KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Vaida Kašubienė

MAZUTO PERKROVIMO PROCESO NAFTOS TERMINALE IMITACINIS MODELIS

Magistro darbas

Darbo vadovas :

Dr. V.Pilkauskas

Kaunas
2004

TURINYS

ĮVADAS

Dažniausiai kroviniai gabenami geležinkeliu bei vandens transportu. Tobulinant ūkio struktūrą, diegiant pažangias technologijas, uosto organizavimas ir valdymas tiesiogiai įtakoja uosto našumą ir sąlygas, kuriomis, krovinio aptarnavimas, atliekamas. Daugeliu atvejų ši įtaka yra pabrėžiama laiko tarpu, per kurį atliekami procesai [1].

Baigiamojo darbo tikslas – sudaryti mazuto perkrovimo proceso, naftos terminale, imitacinį modelį, kuris leistų modeliuoti naftos produktų priėmimą geležinkeliu į naftos terminalą. Naftos terminalas skirtas mazuto, atvežto geležinkeliu kaupimui ir perkrovimui į tankerius. Modelis projektuojamas remiantis formaliu sistemos specifikavimu, panaudojus agregatinį metodą.

Agregatinis požiūris yra sėkmingai taikomas formaliam specifikavimui, teisingumo tikrinimui ir įvairių rūšies sistemų modeliavimui. Tačiau šiuolaikinės programinės įrangos kūrimo tendencijos orientuojasi į objektinį kūrimo procesą. Tam būtina pasitelkti papildomas objektines konstrukcijas, kuriomis galima sėkmingai rengti agregatinį aprašą. Šio uždavinio sprendimui naudojamas JPranas paketas.

Mazuto perkrovimo proceso agregatinė schema, sudaryta iš atitinkamų agregatų:

1. Traukinio agregatas realizuoja traukinių atvykimą į geležinkelio stotį.
2. Stoties agregatas realizuoja atvykstančių traukinių priėmimą geležinkelio stotyje.
3. Valdymo agregatas realizuoja vagonų paskirstymą į naftos produktų perpylimo estakadas.
4. Bazės agregatas realizuoja rezervuarų užpildymą naftos produktais.
5. Tanklaivio agregatas realizuoja naftos produktų pakrovimą į tanklaivius.

Buvo panaudota programa „Agregatų Braižyklė“, kuri leidžia grafiniu būdu nusibraižyti sistemos diagramas ir iš jų susigeneruoti modeliuojamos sistemos imituojamą programą. Pateikti imitacinio modelio programinės realizacijos klasių modeliai. Imitacinis modelis buvo detalizuojamas Java programavimo kalba.

Atlikta analizė, kurios metu buvo atlikti imitacinio modelio eksperimentai, optimalių sprendimų radimui. Perpilant mazutą iš geležinkelio vagonų į rezervuarus, lauko temperatūra turi labai didelę įtaką, nes mazutas, priklausomai nuo esamo jame sieros kiekio, tirštėja jau esant vidutiniškai +20ºC temperatūrai. Esant žemesnei lauko temperatūrai, ilgiau pastovėjusius traukinių vagonus geležinkelio stotyje, tenka šildyti ir tik tada perpilti į rezervuarus. Vagonų šildymas brangiai kainuoja ir padidina naftos terminalo mazuto perkrovimo išlaidas. Todėl yra

labai svarbu naftos terminale priimti tiek mazuto vagonų, kiek bus spėjama iškrauti, kad nesusidarytų eilės ir neatvėstų mazutas žemiau +20ºC temperatūros.

Sukurtas modelis leidžia keisti tokius modelio parametrus:

1. Traukinių atvykimo į geležinkelio stotį intensyvumą.

2. Lauko temperatūrą.

Modeliavimo metu buvo skaičiuojamos gaunamos pajamos už mazuto perkrovimą, bei naftos terminalo išlaidos mazutui perkrauti. Su sukurtu imitaciniu modeliu buvo atlikti eksperimentai, kurių metu pateiktos pajamų, išlaidų ir pelno grafinės išraiškos, kurios parodo traukinių priėmimo traukinių stotyje intensyvumą, atsižvelgiant į esamą lauko temperatūrą.

Baigiamojo darbo metodika: mokslinės literatūros analizė, jos apibendrinimas ir pateikimas darbe; praktikos metu sukauptų duomenų panaudojimas.

Pateiktos išvados ir gauti rezultatai: nustatyta, kokiu intensyvumu gali būti priimami traukiniai, atvykę į geležinkelio stotį, perkrauti mazutą, priklausomai nuo esamos lauko temperatūros, kad naftos terminalo darbas būtų pelningas.

2. JŪROS UOSTŲ IMITACINIŲ MODELIŲ SUDARYMAS INFORMACINĖSE SISTEMOSE

2. 1 Specializuoti imitacinio modeliavimo paketai

Uosto proceso modeliavimo sistemos (Port Process Simulator, toliau PPS) idėja – vientisa, konfigūruojama pagal naudotojo poreikius, programų sistema, galinti modeliuoti uostą ir jo procesus [2]. Tame tarpe suprastruktūros, infrastruktūros, akvastruktūros, įrangos ir žemės transporto priemonių. Sistemos inicializacijai panaudojus uosto prieigų ir jo geografinio išdėstymo grafines bylas, modeliavimo įrenginio programinė įranga gali būti pritaikyta kiekvienam uostui. Ši specifinė visa uosto informacija susijusi su laivais, žemės transportu ir kroviniais. Pradinių duomenų įvedimas PPS leidžia emuliuoti uostą tyrimų procese.

Įvedimo duomenys fokusuojami šešiose srityse. Tai visiškai apibendrina ir adaptuoja modeliavimo įrenginio programinę įrangą bei išsamiai pavaizduoja uosto specifiką. Norint sukurti dominančio uosto modelį, duomenys į modeliavimo įrenginį įvedami tokia tvarka:

- Krovinių srautai.
- Transportavimo režimai.
- Transportavimo priemonės.
- Uosto išdėstymas.
- Krovinių paskirstymo infrastruktūra.
- Uostą reguliuojanti aplinka.

Kaskadinių meniu sistema naudojama parinkimui ir įvedimui krovinių, praeinančių per uostą, kartu susiejant juos su transportavimo charakteristikomis. Pritaikymas pagal poreikius prasideda kuriant tokią seką: prieigos iš jūros, geografinis uosto išdėstymas ir žemės prieigos prie uosto. Tada įvedama informacija apie įrangą ir transporto priemones, kurios reikalingos užtikrinti iš anksto pasirinktą našumo lygį. Kuriant specifinio uosto modelį iš pasirenkamų elementų kaskadinių meniu sistemos pagalba, modeliavimo įrenginys pastoviai tikrina ar tinkama įranga, statiniai ir transporto priemonės parinkti pagal užduoties lygį.

Išvestinė informacija kaupiama penkiose bylose, kurios skirtos neapdorotos informacijos paruošimui, kuri po to gali būti naudojama statistinei analizei ir vertinimams MS Excel, MS Access ar kitose programose.

2.1.1 Uosto funkcinė aplinka

Modeliavimo formavimas yra pagrįstas veiksmais ir procedūromis, kuriuos turi atlikti laivas, atvykstantis į uostą. Tai gali būti laivo valdymas, buksyravimas, judėjimas į stovėjimo vietą ir laukimas pirminių nurodymų uosto prieigose, kanale, baseine prisišvartavimo vietoje. Kai tik muitinės, imigracijos, uosto saugumo ir vyresnybės reikalavimai įvykdyti, galima pradėti krovimo darbus.

Modeliavimo sistema skirta ne tik sekti atvykstančių laivų judėjimą, bet taip pat ir antžeminio transporto judėjimą. Ji apima krovinių transportavimą į uostą ir iš jo į krašto gilumą. Tai gali būti žemės keliai, vidiniai vandenys, transportavimas pakrante ir trumpi pervežimai jūra.

Nors visi veiksmai yra pakankamai bendri, tačiau jų atlikimas konkrečiame uoste yra specifinis. Būtent dėl šios priežasties, modeliavimo sistema PPS reikalauja operatoriaus priežiūros, apimančios vykstančius veiksmus, judėjimo ir laiko analizavimą, laivų įvedimo paskirstymą suvedant sistemos reikalaujamus duomenis. Atsižvelgimas į visus šiuos faktorius specifiniame uoste atskirus proceso etapus derinant su uosto išdėstymo žemėlapiais daro PPS bendrą.

Tinkamos nuorodos, uosto organizavimas ir valdymas tiesiogiai įtakoja uosto našumą ir sąlygas, kuriomis, pvz., krovinio aptarnavimas, atliekamas. Daugumoje atvejų ši įtaka yra pabrėžiama laiko tarpu, per kurį atliekami procesai. Dėl minėtų priežasčių modeliavimo sistema turi vartotojo sąsają, kuri įgalina naudotoją bendrauti su sistema ir duoti nurodymus, kurių efektyvumas išreiškiamas laiku, reikalingu atlikti tam tikras operacijas.

2.1.2 IT ir programų sistemų inžinerijos planas

PPS programinė įranga paremta atskirais įvykiais, kurie vyksta laivui atvykstant į uostą ir kroviniams patenkant į krašto gilumą. Tai leidžia padaryti apibendrintą, statinį, tikimybinį modelį su bendru planu, kurį galima pritaikyti taip, kad uostą atspindėtų tam tikromis nagrinėjamomis aplinkybėmis [3].

PPS dizainas susideda iš daugiapakopio duomenų struktūrų pateikimo, programos struktūros, interfeiso (sąsajos įtaiso) charakteristikų ir procedūrinių detalių.

PPS architektūrinis dizainas paremtas modulinių programų struktūrų pateikimu. Moduliavimas yra vienintelis programinės įrangos bruožas, leidžiantis ją protingai valdyti. PPS turi 31 modulį. Kiekvienas realizacijos modulių turi savo apibrėžimo modulį. Šis metodas sukuria puikų kontrolinį ryšį tarp modulių. Moduliai funkcionaliai yra nepriklausomi vienas nuo kito. Kad labiau suklasifikuotų modulius, įdiegiamas silpnas surišimas ir stipri kohezija,

panaudojant horizontalų struktūrinį suskaidymą. Visa tai sudaro tris dalis: įvestį, duomenų transformavimą ir išvestį. Toks horizontalus PPS struktūros padalijimas duoda šią naudą:

- Lengvas programinės įrangos testavimas;
- Lengvas modeliavimo įrenginio naudojimas;
- Jo architektūrą lengva praplėsti.

Be šių modulių, PPS turi savo bibliotekos bylą, talpinančią visas diagramas, meniu, paletes, pranešimus ir vaizdinius, reikalingus bendravimui su naudotoju ir animacijai.

PPS architektūroje naudojamas labai žemas abstrakcijos lygis. Apsvarstoma menkiausia uosto operacijos detalė ir įdiegiama kaip analitinis, hierarchinis procesas, kuris pasiekė tinklo tikslus, ypač paprastumą.

Duomenų dizainas įdiegiamas naudojant Modsim-III savybes, daugiausia statinius ir dinaminius masyvus. Modsim-III užtikrina fiksuotų dydžių masyvų išskyrimą kompiliavimo metu ir palaiko visišką lankstumą išskiriant dinaminius masyvus, kurių reikalauja programuotojas, kad gautų vietą saugojimui. Duomenų struktūros parodomos tik tiems moduliams, kurie tiesiogiai juos naudoja. Pritaikoma duomenų slėpimo ir surišimo koncepcija. Abstrakčių duomenų tipų šablono naudojimas leidžia dirbti su daugelio lygių heterogeniniais masyvais.

Vartotojo sąsajos dizainas fokusuojamas trijose srityse, tokiose kaip:

- Sąsajos dizainas tarp modeliavimo įrenginio modulių;
- Vartotojo sąsajos dizainas tarp programinės įrangos ir kitų išorinių subjektų (gamintojų ir vartotojų) programinių įrangų;
- Vartotojo sąsajos dizainas tarp naudotojo ir kompiuterio.

Modsim-III palaiko vidinių programos vartotojo sąsajų dizainą, kuris dar gali būti vadinamas tarpmoduliniu vartotojo sąsajos dizainu. Kalba užtikrina priemones stipriam tarpmoduliniam duomenų apsikeitimui. Duomenų srautai PPS moduliuose vyksta tokia pat seka, kaip natūraliai veiksmai vyksta uoste. Išoriniame vartotojo sąsajos dizaine modeliavimo įrenginys turi ryšį su spausdintuvu ir operacine sistema, spausdinimo ir sisteminimo reikmėms.

2.1.3 PPS modelis

Diskretiškas modeliavimo įrenginio veiksmų traktavimas leidžia modeliavimo sistemai tobulėti laiko tėkmėje pritaikant atskirų įvykių pakeitimus. Pavyzdžiui, laivų ir krovinių atvykimas ir išvykimas uoste vyksta tam tikru laiko momentu, kuris nurodomas kaip įvykis. Nieko neatsitinka tarp dviejų nuoseklių įvykių, o nubrėžus grafiką šiai situacijai, jis būtų horizontalus. Uoste vykstančių įvykių skaičius yra baigtinis.

Yra nustatytas laikas įplaukimui į uostą ir vidinio atvykimo laikas konteinervežiams pagal grafiką. Ši informacija nuskaitoma iš išorinių duomenų, pavyzdžiui, naudotojo pateiktos įvesties duomenų. Laivo laukimo laikas gali būti nuleidus inkarą arba prisišvartavimo vietoje.

Pasyvūs subjektai taip pat gauna naudotojo nurodyta įvestį. Šioje sekoje prisišvartavimo vieta ir krovinio aptarnavimo įranga pasirenkami. Naudotojas nustato reikalingus duomenis, kad parinktų kranus, sunkvežimius ir priekabas [4].

Esybė su mažiausiai moduliuojamu įvykiu gauna pranešimą nuo įvykių dispečerio, kad suaktyvintų būseną. Ši žinutė pagrįsta naudotojo nurodytu pavyzdžiu laiko trukmei ir modeliavimui. Duomenų įvedimas „konteinervežis“ užregistruoja šį įvykį, taip pat registruojamas laiko momentas, ir tai reiškia, kad konteinervežio objektas užregistruos prisišvartavimą ir seks įvykius visą laiką kol laivas bus iškrautas ir pakrautas. Įvykio laikas, kuris suaktyvina tam tikrus duomenis, visada nurodo momentinio moduliavimo laiką.

Jei dabartinis įvykis turi ryšį su pasyvia esybe, pavyzdžiui, susijęs su išteklių naudojimu, įvesta informacija „konteinervežis“ nusiunčia atitinkamą pranešimą pasyviai esybei, šiuo atveju „švartavimo vietai“. Toks pranešimas yra kaip užklausimas, o priklausomai nuo to, ar švartavimo vieta laisva, pasekmės gali būti tokios:

1. Pasyvi esybė „švartavimo vieta“ atnaujinama pagal užklausimą ir parodo kokie yra ištekliai (kiek yra laisvos vietos), o po to vyksta aktyvios esybės „konteinervežis“ apdorojimas.

2. Aktyvi esybė „konteinervežis“ laukia, kol reikalingos sąlygos bus įvykdytos. Ji yra blokuojama pasyvioje būsenoje kol pasyvi esybė „švartavimo vieta“ nepasidarys aktyvi.

Įvykis taip pat gali reikšti, kad pasyvi esybė, tokia kaip „švartavimo vieta“, turi būti modifikuota, be aktyvios esybės blokavimo, pavyzdžiui, išteklių reikalavimas daromas patenkinamu. Kai pasyvi esybė atnaujinama šiuo būdu, ji turi leisti dirbti bet kuriai aktyviai esybei, kuri esamu momentu laukia tokio pakeitimo. Jai yra suteikiama galimybė vykdytis nustatant einamą modeliavimo momentą kaip sekančio įvykio laiką. Tai atvejai kai vienas konteinervežis palieka švartavimosi vietą, pastaroji tampa laisva ir tinkama kitam konteinervežiui, kuris laukia nuleidęs inkarą.

Kai aktyvus subjektas palieka aktyvuotą būklę neužsiblokuojant, jis arba modifikuoja sekančio moduliuojamo įvykio laiką ir grįžta į laukimo būseną arba yra naikinamas. Po to dispečeris pasirenka ir suaktyvina esybę, turinčią mažiausią sekančio įvykio laiką. Šis šablonas kartojamas, kol nelieka jokios neterminuotos aktyvios esybės arba moduliavimo laikas išeina iš naudotojų nurodytų ribų.

PPS leidžia atnaujinti statistinius duomenis, nepriklausomai nuo to, ar aktyvi, ar pasyvi būklė pasibaigė, pavyzdžiui:

- konteinervežio iškrovimas baigtas,

- konteinervežio pakrovimas baigtas.

Supažindinimui šie dalykai gali būti atspindėti išankstinėse ataskaitose, surinkus tam tikrą susijusią informaciją, o galutinė ataskaita gali būti suformatuota, kad atitiktų MS Excel išvesties duomenų bylas. Šios bylos atnaujinamos tik kai aktyvi ar pasyvi būsena praėjo visą ciklą.

- konteinervežis išvyko po iškrovimo ir pakrovimo.

Modeliavimo sistema pagrįsta procesų orientavimu. Jis kuriamas perrašant procesus, kurie vyksta natūralioje uosto aplinkoje, o loginė seka vadovauja modeliavimo įsistemai. Šie procesai sumodeliuojami pagal atsitiktinę logiką.

Kad priderintų šį proceso orientavimą „jeigu kas nors dar“ sąlyga plačiai naudojama kartu su paprastų skaičiavimų kombinacijomis, susijusiomis su uosto operacijomis, pavyzdžiui, veiksmų atlikimo uoste laikas, krovininio laivo išvykimas ir švartavimo vietos bei krovinių aptarnavimo įrangos atlaisvinimas kitam laivui.

2.2 Modeliai, realizuoti panaudojant diskrečių įvykių formalias specifikacijas

2.2.1 Formalus sistemos specifikavimas

Sistemų formaliam aprašymui naudojami įvairios metodikos [5]. Vienas iš būdų formaliai aprašyti sistemą yra diskrečių įvykių sistemos specifikacijos (DEVS) formalizmas. DEVS formalizme reikalinga nurodyti:

- bazinius modelius, kuriais remiantis konstruojami stambesni;
- kaip šie modeliai tarpusavyje sujungti hierarchinėje struktūroje.

Baziniai modeliai aprašomi šia struktūra:

$M = \langle X.S, Y, \delta_{\text{int}}, \lambda, ta \rangle$, kur

X – aibė išorinių įvedimo įvykių tipų;

S – būsenos aibė;

Y – išorinių įvykių tipų aibė, sugeneruota išėjimui;

δ_{int} (δ_{ext}) – vidinė (išorinė) perėjimo funkcija, valdanti būsenos kitimus, priklausomai nuo vidinių (išorinių) įvykių;

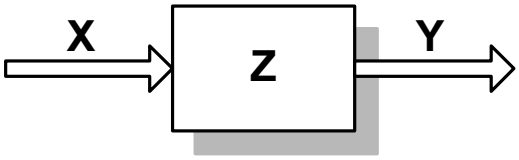
λ - išėjimo funkcija, išėjime generuojanti išorinius įvykius;

ta – išankstinė laiko funkcija.

