KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Aivaras Bakanas

AGREGATINIŲ SPECIFIKACIJŲ INTERAKTYVUSIS REDAGAVIMAS IR IMITACINIS MODELIAVIMAS

Magistro darbas

Vadovas
prof. habil. dr. H.Pranevičius

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
dr. Eduardas Bareiša
2005-05-...

AGREGATINIŲ SPECIFIKACIJŲ INTERAKTYVUS REDAGAVIMAS IR IMITACINIS MODELIAVIMAS

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių k. katedros lek.
dr. J. Mikelionienė
2005-05-16

Recenzentas
.....
2005-05-.....

Vadovas
prof. habil. dr. H.Pranevičius
2005-05-23

Atliko
IFM 9/2 gr. stud.
A.Bakanas
2005-05-23

KAUNAS, 2005

SANTRAUKA (anglų kalba)

INTERACTIVE EDIT OF PLA SPECIFICATIONS AND SIMULATION MODELING

PLA method can be used to formally specify systems. Systems specifications are written as text, what is exhaustive and understandable, but unfortunately are not visual, and user can't quickly acquaintance with the formalized system.

This article presents a way to represent aggregates specifications visually, which eases an acquaintance with specification. This article also presents a software tool to visually design formal specifications.

TURINYS

Lentelių sąrašas

Paveikslėlių sąrašas

ĮVADAS

Kuriant sudėtingas realiojo laiko paskirstytąsias sistemas galima aibė skirtingų realizacijos pasirinkimų. Kadangi šie pasirinkimai daromi pradinėse kūrimo fazėse, todėl jos vaidina lemiamą vaidmenį sistemos finaliniame darbo našume. Siekiant išvystyti, padidinti darbo našumą, priklausantį nuo pasirinktų kūrimo alternatyvų, prieš realizuojant sistemą, galima panaudoti priemones panaudojančias sistemos formalią specifikaciją ir leidžiančias išanalizuoti sistemos našumo parametrus dar prieš pradedant sistemos realizaciją. Tokiomis priemonėmis yra „pereinamos“ pirmosios sistemos kūrimo proceso fazės ir suformuojamas sistemos abstraktus kuriamos sistemos modelis, kuris ir yra skirtas tos sistemos savybėms analizuoti. Tokie įrankiai turi dirbti su kokia nors modeliavimo kalba ir turi turėti darbo našumo analizės įrankius.

Viena iš tokių modeliavimo kalbų yra UML [16]. Ši kalba turi aibę grafinių notacijų, leidžiančių specifikuoti sistemos funkcionalumą. Be to, UML dažnai naudojama sistemos koncepcijai vystyti ar stimuliuoti ir dokumentacijai apipavidalinti. Tačiau iš principo UML nėra vykdomoji modeliavimo kalba, todėl tai kliudo atlikti sistemos (ypač realaus laiko didelių industrinių) darbo našumo parametrų analizę. Bet vis dėl to yra UML įrankių, kurie leidžia papildyti programas vykdomuoju kodu, pvz. C++. Ir dar šie papildomi įrankiai suteikia galimybę įsivesti laiką ir įvykius (sutapimus), nes iš esmės UML semantika to neturi.

Darbe naudojamas formalizavimo ir analizės metodas, kuris remiasi atkarpomis tiesinių agregatų (PLA) formalizavimu [13]. Šis metodas priklauso automatų modelių klasei, tačiau išskirtinis tuo, kad sistemos būsenai aprašyti naudojamos ir diskrečios ir tolydžios koordinatės. Šis formalizavimo metodas yra išraiškingas, su griežtomis formaliomis semantikomis, kurios leidžia atlikti realaus laiko paskirstytų sistemų analizę. Čia nagrinėjamos sistemos objektai aprašomi kaip atskiri agregatai. Čia įtraukta ir laiko sąvoka, todėl galima atlikti sistemų imitavimą tam, kad būtų išgautas sistemos darbo našumo parametras realiai dirbant (imituojant sistemos darbą) sistemai.

1.1. Tikslai ir uždaviniai

Rašant agregatinę specifikaciją, specifikuotojui palikta didelė erdvė savoms interpretacijoms. Todėl tokiu būdu paliekamos spragos klaidoms ir netikslumams. Šio darbo tikslas yra pateikti agregatinės specifikacijos interaktyvaus redagavimo posistemės (redaktoriaus) sukūrimo principus, metodus, kuriais remiantis redaktorius susistemina medžiagą, interaktyviai bendrauja su vartotoju ir nepalieka jam laisvės savaip interpretuoti agregatinės specifikacijos.

Be to, norint įsitikinti specifikuojamos (projektuojamos) sistemos darbo našumu reikalingi metodai, leidžiantys ištirti suspecifikuotos sistemos skaitines charakteristikas. Tam panaudojami imitacinio modeliavimo metodai, kurie adaptuoti objektiniam požiūriui ir objektiniam modeliavimui. Darbe taip pat pateikiamas sukurtos programinės įrangos eksperimentinis išbandymas su konkrečiu uždaviniu.

