KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Marius Gudas

Programinių komponentų funkcionavimo procesorių modulyje imitacinis modeliavimas

Magistro darbas

Darbo vadovas
V.Pilkauskas

Kaunas, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Marius Gudas

Programinių komponentų funkcionavimo procesorių modulyje imitacinis modeliavimas

Magistro darbas

Recenzentas

2011-05

Vadovas

2011-05

Atliko

2011

Kaunas, 2011

SIMULATION OF SOFTWARE COMPONENTS EXECUTION IN THE PROCESSOR MODULE

SUMMARY

The main object of this analysis was to create simulation model to simulate processor unit module (ProcUnit). There were two models created to compare differences and better understand objectives. Also in order to better understand analysis, a simple intelligent network process was taken to describe and use in the experiment. Situation when system receives two incoming requests for data was simulated in the project. As a result according to the amount of experimental PLA internal events to process request from network, it is better to use second simulation experiment model, because it needs less *ProcUnit* operations per experiment ant result of time is also much more less that in the first one model.

TURINYS

1. ĮVADAS

Tiriamojo darbo tikslas - sukurti programą – imitatorių, kuri generuotų SDP (sukurto atviro kodo paslaugų logikos vykdymo aplinkos Mobicents pagrindu) prototipo elgseną procesorių modelyje. Sugeneruotos modelio elgsenos trajektorijos yra apdorojamos nustatant modeliuojamų charakteristikų statistinius įvertinimus.

Telekomunikacinių tinklų sistemos turi teikti paslaugas vartotojams nenutrūkstamai: 24 valandas per parą 7 dienas per savaitę. Tai kelia didelius saugumo, patikimumo reikalavimus ne tik tinklo mazgų techniniai įrangai bet ir juose funkcionuojantiems programiniams komponentams. Be to šios sistemos dirba realiame laike ir užklausų paslaugų įvykdymui keliami griežti laiko apribojimai. Šie apribojimai kelia reikalavimus projektuojamų sistemų greitaveikai. Todėl yra svarbu įvertinti būsimos sistemos saugumą ir jos gyvybingumą. Ir tai yra pageidautina atlikti ankstyvuosiuose projektavimo etapuose prieš atliekant programavimą ir realizacijos testavimą. Padarytos projektavimo klaidos yra sunkiai taisomos programavimo ir realizavimo etapuose. Apibrėžus reikalavimus telekomunikacinių intelektualių tinklų paslaugų saugumo ir gyvybingumo savybėms ir panaudojus formalius modelius su atitinkamais programiniais analizės įrankiais, atsiranda galimybė automatizuotu būdu analizuoti telekomunikacinės sistemos saugumą ir gyvybingumą ankstyvuose projektavimo etapuose.

Kodo analizė pagrindu atlikti analitiniai skaičiavimai nusako tik skaičiavimo ir atminties resursų poreikių tendencijas augant užklausų skaičiui. Norint gauti tikslesnius įvertinimus reikia atlikti eksperimentinius tyrimus, imituoti sistemos darbą virtualioje aplinkoje sukuriant imitacinius modelius.

Tiriamojo darbo uždaviniai – sukurti skirtingus imitacinius modelius, juos ištestuoti, eksperimentiškai įrodyti kurį modelį naudoti prie atitinkamų sąlygų.

Šio dokumento analitinėje dalyje apžvelgiami esami sprendimai pasaulyje, aprašomas PLA metodas, taip pat nustatoma tyrimo problema, bei jos sprendimo būdai. Projektinėje dalyje aprašomi sistemoje sąveikaujantys komponentai, piešiamas jų statinis vaizdas, pateikiamos klasių diagramos. Tyrimo dalyje suformuluojamas tyrimo uždavinys, aprašomas tyrimo procesas. Taip pat grafiškai vaizduojami procesoriaus agregato PLA aprašymas su analitinėmis laiko paskirstymo algoritmo formulėmis.

2. ANALITINĖ DALIS

2.1. Telekomunikacinių intelektualiųjų tinklų paslaugų teikimo platforma

SDP (angl. „Service delivery platform“) koncepcija išsivystė per keletą pastarųjų metų. Terminas SDP reiškia sistemos architektūrą, kuri įgalina efektyvų vienos ar daugiau paslaugų klasių kūrimą, vystymą, organizavimą bei valdymą. SDP atsirado kaip telekomunikacinio tinklo vystymosi pasekmė.

Trečios kartos SDP, dažnai pateikiamos kaip SDP 2.0. Ji suformuota remiantis servisais grįsta architektūra (angl. „SOA – Service Oriented Architecture“), kuri įgalina efektyvų paslaugos integravimą, organizavimą ir valdymą. SDP 2.0 architektūra yra pagrista atvirais standartais ir palaikoma programinio realizavimo technologijomis. Tai suteikia galimybę kurti susijusius paslaugų paketus o taip pat esant reikalui perkelti paslaugas iš paveldėtų tinklų. Trečios kartos SDP architektūra užima pagrindinį vaidmenį sujungiant Telecom su Internetu bei Web 2.0.

Projektuojant sudėtingas sistemas, tokias kaip telekomunikacinės sistemos, būtina patikrinti, ar projektuojama sistema tenkins keliamus reikalavimus sistemos *patikimumui, greitaveikai, funkcionalumui*. Tai patikrinti galima tik sukūrus atitinkamus projektuojamos sistemos matematinius modelius. Šių modelių kūrimui yra naudojami formalūs metodai. Sukurti matematiniai modeliai leidžia analizuoti projektuojamą sistemą projektavimo metu. Matematinių modelių sudarymui reikia projektavimo metu sudarytus modelius (pvz. UML modelius) transformuoti į formalius matematinius modelius. Transformavimui realizuoti reikia ne tik apibrėžti modelių transformavimo taisykles, bet ir apibrėžti aplinkas, kuriose formalūs modeliai bus analizuojami [3].

2.2. Sistemų imitacinis modeliavimas

Sistemos modelis yra abstrakti sistema, kuri skirta analizuoti jos elgseną ir ją pavaizduoti. Sistemos modeliavimas padeda projektuotojui suprasti sistemos funkcionalumą. Modeliai yra abstraktūs, juose nėra aprašoma sisteminė informacija. Formaliaisiais metodais galima aprašyti būsimus modelius.

Aparatinės ir programinės įrangos modeliavimas yra labai aktuali problema. Modelis turi atspindėti sistemos savybes ir apibrėžti funkcionalumą. Modeliavimo eiga turi būti paremta formaliu aprašymu, kad sintezė nuo modelio specifikacijos iki diegimo būtų vykdoma teisingai.

Pavaizduoti nevienalytėms sistemoms buvo sukurta daugybė modelių. Matematinis modelis turėtų idealiai apimti sistemos procesų veikimo lygiagretumą, elgsenų eiliškumą ir ryšius tarp funkcinių modulių. Kai kurie modeliai yra numatyti sistemoms duomenų srautams apdoroti, kiti – labiau pritaikyti valdymui, arba apjungia duomenų srautų apdorojimą ir valdymą.

Kai modeliuojant yra susiduriama su laiku, labai naudinga yra panaudoti sinchroninius veiksmus. Sinchroniškumo hipotezė teigia, kad sistemos išėjimai yra sinchronizuoti su sistemos įėjimais, o sistemos reakcijos laikas yra nulinis. Tokiu būdu laikas yra tarsi nustumiamas į šalį.

Modelio funkcionavimo charakteristikų įvertinimui yra naudojamas imitacinis modeliavimas. Šiuo atveju pagal sudarytą modelį yra kuriama programa-imitatorius, kuri generuoja modelio elgseną. Sugeneruotos modelio elgsenos trajektorijos yra apdorojamos nustatant modeliuojamų charakteristikų statistinius įverčius. Toliau bus apžvelgti pagrindiniai modeliai, naudojami sistemų projektavime. Jie sugrupuoti pagal bendrąsias charakterizuojančias savybes. Trumpai apžvelgsime ir palyginsime pagrindines modelių savybes.

Šiuo metu pasaulyje sukurta daug priemonių, kurios leidžia automatizuoti imitacinių modelių sudarymą. Šias priemones galima suskirstyti į šias grupes: bendros paskirties imitacinio modeliavimo sistemos (pvz. GPSS, SIMSCRIPT, ARENA), probleminiai sričiai orientuotas imitacinio modeliavimo sistemos (pvz. telekomunikacinių tinklų imitacinio modeliavimo sistemos OPNET, Ns-2, OMNet++ ir imitacinių modelių sudarymo priemonės naudojančios sistemų formalius aprašymo metodus (pvz. SDL, Petri tinklai, DEVS, PLA).

Bendros paskirties modeliavimo kalbų trūkumas, kad jos dažnai neleidžia tiesiogiai adekvačiai aprašyti analizuojamų sistemų sudėtingus funkcionavimo algoritmus. Tais atvejais reikia kurti papildomus programinius modelius, kurie būtų įtraukti į pagrindinę imitatoriaus programą.

Tinklų imitatoriai gali būti suskirstyti į keletą grupių (pagal naudojamus protokolus, technologijas ar apdorojimo metodus). Kitas imitatorių skirstymo požymis yra naudojami imitavimo metodai. Yra naudojami du pagrindiniai imitatorių kūrimo metodai: diskrečių įvykių arba analitinis imitavimas. Dažnai abu minėti metodai yra naudojami kartu ir tai užtikrina priimtiną modelio greitaeigiškumą bei pakankamumą tikslumą.

OPNET sistemoje yra kuriami baigtinių būsenų modeliai juos apjungiant su analitiniais modeliais. Šioje sistemoje galima modeliuoti protokolus, įrenginius ir elgsenas panaudojant iki 400 specializuotų modeliavimo funkcijų. Sistema turi grafinį vartotojo interfeisą.

Ns-2 yra įvykiais valdomas tinklų imitatorius. Ši sistema turi daugelį internetinių protokolų modelių. Tinklo animatorius leidžia animuoti duomenų perdavimą tinkle paketų lygyje. Sistema leidžia vartotojams modifikuoti tinklo protokolus bei parametrus skirtinguose protokolų lygiuose.

OMNeT++ yra atviro kodo, komponentinė, atviros charakteristikos imitavimo aplinka. Ši sistema yra naudojama įvairių architektūrų tinklų imitaciniams modeliams kurti. Ši sistema buvo naudota kuriant internetinių protokolų imitatorius [5].

Imitacinių modelių sudarymo priemonės naudojančios formalius metodus turi šiuos privalumus:

- imitacinių modelių sudarymo metodai nėra orientuoti į konkrečią probleminę sritį. Imitacinių modelių sudarymo konkretizacija yra atliekama sudarant analizuojamos sistemos komponentų formalius aprašymus.
- sudaromi sistemos formalūs aprašymai leidžia pagrįsti sudaromo modelio adekvatumą bei to modelio loginį korektiškumą.

Pasaulyje labiausiai paplitęs DEVS (angl. discret evebnt specification) metodas, kurio teorinius pagrindus sudaro atominio ir jungtinio DEVS modelių panaudojimas sistemų elgsenai aprašyti. Atominis DEVS, tai matematinė struktūra turinti pagrindinius automatus modelio komponentus bei papildomas funkcijas aprašančias sistemas būsenų kaitą kuriuos iššaukia vidiniai komponento procesai.

Petri tinklai (PT) buvo sukurti apie 1960 metus ir buvo naudojami kompiuterinių sistemų konkuruojančioms operacijoms modeliuoti. Nuo PT atsiradimo iki šių dienų jie buvo iš esmės praplėsti. Šiuo metu PT leidžia modeliuoti konkuruojančius, asinchroninius, paskirstytus, lygiagrečius bei stochastinius procesus. PT naudojami sistemų būsenų kitimo lygtims ir imitaciniams modeliams sudaryti.

Pagrindiniai skirtumai tarp DEVS ir PLA formalizmų glūdi išėjimo-įėjimo atvaizdavimo ir išrinkimo funkcijų apibrėžimuose. DEVS šios funkcijos formalizuotos; PLA išėjimo-įėjimo atvaizdavimo funkcija neformalizuota, tačiau laikoma, jog vieno agregato išėjimo signalai kanalu pasiekę tikslinį agregatą tampa pastarojo įėjimo signalais. Išrinkimo funkcija (Select) PLA neapibrėžiama - vienalaikių operatorių vykdymo eiliškumo seka paliekama specifikacijos sudarytojo nuožiūrai, o imitacinio modeliavimo kompiuteriu atveju – konkrečiai imitacinio modeliavimo bibliotekos realizacijai.

Atkarpomis-tiesinių agregatų (PLA) formalizavimo metodas daugelį metų yra vystomas Kauno Technologijos Universitete. Šio metodo pagrindinis privalumas, kad jis leidžia vieningo formalaus aprašymo pagrindu atlikti sudaryto formalaus aprašymo teisingumo analizę ir kurti

imitacinius modelius. Šiame projekte numatoma sukurti sudėtingų sistemų modeliavimo sistema, kuri naudos PLA metodą.

2.3. Sistemų funkcionavimo formalizavimas agregatiniu metodu

Šiame projekte verslo procesai bus formalizuojami agregatiniu metodu (angl. „PLA piece-linear agregate“) metodu. Šio metodo privalumas, kad jis leidžia vieningo formalaus aprašymo bazėje kurti specifikuojamos sistemos imitacinius modelius bei formaliai atlikti sudaromo modelio teisingumą.

2.4. Atkarpomis-tiesiniai agregatai

Atkarpomis-tiesinių agreatų metodas yra išsamiai aprašytas G.Kovalenko bei N.Buslenko publikacijose. Šiame paragrafe yra pateikiami šio metodo pagrindiniai principai.

Aprašant sistemą sistemos būsenų aibėje *S* yra išskiriama baigtinė pagrindinių būsenų aibė $I = \{0,1,2,\ldots s\}$. Šios aibės elementai $\nu \in I$ yra pagrindinės agregato būsenos. Kiekvienai pagrindinei būsenai yra priskiriamas sveikas neneigiamas skaičius $\|\nu\|$, kuris vadinamas būsenos rangu bei išgaubtas daugiakampis $Z^{(\nu)}$ užduotas $\|\nu\|$ išmatavimų euklido erdvėje. Skaitoma, kad būsenų aibę $Z = \bigcup_{\nu \in I} Z^{(\nu)}$ sudaro poros $(\nu, z^{(\nu)})$, čia $\nu \in I$, o $z^{(\nu)} \in Z^{(\nu)}$. $z^{(\nu)}$ - vadinamos papildomomis agregato koordinatėmis.

Pradiniu laiko momentu t_0 agregatas randasi būsenoje $z(t_0) = (\nu, z^{(\nu)}(0))$, čia $z^{(\nu)}(0) \in Z^{(\nu)}$. Nesant įėjimo signalo, kai $t > t_0$ taškas $z^{(\nu)}(t)$ juda srityje $Z^{(\nu)}$ iki bus pasiektas šios srities kontūras. Laiko momentas t_1, kai pasiekiamas kontūras, vadinamas atraminiu.

Daugiakampio *Z* kontūras yra aprašomas lygtimis:

$$\sum_{i=1}^{\|\nu\|} \gamma_{ji}^{(\nu)} z_i^{(\nu)} + \gamma_{j0}^{(\nu)} = 0, \quad j = 1,\ldots, m(\nu),$$

čia $m(\nu)$ – kontūrų skaičius;

$z_i^{(\nu)}$ - vektoriaus $z^{(\nu)}$ komponentės, $i = 1,\ldots,\|\nu\|$;

$\gamma^{(\nu)}$ - sistemos parametrais apibūdinami dydžiai.

Atraminiu laiko momentu agregato pagrindinė būsena kinta iš ν į ν', o agregato papildomos koordinatės įgyja reikšmę $z^{(\nu')}(t_1) \in Z^{(\nu')}$.

Toliau papildomos koordinatės kinta srityje $Z^{(\nu')}$ iki nepateks ant šios srities kontūro. Tai įvykus agregatas vėl keičia būseną.

Atkarpomis-tiesinio agregato būsenai patekus į srities kontūrą yra išduodamas išėjimo signalas $y \in Y$, čia Y – išėjimo signalų aibė, kuri yra analogiška aibei Z. Išėjimo signalo struktūra

$$y = \left(\lambda, y^{(\lambda)}\right),$$

čia λ - išėjimo signalo diskretinė dalis;

$y^{(\lambda)}$ - išėjimo signalo papildomų koordinačių vektorius, priklausantis nuo λ

$$y^{(\lambda)} = \left(y_1^{(\lambda)}, \ldots, y_{r(\lambda)}^{(\lambda)}\right).$$

Jei laiko momentu t^* į agregatą yra paduodamas įėjimo signalas, tai agregato papildomos koordinatės $z^{\nu}(t_*)$ nustoja kitusios ir agregato būsena momentaliai pereina į kitą tos pačios srities ar kitos srities $Z^{(\nu'')}$ tašką.

Įėjimo signalas turi struktūrą analogišką išėjimo signalo struktūrai, t.y.

$$x = \left(\mu, x^{(\mu)}\right).$$

Įėjimo signalo atėjimo momentu agregatas išduoda išėjimo signalą ir šis momentas taip pat yra atraminis. Toliau taškas $z^{(\nu'')}(t)$ srityje $Z^{(\nu'')}$ juda taip pat, kaip buvo aprašyta anksčiau.

Kai nėra įėjimo signalų atkarpomis-tiesinio agregato būsenos kitimas apsirašo tokiomis lygtimis:

$$\nu(t) = \nu = const\,;\ \frac{dz^{(\nu)}}{dt} = \alpha^{(\nu)},$$

čia $\alpha^{(\nu)}$ - pastovus vektorius, turintis pavidalą

$$\alpha^{(\nu)} = \left(\alpha_1^{(\nu)}, \ldots, \alpha_{m(\nu)}^{(\nu)}\right).$$

Vektorinėje formoje $z^{(\nu)}(t)$ sprendinys gali būti užrašytas

$$z^{(\nu)}(t) = z^{(\nu)}(0) + \alpha^{(\nu)}(t - t_0).$$

Spręsdami papildomų koordinačių kitimo ir sričių kontūrų lygtis kartu galima išskaičiuoti laiko momentus, kai papildomų koordinačių reikšmės papuola į kontūrą. Pvz., pirmas atraminis laiko momentas

$$t_1 = \min\left[t : z^{(\nu)}(0) + \alpha^{(\nu)}(t - t_0) \in \bigcup_{i=1}^{m(\nu)} Z_j^{(\nu)}, t > t_0\right]$$

arba

$$t_1 = \min\left\{t : t > t_0, \sum_{i=1}^{\|\nu\|} \gamma_{ji}^{(\nu)}\left[z_{ji}^{(\nu)}(0) + \alpha_i^{(\nu)}(t - t_0)\right] + \gamma_{j0}^{(\nu)} = 0\right\},$$

Minimumas yra ieškomas indeksų $j = 1,\ldots, m(\nu)$ aibėje.

