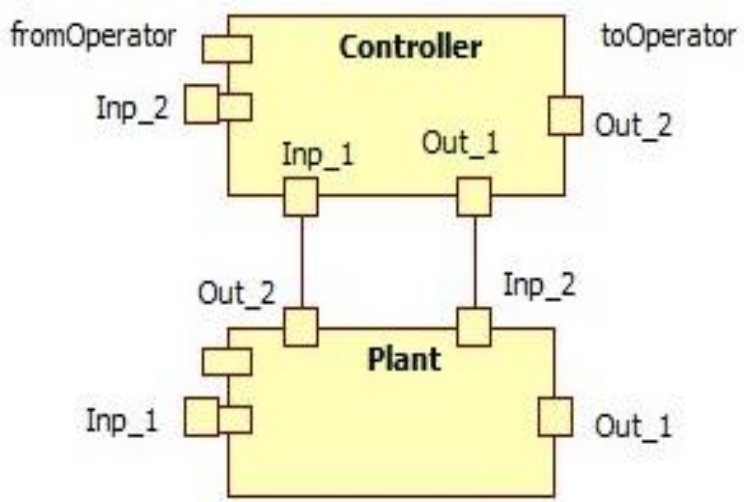
Din cerințele de specificare se construiește modelul $OETPN_R$ (sau un set de modele) care descrie formal modul de comportare cerut al lui Plant.



Exemple de cerințe:

- Dacă Plant se află într-o stare precizată și dacă intrările primesc informații (evenimente sau fluxuride date) specificate, atunci Plant trebuie să evolueze pe o anumită traiectorie cerută de stări, sau să evite o stare, eventual o secvență de stări;
- Dacă Plant se află într-o anumită stare și se produce un eveniment dat, Plant trebuie să recționeze conform unor cerințe date, cum ar fi să producă un eveniment sau o secvență de evenimente;

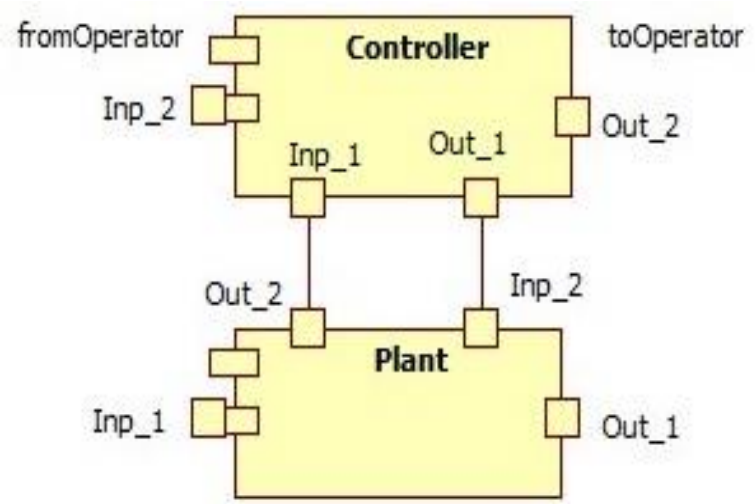
Specificațiile controllerului conțin:



- Canalele de intrare Inp_1 prin care se pot primi informații de la Plant; ele trebuie să fie compatibile cu canalele de ieșire din Plant; ele pot fi de tip eveniment sau fluxuri de date;
- Canalele de ieșire Out_1 prin care Controller poate influența comportamentul lui Plant; acestea furnizează informații de tip eveniment sau fluxuri de date; ele trebuie să fie compatibile cu canalele Inp_2 din Plant;
- Canalele Inp_2 servesc pentru preluarea unor comenzi de la operator sub forma de evenimente sau fluxuri de date;
- Canalele Out_2 furnizează informații operatorului sub formă de evenimente sau fluxuri de date; sunt necesare dispozitive capabile pentru preluarea sau afișarea tipului de informații furnizat.

Problema formală a sintezei controllerului:

- determinarea modelului controlerului $OETPN_{C`}$
- determinarea stării inițiale a lui $OETPN_C$ pentru cazul în care $OETPN_P$ se află în starea M_P^0, iar comportamentul lui Plant corespunde cerințelor $OETPN_R$ cu M_R^0.

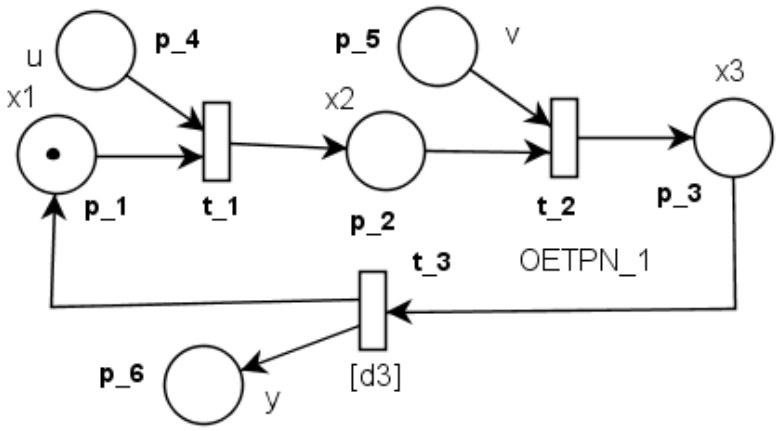


Pentru unele probleme se specifică intrările din exterior pentru Plant și pentru Controller.

Primele se numesc uneori perturbații, iar cele din urmă exprimă deseori dorințele sau comenzile operatorului.

Exemplu: Sinteza unui model OETPN pentru control

Fig. 30. Plant model.



Declarații

Se dă modelul OETPN_1 reprezentat în Fig. 30. cu specificațiile:

type(m(p_i)) = {x: float}; i = 1, 2, ..., 6

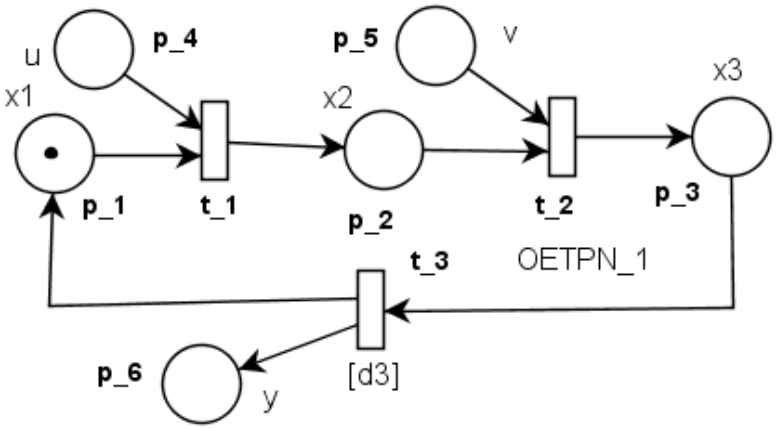
d3 este o întârziere fixă care determină perioada de modificare a stării.

Starea este reprezentată de [x1, x2, x3, u, v, y].

Setul canalelor de intrare este: Inp={p_4, p_5}

Setul canalelor de ieșire este: Out={p_6}

Se notează:
xi = m(p_i).x variabila x stocată în jetonul din locația p_i; i = 1, 2, 3.
u = m(p_4).x; v = m(p_5).x; y = m(p_6).x;
Starea inițială: x1=10



Locațiile p_4 și p_5 sunt canale de intrare, cu *u* o variabilă de control, iar *v* o perturbație aleatoare.
Constrângeri: $-1 \le v \le 1$; $-2 \le u \le 2$;

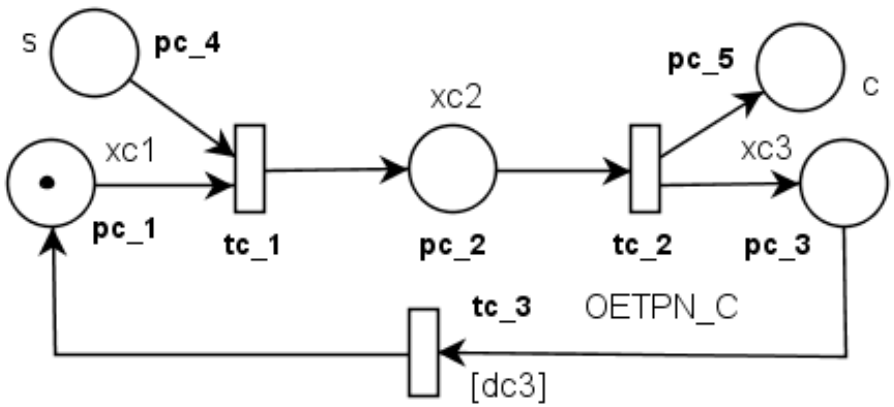
Relațiile de evoluție sunt:

- t_1:
  - grd^1_1:= ((m(p_1) ≠ φ) and (m(p_4) ≠ φ)) ➔ map^1_1:= x2=x1 + u;
  - grd^2_1:= (m(p_1) ≠ φ) and (m(p_4) = φ)) ➔ map^2_1:= x2=x1;
- t_2:
  - grd^1_2:= ((m(p_2) ≠ φ) and (m(p_5) ≠ φ)) ➔ map^1_2:= x3=x2 + v;
  - grd^2_2:= (m(p_2) ≠ φ) and (m(p_5) = φ)) ➔ map^2_2:= x3=x2 – 0.1;
- t_3: grd_3:= m(p_3) ≠ φ ➔ map_3:= (y=x3; x1=x3);

Fig. 31. Controller model.