Tačiau toks aprašymas nebūtų patogus imitacinių modelių sudarymui, tad toliau remiamasi atkarpomis tiesiniais agregatais, kurie išlaiko bazines DEVS bei laikinio automato savybes.

2.2.2 Atkarpomis tiesinis agregatas

Agregatiniame sistemos specifikavimo požiūryje sistema atvaizduojama kaip tarpusavyje sąveikaujančių atkarpomis tiesinių agregratų (angl. *piece-linear agregate* - *PLA*) aibė [6]. PLA suprantamas kaip objektas su apibrėžtų būsenų aibe Z, įėjimo signalų X ir išėjimo signalų Y aibėmis (2.2.1 pav.).



2.2.1 pav. Agregato struktūra

$$X = \{x_1, x_2, ...\}$$

$$Y = \{y_1, y_2, ...\}$$

$$Z = \{z_1, z_2, ...\}$$

Agregato funkcionavimas analizuojamas laiko momentų aibėje $t \in T$. Būsena $z \in Z$, įėjimo signalas $x \in X$ ir išėjimo signalas $y \in Y$ laikomi laikinėmis funkcijomis. Be šių aibių taip pat turi būti apibrėžti perėjimo *H* ir išėjimo *G* operatoriai.

Atkarpomis tiesinio agregato būsena $z \in Z$ kiekvienu laiko momentu atitinka atkarpomis tiesinio Markovo proceso būseną:

$$z(t) = (\nu(t), z_\nu(t)),$$

kur $\nu(t)$ yra diskretusis būsenos komponentas, įgyjantis reikšmes iš baigtinių reikšmių aibės; ir $z_\nu(t)$ yra tolydžioji komponentė, susidedanti iš $z_{\nu_1}(t), z_{\nu_1}(t), z_{\nu_2}, ..., z_{\nu_k}(t)$ koordinačių. Ši komponentė nusako, kada įvyks įvykis, galintis keisti agregato būseną.

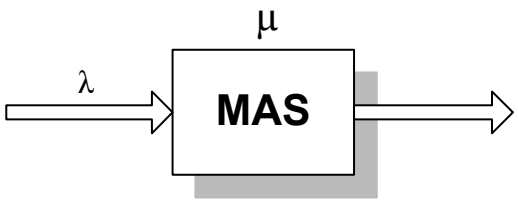
Agregato būsena gali keistis tiktai dėl dviejų priežasčių: kai į agregatą ateina įėjimo signalas arba tolydinė komponentė pasiekia apibrėžtą reikšmę. Įėjimo signalo priėmimo faktas vadinamas išoriniu įvykiu. Nesant įėjimo signalams, agregato būsena kinta pagal sekančią priklausomybę:

$$\nu(t) = const,\ \frac{dz_\nu(t)}{dt} = -\alpha_\nu,$$

kur $\alpha_\nu = (\alpha_{\nu 1}, \alpha_{\nu 2}, ..., \alpha_{\nu k})$ pastovus vektorius.

Tuo atveju, kai negali įvykti kažkoks įvykis, atitinkama tolydinės būsenos dedamosios reikšmė yra begalybė.

Kaip atkarpomis tiesinio agregato pavyzdys pateikiama vieno kanalo masinio aptarnavimo sistema (MAS) (2.2.2 pav.):



2.2.2 pav. Masinio aptarnavimo sistema (MAS)

λ – paraiškų atėjimo į MAS intensyvumas;

μ – paraiškų aptarnavimo intensyvumas;

Agregatų būsenos aprašomos taip:

$z(t) = (\nu(t), z_{\nu_1}(t), z_{\nu_2}(t))$, kur

$\nu(t)$ - paraiškų skaičius sistemoje laiko momentu t;

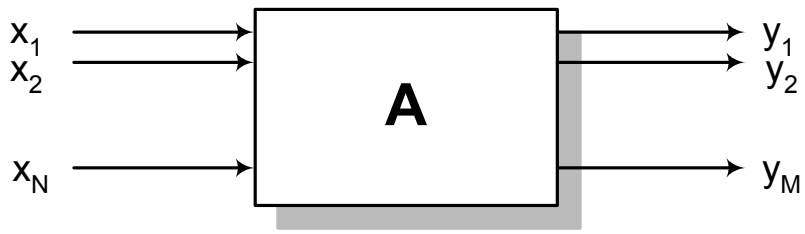
$z_{\nu_1}(t)$ - laiko tarpas, po kurio į sistemą ateis nauja paraiška;

$$z_{\nu_2}(t) = \begin{cases} \infty, jei\, \nu(t) = 0; \\ > 0, jei\, \nu(t) \neq 0; \end{cases}$$

$z_{\nu_2}(t)$ - tai laikas, kada pasibaigs paraiškos aptarnavimas.

Sistemos funkcionavimas apibrėžiamas fizinę reikšmę turinčiomis valdymo ir įėjimo sekomis bei sistemos valdymą nusakančiais algoritmais.

Tarkim, kad agregatas yra sistema, turinti N įėjimų bei M išėjimo sąveikos taškų (2.2.3 pav.):



2.2.3 pav. Agregato įėjimų ir išėjimų schema

Įėjimo signalai $x_1, x_2, ..., x_N \in X$ atsiranda įėjimo sąveikos taške. $x_i \in X_i$, $i = \overline{1,N}$ traktuojamas kaip elementarus signalas, o X_i - elementarių signalų aibė. Bendru atveju elementarus signalas yra vektorinės struktūros: $x_i = \{x_i^1, x_i^2, ..., x_i^{r_i}\}$, t.y. jis gali būti apibrėžiamas keletu rinkinių, kurių kiekvienas turi reikšmę iš tam tikros aibės X_i: $x_i^j \in X_i^j$, $j = \overline{1,r_i}$. Tokiu būdu į i-tąjį sąveikos tašką ateinančių elementarių signalų seka bus:

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

Agregato įėjimų seka yra X_i sekų sąjunga:

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

Analogiškai ir su išėjimo signalais:

$$Y = \{y_1, y_2, ..., y_M\};$$

$$y_1 = \{y_1^1, y_1^2, ..., y_1^{s_i}\} \in Y_1;$$

$$y_l^k \in Y_l^k,\ l = \overline{1,M},\ k = \overline{1,s_1}$$

l-ojo sąveikos taško elementarių signalų seka:

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

Išėjimo signalų aibė:

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

Agregato funkcionavimas nagrinėjamas diskrečiais laiko momentais $T = \{t_1, t_2, ..., t_m, ...\}$, kurias gali įvykti vienas ar keletą įvykių, iššaukiančių agregato būsenos pasikeitimą. Agregato būsenų aibė E susideda iš dviejų poaibių: $E = E' \cup E''$. Į poaibį $E' = \{e_1', e_2', ..., e_N'\}$ įeina įvykių klasės/įvykiai $e_i', i = \overline{1,N}$, sekantys iš įėjimo signalo aibės $X = \{x_1, x_2, ..., x_N\}$. Įvykių klasė

$e_j^{''} = \{e_{ij}^{''}, j = 1,2,...\}$, kur $e_{ij}^{''}$ yra įvykių, įvykstančių j-uoju laiko momentu nuo momento t_0, klasės $e_i^{''}$ įvykis. Poaibio E' įvykiai vadinami išoriniais įvykiais. Agregato įėjimo signalų aibė atvaizduojama į poaibį E': $X \to E'$. Poaibio $E'' = \{e_1^{''}, e_2^{''}, ..., e_f^{''}\}$ įvykiai vadinami vidiniais įvykiais, kur $e_i^{''} = \{e_{ij}^{''}, j = 1,2,...\}$, $i = \overline{1, f}$ yra agregato vidinių įvykių klasės. f apibrėžia agregate vykstančių įvykių skaičių.

Dėl įėjimo įvykių poaibio laiko momentų aibė T dalinama į du poaibius $T = T' \cup T''$: T' - įėjimo signalų atsiradimo poaibis ir T'' - vidinių įvykių įvykimo poaibis. Kiekvienai įvykių $e_i^{''}$ klasei iš poaibio E'' valdymo seka apibrėžiama $\{\xi_j^{(i)}\}$, kur $\xi_j^{(i)}$ - operacijos trukmė.

Agregate įvykstančių operacijų pradžiai ir pabaigai apibrėžti įvedamos kontrolinės sumos sąvokos - $\{s(e_i^{''}, t_m)\}$, $\{w(e_i^{''}, t_m)\}$, $i = \overline{1, f}$, kur $s(e_i^{''}, t_m)$ - operacijos, sekančios po klasės $e_i^{''}$ įvykio, pradžios laiko momentas. Šis laiko momentas yra neapibrėžtas, jei operacija neprasidėjo; $w(e_i^{''}, t_m)$ - operacijos, sekančios po klasės $e_i^{''}$ įvykio, pabaigos laiko momentas. Kontrolinė suma $w(e_i^{''}, t_m)$ apibrėžiama taip:

$$w(e_i^{''}, t_m) = \begin{cases} s(e_i^{''}, t_m) + \xi_{r(e_i^{''}, t_m)+1}, & \text{jei momentu } t_m \text{ įvyksta operacija;} \\ \infty, & \text{priešingu atveju kintamojo reikšmė neapibrėžta.} \end{cases}$$

Diskrečioji būsenos komponentė $\nu(t_m) = \{\nu_1(t_m), \nu_2(t_m), ..., \nu_p(t_m)\}$ nurodo sistemos diskrečiąją būseną. $z_\nu(t_m) = \{w(e_1^{''}, t_m), w(e_2^{''}, t_m), ..., w(e_f^{''}, t_m)\}$ - valdančioji koordinatė, parodanti įvykių įvykimo laiką. Ši koordinatė atitinka kiekvieną $e_i^{''}$ iš įvykių poaibio E'', nes visada tenkinama sąryšis $w(e_i^{''}, t_m) \geq t_m$.

Būsenos koordinatės $z(t_m)$ gali keisti savo reikšmes tik tam tikrais diskrečiais laiko momentais t_m, $m = 0,1,2,...$, kur t_0 nurodo sistemos funkcionavimo pradžios laiką.

Žinant sistemos būseną $z(t_m), m = 0,1,2,...$, sekančio įvykio momentas t_{m+1} apibrėžiamas įėjimo signalo pasirodymu arba lygtimi:

$$t_{m+1} = \min\{w(e_i^{''}, t_m)\},\ 1 \leq i \leq f;$$

Sekančio įvykio e_{m+1} klasė nustatoma pagal įėjimo signalą, jei šis atsiranda laiko momentu t_{m+1} arba apibrėžiama valdymo koordinate, kuri įgyja mažiausią reikšmę laiko momentu t_m.

Perėjimo operatorius $H(e_i)$ nusako naują agregato būseną:

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

Išėjimo signalai y_i iš aibės $Y = \{y_1, y_2, ..., y_m\}$ gali būti agregato generuojami įvykių iš poaibių E' ir E'' įvykimo laiko momentais.

Išvedimo operatorius $G(e_i)$ apibrėžia išėjimo signalų turinį:

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

2.3 Objektiškai orientuotos modeliavimo sistemos biblioteka

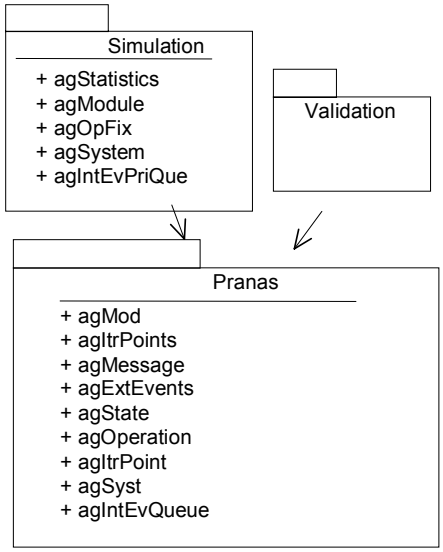
Agregatinis požiūris yra sėkmingai taikomas formaliam specifikavimui, teisingumo tikrinimui ir įvairių rūšies sistemų modeliavimui. Tačiau šiuolaikinės programinės įrangos kūrimo tendencijos orientuojasi į objektinį kūrimo procesą. Tiesiogiai atkarpomis tiesinių agregatų atvaizduoti negalima, nes reikia išlaikyti agregatų sujungimo schemą, realizuoti kiekvieno agregato funkcionalumą bei suteikti tam tikrus mechanizmus sistemos teisingumo tikrinimui ir modeliavimui. Tam būtina pasitelkti papildomas objektines konstrukcijas, kuriomis galima sėkmingai rengti agregatinį aprašą. Šio uždavinio sprendimui naudojamas JPranas paketas [7].

Šis paketas susideda iš trijų stambių posistemių, kurių kiekviena skirta spręsti savo tikslus:

- Agregatų ir jų funcionalumo realizavimo (*Pranas* paketas);
- Modeliavimo (*Simulation* paketas);
- Teisingumo tikrinimo (*Validation* paketas).

Ši Pranas sistemos biblioteka yra naudojama ne tik modeliavimo sistemų sudarymui, bet taip pat gali būti išplėsta vertinimo modelių sudarymui. Išskirti trys paketai.pavaizduoti 2.3.1pav. Šiame paveiksle paketų klasės yra įtrauktos į atitinkamas dalis.

Jpranas paketuose bendros klasės, naudojamos abiejų imitacinio modeliavimo ir teisingumo tikrinimo modelių sudarymui, yra apibrėžtos.



2.3.1 pav. Sistemos PRANAS paketai

2.3.1 Paketas Pranas

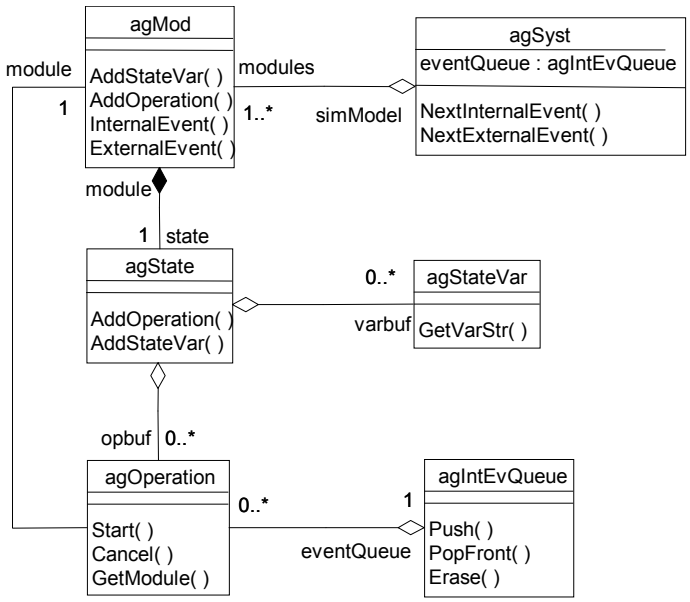
2.3.2 paveiksle pavaizduota sistemos Prano paketo klasių diagrama. Diagrama parodo kad pagrindinė sistemos gavimo klasė agSyst yra skirta modelio sudarymui. Pakete Pranas ši klasė yra abstrakti. Vėliau, modeliavimo ir teisingumo tikrinimo paketuose ji yra detalizuojama.

Kaip parodyta 2.3.1 pav., du metodai *NextExternalEvent* ir *NextInternalEvent* apibrėžia sistemos modeliavimo algoritmo realizaciją.

NextExternallEvent metodas yra skirtas sistemos išorinių įvykių atpažinimui. Po atpažinimo, sistemos valdymas perduodamas atitinkamam sistemos PLA modulio įvedimo/išvedimo operatoriui. *NextExternalEvent* dirba cikle. Jei jis neranda išorinio įvykio, jis yra iškviečiamas vėl. Šis metodas atrenka įvykį iš vidinės įvykių eilės *EventQueue* ir perduoda valdymą atitinkamam sistemos PLA įvedimo/išvedimo operatoriui.

NextInternalEvent yra abstraktus ir jo realizacijos yra pateiktos modeliavimo ir teisingumo tikrinimo moduliuose. Modeliavimo ir teisingumo tikrinimo paketų vidinis įvykių apdorojimas yra skirtingas. Išorinių įvykių eilė *agIntEvQueue* yra taip pat abstrakti klasė.

Klasė *agMod* yra pavaizduota 2.3.2 pav. Tai yra kitokia modeliavimo sistemos abstrakcija. Kiekvienas sistemos modulis turi subklasę *agMod* klasę.

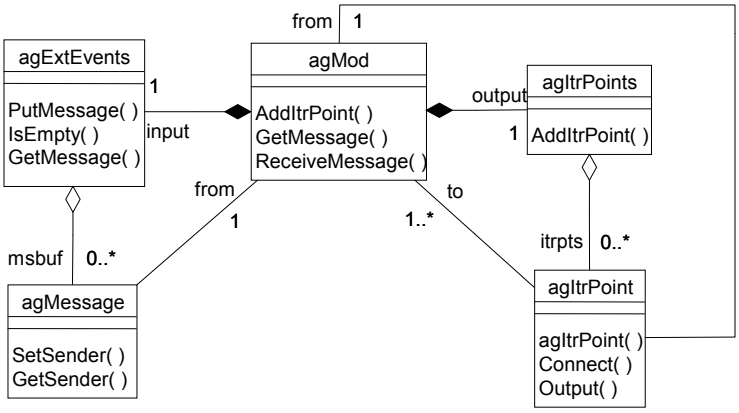


2.3.2 pav. Pranas paketo sistemos modulių klasių diagrama

Pagrindinis modulio elementas yra jo būsena *agState*. Būsena yra saugoma *agStateVar* klasės tipo kintamuosiuose, kur kiekvienas kintamasis aprašo atskirą komponentą ir *agOperation* tipo kintamuosius, kurie aptašo komponentų agregatus. Kitaip sakant, būsena susideda iš dviejų dalių:

- varbuf būsenos kintamųjų;
- opbuf operacijų (vidiniai įvykiai).

Išorinių įvykių klasių diagrama pavaizduota 2.3.3 pav. Kiekvienas *agMod* modulis turi gaunamo signalo buferį *input*. *AgExtEvents* klasė realizuoja šį buferį. Ši klasė kaupia visus iš kitų modulių priimtus pranešimus. Pranešimai *agMessage* klasės arba jos subklasės tipų, gali būti perduodami tarp modulių.

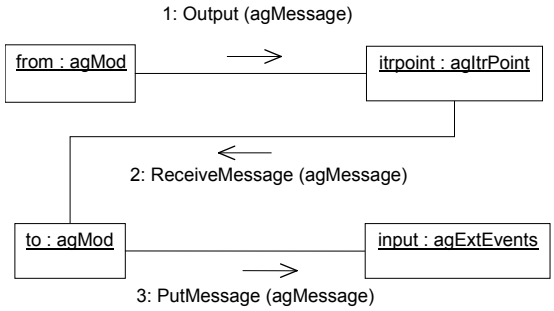


2.3.3 pav. Išorinių įvykių klasių diagrama

Tam kad moduliams perduoti pranešimą, siunčiamas modulis turi būti aprašytas sąveikos taškų *output* aibės pagalba. Ši klasė yra realizuojama konteineriu iš *agItrPoints* klasės. Kiekvienas *agItrPoint* objektas yra sujungtas su jo moduliu (*from* jungtis) naudojant ryšių sąsają ir su moduliais (*to* jungtis) kuriam perduodami pranešimai. Jungtis *to* tipo, sukuria modulių sąveikos kanalus, kurių pagalba perduodami pranešimai tarp modulių.