1.2. Dokumento paskirtis

Šis dokumentas pateikia problemas, kurios egzistuoja sudarinėjant agregatines specifikacijas ir atliekant jų analizę. Pateikiami egzistuojantys metodai toms problemoms spręsti. Dokumente pateikiami šių metodų realizacijos realioje sistemoje architektūros principai.

2. ANALITINĖ DALIS

2.1. Įžanga

Programinės įrangos kūrimo stadija susideda iš standartinių kūrimo procesų: reikalavimų surinkimas, projektavimas, kūrimas, testavimas, diegimas ir palaikymas [6][11][12]. Reikalavimų surinkimo etape, iš vartotojo gaunama aibė neformaliai aprašytų reikalavimų, norų ir pageidavimų, kurie vienaip ar kitaip apibrėžia norimos projektuoti sistemos funkcijas. Šie reikalavimai paprastai užrašomi natūralia kalba. Didžiausias natūralios kalbos privalumas yra tai, kad ne techninės pakraipos vartotojai gali suprasti sistemos reikalavimus [4]. Didžiausias trūkumas yra tai, kad natūralios kalbos neapibrėžtumai padidina tikimybę atsirasti klaidoms. Tačiau, kad šiuos reikalavimus vienareikšmiškai suprastų ir reikalavimų surinkėjai, ir projektuotojai, ir programuotojai, juos reikia užrašyti griežta formalia kalba.

Formalūs metodai ir formalių specifikacijų kalbos yra vienas iš būdų aprašyti reikalavimams išlaikant matematinį tikslumą ir griežtumą. Jų pagrindinis akcentas – sistemos modelio sukūrimas aprašant sistemos struktūrą (struktūriniai reikalavimai) ir jos elgesį (elgesio reikalavimai).

Vienas iš sprendimų yra natūralią kalbą transformuoti į formalią notaciją mechaniškai buvo pasiūlytas Fleiko (Flake) [5]. Tokio metodo trūkumas yra tai, kad ta natūrali kalba, kuri gali būti naudojama aprašyti reikalavimams yra labai ribota.

Būtent šis transliavimas iš natūralios kalbos į formalią iškėlė sinchronizacijos uždavinį tarp šių dviejų formų. Todėl kitas autorius Hunle [7] pristatė įrankį, leidžiantį lygiagrečiai manipuliuoti reikalavimais šiose skirtingose notacijose. Vartotojo reikalavimai saugomi abstrakčiame modelyje ir vartotojas gali matyti per grafinį interfeisą ir natūralioje ir UML kalboje. Čia atsiradusi problema – natūralios kalbos tekstas buvo kuriamas pagal vidinio abstraktaus medžio struktūrą, kuri gali paveikti nenuspėjamos struktūros sakinius.

Alternatyva natūraliai kalbai yra grafinės notacijos. Tai yra plačiai paplitusi ir priimta technika sistemų kūrime. Viena iš tokių notacijų yra UML. UML yra grafinės ir tekstinės notacijų karkasas, leidžiantis specifikuoti struktūrą ir elgesį. Bet kaip ir visi kiti formalūs metodai, taip ir šis reikalauja įvertinimo, kai reikia skaityti ar rašyti reikalavimus.

Panašią situaciją aprašo ir Miller [10] kai skrydžių valdymo sistemos reikalavimai buvo aprašyti naudojant grafinę notaciją – taip vadinamą RSML. Čia reikalavimai buvo iš karto mechaniškai verčiami į laikinas logines savybes, kur buvo iš karto tikrinami. Bet ir čia reikėjo žmogaus su patirtimi, kuris galėtų rašyti ir prižiūrėti reikalavimus.

Šiame darbe aprašytos sistemos darbo principai pagrįsti agregatine specifikacija. Šis formalusis metodas leidžia specifikuojamą sistemą natūraliai suvokti ir aprašinėti kaip objektus (agregatus), kurie turi tarpusavio ryšius ir daro vienas kitam poveikį. Šios formalios notacijos pasirinkimą nulėmė ir jau anksčiau pradėti darbai skaitmeniniame modeliavime. Tačiau kaip ir visos formalios notacijos, taip ir ši reikalauja šiokių tokių vartotojo žinių, norint aprašyti sistemą. Šiai problemai spręsti šiame darbe aprašyta kaip į šį darbą galima įtraukti ir interaktyviąją redagavimo sistemą, kuri prisideda prie pagrindinės problemos sprendimo.

2.2. Agregatinė specifikacija

2.2.1. Formalių specifikacijų metodai

Formalių specifikacijų metodai skiriasi pagal tai, kokioje paradigmoje specifikacija iš tikrųjų yra [9]:

Istorija grįsta specifikacija. Čia sistema specifikuojama charakterizuojant maksimalią aibę priimtinų istorijų (elgesių). Savybės yra specifikuojamos pagal laikinus loginius sprendinius apie sistemos objektus. Tokie sprendiniai priverčia operatorių kreiptis į praeities, esamas ir ateities būsenas. Sprendiniai yra interpretuojami laiko rėmų struktūrose.