Cerințe

Se cere să se determine (sintetizeze) un controller pentru OETPN_1 astfel încât să se mențină valoarea lui *y* cât mai aproape de 5.0: ||5.0 -y|| = 0. (de fapt ➔ 0)

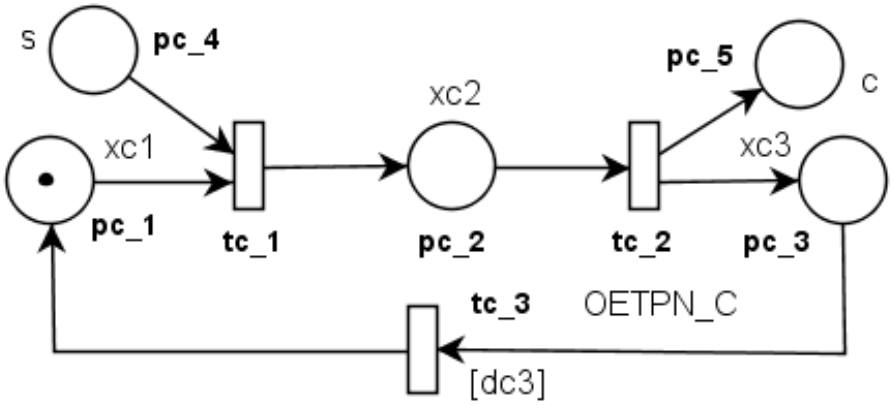


Se mai pot da ca cerințe:

- abaterea maximă a lui *y* față de valoarea 5.0 în contextul în care
  - v este un eveniment asincron având valoarea amplitudinii maximă sau minimă admisă
  - v este un flux de date având un anumit profil specificat prin frecvență, amplitudine (eventual variația ei), offset
- durata de corecție (răspuns) a unei abateri de tip eveniment asincron sau variații de tip treaptă în fluxul de date

Soluție

Soluția propusă (OETPN_C) este reprezentată în Fig. 27.

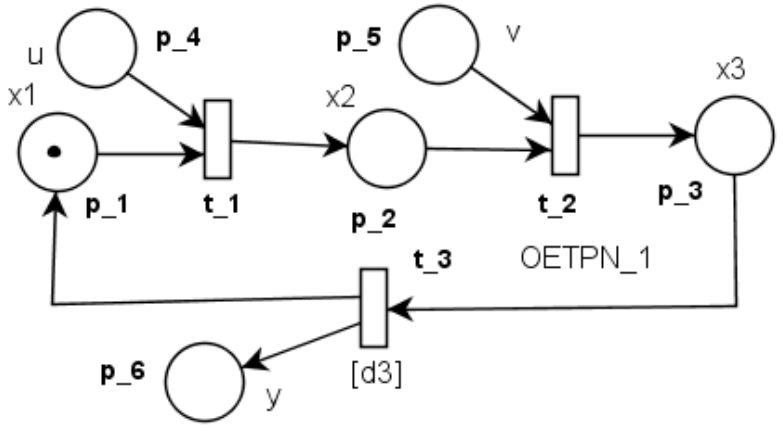


Canalul de intrare pc_4 trebuie conectat cu p_6, iar cel de ieșire pc_5 cu p_4.

Se alege:

type(pc_1) = type(pc_2) = type(pc_3) = type(pc_4) = type(pc_5) = {x: float}

S-au notat: xc1=m(pc_1).x; xc2=m(pc_2).x; xc3=m(pc_3).x; s=m(pc_4).x; c=m(pc_5).x;

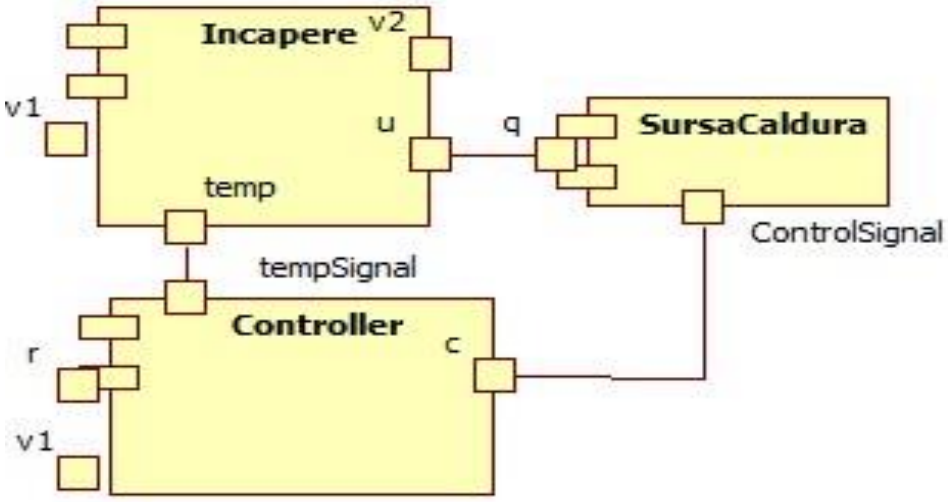


Se propun:

- tc_1:
  - grd^1_1:= ((m(pc_1) ≠ φ) and (m(pc_4) ≠ φ)) ➔ map^1_1:= xc2= u;
  - grd^2_1:= (m(pc_1) ≠ φ) and (m(pc_4) = φ)) ➔ map^2_1:= xc2=xc1;
- tc_2: grd_2:= (m(pc_2) ≠ φ) ➔ map^1_2:= (xc3=xc2; c= sc2 – 5.0;)
- tc_3: grd_3:= m(pc_3) ≠ φ ➔ map_3:= xc1=xc3;

Controlul temperaturii unei încăperi

Se consideră cazul unei încăperi conectată la o sursă de căldură în care trebuie să se controleze temperatura conform referinței *r* date de un operator.



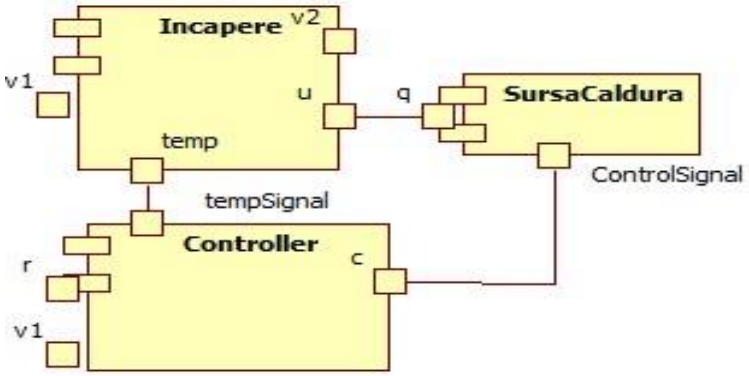
Temperatura din încăpere este perturbată de răcirea determinată de temperatura variabilă *v1* a mediul înconjurător și de căldura *v2* variabilă introdusă de radiațiile solare.

Specificații:

Temperatura *x* din încăpere variază conform ecuației:

$$x(t+1)= x(t) – a((v1 –x), t) +b\cdot v2(t) + u(t)$$

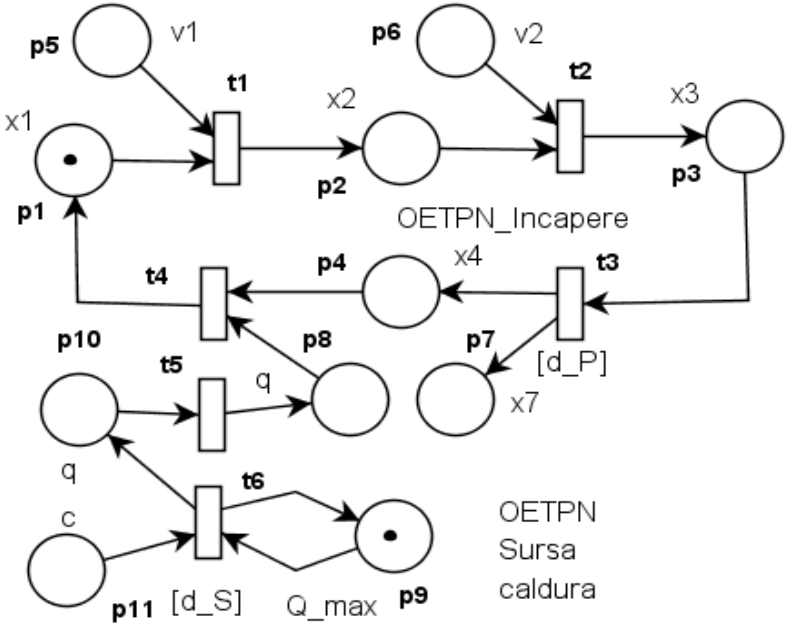
cu $u(t) = e\cdot q(t)$ *și* $a((v1 –x), t)$ o funcție de răcire dependentă de diferența dintre temperaturile din încăpere și mediu. *e* este coeficientul de transformare (conversie) a căldurii (energiei) în temperatură.



Sursa de căldură generează cantitatea de căldură:

$$q(t) = c\cdot Q\_max$$

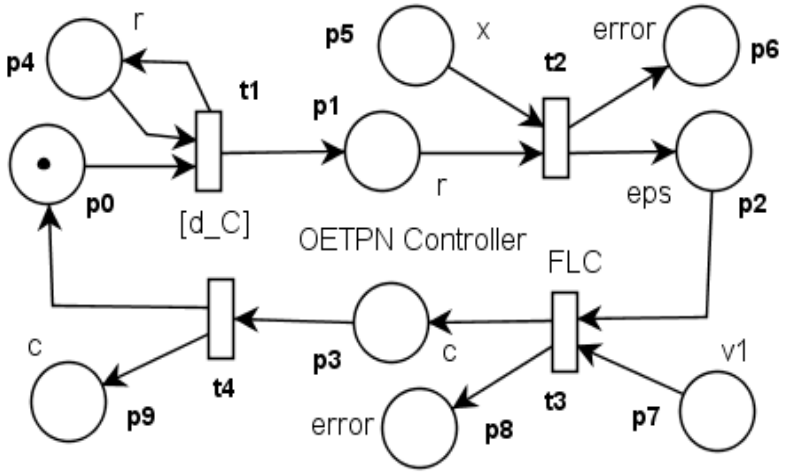
cu Q_max energia (căldura) maximă pe care o poate furniza sursa, iar *c* comanda referitor la cantitatea debitată la momentul *t* pe durata unui tact.