Jeigu pažymėsime

$$\tau_j = -\left(\sum_{i=1}^{\|\nu\|} \gamma_{ji}^{(\nu)} z_i^{(\nu)}(0) + \gamma_{j0}^{(\nu)} \right) \Bigg/ \sum_{i=1}^{\|\nu\|} \gamma_{ji}^{(\nu)} \alpha_i^{(\nu)}$$

ir

$$\tau = \min\{\tau_j : \tau_j > 0\},$$

tai laiko momentas, kai pirmą kartą pasiekiamas kontūras

$$t_1 = t_0 + \tau .$$

Patekus į kontūrą nauja pagrindinė būsena ν' apibrėžiama tikimybių pasiskirstymu P_1, kuri priklauso tik nuo būsenos $z(t_1)$. Tam, kad apibrėžti papildomų koordinačių vektorių $z^{(\nu')}$ sudaromas pagalbinis vektorius η, kurio pasiskirstymo dėsnis nepriklauso nuo proceso istorijos, o priklauso tik nuo ν ir $z^{(\nu)}$.

Papildomų koordinačių vektorius apskaičiuojamas

$$z^{(\nu')} = \left(z_*^{(\nu)}, \eta\right) \times L_z^{(\nu\nu')},$$

čia $L_z^{(\nu\nu')}$ - matrica, kurios elementai priklauso nuo ν, ν' ir $z_*^{(\nu)}$; $z_*^{(\nu)} = z^{\nu}(\tau)$.

Kai ateina įėjimo signalas $\left(\mu, x^{(\mu)}\right)$ naują pagrindinę būseną ν'' nusako tikimybinis pasiskirstymas P_2, kuris priklauso nuo ν, $z_*^{(\nu)}$ ir μ.

Papildomų koordinačių vektoriaus $z^{(\nu'')}$ apibrėžimui sudaromas papildomas vektorius ξ, kurio pasiskirstymas nepriklauso nuo proceso istorijos, o priklauso nuo ν, $z_*^{(\nu)}$ ir μ.

$$z^{(\nu'')} = \left(z_*^{(\nu)}, \xi, x^{(\mu)}\right) \times L_x^{(\nu,\nu'')},$$

čia $L_x^{(\nu,\nu'')}$ - matrica, kurios elementai priklauso nuo ν, $z_*^{(\nu)}$, ν'' ir μ.

Išėjimo signalas formuojamas, kai $z^{(\nu)}$ pasiekia kontūrą arba ateina įėjimo signalas (laiko momentu t').

Pirmu atveju:

$$\lambda = \lambda\left(\nu, \nu', z_*^{(\nu)}\right),$$
$$y^{(\lambda)} = \left(z_*^{(\nu)}, \eta\right) \times L_y^{(\nu\nu')}.$$

Matricos $L_y^{(\nu\nu')}$ elementai priklauso nuo ν, ν' ir $z_*^{(\nu)}$.

Antru atveju:

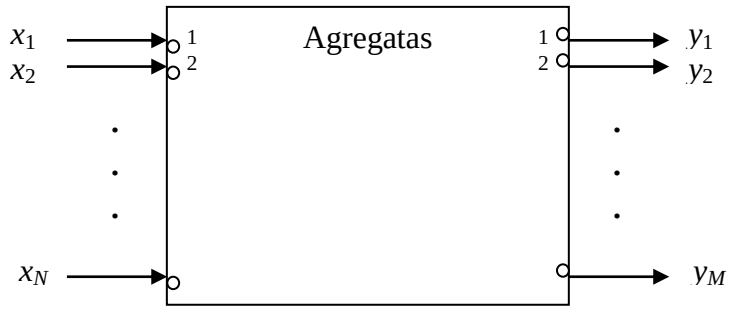
$$\lambda = \lambda\left(\nu, \nu'', z^{(\nu)}(t'), \mu\right),$$

$$y^{(\lambda)} = \left(z^{(\nu)}(t'), \xi, x^{(\mu)}\right) \times L_y^{(\nu\nu'')}.$$

Matricos $L_y^{(\nu\nu'')}$ elementai priklauso nuo ν, ν'' , $z^{(\nu)}(t')$ ir μ.[4]

2.5. Valdančiųjų sekų panaudojimas agregatų funkcionavimo formalizavimui

Sakykime, kad turime agregatą, kuris turi *N* įėjimo ir *M* išėjimo sąveikavimo taškų (*ST*) (1 pav).



1 pav. Agregato schematinis atvaizdavimas.

Įėjimo signalai $x_1, x_2, ..., x_w \in X$ yra paduodami į įėjimo *ST*. Signalai $x_i \in X_i, i = \overline{1, N}$ yra vadinami elementariais signalais, o aibė X_i yra vadinama elementarių signalų aibe. Bendru atveju elementarus signalas yra vektorius, t.y. $x_i = \left\{x_i^1, x_i^2, \ldots x_i^{r_i}\right\}$, be to, šio vektoriaus koordinačių reikšmės priklauso atitinkamai aibei, t.y. $x_i^j \in X_i^j$, $j = \overline{1, r_i}$.

Elementarūs signalai kurie gali ateiti į *i*-jį *ST*, sudaro aibę:

$$X_i = X_i^1, X_i^2, \ldots, X_i^{r_i}, i = \overline{1, N}.$$

Agregatų įėjimo signalų aibė yra lygi aibių X_i sąjungai, t.y.

$$X = \bigcup_{i=1}^{N} X_i ,$$

Analogiškai yra apibrėžiama išėjimo signalų aibė:

$$Y = \left\{y_1, y_2, \ldots y_M\right\},\ y_1 = \left\{y_l^1, y_l^2, \ldots y_l^{s_i}\right\} \in Y_l ,$$

$$y_l^k \in Y_l^k, l = \overline{1, M}, k = \overline{1, s_l} .$$

Elementarių signalų aibė išeinanti iš *l*-ojo ST yra lygi:

$$Y_l = Y_l^1, Y_l^2, \ldots, Y_l^{s_l}, l = \overline{1, M} .$$

Agregato išėjimo signalų aibė:

$$Y = \bigcup_{l=1}^{M} Y_l .$$

Agregato funkcionavimas yra nagrinėjamas diskrečiais laiko momentais, kurie priklauso aibei $T = \{t_0, t_1, ..., t_m, ...\}$. Tais laiko momentais gali įvykti vienas ar keli įvykiai, kurie sukelia agregato būsenos pasikeitimą. Agregato įvykių aibę sudaro du nepersikertantys poaibiai $E' = E' \cup E''$. Aibę $E' = \{e'_1, e'_2, ..., e'_N\}$ sudaro įvykiai, kurie įvyksta dėl įėjimo signalų atėjimo. Tarp aibių X ir E' elementų yra funkcinis ryšys. Poaibis $E'' = \{e''_1, e''_2, ..., e''_f\}$ yra vadinamas vidinių įvykių poaibiu, čia $e''_i = \{e''_{ij}, j = 1, 2, 3, \ldots\}$, $i = \overline{1, f}$, yra agregato vidiniai įvykiai, f – operacijų, kurios gali vykti agregate, skaičius. Aibės E'' įvykiai fiksuoja operacijų pabaigą.

Laiko momentų aibė *T* susideda iš dviejų poaibių:

$$T = T' \cup T'',$$

čia T' - laiko momentų poaibis, kurį sudaro laiko momentai, kada ateina įėjimo signalai,

T'' - vidinių įvykių įvykimo laiko momentai.

Kiekvienam vidiniam įvykiui $e_i \in E''$ yra užduodama valdanti seka, t.y.

$$e''_i \mapsto \{\xi_j^{(i)}\},\ \mathrm{j} = \overline{1, \infty},$$

čia $\xi_j^{(i)}$ - operacijos trukmė, kuriai pasibaigus įvyksta vidinis įvykis.

Taip pat nurodoma skaitliukų aibė:

$$\{r(e''_i, t_m)\},\ \ i = \overline{1, f},$$

čia $r(e''_i, t_m)$ - e''_i įvykių skaičius, kuris įvyko laiko intervale $[t_0, t_m]$.

Tam, kad būtų galima apibrėžti operacijų pradžios ir pabaigos momentus, yra naudojamos aibės valdančiųjų sumų:

$$\{s(e''_i, t_m)\},\quad \{w(e''_i, t_m)\},\ \ i = \overline{1, f},$$

čia $s(e''_i, t_m)$ – operacijos pradžios momentas, kuriai pasibaigus įvyksta e''_i įvykis;

$w(e''_i, t_m)$ - operacijos pabaigos momentas.

Neprioritetinių operacijų atveju, valdanti suma $w(e''_i, t_m)$ yra apibrėžiama taip:

$$w(e''_i, t_m) = \begin{cases} s(e''_i, t_m) + \xi_{r(e''_i, t_m)+1}, & \text{jei laiko momentu } t_m \text{ vyksta operacija, kuriai} \\ & \text{pasibaigus, įvyks įvykis } e''_i; \\ \infty, & \text{priešingu atveju.} \end{cases}$$

Begalybės simbolis (∞) yra naudojamas pažymėti, kad yra nežinomas laiko momentas, kai užsibaigs operacija..

Pateiktas valdančios sumos apibrėžimas yra naudojamas sudarant imitacinius modelius. Kai agregatinis modelis yra naudojamas sistemų formalizavimui ir sistemos funkcionavimo korektiškumo analizei, valdanti suma apibrėžiama supaprastintai:

$$w(e_i'', t_m) = \begin{cases} < \infty, & \text{jei laiko momentu } t_m \text{ vykstaoperacija, kuriai} \\ & \text{pasibaigus, įvyksįvykis } e_i''; \\ \infty, & \text{priešingu atveju.} \end{cases}$$

Įvedus valdančias sumas, agregato būsenos tolydinė komponentė įgauna pavidalą:

$$z_v(t_m) = \{w(e_1'', t_m), w(e_2'', t_m), \ldots, w(e_f'', t_m)\}.$$

Tolydinės komponentės koordinatės apibrėžia laiko momentus, kada agregate gali įvykti įvykiai. Be to, visada $w(e_i'', t_m) \geq t_m$.

Agregato būsena $z(t_m)$ kinta diskrečiais laiko momentais t_m, m = 1, 2, …, ir išlieka pastovi laiko intervalais $[t_m, t_{m+1})$, m = 0, 1, 2, …, čia t_0 - pradinis sistemos funkcionavimo momentas.

Kai yra žinoma agregato būsena $z(t_m)$, m = 0, 1, 2, …, laiko momentas t_{m+1}, kai įvyksta sekantis įvykis, paskaičiuojamas taip:

$$t_{m+1} = \min_i \{w(e_i'', t_m)\},\ 1 \leq i \leq f\ .$$

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:

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

2.7. Laiko paskirstymas Unix OS

Reali sistema, kurios imitacinis modelis yra kuriamas šiame darbe bus eksploatuojama UNIX operacinėje sistemoje. Tokia operacinė sistema buvo pasirinkta neatsitiktinai. UNIX pasirinkimą lėmė aukšti sistemos greitaveikos reikalavimai. Klientas – serveris turi palaikyti nuolatinį ryšį [8]. Žinant detalų komponentų vidinį veikimo algoritmą galima vertinti/prognozuoti reikalingus skaičiavimo resursus, bei operatyvinės atminties poreikius. Šie du parametrai labai svarbūs telekomunikacinių sistemų mazgų apkrovimo išankstiniam vertinimui. Įprasta telekomunikacijose, kad mazgo aprovimas neturi viršyti 40% maksimalaus apkrovimo, kai klasteryje yra du mazgai. Norint užtikrinti nepertraukiamą tokios

telekomunikacinės sistemos veikimą sistemos mazgas turi sugebėti pilnai perimti kito mazgo funkcijas įvykus gedimui kitame mazge. Tokiu atveju likusiame mazge apkrovimas yra ne didesnis nei 80%.

Esant tokiems išoriniams reikalavimams, dar didesni reikalavimai yra keliami interaktyviems skaičiavimams viduje sistemos. Šie reikalavimai tenkinami dedikuotuosiuose mikrokompiuteriuose (angl. „dedicated microcomputers“). Pagrindinis laiko paskirstymo sistemos tikslas – minimizuoti laiko atsaką klientui, kai daugybė paslaugos paraiškų vienu metu patenka į sistemą apdorojimui [9].

2.8. Tyrimo uždavinys

Kaip adekvačiai sumodeliuoti laiko paskirstymo sistemą (angl. „Time-sharing system“)? Galimi iškeltos problemos sprendimo būdai:

1) imitaciniame programinių komponentų funkcionavimo procesorių modulyje modelyje aprašyti ir po to realizuoti visus laiko paskirstymo algoritmo įvykius, tačiau tokiu atveju imitacinio modeliavimo eksperimentas gali pareikalauti labai daug operacinių skaičiavimų;
2) panaudoti analitinę matematinę abstrakciją prarandant tam tikrą modelio adekvatumą.

Šio tyrimo tikslas yra nustatyti kokiais atvejais galima naudoti antrą imitavimo būdą neprarandant modelio tikslumo. Todėl darbe sudaryti du skirtingi imitaciniai modeliai, pirmam ir antram modeliui, bei tyrimai atlikti pasirinkus paprastą intelektualiųjų telekomunikacinių tinklų (angl. „intelligent network“) paslaugą, kai į sistemą ateina dvi paraiškos ir procesoriaus agregatas apdoroja būtent šiuos procesus.

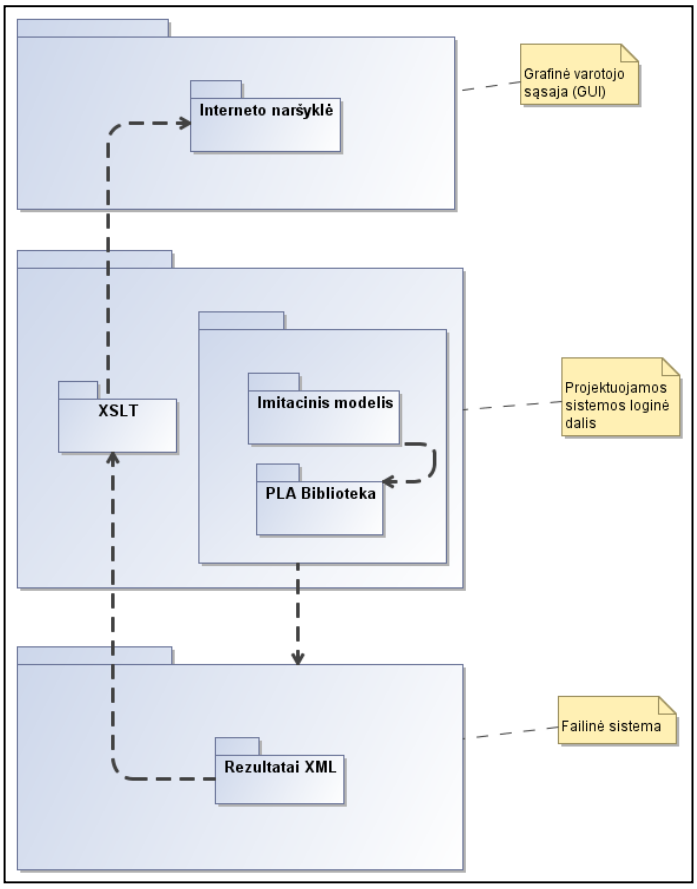
3. PROJEKTINĖ DALIS

Kuriamos sistemos panaudojimo (darbo) aplinka gali nuolat keistis priklausomai nuo vartotojo poreikių, tačiau svarbiausia jog kompiuteryje būtų įdiegta Windows operacinė sistema, kuo greitesnis procesorius (esant dideliam skaičiavimų skaičiui galimas kompiuterio darbo ryškus apkrovimas), DotNet Framework 3.5,bei programinė įranga palaikanti XML tipo failus.

Architektūros sprendimų vizualizavimo priemonės (diagramos) kuriamos MagicDraw 16.6, bei MS Visio paketų priemonių teikiama pagalba.

3.1. Sistemos statinis vaizdas

Sistemą sudarančias klases galima išskaidyti į Duomenų modelių ir Loginių elementų paketus. Tokio skaidymo vaizdas pateikiamas 2 pav. Tarp jų egzistuojantis ryšys rodo, jog loginiai elementai kontroliuoja duomenų modelio klases, kurių objektai neturi informacijos apie loginius elementus. Kaip vienas sistemos blokas yra laikoma PLASim biblioteka, pats imitacinis modelis, bei XSLT transformavimo failas naudojamas rezultatų apipavidalinimui.

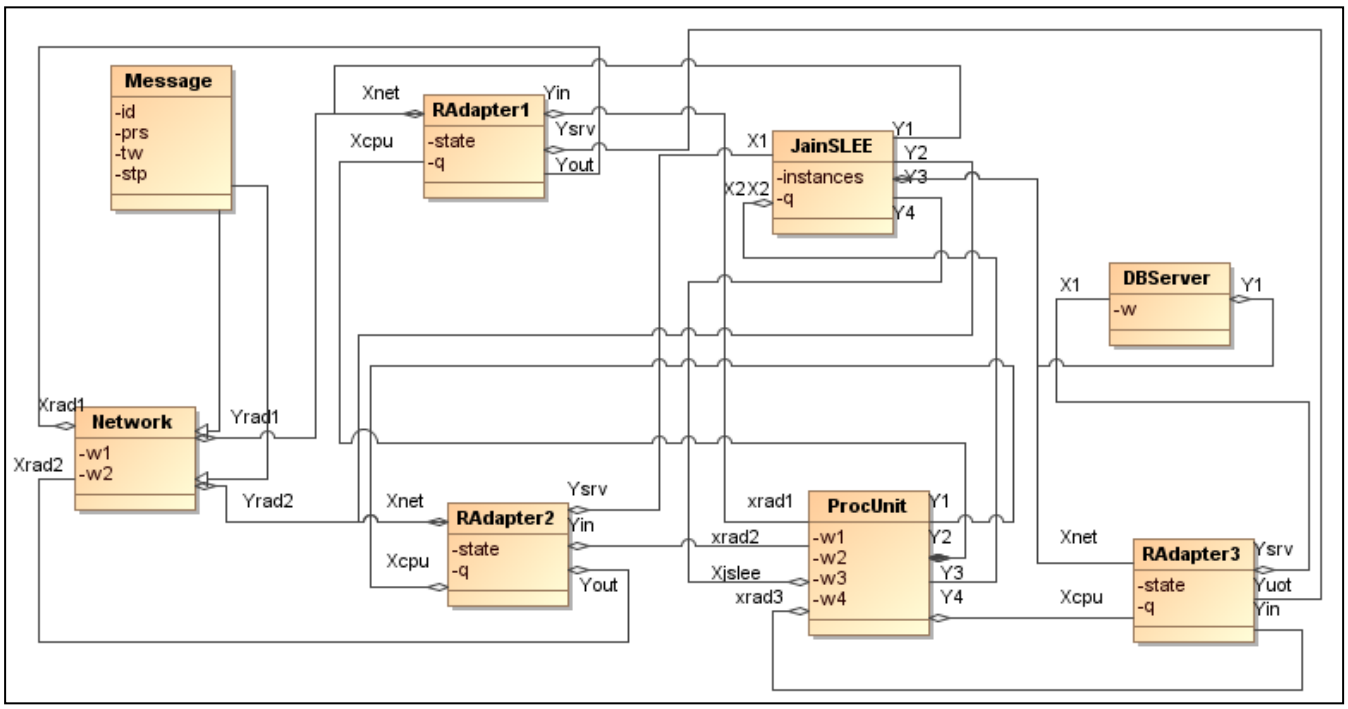


2 pav. Sistemos paketų diagrama

Imitacinio modeliavimo metu visi surinkti statistiniai duomenys pateikiami XML dokumento formoje ***report.xml*** faile. Ši failą formuoja klasės ***StatD*** metodas ***Report()***. Šio metodo realizacija pateikta 3 dokumento priede.