Būsenomis pagrįsta specifikacija. Čia vietoje charakterizuojant priimtinas istorijas yra charakterizuojamos priimtinos sistemos būsenos. Savybės yra specifikuojamos invariantais iš momentinių sistemos objektų būsenų ir iš momentinių prieš tai einančių ir po to einančių sistemos operacijų sprendinių. Prieš tai einantys sprendiniai paima silpniausią reikalingą sąlygą įėjimo būsenoje tai operacijai atlikti, po to einantys sprendiniai – stipriausią sąlygą išėjimo būsenoje jei ta operacija buvo atlikta. Šia paradigma remiasi tokios kalbos kaip Z, VDM ar B. Orientuoti į objektus šio metodo pataisymai taip pat buvo pasiūlyti.

Perėjimais pagrįsta specifikacija. Čia yra specifikuojami sistemos perėjimai iš būsenos į būseną. Savybės yra specifikuojamos kaip aibė perėjimo funkcijų būsenų automate. Perėjimo funkcija kiekvienam sistemos objektui duoda įėjimo būseną ir trigerių įvykį bei atsakančią išėjimo būseną. Trigerio įvykio atsiradimas yra pakankama sąlyga perėjimui prasidėti.

Funkcinė specifikacija. Pagrindinis principas yra specifikuoti sistemą kaip matematinių funkcijų kolekciją. Funkcijos gali būti grupuojamos pagal jų objektų tipus arba pagal abstrakčius duomenų tipus. Kitu atveju, funkcijos gali būti grupuojamos į logines teorijas. Tokios teorijos turi tokią informaciją kaip tipų apbrėžimus, kintamųjų apibrėžimus, aksiomų apibrėžimus.

Kokį specifikavimo būdą pasirinkti priklauso nuo aibės kriterijų, kurie realiai yra visapusiškai priklausomi ir netgi konfliktuojantys. Realiai pasirinkimas daugiau priklauso nuo specifikuotojų prioritetų specifikuojamai konkrečiai sistemai. Šiame darbe yra pasirinkta būsenomis pagrįsta specifikacija. Pasirinkimas grindžiamas tuo, kad ši specifikacija suteikia turtingą struktūrą reikalingą aprašyti sudėtingiems objektams. Toks specifikavimo būdas labiau tinka paskirstytosioms sistemos ar transakcinėms sistemoms.

2.2.2. Formalios specifikacijos apibrėžimas, savybės

Apskritai, formali specifikacija – tai sistemos savybių rinkinio išraiška kažkokioje formalioje kalboje, kažkokiame abstrakcijos lygyje [9]. Tikslesnis apibrėžimas galimas tik tada, kai tiksliai žinome: kas slepiasi po žodžiu „sistema“, kokios savybės yra įdomios, koks abstrakcijos lygis bus naudojamas ir galiausiai, kokia formali kalba bus naudojama.

Žodis „formalus“ dažnai yra maišomas su žodžiu „tikslus“. Aišku, kad pastarasis paveldi pirmąjį, bet tik ne atvirkščiai. Specifikacija yra formali, jei kalba, kuria ji parašyta yra sudaryta laikantis trijų taisyklių:

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

Norint parašyti teisingą specifikaciją, reikia įvertinti šiuos aspektus:

- Specifikacija turi būti adekvati – ji turi labai tiksliai nusakyti problemą.
- Specifikacija turi būti nuosekli – jei paimsime visas specifikuotas savybes į visumą, ta turi būti teisinga.
- Specifikacija turi būti nedviprasmiška – negali turėti dviprasmybių suvokiant kurį nors teiginį kaip tiesą.
- Specifikacija turi būti pilna – žemesnio lygio savybių rinkinys, turi būti pakankamas, kad būtų galima apibūdinti aukštesnio lygio teiginį.
- Specifikacija turi būti minimali – negali turėti perteklinės informacijos, ar savybių kurios nesusijusios su problema.

2.2.3. Tekstinis ir grafinis specifikacijų sudarymo būdai

Sudarant sistemos specifikacijas, jas reikia aprašyti formaliai. Vienas iš aprašymo būdų yra aprašymas formaliomis išraiškomis tekstu. Tačiau toks sistemų specifikavimo būdas nėra patogus, nes

nėra vaizdus, reikalauja didesnio įsigilinimo norint kažką suprasti. Daug laiko sugaištama bandant nustatyti sistemos dalis ir jų tarpusavio ryšius bei santykius [1][2].

Siekiant palengvinti formalios agregatinės specifikacijos sudarymą ir jos peržiūrėjimą, galima naudoti grafinį specifikacijos vaizdavimo, užrašymo būdą. Šiuo atveju agregatinės specifikacijos vaizduojamos diagramomis, galima netgi sukurti CASE priemones, palengvinančias tokių diagramų redagavimą.

2.1 lentelė. Grafinio specifikacijų sudarymo privalumai ir trūkumai

Privalumai	Trūkumai
Vaizdžiai matoma specifikacija	Diagramose negalima atvaizduoti visos informacijos, nes tada jos būtų perkrautos (pvz., nesimato įėjimo ir išėjimo funkcijų).
Lengvai skaitoma	Diagramoje nesimato pradinių būsenų (jas reikia aprašyti atskirai).
Lengvai sudaroma	
Realizuoti programiniai įrankiai gali nuimti pasikartojantį, bereikalingą darbą	

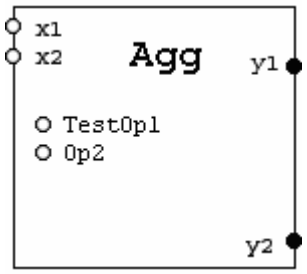
2.2.4. Agregatinės specifikacijos grafinis vaizdavimas

Agregatinės specifikacijos vaizduojamos grafinėmis diagramomis, kurios palengvina jų sudarymą. Agregatinės specifikacijos diagramose vaizduojami tik dviejų tipų elementai: agregatai ir ryšiai tarp jų.