Specificații controller

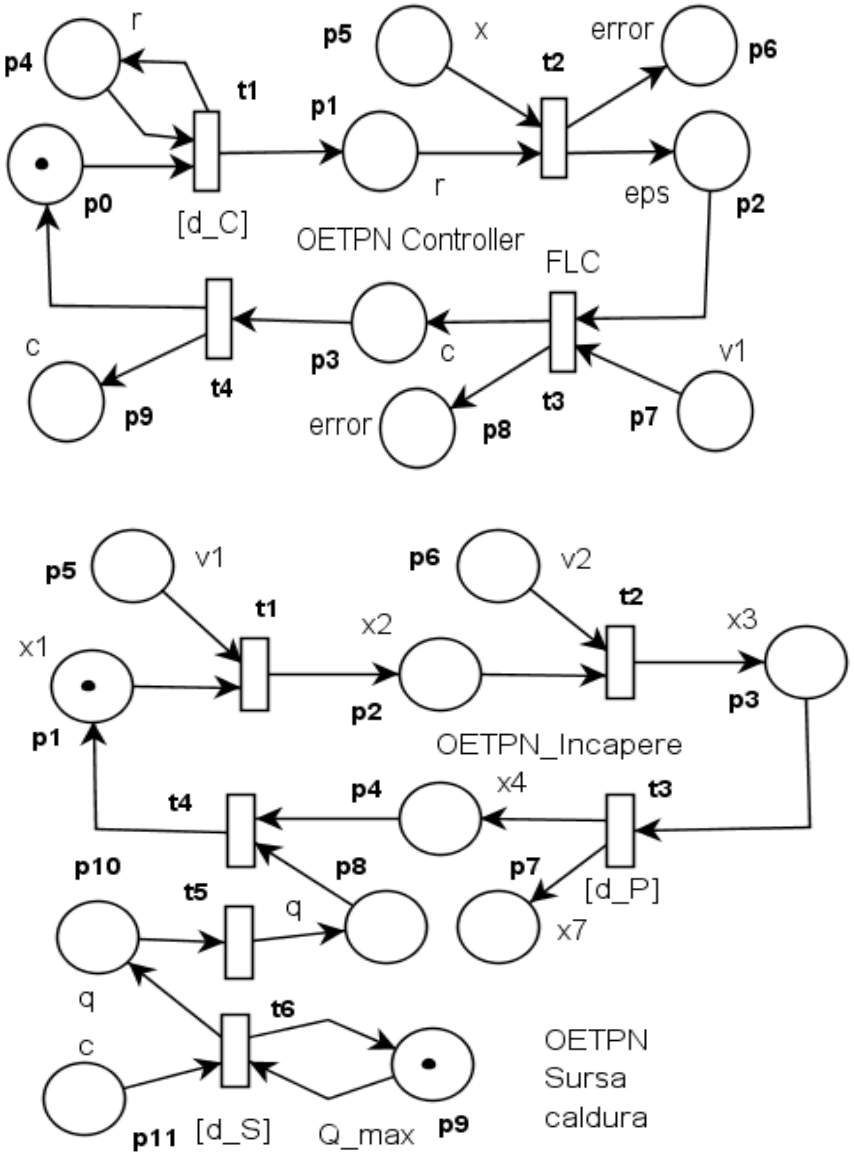
Controllerul primește referința *r* și valoarea temperaturii exterioare *v1*.
El calculează valoarea comenzii *c* pe care o transmite sursei de căldură. Până la modificarea referinței sistemul lucrează cu valoarea anterioară.
Temperatura *v1* poate varia între -20 și +40 ºC.
Perturbația nemăsurată *v2* poate influența temperatura între +10 și +20 ºC.

Cerințe de performanță:
Să se controleze temperatura conform referinței cu o abatere maximă de -1, +1 ºC.
Observații:
Performanța controlului temperaturii poate fi influențată de gradienții variațiilor perturbațiilor *v1* și *v2*.
Dacă poate fi situația pentru v1 astfel încât răcirea este mai mare decât capacitatea maximă de încălzire
$a((v1 - x), t) > e \cdot Q\_max$
atunci sistemul nu poate îndeplini cerința de performanță.



Cerințe pentru situații anormale:
Dacă nu primește controllerul valoarea perturbației *v1*, sistemul lucrează mai departe, dar semnalează anomalia.

Dacă nu primește controllerul valoarea *x* a temperaturii interioare, acesta semnalează anomalia și nu mai introduce căldură în încăpere până la remedierea situației.

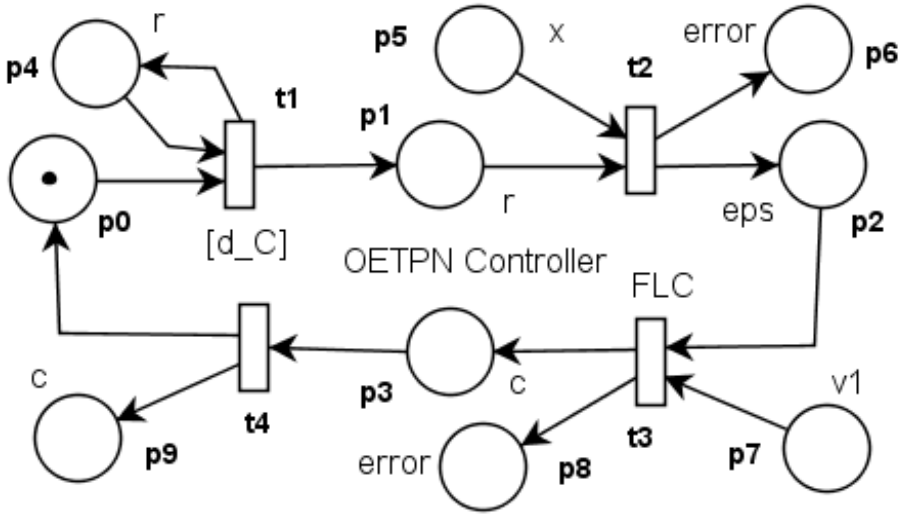


Sinteza controllerului

Se propune modelul din figură.

Variabila *eps* este determinată de $eps = r - x$.

Relația este implementată în mappingul tranziției t2.



Comanda c este calculată de un controller cu logică fuzzy (FLC) dat în tabela alăturată. Această relație este implementată în mappingul tranziției t3.

$eps \wedge v_1$	**NL**	**NM**	**ZR**	**PM**	**PL**	Φ
NL	*PL, Φ*	*PL, Φ*	*PL,Φ*	*PL, Φ*	*PM, Φ*	*Φ, error*
NM	*PL, Φ*	*PL, Φ*	*PM,Φ*	*PM,Φ*	*PM, Φ*	*Φ, error*
ZR	*ZR, Φ*	*ZR, Φ*	*ZR, Φ*	*ZR, Φ*	*ZR, Φ*	*ZR, error*
PM	*NM, Φ*	*NM,Φ*	*NM,Φ*	*NM,Φ*	*ZR, Φ*	*Φ, error*
PL	*NM, Φ*	*NM,Φ*	*NM,Φ*	*NM, Φ*	*NL, Φ*	*Φ, error*

Exemplu: Sinteza controlului traficului vehiculelor printr-o intersecție

Structura rețelei de drumuri și structura sistemului de control (căutat).

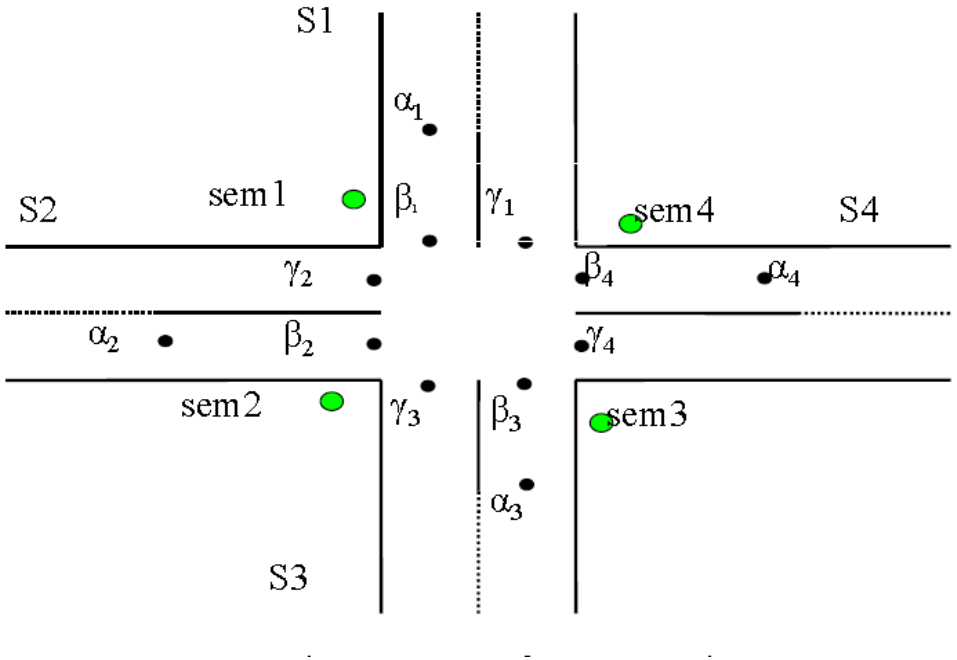


Fig. 32. Structura intersecției

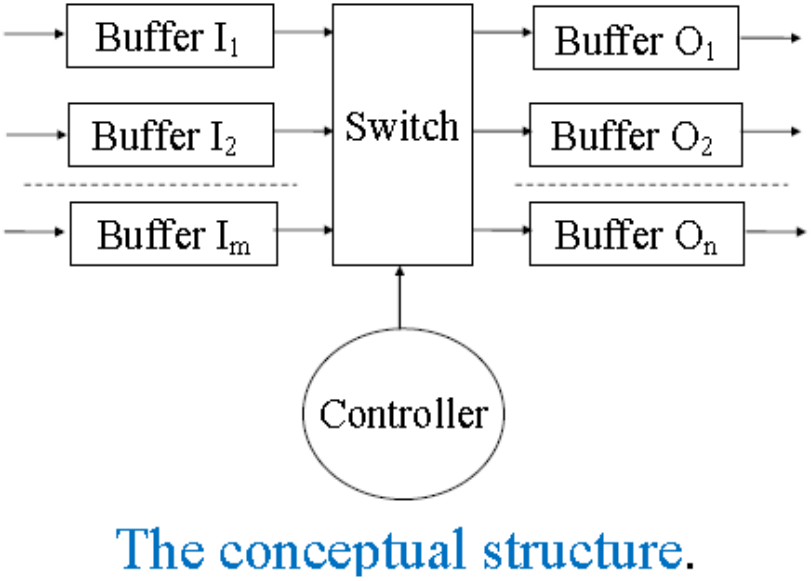


Fig. 33. Structura conceptuală

Infrastructura se compune din 4 benzi de intrare (engl.: input lanes, IL_i; i=1,2,3,4)), 4 benzi de ieșire (engl.: output lanes, OL_i; i=1,2,3,4), 4 semafoare (engl.: traffic lights, TL_i, i=1,2,3,4) și 4 perechi de senzori α_i, β_i care pot măsura intrările și respectiv ieșirile din benzile de intrare corespunzătoare. Senzorii γ_i (i=1,2,3,4) pot măsura intrările în benzile de ieșire.