Išėjimo sąveikos taškų apibrėžimas leidžia prijungti vieną ar daugiau modulių vienam išėjimo sąveikos taškui. Tokiu būdu, prijungiant keletą modulių prie to paties išėjimo sąveikos taško ir jam siunčiant pranešimą, įvyksta automatinis pranešimo perdavimas visiems prijungtiems moduliams.

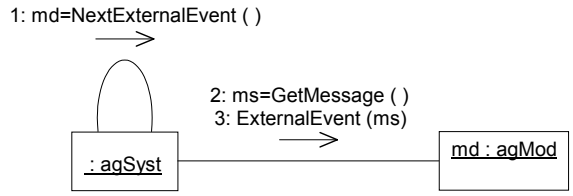
Modulio sąveikos (pranešimų perdavimas) realizavimo algoritmas *agMod* parodytas 2.3.4 pav. diagramoje.



2.3.4 pav. agMod modulio sąveikos algoritmas

Kaip parodyta 2.3.4 pav. diagramoje, tam kad perduoti pranešimą iš vieno modulio kitam, *output* sąveikos taškų metodas modulyje-siųstuve *from* yra paleistas. Šis metodas yra inicijuojamas įvedimo/išvedimo modulio operatoriuje. Po iniciacijos, modeliavimo sistema, kaip parodyta 2.3.5 pav., perduoda metodo parametrus į priėmimo buferį *input* iš *to* modulio-gavėjo.

Po įvedimo/išvedimo operatoriaus įvykdymo modeliavimo sistemos modulyje, modeliavimo sistemos algoritmas ieško pranešimų visuose modulių pranešimo priėmimo buferiuose. Tam tikslui naudojamas *agSyst* metodas *NextExternalEvent*. Šis metodas, kaip parodyta 2.3.5 pav., grąžina identifikuotoją *md* iš pirmos sistemos modulio, kurio pranešimų priėmimo buferis sudarytas iš bent vieno neapdoroto pranešimo. Jei tokių modulių yra, *GetMessage* metodas šiems moduliams atrenka pirmą pranešimą iš *md-th* pranešimų priėmimo buferio. Toliau *ExternalEvent* metodas iš *md*-th pranešimo yra įvykdomas. Šis metodas yra skirtas modulių įvedimo/išvedimo operatorių realizavimui, kai pasitaiko išorinių įvykių. *agMod* klasėje šis metodas yra abstraktus ir turi būti realizuojamas jo paveldėtojo.

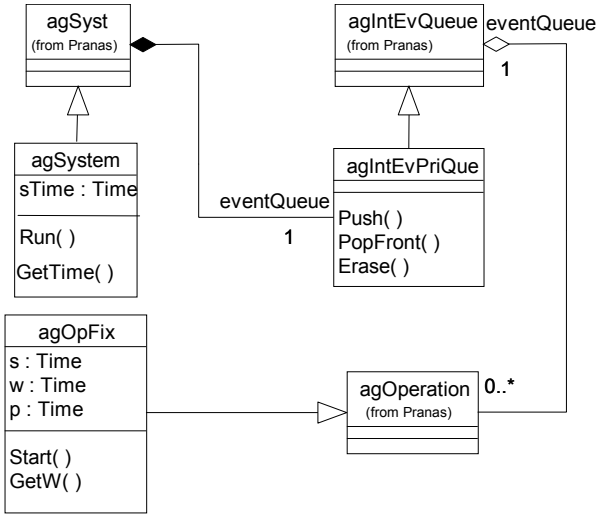


2.3.5 pav. Išorinių įvykių apdorojimo algoritmas

2.3.5pav. pavaizduotas (išorinių įvykių apdorojimo) modulių sąveikos algoritmas yra toks pat modeliavimo ir teisingumo tikrinimo modeliams, ir tokiu būdu pilnai realizuojamas *Pranas* paketuose.

2.3.2 Imitacinio modeliavimo paketas

Vidinių įvykių apdorojimo algoritmai yra skirtingi modeliavimo ir teisingumo tikrinimo modeliuose. Dėl šios priežasties, algoritmai yra realizuojami atitinkamuose paketuose.



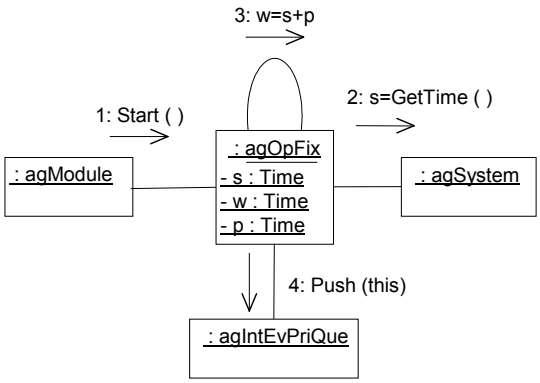
2.3.6 pav. Modeliavimo paketo klasių diagrama

Ši dalis apibrėžia vidinių įvykių apdorojimo algoritmą modeliavimo pakete. Kaip parodyta 2.3.6 pav., subklasė *agSystem* yra apbrėžiama modeliavimo pakete. Ši subklasė suteikia struktūrą veiklos modeliavimui. Modeliavimo modelio klasė *agSystem* suteikia dvi naujas funkcijas. Pirmoji *Run* paleidžia modeliavimą, kol antroji grąžina esamą modeliavimo „laiką“. Duomenų laukas *s* yra taip pat sąlygojamas laikyti esamą modeliavimo „laiką“.

Vidiniai įvykiai yra realizuojami kaip klasės *agOperation* iš paketo Pranas subklasės. Modeliavimas reikalauja palaikyti skirtingų operacijos tipų rinkinį (vidiniai įvykiai). Tokia viena iš subklasių *agOpFix* modeliavimo valdymui pavaizduota 2.3.7 pav. Šios subklasės duomenys:

- s – laiko momentas procesui prasidėjus;
- w – laikas kada įvyko išorinis įvykis;
- p – proceso trukmė.

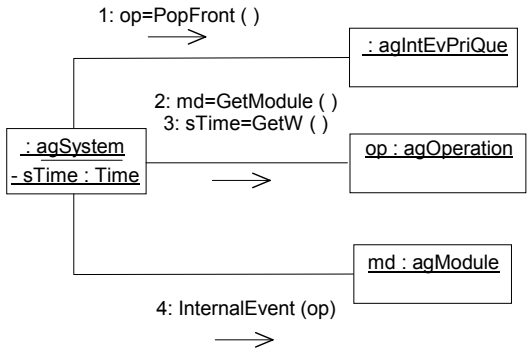
Funkcija *Start* gali būti iškviesta proceso įvykdymui. Prioritetų eilė *agIntEvPriQue,* kuri yra subklasė *agIntEvQueue* klasės, yra naudojama įvykiams, kurie laukia savo eilės įvykti, aprašyti. Ši eilė yra papildoma priklausomai nuo laiko, kai įvykis turi atsitikti, vadinasi žemiausias elementas visada turi būti sekantis modeliuojamas įvykis.



2.3.7 pav. Proceso įvykdymo algoritmas

Proceso įvykdymo algoritmas pavaizduotas 2.3.7 pav. Nurodant atlikti procesą bet kuriame PLA modulio įvedimo/išvedimo operatoriuje, *Start* metodas iš atitinkamo PLA modulių proceso turi būti iškviestas. Po einamo modeliavimo „laiko“ reikšmės priėmimo, šis metodas apskaičiuoja laiko momentą ω, kai įvyks vidinis įvykis. Toliau tas įvykis patalpinamas į išorinio įvykio eilę *eventQueu.*

Išorinių įvykių apdorojimo algoritmas modelių modeliavime pavaizduotas 2.3.8 pav.



2.3.8 pav. Išorinių įvykių apdorojimo algoritmas

Naudojant pateiktas modeliavimo sistemos klasių diagramas, jos realizuotos Java kalboje. Java kalba buvo pasirinkta todėl, kad buvo reikalavimas sukurti modelį kuris turėtų integruotą mazuto perkrovimo naftos terminale informacinę sistemą.

2.4 AgDraw panaudojimas imitacinio modelio programinės realizacijos sudarymui

Programinės įrangos produktas „Agregatų Braižyklė“ yra skirtas sistemų imitaciniams modeliams sudaryti.

Programinė įranga teikia tokias galimybes:

- Agregatų diagramų braižymas
- UML diagramų iš agregatų diagramų generavimas
- UML diagramų redagavimas
- Diagramų spausdinimas spausdintuvu
- Programos kodo generavimas iš agregatų ir UML klasių diagramų:
    - Java kodo generavimas
    - C# kodo generavimas
    - Omnicore CodeGuide 6.0 projekto sudarymas
    - Microsoft Visual Studio.NET projekto sudarymas

Programinė įranga veikia Microsoft® Windows 95/98/ME/NT4.0/2000/XP ™ ir Linux operacinėse sistemose.

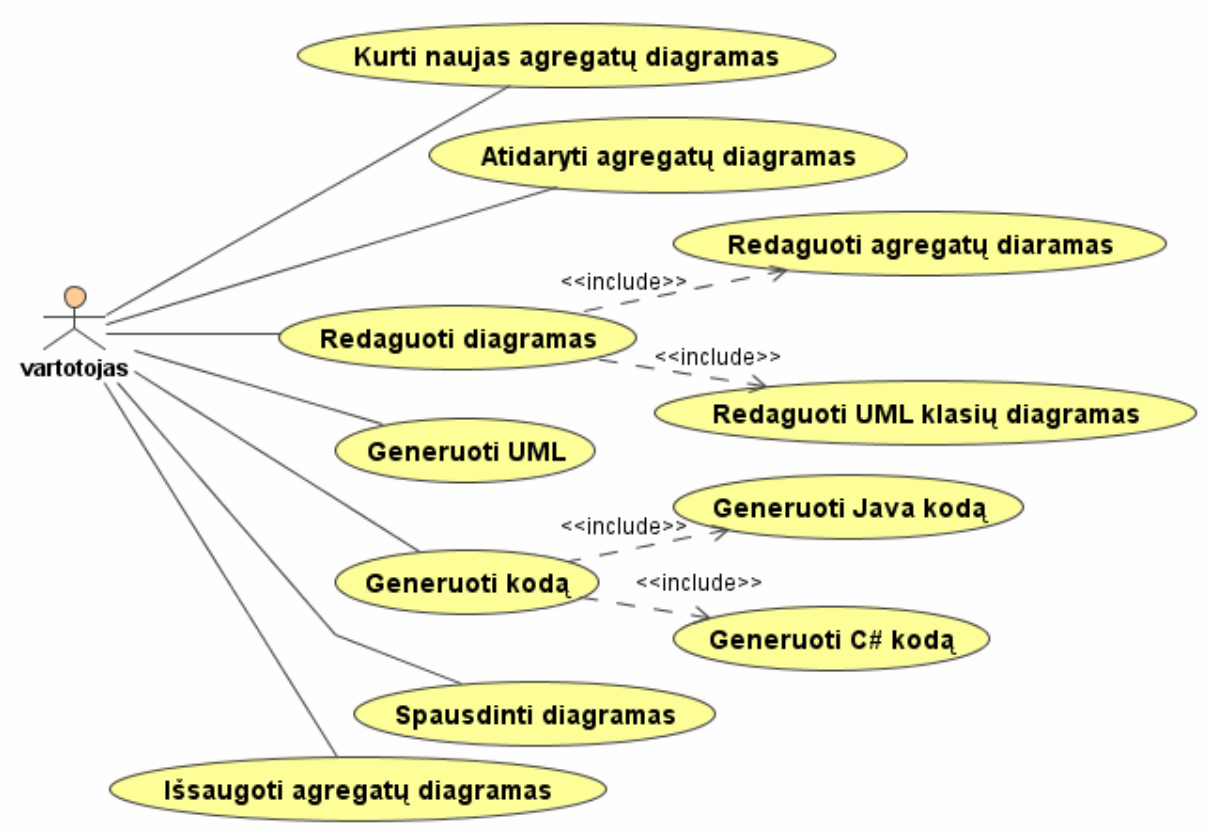
2.4.1 Programinės įrangos panaudojimo atvejai

Programinės įrangos panaudojimo atvejai:

- Kurti naujas diagramas - Galimybė sukurti naują agregatų diagramą (tuščias lapas, kuriame jas galima piešti).
- Atidaryti agregatų diagramas - Galimybė ankščiau redaguotas iš išsaugotas diske diagramas užkrauti ir vėl leisti toliau jas redaguoti.
- Išsaugoti agregatų diagramas - Galimybė išsaugoti diske jau sukurtas, redaguotas ir sugeneruotas diagramas.
- Spausdinti diagramas - Galimybė spausdinti sukurtas, sugeneruotas diagramas. Spausdinamas grafinis vaizdas, turi atrodyti panašiai kaip ir ekrane atvaizduota diagrama.
- Redaguoti diagramas - Gailimybė redaguoti UML, agregatų diagramas. Įterpti naujas klases, naujus agregatus, sudėti naujus ryšius, redaguoti agregato, klasės parametrus.

- Generuoti kodą - Iš nubraižytų agregatų diagramų ir UML diagramų sugeneruoti, agregatų diagramoje modeliuojamos sistemos, programos,modeliuojančios šią sistemą, kodą. Pasirenkamai generuoti C# arba Java kodą.
- Generuoti UML - Iš nubraižytų agregatų diagramų sugeneruoti agregatų diagramoje modeliuojamos sistemos programinį modelį ir jį atvaizduoti UML diagramomis.
- modeliuojamos sistemos programinį modelį ir jį atvaizduoti UML diagramomis.

Programinės įrangos panaudojimo atvejai pateikti 2.4.1 pav.



2.4.1 pav. Programinės įrangos panaudojimo atvejai

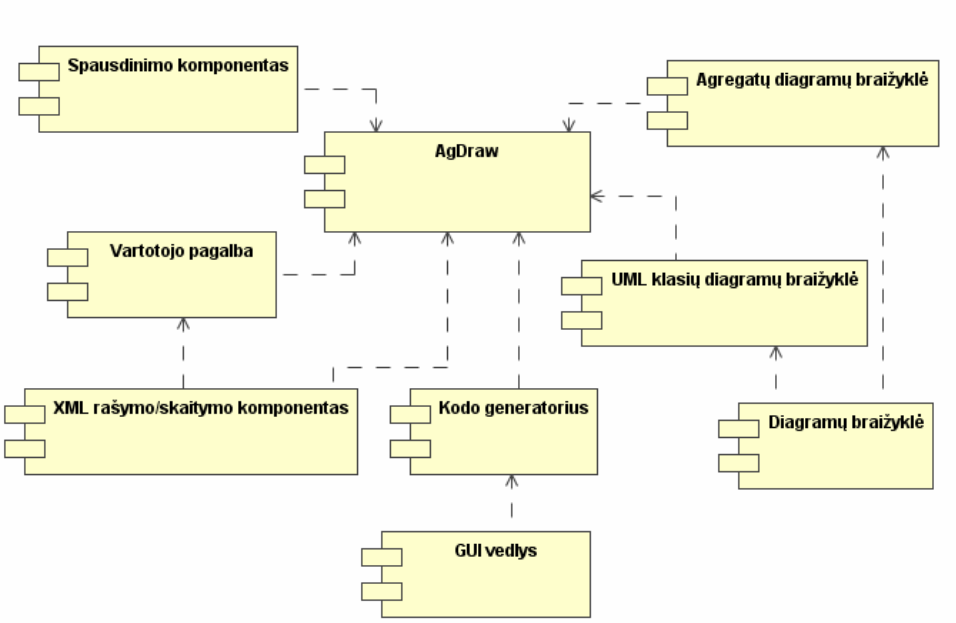
2.4.2 Programinės įrangos komponentės

Programinė įranga sukurta remiantis komponentine technologija.

Programos komponentai yra:

- AgDraw
- Vartotojo pagalba
- Spausdinimo komponentas
- XML rašymo/skaitymo komponentas
- Kodo generatorius

- GUI vedlys
- Diagramų braižyklė
- UML klasių diagramų braižyklė
- Agregatų diagramų braižyklė



2.4.2 pav. Programos komponentai ir ryšiai tarp jų

Komponentas AgDraw – pagrindinis programos komponentas, naudojantis visus kitus komponentus. Jis naudoja Agregatų braižyklę braižyti agregatų diagramoms, UML klasių braižyklę braižyti UML klasių diagramoms, kodo generavimo komponentą – generuoti programos kodą iš agregatų diagramų ir UML klasių diagramų. XML skaitymo/rašymo komponentas naudojamas išsaugoti diagramas diske, o vėliau jas užkrauti tolesniam redagavimui. Spausdinimo komponentas naudojamas spausdinti diagramas spausdintuvu. Diagramų braižyklės naudoja tą modelį atvaizdavimui, o diagramos braižyklės redagavimo įrankiai – redaguoti diagrama.

Agregatų diagramos struktūra naudojama agregatų braižyklėje ją redaguoti, o taip pat kodo generavimo komponente generuojant programos kodą. Taip pat ši struktūra naudojama išsaugoti diagramą diske. Saugojimui diagramos diske naudojamas XML rašymo/skaitymo komponentas. AgDraw komponentas naudoja spausdinimo komponentą atspausdinti diagramas rodomas ekrane spausdintuvu. Spausdinimas vyksta labai paprastai: užtenka diagramos braižyklės komponentui nurodyti paišymo sritį, kuri yra ne programos langas, o spausdintuvas ir diagrama bus pasiųsta į spausdintuvą ir atspausdina. AgDraw komponentas taip pat vartoja vartotojo pagalbos komponentą. Leidžia beveik bet kuriuo metu iškviesti vartotojo pagalbos

langą ir rodyti su dabar atliekamu veiksmu, vartotojo stebimu dialogo langu susijusią pagalbą. AgDraw komponente realizuotas UML klasių generavimas iš agregatų diagramos.

Diagramų braižyklė – komponentas skirtas braižyti diagramas. Naudojamas kaip pagrindas UML klasių diagramų braižyklės ir Agregatų diagramų braižyklės. Komponentas teikias šias funkcijas:

- Įdėti/ pašalinti/ redaguoti dėžutes
- Įdėti/ pašalinti/ redaguoti sujungimus tarp dėžučių.
- Rodyti/ nerodyti redagavimo tinklelį

Komponentas realizuoja programos langelį, kuriame atliekamas diagramų piešimas/redagavimas.

Komponentas yra skirtas naudoti kaip pagrindas kurti įvairių tipų diagramas. Nauji diagramų tipai sukuriami paveldėjus *Diagram* klasę, realizavus jos metodus *createBox(...)* ir *createLink(...)*, kurie atitinkamai sukuria diagramoje braižomus objektus ir sujungimus tarp jų. Taip pat reikia sukurti diagramoje braižomų objektų ir sujungimų klases. Tai realizuojama paveldint *DiagramBox* ir *DiagramConnection* klases ir realizuojant jų abstrakčius metodus.