3.2. Imitacinio modelio agregatų sujungimo schema

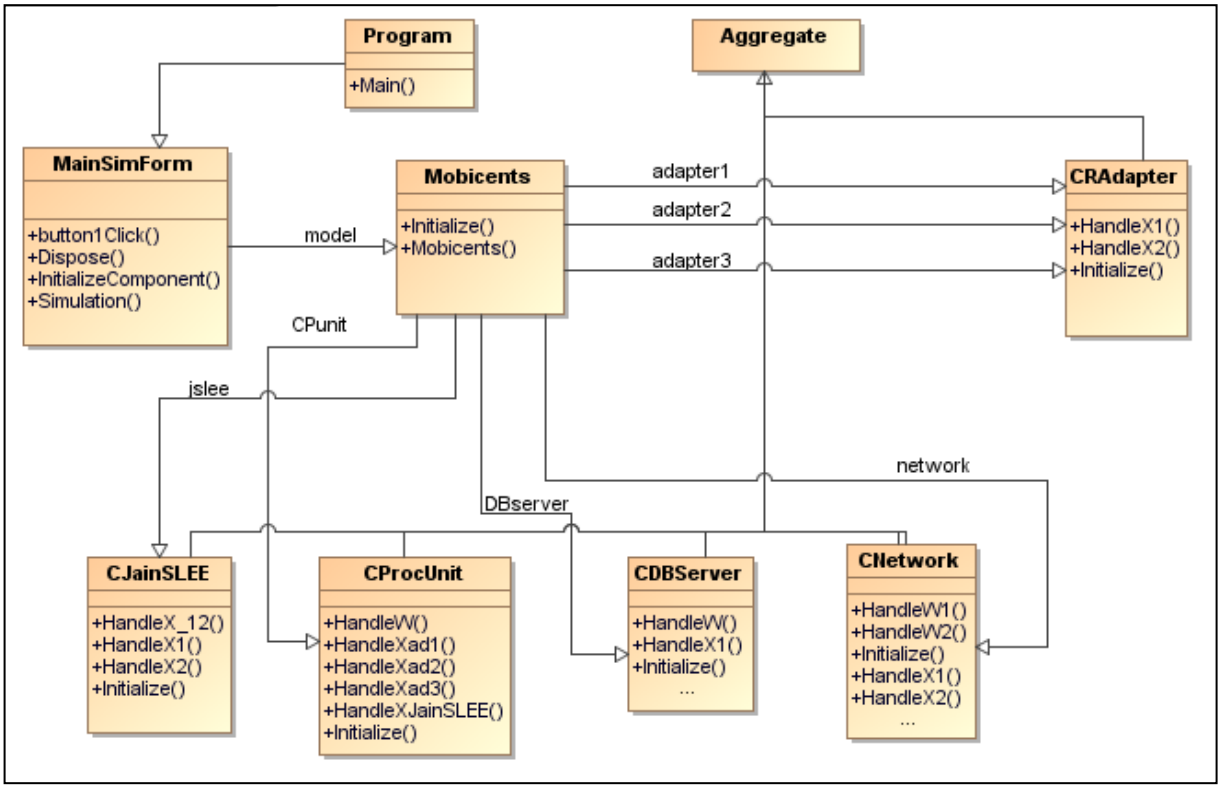
Žemiau (žr. 3 pav.) pateiktas agregatų sujungimo vaizdas.



3 pav. Imitacinio modelio agregatų sujungimo schema

3.3. Imitacinio modelio klasių diagrama

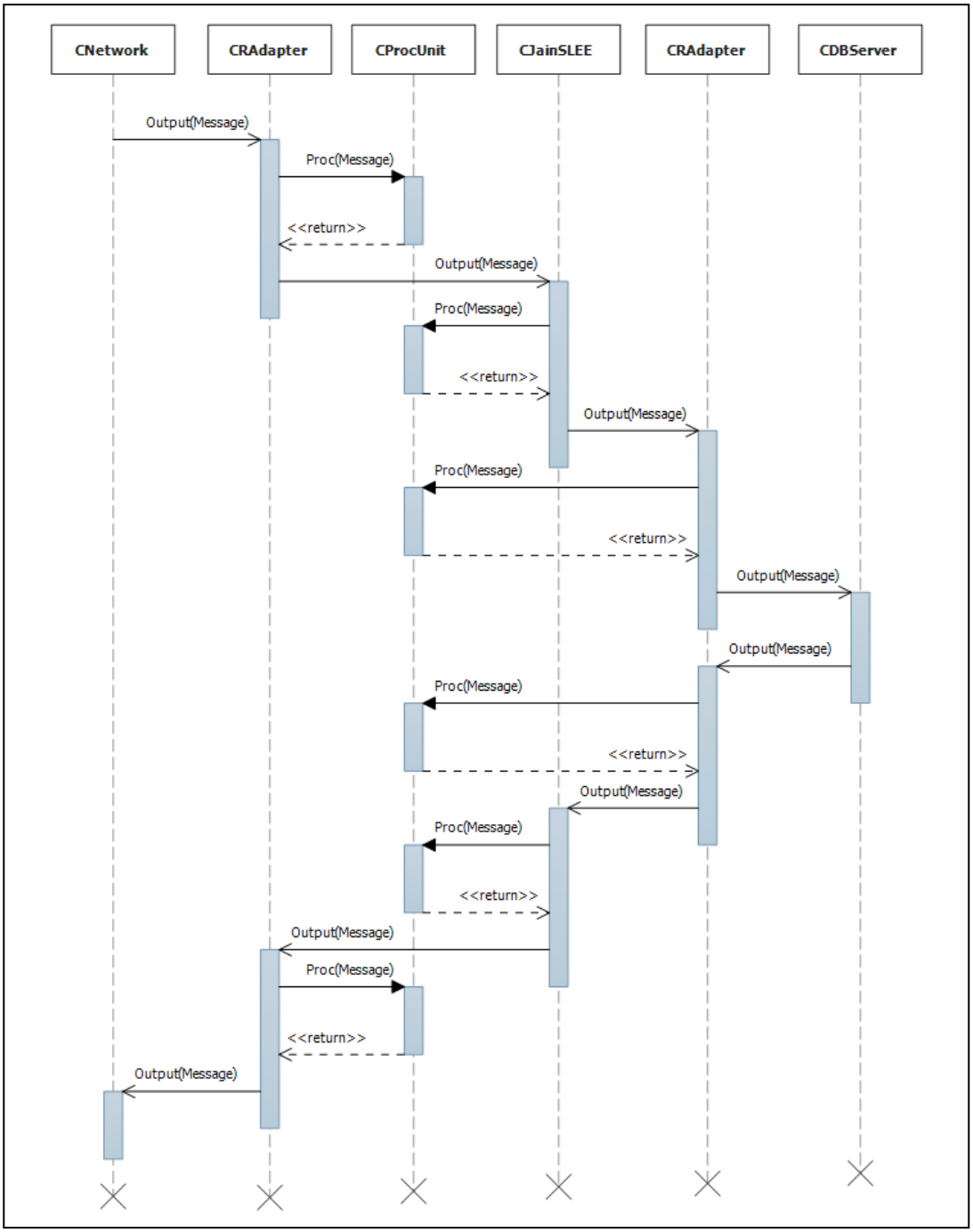
Agregatinės sistemos agregatų sujungimą realizuoja ***Mobicents*** klasė (žr. 4 pav.).Kaip pavaizduota nurodytame paveiksle imitacini modelį sudaro septynios klasės: *MainSim, Mobicents, CProcUnit, CJainSlee, CDBServer, CRAdapter,* bei *CNetwork*. Valdanti klasė Mobicents yra pagrindinis elementas jungiantis sistemos dalis. Į sistemą atkeliavusi paslauga yra resursų adapteriais nusiunčiama į procesorių, tuomet, atgal į adapterį kuris pritaiko signalą duomenų bazei, šioji savo ruožtu grąžina atsaką procesoriui, procesorius perduoda CNetwork, taip paslauga su reikiama informacija išsiunčiama iš sistemos.



4 pav. Imitacinio modelio klasių diagrama

3.4. Imitacinio modelio sąveikos diagramos

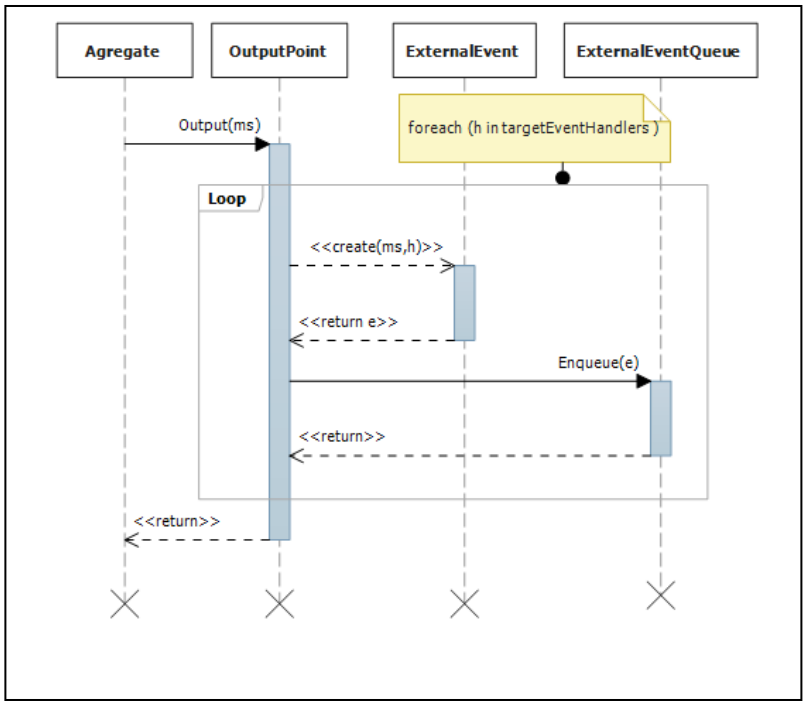
Imitacinio modelio sekos diagrama (žr. 5 pav.).



5 pav. Modelyje imituojamos tipinės IN paslaugos sekų diagrama

Agregatų sąveika siunčiant pranešimus agregatų sujungimo kanalais realizuojama sąveikos taško objekto metodu ***Output()*** (6 pav.). Siunčiamas pranešimas yra objektas, kurio bazinė klasė ***AgEventArg***. Sąveikos taško objekto metode ***Output()*** ši pranešimo struktūra yra įvelkama į naujai sugeneruotą išorinio įvykio ***ExternalEvent*** tipo objektą. Tokių išorinių įvykių sugeneruojama tiek kiek yra saugoma nuorodų sąveikos taško objekto atribute ***targetEventHandlers***. Šiame atribute yra saugomos nuorodos į agregato atitinkamo išorinio

įvykio apdorojimo metodo delegatą ***ExternalEvenHandler***. Visi sugeneruoti išoriniai įvykiai patalpinami ir saugome eilėje ***externalEventQueue***.

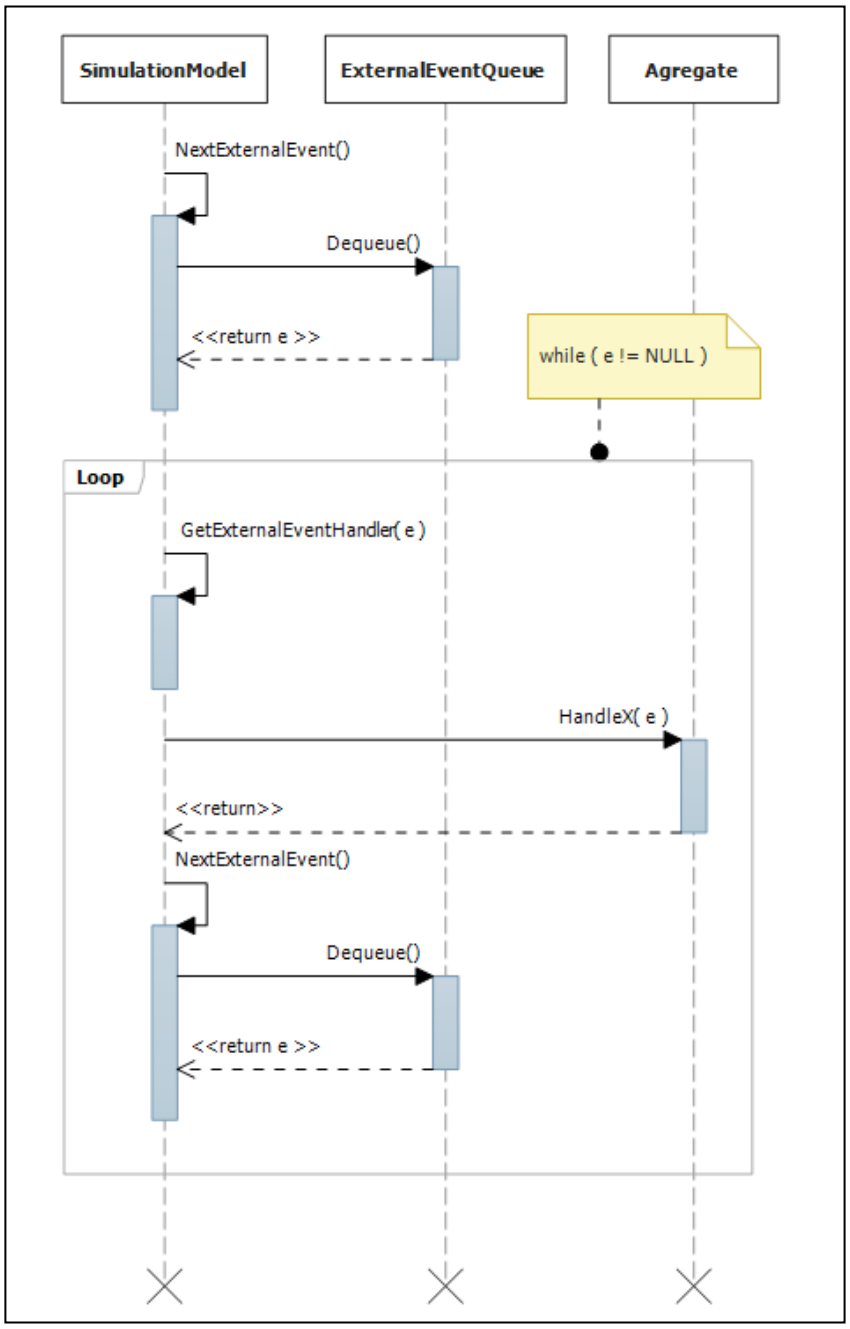


6 pav. Agregatų sąveika siunčiant pranešimus

Agregato išorinio įvykio generavimą inicijuoja agregatas pranešimo siuntėjas metodu ***Send()*** kreipdamasis į atitinkamą sąveikos taško objektą. Siunčiamas pranešimas yra objektas, kurio bazinė klasė ***AgEventArg***. Sąveikos taško objekto metode ***output()*** ši pranešimo struktūra yra įvelkama į naujai sugeneruotą išorinio įvykio ***ExternalEvent*** tipo objektą. Tokių išorinių įvykių sugeneruojama tiek kiek yra saugoma nuorodų sąveikos taško objekto atribute ***targetEventHandlers***. Šiame atribute yra saugomos nuorodos į agregato atitinkamo išorinio įvykio apdorojimo metodo delegatą ***ExternalEvenHandler***. Visi sugeneruoti išoriniai įvykiai patalpinami ir saugome eilėje kol yra išrenkami apdorojimui klasės ***SimulationModel*** metodu ***NextExternalEvent()***.

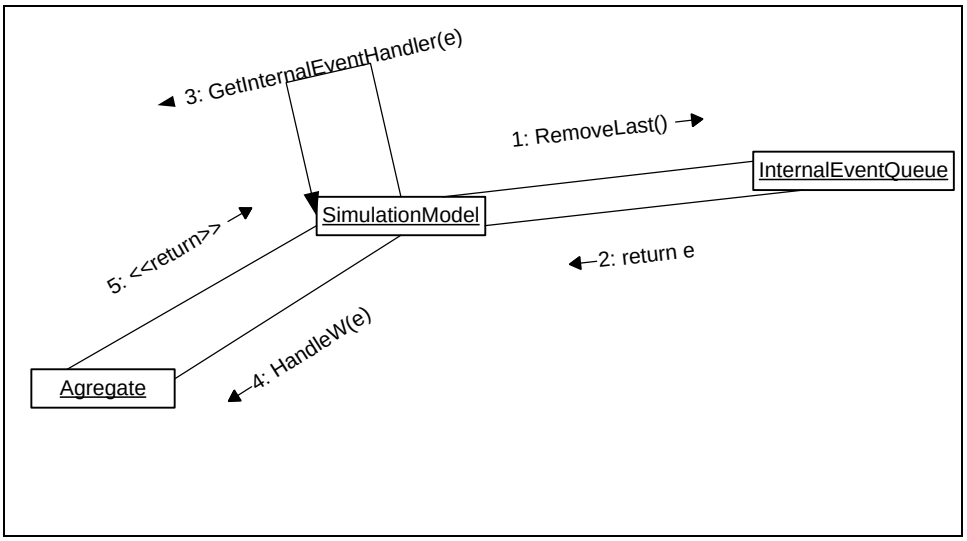
Agregato išorinio įvykio generavimui ir jo apdorojimui bibliotekoje yra skirtos tokios klasės (žr. 7 pav.):

- ***ExternalEvent*** – išorinio įvykio klasė;
- ***ExternalEventHandler*** - agregato išorinio įvykio apdorojimo metodo delegatas;
- ***AgEventArg*** – pranešimo perduodamo tarp agregatų bazinė klasė.



