KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Jonas Padvarskas

SISTEMŲ IMITACINIŲ MODELIŲ AUTOMATIZUOTAS SUDARYMAS PANAUDOJANT FORMALIAS PLA SPECIFIKACIJAS

Magistro darbas

Vadovas:

prof. habil. dr. H. Pranevičius

KAUNAS, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. dr. Eduardas Bareiša
2008.05.__

SISTEMŲ IMITACINIŲ MODELIŲ AUTOMATIZUOTAS SUDARYMAS PANAUDOJANT FORMALIAS PLA SPECIFIKACIJAS

Informatikos mokslo magistro baigiamasis darbas

Vadovas:
prof. habil. dr. H. Pranevičius
2008.05.__

Recenzentas:
doc. dr. Tomas Blažauskas
2008.05.___

Atliko:
IFM 2/2 gr. Studentas
Jonas Padvarskas
2008.05.26

KAUNAS, 2008

Summary

Automated Creation of Systems Imitation Models Using PLA Formal Specifications

Formal specifications can not provide the correctness by themselves, while designing complex systems. Models that are formally specified need correctness proof that can be done by performing imitation modeling on a computer. This problem can be solved by automated creation of imitation models, so that functionality and correctness of various solutions can be verified in early stages of system design.

The article presents the implementation of algorithm for automated creation of imitation models using PLA formal specifications. It also presents an overview of existing modeling techniques and methods. Describes the architecture of formal specifications analysis software and gives experiment results while analyzing network devices interaction.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

1. Įvadas

Vienas svarbiausių programinės įrangos kūrimo etapų – tai sistemos specifikavimas. Labai svarbu, kad sistemos veikimas būtų kuo tiksliau apibrėžtas ankstyvosiose kūrimo stadijose. Kuo detalesnė ir tikslesnė specifikacija, tuo paprasčiau ji yra realizuojama, tuo mažesnė klaidų tikimybė galutinėje realizacijoje. Todėl svarbu teisingai parinkti specifikavimo detalumo lygį sistemai. Jei sistema yra kritinė, klaidų tikimybė turi būti nykstamai maža, o sistema turi tiksliai atitikti reikalavimus. Tokiai sistemai specifikuoti tikslinga naudoti formalius specifikavimo metodus, kurių pagalba sistema bus aprašyta tiksliau, sumažintas klaidų skaičius, ypatingai pradinėse projektavimo fazėse ir pagerintos bendrosios sistemos veikimo charakteristikos. Formalieji metodai gali būti naudojami kaip papildomas kokybės matas, kuris pradinėse kūrimo stadijose gali atskleisti nesuderinamumus, neužbaigtumą ir kitus kuriamos sistemos trūkumus.

Darbe naudojamas formalizavimo ir analizės metodas, kuris remiasi atkarpomis tiesinių agregatų (PLA) formalizavimu [1][2]. Šis metodas priklauso automatų modelių klasei, tačiau išskirtinis tuo, kad sistemos būsenai aprašyti naudojamos ir diskrečios ir tolydžios koordinatės (laiko sąvoka). Šis formalizavimo metodas yra išraiškingas, su griežtomis formaliomis semantikomis, kurios leidžia atlikti realaus laiko paskirstytų sistemų analizę. Atkarpomis tiesinių agregatų metodas ir jo taikymas sudėtingų sistemų modeliavimui pirmą kartą buvo pasiūlytas 1971 metais N. Buslenko . PLA metodas gali būti panaudotas telekomunikacijų srityje protokolų tyrimui [7][8], logistikoje ir kitose pramonės srityse [9].

Projektuojant sudėtingas sistemas, formali specifikacija neužtikrina aprašomos sistemos veikimo teisingumo. Formaliai aprašytam probleminės srities modeliui yra būtinas modelio teisingumo pagrindimas, atliekant jo imitacinį modeliavimą kompiuteryje. Šios problemos sprendimui yra reikalingas integruotos analizės įrankis, kuris leistų automatizuoti projektuojamos sistemos imitacinių modelių sudarymą. Imitacinio modelio dėka funkcionalumas ir atskirų sprendimų teisingumas gali būti patikrinti dar ankstyvosiose kūrimo stadijose. Imitacinis modeliavimas tai formaliai aprašytos sistemos darbo imitavimas vieningoje formalios specifikacijos bazėje, sudarant dirbtinę modelio funkcionavimo aplinką.

1.1. Tikslai ir uždaviniai

Darbo tikslas – sukurti sudėtingų sistemų atkarpomis tiesinių formalių specifikacijų (PLA) integruotos analizės įrankio (FSA) imitacinių modelių automatizuoto sudarymo ir imitacinių modelių trasavimo (derinimo) posistemes, kurių pagalba būtų galima:

- sudaryti sistemos imitacinius modelius, leidžiančius įvertinti funkcionavimo charakteristikas;
- atlikti sudarytų imitacinių modelių trasavimą (derinimą).

Mokslinis elementas:

- diskrečių įvykių imitacinio modeliavimo metodas ir jo panaudojimas PLA specifikacijų imitacinių modelių sudarymui;
- imitacinio modelio sudarymo metodikos panaudojimas modelio elgsenos trasavimui.

1.2. Dokumento paskirtis

Dokumente nagrinėjama agregatinių specifikacijų imitacinių modelių sudarymo problema, pateikiamas sprendimas ir esamų realizacijų metodų analizė. Dokumente aprašyti realizuoto sprendimo programinės įrangos bendrieji principai ir eksperimentai.

1.3. Dokumento turinys

Šiame darbe aprašomi imitacinio modeliavimo algoritmai ir metodai, sukurtos sudėtingų sistemų specifikacijų integruotos analizės automatizavimo įrankio (FSA) imitacinio modeliavimo ir trasavimo posistemės. Antrame skyriuje apžvelgiami egzistuojantys algoritmai ir metodai, pateikiamas darbe realizuoto algoritmo aprašymas. Trečias skyrius skirtas FSA imitacinio modeliavimo ir trasavimo posistemių architektūrai aprašyti. Ketvirtame ir penktame skyriuose pateikiami įrankio kokybės tyrimo ir eksperimentų rezultatai. Dokumento pabaigoje pateikiamos išvados, literatūros sąrašas ir priedai.

2. Analitinė dalis

2.1. Įžanga

Galima teigti, kad imitacinis modeliavimas yra svarbiausia integruotos analizės veika, kadangi tik nuo imitacinio modeliavimo efektyvumo, priklauso kitų sistemos dalių, kaip verifikavimo ir validavimo teisingas ir efektyvus funkcionavimas. Taip pat būtina pabrėžti imitacinio modeliavimo metu surenkamų ir apdorojamų duomenų svarbą. Imitacinis modeliavimas be duomenų – tai visiškai neturintis prasmės procesas, kuris gali būti panaudotas tik kitose veiklose, bet modeliavimas kaip atskiras vienetas, galima sakyti, yra beprasmis.

2.2. Problema

Projektuojant sudėtingas sistemas, formaliai aprašytam probleminės srities modeliui yra būtinas modelio teisingumo pagrindimas, atliekant jo imitacinį modeliavimą kompiuteryje.

2.3. Formali specifikacija

Formali specifikacija – tai sistemos savybių rinkinio išraiška formalioje kalboje, tam tikrame abstrakcijos lygyje [12]. Kitais žodžiais tai matematiškai aprašytas sistemos modelis. Žodis „formalus“ dažnai yra maišomas su žodžiu „tikslus“. Aišku, kad pastarasis paveldi pirmąjį, bet tik ne atvirkščiai. Specifikacija yra formali, jei kalba, kuria ji parašyta yra sudaryta laikantis trijų taisyklių:

- aiškios ir griežtos sintaksės taisyklės;
- taisyklės aprašančios semantiką;
- taisyklės, kuriomis galima įžvelgti naudingą informaciją specifikacijoje.

Norint parašyti teisingą specifikaciją, reikia įvertinti šiuos aspektus:

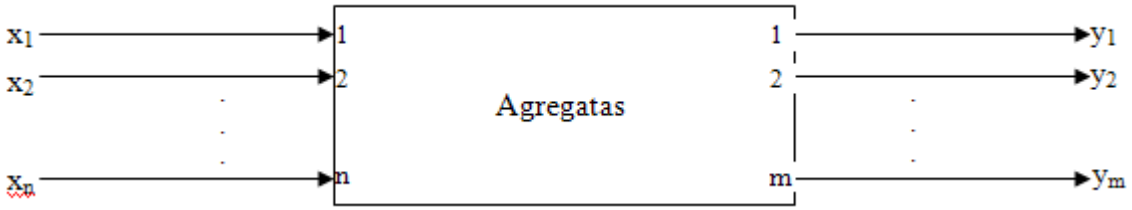
- specifikacija turi būti adekvati – ji turi labai tiksliai nusakyti problemą;
- specifikacija turi būti nuosekli – jei paimsime visas specifikuotas savybes į visumą, tai turi būti teisinga;
- specifikacija turi būti nedviprasmiška – negali turėti dviprasmybių suvokiant kurį nors teiginį kaip tiesą;
- specifikacija turi būti pilna – žemesnio lygio savybių rinkinys, turi būti pakankamas, kad būtų galima apibūdinti aukštesnio lygio teiginį;

- specifikacija turi būti minimali – negali turėti perteklinės informacijos, ar savybių kurios nesusijusios su problema [12].

2.4. Agregatinė specifikacija

Sudėtingų sistemų agregatinio formalizavimo teorinis pagrindas – atkarpomis tiesinių agregatų formalizavimas. Šis formalizavimo metodas yra išskirtinis tuo, kad sistemos būsenai aprašyti naudojamos diskrečios ir tolydžios koordinatės. Atkarpomis tiesiniai agregatai priklauso automatų modelių klasei. Kaip ir automatas, atkarpomis tiesinis agregatas aprašomas nurodant būsenų aibę Z, įėjimo signalų aibę Y bei perėjimo operatorių (atvaizdavimą) H ir išėjimo operatorių G. Tačiau agregatas turi nemažai ypatybių, skiriančių šį modelį nuo automatinių modelių [1].

Aprašinėjant sistemą *PLA* kalba, pirmiausiai sistema suskaidoma į atkarpomis tiesinius agregatus. Atkarpomis tiesiniai agregatai priklauso automatų modelių klasei. Kaip ir automatas, atkarpomis tiesinis agregatas aprašomas nurodant būsenų aibė Z, įėjimo signalų aibė Y bei perėjimo operatorių (atvaizdavimą) H ir išėjimo operatorių G (žr. 2.1 paveikslą).



2.1 pav. Agregatas

Agregato funkcionavimas stebimas laiko momentu $t \in T$ aibėje, t.y. agregato būsena $z \in Z$ yra laiko funkcija $z(t)$. Atkarpomis tiesinio agregato būsenos struktūra yra tokia pat kaip ir atkarpomis tiesinio Markovo proceso, t.y.

$$z(t) = (v(t), z_v(t));$$

čia $v(t)$ - diskrečioji būsenos dedamoji,

$z_v(t)$ - tolydi būsenos dedamoji.

Bendru atveju

$$v(t) = \{v_1(t), v_2(t), ..., v_m(t)\},\ z_v(t) = \{z_{v1}(t), z_{v2}(t), ..., z_{vk}(t)\};$$

čia $v_i(t)$ - i-oji diskrečios dedamosios koordinatė,

$z_{vi}(t)$ - i-oji tolydžiosios dedamosios koordinatė.

Kai nėra įėjimo signalų, agregato būsena kinta taip:

$$v(t) = const\ ,\ \frac{dz_v(t)}{dt} = -\alpha_v;$$

čia $\alpha_v = (\alpha_{v1}, \alpha_{v2}, ..., \alpha_{vk})$ - pastovusis vektorius.

Agregato būsena gali pakisti tik dviem atvejais: kai į agregatą siunčiamas įėjimo signalas arba kai viena iš tolydžiųjų dedamosios koordinačių įgyja tam tikrą reikšmę [10][11].

Aprašinėjant agregatus kompiuteryje, naudojama *PLA-CA* kalba. *PLA-CA* – tai *PLA* formalizavimo kalbos versija, pritaikyta kompiuteriams. Kiekvienas agregatas aprašomas atskirai pagal tokius požymius:

1. Įėjimų signalų aibė *X*;
2. Išėjimo signalų aibė *Y*;
3. Išorinių įvykių aibė *E´*;
4. Vidinių įvykių aibė *E"*;
5. Valdymo sekos;
6. Diskrečiosios agregato būsenų dedamosios;
7. Tolydžiosios agregato būsenų dedamosios;
8. Pradinė agregato būsena;
9. Perėjimų iš išėjimų operatoriai [1].

Agregatinėms specifikacijoms validuoti yra sukurti pasiekiamų būsenų ir invariantų metodai. Pasiekiamų būsenų metodo esmę sudaro tai, kad turint sistemos agregatinę specifikaciją generuojama visų sistemos trajektorijų aibė. Tada šios trajektorijos nagrinėjamos sistemos tiriamų savybių atžvilgiu. Taikant invariantų metodą, reikia sudaryti sistemos invariantą ir patikrinti, ar jis teisingas visose galimose sistemos būsenose [1].

2.5. Modeliavimo principai

Nėra griežtai apibrėžtų modeliavimo principų, todėl imitacinis modeliavimas yra laikomas „kūrybine veikla“ [15]. Dėka šio požiūrio modeliavimas yra laikomas sudėtingu uždaviniu, reikalaujančiu išskirtinės kvalifikacijos. Modelis – tai sistemos abstrakcija. Modeliavimo metu apsprendžiama, kurios sistemos dalys turi būti įtrauktos, paprastai yra atsižvelgiama į modeliavimo sritį. Dažniausiai sutinkama paradigma – tai minimalistinis modeliavimas, kuomet sistemos modeliavimui naudojamas kiek įmanoma supaprastintas sistemos prototipas, kuris tenkintų reikalavimus. Taigi modelis turi būti tik tiek geras, kad

atliktų svarbiausius uždavinius. Tačiau tai labai trumparegiškas požiūris, kadangi modeliavimo metu paprastai nėra aišku, kokias tikslais ateityje bus naudojamas modelis ir vėliau gali būti labai sudėtinga realizuoti funkcionalumą, kuris pradžioje nebuvo numatytas ir įvertintas. Šis modeliavimo supratimas atėjo nuo tų laikų, kada skaičiavimo pajėgumai buvo labai riboti ir brangūs. Buvo gaunami imitaciniai modeliai, kuriems reikėjo mažiau programavimo pastangų ir kompiuterio skaičiavimo laiko. Orientavimasis į minimalizmą, sąlygoja sudėtingesnį pakartotinį panaudojimą. Autorius Mueller Ralph savo darbe [5] išskiria šiuos modeliavimo tipus:

- koncepcinis modeliavimas;
- deklaratyvusis modeliavimas;
    - būsenomis grįstas;
    - įvykiais grįstas;
    - hibridinis;
- diskrečių įvykių modeliavimas.

Koncepcinis modeliavimas

Koncepcinis modeliavimas – tai tiriamos sistemos aprašymas natūralia kalba ir/arba grafiniu būdu, vadovaujantis įvairiomis prielaidomis apie sistemą. Kadangi nėra formalaus metodo koncepcinio modelio kūrimui, iškyla dviprasmiškumo problema. Du skirtingi programuotojai sukurs du skirtingus modelius, remdamiesi tuo pačiu aprašymu.

Deklaratyvusis modeliavimas

Būsenos ir įvykiai – tai du pagrindiniai deklaratyvaus modelio komponentai. Tiriamos sistemos elgsena aprašoma būsenos pasikeitimų seka arba tiesiog būsenų perėjimais.

Diskrečių įvykių modeliavimas

Diskrečių įvykių modeliavimas apima *įvykių planavimo*, *veiklų peržiūros* ir *procesų bendravimu* paremtus modelius. Šie modeliai negali būti lyginami tarpusavyje, kadangi jie aprašo sistemos funkcionalumą skirtingais aspektais. Jie neturi griežto formalaus aprašymo, išskyrus tik *įvykių planavimo* modelį, kadangi pastarasis gali būti surištas su įvykių grafais. *Įvykių planavimo* modelis aprašo kiekvieną įvykį atsižvelgiant į vėlesnius įvykius ir būsenos pasikeitimus. *Procesų bendravimo* modelis aprašo kaip esybės „keliauja“ per sistemą, kokių resursų joms reikia, bet pagrindinis vykdymo mechanizmas visgi yra pagrįstas *įvykių planavimo* modeliu. *Veiklų peržiūros* modelis yra panašus į *įvykių planavimo* modelį, bet naudoja fiksuotą laiko didinimą, kas gali sukelti klaidas. Be to šis metodas nėra tinkamas modeliuoti gamybos procesus, tačiau gali būti naudojamas inventorinėms sistemoms su periodine peržiūra.

Diskrečių įvykių modeliavimo kompiuteryje metu, *įvykių planavimo* metodo atveju, būsenų pasikeitimai yra atvaizduojami diskrečių įvykių rinkiniu. Laiko intervalai gali būti atsitiktinio dydžio arba deterministiniai. Jei intervalai yra deterministiniai, jie gali skirtis priklausomai nuo plano arba turi būti pastovūs. Diskrečių įvykių sistemoje būsenos pasikeitimas reiškia, kad įvyko įvykis. Kadangi modelio būsena yra pastovi tarp įvykių, nėra reikalo stebėti šį laiko tarpą. Sistemos būsena laiko momentu aprašoma diskrečiais ir tolydžiais kintamaisiais. Modeliavimo metu vykdomas įvykių planavimas. Planavimo metu modelyje generuojamas įvykis, kuris turės būti įvykdytas ir jam priskiriama laiko dedamoji, kada turės įvykti įvykis. Įvykis tuomet pridedamas prie *būsimų įvykių* eilės. Įvykiai įvyks pagal nustatytą laiko dedamąją eilės tvarka [20].