UML klasių diagramų braižyklė – komponentas skirtas braižyti UML klases. Naudoja diagramų braižyklės komponento klases paveldėjimo keliu ir teikias šias funkcijas:

- Įdėti/pašalinti/redaguoti UML klases
- Įdėti/pašalinti/redaguoti paveldėjimo ryšį
- Įdėti/pašalinti/redaguoti agregavimo ryšį
- Įdėti/pašalinti/redaguoti asociacijos ryšį

Agregatų diagramų braižyklė – komponentas skirtas braižyti agregatų diagramas ir ryšius tarp jų. Naudoja diagramų braižyklės komponento klases paveldėjimo keliu ir teikias šias funkcijas:

- Įdėti/ pašalinti/ redaguoti agregatus
- Įdėti/ pašalinti/ redaguoti sujungimus tarp agregatų

3. MAZUTO PERKROVIMO PROCESO NAFTOS TERMINALE IMITACINIO MODELIO SUDARYMAS

3.1 Mazuto perkrovimo proceso konceptualinis modelis

Į terminalą mazutas pristatomas traukiniu. Perkrovimo trukmė iš traukinio cisternos į rezervuarą priklauso nuo mazuto temperatūros ir ši priklausomybė yra žinoma. Perpilant mazutą iš geležinkelio vagonų į rezervuarus didelę įtaką turi lauko temperatūra, nes mazutas, priklausomai nuo jame esančio sieros kiekio tirštėja jau esant +20ºC temperatūrai. Mazuto perkrovimui į terminalą yra naudojamos dvi platformos, kuriose yra atliekas traukinių vagonų iškrovimas. Kada abi platformos yra užimtos, atvykę traukiniai yra statomi į eilę traukinių stotyje. Traukinių vagonai yra iškraunami pagal jų atvykimo tvarką.

3.2 Mazuto perkrovimo proceso matematinis modelis

Žemiau pateikti imitacinio modelio agregatinės sistemos ir atitinkamų agregatų matematiniai aprašymai.

3.2.1 Agregatinė schema

Naftos terminalas yra skirtas naftos ir naftos produktų, taip pat mazuto, atvežto geležinkeliu, kaupimui ir krovimui į tanklaivius. Šiuo atveju analizuojamas mazuto perkrovimo procesas naftos terminale [8]. Šiam procesui modeliuoti yra apibrėžiamos pagrindinės analizuojamos sistemos:

- Traukinys;
- Stotis;
- Valdymas;
- Bazė;
- Tanklaivis.

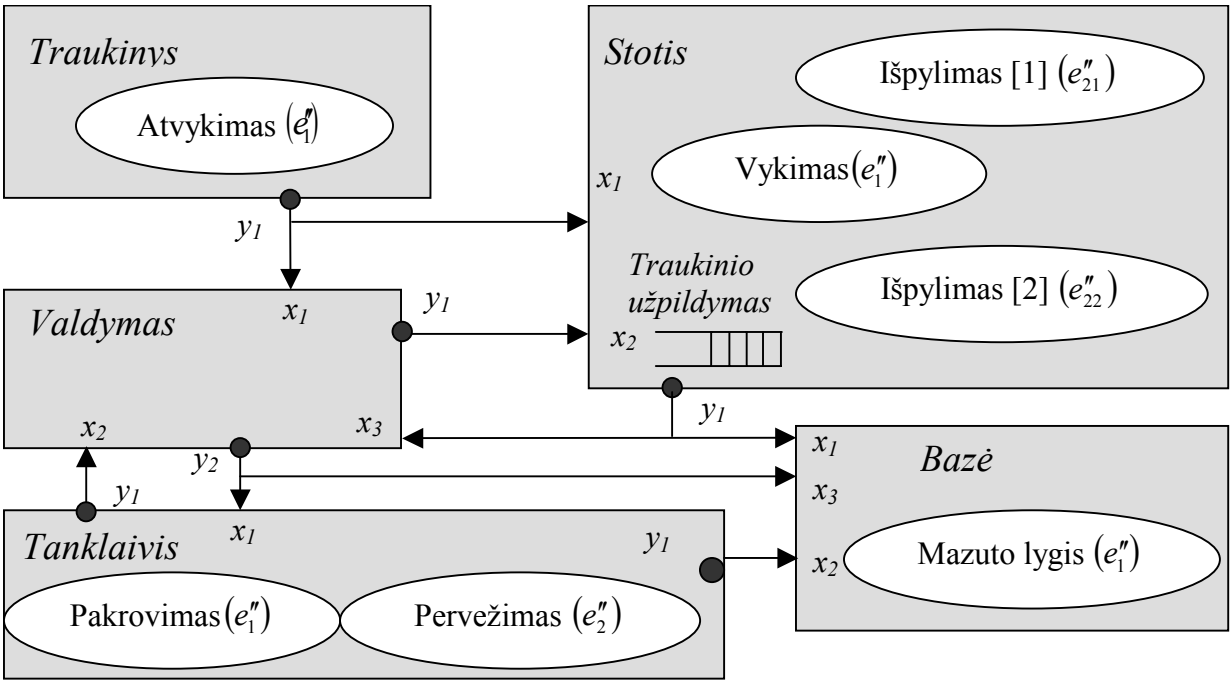
Mazuto perkrovimo proceso agregatinė schema, pavaizduota 3.2.1 pav., sudaroma iš atitinkamų agregatų:

1. Traukinio agregatas realizuoja traukinių atvykimą į geležinkelio stotį.
2. Stoties agregatas realizuoja atvykstančių traukinių priėmimą geležinkelio stotyje.

3. Valdymo agregatas realizuoja vagonų paskirstymą į naftos produktų perpylimo estakadas.

4. Bazės agregatas realizuoja rezervuarų užpildymą naftos produktais.

5. Tanklaivio agregatas realizuoja naftos produktų pakrovimą į tanklaivius.



3.2.1 pav. Agregatinė schema

3.2.2 Traukinio agregatas

1. Įėjimo signalai: $X = \varnothing$.
2. Išėjimo signalai: $Y = \{y_1\}$, $y_1 \in \{atvyko\_traukinys\}$, kai *atvyko_traukinys* – traukinys jau yra atvykęs.
3. Išoriniai įvykiai: $E' = \varnothing$.
4. Vidiniai įvykiai: $E'' = \{e_1''\}$, e_1'' – įvykis, kuris atsitinka atvykus traukiniui.
5. Valdymo sekos: $e_1'' \to \{\eta\}_{i=0}^{\infty}$, η – laiko trukmė iki kada atvyksta sekantis traukinys.
6. Diskretinės būsenų koordinatės: $\nu(t) = \varnothing$.
7. Tolydinės būsenų koordinatės: $z_\nu(t) = \{w(e_1'', t)\}$, kur $w(e_1'', t)$ – laikas, kai atvyksta sekantis traukinys.
8. Pradinė būsena: $w(e_1'', t_0) = t_0 + \eta_0$.
9. Perėjimo operatoriai:

$H(e_1''):\ w(e_1'', t_{m+1}) = t_m + \eta_m$.

$G(e_1''):\ Y = \{y_1\}$, kur $y_1 = \{atvyko\_traukinys\}$.

3.2.3 Stoties agregatas

1. Įėjimo signalai: $X = \{x_1, x_2\},\ x_1 \in \{atvyko\_traukinys\},\ x_2 \in \{shunt\}$
2. Išėjimo signalai: $Y = \{y_1\},\ y_1 \in \{start\_pump, end\_pump\}$.
3. Išoriniai įvykiai: $E' = \{e_1', e_2'\}$.
4. Vidiniai įvykiai: $E'' = \{e_1'', e_{21}'', e_{22}''\}$.
5. Valdymo sekos: $e_1'' \to \{\eta_i\}_{i=1}^{\infty},\ e_{21}'' \to \{\xi_{1i}\}_{i=1}^{\infty},\ e_{22}'' \to \{\xi_{2i}\}_{i=1}^{\infty},\ \eta_i,\ \xi_{ij}$ – procesų trukmės.
6. Disketinės būsenų koordinatės: $\nu(t) = \{Q(t), rate_{21}(t), rate_{22}(t)\}$, $Q(t)$–eilė, kurioje yra fiksuojami traukinių atvykimo laikai, $rate_{2i}(t)$– naftos pripylimo intensyvumas iš *i*-tosios platformos.
7. Tolydinės būsenų koordinatės: $z_\nu(t) = \{w(e_1'', t), w(e_{21}'', t), w(e_{22}'', t)\}$.
8. Pradinė būsena: $\#Q(t_0) = 0$, $rate_{21}(t_0) = 0$, $rate_{22}(t_0) = 0$.
9. Perėjimo operatoriai:

$H(e_1'):\quad Q(t_{m+1}) = Enq(Enq(Q(t_m), t_m), t_m)$, kur Enq – naujo elemento operatoriaus priklausomybė nuo $Q(t\ )$.

$H(e_2'):\ Q(t_{m+1}) = Deq(Q(t)),\ w(e_1'', t_{m+1}) = t_m + \eta_m$.

$H(e_1''):\ w(e_{2i}'', t_{m+1}) = t_m + \xi_{im},\ rate_{2i}(t_{m+1}) = \xi_{im},\ i = \min\{j \mid rate_{2j}(t_m) = 0\}$.

$G(e_1''): Y = \{y_1\}$, kur $y_1 = \{start\_pump\}$.

$H(e_{2i}''):\ rate_{2i}(t_{m+1}) = \xi_{im}$,

$G(e_{2i}''): Y = \{y_1\}$, kur $y_1 = \{end\_pump\}$.

3.2.4 Valdymo agregatas

1. Įėjimo signalai: $X = \{x_1, x_2, x_3\},\ x_1 \in \{atvyko\_traukinys\},\ x_2 \in \{atvyko\_tanklaivis\}$, $x_3 \in \{start\_pump, end\_pump\}$.
2. Išėjimo signalai: $Y = \{y_1, y_2\},\ y_1 \in \{shunt\},\ y_2 \in \{start\_loading\}$.
3. Išoriniai įvykiai: $E' = \{e_1', e_2', e_{31}', e_{32}'\}$.
4. Vidiniai įvykiai: $E'' = \varnothing$.

5. Valdymo sekos: $\varnothing$.
6. Diskretinės būsenų koordinatės: $\nu(t)=\{trn(t),eng(t),plt(t)\}$, $trn(t)$–traukinių skaičius stotyje, $eng(t)$– lokomotyvo būsena, $plt(t)$– platformos būsena.
7. Tolydinės būsenų koordinatės: $z_\nu(t)=\varnothing$.
8. Pradinė būsena: $trn(t_0)=0, eng(t_0)=0, plt(t_0)=0$.
9. Perėjimo operatoriai:

$H(e_1')$: $trn(t_{m+1})=trn(t_m)+1$, jei $(eng(t_m)=0 \text{^} plt(t_m)<=1)$, tai $eng(t_{m+1})=1, plt(t_{m+1})=plt(t_m)+1$

$G(e_1')$: jei $(eng(t_m)=0 \text{^} plt(t_m)<=1)$ tai $Y=\{y_1\}$, kur $y_1=\{shunt\}$.

$H(e_2')$: $\varnothing$.

$G(e_2')$: $Y=\{y_2\}$, kur $y_2=\{start\_loading\}$.

$H(e_{31}')$: $trn(t_{m+1})=trn(t_m)-1$, jei $(trn(t_m)\neq 0 \text{^} plt(t_m)\leq 1)$ tai $plt(t_{m+1})=plt(t_m)+1$ toliau $eng(t_{m+1})=0$

$G(e_{31}')$: jei $(trn(t_m)\neq 0 \text{^} plt(t_m)\leq 1)$ tai $Y=\{y_1\}$, kur $y_1=\{shunt\}$.

$H(e_{32}')$: jei $(trn(t_m)\neq 0 \text{^} eng(t_m)=0)$ tai $eng(t_{m+1})=1$ toliau $eng(t_{m+1})=0$

$G(e_{32}')$: jei $(trn(t_m)\neq 0 \text{^} plt(t_m)\leq 1)$ tai $Y=\{y_1\}$, kur $y_1=\{shunt\}$.

3.2.5 Tanklaivio agregatas

1. Įėjimo signalai: $X=\{x_1\}$, $x_1\in\{start\_pump\}$.
2. Išėjimo signalai: $Y=\{y_1,y_2\}$, $y_1\in\{atvyko\_tanklaivis\}$, $y_2\in\{end\_pump\}$.
3. Išoriniai įvykiai: $E'=\{e_1'\}$.
4. Vidiniai įvykiai: $E''=\{e_1'',e_2''\}$.
5. Valdymo sekos: $e_1''\rightarrow\{\eta_i\}_{i=1}^{\infty}$, $e_2''\rightarrow\{\xi_i\}_{i=1}^{\infty}$, η_i, ξ_j – procesų trukmės.
6. Diskretinės būsenų koordinatės: $\nu(t)=\varnothing$.
7. Tolydinės būsenų koordinatės: $z_\nu(t)=\{w(e_1'',t),w(e_2'',t)\}$.
8. Pradinė būsena: $w(e_1'',t_0)=t_0+\eta_0$.
9. Perėjimo operatoriai:

$H(e_1')$: $w(e_1'',t_{m+1})=t_m+\eta_m$.

$G(e_1')$: $\varnothing$

$H(e_1''): \varnothing$.

$G(e_1''): Y = \{y_1\}$, kur $y_1 = \{atvyko\_tanklaivis\}$.

$H(e_2''): w(e_2'', t_{m+1}) = t_m + \xi_m$.

$G(e_2''): Y = \{y_2\}$, kur $y_2 = \{end\_pump\}$.

3.2.6 Bazės agregatas

1. Įėjimo signalai: $X = \{x_1, x_2, x_3\}$, $x_1 \in \{start\_loading\}$, $x_2 \in \{end\_loading\}$, $x_3 \in \{start\_pump, end\_pump\}$.
2. Išėjimo signalai: $Y = \varnothing$
3. Išoriniai įvykiai: $E' = \{e_1', e_2', e_3'\}$.
4. Vidiniai įvykiai: $E'' = \{e_1''\}$.
5. Valdymo sekos: $e_1'' \to \{F(tank\_rate(t))\}$.
6. Diskretinės būsenų koordinatės: $\nu(t) = \{tank\_rate(t)\}$.
7. Tolydinės būsenų koordinatės: $z_\nu(t) = \{w(e_1'', t)\}$.
8. Pradinė būsena: $tank\_rate(t_0) = start\_amount$.
9. Perėjimo operatoriai:

   $H(e_1'): tank\_rate(t_{m+1}) = tank\_rate(t_m) + traukin\_rate$.

   $H(e_2'): tank\_rate(t_{m+1}) = tank\_rate(t_m) - traukin\_rate$.

   $H(e_3')$: jei $x_3 = start\_pump$ tai $tank\_rate(t_{m+1}) = tank\_rate(t_m) - tanklaivis\_rate$

   toliau $tank\_rate(t_{m+1}) = tank\_rate(t_m) + tanklaivis\_rate$

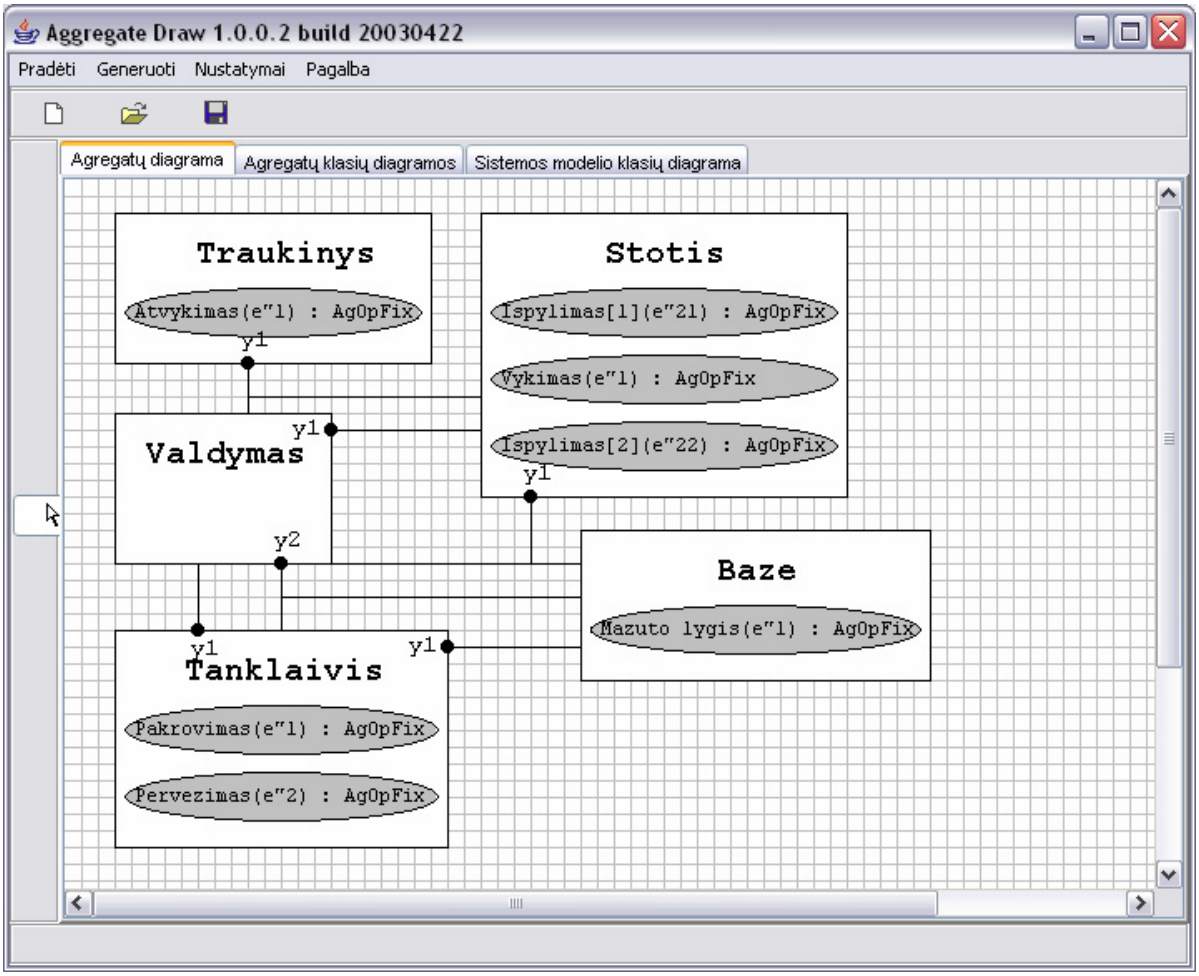
   $H(e_1'', t_m): \varnothing$.

3.3 Imitacinio modelio programinė realizacija

Norint sukurti kokios nors sistemos modelį ir atlikti mums domimus skaičiavimus kartais tenka realizuoti programą imituojančią tą sistemą ir atlikti skaičiavimus. Kadangi imituojama sistema būna iš anksto suprojektuota ir atvaizduota diagramomis, vartotojui tenka pačiam susiprogramuoti pagal tas diagramas sistemą modeliuojančia imitacinę programa. Tai įveda daug nuobodaus ir nekūrybingo darbo. Programa „Agregatų Braižyklė“ (AgDraw) palengvins šitą darbą ir leis grafiniu būdu nusibraižyti sistemos diagramas ir iš jų susigeneruoti modeliuojamą sistemą imituojančią programą.