TL_i are 3 stări: roșu (red, r), galben (yellow, y) și verde (green, g).

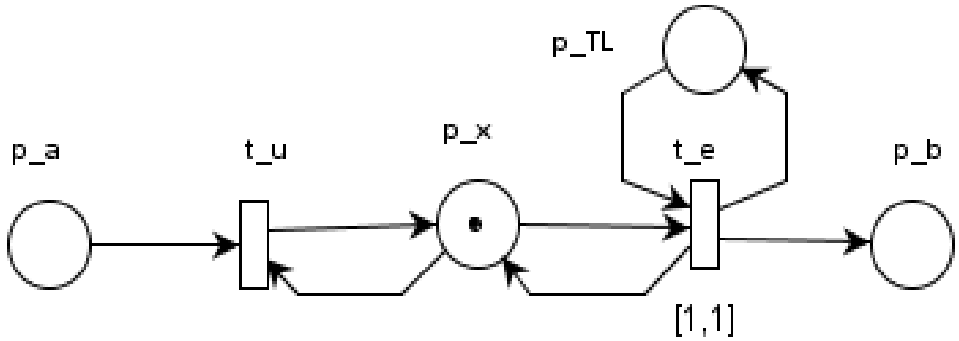
Mașinile pot trece numai când semaforul indică culoarea g.

Durata culorii y este de 5 secunde (sec.). Regula de trecere prin intersecție este ’numai înainte sau la dreapta’.

Fig. 30. Modelul unei benzi.

Modelul lui Plant bazat pe evenimente

Modelul unei benzi este:



$x_c = x_a + u - e$

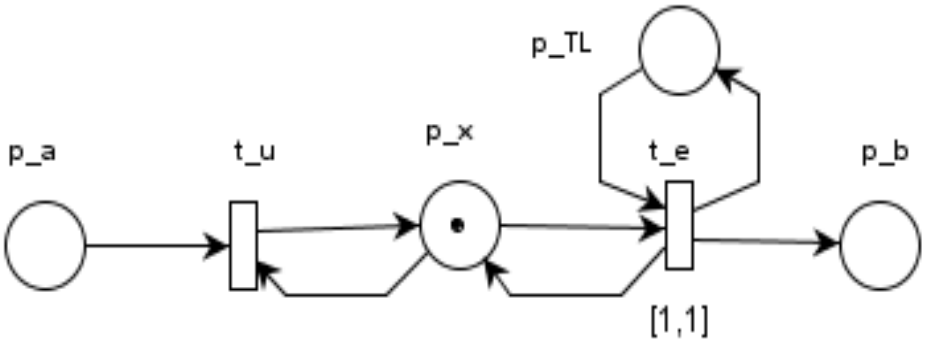
unde x_c și x_a reprezintă numărul de mașini de pe bandă curent respectiv anterior, iar u și e sunt evenimentele când se semnalează intrarea/ieșirea unei mașini. *u* și *e* au valoarea unu, doar când senzorii respectivi semnalează trecerea mașinii, iar în rest au valoarea zero.

S-au notat cu:

- p_a canalul de intrare de la senzorul de intrare α_i,
- p_b canalul de ieșire de la senzorul de ieșire β_i,
- t_u tranziția care preia evenimentul de intrare și incrementează numărul de mașini x de pe bandă (stocat în p_x
- p_TL canalul de intrare prin care se primesc comenzi de la TL (r, y, g)
- t_e tranziția care modelează (și determină) ieșirea unei mașini de pe bandă

Tipurile locațiilor sunt:

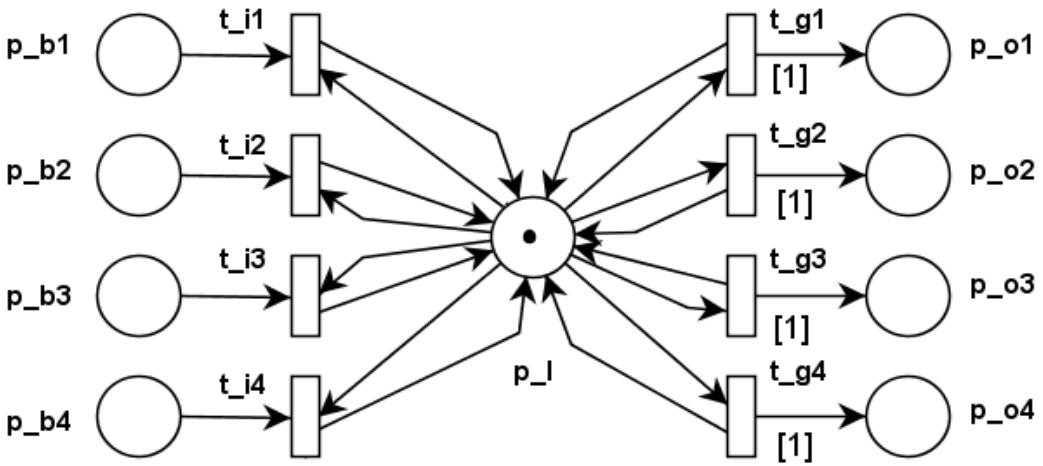
- type(p_a) = InputEvent
- type(p_x) = integer
- type(p_TL) = char = {r, y, g}
- type(p_b) = OutputEvent = {forward, right; f, r}



Regulile de evoluție asociate tranzițiilor sunt:

- t_u: grd:= m(p_a) ≠ φ; map_u:= x=x+1
- t_e: grd:= m(p_TL) = g; map_e: m(p_b) = event_f sau event_r; random();

Fig. 31. Modelul intersecției necongestionabile

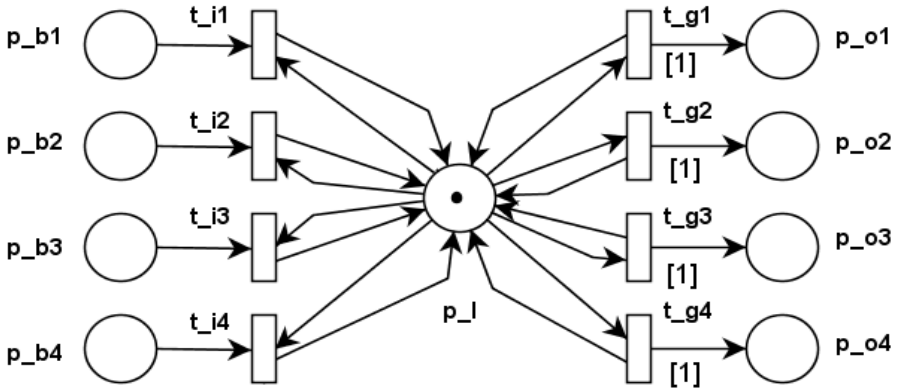


Modelul intersecției:
Traiectoriile benzilor deschise într-o fază nu se pot intersecta.
Sistemul are două faze notate cu: phase_1 (pentru IL_1 și IL_3) și phase_2 (pentru IL_2 și IL_4).
Modelul alăturat descrie comportamentul intersecției. Semnificațiile nodurilor:

- locațiile de ieșire din benzile de intrare în intersecție p_b1, p_b2, p_b3, p_b4
- tranzițiile de intrare în intersecție t_i1, t_i2, t_i3, t_i4
- p_I modelează mașini aflate în intersecție în momentul curent
- tranzițiile t_g1, t_g2, t_g3, t_g4 întârziate cu 1 sec. Modelează evenimentele de intrare în benzile de ieșire ale intersecției
- locațiile p_o1, p_o2, p_o3, p_o4 modelează mașinile care ies în benzile de ieșire

Tipurile locațiilor sunt:

- type(p_b1) = type(p_b2) = type(p_b3)=type(p_b4)= Event(f, r);
- type(p_o1) = type(p_o2) = type(p_o3) = type(p_o4) = Event
- type(p_I) = ListVector cu o listă pentru fiecare bandă de ieșire.



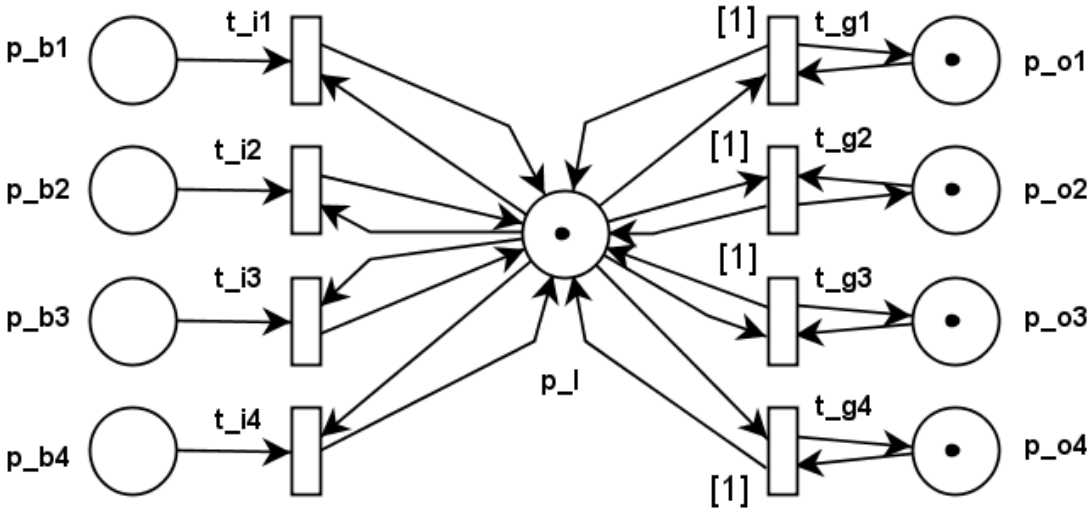
Regulile de evoluție asociate tranzițiilor sunt:

- t_i1, ..., t_i4: (grd_b1: m(p_b1) ≠ φ; map_b1: I.x= I.x +1)
- t_g1, ..., t_g4: (grd_g1: m(p_I) ≠ φ; map_o1: I.x= I.x +1, m(p_o1).v= 1)

Fig. 32. Modelul intersecției congestionabile

Modelul intersecției cu simularea congestionării

Modelul anterior nu ține de cont de disponibilitatea locurilor pentru ca mașinile să treacă în benzilor de ieșire.