DEVS – diskrečių įvykių specifikacija

Diskrečių įvykių specifikaciją (DEVS – *Discrete-Event System Specification*) sukūrė Zeigler B. P. [16], kuri yra formalus pagrindas žemo lygio diskrečių įvykių modelių ir jų simuliatorių atvaizdavimui. Ši specifikacija aprašo kalbą, kuri išreiškiama įėjimais, išėjimais, būsenomis ir perėjimų funkcijomis. DEVS yra dalis didelio karkaso, kuriuo siekiama apjungti diskrečius įvykius ir tolydines dinamines sistemas. Taip pat tai vienas išsamiausių imitacinio modeliavimo karkasų, sutinkamų literatūroje, pagrįstas tiksliu formaliu matematiniu būsenų ir perėjimo funkcijų aprašu [5]. Pagrindinė DEVS sudedamoji dalis tai hierarchiniai komponentai. Žemesnio lygio komponentai gali būti apjungti, kad sudarytų agregatinius komponentus. Sujungiantys sąryšiai aprašo kaip komponentai yra apjungti ir kaip bendrauja tarpusavyje. DEVS pagalba sistemos specifikacijoje nelieka dviprasmybių.

Atominis DEVS modelis aprašomas:

$$M = (X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ta)$$

Čia X įėjimo reikšmių aibė, S būsenų aibė, Y išėjimo reikšmių aibė, $\delta_{int} : S \rightarrow S$ vidinio perėjimo funkcija; $ta : S \rightarrow R^+$ laiko poslinkio funkcija; $\delta_{ext} : Q \times X \rightarrow S$ išorinio perėjimo funkcija, kur $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ yra pilna būsena ir e – laikas praėjęs nuo paskutinio perėjimo; $\lambda : S \rightarrow Y$ yra išėjimo funkcija.

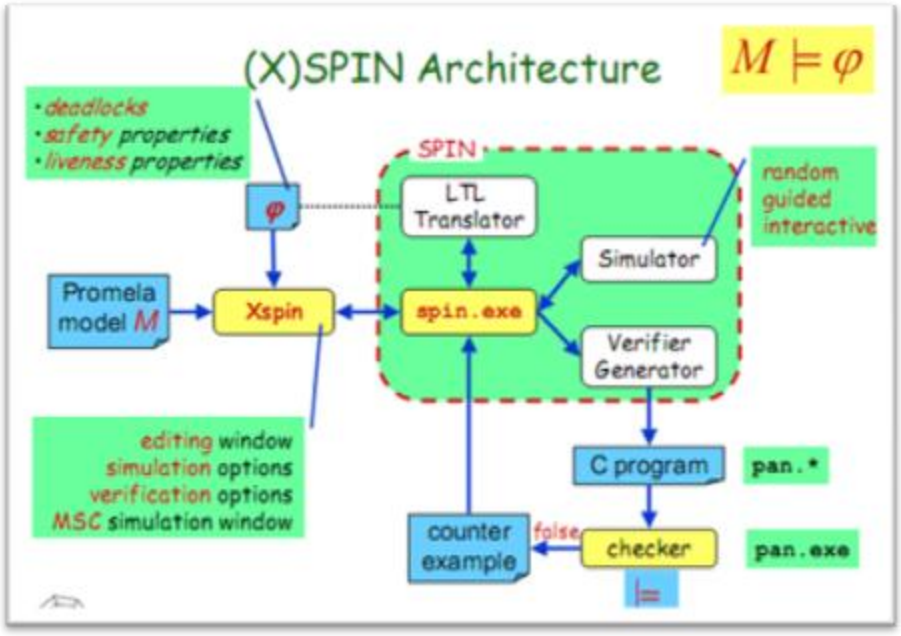
2.6. Egzistuojantys sprendimai

Formalių specifikacijų integruotos analizės įrankių egzistuoja išties nemažai. Tačiau kiekvienas jų yra pritaikytas specifinėms formalioms specifikacijoms. Kaip bebūtų gaila, kūrėjai nelinkę publikuoti algoritmų, kurių pagrindu atliekamas specifikacijų imitacinis modeliavimas. Dažniausiai publikuojamas tik teorinis verifikavimo ir validavimo pagrindas ir bendrieji grafų teorijos algoritmai, skirti patikros užduotims atlikti. Pats imitacinis

modeliavimas laikomas tiesiog kaip esantis funkcionalumas, kurio pagrindu pagrįsta pati integruotos analizės sistema. Šiame skyrelyje apžvelgsime keletą imitacinio modeliavimo realizacijų skirtingoms formaliosioms specifikacijoms ir pateiksime šiame darbe realizuotą imitacinio modeliavimo algoritmą, skirtą FSA integruotos analizės įrankiui.

2.6.1. „SPIN“ imitacinis modeliavimas

SPIN – tai specifikavimo, imitacinio modeliavimo ir validavimo sistema plačiai naudojama kompiuterių protokolams tirti. *SPIN* buvo sukurta kompanijos „Bell Labs“ dar 1980 metais. 2002 metais įrankis gavo prestižinį „System Software Award“ apdovanojimą. *SPIN* naudoja *C* specifikacijos notaciją, kuri padidina jo pritaikomumą pirmoje kūrimo stadijoje [4][17]. *SPIN* leidžia simuliavimą ir validavimą *PROMELA* kalba parašytai specifikacijai. *SPIN* architektūra pateikta 2.2 paveiksle.



2.2 pav. SPIN architektūra [18]

Analizė atliekama su atsitiktiniu arba interaktyviu (dialoginiu) imitavimu. Detalesniam sistemos nagrinėjimui validavimo įrankis patikrina specifikaciją aklaviečių, ciklų be išėjimo ir kt. atžvilgiu. Jeigu sistema yra tiek didelė, kad neužtenka sisteminių resursų (kompiuterio atminties), validavimas atliekamas su atsitiktinai parinktų būsenų aibėmis. Tokios priemonės yra pakankamos korektiškumui ir funkciniams reikalavimams, kurie gali būti realizuoti sistemos prototipe, įvertinti.

Sistemai, specifikuotai *PROMELA* kalboje, *SPIN* gali atlikti imitacinį modeliavimą tos sistemos vykdymo, arba jis gali generuoti programą *C* kalba, kuri vykdo sistemos savybių teisingumo patikrinimą realiu laiku. Tikrintojas taipogi gali būti naudojamas tikrinti sistemos

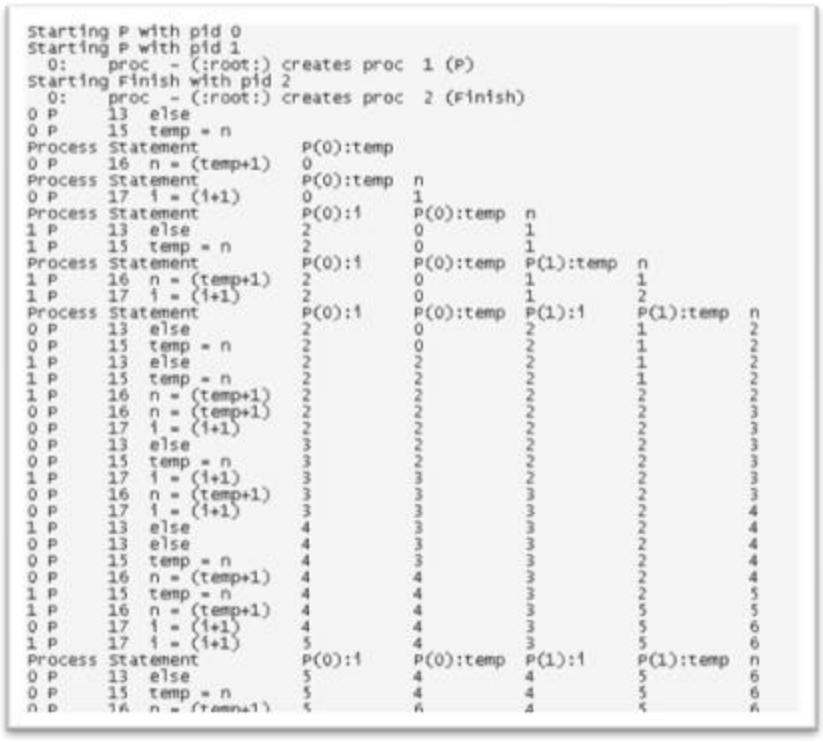
kintamųjų teisingumui, jis gali rasti nereikalingus ciklus, gali patikrinti sekančio laiko momento loginių formuluočių teisingumą. Tikrinimas turi būti naudingas ir naudoti minimalų atminties kiekį. Išsamus tikrinimas gali su matematine tikimybe nustatyti ar aprašytas sistemos elgesys yra be klaidų. Labai didelės tikrinimo problemos, kurios negali būti išspręstos su esama kompiuterine technika, gali būti bandomos spręsti su ekonomiška „būsenos saugojimo bitu“ technika. Šiuo metodu būsenos užima vieta suskirstoma į mažą bitų skaičių pasiekiamoje sistemos būsenoje, su minimaliu pašaliniu efektu [18][19].

SPIN programos kalba PROMELA susideda iš procesų, pranešimų kanalų bei kintamųjų. Procesai yra globalūs objektai kurie atvaizduoja paskirstytos sistemos esybes. Pranešimų kanalai ir kintamieji gali būti aprašyti kaip globalūs arba pačiame procese. Procesai aprašo elgseną, kanalai ir globalūs kintamieji aprašo aplinką, kurioje procesas veikia. [4].

SPIN gali būti naudojamas kaip:

- Imitatorius, leidžiantis greitą analizę naudojant atsitiktinę, valdomą arba dialoginę simuliaciją.
- Nuodugnus verifikatorius, galintis kruopščiai įrodyti vartotojo aprašytų reikalavimų specifikacijos teisėtumą.
- Apytikrio skaičiavimo sistema, leidžianti validuoti net labai didelius sistemų modelius maksimaliai išnaudojant būsenos vietą [4].

Nepaisant minėtų teigiamų SPIN savybių sistema turi trūkumų. Nagrinėjant imitacinį modeliavimą, negalima nepaminėti, jog šis modeliavimas skirtas tik vykdymui, kad būtų galima atlikti modelio patikrą. Imitacinis modeliavimas yra labai ribotas, kadangi jo metu vartotojas gauna labai mažai informacijos, kuri yra visai neapibendrinta ir iš tokios informacijos vartotojas negali gauti bendro vaizdo apie modelio vykdymą ar paties modeliavimo teisingumą (žr. 2.3 pav.). Todėl galima teigti, kad imitacinio modeliavimo SPIN neturi.



```
Starting P with pid 0
Starting P with pid 1
  0:    proc  - (:root:) creates proc  1 (P)
Starting Finish with pid 2
  0:    proc  - (:root:) creates proc  2 (Finish)
0 P     13  else
0 P     15  temp = n
Process Statement           P(0):temp
0 P     16  n = (temp+1)    0
Process Statement           P(0):temp  n
0 P     17  i = (i+1)       0          1
Process Statement           P(0):1     P(0):temp  n
1 P     13  else            2          0          1
1 P     15  temp = n        2          0          1
Process Statement           P(0):1     P(0):temp  P(1):temp  n
1 P     16  n = (temp+1)    2          0          1          1
1 P     17  i = (i+1)       2          0          1          2
Process Statement           P(0):1     P(0):temp  P(1):1     P(1):temp  n
0 P     13  else            2          0          2          1          2
0 P     15  temp = n        2          0          2          1          2
1 P     13  else            2          2          2          1          2
1 P     15  temp = n        2          2          2          1          2
1 P     16  n = (temp+1)    2          2          2          2          2
0 P     16  n = (temp+1)    2          2          2          2          3
0 P     17  i = (i+1)       2          2          2          2          3
0 P     13  else            3          2          2          2          3
0 P     15  temp = n        3          2          2          2          3
1 P     17  i = (i+1)       3          3          2          2          3
0 P     16  n = (temp+1)    3          3          3          2          3
0 P     17  i = (i+1)       3          3          3          2          4
1 P     13  else            4          3          3          2          4
0 P     13  else            4          3          3          2          4
0 P     15  temp = n        4          3          3          2          4
0 P     16  n = (temp+1)    4          4          3          2          4
1 P     15  temp = n        4          4          3          2          5
1 P     16  n = (temp+1)    4          4          3          5          5
0 P     17  i = (i+1)       4          4          3          5          6
1 P     17  i = (i+1)       5          4          3          5          6
Process Statement           P(0):1     P(0):temp  P(1):1     P(1):temp  n
0 P     13  else            5          4          4          5          6
0 P     15  temp = n        5          4          4          5          6
```

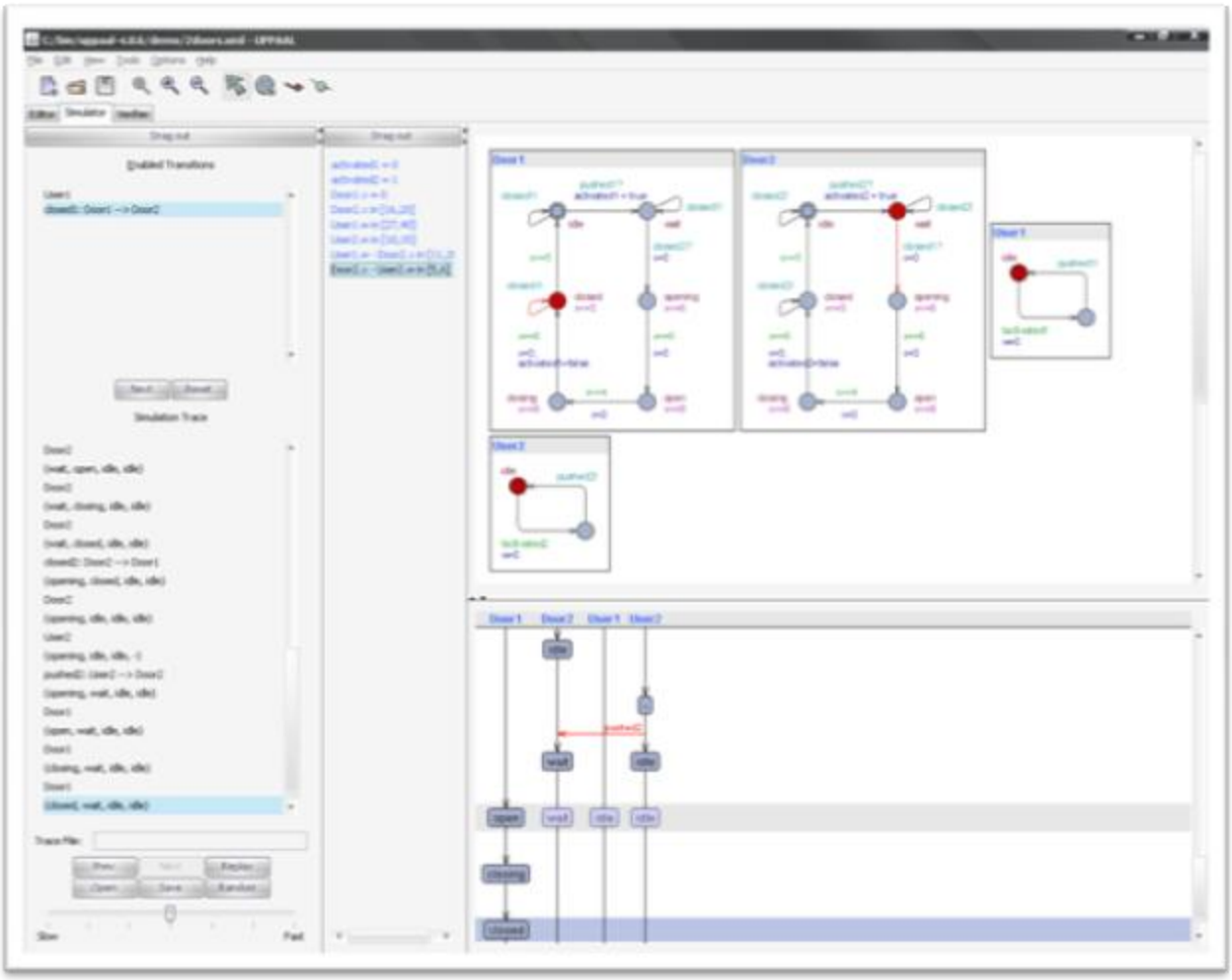
2.3 pav. SPIN imitacinio modeliavimo rezultatai

2.6.2. „UPAAL“ imitacinis modeliavimas

Uppaal yra integruotų įrankių aplinka, skirta formaliai aprašytų sudėtingų sistemų simuliavimui ir tikrinimui. Ši sistema dažniausiai naudojama realaus laiko kontroleriams, komunikavimo protokolams tirti. Įrankis sukurtas bendradarbiaujant Danijos „Aalborg“ ir Švedijos „Uppsala“ universitetams [13][14].

Uppaal susideda iš trijų pagrindinių dalių: kalbos aprašymo, imitatoriaus ir modelio tikrintojo. Aprašymui naudojami laikiniai automatai. Aprašymo kalba yra nedeterminuota kalba su paprastais duomenų tipais (sveiki skaičiai, masyvai ir kt.). Ji naudojama formaliai aprašant tiriamą sistemą.

Imitatorius leidžia tikrinti sistemos veikimą, tikrinant visus galimus veikimo kelius. Imitacinis modeliavimas gali būti atliekamas jau ankstyvojo modeliavimo etapuose ir tokiu būdu suteikia galimybę aptikti ir ištaisyti galimas klaidas pačioje pradžioje. Imitatorius leidžia atlikti sistemos imitacinį modeliavimą atsitiktine tvarka automatiškai arba pažingsniui pagal vartotoją. Imitacinio modeliavimo metu matoma informacija, kaip keičiasi sistemos būsena, atskiri sistemos parametrai. Tačiau imitavimo metu surinkti rezultatai nėra apibendrinami. Imitatoriaus vartotojo sąsaja pateikta 2.4 paveiksle.