3.3.1 Imitacinio modelio agregatinės sujungimo schemos sudarymas

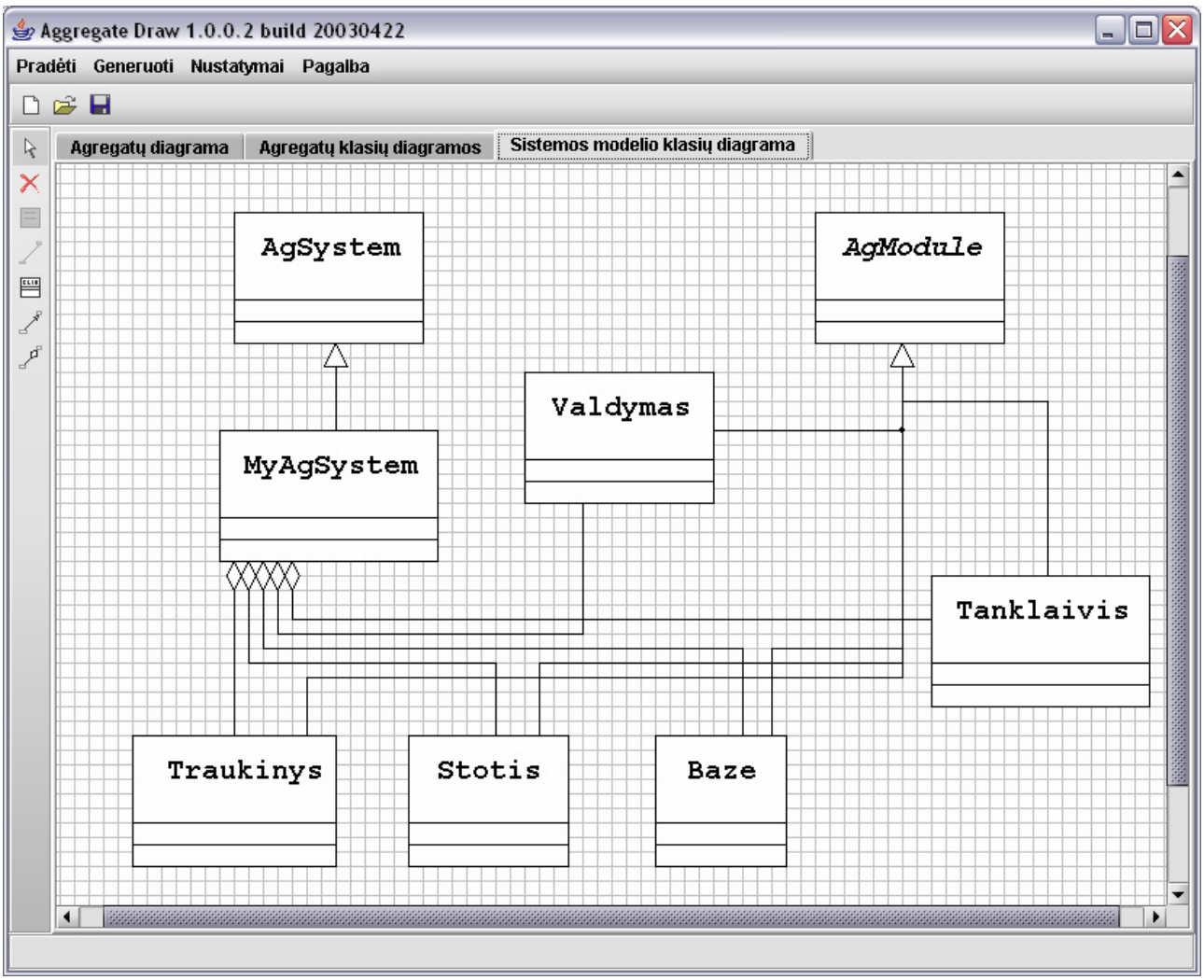
Naudojantis AgDraw priemone, sukuriami agregatai, remiantis 3.2.1 pav. schema ir sujungiami atitinkamais kanalais, kaip parodyta 3.3.1. pav.



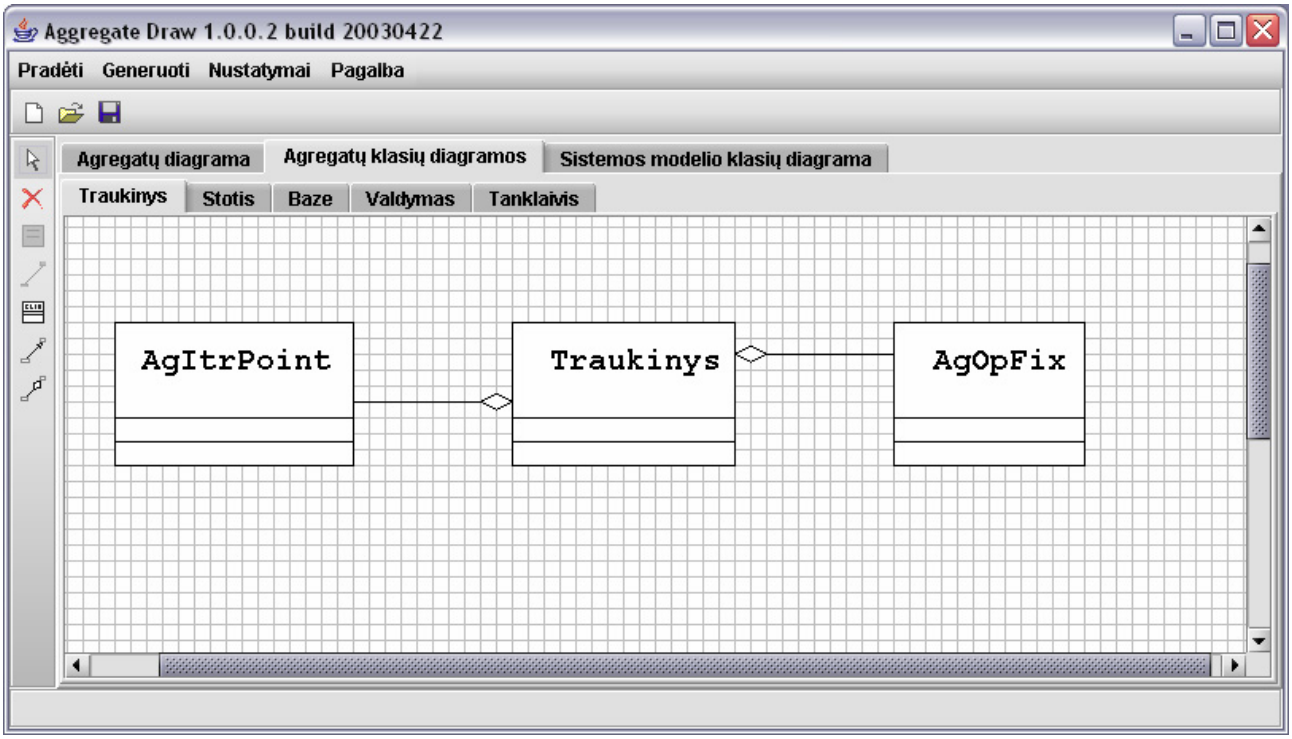
3.3.1 pav. Imitacinio modelio agregatinė schema

3.3.2. Imitacinio modelio programinės realizacijos klasių modelio sudarymas

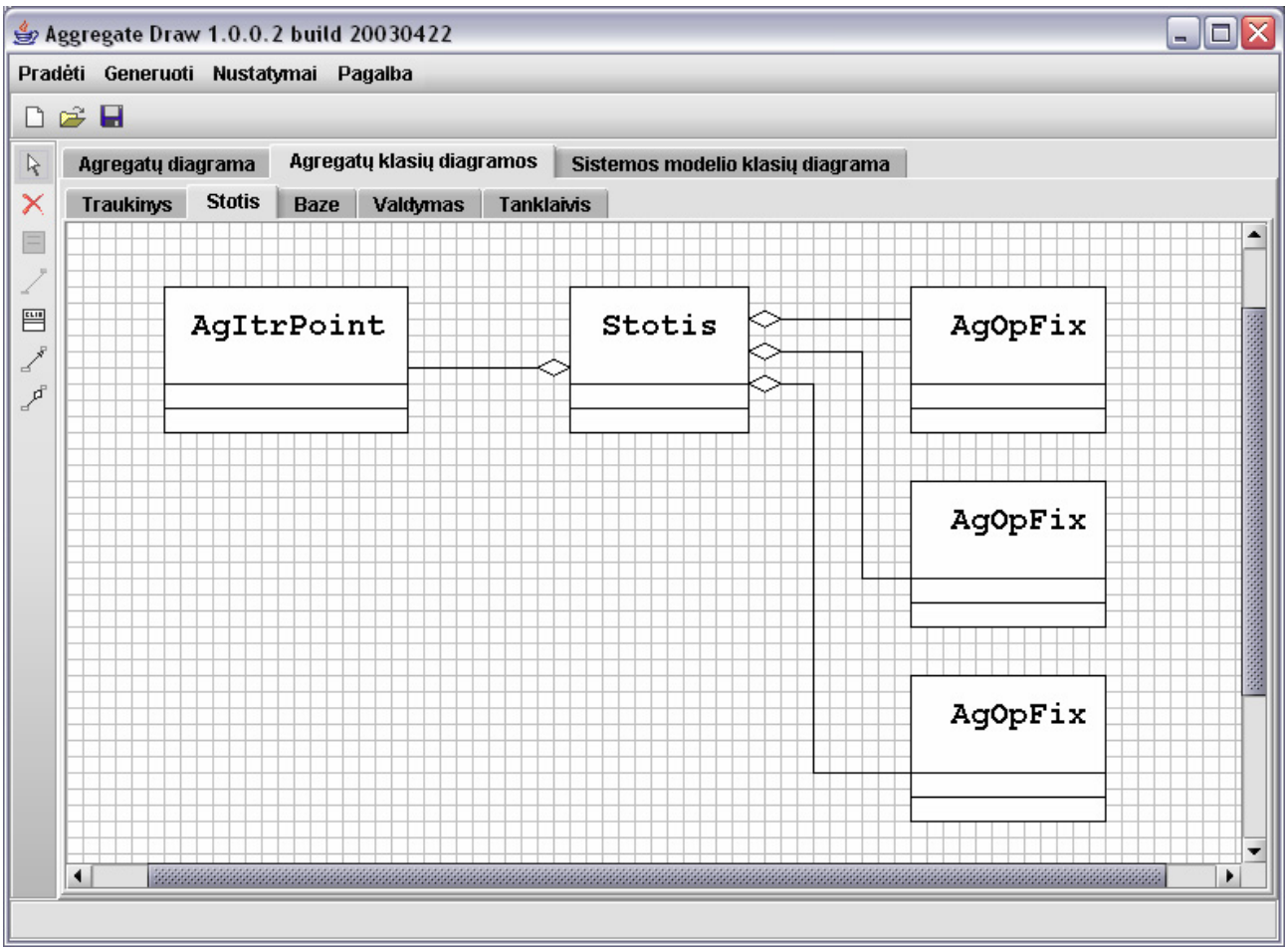
Pagal sudarytą 3.3.1 pav. agregatinę sujungimo schemą, AgDraw priemonė automatiškai sugeneruoja programinės realizacijos klasių diagramas. Bendras klasių modelis pavaizduotas 3.3.2 pav. 3.3.2.1 pav. vaizduojamas imitacinio modelio programinės realizacijos Traukinio klasės modelis, 3.3.2.2 pav. – imitacinio modelio Stoties klasės modelis, 3.3.2.3 pav. – imitacinio modelio Bazės klasės modelis, 3.3.2.4 pav**.** – imitacinio modelio Valdymo klasės modelis, 3.3.2.5 pav**.** – imitacinio modelio Tanklaivio klasės modelis.



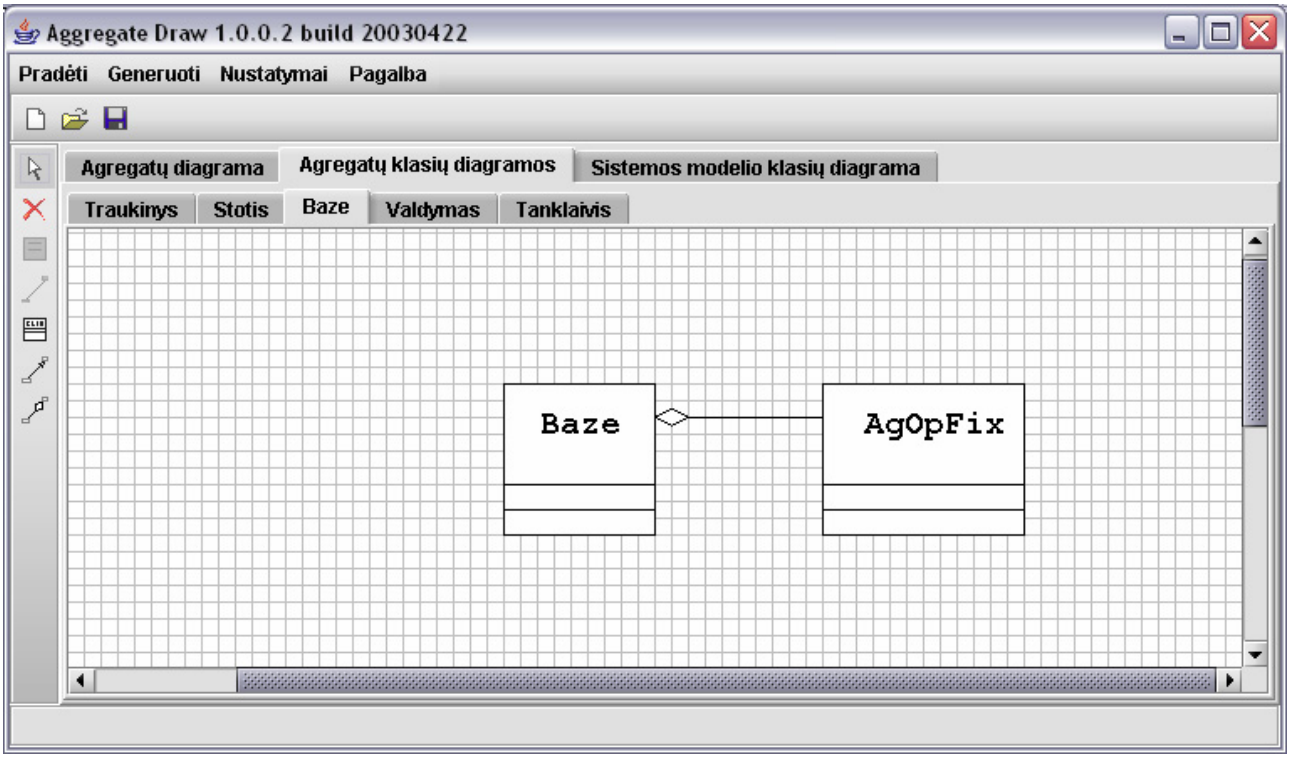
3.3.2 pav. Imitacinio modelio programinės realizacijos klasių modelis



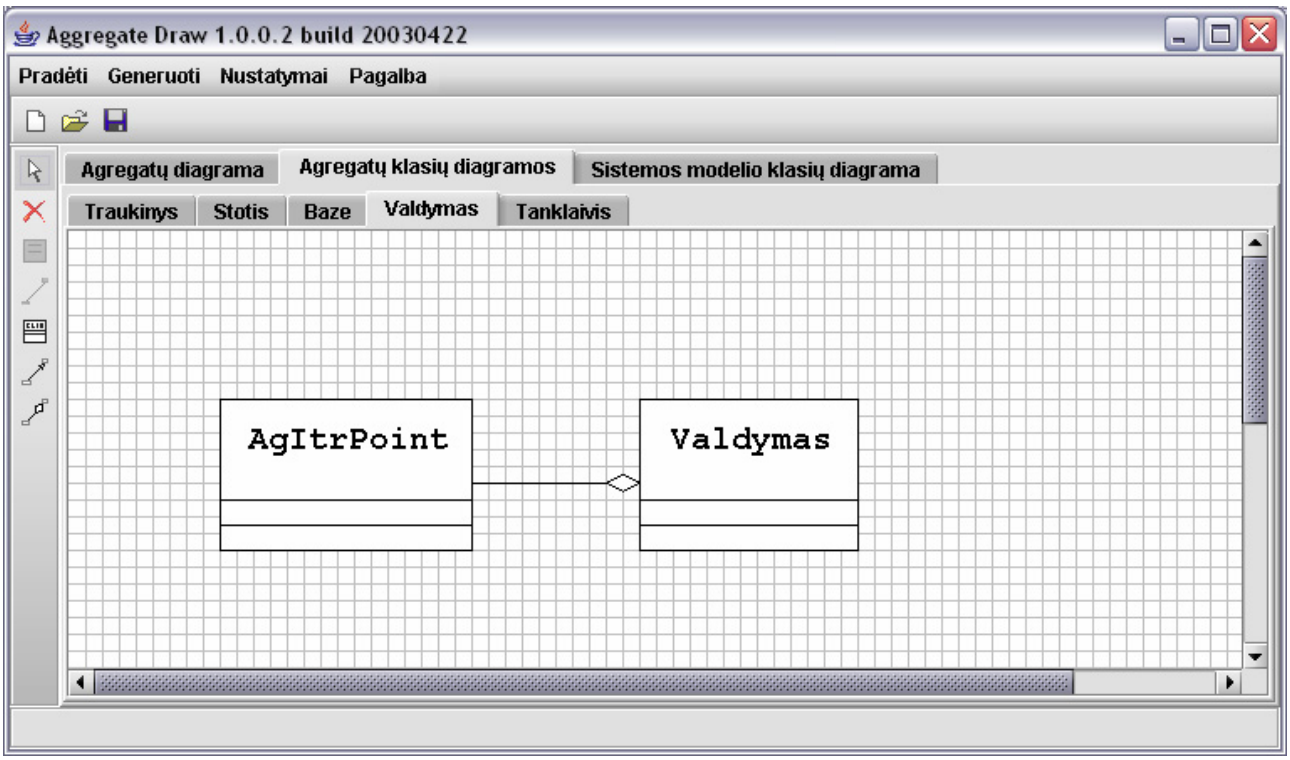
3.3.2.1 pav. Imitacinio modelio programinės realizacijos Traukinio klasės modelis