7 pav. Išorinio įvykio apdorojimo sekų diagrama

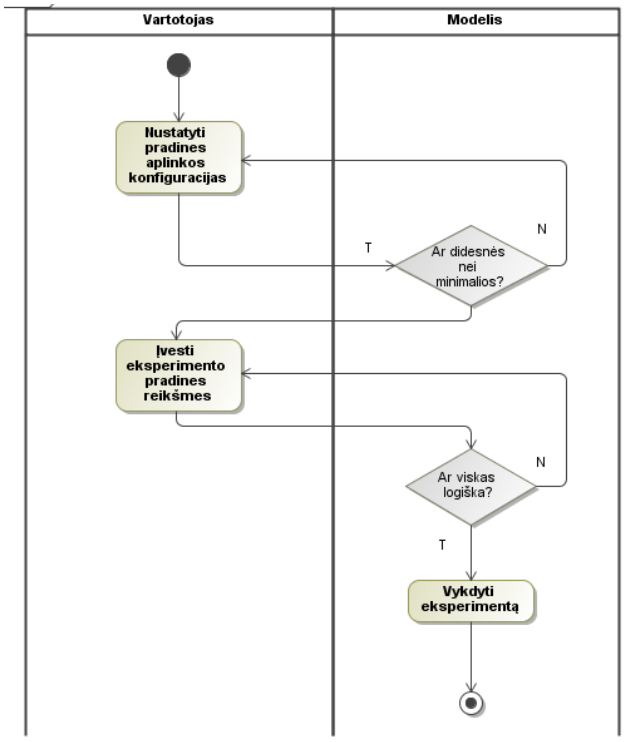
Agregato vidinio įvykio generavimą inicijuoja agregato valdančios sumos ***ControlSum*** metodas ***CreateInternalEvent(w)***. Šio metodo parametru ***w*** nustatome laiko momentą kada yra numatomas vidinis įvykis. Sugeneruotas vidinis įvykis patalpinamas į surūšiuotą vidinių įvykių sąrašą ***internalEventQuque***. Rūšiavimas yra atliekamas pagal vidinio įvykio atributą ***w*** (žr. 8 pav.).



8 pav. Vidinio įvykio apdorojimo algoritmo bendradarbiavimo diagrama

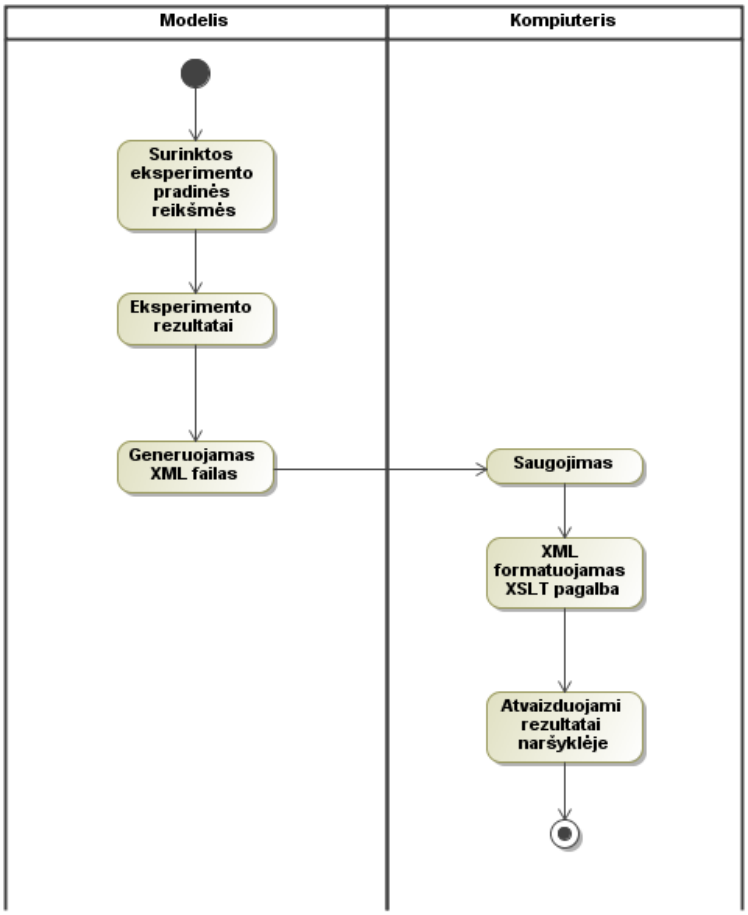
3.5. Imitacinio modelio veiklos diagramos

Imitacinis eksperimentas pradedamas įvedant pradines charakteristikas kuriomis vėliau naudojantis analitinėmis formulėmis bus sugeneruotas rezultatas atitinkantis sistemos įvairius parametrus. 9-ajame paveikslėlyje pateikiama eksperimento eigos veiklos diagrama.



9 pav. Eksperimento eigos veiklos diagrama

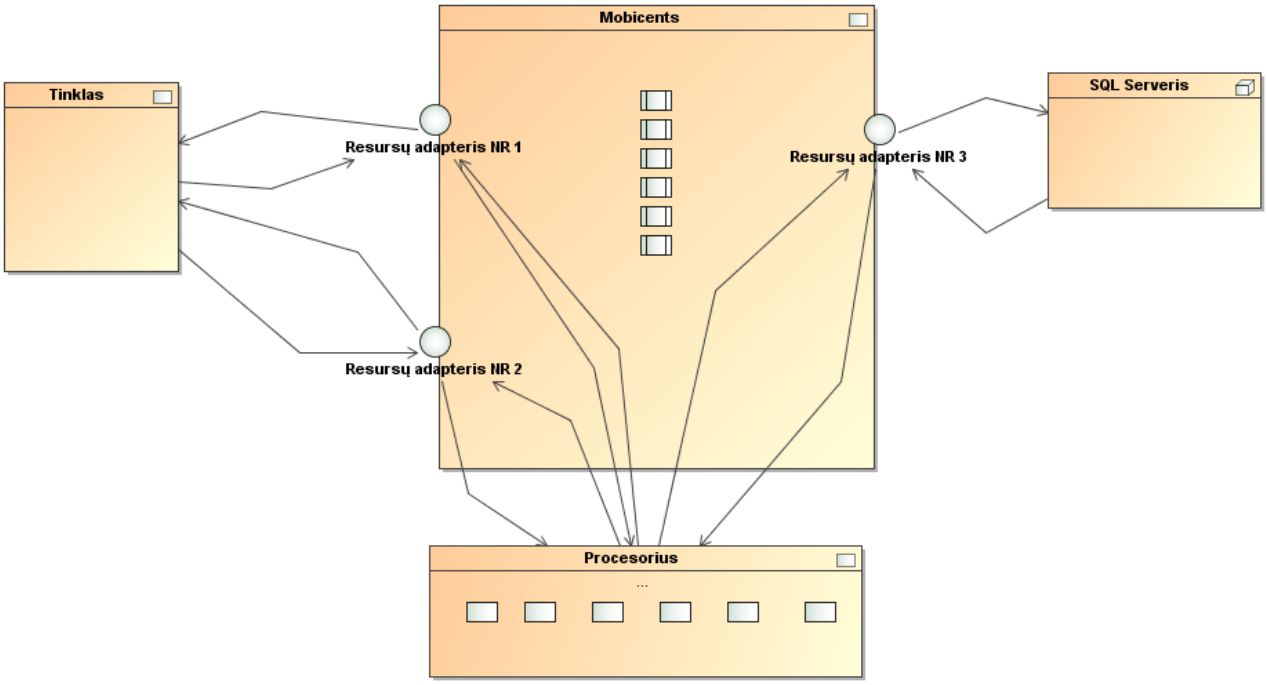
Sėkmingai atlikus eksperimentą, imitacinis modelis automatiškai išsaugoja rezultatus. Duomenys suformuojami XSLT bylos pagalba, kadangi pirminiai duomenys yra XML tipo failas. 10-ajame paveikslėlyje pateikiama rezultatų saugojimo veiklos diagrama.



10 pav. Rezultatų saugojimo veiklos diagrama

3.6. Imitacinio modelio išdėstymo vaizdas

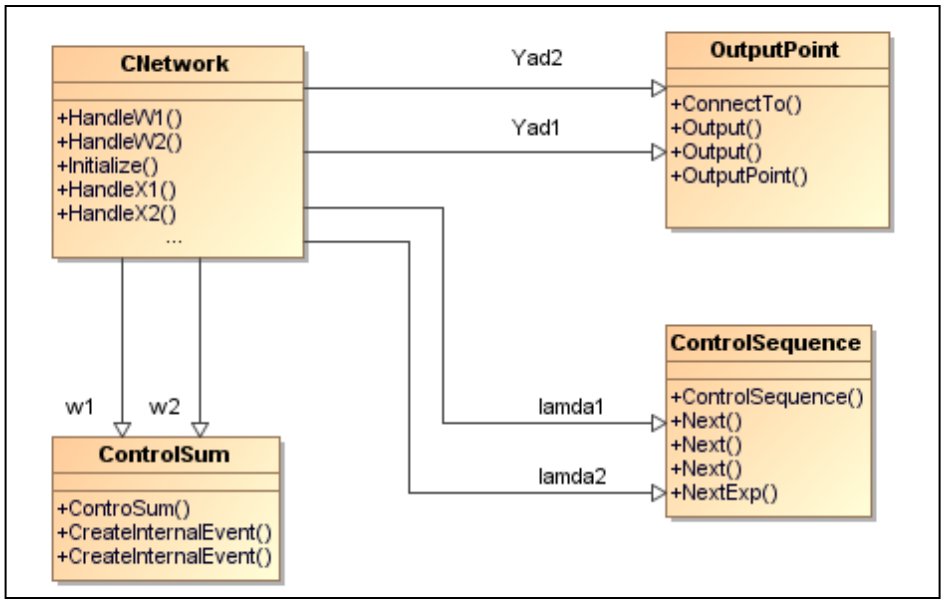
Imituojamą sistemą (žr. 11 pav.) sudaro intelektualusis tinklas iš kurio ateina užklausa, trys resursų adapteriai, kurie žinutę pritaiko sistemai, serveriui, bei duomenų bazei, taip pat Mobicents atviro kodo sistema veikianti pagal JSlee standartą, apdorojanti visas užklausas ir formuojanti jų eiles [11]. SQL serveris duomenims saugoti, bei dviejų branduolių Intel procesorius.



11 pav. Imituojamos sistemos išdėstymo modelis

3.7. Detali sistemos paketų specifikacija

3.7.1. Paketas Network

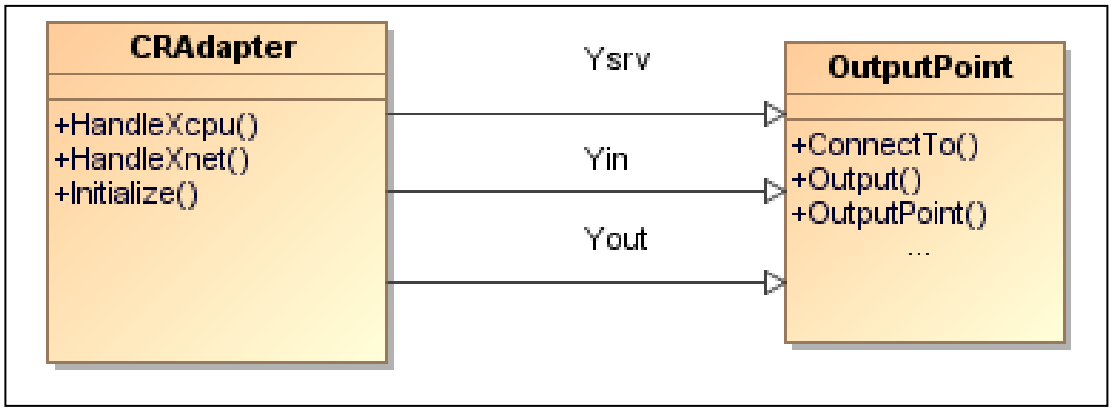


12 pav. Agregato Network klasių diagrama

Agregatas (žr. 12 pav.) turi du išėjimo sąveikos taškas ***Yad1*** ir ***Yad2*** skirtus prijunkti dviejų tipų resursų adapterius. Agregatas leidžia generuoti du paraiškų srautus su eksponentiniu

pasiskirstymo dėsniu. Šio pasiskirstymo parametrai apibrėžiami atitinkamai kintamuosiuose ***p_lamda1*** ir ***p_lamda2***. Vidinių įvykių ***w1*** ir ***w2*** susijusių su šių srautų generavimu ir apdorojimu agregato klasėje apibrėžtos dvi valdančios sekos ***lamda1*** ir ***lamda2***, bei šių įvykių perėjimo-išėjimo operatorių metodai ***HandleW1()*** ir ***HandleW2()***. Taip pat agregate numatyti du įėjimo sąveikos taškai skirti priimti pranešimams iš dviejų skirtingo tipo resursų adapterių. Šių pranešimų apdorojimui skirti metodai ***HandleX1()*** ir ***HandleX2().*** Šie metodai realizuoja agregato išorinių įvykių perėjimo-išėjimo operatorius. Statistinių duomenų rinkimui apie abejų paraiškų srautus skirti ***Output()*** metodai.

3.7.2. Paketas RAdapter

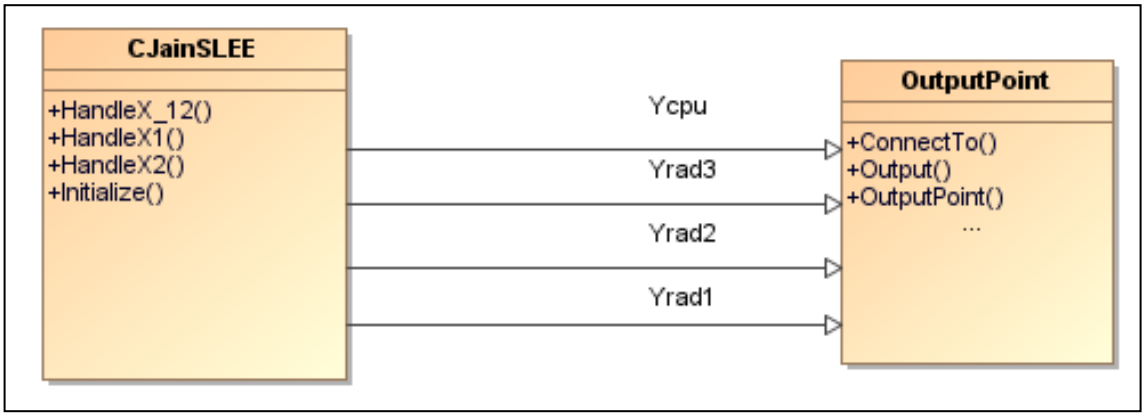


13 pav. Agregato RAdapter klasių diagrama

Agregatas (žr. 13 pav.) turi du išėjimo sąveikos taškai ***Yin*** ir ***You***t skirtus prisijungti prie JainSLEE konteinerio bei ***Ysrv*** - CPU agregatų. Taip pat agregate numatyti du įėjimo sąveikos taškai Xnet ir Xcpu skirti priimti pranešimams iš tinklo bei CPU agregatų. Šių pranešimų apdorojimui skirti metodai ***HandleXnet()*** ir ***HandleXcpu().*** Šie metodai realizuoja agregato išorinių įvykių perėjimo-išėjimo operatorius. Priimtų pranešimų išsaugojimui iki apdorojimo pradžios, kai trūksta skaičiavimo resursų CPU agregate, skirtas buferis ***OutputPoint()***.

Agregato sąveikos taško objekto metodu ***ConnectTo()*** sujungiami sujungimo kanalais į uždarą sistemą. Agregatų sąveika siunčiant pranešimus agregatų sujungimo kanalais realizuojama sąveikos taško objekto metodu ***Output().***

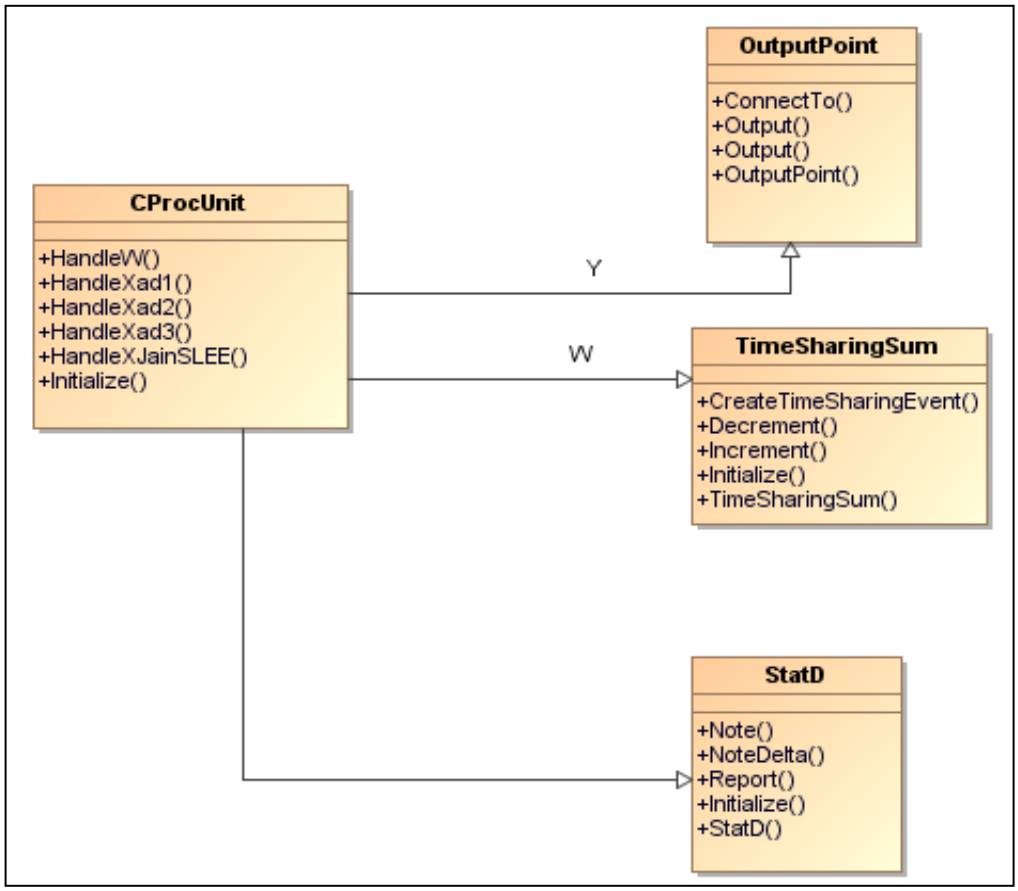
3.7.3. Paketas JainSLEE



14 pav. Agregato JainSLEE klasių diagrama

Agregatas (žr. 14 pav.) turi išėjimo sąveikos taškai ***Yrad1***, ***Yrad2*** ir ***Yrad3*** skirtus prijungti tris skirtingus resursų adapterio bei ***Ycpu*** - CPU agregatus. Taip pat agregate numatyti du įėjimo sąveikos taškai ***Xrad*** ir ***Xcpu*** skirti priimti pranešimams iš resursų adapterių bei CPU agregatų. Šių pranešimų apdorojimui skirti metodai ***HandleX1()*** ir ***HandleX2().*** Šie metodai realizuoja agregato išorinių įvykių perėjimo-išėjimo operatorius. Priimtų pranešimų išsaugojimui iki apdorojimo pradžios, kai trūksta skaičiavimo resursų CPU agregate, skirtas buferis ***OutputPoint.*** Agregatų sąveika siunčiant pranešimus agregatų sujungimo kanalais realizuojama sąveikos taško objekto metodu *Output().* Agregato sąveikos taško objekto metodu ***ConnectTo()*** sujungiami sujungimo kanalais į uždarą sistemą.