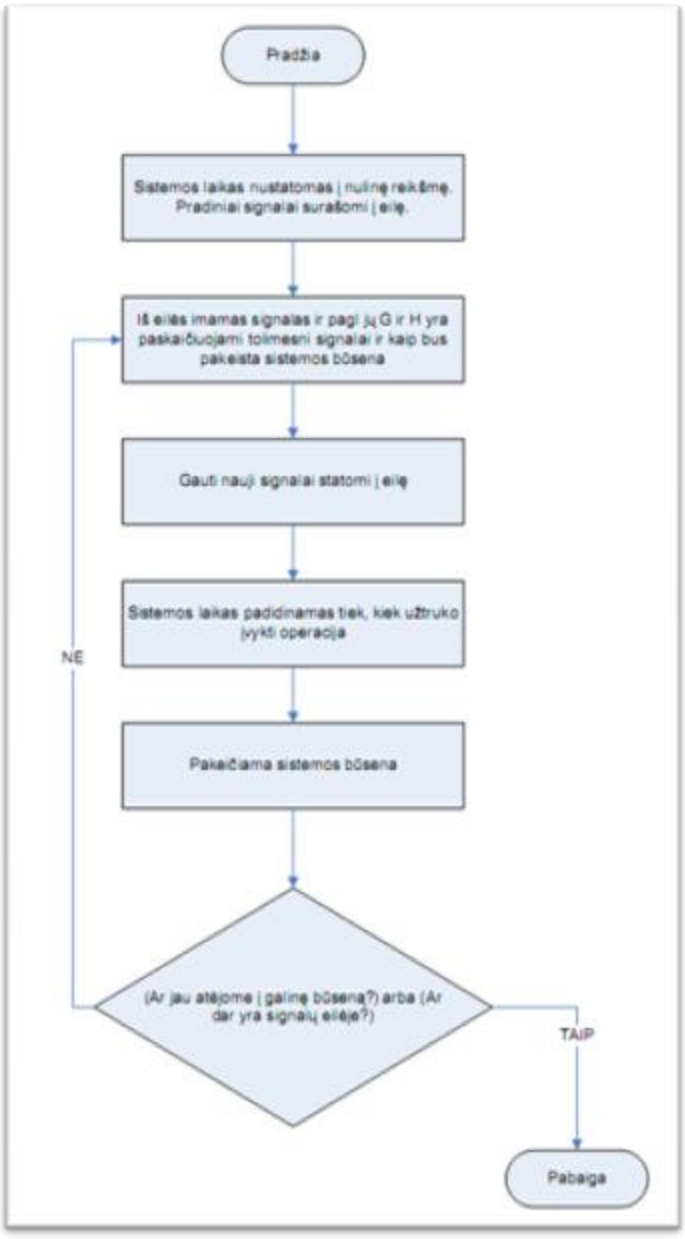


2.4 pav. UPAAL imitatoriaus grafinė vartotojo sąsaja

Pats imitacinio modeliavimo algoritmas nėra viešai platinamas, todėl jo privalumų ir trūkumų įvertinti negalima.

2.6.3. „AgDraw“ imitacinis modeliavimas

AgDraw sistemos detalus aprašymas pateikiamas [6] darbe. Ši sistema pagrinde skirta PLA specifikacijos grafiniam redagavimui, tačiau taip pat turi ir imitacinio modeliavimo galimybes. AgDraw sistemoje naudojamas imitacinio modeliavimo algoritmas yra grindžiamas tuo, kad iš turimos specifikacijos pagal joje aprašytas taisykles generuojamas programinis kodas, kuris vėliau kompiliuojamas ir gaunama pritaikyta programa specifinei probleminei sričiai, kuri vykdo imitacinį modeliavimą. Kiekvienas agregatas yra traktuojamas kaip atskiras objektas, todėl jis atitinka atskirą klasę objektinėje kalboje, agregatų sistema taip pat sudaro atskirą klasę. Generuojamas kodas JAVA programavimo kalboje. Kadangi agregatai yra traktuojami kaip objektai, o sujungimai atstoja ryšius tarp objektų, tai visa agregatinė specifikacija objektiniu požiūriu veikia kaip objektų visuma. Sujungimų pagalba objektai keičiasi pranešimais ir tokiu būdu perduoda informaciją [6]. Imitacinio modeliavimo algoritmas pateikiamas 2.5 paveiksle.

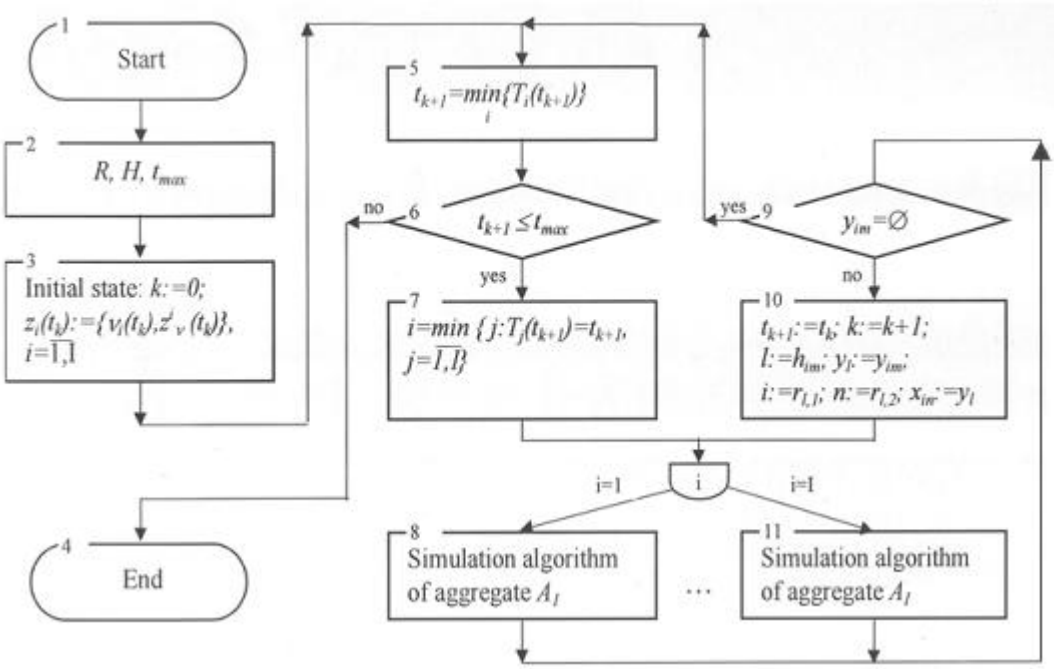


2.5 pav. AgDraw imitacinio modeliavimo algoritmas

Algoritmas yra pritaikytas PLA formaliosioms specifikacijoms, o principas yra artimas SPIN sistemoje naudojamo algoritmo principui: generuojamas programos kodas, kuris vėliau vykdomas kaip imitacinis modelis. Šio sprendimo trūkumas yra tas, kad modelis yra statinis ir atsiradus pakeitimams specifikacijoje reikia iš naujo atlikti modelio generavimą.

2.7. Imitacinio modeliavimo ir trasavimo algoritmai

Darbo tikslas yra sukurti imitacinio modeliavimo algoritmą, pritaikytą kuriamai sistemai. Algoritmas yra pritaikytas PLA formaliosioms specifikacijoms. Apibendrintas imitacinio PLA modeliavimo algoritmas pateiktas 2.6 paveiksle.

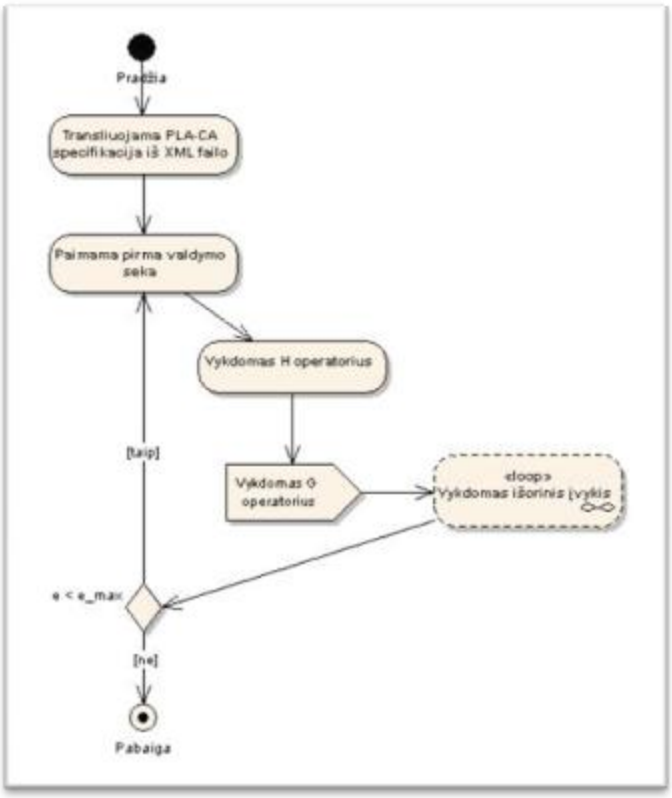


2.6 pav. PLA imitacinio modeliavimo algoritmas

FSA sistemoje imitacinis modeliavimas yra unikalus tuo, kad formalųjį PLA aprašą traktuoja kiekvienai probleminei sričiai vienodai, priešingai nei ankstesniuose skyreliuose nagrinėti algoritmai. Kaip matėme šiuose algoritmuose, kiekvienai probleminei sričiai kuriamas naujas sprendimas, t.y. generuojami probleminės srities objektai ir ryšiai tarp jų. FSA imitacinio modeliavimo metu agregatai traktuojami ne kaip atskiros klasės, bet kaip agregato tipo objektai. Kadangi PLA formali specifikacija yra griežtai apibrėžtos struktūros, kaip minėta ankstesniuose skyriuose, ją sudaro agregatai, sujungimai, agregatus sudaro įėjimai, išėjimai, diskretūs kintamieji, tolydžios dedamosios, H operatoriai, G operatoriai, išraiškos, vidiniai ir išoriniai kintamieji. Todėl galima teigti, jog kiekvienos probleminės srities sprendimą sudarys būtent šių dedamųjų kompozicija. Darome prielaidą, jog netikslinga kiekvieną kartą kuriant ar keičiant specifikaciją generuoti naują imitacinio modelio kodą, jį kompiliuoti ir tik tuomet turėti veikiantį modelį. Pasirinktas sprendimas – transliuoti formalųjį PLA aprašą ne į klases, bet į objektus, tokiu būdu gaunant lankstų imitacinį modelį, kuris yra iš karto veikiantis ir naudojamas. Šis modelis sukuriamas vykdymo metu, todėl dirbant su specifikacija, modelis gali būti keičiamas net vykdymo metu, nereikalaujant pradėti darbo iš naujo, pergeneruoti ir perkompiliuoti modelio kodą. Ši savybė suteikia labai daug lankstumo ir galimybes imitavimo ir specifikacijų išplečiamumui. Specifikacija yra objektai aprašytų klasių, kurios atitinka PLA struktūrą. Natūralu, kad vykdymo metu galime objektus kurti ir/arba keisti. Atsiranda galimybė realizuoti dinaminį PLA modelį (dynPLA). Dinaminis PLA modelis skirtas aprašyti dinaminėms sistemoms, kurių elgsena ir struktūra gali kisti laike [3].

Algoritmo pagrindas

Algoritmo pagrindas susideda tik iš paties imitacinio modeliavimo.



2.7 pav. FSA imitacinio modeliavimo algoritmas

Imitacinio modeliavimo algoritmas nėra sudėtingas (žr. 2.7 pav.). Algoritmui reikalingas sistemos formalusis aprašas, kuris gali būti perduotas iš kitos FSA posistemės arba sutransliuojamas iš duomenų failo. Antrame žingsnyje suformuojama valdymo sekų eilė. Jų pirmumą apsprendžia tolydžioji dedamoji (laiko momento įvertis) – kuo ji mažesnė, tuo aukščiau eilėje bus valdymo seka. Iš eilės paimama pirmoji seka. Trečiajame žingsnyje vykdomas vidinio įvykio H (būsenos pakeitimo) operatorius, o po to G (signalo perdavimo) operatorius. Svarbu pabrėžti, kad šių operatorių „vykdymas“ priklauso nuo išraiškų, nurodytų specifikacijoje. Taigi vykdant G operatorių, bus siunčiamas signalas (skaičiuojama išėjimo vertė) į sujungimais sujungtus agregatus. Pastarieji gavę signalą, vykdys su tuo signalu susietus išorinius įvykius, o tai gali sąlygoti tolimesnius signalų perdavimus ir išorinių įvykių sužadinimus. Išorinio įvykio vykdymas atitinka vidinio įvykio vykdymą – vykdomi H ir G operatoriai. Pasibaigus šiai grandininei reakcijai, bus pakeičiama sistemos būsena įvertinant laiko intervalą ir pertvarkoma valdymo sekų eilė. Kitais žodžiais, žinant agregato būseną $z(\boldsymbol{t}_m), m = 0, 1, 2, ...$, laiko momentas t_{m+1}, kai įvyksta kitas įvykis, apskaičiuojamas taip:

$$\boldsymbol{t}_{m+1} = \min_i \{w(e_i'', \boldsymbol{t}_m)\}, 1 \leq i \leq f.$$

$w(e_i'', \boldsymbol{t}_m)$ – operacijos pabaigos momentas.

Operatorius *H* apibrėžia naują agregato būseną:

$$z(t_{m+1}) = H[z(t_m, e_i)], e_i \in E' \cup E''.$$

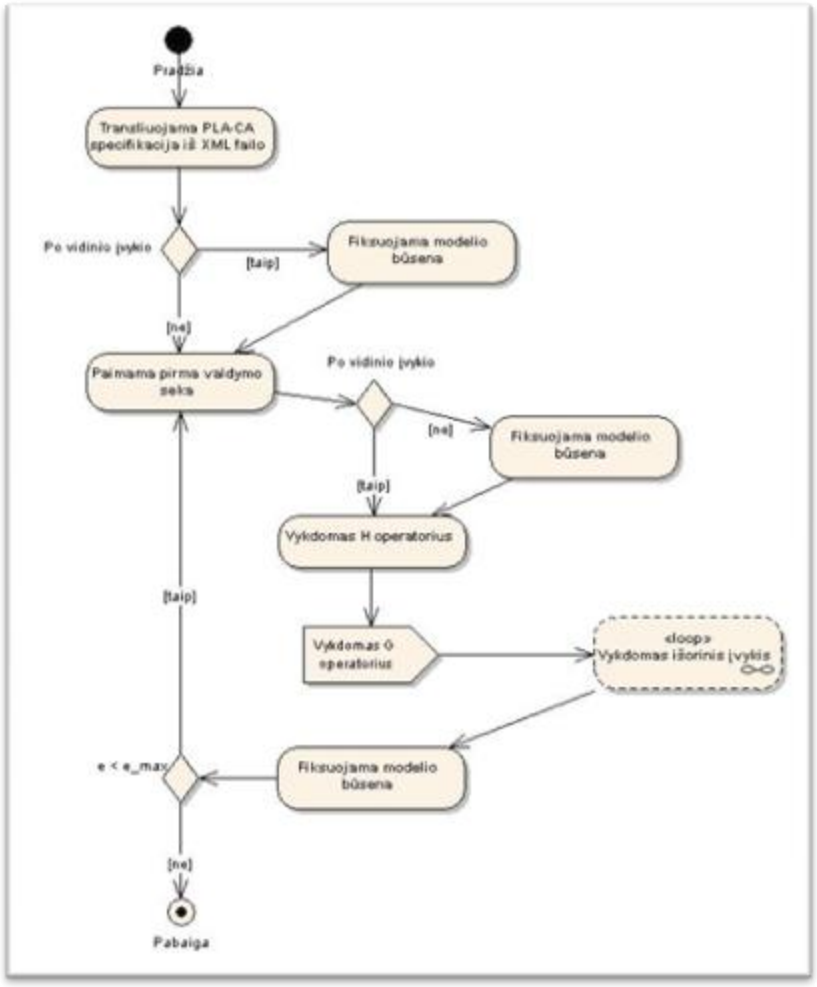
Operatorius *G* apibrėžia išėjimo signalus:

$$y = G[z(t_m, e_i)], e_i \in E' \cup E''.$$

Algoritmas kartojamas kol modelyje įvyksta užduotas skaičius vidinių įvykių e_{max}. Tai apriboja modelio vykdymo laiką, priešingu atveju imitacinis modeliavimas gali būti begalinis, kaip ir realaus gyvenimo procesai: naftos terminalas dirba tol kol jis egzistuoja, duomenų perdavimas tinklu vyksta tol kol egzistuoja tinklas, t.y. kol egzistuoja pats modelis.

Imitacinio modeliavimo duomenų rinkimas ir analizė

Imitacinio modeliavimo algoritmą šiek tiek patobulinę galime gauti informaciją apie modelio būsenos kitimą imitacinio modeliavimo metu ir įvertinti kitimo laike tendencijas, atskirus atvejus ir priklausomybes (žr. 2.8 pav.).



2.8 pav. FSA imitacinio modeliavimo algoritmas ir duomenų rinkimas

Norint gauti modelio būsenos kitimo informaciją, turime nustatyti laiko momentus, kada bus fiksuojama modelio būsena. Algoritme realizuoti du skirtingi atvejai. Pirmuoju

atveju modelio būsena bus fiksuojama pačioje pradžioje ir priklausys nuo valdymo sekos, t.y. po kiekvieno vidinio įvykio. Bus užfiksuota tiek būsenų (neskaičiuojant pradinės), kiek nurodyta iteracijų (vidinių įvykių). Šiuo atveju duomenų bus nedaug, juos lengva analizuoti, aiški imitacinio modeliavimo proceso eiga. Antruoju atveju gaunama ženkliai detalesnė informacija. Šiuo atveju modelio būsena fiksuojama prieš įvykstant bet kokiam įvykiui, t.y. fiksuojama įvykstant tiek vidiniam tiek ir išoriniam įvykiui. Kadangi įvykiai gali sąlygoti išorinius įvykius, o pastarieji kitus išorinius įvykius ir t.t. informacijos kiekis gali būti labai didelis ir tokia informaciją analizuoti gali būti kur kas sudėtingiau. Informacijos kaupimo metodas turėtų būti pasirenkamas įvertinant probleminę sritį, žinant kokia informacija yra aktualiausia sprendžiamam uždaviniui.