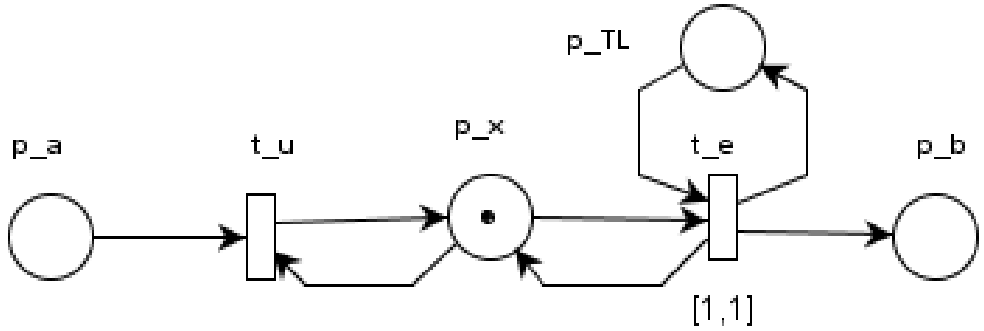
Dacă banda de ieșire este ocupată complet, pot ieși mașini din intersecție pe direcția respectivă doar după ce au apărut locuri libere.

Modelul alăturat care are în plus arcele (t_g1-p_o1) etc. poate realiza așa ceva.

Condițiile de gardă ale tranzițiilor t_g1 etc. pot modela și această nouă resptricție.

Modelul lui *Plant* bazat pe fluxuri de date

Spre deosebire de modelul precedent care ia în considerare evenimentele produse de mașini, acest model folosește fluxuri de date. Canalele de intrare furnizează fluxuri de date.



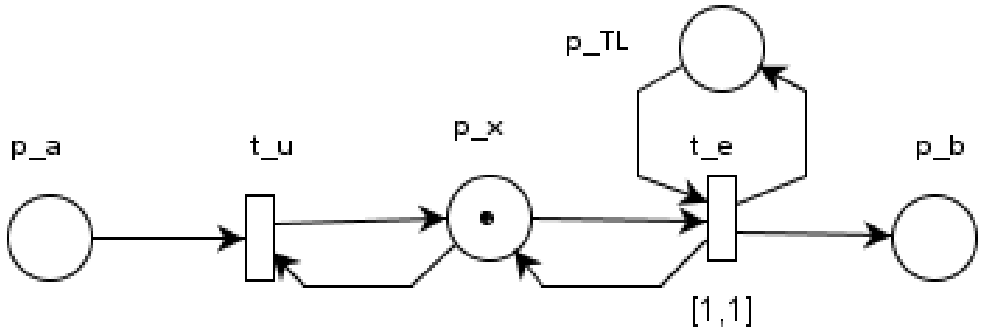
Procesul real al sistemului traficlui de mașini este privit în acest caz ca fluxuri de vehicule (engl.: vehicle flow) care sunt modelate ca fluxuri de date (engl.: data stream). În procesul real este controlat fluxul de mașini, iar în model se controlează fluxul de date.

Corespondența dintre cele două tipuri de fluxuri este realizată de către dispozitive intrare/ieșire.

Există senzori care pot fi citiți periodic sau care pot furniza automat parametrii fluxului de mașini. Acestea sunt generatoare de informații pentru controller care va calcula pe baza lor comenzile care sunt transmise semafoarelor și astfel se controlează fluxul de mașini.

Fluxul de date are o frecvență și un tip de date. Citirea și scrierea în flux (stream) se realizează cu o perioadă δ specificată și există un anumit tip (structură) de date (ex. : real, întreg) care sunt acceptate.

Modelul benzii anterior poate fi transformat într-un model bazat pe fluxuri.

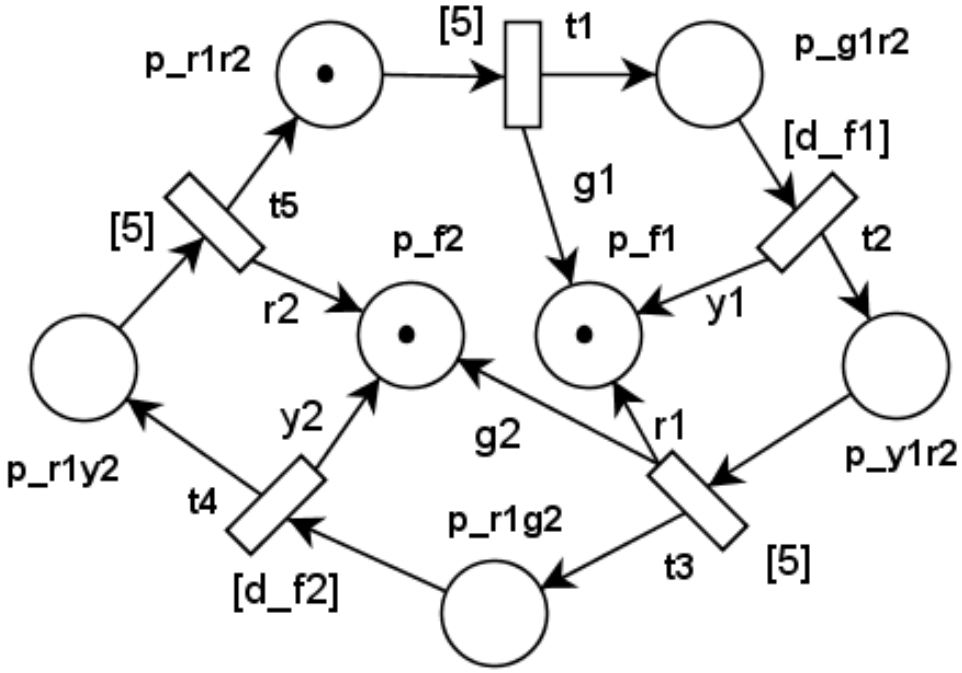


Se presupune că senzorul α furnizează cu perioada de 1 sec, numărul de mașini intrat pe bandă, iar senzorul β pe cel al mașinilor care au ieșit în aceeași perioadă de 1 sec. Locația p_x va stoca numărul de mașini de pe bandă în acel moment de timp.

Modelul matematic realizat este:
$x(\tau+1) = x(\tau) + u(\tau) – e(\tau)$
cu τ s-a notat timpul.

Există condiția $max \geq x(\tau)$.
În timp ce u(τ) este o mărime necontrolabilă, ieșire e(τ) poate fi controlată prin valoarea introdusă în locația p_TL.
Există condiția reală ca numărul de mașini de la un moment dat să nu depășească numărul maxim de mașini care reprezintă capacitatea benzii.

Fig. 33. Modelul unui controller



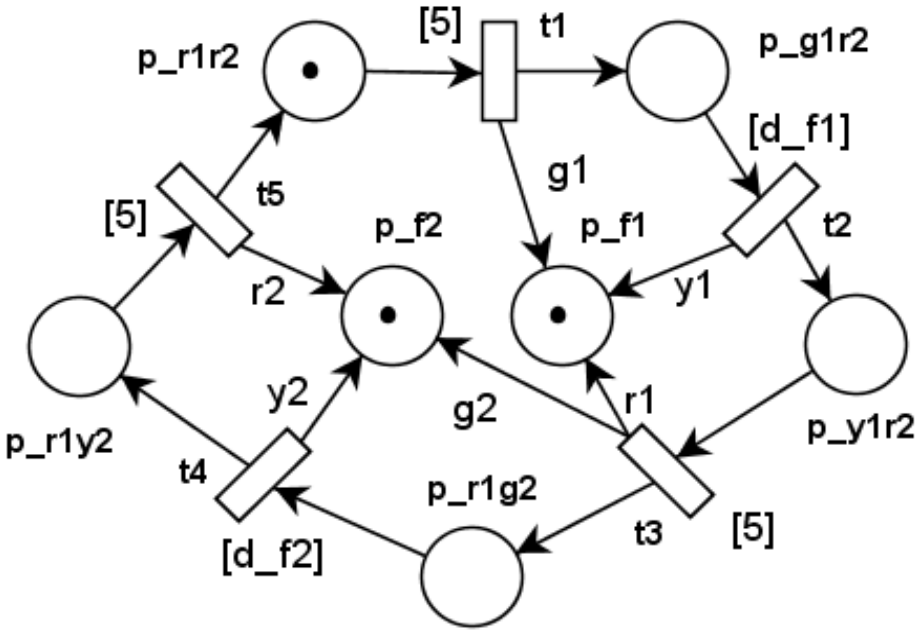
Modelul controllerului cu durate fixe ale fazelor

Pentru sinteza controllerului se cere ca Plant să permită trecerea prin intersecție periodic cu perioada specificată divizată între cele două faze cu duratele d_f1 și d_f2, pentru schimbarea fazelor durata culorii galben să fie de 5 sec., iar durata de roșu peste tot să fie de 5 sec.

Figura alăturată modelează controllerul cu durate fixe ale fazelor marcate pe figură cu d_f1 și d_f2.

Semnificaația nodurilor:

- p_r1r2 este locația care exprimă că semafoarele arată culoarea roșie peste tot (engl.: clearence red)
- p_g1r2 precizează culoarea verde pentru faza 1 și roșie pentru faza 2
- p_y1r2 precizează culoarea galbenă pentru faza 1 și roșie pentru faza 2
- p_r1g2 are jeton când faza 1 are roșu, iar faza 2 verde



- p_r1y2 are jeton dacă faza 1 are roșu, iar faza 2 are galben
- Tranzițiile t1, t2, ⋯, t5 au marcate pe arce culorile pe care le impun celor două faze.

Tranzițiile t1, t3, t5 au întârzieri de 5 sec. fiecare.

Locațiile p_f1 și p_f2 sunt canale de ieșire care trebuie să fie conectate la semafoarele p_TL1, ⋯, p_TL4 ale benzilor de intrare în intersecție. Inițial aceste canale au culoarea roșie, iar controlerul este în starea inițială p_r1r2.