Agregatas vaizduojamas:

- Stačiakampiu – agregatą diagramoje atitinka stačiakampis su jame pažymėta kita informacija. Stačiakampis gali būtį įvairaus dydžio.
- Agregato vardu stačiakampyje – stačiakampyje vaizduojančiame agregatą, viršuje, viduryje užrašomas agregato vardas.
- Vidiniai įvykai atvaizduojami apskritimu ir operacijos pavadinimu – kiekvienas agregate vykstantis vidinis įvykis atvaizduojamas apskritimu ir šalia jo užrašomas jo pavadinimas. Vienas įvykis užrašomas žemyn į apačia po kito įvykio. Užrašai išlygiuojami pagal dešinį agregato kraštą. Vidiniai įvykiai vaizduojami agregato stačiakampio viduje, pradedami žymėti iš kart po pavadinimo.
- Išėjimai žymimi tamsiu apskritimu – agregato išėjimai pažymimi užtamsintu apskritimu ant agregato krašto, šalia užrašomas išėjimo signalas. Užrašas daromas agregato stačiakampio vidinėje dalyje.

- Įėjimai žymimi tuščiaviduriu apskritimu – agregato įėjimai pažymimi tuščiaviduriu apskritimu ant agregato krašto, šalia užrašomas įėjimo signalas. Užrašas daromas agregato stačiakampio vidinėje dalyje.



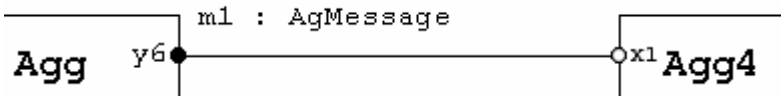
2.1 pav. Agregato pavyzdys

Paveikslėlyje 2.1 pavaizduotame agregate matome, kad jo pavadinimas yra „Agg“, įėjimo signalų aibė yra X={x1, x2}, išėjimo signalų aibė Y={y1,y2} ir vidiniai įvykiai ya „TestOp1“ ir „Op2“.

Tačiau grafinėje agregato vaizdavime neatsispindi visos jo savybės. Diagramoje nematome pavaizduotos išorinių įvykių aibės, bet juos galima numatyti pagal įeinančius signalus. Taip pat nematome valdančiųjų sekų, tolydinės ir diskrečios agregato būsenos, pradinių būsenų. Diagramoje taip pat neatsispindi perėjimo ir išėjimo operatoriai. Juos galima atvaizduoti atskirose diagramose vaizduojant UML „activity“ diagramas. Trūkstama informacija pateikiama priede prie specifikacijos. Diskrečias ir tolydžias agregato būsenos dedamąsias galima vaizduoti po vidiniu operacijų, išorinius įvykius po vidinių įvykiu kokiu nors kitokiu žymėjimu, pavyzdžiui stačiakampiais. Tačiau tai apkrautų per daug diagramą. Programinėje įrangoje būtų galima realizuoti paslėpimą arba dalinį rodymą laukų.

Kanalai tarp agregatų vaizduojami grafiškai diagramoje sujungiant agregatus diagramoje. Tai yra daug patogiau negu naudojant lenteles, kuriose pažymėta koks agregatas su kuriuo agregatu sujungtas. Kanalai tarp agregatų vaizduojami:

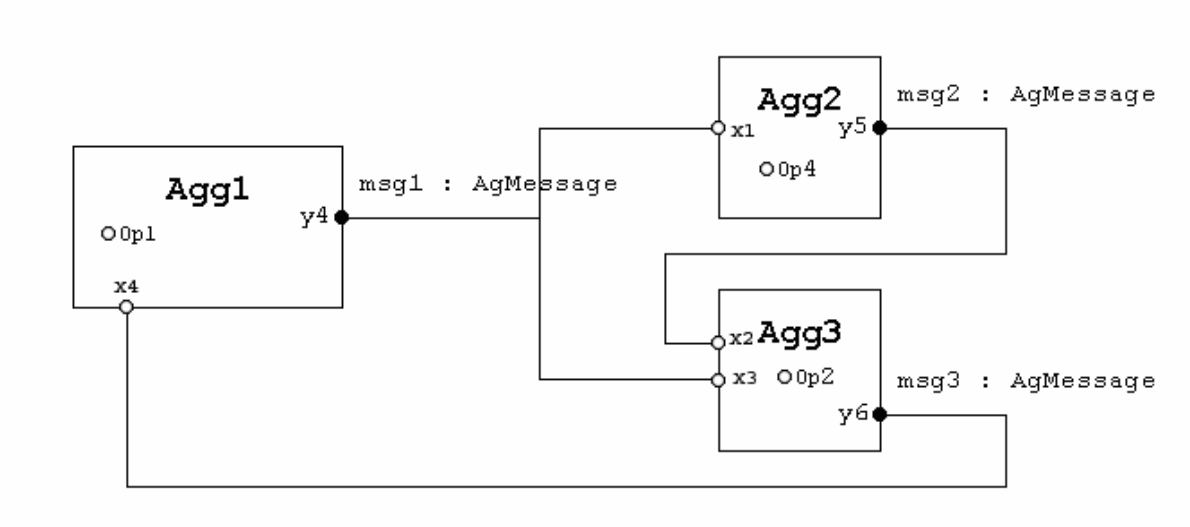
- Sujungiamai agregato išėjimai su kitų agregatų įėjimais - jungiami vieno agregato išėjimai su kito agregato įėjimais linija (esant poreikiui sudarytai iš atkarpų). Linijos jungia agregato tuščiavidurius apskritimus su užpildytais. Linijos neina per agregatą vaizduojančio stačiakampio vidų. Kanalai taip pat gali išsišakoti ir tas pats kanalas išeidamas iš vieno agregato gali būti prijunktas prie kelių agregatų įėjimų.
- Ant kanalo vaizduojami agregato siunčiami pranešimai - ant kanalo, jungiančio agregatus, rašomi juo perduodami pranešimai. Nurodoma pranešimo pavadinimas ir tipas.



2.2 pav. Kanalas tarp dviejų agregatų

Paveikslėlyje 2.2 pateiktame agregatų sujungime vaizduojami sujungtu kanalu du agregatai „Agg“ ir „Agg4“, agregatas „Agg“ išduoda išėjimo signalą „y6“, agregatas „Agg4“ prima įėjimo signalą „x1“ ir kanalu jungiančiu agregatus perduodamas pranešimas „m1“.

Sujungus agregatus ir kanalus tarp jų gaunama agregatų diagrama. Diagramos pavyzdys pateiktas 2.3 paveikslėlyje.



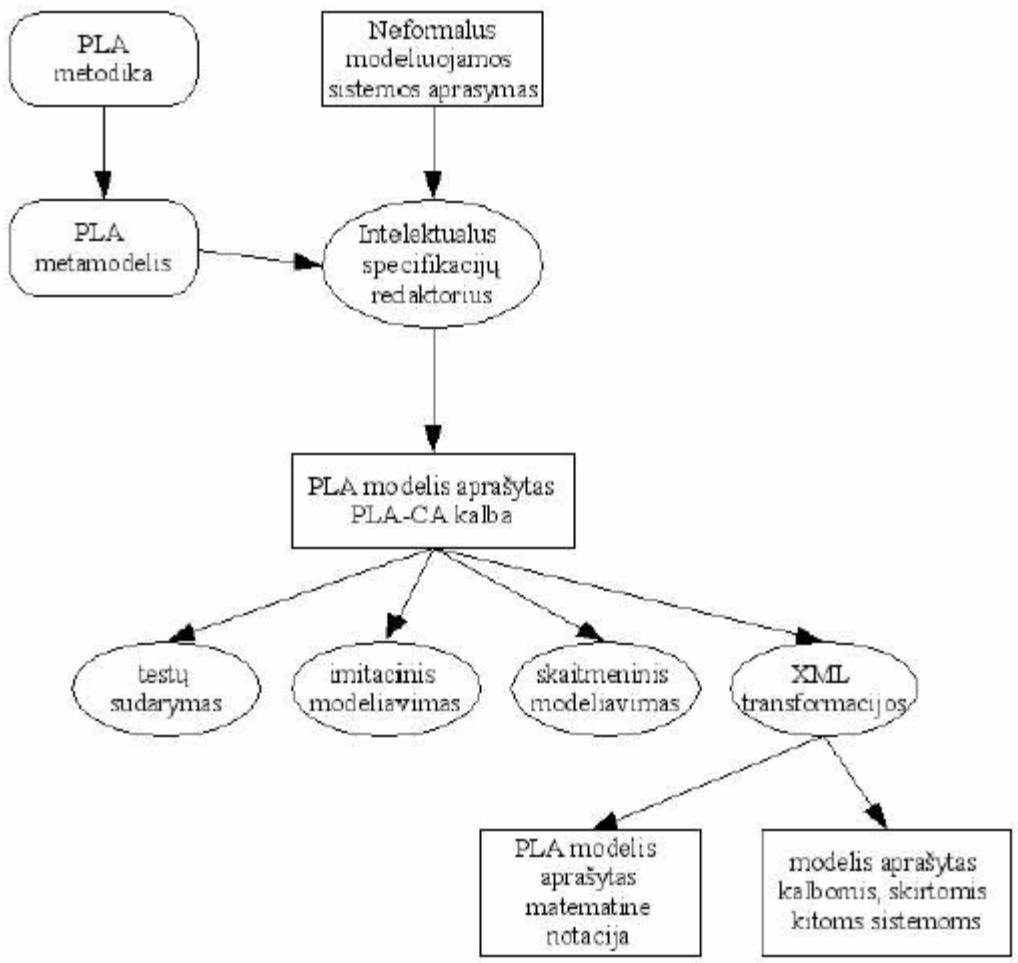
2.3 pav. Agregatų diagrama

Paveikslėlyje 2.3 pavaizduota agregatų diagrama. Joje yra aprašyti 3 agregatai „Agg1“, „Agg2“, „Agg3“, jų įėjimo ir išėjimo signalai, kanalai jungiantys agregatus, bei jais perduodami pranešimai. Taip pat pavaizduotas kanalas jungiantis iškarto tris agregatus.