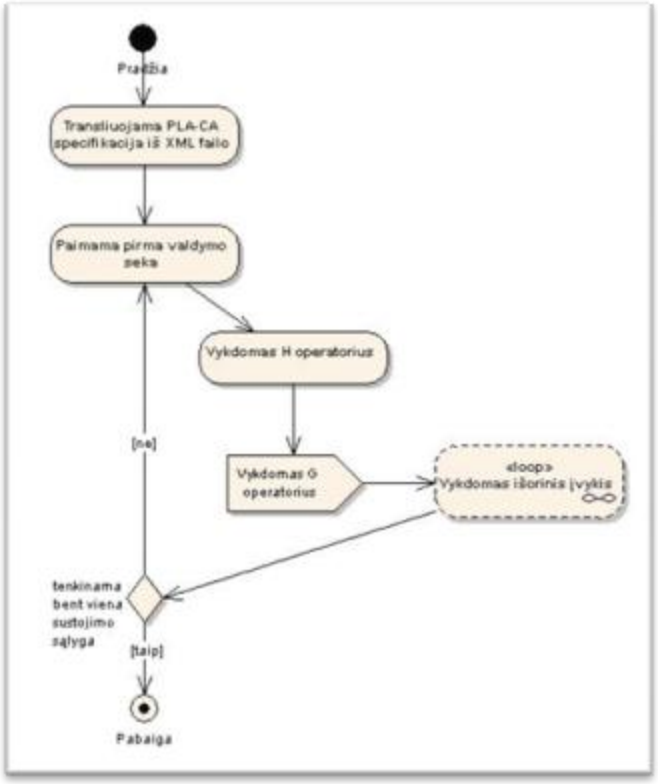
Kadangi imitacinio modeliavimo duomenys yra visiškai priklausomi nuo specifikacijos, modeliavimo metu renkami visi modelio būsenos kitimo duomenys, o vartotojas pats turi pasirinkti jam aktualius duomenis ir jų vaizdavimo būdą. Pavyzdžiui tolydžios dedamosios vaizdžiausiai pateikiamos kreive, o diskretūs kintamieji stulpeline diagrama. Taip pat vaizdavimo būdas labai daug priklauso nuo specifikacijos probleminės srities, todėl modeliavimo metu to parinkti negalima – tai paliekama vartotojui.

Trasavimo algoritmas

Trasavimo uždaviniui išspręsti, modifikuosime imitacinio modeliavimo algoritmą. Trasavimo metu svarbiausia gauti informaciją apie imitacinio modelio būseną, pasirinktu metu arba kai tenkinama užduota sąlyga. Taip pat labai svarbu yra keisti modelio būseną vykdymo metu. Tai pagrindiniai uždaviniai, kurie turi būti išspręsti trasavimo algoritmo pagalba.

Trasavimas tai tas pats imitacinis modeliavimas, tik stipriau kontroliuojamas vartotojo. Imitacinis modeliavimas vykdomas pagal aukščiau aprašytą algoritmą. Tačiau trasavimui reikia sąlygų, kurių pagalba modelio vykdymas būtų valdomas. Imitacinio modeliavimo algoritme modeliavimas buvo apribotas maksimaliu vidinių įvykių įvykdymo skaičiumi e_{max}. Trasavimo atveju tai galėtų būti tik viena iš sąlygų. Algoritme reikalingos sąlygos, kuriomis patikrintume modelio būseną ir imitacinį modeliavimą sustabdytume kai modelis tenkina šias sąlygas. Taigi trasavimo algoritmą sudaro tokios dalys: imitacinio modeliavimo algoritmas ir sustojimo sąlygos. Pirmame žingsnyje gaunamas sistemos aprašas, sekančiame žingsnyje vykdoma anksčiau aprašyto imitacinio modeliavimo algoritmo dalis. Po vidinio įvykio modeliavimo tikrinamos sustojimo sąlygos, jei bent viena sustojimo sąlyga (sustojimo sąlygų gali būti neribotas skaičius) tenkinama imitacinis modeliavimas sustabdomas (gali būti

pratęsiamas su vartotojo įsikišimu), jei ne – imitacinis modeliavimas tęsiamas, vykdomas kitas vidinis įvykis (žr. 2.9 pav.).



2.9 pav. FSA trasavimo algoritmas

Trasavimas arba pilnai valdomas imitacinis modeliavimas atitinka integruotose programavimo aplinkose naudojamą derintuvą (*angl. debugger*), jo pagalba galima keisti kintamųjų reikšmes (čia keičiame modelio būseną, kurią taip pat sudaro kintamųjų rinkinys), sustabdyti kodo vykdymą pasirinktoje eilutėje ir/arba kai pasirinktoje eilutėje yra tenkinama užduota sąlyga (čia kadangi modeliavimas vykdomas ne pagal eilutes, sustojimas gali būti apibrėžiamas po nurodyto vidinių įvykių įvykdymo skaičiaus arba modelio būsenai tenkinant nurodytą sąlygą).

3. Projektinė dalis

3.1. Sistemos paskirtis

Darbo metu buvo kuriama sudėtingų sistemų formalių PLA (*Piece Linear Aggregates*) specifikacijų integruotos analizės sistema (FSA).

Lyginant sistemos funkcionalumą su iki šiol sukurtais analogais, tai vienintelė sistema, kuri apima įvedimo, verifikavimo, imitavimo ir trasavimo galimybes. Tai yra pagrindiniai analizės metodai, reikalingi pilnai formalaus modelio analizei atlikti. Šiame darbe nagrinėjamos imitacinio modeliavimo (toliau imitavimo) ir trasavimo posistemės, kurios yra sudedamosios FSA dalys.

Pagrindinės imitavimo posistemės funkcijos:

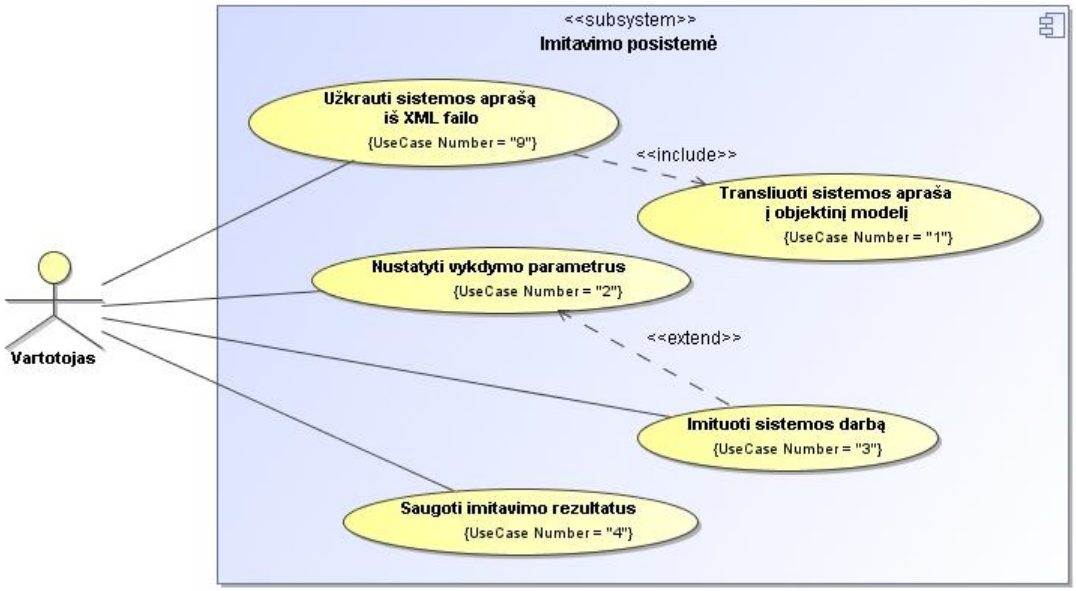
- imitacinio modelio vykdymas;
- rezultatų kaupimas;
- rezultatų pateikimas;
- rezultatų išsaugojimas.

Pagrindinės trasavimo posistemės funkcijos

- imitacinio modelio vykdymas;
- imitacinio modelio keitimas vykdymo metu;
- imitacinio modelio informacijos pateikimas;
- sustojimo sąlygų įvedimas ir redagavimas;
- įvykių sekos valdymas.

3.2. Panaudojimo atvejų vaizdas

3.2.1. Imitavimo posistemė



3.1 pav. Imitavimo posistemės panaudojimo atvejų diagrama

Imitavimo posistemės panaudojimo atvejai

3.1 lentelė

„Užkrauti sistemos aprašą iš XML failo“ aprašymas

Panaudojimo atvejis	Užkrauti sistemos aprašą iš *XML* failo
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas atidaro analizuojamos sistemos aprašą, saugomą kompiuterio kietajame diske *XML* formatu.
Prieš-sąlyga	Analizuojamos sistemos aprašas nėra užkrautas. Vartotojas atliks analizuojamos sistemos veikimo imitavimą.
Sužadinimo sąlyga	Reikia atlikti analizuojamos sistemos veikimo imitavimą.
Po-sąlyga	Analizuojamos sistemos aprašas užkraunamas iš *XML* failo. Pasiruošiama aprašo transliavimui į analizuojamos sistemos objektinį modelį.

3.2 lentelė

"Transliuoti sistemos aprašą į objektinį modelį“ aprašymas

Panaudojimo atvejis	Transliuoti sistemos aprašą į objektinį modelį.
Vartotojas/Aktorius	Objektinio modelio posistemė
Aprašas	Apima procesą, kurio metu užkrautas analizuojamos sistemos aprašas transliuojamas į analizuojamos sistemos objektinį modelį, saugomą kompiuterio atmintyje.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo.
Sužadinimo sąlyga	Reikia atlikti analizuojamos sistemos veikimo imitavimą.
Po-sąlyga	Kompiuterio atmintyje saugomas analizuojamos sistemos objektinis modelis.

3.3 lentelė

„Nustatyti vykdymo parametrus“ aprašymas

Panaudojimo atvejis	Nustatyti vykdymo parametrus
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas nustato arba palieka standartinius parametrus, reikalingus atlikti sistemos veikimo imitavimą.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį.
Sužadinimo sąlyga	Vartotojas nori pakeisti standartinius veikimo imitavimo parametrus.
Po-sąlyga	Išsaugomi naujai nustatyti veikimo imitavimo parametrai.

3.4 lentelė

„Imituoti sistemos darbą“ aprašymas

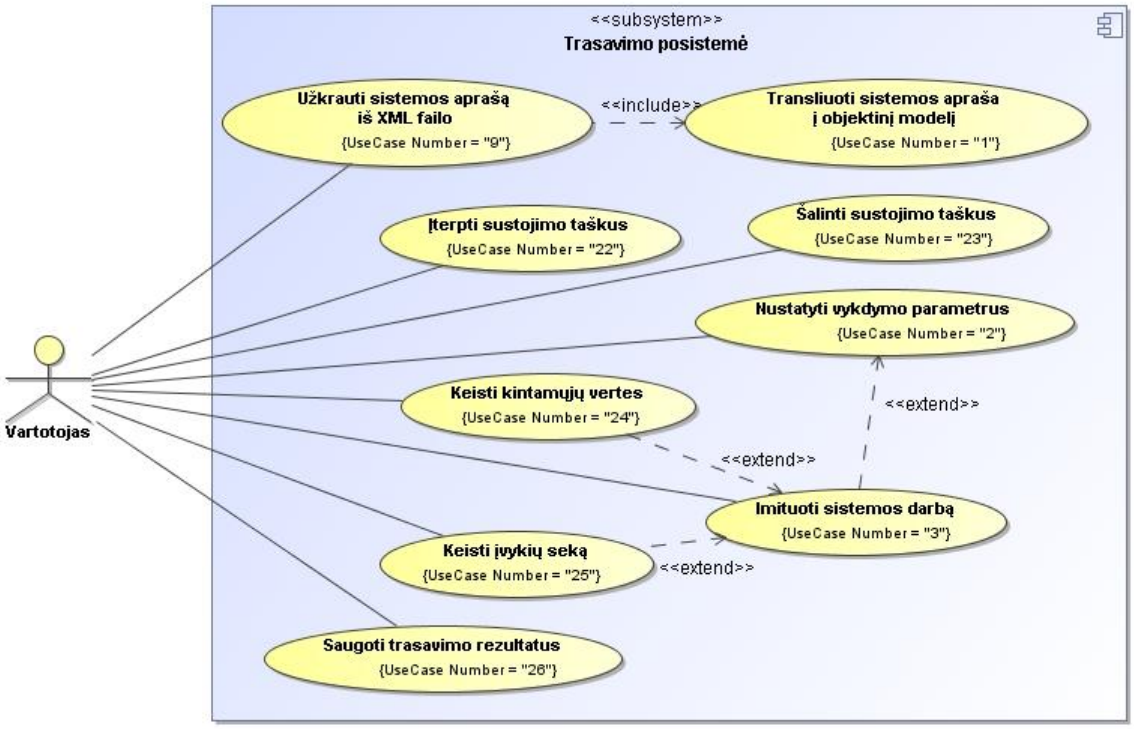
Panaudojimo atvejis	Imituoti sistemos darbą
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu yra vykdomas analizuojamos sistemos veikimo imitavimas.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį.
Sužadinimo sąlyga	Vartotojas nori atlikti analizuojamos sistemos veikimo imitavimą.
Po-sąlyga	Atliktas analizuojamos sistemos veikimo imitavimas. Vartotojui rodomi imitavimo rezultatai.

3.5 lentelė

„Saugoti imitavimo rezultatus“ aprašymas

Panaudojimo atvejis	Saugoti imitavimo rezultatus
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu vartotojas saugo imitavimo rezultatus pasirinktame faile kompiuterio kietajame diske.
Prieš-sąlyga	Sėkmingai atliktas sistemos veikimo imitavimas.
Sužadinimo sąlyga	Imitavimo rezultatus reikia išsaugoti kompiuterio kietajame diske.
Po-sąlyga	Kompiuterio kietajame diske sukuriamas naujas arba perrašomas jau buvęs imitavimo rezultatų failas naujai gautais imitavimo rezultatais.

3.2.2. Trasavimo posistemė



3.2 pav. Trasavimo posistemės panaudojimo atvejų diagrama

Trasavimo posistemės panaudojimo atvejai

3.6 lentelė

„Užkrauti sistemos aprašą iš XML failo“ aprašymas

Panaudojimo atvejis	Užkrauti sistemos aprašą iš *XML* failo
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas atidaro analizuojamos sistemos aprašą, saugomą kompiuterio kietajame diske *XML* formatu.
Prieš-sąlyga	Analizuojamos sistemos aprašas nėra užkrautas. Vartotojas atliks analizuojamos sistemos veikimo trasavimą.
Sužadinimo sąlyga	Reikia atlikti analizuojamos sistemos veikimo trasavimą.
Po-sąlyga	Analizuojamos sistemos aprašas užkraunamas iš *XML* failo. Pasiruošiama aprašo transliavimui į analizuojamos

	sistemos objektinį modelį.

3.7 lentelė

„Transliuoti sistemos aprašą į objektinį modelį“ aprašymas

Panaudojimo atvejis	Transliuoti sistemos aprašą į objektinį modelį.
Vartotojas/Aktorius	Objektinio modelio posistemė
Aprašas	Apima procesą, kurio metu užkrautas analizuojamos sistemos aprašas sutransliuojamas į analizuojamos sistemos objektinį modelį, saugomą kompiuterio atmintyje.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo.
Sužadinimo sąlyga	Reikia atlikti analizuojamos sistemos veikimo trasavimą.
Po-sąlyga	Kompiuterio atmintyje saugomas analizuojamos sistemos objektinis modelis.

3.8 lentelė

„Įterpti sustojimo taškus“ aprašymas

Panaudojimo atvejis	Įterpti sustojimo taškus
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas įterpia trasavimo sustojimo taškus (sustojimo sąlygas).
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį.
Sužadinimo sąlyga	Vartotojas nori sustabdyti sistemos darbą esant duotoms sustojimo sąlygoms, kurias aprašo sustojimo taškai.
Po-sąlyga	Sukurti nauji trasavimo sustojimo taškai.

3.9 lentelė

„Šalinti sustojimo taškus“ aprašymas

Panaudojimo atvejis	Šalinti sustojimo taškus
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas panaikina trasavimo sustojimo taškus.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį. Yra nustatytas bent vienas trasavimo sustojimo taškas.
Sužadinimo sąlyga	Vartotojas nori panaikinti konkrečius trasavimo sustojimo taškus.
Po-sąlyga	Panaikinti pasirinkti trasavimo sustojimo taškai.

3.10 lentelė

„Imituoti sistemos darbą“ aprašymas

Panaudojimo atvejis	Imituoti sistemos darbą
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu yra vykdomas analizuojamos sistemos veikimo imitavimas. Jei yra trasavimo sustojimo taškų, tai veikimo imitavimas sustabdomas ties konkrečiais trasavimo sustojimo taškais.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį.
Sužadinimo sąlyga	Vartotojas nori atlikti analizuojamos sistemos veikimo trasavimą.
Po-sąlyga	Atliktas analizuojamos sistemos veikimo imitavimas. Jei yra trasavimo sustojimo taškų, imitavimas

	sustabdomas ties konkrečiu trasavimo sustojimo tašku ir ekrane rodoma esama analizuojamos sistemos būsena. Vartotojas gali keisti sistemos parametrus, pridėti ar panaikinti sustojimo taškus.

3.11 lentelė

„Nustatyti vykdymo parametrus“ aprašymas

Panaudojimo atvejis	Nustatyti vykdymo parametrus
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas nustato arba palieka standartinius parametrus, reikalingus atlikti sistemos veikimo imitavimą.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį.
Sužadinimo sąlyga	Vartotojas nori pakeisti standartinius veikimo imitavimo parametrus.
Po-sąlyga	Išsaugomi naujai nustatyti veikimo imitavimo parametrai.

3.12 lentelė

„Keisti kintamųjų vertes“ aprašymas

Panaudojimo atvejis	Keisti kintamųjų vertes
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas keičia konkrečius analizuojamos sistemos kintamųjų parametrus sistemos veikimo imitavimo metu.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį. Yra nustatytas bent vienas trasavimo sustojimo

	taškas.
Sužadinimo sąlyga	Reikia pakeisti analizuojamos sistemos veikimo parametrus sistemos veikimo imitavimo metu sustojus tenkinant konkrečią trasavimo sustojimo tašku aprašytą sustojimo sąlygą.
Po-sąlyga	Pakeisti analizuojamos sistemos veikimo parametrai. Tolimesnis analizuojamos sistemos veikimo imitavimas tęsiamas su naujai išsaugotais veikimo parametrais.