Modelul Controllerului cu durate variabile ale fazelor (engl.: closed-loop controller)

Controllerul cu durate fixe (engl.: open loop) nu ia în considerare modificarea fluxurilor de mașini pentru care s-au calculat duratele fazelor.

Controllerul cu durate variabile trebuie să le adapteze în funcție de informațiile primite de la senzori (sau detectori).

Senzorii pot furniza evenimente de intrare sau ieșire a mașinilor, ratele de intrare și ieșire a mașinilor, deci fluxurile de vehicule sau pot măsura (estima) lungimile cozilor de mașini pentru așteptare

Verificarea modelelor OETPN sintetizate

În general, verificarea modelelor OETPN are două obiective:

a. Verificarea unui model OETPN sintetizat
b. Verificarea unui model OETPN implementat

Presupuneri pentru verificarea unui modelul sintetizat:

- Execuțiile gărzilor și a mappingurilor nu au durate
- Mediul cu care colaborează modelul se comportă (ideal) conform cu specificațiile date

Pentru un model OETPN sintetizat se pot verifica:

- logica, adică la producerea unui eveniment Controllerul reacționează (generează evenimente) conform cu cerințele din faza de specificații și nu generează alte evenimente nedorite
- funcționalitatea, adică dacă evoluția unor variabile de stare este conformă cu cerințele

Verificarea formală

- C este modelul OETPN sintetizat
- M este mediul cu care colaborează (în care este integrat) C
- C&M este modelul OETPN al compusului C și M aflate în colaborare
- R cerințele (requirements)

În specificațiile structurii se dau interfețele (intrările și ieșirile) lui M.

Deci sinteza structurii lui C începe cu determinarea legăturii dintre C și M care corespunde interfeței oferite de M.

În specificații de comportament se dau modelul (sau setul de modele ale) lui M și cerințele de comportament (R) ale compusului C&M.

Sinteză: se cere C astfel încât C&M satisface R

Verificare: C&M satisface R ?

5.4 Implementarea

Fig. 34. Diagrama claselor unui model OETPN

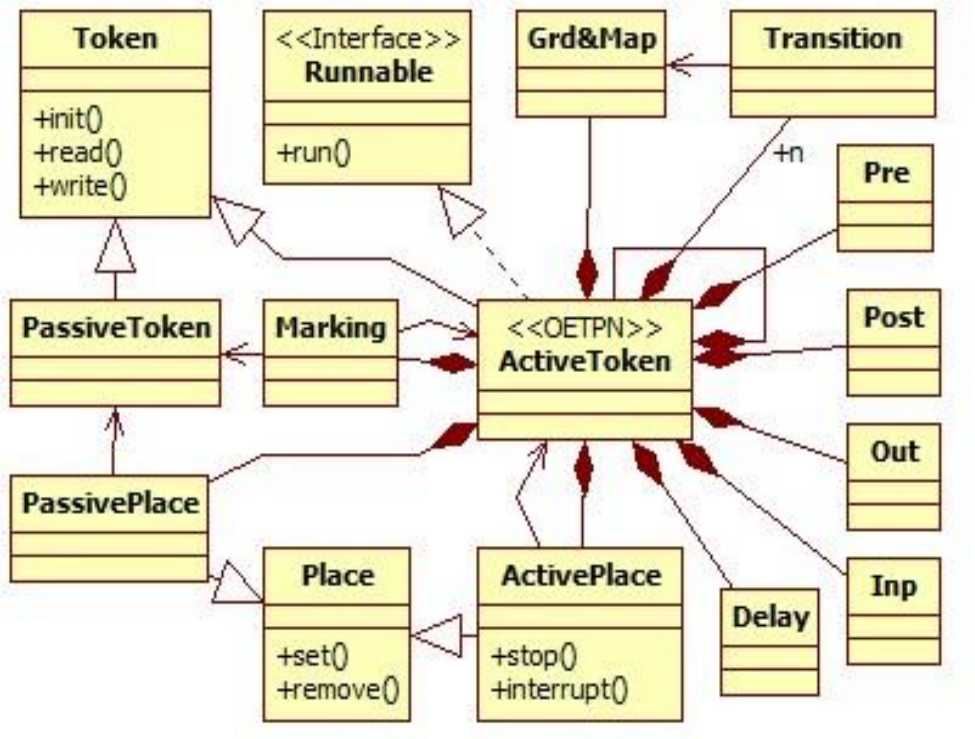
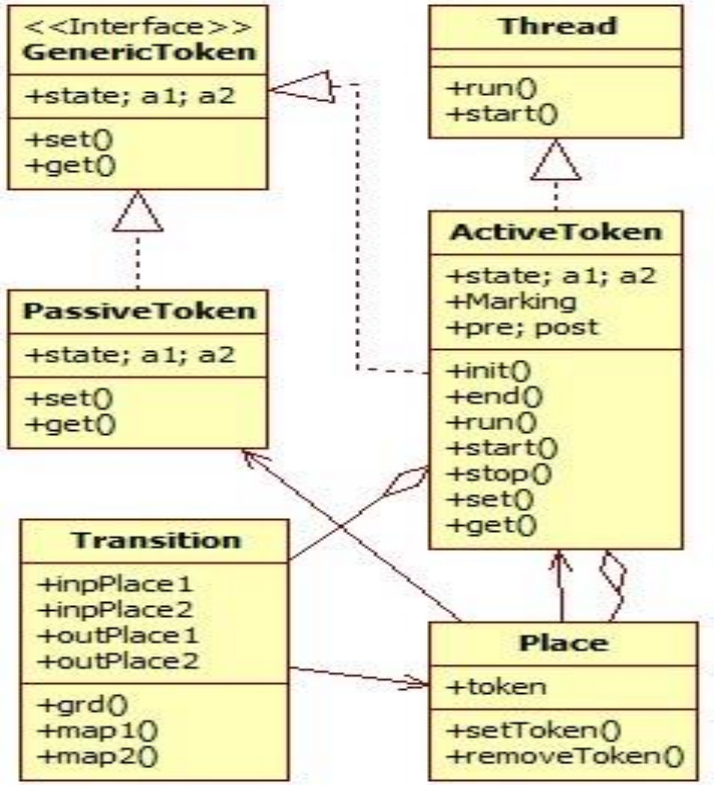


Fig. 35. Relația dintre jetoane și noduri



Implementarea unui model OETPN constă din obiecte de tip locație și de tip tranziție care operează asupra jetoanelor.

Jetoanele au atributele precizate de type(p) și pot avea metode de tip *setter* sau *getter* care permit accesarea jetonului și preluarea informațiilor din el sau setarea unor valori ale jetonului de către tranzții.

Obiectele de tip tranziții au metode de tip grd și map potrivite cu locațiile de intrare și de ieșire ale tranziției.

Obiectele de tip Tranzition conține metodele de prin care se evaluează verificarea condițiilor de gardă și metodele mapping prin care se setează jetoane în locațiile de ieșire.

Setarea unui jeton pasiv într-o locație constă din setarea atributelor jetonului și marcarea locației ca având jeton ($m(p)\neq\varphi$) . Eliminarea jetonului constă din setarea marcajului ca ne-având jeton ($m(p)=\varphi$).

Setarea unui jeton activ (adică a unui OETPN) constă din execuția metodei *init()*, adică se instanțiază modelul cu marcajul inițial și este urmată de startarea executorului submodelului OETPN.

Crearea, accesibilitatea și distrugerea obiectelor

Un obiect este *creat* de o tranziției cu un mapping al ei la prima execuție a tranziției. Obiectul jeton este *accesibil pentru citire* numai când marcajul locației este nenul, adică are un jeton.

Obiectul nu este șters atunci când jetonul dispare, adică este eliminat jetonul, doar că nu mai este accesibil tranzițiilor conectate la locație și evident, nu poate fi accesat de nicio altă tranziți.

Când o tranziție pune din nou un jeton în acea locație, obiectul creat anterior devine accesibil după „împrospătarea" valorilor lui.

Pentru moelul OETPN din Fig. 36 se specifică:

- map_0_1;
- type(p_1);
- grd_1_1; grd_1_2

  Când se setează jetonul în p_1 se execută:

  p_1 = new type(p_1);

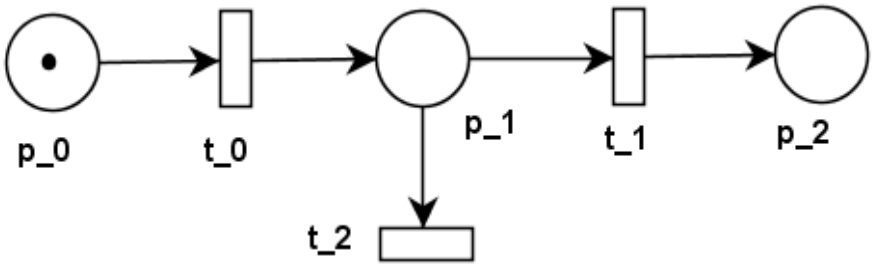


Fig. 36. Exemplu de OETPN

5.5 Analiza și verificarea implementării

Verificarea temporală a implementării implică măsurarea duratelor execuțiilor condițiilor de gardă și a mappingurilor împreună cu metodele pe care le apelează din jetoane (getter și setter).

Cu aceste aceste valori obținute prin măsurare se construiește modelul OETPN al implementării care urmează să fie verificat din punct de vedere temporal, logic și funcțional.

Când implementarea este realizată din mai multe componente, verificarea se face separat în următoarele etape:

1) Se consideră fiecare componentă ca fiind executată separat într-un mediu compus din celelalte componente cu care este în interacțiune și mediul extern. Toate acestea compun mediul de colaborare a componentei
2) Pentru fiecare componentă se fac presupuneri asupra comportamentului mediului componentei.
3) Se verifică modul de reacție a componentei la evenimentele și fluxurile de intrare din mediu.
4) Se verifică modul de reacție a mediului la evenimentele și fluxurile generate de componentă.
5) Dacă unele componente au abateri ale comportamentului de la preesupunerile din etapa 2), atunci se reia verificare prin corectarea presupunerilor.
6) Verificarea se termină atunci când nu mai există abateri între presupuneri și comportamentul rezultat al
   tuturor componentelor.