2.3. PLA modelis, PLA-CA metakalba

Kadangi kuriant dideles sistemas iškyla įvairūs šių sistemų validavimo klausimai, juos galima išspręsti pasitelkus sistemų modeliavimą. Modeliavimo metu yra atliekama labai daug skaičiavimų, todėl tikslinga modeliavimo metodikas automatizuoti.

Konkretus modeliavimo automatizavimas pateikiamas 2.4 paveikslėlyje:



2.4 pav. Automatizuoto PLA modelio sudarymo schema

Norint įgyvendinti automatizavimo procesą svarbu išspręsti modeliuojamos sistemos PLA aprašo pateikimo kompiuteriui užduotį. Tam, kad kompiuteris priimtų, mokėtų apdoroti pateiktą aprašą, jis turi būti aprašytas griežtai struktūrizuota kalba. Be to, ši kalba turi būti pakankama bet kokio PLA modelio aprašymui, kuris po to galėtų būti naudojamas tolimesniems PLA metodikos taikymams, tokiems kaip testų sudarymas, skaitmeninis modeliavimas, imitacinis modeliavimas. Todėl PLA-CA (CA – Computer Adopted) kalbos sukūrimui buvo sudarytas PLA kalbos metamodelis, kuris buvo gautas remiantis PLA metodika.

Naudojant PLA metamodelį buvo sukurta PLA-CA kalba, kurios sintaksiniu pagrindu bvo pasirinkta XML (eXtended Markup Language) kalba. Be minėtų PLA metodikos taikymų, galimas dar vienas labai svarbus taikymas – PLA-CA kalboje aprašytas modelis gali būti transformuojamas į aibę kitų kalbų, pasinaudojant XML transformacijomis [18].

PLA-CA kalba aprašytą modelį gali perskaityti ir žmogus, nes jis turi aiškias struktūras su pradžia ir pabaiga, tačiau tai nėra patogu. Svarbus pateiktos automatizavimo metodikos privalumų yra tai, kad sistemos modeliuotojas, naudodamas interaktyvų PLA sistemos redaktorių, gali kurti PLA modelius

turėdamas tik neformalų PLA sistemos aprašymą ir jam nebūtina žinoti PLA-CA kalbos sintaksės, o pakanka vien tik suprasti PLA metamodelį.

2.4. Agregatinio specifikavimo metodo metamodelis

Pilnas agregatinės specifikacijos aprašymas susideda iš 9 punktų [1][2]:

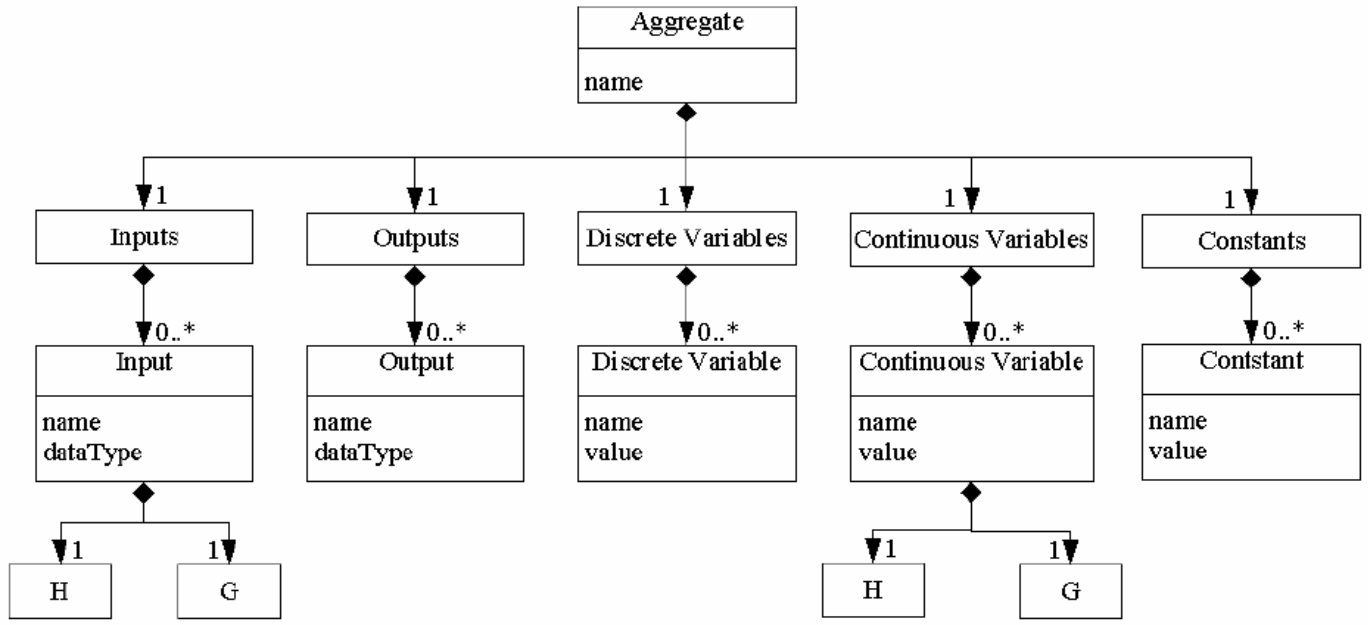
1. Įėjimų aibės apibrėžimas – čia kiekvienam agregato įėjimui yra priskiriamas konkreti perduodamo signalo duomenų struktūra.
2. Išėjimų aibės apibrėžimas – analogiškai pirmajam punktui.
3. Išorinių įvykių aibė – įvykiai, įvykę agregato išorėje, bet veikiantys agregato būseną.
4. Vidinių įvykių aibė – įvykiai, įvykę agregato viduje ir keičiantys agregato būseną.
5. Valdančios sekos
6. Diskretinės būsenos vektoriaus komponentės – aibė agregato elementų būsenų reikšmių, galinčių įgyti diskretines reikšmes.
7. Tolydinės būsenos vektoriaus komponentės – aibė agregato elementų, galinčių įgyti tolydines reikšmes.
8. Pradinė būsena – diskretinių ir tolydinių būsenos vektoriaus komponenčių pradinės reikšmės
9. Būsenos pakeitimo (H) ir signalų perdavimo (G) operatoriai – algoritmai (sakinių sekos), pagal kuriuos keičiama agregato būsena ir išduodami signalai į agregato išorę.