3.13 lentelė

„Keisti įvykių seką“ aprašymas

Panaudojimo atvejis	Keisti įvykių seką
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu sistemos vartotojas keičia analizuojamos sistemos įvykių seką veikimo imitavimo procese.
Prieš-sąlyga	Analizuojamos sistemos aprašas užkrautas iš *XML* failo ir sutransliuotas į kompiuterio atmintyje saugomą objektinį modelį. Yra nustatytas bent vienas trasavimo sustojimo taškas.
Sužadinimo sąlyga	Reikia pakeisti analizuojamos sistemos įvykių seką veikimo imitavimo metu sustojus ties konkrečiu trasavimo sustojimo tašku.
Po-sąlyga	Pakeista analizuojamos sistemos įvykių seka. Tolimesnis analizuojamos sistemos veikimo imitavimas tęsiamas su naujai išsaugota įvykių seka.

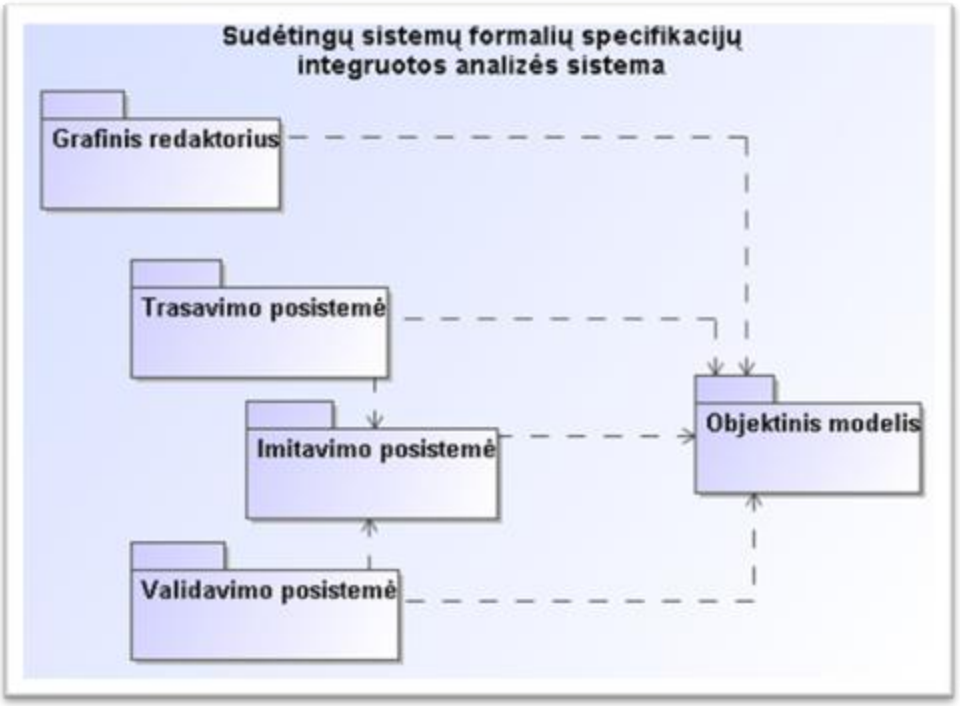
3.14 lentelė

„Saugoti trasavimo rezultatus“ aprašymas

Panaudojimo atvejis	Saugoti trasavimo rezultatus
Vartotojas/Aktorius	Sistemos vartotojas
Aprašas	Apima procesą, kurio metu vartotojas saugo trasavimo rezultatus pasirinktame faile kompiuterio kietajame diske.
Prieš-sąlyga	Sėkmingai atliktas sistemos veikimo trasavimas.
Sužadinimo sąlyga	Trasavimo rezultatus reikia išsaugoti kompiuterio kietajame diske.
Po-sąlyga	Kompiuterio kietajame diske sukuriamas naujas arba perrašomas jau buvęs trasavimo rezultatų failas naujai gautais trasavimo rezultatais.

3.3.Sistemos statinis vaizdas

3.3.1. Apžvalga



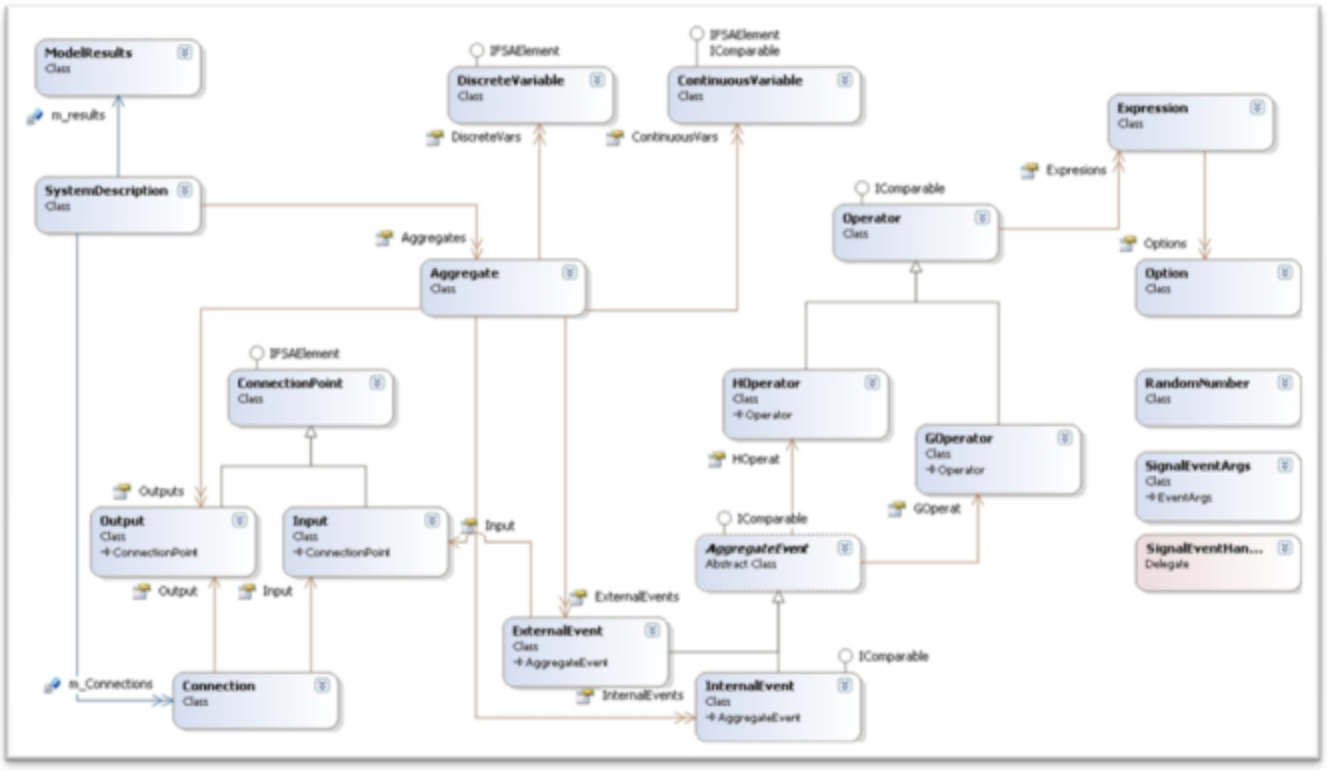
3.3 pav. FSA sistemos paketų diagrama

3.3.2. Paketų detalizavimas

3.3.2.1. Objektinio modelio posistemė

Objektinis modelis skirtas analizuojamos sistemos aprašo saugojimui kompiuterio atmintyje darbo metu. Objektiniame modelyje saugoma visa PLA formalizavimo kalba aprašyta sistemos informacija: sistemos sujungimai, signalai, agregatai ir su jais susijusi informacija (įėjimai, išėjimai, diskretieji ir tolydieji kintamieji, H ir G operatoriai). Taip pat objektinis modelis atsakingas ir už aprašytos sistemos vykdymą.

Tai pagrindinė sistemos dalis, šios dalies klasių diagrama pateikiama 3.4 paveiksle.



3.4 pav. Objektinio modelio klasių diagrama

3.15 lentelė

Paketo „Objektinis modelis" aprašymas

	
Klasifikacija	Posistemė
Apibrėžimas	Objektinis modelis skirtas saugoti formalų sistemos aprašą, sutransliuotą į kalbą, kurią supranta ir gali naudoti sistema (PLA-CA).

Atsakomybės	Posistemės pagrindinis panaudojimo tikslas yra saugoti formalų sistemos aprašą, kurį gali naudoti visos kitos sistemos posistemės. Sistemai aprašyti naudojama PLA-CA formalizavimo kalba. Posistemė saugo analizuojamos sistemos komponentus – agregatus, jų įėjimus, išėjimus, sujungimus tarp agregatų, tolydžiuosius bei diskrečiuosius kintamuosius, H ir G operatorius, valdymo sekas, ir imitacinio modeliavimo duomenis.
Apribojimai	Objektinis modelis transliuoja tik griežtos struktūros sistemos aprašą. Kitokios struktūros sistemos aprašai nėra transliuojami į objektinį modelį. Sistemų aprašai saugomi *XML* formatu.
Struktūra	Posistemę sudaro 20 klasių: • *Aggregate* – klasė aprašanti agregatą; • *AggregateEvent* – abstrakti klasė aprašanti agregato įvykį; • *Connection* – klasė aprašanti sujungimą, kuris susideda iš įėjimo ir išėjimo taškų; • *ConnectionPoint* –klasė aprašanti sujungimą; • *ContinuousVariable* – klasė aprašanti tolydų kintamąjį; • *DiscreteVariable* – klasė aprašanti diskretų kintamąjį; • *Expression* – klasė aprašanti išraišką; • *ExternalEvent* – klasė aprašanti išorinį įvykį, paveldi klasę *AggregateEvent*; • *GOperator* – klasė aprašanti G operatorių, paveldi Operator klasę; • *HOperator* – klasė aprašanti H operatorių, paveldi Operator klasę; • *Input* – klasė aprašanti įėjimą, paveldi *ConnectionPoint* klasę; • *InternalEvent* – klasė aprašanti vidinį įvykį, paveldi klasę *AggregateEvent*; • *ModelResults* – klasė atsakinga už imitacinio modeliavimo rezultatų saugojimą, eksportavimą; • *Operator* – klasė aprašanti visus operatorius; • *Option* – klasė aprašanti išraiškos alternatyvą; • *Output* – klasė aprašanti išėjimą, paveldi *ConnectionPoint* klasę;

	• *RandomNumber* – klasė generuojanti atsitiktinius skaičius; • *SystemDescritpion* – sistemos aprašas.
Sąveikavimas	Posistemė sąveikauja su visomis kitomis posistemėmis – objektiniame modelyje saugomas sistemos aprašas yra visų skaičiavimų pagrindas.
Resursai	Posistemė nenaudoja jokių specialių resursų.
Sąsaja/eksportas	Posistemė yra visų sistemos skaičiavimų pagrindas, todėl ją naudoja visos kitos posistemės. Posistemė naudojama turint klasės *SystemDescription* objektą.

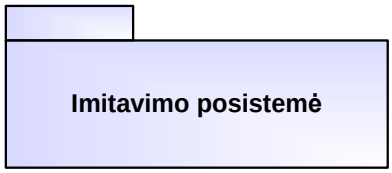
3.3.2.2. Imitacinio modeliavimo posistemė

Imitavimo posistemė skirta atlikti imitacinį modeliavimą bei pateikti vykdymo rezultatus. Posistemė leidžia atlikti formalaus aprašo veikimo analizę, bei gauti vykdymo rezultatus. Tai leidžia gauti tuos sistemos veikimo rezultatus, kurie domina sistemos aprašo sudarytojus. Posistemės, atsakingos už imitacinį modeliavimą, klasių diagrama pateikiama 3.5 paveiksle.

3.5 pav. Imitavimo posistemės klasių diagrama

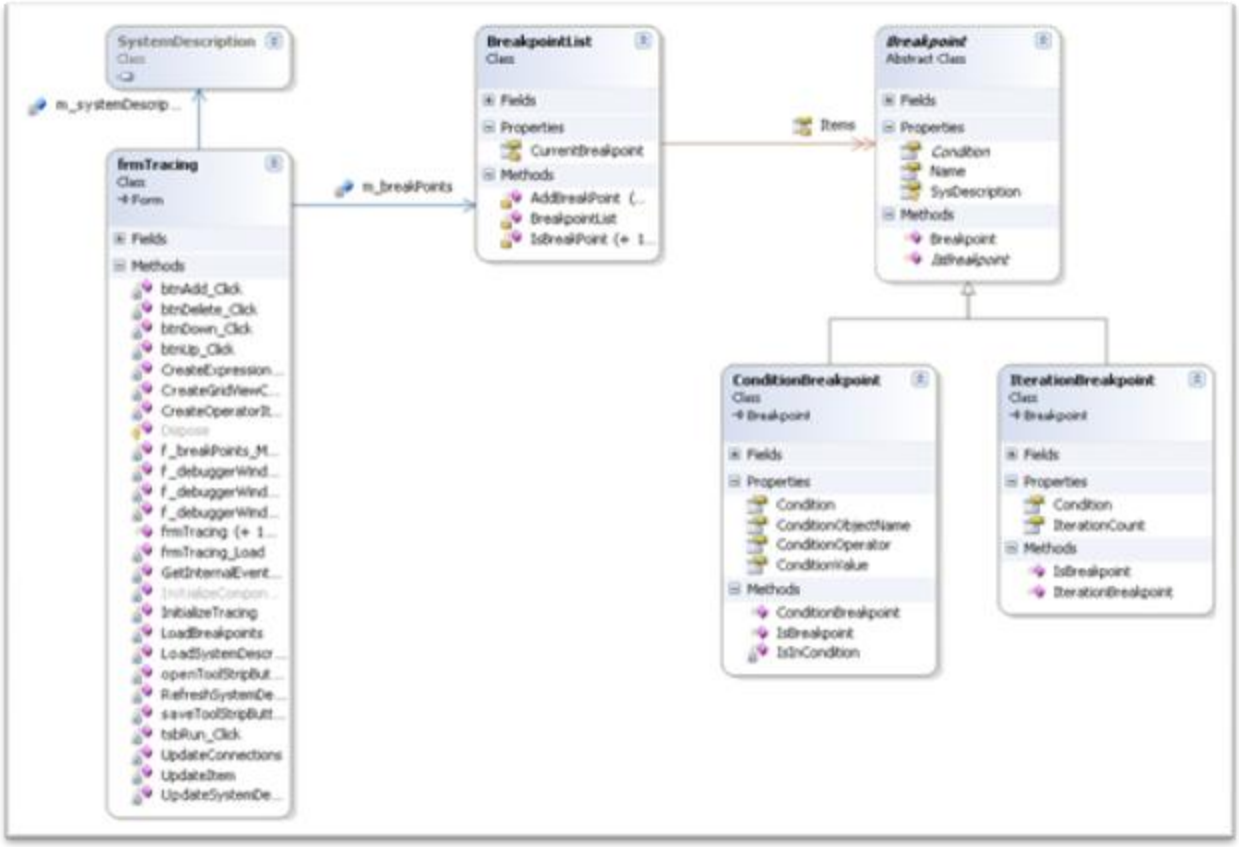
3.16 lentelė

Paketo „Imitavimo posistemė“ aprašymas

	
Klasifikacija	Posistemė
Apibrėžimas	Imitavimo posistemė skirta analizuojamos sistemos atlikti imitacinį modeliavimą ir pateikti rezultatus.
Atsakomybės	Posistemės pagrindinis panaudojimo tikslas yra atlikti analizuojamos sistemos imitacinį modeliavimą, gauti modeliavimo rezultatus ir reikalui esant, rezultatus išsaugoti faile. Imitavimo posistemė yra viena svarbiausių posistemių sistemoje, nes jos veikimu paremtos trasavimo ir validavimo posistemės.
Apribojimai	Imitavimo posistemė yra glaudžiai susijusi su objektinio modelio posisteme, todėl imitavimą galima atlikti tik tokios sistemos, kurią galima konvertuoti į objektinį modelį.
Struktūra	Posistemę sudaro 2 pagrindinės klasės: • *ImitationForm* – grafinės vartotojo sąsajos forma, pagrindinis langas. • *ResultsCtrl* – grafinis komponentas, atsakingas už rezultatų atvaizdavimą. Už pati modeliavimą, atsakinga objektinio modelio posistemė.
Sąveikavimas	Posistemė sąveikauja su objektinio modelio, validavimo ir trasavimo posistemėmis. Tiek trasavimo, tiek validavimo posistemių veikimas yra paremtas trasavimo posistemės *Execution* klasės veikimu. Objektinio modelio posistemė saugo visus reikalingus duomenis sistemos veikimo imitavimui atlikti.
Resursai	Posistemė nenaudoja jokių specialių resursų.
Sąsaja/eksportas	Posistemė bendrauja su kitomis posistemės per pagrindinę klasę *ImitationForm*.

3.3.2.3. Trasavimo posistemė

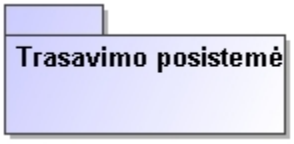
Trasavimo posistemė skirta analizuojamo formalaus aprašo trasavimui. Posistemės dėka gali būti tikrinamas aprašo teisingumas ir stebima analizuojamos sistemos būsena konkrečiu laiko momentu. Trasavimo posistemė yra ypač naudinga formalios specifikacijos sudarymo metu, jos pagalba greičiau randamos ir lokalizuojamos sistemos formalaus aprašo klaidos. Posistemė leidžia imituoti tiriamos sistemos darbą vykdant specifikacijoje aprašytas operacijas ir perėjimus tarp būsenų pažingsniui, su galimybe keisti sistemos būseną vykdymo metu ir vykdymo tvarką. Trasavimo posistemės klasių diagrama pateikiama 3.6 paveiksle.