3.7.4. Paketas ProcUnit

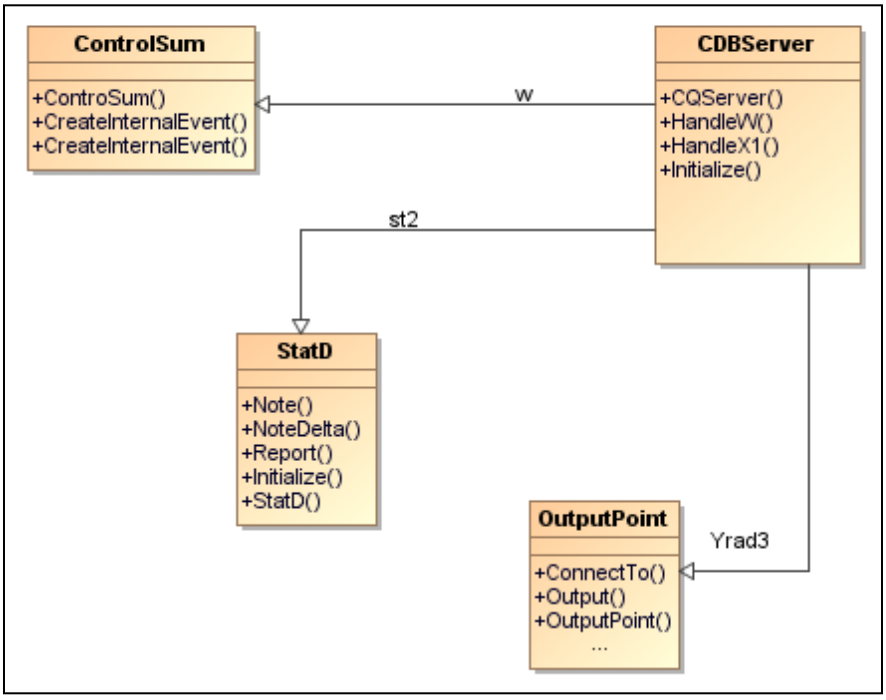


15 pav. Agregato ProcUnit klasių diagrama

Agregatas (žr. 15 pav.) turi išėjimo sąveikos taškų ***Y*** masyvą skirtą prijungti skirtingus resursų adapterių bei JainSLEE konteinerio agregatus. Taip pat agregate numatyti keturi įėjimo sąveikos taškai ***Xrad1, Xrad2, Xrad3*** ir ***Xjslee*** skirti priimti pranešimams iš resursų adapterių bei JainSLEE agregatų. Šių pranešimų apdorojimui skirti metodai ***HandleXad1(), HandleXad2(), HandleXad3()*** ir ***HandleXJainSLEE().*** Vidinių įvykių valdymui apibrėžta valdanti suma ***w*** bei šių įvykių apdorojimui skirtas metodas ***HandleW().***

Priimtų pranešimų išsaugojimui iki apdorojimo pradžios, kai trūksta skaičiavimo resursų CPU agregate, skirtas buferis ***qms***. Statistinių duomenų rinkimui apie resursų adapteryje apdorojamus pranešimus skirtas ***Note()*** ir ***NoteDelta()*** metodas, bei ***StatdD*** skirtas rinkti informaciją apie laukiančių apdorojimo pranešimų eilę realizuotą buferyje, o **Report()** formuoja rezultatų failą xml formatu.

3.7.5. Paketas DBServer



16 pav. Agregato DBServer klasių diagrama

OutputPoint – agregato sąveikos taško klasė. Agregato (žr. 16 pav.) sąveikos taško objekto metodu ***ConnectTo()*** sujungiami sujungimo kanalais į uždarą sistemą. Agregatų sąveika siunčiant pranešimus agregatų sujungimo kanalais realizuojama sąveikos taško objekto metodu ***Output()*.** **CDBServer** klasė realizuoja duomenų bazę su kuria bendrauja agregatas. **HandleW()** bei **HandleX1()** metodai užtikrina korektišką duomenų perdavimą iš ir į duomenų bazę. Statistinių duomenų rinkimui apie DB serveryje apdorojamus pranešimus skirtas ***st2*** objektas realizuotas **StatD** klasėje metodais **Report()**, **StatD()**. Statinių duomenų surinkimui skirtas **Note()** metodas.

Agregato vidinio įvykio generavimą inicijuoja agregato valdančios sumos ***ControlSum*** metodas ***CreateInternalEvent(w)***. Šio metodo parametru ***w*** nustatome laiko momentą kada yra numatomas vidinis įvykis. Sugeneruotas vidinis įvykis patalpinamas į surūšiuotą vidinių įvykių sąrašą ***internalEventQuque***. Rūšiavimas yra atliekamas pagal vidinio įvykio atributą ***w***.

4. TYRIMO DALIS

4.1. Tyrimo uždavinio formulavimas

Darbo 2.8. skyriuje aprašytai problemai:

Kaip adekvačiai sumodeliuoti laiko paskirstymo sistemą (angl. „Time-sharing system“)?

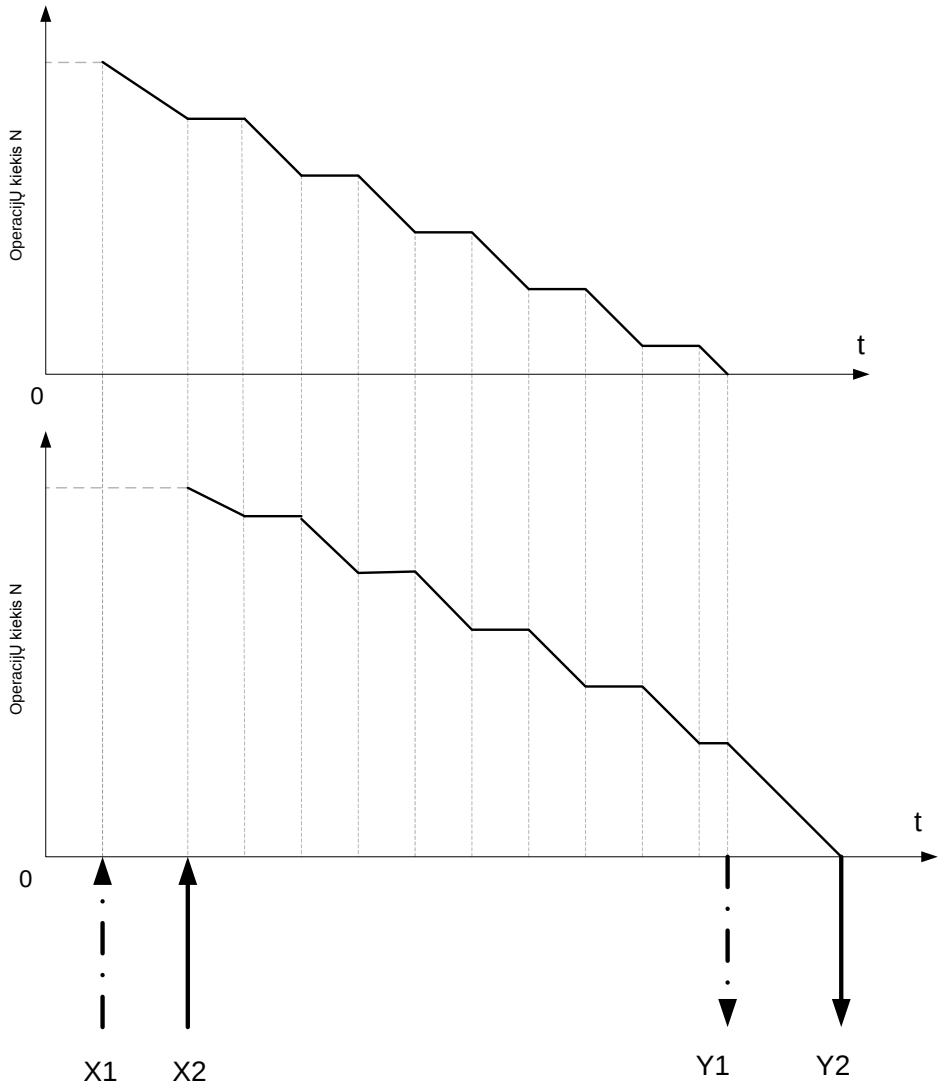
Galimi iškeltos problemos sprendimo būdai:

1) *imitaciniame programinių komponentų funkcionavimo procesorių modulyje aprašyti ir po to realizuoti visus laiko paskirstymo algoritmo įvykius, tačiau tokiu atveju imitacinio modeliavimo eksperimentas gali pareikalauti labai daug operacinių skaičiavimų;*
2) *panaudoti analitinę matematinę abstrakciją prarandant tam tikrą modelio adekvatumą.*

spręsti buvo sudaryti 2 imitaciniai modeliai. Šių skirtingų modelių paaiškinimai pateikti 17 ir 18 paveiksluose.

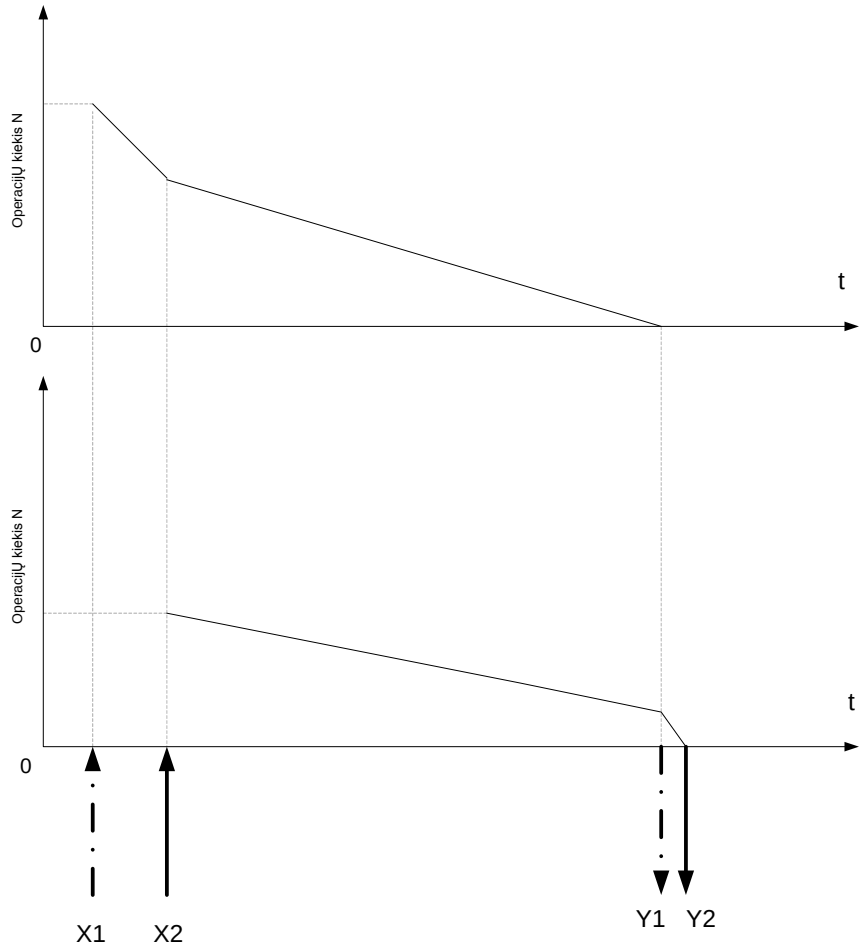
Jei vienas procesoriaus branduolys aptarnauja du procesus, tai jie apdorojami 2 kartus lėčiau, jei aptarnauja tris procesus, tai 3 kartus lėčiau. Ši prielaida visiškai adekvačiai sumodeliuotų laiko paskirstymo sistemą, jei kiekvieno proceso aptarnavimo trukmė būtų kartotinė kvanto dydžiui.

Pagal **pirmąjį** sprendimo būdą 17-ajame paveiksle vaizduojamos dvi tinklo paslaugos patenkančios į sistemą skirtingais laikais, taip pat užimančios skirtingus laikus skaičiavimuose. X1 paslauga į sistemą ateina anksčiau su pradiniu reikalingų operacijų skaičiumi *N*, tada po laiko *t* ateina antroji paslauga. Šiuo laiko momentu pradedamas procesoriaus laiko paskirstymo dėsnio taikymas, t.y. kol X2 paslauga užima procesorių, tol tą patį laiko momentą su X1 paslauga nėra vykdomi skaičiavimai ir yra laukiama X2 paslaugos nustatytos trukmės skaičiavimų pabaigos. Tuomet X2 paslauga pradeda „laukti“ kol X1 baigs skaičiavimus. Y1 pažymėtas pirmos paslaugos baigimo laikas, Y2 – antros. Pasibaigus X1 paslaugos skaičiavimams, procesorius tampa laisvas X2 paslaugai, tuomet šioji paslauga pasibaigia be pertraukimų, matome vienodą paskutinės atkarpos kampą grafike kol operacijų skaičius N netampa 0.



17 pav. Proceso skaičiavimo operacijų kiekis likęs iki proceso apdorojimo pabaigos

Kitu atveju šis procesas modeliuojamas su tam tikra paklaida. Tai galime nustatyti darant eksperimentus **antruoju** imitaciniu modeliu, kuris buvo sukurtas remiantis analitinėmis formulėmis skirtomis modeliuoti laiko paskirstymo sistemą (angl. „time-sharing system“). Tokį procesą galima pavaizduoti grafike (žr. 18 pav.).

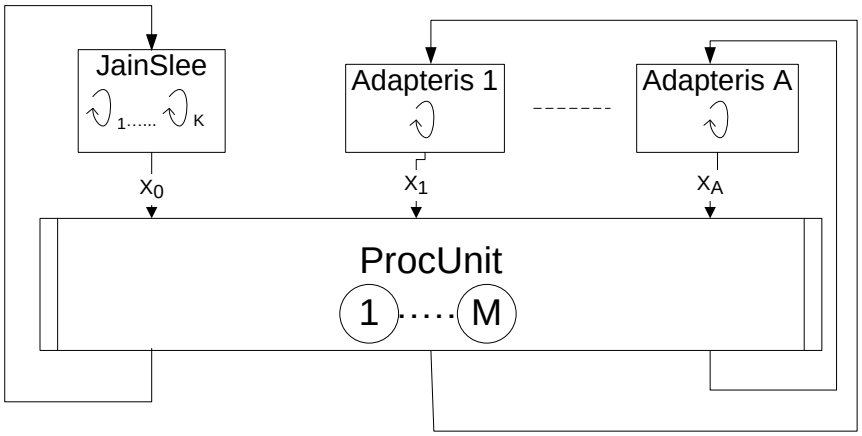


18 pav. Proceso skaičiavimo operacijų kiekis likęs iki apdorojimo pabaigos

Šiuo atveju į sistemą taip pat ateina dvi paraiškos (žr. 18 pav.). X1 patenka su didesniu skaičiavimų kiekiu, bei anksčiau, nei X2 paraiška. Procesoriui užfiksavus naujos paraiškos atėjimą (X2), anksčiau vykęs procesas nėra stabdomas kaip 17 pav. , bet skaičiavimų greitis lėtėja, tai galima spręsti iš grafiko kampo mažėjimo [7].

4.2. ProcUnit agregato sąveikos modelis su JAINSLEE ir resursų adapterių programiniais komponentais

Procesorius modelyje sąveikauja su resursų adapteriais ir JainSlee (žr. 19 pav.). Agregatas ProcUNIT turi išėjimo sąveikos taškų masyvą skirtą prijunkti skirtingus resursų adapterių bei JainSLEE konteinerio agregatus. Taip pat agregate numatyti įėjimo sąveikos taškai ***X1, X2, X3....XA*** ir ***X0*** skirti priimti pranešimams iš resursų adapterių bei JainSLEE agregato.



19 pav. Procesoriaus sąveika su JainSlee ir resursų adapteriais

$\circlearrowleft_1$ -žymimos gijos., (M) - žymimi procesoriaus branduoliai.

4.3. Agregato ProcUnit PLA aprašymas su analitinėmis laiko paskirstymo algoritmo formulėmis

Įėjimų signalų aibė: $X = \{x_0, x_1, x_2 \ldots x_n\}$, $i = \overline{1, A}$, $x_i = \langle st, nr, pr, c, u, o, unr \rangle$,

st – adapterio, per kurį paraiška patenka į sistemą, numeris

nr – paraiškos numeris,

pr – paraiškos išsiuntimo pradžios laikas,

c – paslaugos paraiškos tipas, $c \in \overline{1, l}$, l – paslaugų tipų skaičius,

u – paraiškos užduočių sąrašas, $u \in \{u_1, ..., u_l\}$, čia $u_i = u_{i1}, u_{i2}, u_{i3}, \ldots u_{i,s_i}$, čia s_i $i-tos$ paslaugos užduočių skaičius.

o – paraiškos operacijų skaičiaus sąrašas, $o \in \{o_1, ..., o_l\}$, čia $o_i = o_{i1}, o_{i2}, o_{i3}, \ldots o_{i,s_i}$,

unr – atliktos užduoties numeris.

Išėjimo signalų aibė: $Y = \{y_0, y_1, y_2 \ldots y_n\}$, $i = \overline{1, A}$

$y_i = \langle st, nr, pr, c, u, o, unr \rangle$,

st – adapterio, per kurį paraiška patenka į sistemą, numeris,

nr – paraiškos numeris,

pr – paraiškos išsiuntimo pradžios laikas,

c – paslaugos paraiškos tipas, $c \in \overline{1, l}$, l – paslaugų tipų skaičius,

u – paraiškos užduočių sąrašas, $u \in \{u_1,...,u_l\}$, čia $u_i = u_{i1}, u_{i2}, u_{i3}, \ldots u_{i,s_i}$, čia s_i i-tosios paslaugos užduočių skaičius.

o – paraiškos operacijų skaičiaus sąrašas, $o \in \{o_1,...,o_l\}$, čia $o_i = o_{i1}, o_{i2}, o_{i3}, \ldots o_{i,s_i}$,

unr – atliktos užduoties numeris.

Išorinių įvykių aibė: $E' = \{e'_0, e'_1, e'_2, \ldots e'_n\}$, e'_0 – gautas signalas iš JainSLEE aplinkos; e'_i – gautas įėjimo signalas iš i-to adapterio, $i = \overline{1, A}$.

Parametrai. K – maksimalus procesų skaičius aptarnaujantis JainSLEE, A – adapterių skaičius, M – branduoliu skaičius.

Vidinių įvykių aibė: $E'' = \{e''_{11}, e''_{11}, \ldots e''_{1,A}; e''_{21}, e''_{22}, \ldots e''_{2K}\}$, čia $w(e''_{1i}, t_m)$ – i-tojo adapterio užduoties vykdymo pabaiga; $w(e''_{2j}, t_m)$ – j-tojo JainSLEE aplinkos proceso vykdymo pabaiga.