Pagal šiuos 9 punktus galima pilnai aprašyti kiekvieną sistemoje esantį agregatą, bet prieš tai reikia būti aprašius galimas perduodamų signalų struktūras.

2.5. Agregato metamodelis

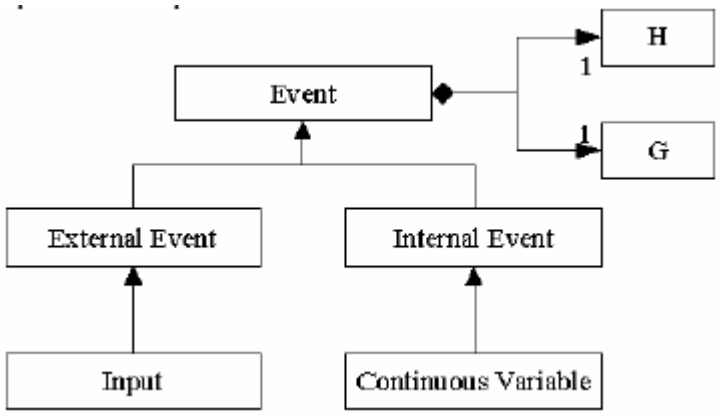
Agregatas susideda iš:

1) Įėjimų aibės, kurioje gali būti neribotas kiekis įėjimų. Įėjimą aprašo: pavadinimas ir tipas, kuris rodo, kokio tipo užklausos gali patekti į agregatą per įėjimą.
2) Išėjimo aibės, kurio gali būti neribotas kiekis išėjimų. Išėjimą aprašo: pavadinimas ir tipas, kuris rodo, kokio tipo užklausos yra siunčiamos iš agregato per išėjimą.
3) Diskrečiųjų kintamųjų aibės, kuriai gali priklausyti neribotas kiekis diskrečiųjų kintamųjų. Diskretūs kintamieji naudojami aprašyti agregato būsenos dedamąsias.
4) Tolydžiųjų kintamųjų aibės, kuriai gali priklausyti neribotas kiekis tolydžiųjų kintamųjų. Tolydūs kintamieji yra naudojami aprašyti laiko momentus kuriais įvyksta vidiniai agregato įvykiai pasibaigus operacijoms.