3.6 pav. Trasavimo posistemės klasių diagrama

3.17 lentelė

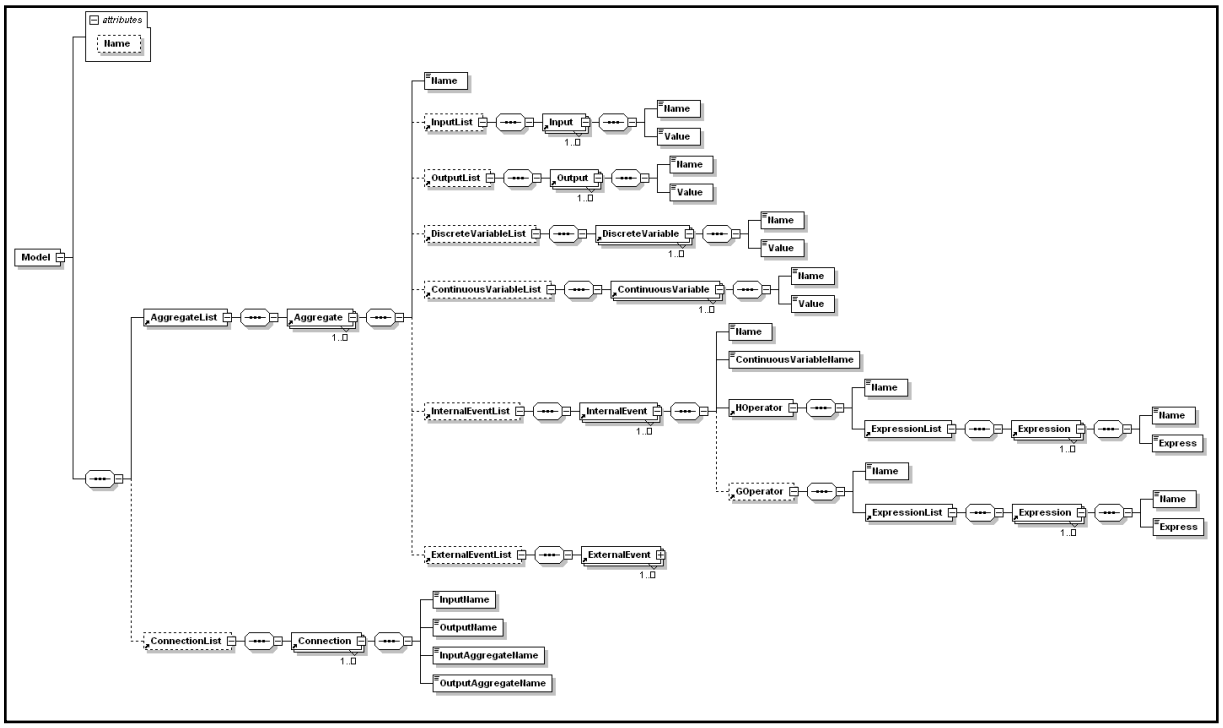
Paketo „Trasavimo posistemė“ aprašymas

	
Klasifikacija	Paketas
Apibrėžimas	Trasavimo posistemė skirta analizuojamo formalaus aprašo trasavimui.
Atsakomybės	Posistemės dėka gali būti tikrinamas aprašo teisingumas ir stebimas

	analizuojamos sistemos būsena konkrečiu laiko momentu. Trasavimo posistemė yra ypač naudinga formalios specifikacijos sudarymo metu, jos pagalba greičiau randamos ir lokalizuojamos sistemos formalaus aprašo klaidos. Posistemė leidžia imituoti tiriamos sistemos darbą vykdant specifikacijoje aprašytas operacijas ir perėjimus tarp būsenų pažingsniui, su galimybe keisti sistemos būseną vykdymo metu ir keisti įvykių vykdymo eiliškumą.
Apribojimai	Posistemei reikalingi vykdymo parametrai, kurie arba nustatomi arba parenkami pagal nutylėjimą. Posistemės darbas gali būti baigtas pasiekus būseną, kuri tenkina bent vieną sustojimo sąlygą ir vartotojui nebetęsiant trasavimo.
Struktūra	Paketą sudaro 5 pagrindinės klasės: • *frmTracing* – vartotojo grafinė sąsaja ir loginė trasavimo dalis; • *BreakPointList* – sustojimo sąlygų sąrašas; • *BreakPoint* – abstrakti klasė sustojimo sąlygai apibrėžti; • *ConditionBreakPoint* – klasė aprašo sustojimo sąlygą, kuri yra tenkinama, kai modelio parametras tenkina užduotąją sąlygą, paveldi *BreakPoint* klasę. • *IterationBreakPoint* – klasė aprašo sustojimo sąlygą, kuri yra tenkinama, kai įvyksta nustatytas skaičius vidinių įvykių, paveldi *BreakPoint* klasę.
Sąveikavimas	Trasavimo posistemė naudoja imitavimo ir objektinio modelio posistemes. Imitavimo posistemės pagalba bus vykdomas aprašytos sistemos imitacinio modelio vykdymas. Objektinis modelis tai duomenų struktūra, kuri naudojama visose posistemėse ir bendravimui tarp atskirų posistemių. Ši posistemė naudojama agregato būsenai gauti/keisti ir pačios imitavimo posistemės darbui kaip įėjimo duomenys.
Resursai	Posistemės duomenys gaunami iš objektinio modelio. Jie gali būti keičiami.
Skaičiavimai	Pagal pasirinktus parametrus vykdomas sistemos aprašo imitavimas. Vykdymas sustoja nurodytose vietose, kad vartotojas galėtų analizuoti esamą modelio būseną
Sąsaja/eksportas	Sąsajai naudojama tik *TracingForm* klasė.

3.4. Duomenų vaizdas

Kuriama sistema naudos labai daug pradinių duomenų (priklausomai nuo nagrinėjamos sistemos). Dėl specifinės duomenų paskirties ir struktūros duomenims saugoti nebus naudojama reliacinių duomenų bazių valdymo sistema (RDBVS). Nors ateityje galima būtų realizuoti tokio tipo sprendimą. Tai pagreitintų sistemos interpretatoriaus darbą, tačiau tik tuo atveju jei duomenų kiekis būtų labai didelis (virš 1GB), o tai visgi pasitaiko labai retai. Todėl atsižvelgiant į duomenų specifiką, sistemos duomenis patogiausia saugoti XML formato faile. Dėl savo universalumo XML leis paprastai ir patogiai išsaugoti sistemos aprašą, kurį bus lengva sukurti ar redaguoti be papildomų priemonių. Kadangi XML formatas yra nepriklausomas nuo technologijos ir griežtos struktūros, sistemos duomenys yra tinkami naudoti kitų programų ar papildomų modulių. XML specifikacijos dokumento struktūros schema pateikiama 3.7 paveiksle.



3.7 pav. XML specifikacijos dokumento schema

Pavyzdiniai duomenys

```
<?xml version="1.0" encoding="UTF-8"?>
<Model Name="Tumbleris" xmlns="http://tempuri.org/ObjectModel.xsd">
  <AggregateList>
    <Aggregate>
      <Name>Tumbleris</Name>
      <DiscreteVariableList>
        <DiscreteVariable>
          <Name>būsena</Name>
          <Value>0</Value>
        </DiscreteVariable>
      </DiscreteVariableList>
```

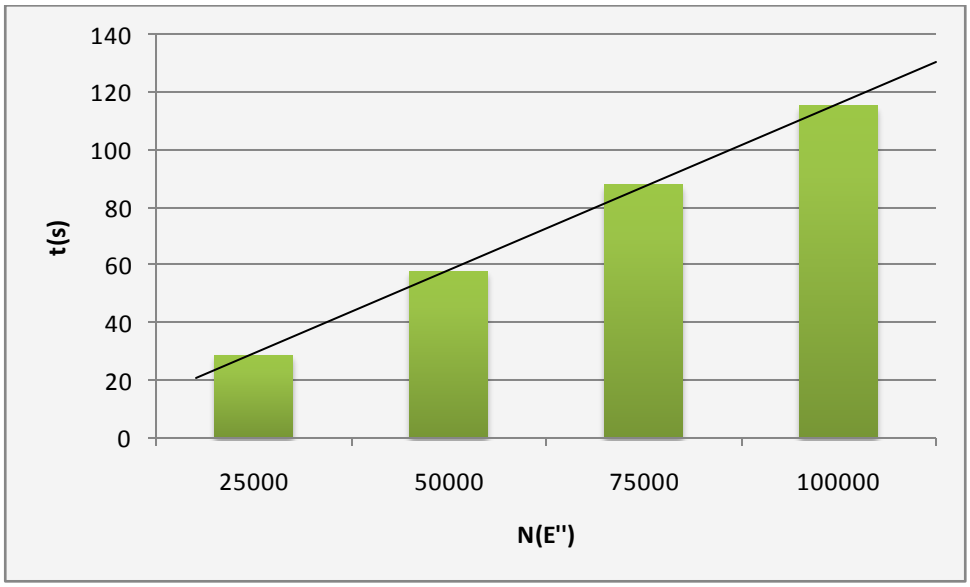
```
      <ContinuousVariableList>
        <ContinuousVariable>
          <Name>laikas</Name>
          <Value>0</Value>
        </ContinuousVariable>
      </ContinuousVariableList>
      <InternalEventList>
        <InternalEvent>
          <Name>Jungti</Name>
          <ContinuousVariableName>laikas</ContinuousVariableName>
          <HOperator>
            <Name>H_jungti</Name>
            <ExpressionList>
              <Expression>
                <Name>Jungti</Name>
                <Express>būsena := [1 WHEN (būsena = 0); 0 WHEN (būsena = 1)]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
        </InternalEvent>
      </InternalEventList>
    </Aggregate>
  </AggregateList>
</Model>
```

4. Tyrimo dalis

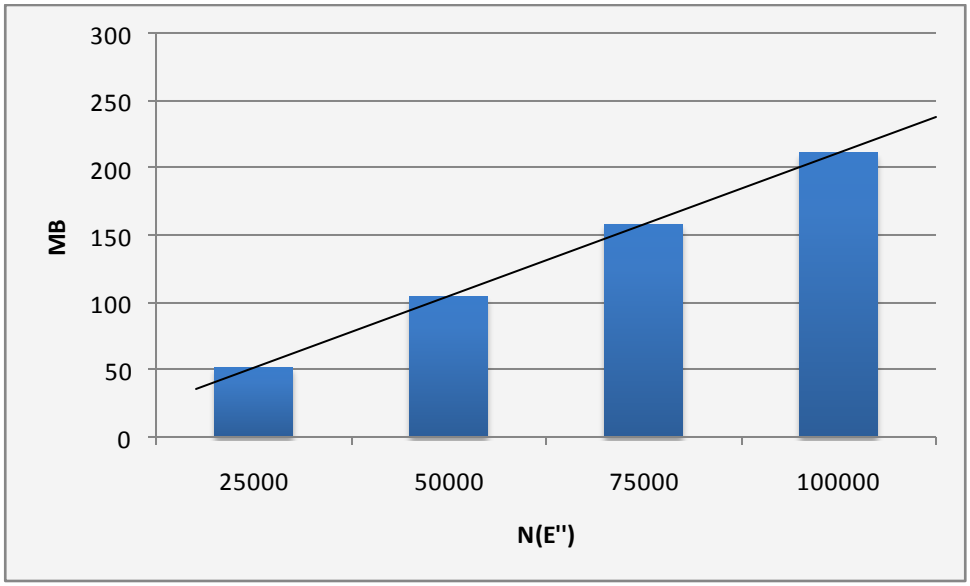
4.1. Imitacinio modeliavimo efektyvumo tyrimas

Imitacinio modeliavimo efektyvumui analizuoti visiems bandymams buvo naudota viena formali PLA specifikacija. Buvo vykdomas specifikacijoje aprašytos sistemos imitacinis modeliavimas. Modelis buvo vykdomas ir stebimas vykdymo laikas ir gautų rezultatų kiekis. Šie parametrai buvo matuojami po nurodyto skaičiaus vidinių įvykių.

Iš tyrimo metu surinktų vykdymo rezultatų matome, kad vykdymo laikas (žr. 4.1 pav.) ir duomenų kiekis (žr. 4.2 pav.) tiesiogiai proporcingi įvykdytam vidinių įvykių skaičiui.



4.1 pav. Vykdymo laiko kitimas



4.2 pav. Duomenų kiekio kitimas

Kad įrodyti, jog naudojamų resursų priklausomybė nuo įvykių skaičiaus yra tiesinė, rasime regresijos tiesės lygtį. Tiesės lygtis yra $y = a + bx$, kur $b = \frac{\sum xy - \frac{\sum x \sum y}{n}}{\sum x^2 - \frac{(\sum x)^2}{n}}$, $a = \overline{y} - b\overline{x}$. Čia n – imties dydis, $\overline{y}, \overline{x}$ - vidurkiai. Naudojantis šiomis formulėmis, gauname regresijos tiesių lygtis. Laiko matavimų tiesės lygtis yra

$$y = 0{,}64 + 0{,}00115848 \cdot x,$$

o duomenų kiekio

$$y = 0{,}0021312 \cdot x - 4{,}4.$$

Turėdami regresijos tiesių lygtis, galime įvertinti, kiek laiko truks nagrinėjamos sistemos imitacinis modeliavimas ir koks duomenų kiekis bus gautas. Šios lygtis tinka tik šio eksperimento metu naudotai sistemai, analizuojant kitos sistemos specifikaciją tiesių a ir b koeficientai skirsis. Įvertinę rezultatus galime teigti, kad imitavimo laikas ir rezultatų kiekis yra priimtini, atsižvelgiant į imitavimo apimtį. Daugeliui modelių pakanka gerokai mažesnio vidinių įvykių įvykdymo skaičiaus.

4.2. Imitacinio modeliavimo ir trasavimo posistemių kokybinis įvertinimas

Išplečiamumas. Sistema sudaryta iš atskirų modulių, kurių didžioji dalis yra nepriklausomi vieni nuo kitų. Ši sistemos architektūros savybė leis tęsti projektą nereikalaujant didelių papildomų sąnaudų esamos sistemos funkcionalumo derinimui su naujuoju. Esamus modulius galima panaudoti ir plėsti naujo funkcionalumo sukūrimui, pasikeitus ar atsiradus naujiems reikalavimams. Pagrindinės sistemos dalys, kurios yra labai silpnai tarpusavyje surištos, duomenų apsikeitimui ir saugojimui naudoja XML formatą. XML dėka šios dalys tampa visai nepriklausomos nuo realizacijos ypatybių ar darbo aplinkos apribojimų. Ateityje šios dalys gali būti išplečiamos ar kuriamos alternatyvos visai nepriklausomai vienai nuo kitos.

Pernešamumas. Kadangi sistema numatyta darbui viename personaliniame kompiuteryje, sistemos pernešamumas yra itin aukšto kokybės lygio. Tai gali būti programos archyvas ar įdiegimo paketas. Duomenų pernešamumas užtikrinimas bendru XML formatu. XML formatas buvo pasirinktas dėl standartizuotos struktūros ir palaikomumo tarp įvairių sistemų ir technologijų.

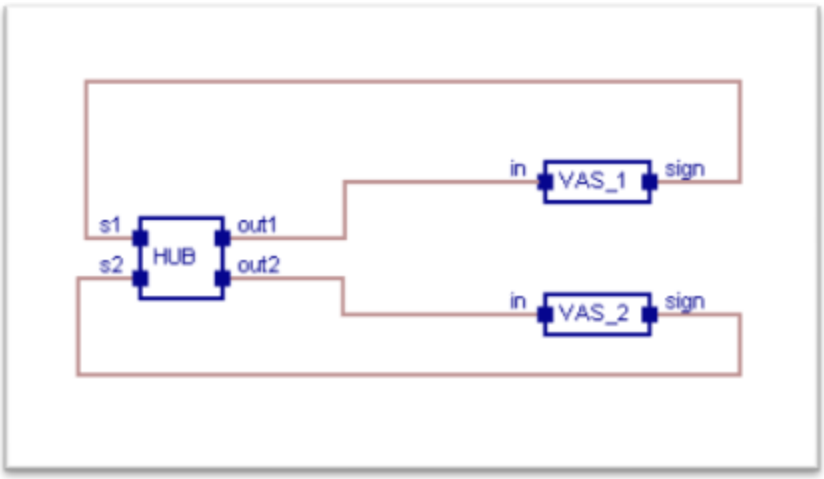
Patikimumas. Pagrindinė sistemos dalis yra objektinis modelis, kurį naudoja visos likusios sistemos. Ši posistemė yra kritinė ir turi būti itin gerai ištestuota. Likusios sistemos

dalys yra tarpusavyje nesusietos, todėl visos sistemos patikimumas nepriklausys nuo atskirų dalių. Kadangi sistemos architektūra sudaryta iš atskirų ir kiek įmanoma savarankiškų komponentų, sistemos patikimumas (sistemos architektūros kontekste) yra apsprendžiamas pagrindinių dalių patikimumo įvertinimu.

Palaikomumas. Programos palaikomumas nereikalaus didelių pastangų. Pirmiausia dėl to, kad įvykus klaidai nebus sugadinami duomenys, kadangi operuojama tik atmintyje esančiais duomenimis, o duomenys saugomi išorėje. Sistema nereikalauja ypatingų aplinkos sąlygų, todėl pasikeitusi aplinka (reikalavimuose specifikuotų apribojimų ribose) negali iššaukti sistemos klaidingo veikimo.

5. Eksperimentinė dalis

Eksperimentui atlikti naudojama tinklo įrenginių darbo analizė. Aprašomą sistemą sudaro trys agregatai: HUB, VAS_1 ir VAS_2. Sistemos schema pateikta 5.1 paveiksle.



5.1 pav. Tinklo įrenginių agregatinės specifikacijos schema

Schemoje stačiakampiai vaizduoja agregatus, taškai ant agregatų kraštinių vaizduoja sujungimo taškus, kurie gali būti išėjimai ir įėjimai. Linijos vaizduoja sujungimus tarp agregatų, kuriais perduodami signalai. Sujungimas jungia du taškus, vienas jų yra įėjimo, kitas išėjimo taškas. Stačiakampio viduje nurodytas vardas yra agregato pavadinimas, o šalia sujungimo taškų esantys užrašai žymi įėjimų arba išėjimų pavadinimus.

Agregatas HUB – tai šakotuvo tipo įrenginys. Jis skirsto paraiškas aptarnaujantiems tinklo įrenginiams. Paraiška pirmiausiai yra siunčiama VAS_1 įrenginiui, jei šis įrenginys dirba ir jo buferis jau pilnas, paraiška siunčiama kitam įrenginiui VAS_2. Jei VAS_2 įrenginys taip dirba ir jo buferis pilnas, paraiška liks neaptarnauta, priešingu atveju paraiška bus įrašyta į buferį ir kai aptarnaujantis įrenginys bus pasiruošęs, ją aptarnaus.