Tolydžioji komponentė:

$z_v(t_m) = \langle w(e''_{11}, t_m), w(e''_{12}, t_m), \ldots w(e''_{1A}, t_m); w(e''_{21}, t_m), w(e''_{22}, t_m), \ldots w(e''_{2K}, t_m) \rangle$.

Diskrečioji komponentė:

$v(t_m) = \langle l(t_m), par_{11}(t_m), par_{12}(t_m), \ldots par_{1A}(t_m); par_{21}(t_m), par_{22}(t_m) par_{2K}(t_m) \rangle$, čia $par_{ij}(t_m)$ – ij proceso parametrai laiko momentu t_m, $l(t_m)$ – aktyvių procesų skaičius.

Pradinė būsena: $l(t_0) = 0$, $par_{ij}(t_0) = \langle\ \rangle$.

Perėjimo operatoriai:

$H(e'_0)$:

Pažymėkime $k - \min(i | w(e''_{2i}, t_{m-1}) < t_m)$.

$par_{1k}(t_m) = x_i$,

$l(t_m) = l(t_{m-1}) + 1$,

$$w(e''_{1j}, t_m) = \begin{cases} t_m + \dfrac{(w(e''_{1j}, t_{m-1}) - t_m)\beta(t_{m-1})}{\beta(t_m)}, & w(e''_{1j}, t_{m-1}) > t_m, \\ w(e''_{1j}, t_{m-1}), & w(e''_{1j}, t_{m-1}) < t_m, \end{cases} \quad (1)$$

$$w\left(e''_{2j},t_m\right)=\begin{cases} t_m+\dfrac{o_{unr}M}{l\left(t_m\right)}, & j=k, \\ t_m+\dfrac{\left(w\left(e''_{2j},t_{m-1}\right)-t_m\right)\beta\left(t_{m-1}\right)}{\beta\left(t_m\right)}, & j\neq k\wedge w\left(e''_{2j},t_{m-1}\right)>t_m \\ w\left(e''_{2j},t_{m-1}\right), & j\neq k\wedge w\left(e''_{2j},t_{m-1}\right)<t_m, \end{cases} \quad (2)$$

čia $x_0=\left\langle st,nr,pr,c,u,o,unr\right\rangle$, $\beta\left(t\right)=\begin{cases} 1, & l\left(t\right)\leq M, \\ \dfrac{M}{l\left(t\right)}, & l\left(t\right)>M. \end{cases}$

$H\left(e''_{2i}\right)$:

$par_{1k}\left(t_m\right)=\left\langle\ \right\rangle$,

$l\left(t_m\right)=l\left(t_{m-1}\right)-1$,

$$w\left(e''_{1j},t_m\right)=\begin{cases} t_m+\dfrac{\left(w\left(e''_{1j},t_{m-1}\right)-t_m\right)\beta\left(t_{m-1}\right)}{\beta\left(t_m\right)}, & w\left(e''_{1j},t_{m-1}\right)>t_m, \\ w\left(e''_{1j},t_{m-1}\right), & w\left(e''_{1j},t_{m-1}\right)<t_m, \end{cases} \quad (3)$$

$$w\left(e''_{2j},t_m\right)=\begin{cases} t_m+\dfrac{\left(w\left(e''_{2j},t_{m-1}\right)-t_m\right)\beta\left(t_{m-1}\right)}{\beta\left(t_m\right)}, & w\left(e''_{2j},t_{m-1}\right)>t_m, \\ w\left(e''_{2j},t_{m-1}\right), & w\left(e''_{2j},t_{m-1}\right)<t_m, \end{cases}, \quad (4)$$

čia $\beta\left(t\right)=\begin{cases} 1, & l\left(t\right)\leq M, \\ \dfrac{M}{l\left(t\right)}, & l\left(t\right)>M. \end{cases}$

$G\left(e''_{2i}\right)$:

$Y=\left\{y_0\right\}$,čia $y_i=par_{2i}\left(t_{m-1}\right)$.

$H\left(e'_i\right)$:

$par_{1i}\left(t_m\right)=x_i$,

$l\left(t_m\right)=l\left(t_{m-1}\right)+1$,

$$w\left(e''_{1j},t_m\right)=\begin{cases} t_m+\dfrac{o_{unr}M}{l\left(t_m\right)} & i=j, \\ t_m+\dfrac{\left(w\left(e''_{1j},t_{m-1}\right)-t_m\right)\beta\left(t_{m-1}\right)}{\beta\left(t_m\right)}, & i\neq j\wedge w\left(e''_{1j},t_{m-1}\right)>t_m, \\ w\left(e''_{1j},t_{m-1}\right), & i\neq j\wedge w\left(e''_{1j},t_{m-1}\right)<t_m, \end{cases} \quad (5)$$

$$w(e''_{2j}, t_m) = \begin{cases} t_m + \dfrac{(w(e''_{2j}, t_{m-1}) - t_m)\beta(t_{m-1})}{\beta(t_m)}, & w(e''_{2j}, t_{m-1}) > t_m, \\ w(e''_{2j}, t_{m-1}), & w(e''_{2j}, t_{m-1}) < t_m, \end{cases} \quad (6)$$

čia $x_i = \langle st, nr, pr, c, u, o, unr \rangle$, $\beta(t) = \begin{cases} 1, & l(t) \le M, \\ \dfrac{M}{l(t)}, & l(t) > M. \end{cases}$

$H(e''_{1i})$:

$par_{1i}(t_m) = \langle\ \rangle$,

$l(t_m) = l(t_{m-1}) - 1$,

$$w(e''_{1j}, t_m) = \begin{cases} t_m + \dfrac{(w(e''_{1j}, t_{m-1}) - t_m)\beta(t_{m-1})}{\beta(t_m)}, & w(e''_{1j}, t_{m-1}) > t_m, \\ w(e''_{1j}, t_{m-1}), & w(e''_{1j}, t_{m-1}) < t_m, \end{cases} \quad (7)$$

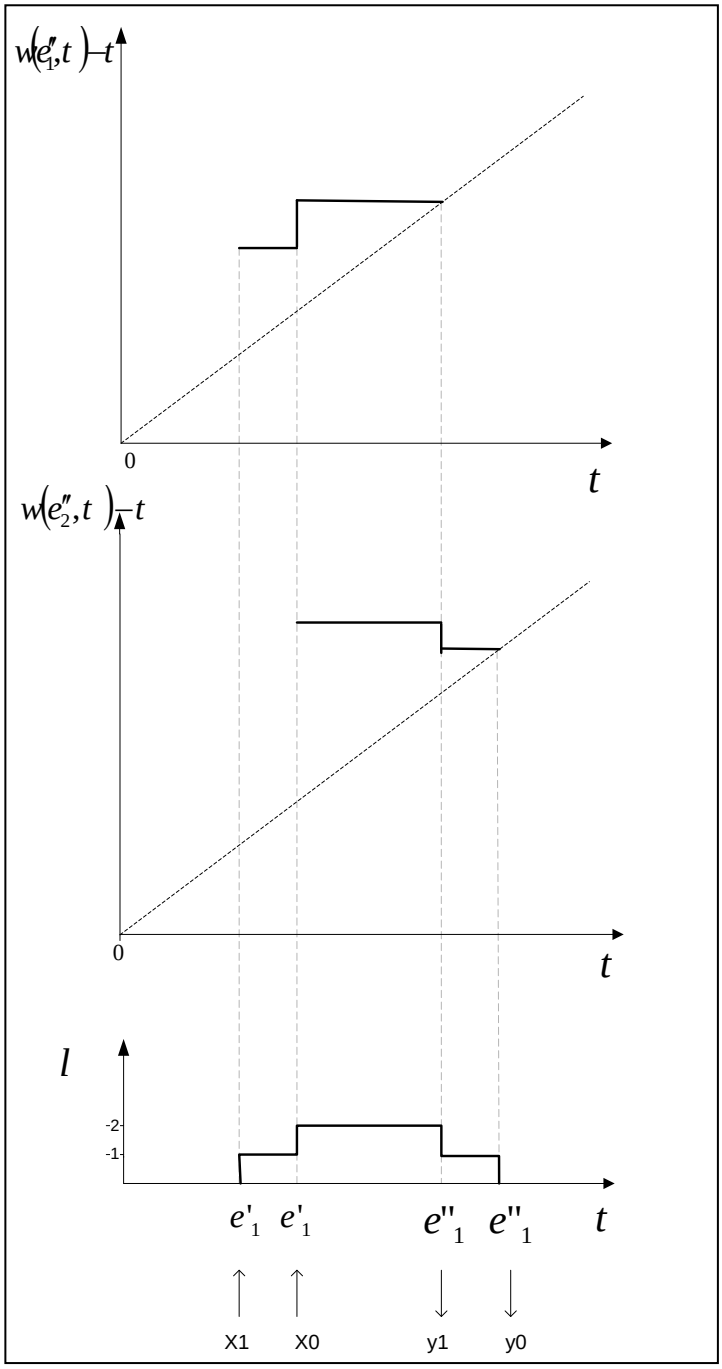
$$w(e''_{2j}, t_m) = \begin{cases} t_m + \dfrac{(w(e''_{2j}, t_{m-1}) - t_m)\beta(t_{m-1})}{\beta(t_m)}, & w(e''_{2j}, t_{m-1}) > t_m, \\ w(e''_{2j}, t_{m-1}), & w(e''_{2j}, t_{m-1}) < t_m, \end{cases} \quad (8)$$

čia $\beta(t) = \begin{cases} 1, & l(t) \le M, \\ \dfrac{M}{l(t)}, & l(t) > M. \end{cases}$

$G(e''_{1i})$:

$Y = \{y_i\}$,čia $y_i = par_{1i}(t_{m-1})$.

Modeliuojami įvykiai e_1, e_2, e_3. Formulių (1-8) paaiškinimai pateikiami grafiškai nagrinėjant atvejį, kai M=1, K=1, A=1 (žr. 20 pav.).



20 pav. Procesų pabaigos priklausomybės nuo laiko

5. EKSPERIMENTINĖ DALIS

Pagal atkarpomis – tiesinių agregatų aprašymus buvo programiškai realizuoti du skirtingo tipo imitaciniai modeliai:

- adekvačia įvykių seką modeliuojantis algoritmas;
- analitinėmis formulėmis grįstas imitacinis modelis.

5.1. Analitinėmis matematinėmis formulėmis grįstas laiko paskirstymo algoritmo imitacinis modelis

Eksperimentinei analizei buvo sukurti du imitaciniai modeliai. Norint išspręsti 2.9. skyrelyje aprašytą problemą, eksperimentai buvo atliekami naudojant abu imitacinis modelius su vienodomis pradinėmis charakteristikomis:

2-asis imitacinis modelis (žr. 21 pav.). Šio modelio kūrimas buvo paremtas 4.3. skyriuje aprašytomis laiko paskirstymo dėsnio modeliavimo analitinėmis formulėmis.

C1 (procesorių kiekis) – 1.0

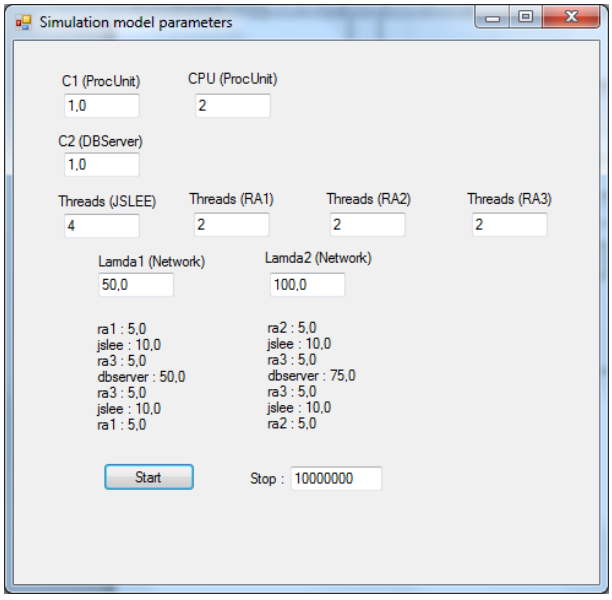
CPU (branduolių kiekis procesoriuje) – 2

Threads (gijų skaičius agregate) – JSLEE=4, RA1(resursų adapteris nr.1)= 2, RA2 (resursų adapteris nr. 2) = 2, bei RA3 (resursų adapteris nr. 3) = 2.

Lamda1 (pirmosios paraiškos atėjimo į sistemą pradinis laikas) = 50,00

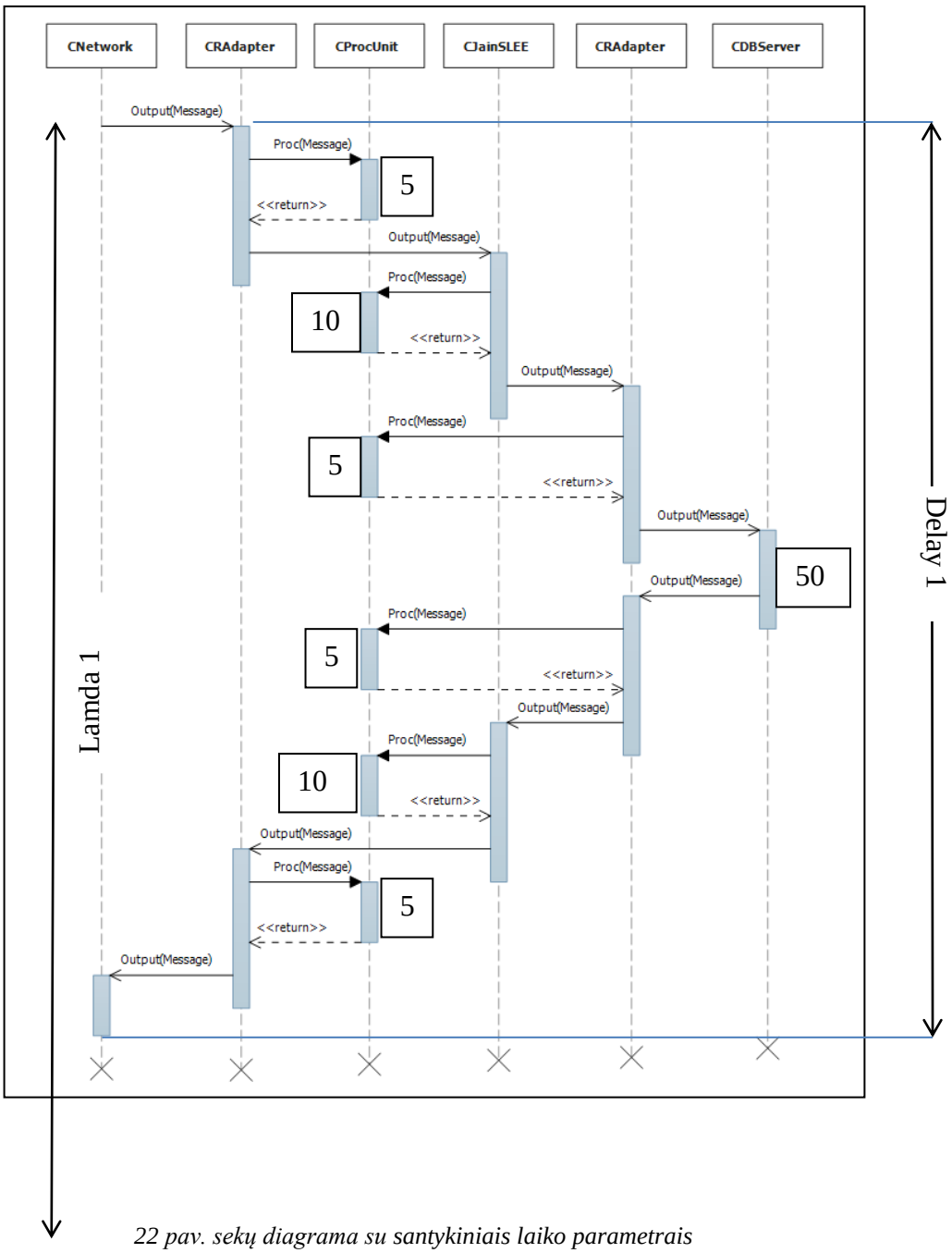
Lamda2 (antrosios paraiškos atėjimo į sistemą pradinis laikas) = 75,00

Stop (eksperimento imituojamas pabaigos laikas) = 10000000 laiko vnt.



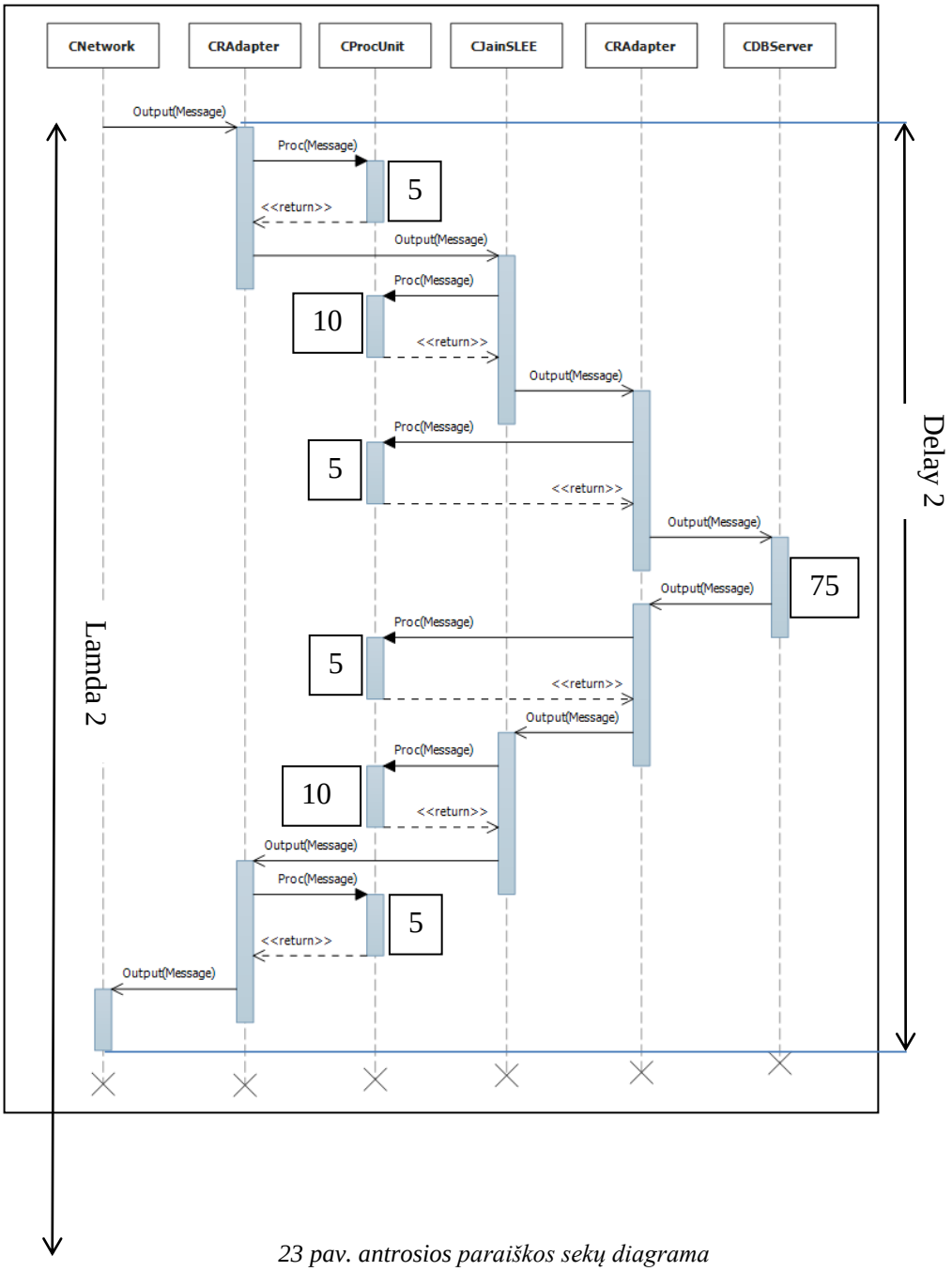
21 pav. Analitinėmis formulėmis grįstas algoritmas

Klasių sekų diagramoje (žr. 22 pav.) atsispindi visų modelyje sąveikaujančių klasių nuoseklus paraiškos atėjusios iš *CNetwork* apdorojimas. Lamda 1 kuris aprašytas šiame skyrelyje vaizduoja jog sekanti pirmo tipo paraiška pateks į sistemą kai bus aptarnauta pirmoji. Taip pat paveiksle matome laikus pažymėtus kvadratais prie atitinkamų agregatų. CDBServer reikalauja daugiausiai santykinių laiko vienetų apdoroti paraišką – 50. Tai įtakoja skaičiavimų pabaigos laiką DELAY 1 pažymėtą vertikalia ašimi per visus agregatus. Tai kurio reikia, kad aptarnauti vieną paraišką, t.y. laikas nuo atėjimo į sistemą ir išėjimo agregate *CNetwork*.