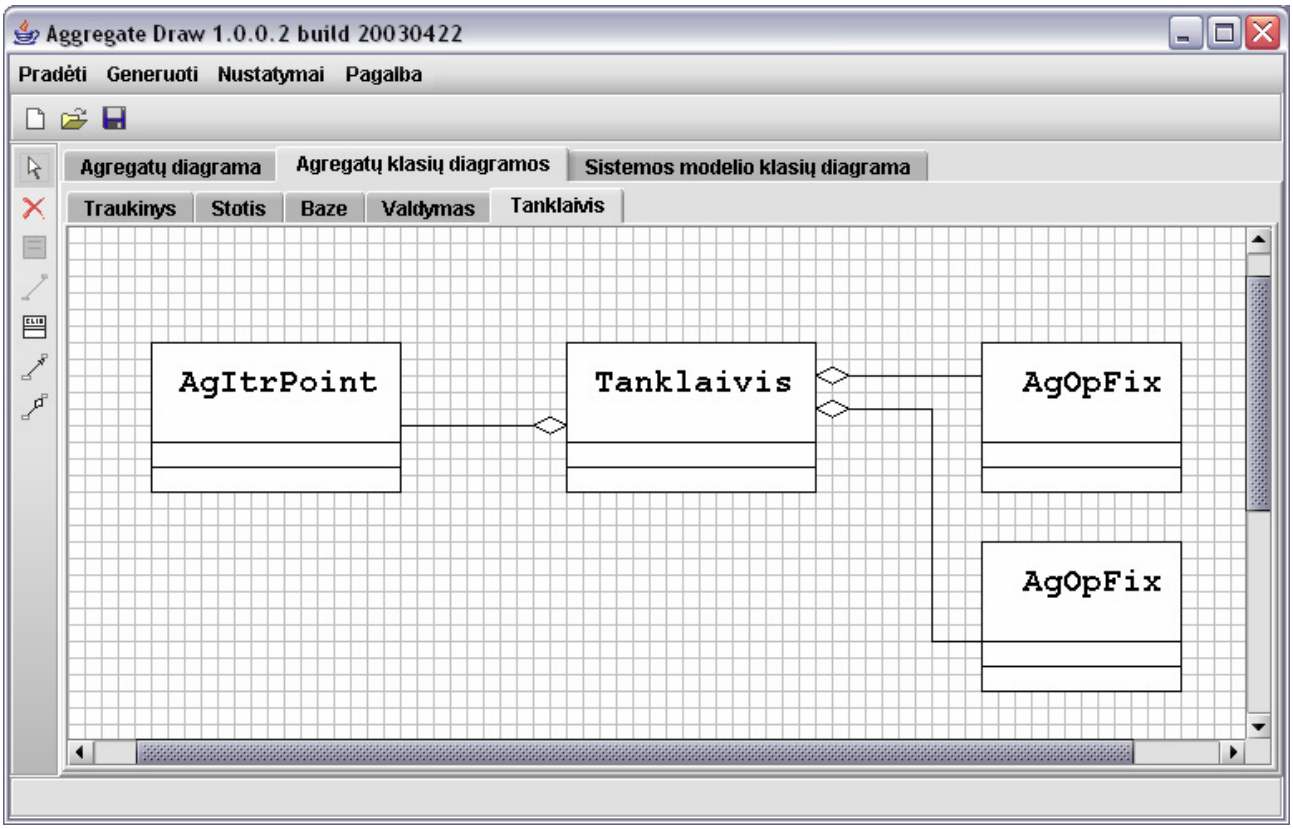
3.3.2.2 pav. Imitacinio modelio programinės realizacijos Stoties klasės modelis



3.3.2.3 pav. Imitacinio modelio programinės realizacijos Bazės klasės modelis



3.3.2.4 pav. Imitacinio modelio programinės realizacijos Valdymo klasės modelis



3.3.2.5 pav. Imitacinio modelio programinės realizacijos Tanklaivio klasės modelis

3.3.3 Imitacinio modelio programinio kodo sudarymas

Pagal 3.3.2 skyrelyje sudarytas modelio klasių diagramas, AgDraw priemonė sugeneruoja imitacinio modelio programinio kodo karkasą. Šio modelio programinį kodą sudaro tokios Java klasės (Imitacinio modelio programos kodai pateikti priede 7.1):

- Eksperimentas – pagrindinė imitacinio modelio klasė, realizuojanti imitacinio modelio eksperimento valdymą (Priedas 7.1.1).
- Modelis – realizuoja agregatų sujungimo schemą (Priedas 7.1.2).
- Traukinys – realizuoja traukinio agregatą, kuris modeliuoja traukinių atvykimą į geležinkelio stotį (Priedas 7.1.3).
- Stotis – realizuoja stoties agregatą, kuris modeliuoja atvykstančių traukinių priėmimą geležinkelio stotyje (Priedas 7.1.4).
- Valdymas – realizuoja valdymo agregatą, kuris modeliuoja traukinių vagonų paskirstymą į naftos produktų perpylimo estakadas (Priedas 7.1.5).
- Bazė – realizuoja bazės agregatą, kuris modeliuoja rezervuarų užpildymą naftos produktais (Priedas 7.1.6).
- Tanklaivis – realizuoja tanklaivio agregatą, kuris modeliuoja naftos produktų pakrovimą į tanklaivius (Priedas 7.1.7).

Turint šio modelio karkasą Java kalboje, penkios paskutinės klasės buvo papildytos agregatų perėjimo – išėjimo operatoriais. Šių operatorių matematiniai modeliai pateikti 3.2 skyrelyje. Tokiu būdu buvo gautas visas imitacinio modelio programinis kodas.

3.3.4 Mazuto perkrovimo proceso imitacinio modeliavimo rezultatai

Panaudojus 3.3.3 skyrelyje sukurtą naftos terminalo imitacinio modelio programą, buvo atlikti imitacinio modeliavimo eksperimentai. Buvo modeliuojamas naftos terminalo darbas perkraunant mazutą, esant įvairioms lauko temperatūroms nuo -10ºC iki +10ºC. Perpilant mazutą iš geležinkelio vagonų į rezervuarus, lauko temperatūra turi labai didelę įtaką, nes mazutas, priklausomai nuo esamo jame sieros kiekio, tirštėja jau esant vidutiniškai +20ºC temperatūrai. Esant žemesnei lauko temperatūrai, ilgiau pastovėjusius traukinių vagonus geležinkelio stotyje, tenka šildyti ir tik tada perpilti į rezervuarus. Vagonų šildymas brangiai kainuoja ir padidina naftos terminalo mazuto perkrovimo išlaidas. Todėl yra labai svarbu naftos terminale priimti tiek mazuto vagonų, kiek bus spėjama iškrauti, kad nesusidarytų eilės ir neatvėstų mazutas žemiau +20ºC temperatūros. Todėl šiame eksperimente buvo parinkta temperatūra nuo -10ºC iki +10ºC, ir keičiamas atvykstančių traukinių intensyvumas. Atvykstančių traukinių intensyvumas

matuojamas santykiniu dydžiu λ. Intensyvumo reikšmė 1 santykinai rodo maksimalų terminalo apkrovimą, intensyvumo reikšmė 0,5 reiškia, kad į naftos terminalą atvyksta 50 proc.galimų vagonų per parą, kuriuos galėtų iškrauti terminale, esant tam tikrai lauko temperatūrai.

Visuomet yra taikomas metodas – dirbti pelningai, t.y. gauti numatytą pelną, kartu reguliuoti kainų tarifus, siekiant kad jie būtų konkurencingi.

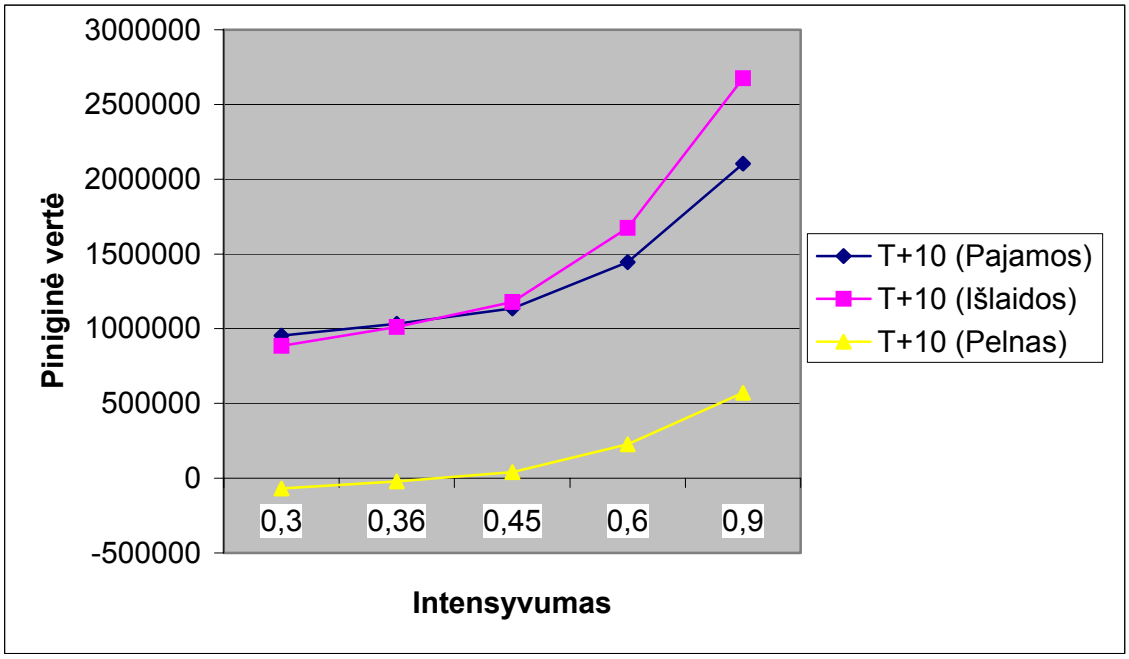
Modeliavimo metu buvo skaičiuojamos gaunamos pajamos už mazuto perkrovimą, bei naftos terminalo išlaidos mazutui perkrauti. Šių dydžių skirtumas duoda pelną, esant skirtingiems intensyvumams, bei esant skirtingoms temperatūroms.

1 lentelėje pateikiami rezultatai, kai T=+10ºC

1 lentelė

λ	T+10 (Pajamos)	T+10 (Išlaidos)	T+10 (Pelnas)
0,3	954000	886400	-67600
0,36	1032000	1011200	-20800
0,45	1136000	1177600	41600
0,6	1446000	1673600	227600
0,9	2103115	2675200	572085

Pajamų, išlaidų ir pelno grafinė išraiška pavaizduota 3.3.4.1 pav., kai lauko temperatūra T=+10ºC



3.3.4.1 pav. imitacinio modelio eksperimento grafikas, kai T=+10ºC

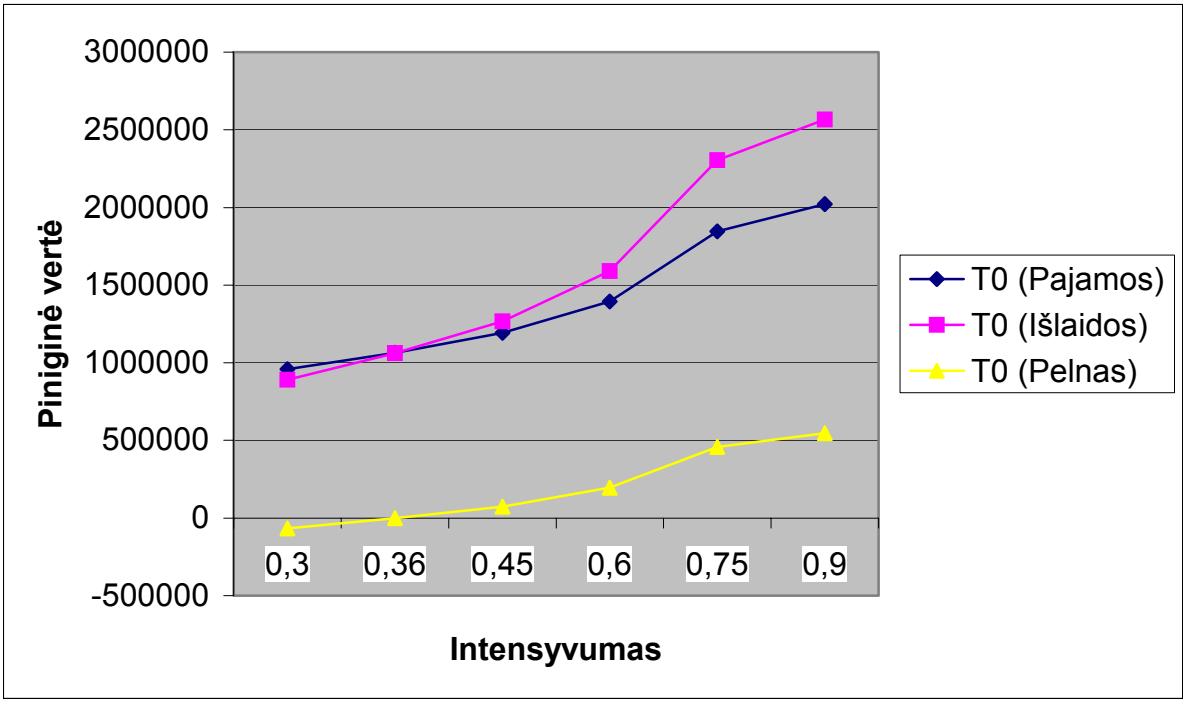
Atliekant analizę, reikia rasti optimalius sprendimus analizuojamam imitaciniam modeliui.

2 lentelėje pateikiami rezultatai, kai T=0ºC

2 lentelė

λ	T0 (Pajamos)	T0 (Išlaidos)	T0 (Pelnas)
0,3	958000	891200	-66800
0,36	1064000	1062400	-1600
0,45	1192000	1 265 600	73600
0,6	1394059	1590400	196341
0,75	1846671	2304000	457329
0,9	2021006	2566400	545394

Pajamų, išlaidų ir pelno grafinė išraiška pavaizduota 3.3.4.2 pav., kai lauko temperatūra T=0ºC



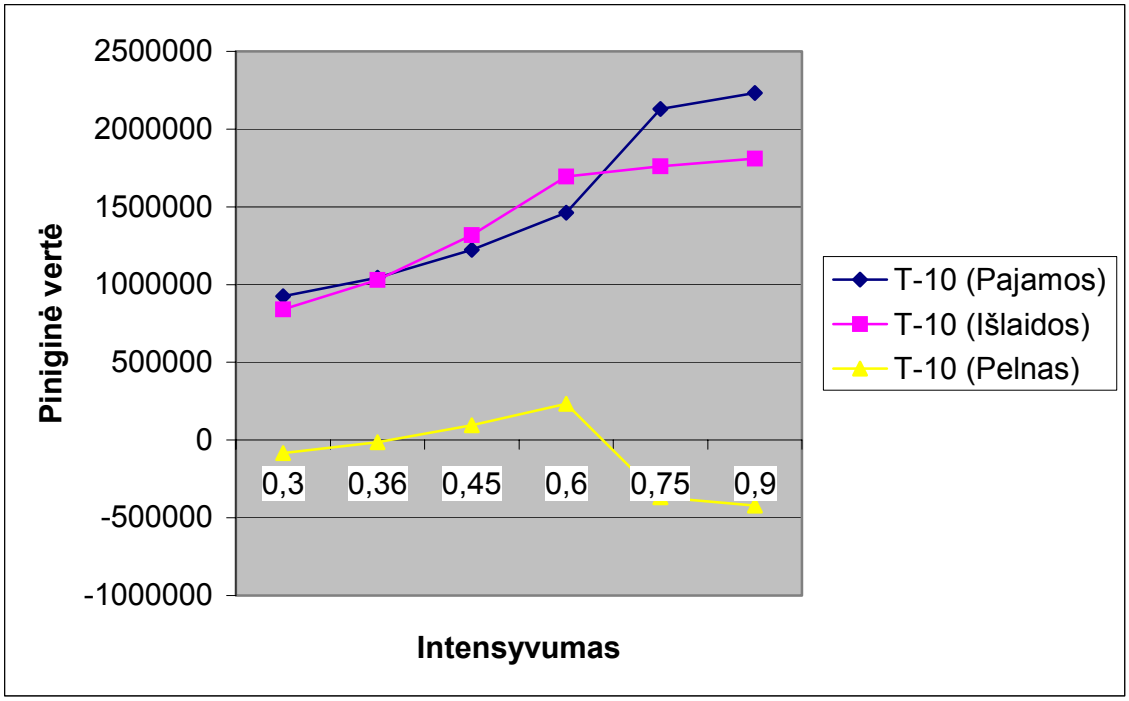
3.3.4.2 pav. imitacinio modelio eksperimento grafikas, kai T=0ºC

3 lentelėje pateikiami rezultatai, kai T=-10ºC

3 lentelė

λ	T-10 (Pajamos)	T-10 (Išlaidos)	T-10 (Pelnas)
0,3	926000	841600	-84400
0,36	1044029	1030400	-13629
0,45	1224000	1318400	94400
0,6	1462438	1696000	233562
0,75	2129028	1761600	-367428
0,9	2231956	1811200	-420756

Pajamų, išlaidų ir pelno grafinė išraiška pavaizduota 3.3.4.3 pav., kai lauko temperatūra T=-10ºC



3.3.4.3 pav. imitacinio modelio eksperimento grafikas, kai T=-10ºC

Reikia nustatyti ribą, kad mazuto perkrovimo procesas nebūtų nuostolingas, o pasiekus pelno zoną, reguliuoti traukinių priėmimą traukinių stotyje taip kad darbas vyktų sklandžiai. Traukinių priėmimo santykinis intensyvumas gali siekti nuo minimalios iki maksimalios, t.y lygios vienetui vertės. Atlikus analizę, remiantis pajamų, išlaidų ir pelno grafinėmis išraiškomis matyti, kad esant aukštesnei lauko temperatūrai pvz. +10ºC ir daugiau, traukinių priėmimo apkrovimas gali būti maksimalus,t.y iki 100proc., priešingu atveju, esant žemai lauko temperatūrai pvz. -10ºC ir mažiau, traukinių priėmimo darbas bus efektyvus ir pelningas, jei intensyvumas sieks nuo 0,4 iki 0,6, t.y į traukinių stotį atvyktų traukinių nuo 40proc. iki 60 proc.

Mazuto perkrovimo proceso imitaciniame modelyje svarbiausi kriterijai yra perkrovimo kaina ir aptarnavimo laikas. Galima daryti prielaidą, kad mazuto perkrovimo proceso imitacinio modelio darbo našumą ir pelningumą nusako optimalus traukinių priėmimas traukinių stotyje, atsižvelgiant į esamą lauko temperatūrą.

4. VARTOTOJO DOKUMENTACIJA

4.1 Mazuto perkrovimo proceso imitacinio modeliavimo sistemos funkcinis aprašymas

Mazuto perkrovimo proceso imitacinio modeliavimo sistema yra skirta modeliuoti naftos produktų priėmimą geležinkeliu į naftos terminalą. Modeliavimo sistema modeliuoja tokius procesus:

1. Į terminalą naftos produktai pristatomi traukiniais.
2. Traukiniai laukia iškrovimo traukinių stotyje.
3. Traukinių iškrovimo valdymą atlieka naftos terminalo dispečeris.
4. Pagal dispečerio duotas komandas, vagonai yra perstumiami į išpylimo estakadas, kurių yra dvi.
5. Iškrovimo trukmė iš vagonų į terminalo rezervuarus priklauso nuo mazuto temperatūros traukinių cisternose. Modelyje priimta, kad visi traukiniai atvyksta esant apie 45˚C temperatūrai.
6. Traukiniai iškraunami pagal jų atvykimo eilę.
7. Tanklaivio pakrovimas yra atliekamas prie atitinkamos krantinės, kuri yra viena.
8. Tanklaivio pakrovimo laikas priklauso tik nuo perkraunamo mazuto kiekio.
9. Tanklaivio pakrovimo valdymą atlieka tas pats dispečeris.