Agregatai VAS_1 ir VAS_2 – aptarnaujantieji įrenginiai. Šie įrenginiai turi riboto dydžio buferį, o jų paraiškų aptarnavimo laikas yra atsitiktinis dydis. Jei VAS aptarnauja paraišką, ir tuo metu ateina dar viena paraiška, pastaroji įrašoma į buferį. Kai buferis pasiekia maksimalų dydį, HUB įrenginiui siunčiamas signalas, kad pastarasis nebesiųstų paraiškų. Agregatinė specifikacija pateikta 5.1 skyrelyje, o agregatinės specifikacijos XML failo turinys pateikiamas 9.1 1.Priedas. Eksperimento „VAS_HUB“ formali specifikacija.

5.1. Agregatinė specifikacija

Agregatas „HUB“

1. **Įėjimo signalų aibė** *X = {s1, s2}*, pradinės reikšmės: *s1 = 0, s2 = 0.*

Jei atėjo signalas į *s1* įėjimą, reiškia VAS_1 agregato buferis pilnas ir jis nebegali priimti paraiškų, taip pat su *s2*.

2. **Išėjimo signalų aibė** *Y = {out1, out2}*, pradinės reikšmės *out1 = 0, out2 = 0*.
3. **Diskretieji kintamieji** *{state1, state2}*, pradinės reikšmės *state1 = 0, state2 = 0*.

Diskrečių kintamųjų reikšmės rodo kuris VAS gali priimti paraišką. Jei *state1=1*, reiškia VAS_1 paraiškos priimti negali.

4. **Tolydieji kintamieji** *{par_gen}*, pradinė reikšmė *par_gen = 0,1*.

Tolydusis kintamasis *par_gen* nurodo laiką, kad bus generuojama nauja paraiška.

5. **Vidiniai įvykiai** *{gen_par}*

Vidinis įvykis *gen_par* generuoja naują paraišką ir siunčia ją VAS įrenginiui.

H(gen_par):

par_gen := [exprnd]

state1 := [state1]

state2:= [state2]

G(gen_par):

$$out1 := \begin{cases} 1, & state1 = 0 \\ 0, & state1 = 1 \end{cases}$$

$$out2 := \begin{cases} 1, & state1 \neq 0 \ AND \ state2 = 0 \\ 0, & state2 = 1 \ AND \ state1 = 0 \end{cases}$$

6. **Išoriniai įvykiai** *{atejo_s1, atejo_s2}*

Įvykis *atejo_s1* įvyksta, kai ateina signalas į *s1* įėjimą (VAS_1 įrenginio buferis persipildė), o įvykis *atejo_s2*, kai ateina signalas į *s2* įėjimą.

H(atejo_s1):

state1 := [s1]

H(atejo_s2):

state2 := [s2]

Agregatas „VAS_1“

1. **Įėjimo signalų aibė** *X = {in}*, pradinė reikšmė: *in = 0*.
2. **Išėjimo signalų aibė** *Y = {sign}*, pradinė reikšmė *sign = 0*.
3. **Diskretieji kintamieji** *{buf, buf_max, dirba}*, pradinės reikšmės *buf = 0,buf_max = 3, dirba = 0*.

Diskretus kintamasis *buf* rodo kiek paraiškų laukia buferyje, *buf_max* – buferio talpa, *dirba* – nurodo, kad įrenginys aptarnauja paraišką.

4. **Tolydieji kintamieji** *{par_apt}*, pradinė reikšmė *par_apt = 500.*

Tolydusis kintamasis *par_apt* rodo paraiškos aptarnavimo laiką.

5. **Vidiniai įvykiai** *{apt_par}*

Vidinis įvykis *apt_par* – paraiškos aptarnavimas.

H(apt_par):

$$par\_apt := \begin{cases} [exprnd], buf \neq 0 \\ 1000, \ buf = 0 \end{cases}$$

$$dirba := \begin{cases} 0, buf = 0 \\ 1, \ buf \neq 0 \end{cases}$$

$$buf := \begin{cases} buf, buf = 0 \\ buf - 1, \ buf \neq 0 \end{cases}$$

G(apt_par):

$$sign := \begin{cases} 0, buf \neq buf\_max \\ 1, \ buf = buf\_max \end{cases}$$

6. **Išoriniai įvykiai** *{atejo_par}*

Išorinis įvykis *atejo_par* – atėjo paraiška.

H(atejo_ par):

$$par\_apt := \begin{cases} [exprnd], buf = 0 \ AND \ sign = 0 \ AND \ dirba = 0 \ AND \ in = 1 \\ par\_apt, \ buf \neq 0 \ OR \ sign = 1 \ OR \ dirba = 1 \ OR \ in = 0 \end{cases}$$

$$buf := \begin{cases} buf, buf = buf\_max \ OR \ in = 0 \ OR \ dirba = 0 \ OR \ sign = 1 \\ buf + 1, \quad buf \neq buf\_max \ AND \ in = 1 \ AND \ dirba = 1 \ AND \ sign = 0 \end{cases}$$

$$dirba := \begin{cases} dirba, \ in = 0 \\ 1, \ in = 1 \end{cases}$$

G(atejo_ par):

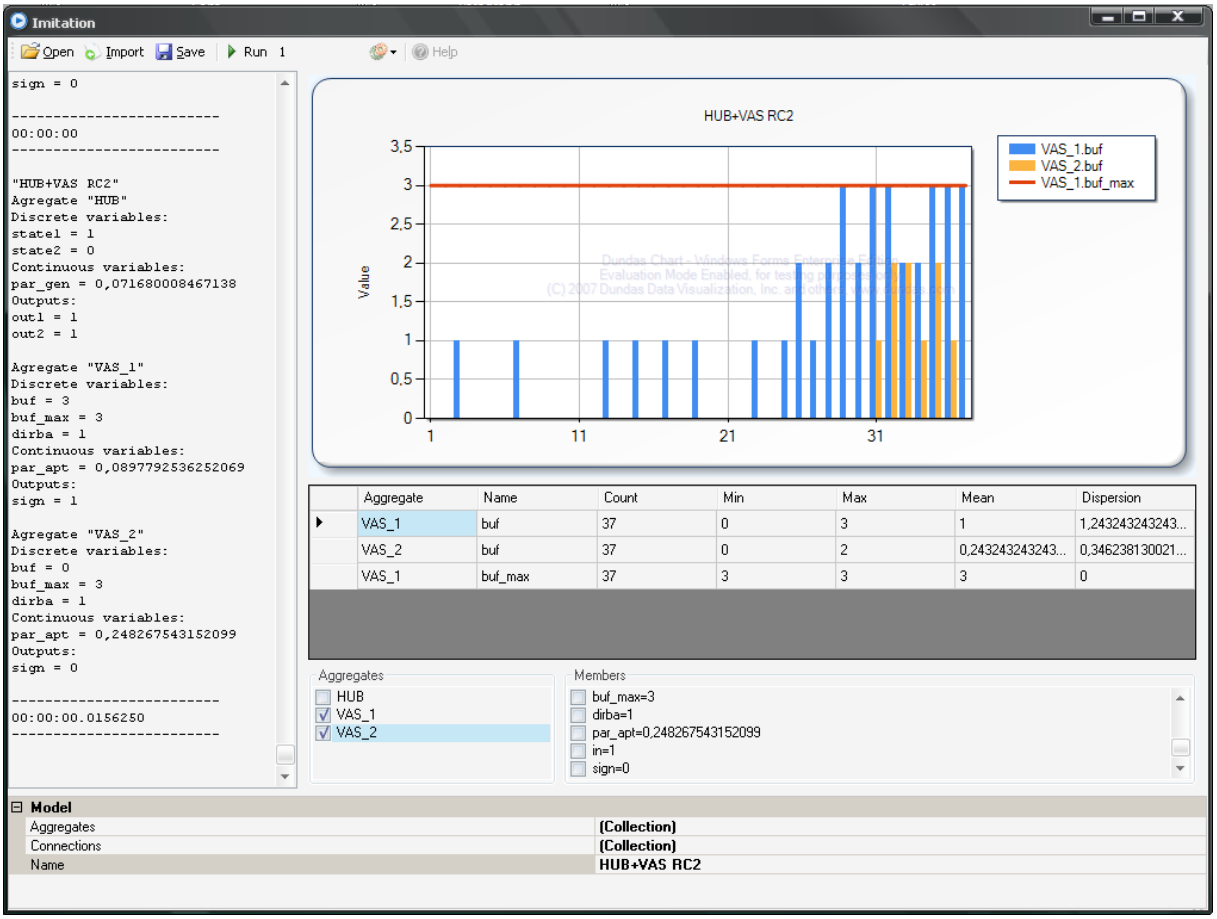
$$sign := \begin{cases} 1, sign = 0 \ AND \ in = 1 \ AND \ buf = buf\_max \\ sign, \ sign \neq 0 \ OR \ in = 0 \ OR \ buf \neq buf\_max \end{cases}$$

Agregatas „VAS_2“

Agregato VAS_2 aprašas identiškas VAS_1 aprašui.

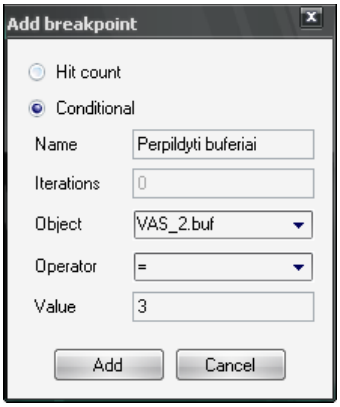
5.2. Uždavinio sprendimas naudojant FSA įrankius

Atliekant aprašytos sistemos imitacinį modeliavimą gaunami rezultatai, kurie pateikiami ekrane. Rezultatai pateikiami laiko funkcijomis, matomas tiriamos sistemos būsenos kitimas diskrečiais laiko momentais.



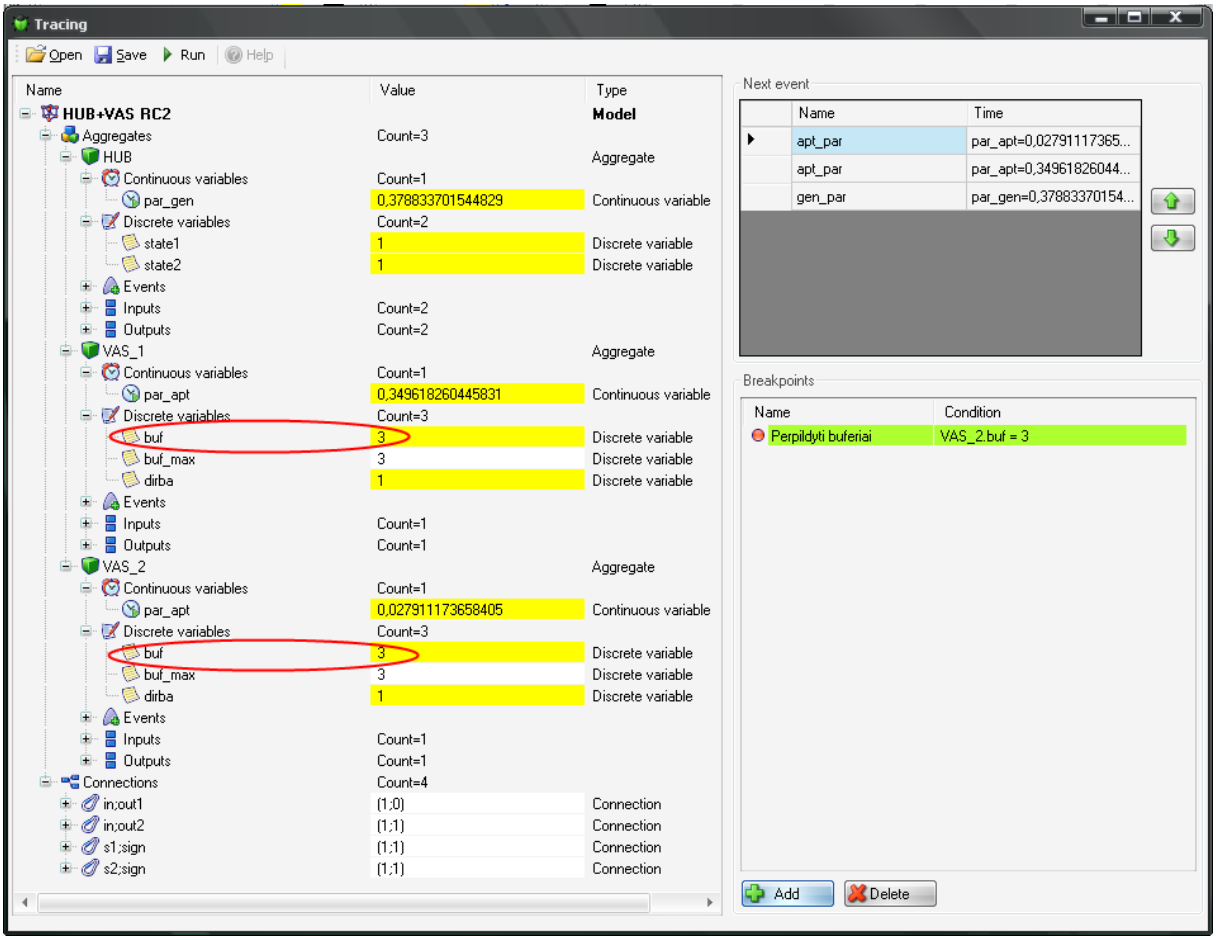
5.2 pav. Imitacinio modeliavimo rezultatai

Paveiksle 5.2 matomas grafinės vartotojo sąsajos langas, jame pateikti modeliavimo rezultatai. Iš grafiko matome, kad aprašyta sistema funkcionuoja teisingai: kai persipildo VAS_1 įrenginio buferis, užklausos yra siunčiamos VAS_2 įrenginiui. Sekančiame žingsnyje naudojantis trasavimo posisteme patikrinsime ar imitacinio modeliavimo metu atsitinka taip, kad paraiška negali būti aptarnaujama ir kokia sistemos būsena yra tuo laiko momentu. Kad paraiška nebūtų aptarnaujama įrenginių VAS_1 ir VAS_2 buferiai turi būti persipildę, t.y. sistemos būsena turi tenkinti sąlygą VAS_2.buf= VAS_2.buf_max. Tam tikslui sukursime sustojimo sąlygą (žr. 5.3 pav.).



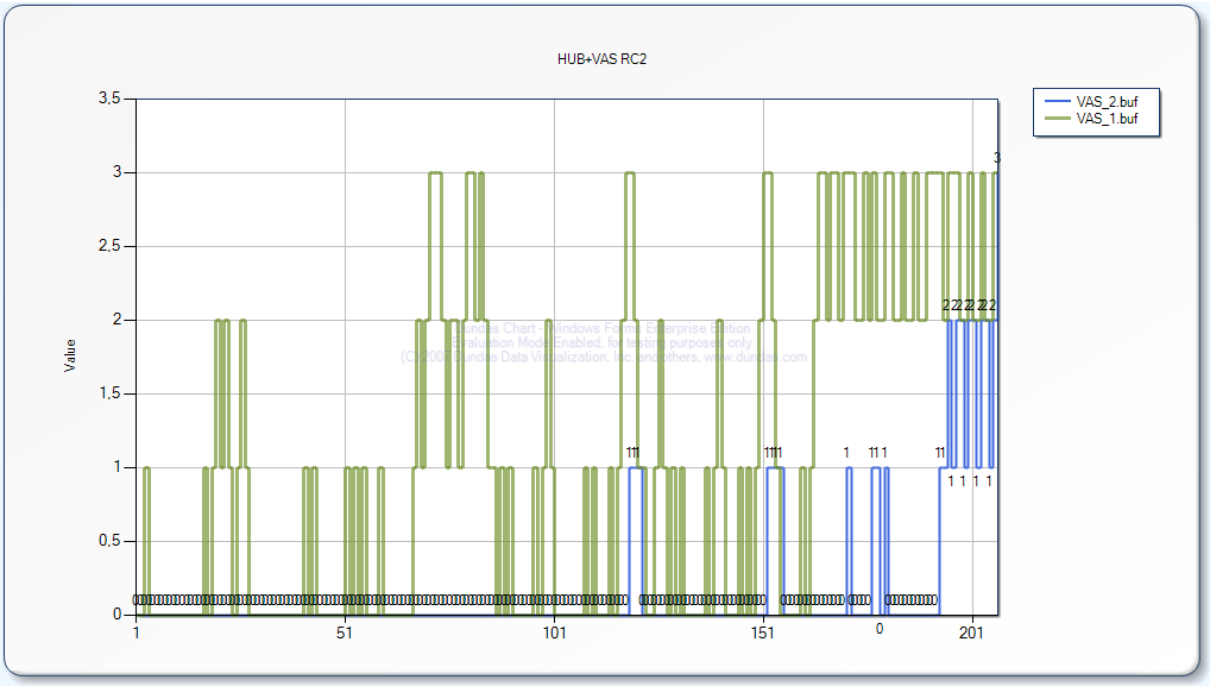
5.3 pav. Nauja sustojimo sąlyga

Imitacinis modeliavimas bus vykdomas kol modelio būsena tenkins sustojimo sąlygą. Sustojus vykdymui, matoma esama modelio būsena (žr. 5.4 pav.). Būsena gali būti keičiama ir modeliavimas tęsiamas. Kol modeliavimas yra sustabdytas galima įvesti naujas sustojimo sąlygas ir pašalinti esančias, taip pat galima keisti būsimo vykdymo eiliškumą.



5.4 pav. Trasavimo posistemės grafinė vartotojo sąsaja

Imitacinis modeliavimas buvo vykdomas, kol persipildė VAS_1 ir VAS_2 įrenginių buferiai. Šio imitacinio modeliavimo rezultatai pateikti 5.5 paveiksle. Grafike matomos įrenginių buferyje esančių paraiškų kiekio kitimo funkcijos. Y ašyje yra paraiškų kiekis, o X ašyje diskretūs laiko momentai.