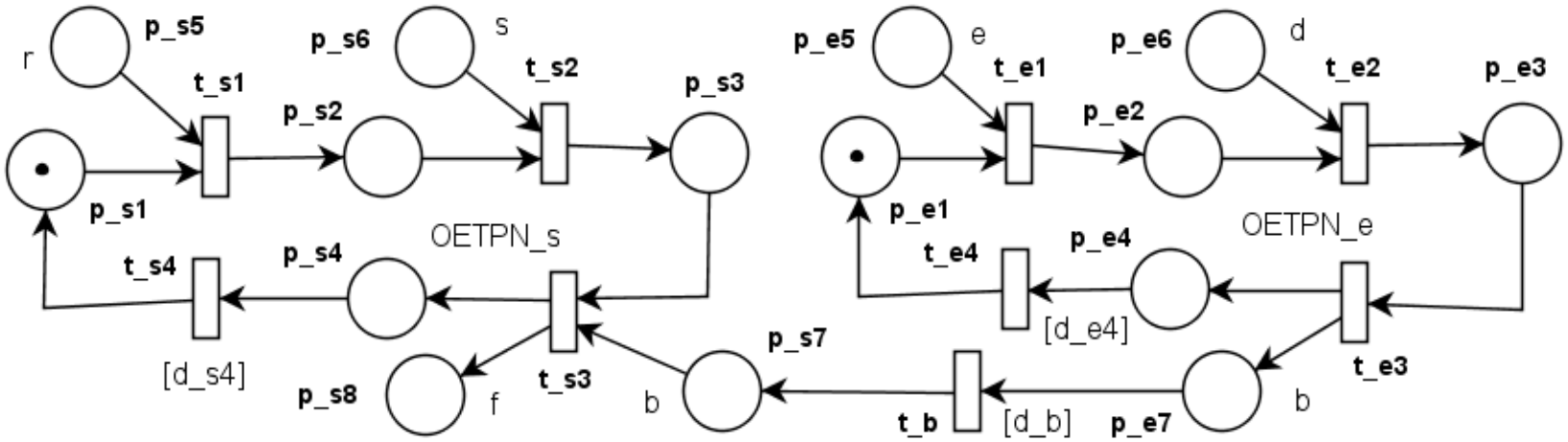
Verificarea logică are în vedere analiza gărzilor pentru a determina evoluția modeleului, adică secvența de tranziții care este executată.

Verificarea temporală are în vedere când răspunde modelul implementat la evenimente interne și externe luând în considerare încărcarea (în general variabilă a) procesoarelor.

Verificarea funcțională are în vedere variația variabilelor programului și determinarea stărilor în care acesta ajunge.

Exemplu: Sistem de asistare a unui șofer (engl.: driver asistent)

Fig. 43. Driver asistent

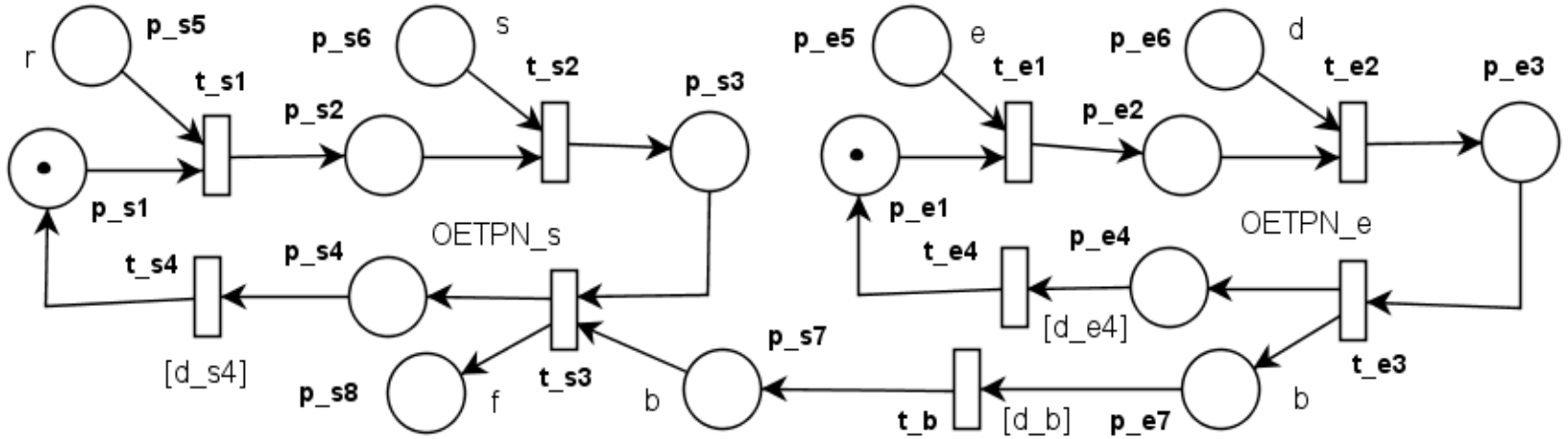


Specificații:

Se consideră un vehicul care are:

- *r* un canal de intrare pentru preluarea referinței vitezei cerute de șofer
- *s* un canal de intrare pentru măsurarea vitezei curente (speed)
- *e* un canal de intrare prin care se primesc semnale despre producerea unui eveniment neașteptat în fața vehiculului
- *d* un canal de intare de citire a distanței de la vehicul până la locul de producere a evenimentului

- *f* un canal de ieșire pentru comanda debitului de combustibil (fuel), deci a puterii curente
- *b* un canal de ieșire pentru comanda frânării (brake)



Cerințe:

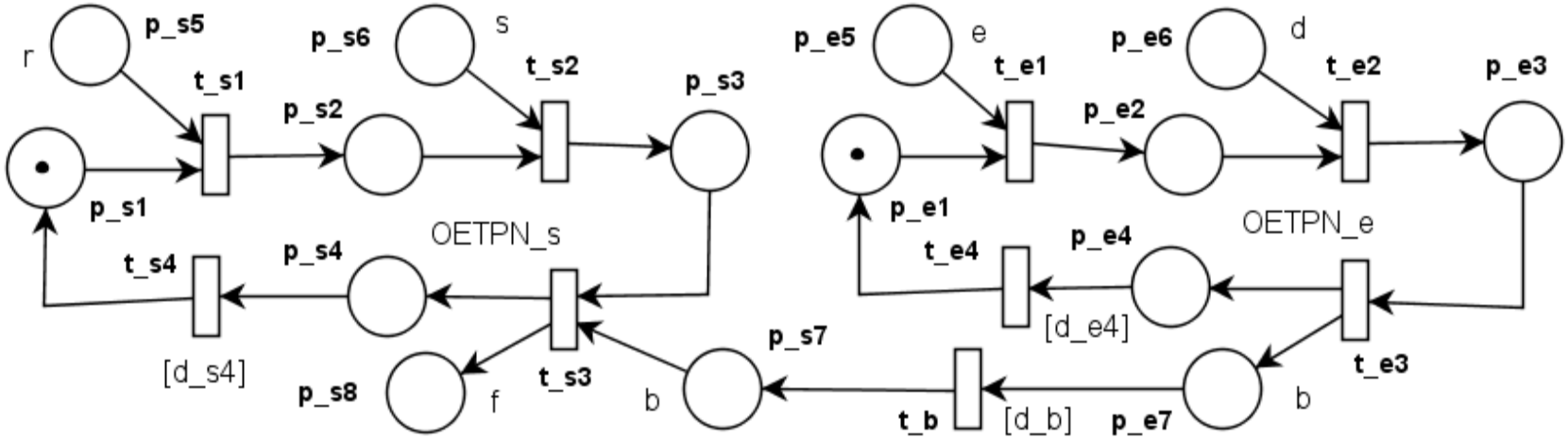
Se cere realizarea unui sistem de asistare a șoferului pentru controlul vitezei în regim normal de rulare și de frânare automată în cazul producerii unui eveniment în fața autovehiculului, cum ar fi un pieton care sare în față sau un vehicul care iese din parcare. Asistentul primește referința *r* a vitezei de la șofer, citește viteza *s (speed)* a vehiculului, calculează accelerația sau decelerația necesară și comandă creșterea sau descreșterea puterii generate prin semnalizarea *f (fuel)* transmisă motorului. Activitățile trebuie să se reia cu perioada de d_s4 milisecunde. Dacă sistemul de frânare semnalează începerea activității de frânare, trebuie să se comande motorului alimentarea pentru regimul de relantiu.

Asistentul este semnalizat asincron despre producerea unui eveniment *e* care necesită reacția urgentă prin frânare astfel încât să nu se ajungă la locul de producere. Asistentul citește de la radar distanța până la eveniment, calculează decelerația necesară și comandă frâna. Tmpul de reacție trebuie să fie sub 0.1 secunde. Dacă dispare cauza evenimentului, trebuie să se reia controlul vitezei conform referinței șoferului în mai puțin de 0.1 sec..

Dacă la reducerea debitului combustibilului până la regim de *idle* nu se reduce viteza la valoarea referinței, șoferul acționează frâna.

Sinteza

S-a sintetizat modelul din Fig. 43. Unde OETPN_s realizează controlul vitezei, iar OETPN_e modelează reacția la un eveniment urgent asincron.

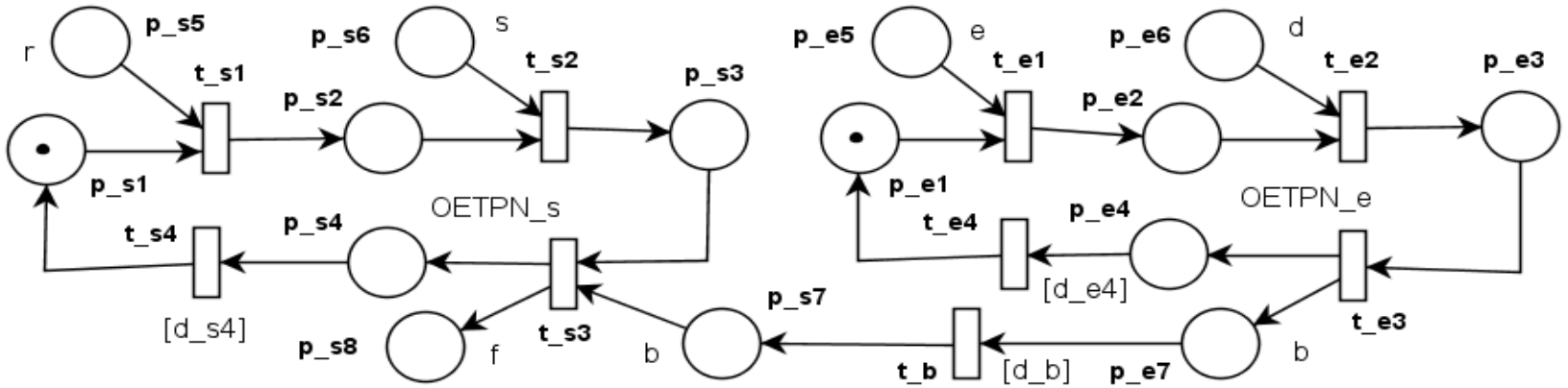


Inp_s = {}; Out_s = {}

Inp_e = {} ; Out_e = {}

Relațiile de evoluție: ?????