Sukurtas modelis leidžia keisti tokius modelio parametrus:

1. Traukinių atvykimo į geležinkelio stotį intensyvumą.
2. Lauko temperatūrą.

4.2 Mazuto perkrovimo proceso imitacinio modeliavimo sistemos vartotojai

Modeliavimo sistema yra skirta vienam vartotojui, kuris atlieka imitacinius eksperimentus su sukurtu modeliu.

4.3 Mazuto perkrovimo proceso imitacinio modeliavimo sistemos darbo aplinka

Programinė įranga dirba Microsoft® Windows™ (95, 98, 2000, XP, 2003) ir Linux (reikalingas XWindows serveris) operacinėse sistemose, jei ten bus įdiegta Sun Microsystems Java Virtal Machine (JVM) 1.4.1 ar naujesnė versija.

4.4 Mazuto perkrovimo proceso imitacinio modeliavimo sistemos realizavimo apribojimai

Modelis realizuotas Java programavimo kalba, naudojama Sun Microsystems Java Development Kit (JDK) versija 1.4.1. Jokių kitų apribojimų nėra.

4.5 Mazuto perkrovimo proceso imitacinio modeliavimo sistemos techninės įrangos reikalavimai

- Intel ar 100% suderinamas procesorius nuo 166MHz,
- 32MB atminties,
- mažiausiai 371 MB laisvos vietos kietajame diske.

4.6 Mazuto perkrovimo proceso imitacinio modeliavimo sistemos diegimas

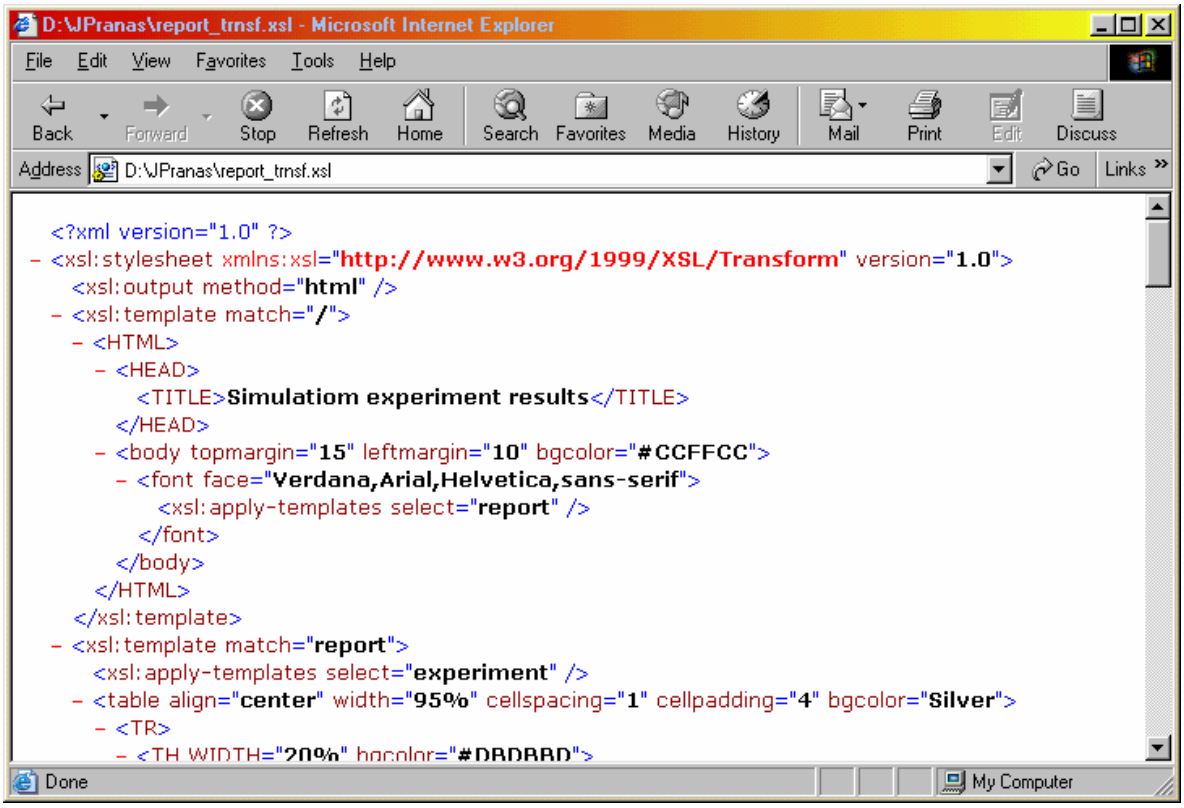
Sistema yra diegiama failų kopijavimo būdu. Iš cdrom į darbinį diską turi būti perkopijuota JPranas direktorija. Turi būti atitinkamai paredaguoti make.bat, build.bat, run.bat komandiniai failai. Komanda SET.PATH nurodo, kokioje direktorijoje yra instaliuojama Java virtuali mašina.

4.7 Mazuto perkrovimo proceso imitacinio modeliavimo sistemos programos vykdymas

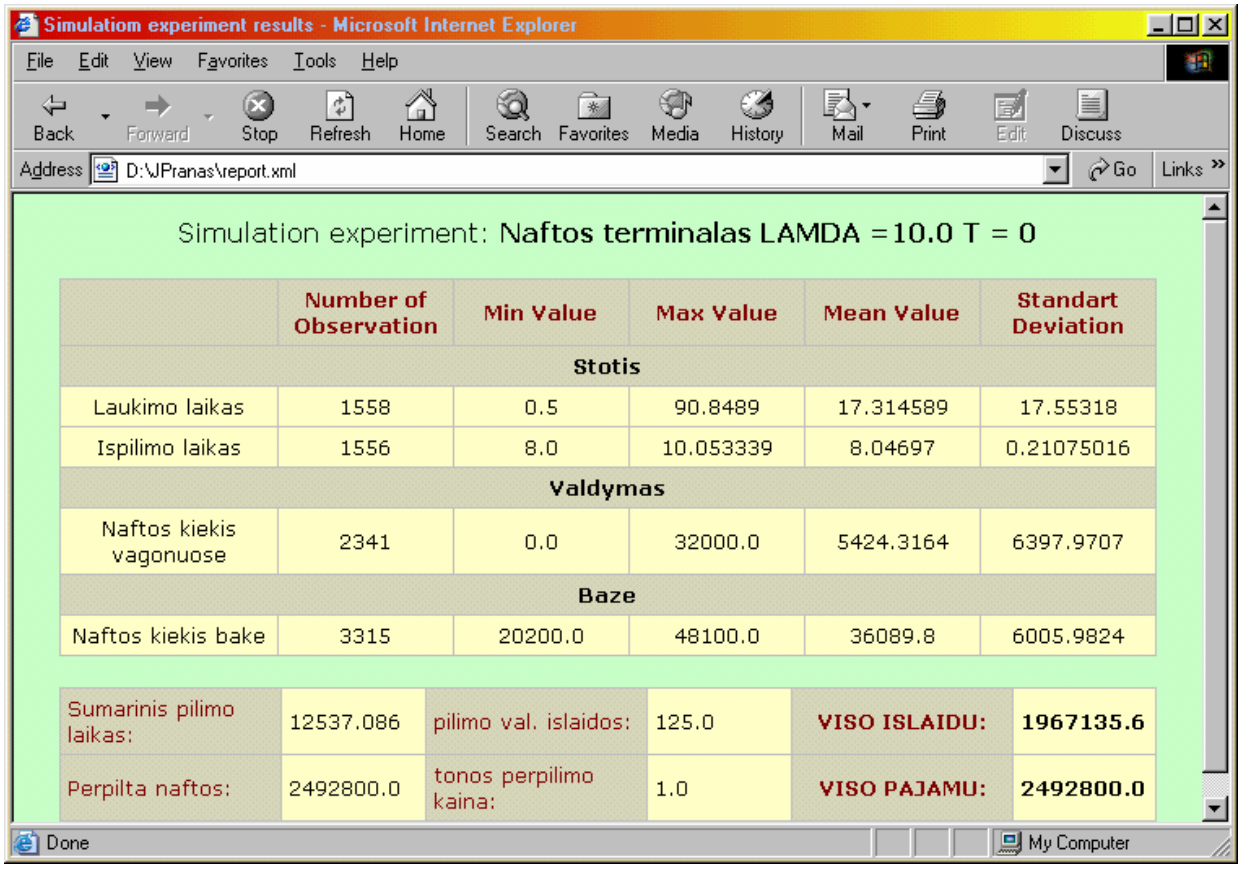
Mazuto perkrovimo proceso imitacinio modeliavimo programa yra paleidžiama iš programinio failo run.bat. Programai pabaigus darbą, yra sukuriamas modeliavimo rezultatų failas report.xml. Rezultatus galima pasižiūrėti standartine interneto naršykle, arba bet kuriuo teksto redaktoriumi.

Pirmuoju atveju (4.7.1 pav), panaudojus interneto naršyklę kaip Microsoft Internet explorer, kartu yra atliekama failo report.xml duomenų XSL transformacija. Ši transformacija atliekama pagal report_trnsf.xsl faile apibrėžtą algoritmą.

4.7.2 pav. pavaizduoti tie patys rezultatai, kaip ir 4.7.1 pav., tik vartotojui patogioje formoje, atlikus minėtą XSL transformaciją. Šios transformacijos aprašymą vartotojas gali keisti pagal savo pageidavimus.



4.7.1 pav. Imitacinio modeliavimo programos vykdymo rezultatai XSL forma



4.7.2 pav. Imitacinio modeliavimo programos vykdymo rezultatai XML forma

4.8. Mazuto perkrovimo proceso imitacinio modeliavimo sistemos testavimo dokumentas

4.8.1 Mazuto perkrovimo proceso imitacinio modelio instaliavimas

Iš cdrom nukopijuojama JPranas direktorija. Paredaguojamas komandinis failas run.bat taip:

„<kelias į Java virtualią mašiną>\java“-classpath.;agsim.jar Terminal/Eksperimentas

4.8.2 Mazuto perkrovimo proceso imitacinio modelio paleidimas

Įvykdomas imitacinis eksperimentas, iškviečiant run.bat komandinį failą. Notepad pagalba patikrinamas report.xml failo turinys, kuris turi būti toks, koks pateiktas priede 7.2.

Internet Explorer pagalba patikrinamas rezultatų transformacijos teisingumas (gaunamas vaizdas turi būti kaip 4.7.2 pav.).

5. IŠVADOS IR GAUTI REZULTATAI

1. Atlikus naftos produktų perkrovimo proceso analizę naftos terminale, buvo sukurtas šio proceso agregatinis modelis, kurį sudaro atitinkami agregatai:

1. Traukinio agregatas, realizuojantis traukinių atvykimą į geležinkelio stotį.

2. Stoties agregatas, realizuojantis atvykstančių traukinių priėmimą geležinkelio stotyje.

3. Valdymo agregatas, realizuojantis vagonų paskirstymą į naftos produktų perpylimo estakadas.

4. Bazės agregatas, realizuojantis rezervuarų užpildymą naftos produktais.

5. Tanklaivio agregatas, realizuojantis naftos produktų pakrovimą į tanklaivius.

2. Panaudojus objektiškai orientuotą imitacinio modeliavimo biblioteką JPranas ir „Agregatų Braižyklė“ programą, buvo sukurta imitacinio modelio programinė realizacija Java programavimo sistemoje.

3. Sukurtas modelis leidžia modeliuoti naftos terminalo darbą perkraunant mazutą, esant skirtingoms lauko temperatūroms ir skirtingiems atvykstančių traukinių srautams.

4. Atlikus eksperimentus su sukurtu modeliu, gautos naftos terminalo pajamų, išlaidų ir pelno grafinės išraiškos, perkraunant mazutą.

5. Šio sukurto mazuto perkrovimo imitacinio modelio pagalba gali būti atliekami eksperimentai keičiant traukinių intensyvumą ir lauko temperatūrą. Galima nustatyti konkrečiu atveju koks yra reikalingas terminalo darbo apkrovimas, t.y kiek gali būti priimama traukinių atvykusių į geležinkelio stotį, kad mazuto perkrovimo darbas būtų pelningas. Nustatyta, kad esant -10ºC temperatūrai, naftos terminalas gaus pelną už mazuto perkrovimą esant nuo 40 iki 60 proc.gamybiniam pajėgumui. Esant aukštesnei lauko temperatūrai pvz. +10ºC ir daugiau, traukinių priėmimo apkrovimas gali būti maksimalus, t.y iki 100proc., tokiu atveju traukinių priėmimo darbas bus efektyvus ir pelningas.

6. LITERATŪROS SĄRAŠAS

1. Paulauskas V. Uostų plėtra. Klaipėda, 2000, 27p.

2. Blumel E., Novitsky L and other. Applications of simulation and IT solutions I the Baltic Port areas of the associated candidate countries. Riga, Latvia, 2003, 81-90p.

3. Novitsky L., Ginters E., Merkutyev Y., Merkuryeva G. and Ragozin V. Industrial customization of Maritime simulation and information systema in the Latvian port areas; In: proceedings of the ESS 2001 simulation in industry, 13th European simulation symposium, Marseille, France, october 18-20, 2001, 78p.

4. Giribone P. and Bruzzone G.A., Corbone A. Harbour services and lay-out re-engineering by using simulation. In proceedings of the international workshop „Modelling and simulation within a Maritime environment“. Riga, september 6-8.

5. Pranevičius H. Formal specification and analysis of distributed systems // Journal of Intelligent Manufacturing, N.9, 1998, p. 23-3l.

6. Pranevičius H., Pilkauskas V., Chmieliauskas A. Aggregate approach for specification and analysis of computer network protocols. Kaunas: Technologija, 1994, 152 p.

7. Pranevičius H., Pilkauskas V., Makackas D. The use UML and aggregate approach for developing Klaipėda oil simulation model // Simulation und Visualisierung , 28 February - 1 March, 2002, Magdeburg, p. 161-171.

8. Lapinskas Č. Kodėl mums reikalingas naftos terminalas. Lietuvos informacijos institutas, 1992, 33p.

Vaida Kasubiene. The simulation model of reloading of oil at an oil terminal: Master's work / supervisor doc. doctor V. Pilkauskas; Kaunas University of Technology. – Kaunas, 2004.– 58p.

SUMMARY

The purpose of the final work is forming a simulation model of the process of reloading of oil in the oil terminal that would let make a simulation model of accepting oil products by railway to the oil terminal.

In order to fulfil the purpose of the final work there were some tasks risen: a conceptual model of reloading process of oil was made, a mathematical model of reloading process of oil was made, too, an aggregate diagram of the simulation model was made an the models of the classes of the program realization were shown as well.

The object of the final work is the simulation model of the process of reloading oil created in the oil terminal.

Having used an object–oriented simulation system library of simulation modelling JPranas and AgDraw program, the programming realization of simulation model in Java programming system was created.

The results of the final work: experiments with the created simulation model of reloading process of oil in order to find out the optimum solution have been created. The created simulation model allows changing these parameters of the model:

1. The intensity of arriving of trains at the railway station.
2. The temperature outside.

During modelling the returns got for reloading oil were bring counted and, on the other hand, expenses of oil terminal for reloading oil were being counted as well. The graphic expression of the returns, the expenses and the profits showing the optimum industrial capacity of oil terminal have been presented. The intensity of trains for the effective and profitable work in reloading oil at an oil terminal has been established.

Key words: simulation modelling, oil terminal, reloading of oil, aggregate specification.

8. TERMINŲ IR SANTRUPŲ ŽODYNAS

PPS – uosto proceso modeliavimo sistema (Port Process Simulator).

DEVS – diskrečių įvykių sistemos specifikacijos.

PLA – atkarpomis tiesinis agregatas (Piece-Linear Agregate). Vienas iš sistemos formalaus aprašymo metodų, besiremiantis atkarpomis tiesiniais procesais.

MAS – masinio aptarnavimo sistema.

UML – unifikuota modeliavimo kalba, įgalinanti aprašyti ir vizualizuoti įvairias objektinės orientacijos sistemas (Unified Modeling Language).

XML – duomenų aprašymo standartas (eXtensible Markup Language).

JVM – Javos virtuali mašina, sluoksnis tarp programos ir operacinės sistemos leidžiantis tą pačia programą vykdyti įvairiose operacinėse sistemose (Java Virtual Machine).

UML klasių diagrama – diagrama, kurioje nubraižytos klasės, pavaizduoti ryšiai tarp jų.

AgDraw – „Agregatų Braižyklė“ programa, skirta sistemų imitaciniams modeliams sudaryti.