22 pav. sekų diagrama su santykiniais laiko parametrais

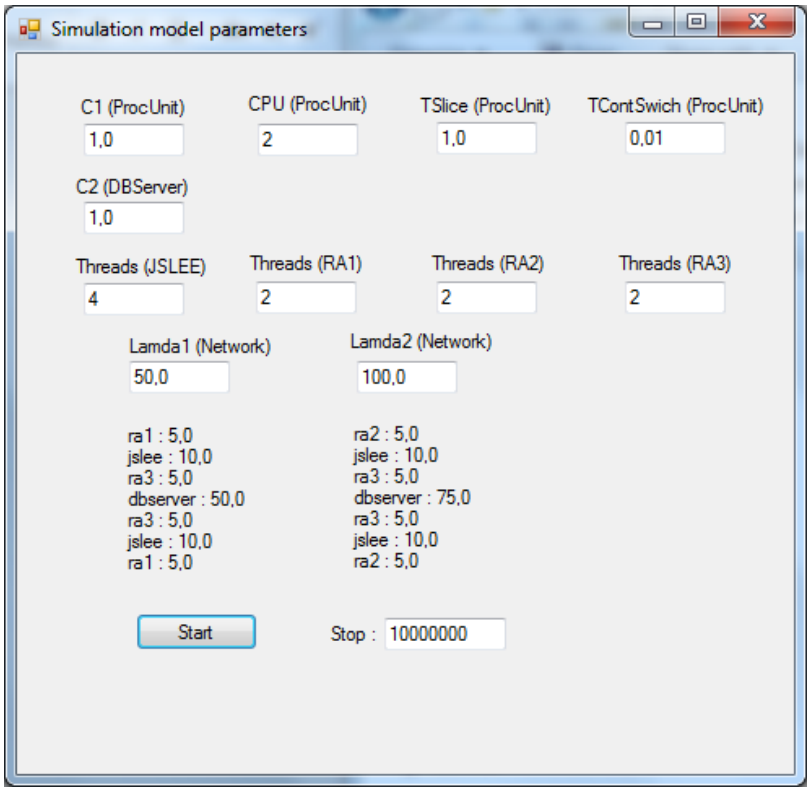
Kaip buvo aprašyta dokumente anksčiau, sistema modeliuojama su dviejų paslaugų tipais, todėl antroji paslauga (pagal 23 pav.) ateis į sistemą kas 100 santykinių laiko vienetų. Agregatų reikalaujami laikai vienodi kaip ir pirmos paraiškos, skiriasi tik CDBServer santykinis laikas – 75. Šioji paslauga reikalaus daugiau resursų duomenų bazės agregate.



23 pav. antrosios paraiškos sekų diagrama

5.2. Laiko paskirstymo algoritmo įvykių seką adekvačiai modeliuojantis imitacinis modelis

Pirmasis imitacinis modelis (žr. 24 pav.). Šiame imitaciniame modelyje pradinės charakteristikos buvo pasirinktos vienodos lyginant su analitiniu modeliu. Atsiranda naujas parametras TSlice – tai ProcUnit agregato parametras nusakantis kokio dydžio kvantais suskirstomas eksperimentas, t.y. į kokį santykinį laiko vienetą turėtų tilpti vieno proceso apdorojimas.



24 pav. Pirmasis imitacinis modelis

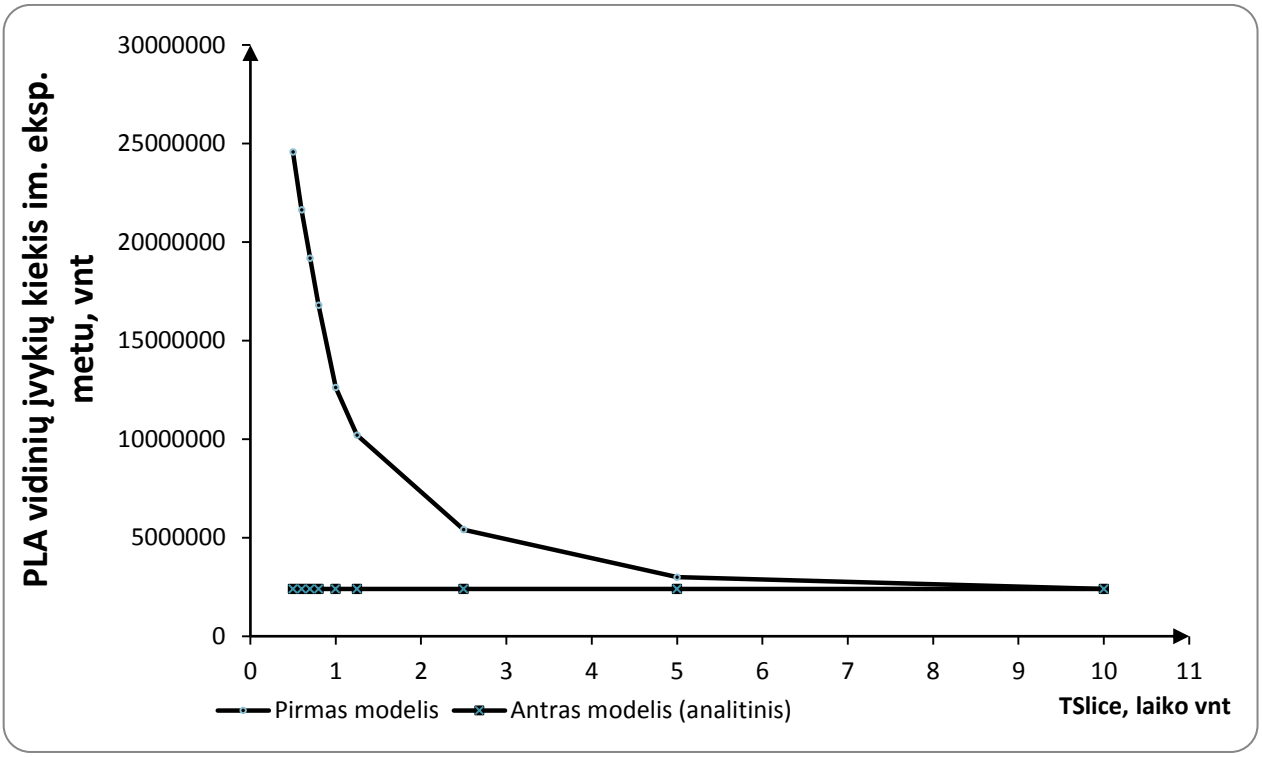
5.3. Pirmojo ir antrojo imitacinių modelių eksperimentas

Eksperimentas su analitiniu imitaciniu modeliu buvo vykdytas vieną kartą, kadangi parametrai nebuvo keičiami, todėl ir rezultatas yra vienas. Rezultate buvo gauti laikai reikalingi apdoroti vieną ir kitą paraiškas, vėlavimai, apkrovimų dydžiai. Eksperimento rezultatai pateikiami prieduose.

Vykdant eksperimentus su pirmajai problemai spęsti sudarytu modeliu (24 pav.) buvo atliekami keli bandymai, tam kad rezultatai būtų kuo tikslesni. Visos svarbiausios rezultatų reikšmės atidėtos grafike (žr. 25 pav.) pagal rezultatus gautus su imitaciniu modeliu (žr. 1 lentelė).

1 lentelė. TSlice priklausomybės nuo vidinių įvykių

Tslice	Vidiniai įvykiai	Vidiniai įvykiai 2 modelyje
0,5	24577903	2396016
0,6	21627813	2396016
0,7	19179925	2396016
0,8	16800521	2396016
1	12620146	2396016
1,25	10210268	2396016
2,5	5407588	2396016
5	3000535	2396016
10	2399760	2396016



25 pav. Skaičiavimo operacijų kiekio kitimas pirmame ir antrame modeliuose

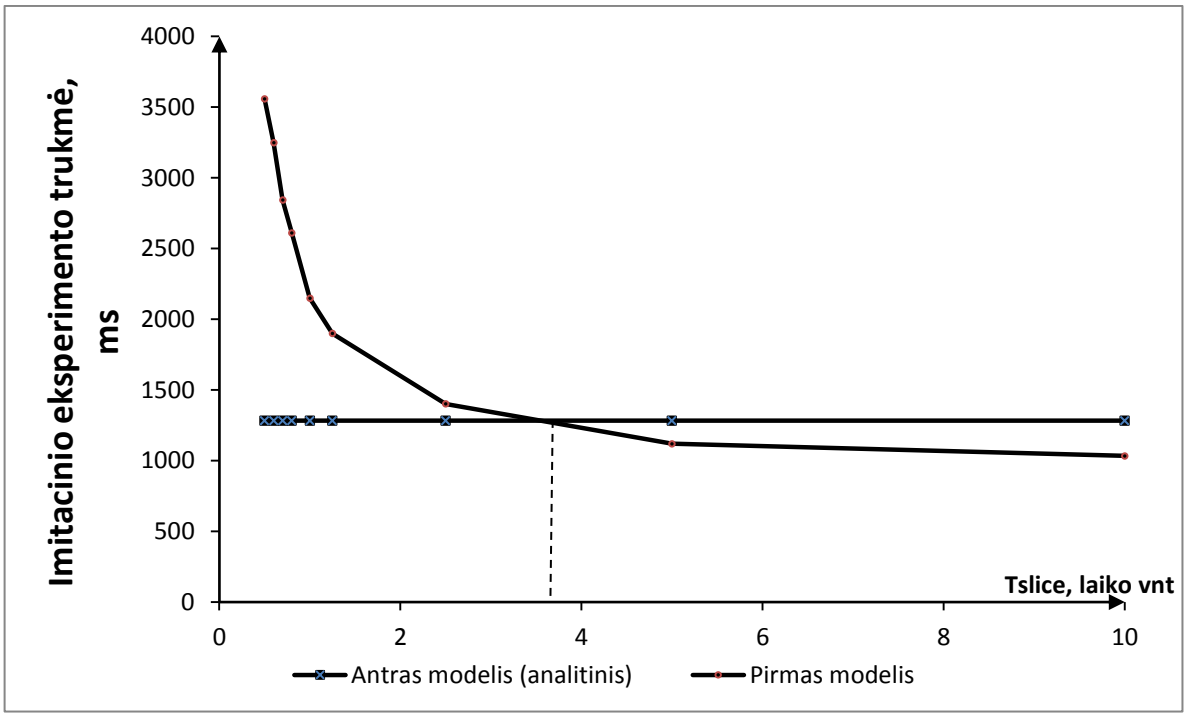
Pirmojo imitacinio modelio skaičiavimo operacijų kiekis žymiai išsauga kai TSlice kvanto trukmė yra bent **2 kartus mažesnė** už gijo proceso trukmę apdorojant ProcUnit. Kadangi eksperimentas su antruoju modeliu buvo vykdytas vieną kartą, jo grafikas vaizduojamas kaip tiesė. Iš grafike pateiktų rezultatų galima formuoti išvadą , jog pagal PLA vidinių įvykių kiekį

eksperimento metu reikalingą apdoroti paraiškai geriau yra naudoti antrąjį (analitinį) imitacinį modelį, kuriam reikia žymiai mažiau operacijų.

Vykdant eksperimentą su abiem modeliais buvo gauti imitacinio eksperimento priklausomybės nuo TSlice laiko kvanto dydžio (žr. 2 lentelė).

***2 lentelė**. Eksperimento trukmės priklausomybė nuo TSlice*

Tslice, laiko vienetai	Eksperimento trukmė	Eksp. trukmė 2 modelyje
0,5	3558	1283
0,6	3249	1283
0,7	2844	1283
0,8	2611	1283
1	2149	1283
1,25	1899	1283
2,5	1401	1283
5	1120	1283
10	1033	1283



26 pav. Eksperimento trukmės kitimas pirmame ir antrame modeliuose

26 pav. pateikiami imitacinio eksperimento trukmės priklausomybės nuo *TSlice* santykinio laiko kvanto. Iš šio grafiko galima daryti išvadą, jog **pirmasis imitacinis modelis** yra naudingesnis, kadangi imitacinio modelio eksperimento trukmė yra nemažesnė kaip pusė proceso

trukmės. Tai galima paaiškinti, jog pirmasis imitacinis modelis vidinių įvykių apdorojimo matematinės išraiškos yra žymiai paprastesnės nei antrame modelyje.

6. IŠVADOS

1. Pasirinktas PLA (angl. „Piece-Linear Aggregate“) specifikavimo būdas leido programiškai realizuoti du skirtingus imitacinius laiko paskirstymo algoritmo modelius:

    1) kai algoritmas modeliuojamas adekvačia įvykių seką;

    2) kai algoritmas modeliuojamas analitinėmis matematinėmis formulėmis.

2. Atlikti eksperimentiniai tyrimai su imitaciniais modeliais parodė, jog galima naudoti tiek pirmąjį, tiek antrąjį imitacinius modelius, kadangi jų statistiniai modeliavimo rezultatai sutampa modeliavimo paklaidos ribose.

3. Pagal imitacinio eksperimento PLA vidinių įvykių kiekį, reikalingą sumodeliuoti algoritmo funkcionavimą, optimalesnis yra antrasis (analitinis) imitacinis modelis, kurio realizacija reikia mažesnio imitacinio modeliavimo įvykių skaičiaus.

4. Pirmasis imitacinis modelis yra naudingesnis, kai laiko paskirstymo algoritmo kvanto trukmė yra nemažesnė kaip pusė modeliuojamo proceso trukmės, kadangi tokiu atveju imitacinio eksperimento vykdymo laikas yra trumpesnis už antrojo algoritmo imitacinio eksperimento vykdymo laiką.

7. LITERATŪRA

[1] **Nacionalinis Inžinerijos Konsorciumas**. *Intelektualiojo telekomunikacinio tinklo aprašas* (Intelligent Network(IN)). [Žiūrėta 2011.03.21] Prieiga per internetą: http://www.iec.org/online/tutorials/acrobat/in.pdf

[2] **Igor Faynberg , Lawrence R. Gabuzda, Marc P. Kaplan, Nittin J. Shah**. The Intelligent Network Standards: Their Application to Services. *McGraw-Hill Professional*, 1997.

[3] **The Moriana Group**. *SDP 2.0. Service Delivery Platforms in the Web 2.0 Era*, 2008, 1- 36 psl.

[4] **H. Pranevičius**. „*Sudėtingų sistemų formalizavimas ir analizė*“. Kauno Technologijos Universitetas, 2008. ISBN 978-9955-591-51-1.

[5] **Xiaodong Xian, Weiren Shi and He Huang**. College of Automation, Chongqing University. *Comparison of OMNET++ and Other Simulator for WSN Simulation.* [Žiūrėta 2011.03.13]. Prieiga per internetą: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4582757&isnumber=4582468

[6] **Guosong Tian; Fidge, C.; Yu-Chu Tian**. Hybrid system simulation of computer control applications over communication networks. *Sch. of Inf. Technol., Queensland Univ. of Technol., Brisbane, QLD, Australia , 2009*

[7] **Giambiasi N., Escude B., Ghosh S.** GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. *Proceedings. 5th International Symposium on Autonomous Decentralized Systems.2001.*

[8] **Rich Wolski, Neil Spring, Jim Hayes**. Predicting the CPU Availability of Time-shared Unix Systems on the Computational Grid. *Indian Institute of Technology, Kharagpur, 2000*

[9] **Dennis M. Ritchie**. *The UNIX Timesharing System_A Retrospective***.** Bell Laboratories Murray Hill, New Jersey, 2007.

[10] **Igor Faynberg , Lawrence R. Gabuzda, Marc P. Kaplan, Nittin J. Shah**. *The Intelligent Network Standards: Their Application to Services. McGraw-Hill Professional, 1997.*

[11] **Mohsen Guizani, Ammar Rayes, Bilal Khan, Ala Al-Fuqaha**. *Network modeling and simulation. A practical perspective*. John Wiley & Sons Ltd. UK, 2010.

[12] **Frank L. Severance**. *System modeling and simulation.*Western michigan university, 2009.