5.5 pav. Sistemos kintamųjų kitimo grafikas

6. Išvados

- Magistrinio darbo tezėse apžvelgti egzistuojantys imitacinio modeliavimo algoritmai ir metodai.
- Realizuotas imitacinio modeliavimo algoritmas PLA formaliosioms specifikacijoms, remiantis diskrečių įvykių imitacinio modeliavimo metodais. Sukurtas algoritmas panaudotas formalių specifikacijų integruotos analizės automatizavimo sistemos (FSA) imitacinio modeliavimo ir trasavimo posistemių realizacijai.
- Sukurtos FSA sistemos imitacinio modeliavimo ir trasavimo posistemės, leidžiančios vieningos formaliosios specifikacijos bazėje atlikti imitacinio modeliavimo eksperimentus su sistemos formalia specifikacija. Sukurtos posistemės integruotos į formalių specifikacijų integruotos analizės automatizavimo sistemos (FSA) bendrą kontekstą. FSA – tai grupinis projektas, realizuotas šio magistrinio darbo metu.
- Atliktas eksperimentas, naudojant sukurtus įrankius. Eksperimento metu buvo ištirta tinklo įrenginių darbą aprašanti sistema, pademonstruojanti sukurto įrankio galimybes. Remiantis eksperimento rezultatais, galima teigti, kad FSA įrankių dėka atliekama pilna formaliai aprašytos sistemos analizė.
- Tyrimo metu buvo įvertintos imitacinio modeliavimo posistemės charakteristikos. Nustatyta, kad posistemė gali atlikti išsamų imitacinį modeliavimą, nereikalaujant ypatingai didelių kompiuterio resursų. Nustatyta, kad naudojamų resursų (procesoriaus laiko ir atminties kiekio) augimas yra tiesinė funkcija, priklausanti nuo modeliavimo pilnumo (įvykių skaičiaus).
- Lyginant sukurtos sistemos (FSA) funkcionalumą su iki šiol sukurtais analogais, tai vienintelė sistema, kuri apima įvedimo, verifikavimo, imitavimo ir trasavimo galimybes. Tai yra pagrindiniai analizės metodai, reikalingi pilnai formalaus modelio analizei atlikti.

6.1. Tolimesni darbai

Artimiausi planai – realizuoti dinaminio PLA modelio (dynPLA) palaikymą. Dinaminis PLA modelis tai PLA papildymas ir pritaikymas dinaminėms sistemoms, kurios keičia savo struktūrą ir/arba elgseną laike [3]. Dinaminis PLA įgalintų modeliuoti sistemas, gebančias evoliucionuoti laike, o tai atvertų plačias PLA taikymo galimybes. Kadangi esamam sprendime yra galimybės keisti sistemos būseną ir dalinai keisti struktūrą reikėtų papildyti PLA modelį ir pritaikyti imitacinį modeliavimą papildytam PLA modeliui. Pakeitus PLA modelį, keistųsi sistemos formaliojo aprašo struktūra, jo interpretatorius ir kitos FSA dalys.

Ateityje planuojama realizuoti imitacinių modelių rezultatu kaupimą duomenų bazių valdymo sistemoje (DBVS). Tai įgalintų išsaugoti visų modeliavimų rezultatus, juos panaudoti tolimesnėje analizėje, taip pat panaudoti DBVS įrankius duomenų apdorojimui ir ataskaitų generavimui. Tokiu būdu šie duomenys būtų prieinami grupės specialistų, užsakovų ir pan. Tai padėtų dalintis duomenimis, atlikti skirtingas užduotis, pateikti ataskaitas užsakovams.

Siekiant, kad sukurtas įrankis ir PLA formalizmas būtų plačiai naudojamas ir žinomas, reikalinga sukurti ir realizuoti modelių transformacijas iš ir į PLA modelį. Tokiu atveju bus galima panaudoti įrankius, kurie yra realizuoti kitiems modeliams ir kitus modelius transformavus į PLA, jų analizei naudoti FSA funkcionalumą.

FSA naudojamas specifikacijos formatas yra naujas, pritaikytas ir optimizuotas šiai sistemai. Šis naujas formatas nebus suprantamas kitose PLA automatizavimo sistemose kaip ValSys ir atvirkščiai. Todėl reikia realizuoti importo/eksporto funkcionalumą, kad sistemos galėtų dirbti su bendra specifikacija.

7. Literatūra

[1] Pranevičius H., *Kompiuterinių tinklų protokolų formalusis specifikavimas ir analizė: agregatinis metodas.* Monografija. Kaunas: Technologija, 2005.

[2] Pranevičius H., *Creation of simulation models of complex systems using piece-linear aggregate formalism.* 2006 International Conference on modeling, Simulation & Visualization Methods /MSV'06/. Las Vegas, Nevada, USA, 2006.

[3] Packevičius Š., Kazla A., Pranevičius H., *Extension of PLA specification for dynamic system formalization*. Information Technology and Control, 2006, Vol.35, No.3, ISSN 1392 – 124X.

[4] Holzmann G. J., *The Spin Model Checker, Primer and Reference Manual.* Addison Wesley, 2003, ISBN 0-321-22862-6.

[5] Mueller R., *Specifications and automatic generation of simulation models with applications in semiconductor manufacturing.* Daktaro disertacijos tezės. Georgia Institute of Technology, 2007.

[6] Bakanas A., *Agregatinių specifikacijų interaktyvusis redagavimas ir imitacinis modeliavimas.* Magistro darbas. KTU, 2005.

[7] Pranevičius H., *Integrated analysis of communication protocols by means of PLA formalism*. Journal of Telecomunications and Information Technology, 2004.04

[8] Pranevičius H., Pranevičienė I., Benkunskis R., *Formalization and Simulation Telecommunication Protocols and Systems using PLA Method*. Elektronika ir elektrotechnika, 2005, Nr. 4(60) ISSN 1392-1215.

[9] Pranevičius H., Makackas D., *Simiulation of stevedoring work in the Klaipėda Oil Terminal.* Transport, 2002, Vol XVII, No 5, 188-193.

[10] Pranevičius H., Pilkauskas V., Chmieliauskas A. *Aggregate approach for specification and analysis of computer network protocols*. Kaunas: Technologija, 1994.

[11] Pranevičius H., Knight J. A. G., Pranevičienė I. *Formal Specification and Analysis of Distributed Systems Used Aggregate Approach*. Information Technology and Control. No.22(1).

[12] Lamsweerde A. *Formal specification: a Roadmap*. 2000: International Conference on Software Engineering p.147-159. Limerick, Ireland 2000.

[13] Bengtsson J., Yi, W. *Timed Automata: Semantics, Algorithms and Tools.* Springer-Verlag, 2004

[14] *Uppaal* [interaktyvus] [žiūrėta 2008-03-14]. Prieiga internete <http://www.uppaal.com/>.

[15] Banks J., *Handbook of simulation: principles, methodology, advances, applications, and practice*. New York, NY: John Wiley & Sons, 1998

[16] Zeigler B. P., Kim, T. G., and Praehofer, H., *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*. San Diego, CA: Academic Press, 2nd ed., 2000

[17] „ON-THE-FLY, LTL MODEL CHECKING with SPIN“[interaktyvus] [žiūrėta 2008-03-14]. Prieiga internete <http://spinroot.com/spin/whatispin.html>.

[18] Ruys T. C. *SPIN Beginners' Tutorial*. SPIN 2002 Workshop, Grenoble, 2002.

[19] Ruys T. C., Holzmann G. J. *Advanced SPIN Tutorial*. SPIN 2004, Barcelona, Spain, 2004.

[20] Fishman G. S. *Discrete-event Simulation: Modeling, Programming, and Analysis.* Springer, 2001, ISBN:0387951601.

8. Terminų ir santrumpų žodynas

DBVS – duomenų bazių valdymo sistema.

DEVS – formalizavimo kalba, sukurta *DEVS* standartizavimo grupės.

FSA – Formalių specifikacijų integruotos analizės sistema, skirta PLA kalba aprašytų specifikacijų analizei. Sistema buvo sukurta J. Padvarsko, G. Šuklevičiaus ir T. Krivoūso magistrinio darbo metu.

*PLA (**P**iece **L**inear **A**ggregate)* – atkarpomis tiesinis agregatas.

*PLA-CA (**P**iece **L**inear **A**ggregate – **C**omputer **A**ided)* – *PLA* formalizavimo kalba, pritaikyta kompiuteriams SPIN – Simple Promela Interpreter

PROMELA (***PRO**cess **ME**ta **LA**nguage)* - SPIN naudojama modeliavimo kalba.

*UPAAL – **UPP**sala **AAL**borg*

XML – eXtended Markup Language

9. Priedai

9.1.1. Priedas. Eksperimento „VAS_HUB“ formali specifikacija

```
<?xml version="1.0" encoding="UTF-8"?>
<Model Name="HUB+VAS RC2" xmlns="http://tempuri.org/ObjectModel.xsd">
  <AggregateList>
    <Aggregate>
      <Name>HUB</Name>
      <InputList>
        <Input>
          <Name>s1</Name>
          <Value>0</Value>
        </Input>
        <Input>
          <Name>s2</Name>
          <Value>0</Value>
        </Input>
      </InputList>
      <OutputList>
        <Output>
          <Name>out1</Name>
          <Value>0</Value>
        </Output>
        <Output>
          <Name>out2</Name>
          <Value>0</Value>
        </Output>
      </OutputList>
      <DiscreteVariableList>
        <DiscreteVariable>
          <Name>state1</Name>
          <Value>0</Value>
        </DiscreteVariable>
        <DiscreteVariable>
          <Name>state2</Name>
          <Value>0</Value>
        </DiscreteVariable>
      </DiscreteVariableList>
      <ContinuousVariableList>
        <ContinuousVariable>
          <Name>par_gen</Name>
          <Value>0.1</Value>
        </ContinuousVariable>
      </ContinuousVariableList>
      <InternalEventList>
        <InternalEvent>
          <Name>gen_par</Name>
          <ContinuousVariableName>par_gen</ContinuousVariableName>
          <HOperator>
            <Name>H_gen_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>par_gen := [exprnd]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_2</Name>
                <Express>state1 := [state1]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_3</Name>
                <Express>state2 := [state2]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
          <GOperator>
            <Name>G_gen_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>G_exp_1</Name>
                <Express>out1 := [1 WHEN (state1 = 0 ); 0 WHEN (state1 = 1 )]</Express>
              </Expression>
              <Expression>
                <Name>G_exp_2</Name>
```

```
                <Express>out2 := [1 WHEN ( ( state1 != 0 ) AND ( state2 = 0  ) ); 0 WHEN ( (
state2 = 1 ) OR ( state1 = 0  ) )]</Express>
              </Expression>
            </ExpressionList>
          </GOperator>
        </InternalEvent>
      </InternalEventList>
      <ExternalEventList>
        <ExternalEvent>
          <Name>atejo_s1</Name>
          <InputName>s1</InputName>
          <HOperator>
            <Name>H_atejo_s1_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>state1 := [s1]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
        </ExternalEvent>
        <ExternalEvent>
          <Name>atejo_s2</Name>
          <InputName>s2</InputName>
          <HOperator>
            <Name>H_atejo_s2_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>state2 := [s2]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
        </ExternalEvent>
      </ExternalEventList>
    </Aggregate>
    <Aggregate>
      <Name>VAS_1</Name>
      <InputList>
        <Input>
          <Name>in</Name>
          <Value>0</Value>
        </Input>
      </InputList>
      <OutputList>
        <Output>
          <Name>sign</Name>
          <Value>0</Value>
        </Output>
      </OutputList>
      <DiscreteVariableList>
        <DiscreteVariable>
          <Name>buf</Name>
          <Value>0</Value>
        </DiscreteVariable>
        <DiscreteVariable>
          <Name>buf_max</Name>
          <Value>3</Value>
        </DiscreteVariable>
        <DiscreteVariable>
          <Name>dirba</Name>
          <Value>0</Value>
        </DiscreteVariable>
      </DiscreteVariableList>
      <ContinuousVariableList>
        <ContinuousVariable>
          <Name>par_apt</Name>
          <Value>500</Value>
        </ContinuousVariable>
      </ContinuousVariableList>
      <InternalEventList>
        <InternalEvent>
          <Name>apt_par</Name>
          <ContinuousVariableName>par_apt</ContinuousVariableName>
          <HOperator>
            <Name>H_apt_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>par_apt := [exprnd WHEN (buf != 0 ); 1000 WHEN (buf = 0 )]</Express>
```

```
              </Expression>
              <Expression>
                <Name>H_exp_3</Name>
                <Express>dirba := [0 WHEN (buf = 0 ); 1 WHEN (buf != 0 )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_2</Name>
                <Express>buf := [buf WHEN (buf = 0 ); buf - 1 WHEN (buf != 0 )]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
          <GOperator>
            <Name>G_apt_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>G_exp_1</Name>
                <Express>sign := [0 WHEN (buf != buf_max ); 1 WHEN (buf = buf_max )]</Express>
              </Expression>
            </ExpressionList>
          </GOperator>
        </InternalEvent>
      </InternalEventList>
      <ExternalEventList>
        <ExternalEvent>
          <Name>atejo_par</Name>
          <InputName>in</InputName>
          <HOperator>
            <Name>H_atejo_par_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>par_apt := [exprnd WHEN ( ( buf = 0 ) AND ( sign = 0  ) AND ( dirba =
0 ) AND ( in = 1 ) ); par_apt WHEN ( ( buf != 0 ) OR ( sign = 1 ) OR ( dirba = 1 ) OR ( in =
0) )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_2</Name>
                <Express>buf := [buf WHEN ( ( buf = buf_max ) OR ( in = 0 ) OR ( dirba = 0  )
OR ( sign = 1  ) ); buf + 1 WHEN ( ( buf != buf_max ) AND ( in = 1 ) AND ( sign = 0  ) AND (
dirba = 1 ) )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_3</Name>
                <Express>dirba := [dirba WHEN (in = 0 ); 1 WHEN (in = 1 )]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
          <GOperator>
            <Name>G_atejo_par_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>G_exp_1</Name>
                <Express>sign := [1 WHEN ( ( sign = 0 ) AND ( in = 1 ) AND ( buf = buf_max  )
); sign WHEN ( ( sign != 0 ) OR ( in = 0 ) OR ( buf != buf_max  ) )]</Express>
              </Expression>
            </ExpressionList>
          </GOperator>
        </ExternalEvent>
      </ExternalEventList>
    </Aggregate>
    <Aggregate>
      <Name>VAS_2</Name>
      <InputList>
        <Input>
          <Name>in</Name>
          <Value>0</Value>
        </Input>
      </InputList>
      <OutputList>
        <Output>
          <Name>sign</Name>
          <Value>0</Value>
        </Output>
      </OutputList>
      <DiscreteVariableList>
        <DiscreteVariable>
          <Name>buf</Name>
          <Value>0</Value>
        </DiscreteVariable>
        <DiscreteVariable>
```

```
          <Name>buf_max</Name>
          <Value>3</Value>
        </DiscreteVariable>
        <DiscreteVariable>
          <Name>dirba</Name>
          <Value>0</Value>
        </DiscreteVariable>
      </DiscreteVariableList>
      <ContinuousVariableList>
        <ContinuousVariable>
          <Name>par_apt</Name>
          <Value>600</Value>
        </ContinuousVariable>
      </ContinuousVariableList>
      <InternalEventList>
        <InternalEvent>
          <Name>apt_par</Name>
          <ContinuousVariableName>par_apt</ContinuousVariableName>
          <HOperator>
            <Name>H_apt_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>par_apt := [exprnd WHEN (buf != 0 ); 1000 WHEN (buf = 0 )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_3</Name>
                <Express>dirba := [0 WHEN (buf = 0 ); 1 WHEN (buf != 0 )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_2</Name>
                <Express>buf := [buf WHEN (buf = 0 ); buf - 1 WHEN (buf != 0 )]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
          <GOperator>
            <Name>G_apt_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>G_exp_1</Name>
                <Express>sign := [0 WHEN (buf != buf_max ); 1 WHEN (buf = buf_max )]</Express>
              </Expression>
            </ExpressionList>
          </GOperator>
        </InternalEvent>
      </InternalEventList>
      <ExternalEventList>
        <ExternalEvent>
          <Name>atejo_par</Name>
          <InputName>in</InputName>
          <HOperator>
            <Name>H_atejo_par_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>par_apt := [exprnd WHEN ( ( buf = 0 ) AND ( sign = 0  ) AND ( dirba =
0 ) AND ( in = 1 ) ); par_apt WHEN ( ( buf != 0 ) OR ( sign = 1 ) OR ( dirba = 1 ) OR ( in =
0) )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_2</Name>
                <Express>buf := [buf WHEN ( ( buf = buf_max ) OR ( in = 0 ) OR ( dirba = 0 )
OR ( sign = 1 ) ); buf + 1 WHEN ( ( buf != buf_max ) AND ( in = 1 ) AND ( sign = 0 ) AND (
dirba = 1 ) )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_3</Name>
                <Express>dirba := [dirba WHEN (in = 0 ); 1 WHEN (in = 1 )]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
          <GOperator>
            <Name>G_atejo_par_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>G_exp_1</Name>
                <Express>sign := [1 WHEN ( ( sign = 0 ) AND ( in = 1 ) AND ( buf = buf_max  )
); sign WHEN ( ( sign != 0 ) OR ( in = 0 ) OR ( buf != buf_max  ) )]</Express>
              </Expression>
            </ExpressionList>
```

```
          </GOperator>
        </ExternalEvent>
      </ExternalEventList>
    </Aggregate>
  </AggregateList>
  <ConnectionList>
    <Connection>
      <InputName>in</InputName>
      <OutputName>out1</OutputName>
      <InputAggregateName>VAS_1</InputAggregateName>
      <OutputAggregateName>HUB</OutputAggregateName>
    </Connection>
    <Connection>
      <InputName>in</InputName>
      <OutputName>out2</OutputName>
      <InputAggregateName>VAS_2</InputAggregateName>
      <OutputAggregateName>HUB</OutputAggregateName>
    </Connection>
    <Connection>
      <InputName>s1</InputName>
      <OutputName>sign</OutputName>
      <InputAggregateName>HUB</InputAggregateName>
      <OutputAggregateName>VAS_1</OutputAggregateName>
    </Connection>
    <Connection>
      <InputName>s2</InputName>
      <OutputName>sign</OutputName>
      <InputAggregateName>HUB</InputAggregateName>
      <OutputAggregateName>VAS_2</OutputAggregateName>
    </Connection>
  </ConnectionList>
</Model>
```