9. PRIEDAI

9.1 Imitacinio modelio programos kodai

9.1.1 Eksperimentas programos kodas

```
package Terminal;
import ktu.vik.jpranas.jsimulation.*;

public class Eksperimentas {
    final public static double MODLAIKAS = 8000.0;
    final public static double LAMDA = 10.0;
    final public static int LAUKO_TEMPERATURA = 0;
    final public static double PERPILIMO_KAINA = 1.0;
    final public static double VALPERPILIMO_ISLAIDOS = 125.0;
    final public static double VALPASTOVIOS_ISLAIDOS = 50.0;
    final public static double PASTOVIOSISLAIDOS = VALPASTOVIOS_ISLAIDOS *
MODLAIKAS;

    /** Creates new Eksperimentas */
    public Eksperimentas() {
    }
    public static void main (String args[]) {
        agSystem.SetDuration(MODLAIKAS);
        agSystem.SetPatch("./");
        agSystem.SetTitle("Naftos terminalas LAMDA ="+String.valueOf(LAMDA)+
            " T = "+String.valueOf(LAUKO_TEMPERATURA));
        Modelis testas = new Modelis();
        testas.Initialize();
//        testas.Debug();
        testas.Exec();
        testas.ReportTable();
    }
}
```

9.1.2 Modelis programos kodas

```
package Terminal;
import ktu.vik.jpranas.jsimulation.*;

public class Modelis extends agSystem{
    private Traukinys traukinys;
    private Stotis stotis;
    private Valdymas valdymas;
    private Baze baze;
    private Tanklaivis tanklaivis;

    /** Creates new Modelis */
    public Modelis() {
        traukinys = new Traukinys(this);
        stotis = new Stotis(this);
        valdymas = new Valdymas(this);
        baze = new Baze(this);
        tanklaivis = new Tanklaivis(this);
        traukinys.y1.Connect(stotis);
        traukinys.y1.Connect(valdymas);
        stotis.y1.Connect(valdymas);
        stotis.y1.Connect(baze);
        valdymas.y1.Connect(stotis);
        valdymas.y2.Connect(baze);
        valdymas.y2.Connect(tanklaivis);
        tanklaivis.y1.Connect(valdymas);
```

```
        tanklaivis.y2.Connect(baze);
        tanklaivis.y2.Connect(valdymas);
    }
}
```

9.1.3 Traukinys programos kodas

```
package Terminal;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;


public class Traukinys extends agModule{

    public agItrPoint y1;
    final public static int ATVYKO = 11;
    final public static double SASTATO_TALPA = 3200;
    final public static double GRVAGONU_TALPA = SASTATO_TALPA/2;
    final public static double ISPILIMO_LAIKAS = 8.0;

    private double TX[];
    int IX;

    private agOpExp op1;
    private agMessage p1;
    final private static int ATVYKIMAS = 1;

    /** Creates new Gelezinkelis */
    public Traukinys(agSystem sm) {
        super(sm,"Traukinys");
        y1 = new agItrPoint(this);
        op1 = new agOpExp(this,ATVYKIMAS,Eksperimentas.LAMDA);
        p1 = new agMessage(ATVYKO);
    }
    public void Initialize(){
/*        TX = new double[10];
        TX[0] = 1.0;
        TX[1] = 11.0;
        TX[2] = 11.0;
        TX[3] = 4.0;
        TX[4] = 4.0;
        TX[5] = 4.0;
        TX[6] = 35.0;
        TX[7] = 35.0;
        TX[8] = 25.0;
        TX[9] = 25.0;
        IX=0;
        op1.Start(TX[IX]);  IX++;
 */
        op1.Start();
    }
    public void InternalEvent(agInternalEvent ev){
        y1.Output(p1);
        op1.Start();
//        op1.Start(TX[IX]);  IX++;
    }

}
```

9.1.4 Stotis programos kodas

```
package Terminal;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;

public class Stotis extends agModule{

    public agItrPoint y1;
    final public static int ISPILIMAS_PRADETAS = 21;
    final public static int ISPILIMAS_BAIGTAS = 22;

    private agOpFix manevravimas;
    private agOpFix ispilimas;
    private Factory_Ispilimas factoty;
    private agQueue q1;
    private Message_Ispilimas ispilimas_pradetas,ispilimas_baigtas;
    final private static int MANEVRAVIMAS = 1;
    final private static int ISPILIMAS = 2;
    private agStatD laukimo_laikas;
    private agStatD ispilimo_laikas;
    private double sum_ispilimo_laikas;
    private double tl;

    public Stotis(agSystem sm) {
        super(sm,"Stotis");
        y1 = new agItrPoint(this);
        manevravimas = new agOpFix(this,MANEVRAVIMAS,0.5);
        ispilimas = new agOpFix(this,ISPILIMAS,Traukinys.ISPILIMO_LAIKAS,2);
        factoty = new Factory_Ispilimas(ispilimas);
        q1 = new agQueue();
        ispilimas_pradetas = new Message_Ispilimas(ISPILIMAS_PRADETAS);
        ispilimas_baigtas = new Message_Ispilimas(ISPILIMAS_BAIGTAS);
        laukimo_laikas=new agStatD(this,"Laukimo laikas");
        ispilimo_laikas=new agStatD(this,"Ispilimo laikas");
    }
    public void Initialize(){
        sum_ispilimo_laikas = 0.0;
    }
    public void ExternalEvent(agMessage ms){
        switch(ms.GetMessageId()){
            case Traukinys.ATVYKO:
                q1.Push(new Vagonai(agSystem.GetTSyst()));
                q1.Push(new Vagonai(agSystem.GetTSyst()));
                break;
            case Valdymas.MANEVRUOTI:
                manevravimas.Start();
                break;
        }
    }
    public void InternalEvent(agInternalEvent ev){
        double tt,ss,vv;
        Event_Ispilimas e;
        switch(ev.GetOperationId()){
            case MANEVRAVIMAS:
                Vagonai v = (Vagonai) q1.Pop();
                tt = agSystem.GetTSyst();
                tl = tt-v.atv_laikas;
                laukimo_laikas.Note(tl);
                vv = IspilimoGreitis(Eksperimentas.LAUKO_TEMPERATURA,tl);
                ss = Traukinys.GRVAGONU_TALPA/vv;
                sum_ispilimo_laikas += ss;
```

```
                e = (Event_Ispilimas)ispilimas.Start(ss);
                e.SetTV(tt,vv);
                ispilimas_pradetas.SetV(vv);
                y1.Output(ispilimas_pradetas);
                break;
            case ISPILIMAS:
                e = (Event_Ispilimas) ev;
                ispilimo_laikas.Note(agSystem.GetTSyst()-e.GetT());
                ispilimas_baigtas.SetV(e.GetV());
                y1.Output(ispilimas_baigtas);
                break;
        }
    }

    private double IspilimoGreitis(int temperature, double time){
        double v;
        v = 250 - time + 2.5 * temperature;
        if(v > 200) v = 200;
        if(v < 100) v = 100;
        return v;
    }
    public void Report(){
        try{

agSystem.theOutput.writeNode("Sumpilimolaikas",String.valueOf((float)sum_ispi
limo_laikas));

agSystem.theOutput.writeNode("Pilimovalislaidos",String.valueOf((float)Eksper
imentas.VALPERPILIMO_ISLAIDOS));
            agSystem.theOutput.writeNode("Islaidos",

String.valueOf((float)(sum_ispilimo_laikas*Eksperimentas.VALPERPILIMO_ISLAIDO
S+Eksperimentas.PASTOVIOSISLAIDOS)));
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
    class Vagonai {
        public double atv_laikas;
        public Vagonai(double t){
            atv_laikas = t;
        }
    }

    public void Debug(){
        try{

agSystem.theDebugOutput.writeNode("Eile",String.valueOf(q1.Count()));

agSystem.theDebugOutput.writeNode("Laukimolaikas",String.valueOf(tl));
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

9.1.5 Valdymas programos kodas

```
package Terminal;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;

public class Valdymas extends agModule{
    public agItrPoint y1,y2;
    final public static int MANEVRUOTI = 31;
    final public static int PAKROVIMAS_PRADETAS = 32;

    private int vagonusk;
    private int garvezys;
    private int estakada;
    private boolean laukimas;
    private agStatP naftos_kiekis;
    private double naftos_apskaita;
    private agMessage manevruoti;
    private agMessage pakrovimas_pradetas;

    /** Creates new Valdymas */
    public Valdymas(agSystem sm) {
        super(sm,"Valdymas");
        y1 = new agItrPoint(this);
        y2 = new agItrPoint(this);
        manevruoti = new agMessage(MANEVRUOTI);
        pakrovimas_pradetas = new agMessage(PAKROVIMAS_PRADETAS);
        naftos_kiekis=new agStatP(this,"Naftos kiekis vagonuose");
    }
    public void Initialize(){
        naftos_apskaita=0.0;
        naftos_kiekis.Note(naftos_apskaita);
        laukimas = false;
    }
    public void ExternalEvent(agMessage ms){
        agInternalExEvent ex;
        switch(ms.GetMessageId()){
            case Traukinys.ATVYKO:
                naftos_apskaita += Traukinys.SASTATO_TALPA;
                naftos_kiekis.Note(naftos_apskaita);
                vagonusk +=2;
                if((garvezys==0)&&(estakada<=1)) {
                    y1.Output(manevruoti);
                    garvezys = 1;
                    estakada++;
                }
                break;
            case Stotis.ISPILIMAS_PRADETAS:
                naftos_apskaita -= (Traukinys.GRVAGONU_TALPA);
                naftos_kiekis.Note(naftos_apskaita);
                vagonusk--;
                if((vagonusk!=0)&&(estakada<=1)){
                    y1.Output(manevruoti);
                    estakada++;
                }else
                    garvezys = 0;
                break;
            case Stotis.ISPILIMAS_BAIGTAS:
                if((vagonusk!=0)&&(garvezys==0)){
                    y1.Output(manevruoti);
                    garvezys = 1;
                }else
```

```
                    estakada--;
                ex = agSystem.cntEvList.GetExEvent(0);
                if(laukimas && (ex.GetValue()>Baze.TALPA/2)){
                    y2.Output(pakrovimas_pradetas);
                    laukimas = false;
                }
                break;
            case Tanklaivis.ATPLAUKE:
                ex = agSystem.cntEvList.GetExEvent(0);
                if(ex.GetValue()>Baze.TALPA/2)
                    y2.Output(pakrovimas_pradetas);
                else
                    laukimas = true;
                break;
            case Tanklaivis.PAKROVIMAS_BAIGTAS:
                break;
        }
    }
    public void Debug(){
        try{
//
agSystem.theDebugOutput.writeNode("Nafta",String.valueOf(naftos_apskaita));

agSystem.theDebugOutput.writeNode("Vgonusk",String.valueOf(vagonusk));

agSystem.theDebugOutput.writeNode("Garvezys",String.valueOf(garvezys));

agSystem.theDebugOutput.writeNode("Estakada",String.valueOf(estakada));
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

9.1.6 Baze programos kodas

```
package Terminal;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;
import java.io.*;

public class Baze extends agModule{
//    final public static int PILNA = 31;
    final public static double TALPA = 90000.0;
    final private static int BAKAS = 1;

//    public agItrPoint y1;
    private agInternalExEvent ev1;
//    private agMessage p1;
    private ktu.vik.util.agOutputANSI theOutput;
    private agStatXP naftos_kiekis;
    private double perpiltos_naftos_kiekis;

    public Baze(agSystem sm) {
        super(sm,"Baze");
//        y1 = new agItrPoint(this);
        ev1 = new agInternalExEvent(this,0.0,TALPA,BAKAS);
//        p1 = new agMessage(PILNA);
    }
    public void Initialize(){
        ev1.Initialize(TALPA/2,0.0);
```

```
        naftos_kiekis=new agStatXP(this,"Naftos kiekis bake");
        naftos_kiekis.Note(ev1.GetValue());
        perpiltos_naftos_kiekis = 0.0;
    }
    public void ExternalEvent(agMessage ms){
        Message_Ispilimas ims;
        naftos_kiekis.Note(ev1.GetValue());
        switch(ms.GetMessageId()){
            case Stotis.ISPILIMAS_PRADETAS:
                ims = (Message_Ispilimas)ms;
                ev1.Increment(ims.GetV());
                perpiltos_naftos_kiekis += Traukinys.GRVAGONU_TALPA;
                break;
            case Stotis.ISPILIMAS_BAIGTAS:
                ims = (Message_Ispilimas)ms;
                ev1.Decrement(ims.GetV());
                break;
            case Valdymas.PAKROVIMAS_PRADETAS:
                ev1.Decrement(Tanklaivis.PAKROVIMO_GREITIS);
                break;
            case Tanklaivis.PAKROVIMAS_BAIGTAS:
                ev1.Increment(Tanklaivis.PAKROVIMO_GREITIS);
                break;
        }
    }
    public void InternalExEvent(agInternalExEvent ae){
    }

    public void Report(){
        try{

agSystem.theOutput.writeNode("Perpiltanaftos",String.valueOf((float)perpiltos
_naftos_kiekis));

agSystem.theOutput.writeNode("Perpilkaina",String.valueOf((float)Eksperimenta
s.PERPILIMO_KAINA));

agSystem.theOutput.writeNode("Pajamos",String.valueOf((float)(perpiltos_nafto
s_kiekis*Eksperimentas.PERPILIMO_KAINA)));
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
/*
    public void Debug(){
        try{

agSystem.theDebugOutput.writeNode("Bakas",String.valueOf(ev1.GetValue()));
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
*/
}
```

9.1.7 Tanklaivis programos kodas

```
package Terminal;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;

public class Tanklaivis extends agModule{

    final public static double TALPA = 25000.0;
    final public static double PAKROVIMO_LAIKAS = 16.0;
    final public static double PAKROVIMO_GREITIS = TALPA/PAKROVIMO_LAIKAS;

    public agItrPoint y1,y2;
    final public static int ATPLAUKE = 41;
    final public static int PAKROVIMAS_BAIGTAS = 42;

    final private static int TRANSPORTAVIMAS = 1;
    final private static int PAKROVIMAS = 2;
    private agOpFix transportavimas;
    private agOpFix pakrovimas;
    private agMessage atplauke;
    private agMessage pakrovimas_baigtas;

    public Tanklaivis(agSystem sm) {
        super(sm,"Laivas");
        y1 = new agItrPoint(this);
        y2 = new agItrPoint(this);
        transportavimas = new agOpFix(this,TRANSPORTAVIMAS,24.0);
        pakrovimas = new agOpFix(this,PAKROVIMAS,PAKROVIMO_LAIKAS);
        atplauke = new agMessage(ATPLAUKE);
        pakrovimas_baigtas = new agMessage(PAKROVIMAS_BAIGTAS);
    }
    public void Initialize(){
        transportavimas.Start();
    }
    public void ExternalEvent(agMessage ms){
        switch(ms.GetMessageId()){
            case Valdymas.PAKROVIMAS_PRADETAS:
                pakrovimas.Start();
                break;
        }
    }
    public void InternalEvent(agInternalEvent ev){
        switch(ev.GetOperationId()){
            case TRANSPORTAVIMAS:
                y1.Output(atplauke);
                break;
            case PAKROVIMAS:
                y2.Output(pakrovimas_baigtas);
                transportavimas.Start();
                break;
        }
    }
}
```

9.1.8 Ispilimas programos kodas

```
package Terminal;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;

public class Event_Ispilimas extends agInternalEvent{
    private double v;
    private double t;
    /** Creates new Event_Ispilimas */
    public Event_Ispilimas(double as, double aw, agOperation aop) {
        super(as,aw,aop);
    }
    public void SetTV(double at,double av){
        t = at;
        v = av;
    }
    public double GetV(){
        return v;
    }
    public double GetT(){
        return t;
    }
}


package Terminal;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;

public class Factory_Ispilimas extends agIntEvFactory{

    public Factory_Ispilimas(agOpFix aop){
        super(aop);
        aop.SetFactory(this);
    }
    public agInternalEvent CreateInternalEvent(double as, double aw){
        return  (agInternalEvent) new Event_Ispilimas(as,aw,op);
    }
}
```

Masage_ispilimas

```
package Terminal;
import ktu.vik.jpranas.agMessage;

public class Message_Ispilimas extends agMessage{
    private double v;

    public Message_Ispilimas(int id){
        super(id);
    }
    public void SetV(double av) {
        v = av;
    }
    public double GetV() {
        return v;
    }
}
```

9.2 Imitacinio modeliavimo rezultatų XSL transformacijos aprašymas

```
<?xml version="1.0" ?>
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output method="html" />
  - <xsl:template match="/">
    - <HTML>
      - <HEAD>
          <TITLE>Simulatiom experiment results</TITLE>
        </HEAD>
      - <body topmargin="15" leftmargin="10" bgcolor="#CCFFCC">
        - <font face="Verdana,Arial,Helvetica,sans-serif">
            <xsl:apply-templates select="report" />
          </font>
        </body>
      </HTML>
    </xsl:template>
  - <xsl:template match="report">
      <xsl:apply-templates select="experiment" />
    - <table align="center" width="95%" cellspacing="1" cellpadding="4"
        bgcolor="Silver">
      - <TR>
        - <TH WIDTH="20%" bgcolor="#DBDBBD">
            <font size="-1" COLOR="maroon" />
          </TH>
        - <TH WIDTH="16%" bgcolor="#DBDBBD">
            <font size="-1" COLOR="maroon">Number of
              Observation</font>
          </TH>
        - <TH WIDTH="16%" bgcolor="#DBDBBD">
            <font size="-1" COLOR="maroon">Min Value</font>
          </TH>
        - <TH WIDTH="16%" bgcolor="#DBDBBD">
            <font size="-1" COLOR="maroon">Max Value</font>
          </TH>
        - <TH WIDTH="16%" bgcolor="#DBDBBD">
            <font size="-1" COLOR="maroon">Mean Value</font>
          </TH>
        - <TH WIDTH="16%" bgcolor="#DBDBBD">
            <font size="-1" COLOR="maroon">Standart
              Deviation</font>
          </TH>
        </TR>
        <xsl:apply-templates select="module" />
      </table>
      <p />
      <p />
    - <table align="center" width="95%" cellspacing="1" cellpadding="4"
        bgcolor="Silver">
      - <TR>
        - <TD WIDTH="20%" bgcolor="#DBDBBD">
            <font size="-1" COLOR="maroon">Sumarinis pilimo
              laikas:</font>
          </TD>
```

```
- <TD WIDTH="13%" bgcolor="#FFFFCC">
  - <font size="-1">
      <xsl:value-of select="Sumpilimolaikas" />
    </font>
  </TD>
- <TD WIDTH="20%" bgcolor="#DBDBBD">
    <font size="-1" COLOR="maroon">pilimo val.
      islaidos:</font>
  </TD>
- <TD WIDTH="13%" bgcolor="#FFFFCC">
  - <font size="-1">
      <xsl:value-of select="Pilimovalislaidos" />
    </font>
  </TD>
- <TH WIDTH="20%" bgcolor="#DBDBBD">
    <font size="-1" COLOR="maroon">VISO
      ISLAIDU:</font>
  </TH>
- <TD WIDTH="13%" bgcolor="#FFFFCC">
  - <font size="-1">
    - <b>
        <xsl:value-of select="Islaidos" />
      </b>
    </font>
  </TD>
</TR>
- <TR>
  - <TD WIDTH="20%" bgcolor="#DBDBBD">
      <font size="-1" COLOR="maroon">Perpilta
        naftos:</font>
    </TD>
  - <TD WIDTH="13%" bgcolor="#FFFFCC">
    - <font size="-1">
        <xsl:value-of select="Perpiltanaftos" />
      </font>
    </TD>
  - <TD WIDTH="20%" bgcolor="#DBDBBD">
      <font size="-1" COLOR="maroon">tonos perpilimo
        kaina:</font>
    </TD>
  - <TD WIDTH="13%" bgcolor="#FFFFCC">
    - <font size="-1">
        <xsl:value-of select="Perpilkaina" />
      </font>
    </TD>
  - <TH WIDTH="20%" bgcolor="#DBDBBD">
      <font size="-1" COLOR="maroon">VISO
        PAJAMU:</font>
    </TH>
  - <TD WIDTH="13%" bgcolor="#FFFFCC">
    - <font size="-1">
      - <b>
          <xsl:value-of select="Pajamos" />
        </b>
      </font>
    </TD>
```

```
        </TR>
      </table>
    </xsl:template>
- <xsl:template match="experiment">
  - <p align="center">
      Simulation experiment:
    - <b>
        <xsl:value-of select="@title" />
      </b>
    </p>
  </xsl:template>
- <xsl:template match="module">
  - <TH colspan="6" bgcolor="#DBDBBD" align="center">
    - <font size="-1">
        <xsl:value-of select="@name" />
      </font>
    </TH>
    <xsl:apply-templates select="stat" />
  </xsl:template>
- <xsl:template match="stat">
  - <TR>
    - <TD bgcolor="#FFFFCC" align="center">
      - <font size="-1">
          <xsl:value-of select="@name" />
        </font>
      </TD>
    - <TD bgcolor="#FFFFCC" align="center">
      - <font size="-1">
          <xsl:value-of select="observation" />
        </font>
      </TD>
    - <TD bgcolor="#FFFFCC" align="center">
      - <font size="-1">
          <xsl:value-of select="min" />
        </font>
      </TD>
    - <TD bgcolor="#FFFFCC" align="center">
      - <font size="-1">
          <xsl:value-of select="max" />
        </font>
      </TD>
    - <TD bgcolor="#FFFFCC" align="center">
      - <font size="-1">
          <xsl:value-of select="mean" />
        </font>
      </TD>
    - <TD bgcolor="#FFFFCC" align="center">
      - <font size="-1">
          <xsl:value-of select="deviation" />
        </font>
      </TD>
    </TR>
  </xsl:template>
</xsl:stylesheet>
```