8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

- XSLT - (angl. eXtensible Stylesheet Language (XSL) transformations) – kalba, aprašanti XML dokumento transformaciją į HTML dokumentą arba į kitokios struktūros XML dokumentą.
- PLA – (angl. „piece-linear agregate“) atkarpomis-tiesinių agregatų formalizavimo metodas. Matematinis formalus sistemos aprašymas bus grįstas PLA metodu. Šio metodo privalumas, kad jis leidžia vieningo formalaus aprašymo bazėje kurti specifikuojamos sistemos imitacinius modelius bei formaliai atlikti sudaromo modelio teisingumą.
- Objektinė imitacinio modeliavimo biblioteka PLASim – KTU Verslo informatikos katedroje sukurta modeliavimo biblioteka. Bus naudojamasi šia paprograme norint palengvinti ir paspartinti imitacinio modeliavimo programos kūrimą. Imitacinio modelio PLA specifikaciją sudaro dvi bazines komponentes – agregatų sujungimo schemą bei sujungimo schemoje dalyvaujančių agregatų specifikacijas.
- SDP – (angl. „Service Delivery Platform“) – paslaugų teikimo platforma. Kuriamos platformos veikimas bus tiriamas šiuo imitaciniu modeliu kuris yra kuriamas projekto eigoje.
- IN – angl. *Intelligent network.”* Intelektualusis tinklas.”
- JSLEE - angl. *Service Logic Execution Environment (SLEE) for the Java Platform*

9. PRIEDAI

Priedas Nr.1
PIRMASIS IMITACINIS MODELIS

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					
Delay1	200070	45,06149	449,7883	98,17637	36,17586
Delay2	99659	45,06276	437,5651	109,8381	38,13614
SimApp.CRAdapter					
K_ra1	769568	0	2	0,2942672	0,5431446
Q_ra1	30716	0	7	0,006826 15	0,09437105
Delay_in_ra1	200072	5,01	95,75443	7,472847	3,978124
Delay_out_ra1	200070	5,01	89,40092	6,393947	3,111479
SimApp.CRAdapter					
K_ra2	394586	0	2	0,1474647	0,3872931
Q_ra2	4052	0	3	0,000825 27	0,03006502
Delay_in_ra2	99660	5,01	49,84149	7,496036	3,759976
Delay_out_ra2	99659	5,01	42,33935	6,598188	2,993177
SimApp.CRAdapter					
K_ra3	1089568	0	2	0,4410014	0,6586263
Q_ra3	109354	0	10	0,033759 34	0,2547895
Delay_in_ra3	299731	5,01	108,7869	6,783008	4,251553
Delay_out_ra3	299730	5,01	103,5525	6,995668	4,300453
SimApp.CJainSLEE					
K_jslee	1163194	0	4	0,8910058	1,050438
Q_jslee	35728	0	9	0,015913 94	0,1769295
SimApp.CDBServer					
K_db	402133	0	2	1,02299	0,8085895
Delay_db	299730	5,000369	206,6888	40,96349	22,9647
Q_db	197328	0	9	0,2048095	0,5933362

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =21627813 Run time =3249 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					

Delay1	200914	45,06019	473,7676	98,51354	36,41193
Delay2	99470	45,06469	412,6635	110,3426	38,17297
		SimApp.CRAdapter			
K_ra1	771758	0	2	0,2988392	0,5478792
Q_ra1	31902	0	6	0,00735969	0,09914397
Delay_in_ra1	200916	5,01	84,89765	7,544219	4,118416
Delay_out_ra1	200914	5,01	79,6	6,444521	3,23565
		SimApp.CRAdapter			
K_ra2	393723	0	2	0,1491104	0,3898455
Q_ra2	4162	0	4	0,00087233	0,03109952
Delay_in_ra2	99472	5,01	52,81339	7,568142	3,842434
Delay_out_ra2	99470	5,01	58,22421	6,645467	3,103927
		SimApp.CRAdapter			
K_ra3	1087597	0	2	0,4467567	0,6641163
Q_ra3	113950	0	11	0,03707613	0,2763367
Delay_in_ra3	300387	5,01	122,2044	6,862794	4,475642
Delay_out_ra3	300386	5,01	121,4548	7,069391	4,486127
		SimApp.CJainSLEE			
K_jslee	1169622	0	4	0,8858985	1,041103
Q_jslee	31924	0	8	0,01413016	0,1637001
		SimApp.CDBServer			
K_db	400865	0	2	1,025501	0,8095959
Delay_db	300386	5,000189	224,4169	41,17478	23,10406
Q_db	199908	0	9	0,2113347	0,6087791

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =19179925 Run time =2844 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
		SimApp.CNetwork			
Delay1	199836	45,06019	452,1043	98,21853	35,88967
Delay2	99849	45,06767	442,7917	109,8234	37,57677
		SimApp.CRAdapter			
K_ra1	768167	0	2	0,2960714	0,5453539
Q_ra1	31182	0	5	0,00713366	0,09731489
Delay_in_ra1	199838	5,01	68,8819	7,515501	4,043966
Delay_out_ra1	199836	5,01	66,78318	6,418605	3,175931
		SimApp.CRAdapter			
K_ra2	395346	0	2	0,1487226	0,3890527
Q_ra2	4050	0	3	0,00082176	0,03000226

Delay_in_ra2	99849	5,01	46,38674	7,514169	3,754558
Delay_out_ra2	99849	5,01	51,6	6,621934	3,057653
			SimApp.CRAdapter		
K_ra3	1088383	0	2	0,4434425	0,6611357
Q_ra3	110362	0	12	0,03501882	0,2682382
Delay_in_ra3	299686	5,01	132,8992	6,811438	4,356941
Delay_out_ra3	299686	5,01	144,1767	7,018729	4,389199
			SimApp.CJainSLEE		
K_jslee	1169208	0	4	0,8785257	1,033495
Q_jslee	29538	0	7	0,01242826	0,1492123
			SimApp.CDBServer		
K_db	399830	0	2	1,02445	0,8083627
Delay_db	299686	5,000186	224,653	41,28256	23,22658
Q_db	199542	0	11	0,2127305	0,6135589

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =16800521 Run time =2611 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
			SimApp.CNetwork		
Delay1	200253	45,06084	463,0081	97,41523	35,05724
Delay2	99754	45,07489	434,9348	109,016	37,17725
			SimApp.CRAdapter		
K_ra1	770266	0	2	0,2949517	0,5442039
Q_ra1	30754	0	6	0,00680684	0,09389821
Delay_in_ra1	200257	5,01	73,15671	7,468008	3,971627
Delay_out_ra1	200253	5,01	66,72	6,399182	3,121502
			SimApp.CRAdapter		
K_ra2	394916	0	2	0,1477103	0,3879309
Q_ra2	4102	0	4	0,0008268	0,03070128
Delay_in_ra2	99755	5,01	67,56613	7,474365	3,723329
Delay_out_ra2	99754	5,01	64,05	6,618463	3,035723
			SimApp.CRAdapter		
K_ra3	1090650	0	2	0,4410357	0,6592395
Q_ra3	109386	0	11	0,03438377	0,2652821
Delay_in_ra3	300010	5,01	118,8	6,779359	4,290481
Delay_out_ra3	300007	5,01	119,3985	6,986119	4,317986
			SimApp.CJainSLEE		
K_jslee	1171917	0	4	0,871998	1,025738
Q_jslee	28120	0	9	0,0117876	0,1484898

SimApp.CDBServer					
K_db	402188	0	2	1,023008	0,8083916
Delay_db	300008	5,0002	264,24 24	40,95778	22,92453
Q_db	197830	0	9	0,205769 6	0,5966655

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =12620146 Run time =2149 ms

12620146 2149 1

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					
Delay1	200310	45,06482	392,00 61	96,81886	34,15008
Delay2	100168	45,06069	387,62 36	108,4915	36,43078
SimApp.CRAdapter					
K_ra1	772440	0	2	0,288745 5	0,5368377
Q_ra1	28802	0	6	0,006085 98	0,08735403
Delay_in_ra1	200311	5,01	70,523 38	7,316392	3,733278
Delay_out_ra1	200310	5,01	65,198 79	6,316464	2,937299
SimApp.CRAdapter					
K_ra2	397021	0	2	0,145143	0,3833691
Q_ra2	3656	0	3	0,000715 95	0,02812185
Delay_in_ra2	100170	5,01	44,121 57	7,301925	3,47316
Delay_out_ra2	100168	5,01	38,940 49	6,509531	2,850083
SimApp.CRAdapter					
K_ra3	1097809	0	2	0,433824	0,6520737
Q_ra3	104110	0	8	0,028584 78	0,21737
Delay_in_ra3	300480	5,01	81,81	6,658251	3,787995
Delay_out_ra3	300479	5,01	79,189 45	6,839949	3,840466
SimApp.CJainSLEE					
K_jslee	1170661	0	4	0,877788 5	1,033761
Q_jslee	31258	0	7	0,012732 3	0,1510283
SimApp.CDBServer					
K_db	402395	0	2	1,025537	0,8089544
Delay_db	300479	5,000158	236,70 92	41,01809	22,96612
Q_db	198564	0	9	0,206972 5	0,5972278

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =10210268 Run time =1899 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					
Delay1	199989	45,06059	404,21	96,492	33,921
Delay2	100313	45,06084	388,2703	108,2662	36,2675
SimApp.CRAdapter					
K_ra1	771544	0	2	0,2861392	0,5342023
Q_ra1	28412	0	6	0,00591496	0,08585649
Delay_in_ra1	199989	5,01	63,3851	7,217071	3,630764
Delay_out_ra1	199989	5,01	59,25687	6,298943	2,897201
SimApp.CRAdapter					
K_ra2	397380	0	2	0,1444196	0,3822809
Q_ra2	3872	0	3	0,00075096	0,02906724
Delay_in_ra2	100313	5,01	48,47293	7,240913	3,454152
Delay_out_ra2	100313	5,01	46,62	6,498295	2,81738
SimApp.CRAdapter					
K_ra3	1099616	0	2	0,4308723	0,6497175
Q_ra3	101592	0	7	0,02838171	0,2196565
Delay_in_ra3	300302	5,01	81,91985	6,639667	3,784698
Delay_out_ra3	300302	5,01	82,64003	6,787583	3,825346
SimApp.CJainSLEE					
K_jslee	1169990	0	4	0,8747813	1,031697
Q_jslee	31218	0	8	0,01261869	0,1503441
SimApp.CDBServer					
K_db	402454	0	2	1,02652	0,8076992
Delay_db	300302	5,000593	219,7868	41,02184	22,89394
Q_db	198150	0	8	0,2053654	0,5903475

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =5407588 Run time =1401 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					
Delay1	200419	45,0632	449,3499	95,81495	34,0173
Delay2	100000	45,06448	421,6979	107,5873	36,50987
SimApp.CRAdapter					
K_ra1	774184	0	2	0,2822056	0,5300948
Q_ra1	27498	0	5	0,00589307	0,08661988

Delay_in_ra1	200422	5,01	63,4935	7,058984	3,534981
Delay_out_ra1	200419	5,01	65,26	6,280619	2,884538
			SimApp.CRAdapter		
K_ra2	396294	0	2	0,1414349	0,3783084
Q_ra2	3712	0	3	0,000777 09	0,02968253
Delay_in_ra2	100003	5,01	53,940 54	7,046932	3,332539
Delay_out_ra2	100000	5,01	40,783 35	6,456865	2,79639
			SimApp.CRAdapter		
K_ra3	1100061	0	2	0,4255122	0,6455637
Q_ra3	101624	0	8	0,029917 29	0,2314697
Delay_in_ra3	300423	5,01	90,36	6,64325	3,844541
Delay_out_ra3	300419	5,01	91,135 84	6,679945	3,82969
			SimApp.CJainSLEE		
K_jslee	1172153	0	4	0,8701815	1,025214
Q_jslee	29534	0	8	0,013115 65	0,159222
			SimApp.CDBServer		
K_db	404408	0	2	1,025328	0,8070766
Delay_db	300419	5,000868	212,30 03	40,84853	22,76657
Q_db	196434	0	9	0,201863 5	0,5868659

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =3000535 Run time =1120 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
			SimApp.CNetwork		
Delay1	200043	45,06024	538,723	93,90762	32,35943
Delay2	100010	45,06395	467,1667	105,7362	34,97984
			SimApp.CRAdapter		
K_ra1	774756	0	2	0,2739471	0,5211986
Q_ra1	25418	0	5	0,005217 98	0,08176159
Delay_in_ra1	200044	5,01	73,788 29	6,821021	3,280249
Delay_out_ra1	200043	5,01	68,845 9	6,208485	2,786206
			SimApp.CRAdapter		
K_ra2	396750	0	2	0,1373647	0,3716072
Q_ra2	3290	0	3	0,000615 79	0,0258733
Delay_in_ra2	100010	5,01	44,442 92	6,782744	3,06585
Delay_out_ra2	100010	5,01	45,09	6,382781	2,746446
			SimApp.CRAdapter		
K_ra3	1105926	0	2	0,4149225	0,6366225
Q_ra3	94288	0	11	0,026178 36	0,2141612

Delay_in_ra3	300054	5,01	125,25	6,554962	3,613227
Delay_out_ra3	300053	5,01	120,73 83	6,485481	3,509004
SimApp.CJainSLEE					
K_jslee	1176810	0	4	0,851291 2	1,000773
Q_jslee	23404	0	11	0,010167 77	0,1428042
SimApp.CDBServer					
K_db	407315	0	2	1,023313	0,8051888
Delay_db	300053	5,000239	211,79 84	40,53682	22,6012
Q_db	192792	0	8	0,193006 2	0,5675308

Simulation experiment: PIRMASIS IMITACINIS MODELIS

Event count **Run time** **Tslice** Event count =2399760 Run time =1033 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					
Delay1	199771	45,0602	416,21 74	93,06358	31,90075
Delay2	100199	45,06087	421,42 77	105,0002	34,35218
SimApp.CRAdapter					
K_ra1	764040	0	2	0,292363 2	0,5436873
Q_ra1	35044	0	7	0,008885 8	0,1120479
Delay_in_ra1	199771	5,01	84,386 71	7,332719	4,290813
Delay_out_ra1	199771	5,01	100,16	6,510464	3,524462
SimApp.CRAdapter					
K_ra2	396102	0	2	0,147254 9	0,3879707
Q_ra2	4694	0	4	0,001054 63	0,03535428
Delay_in_ra2	100199	5,01	61,580 37	7,308821	4,006925
Delay_out_ra2	100199	5,01	70,12	6,707233	3,418008
SimApp.CRAdapter					
K_ra3	1074106	0	2	0,442309 6	0,6634842
Q_ra3	125774	0	14	0,048109 78	0,340218
Delay_in_ra3	299970	5,01	130,40 24	7,058341	4,98429
Delay_out_ra3	299970	5,01	130,09 53	6,965469	4,869342
SimApp.CJainSLEE					
K_jslee	1196726	0	4	0,756306 7	0,8563396
Q_jslee	3154	0	4	0,000828 71	0,03143832
SimApp.CDBServer					
K_db	409168	0	2	1,024177	0,804379
Delay_db	299970	5,000075	245,87 08	40,4744	22,54875
Q_db	190772	0	10	0,189928 1	0,56483

Simulation experiment: PIRMASIS IMITACINIS

MODELIS

Event count =2406421 Run time =1252 ms 12

2406421 1252

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					
Delay1	200597	45,06058	422,9838	93,41899	31,63519
Delay2	100204	45,07222	388,8549	105,3151	34,32483
SimApp.CRAdapter					
K_ra1	766810	0	2	0,2944004	0,545678
Q_ra1	35584	0	5	0,00883905	0,1094348
Delay_in_ra1	200600	5,01	75,6425	7,357353	4,257114
Delay_out_ra1	200597	5,01	79,84554	6,535369	3,544778
SimApp.CRAdapter					
K_ra2	395982	0	2	0,1477448	0,3884713
Q_ra2	4834	0	3	0,0011093	0,03588891
Delay_in_ra2	100204	5,01	57,53076	7,323525	4,058101
Delay_out_ra2	100204	5,01	73,87488	6,741029	3,441616
SimApp.CRAdapter					
K_ra3	1076475	0	2	0,444909	0,6649123
Q_ra3	126736	0	15	0,0466907	0,3221529
Delay_in_ra3	300803	5,01	115,19	7,069579	4,857615
Delay_out_ra3	300802	5,01	114,9885	6,970066	4,745734
SimApp.CJainSLEE					
K_jslee	1200067	0	4	0,7601692	0,8584782
Q_jslee	3144	0	3	0,00082996	0,03104549
SimApp.CDBServer					
K_db	406593	0	2	1,028328	0,805924
Delay_db	300802	5,000374	206,4434	40,71029	22,6497
Q_db	195012	0	9	0,1962459	0,5732776

Priedas Nr.2 . Antrojo imitacinio modelio eksperimentiniaia rezultatai

Simulation experiment: Antrasis imitacinis modelis (analitinis)

Event count =2396016 Run time =1283 ms

	Number of Observation	Min Value	Max Value	Mean Value	Standart Deviation
SimApp.CNetwork					
Delay1	199495	45,00213	408,8731	96,47478	34,08166
Delay2	100006	45,01798	391,3	108,2656	36,45335

SimApp.CRAdapter					
K_ra1	768948	0	2	0,286754	0,536067
Q_ra1	29032	0	5	0,006092	0,087954
Delay_in_ra1	199495	5	62,9817	7,316896	3,751482
Delay_out_ra1	199495	5	59,36192	6,248127	2,872206
SimApp.CRAdapter					
K_ra2	396408	0	2	0,144466	0,382686
Q_ra2	3620	0	3	0,000708	0,027982
Delay_in_ra2	100008	5	56,79029	7,306777	3,493776
Delay_out_ra2	100006	5	49,57114	6,433305	2,762199
SimApp.CRAdapter					
K_ra3	1096044	0	2	0,429961	0,649969
Q_ra3	101964	0	9	0,027226	0,218355
Delay_in_ra3	299503	5	94,46366	6,55607	3,731204
Delay_out_ra3	299501	5	104,5086	6,814851	3,832737
SimApp.CJainSLEE					
K_jslee	1167516	0	4	0,865882	1,028993
Q_jslee	30492	0	7	0,013043	0,154659
SimApp.CDBServer					
K_db	400416	0	2	1,022766	0,809078
Delay_db	299501	5,000805	231,846	41,17571	23,18387
Q_db	198588	0	12	0,21046	0,608307

Priedas Nr. 3. XML ataskaitų generavimo algoritmas.

```
public void ReportTable()
{
    XmlWriterSettings settings = new XmlWriterSettings();
    settings.Indent = true;
    XmlWriter theOutput = XmlWriter.Create("report.xml", settings);
    theOutput.WriteProcessingInstruction("xml-stylesheet",
        "type=\"text/xsl\" href=\"PLASimClassLibrary/report_trnsf.xsl\"");
    theOutput.WriteStartElement("report");
    theOutput.WriteStartElement("experiment");
    theOutput.WriteAttributeString("title", title);
    theOutput.WriteEndElement();
    theOutput.WriteStartElement("statistics");
    foreach (Aggregate item in modelAggregates)
    {
        if (item.StatNotEmty())
        {
            theOutput.WriteStartElement("smodule");
            theOutput.WriteAttributeString("name", item.ToString());
            item.StatTable(theOutput);
            theOutput.WriteEndElement();
        }
    }
    theOutput.WriteEndElement();
    theOutput.WriteStartElement("histograms");
    foreach (Aggregate item in modelAggregates)
    {
        if (item.HistNotEmty())
        {
            theOutput.WriteStartElement("hmodule");
            theOutput.WriteAttributeString("name", item.ToString());
            item.HistTable(theOutput);
            theOutput.WriteEndElement();
        }
    }
    theOutput.WriteEndElement();
    theOutput.WriteEndElement();
    theOutput.Flush();
    theOutput.Close();
}
```