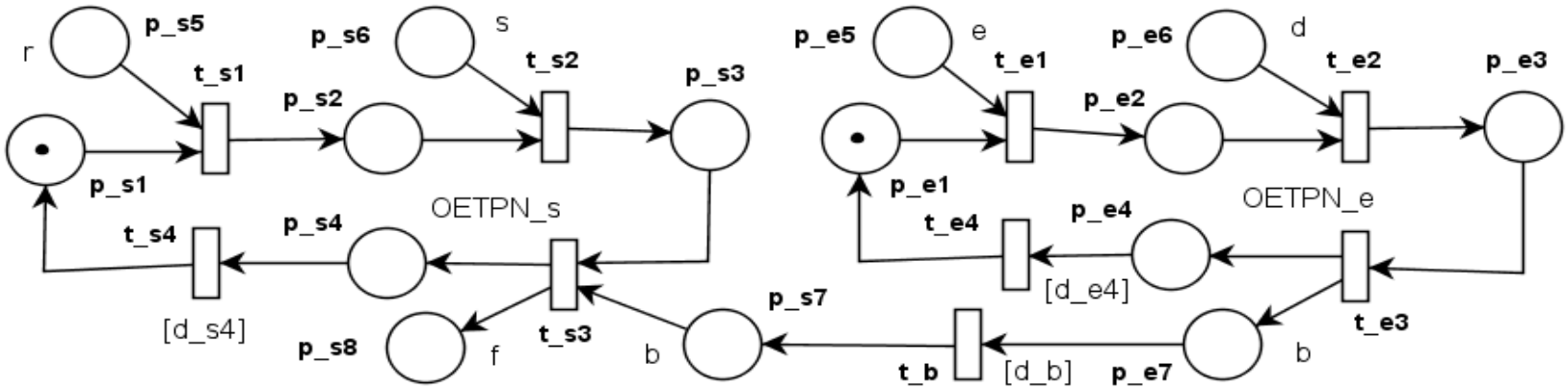
Verificarea logică (logical verification) a modelului sintetizat



Verificarea temporală (temporal verification) a modelului sintetizat

Verificarea funcțională (functional verification) a modelului sintetizat

Verificarea logică (logical verification) a modelului implementat

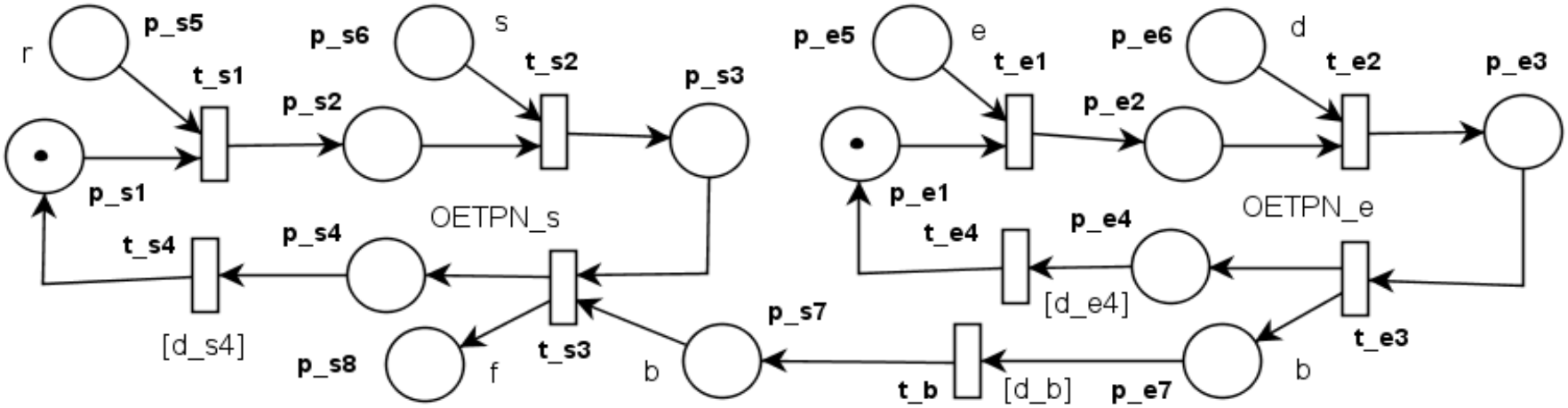


Verificarea temporală (temporal verification) a modelului implementat

Verificarea funcțională (functional verification) a modelului implementat

5.5 Testarea

Se cere determinarea secvențelor de test pentru exemplul anterior

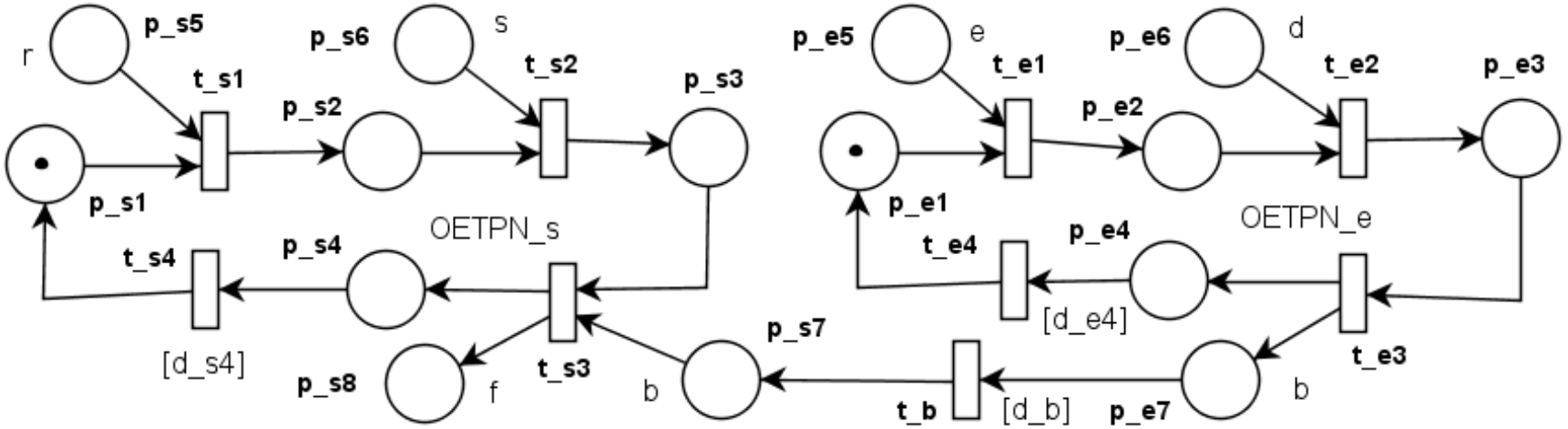


Testarea regimului normal de funcționare

Testarea reacției la evenimente diferite

Testarea timpilor de reacție

5.6 Integrarea

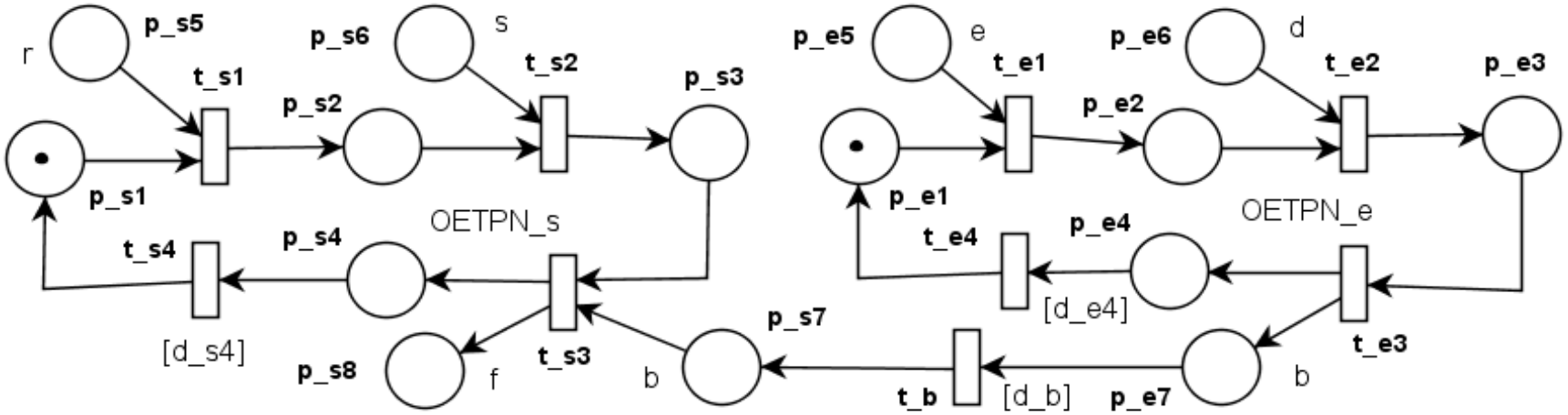


➔ Adăugarea acestei funcții în sistemul anterior de control

Cu cine interacționează?

Poate influența negativ alte părți ale programului?

5.7 Mentenanța sau updatarea software-ului



S-au determinat anomalii. ← Comportament incorect.

Corectarea lor influențează negativ comportamentul altor părți ale programului?

Care sunt părțile cu care interacționează componenta (funcția) care este modificată?

*

***

END

***

*