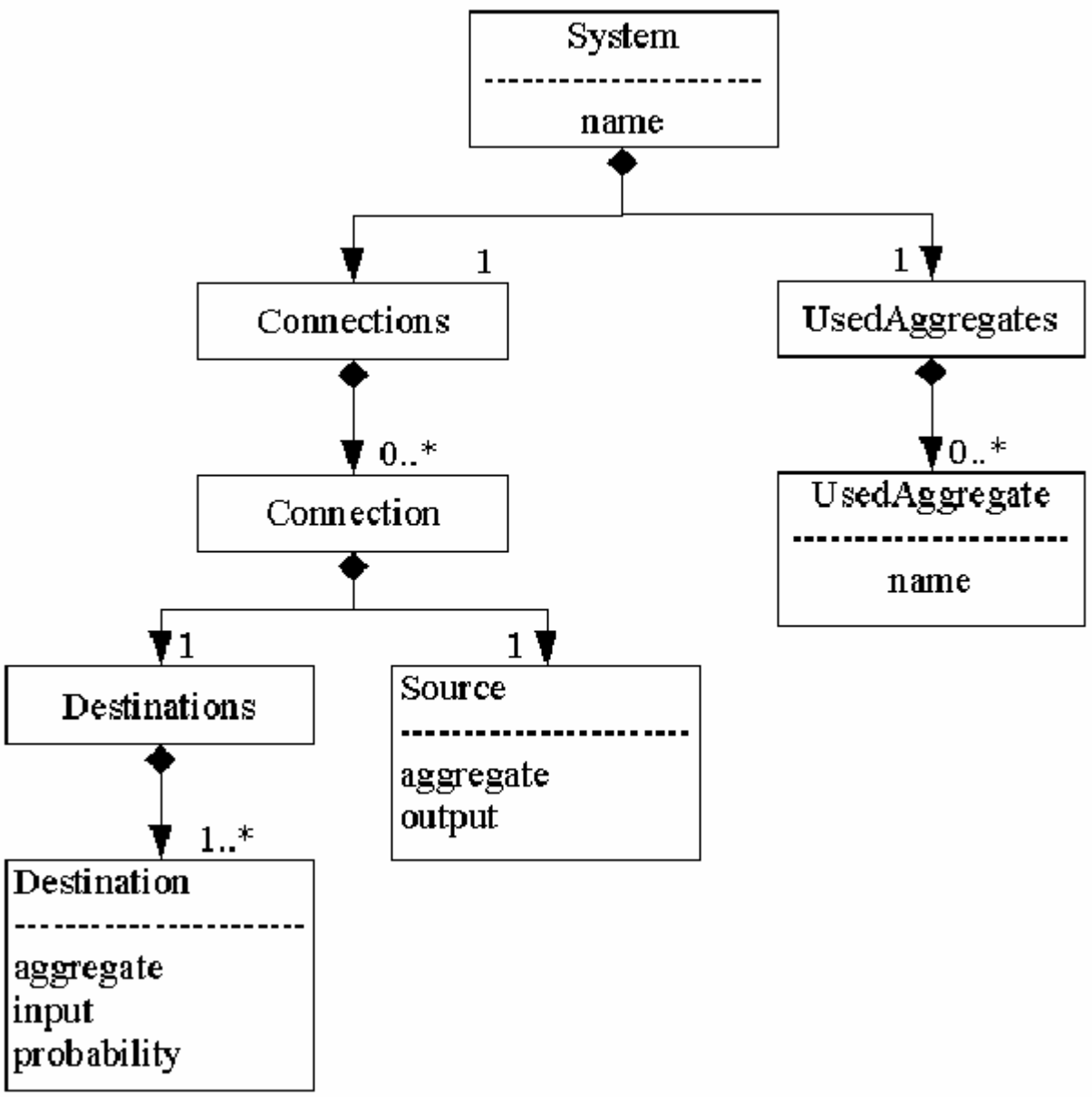
2.5 pav. Agregato metamodelis UML

Agregato įėjimų signalai iššaukia išorinius įvykius. Vidiniai ir išoriniai įvykiai gali keisti agregato vidinę būseną – šiuos pokyčius aprašo H operatorius; ir gali keisti signalus – šiuos pokyčius aprašo G operatorius.



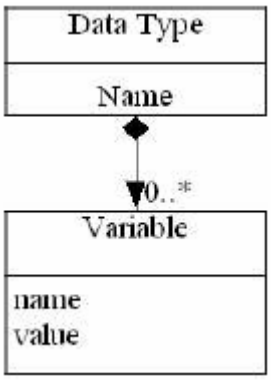
2.6 pav. H ir G operatoriai

Visa modeliuojama sistema yra sudaryta iš keleto tarpusavyje sujungtų agregatų. Sujungimai gali jungti vieno agregato išėjimą su keleto ar vieno agregato įėjimu. Tokios sistemos UML metamodelis pavaizduotas 2.7 paveikslėlyje.



2.7 pav. Agregatų sistemos metamodelis

Kai agregatai yra sujungti tarpusavyje, jie sąveikauja. Jų sąveikai aprašyti ir naudojami sujungimai. Sujungimais tarp agregatų gali būti perduodami duomenų signalai. Signalo duomenų struktūra pavaizduota 2.8 paveikslėlyje:



2.8 pav. Įėjimų ir išėjimų signalų duomenų struktūra

Sujungimai jungia vieno agregato išėjimą su kito agregato įėjimu. Gali būti sujungimas ir į kelių agregatų įėjimus, bet būtinai to pačio tipo [14].

3. PROJEKTINĖ DALIS

3.1. Sistemos paskirtis

Didelės apimties taikomosios programos, ypač tos kurios apima visą kompanijos darbo veiklą – turi būti kur kas daugiau nei programinio kodo modulių rinkinys. Moduliai turi sąveikauti tokiu būdu, kad visa informacinė sistema būtų saugi, greita, patikima dirbant su maksimaliu apkrovimu. Architektūra turi būti tiksliai apibrėžta, kad atradus sistemos klaidą, bet koks programuotojas sugebėtų ją greitai ištaisyti, net jei tikrieji sistemos autoriai klaidos taisyme nedalyvautų.

Sudėtingų sistemų projektavimo procesą sudaro užduočių seka – nuo sistemos prototipo kūrimo iki programinio kodo generavimo. Šis procesas apima reikalavimų sistemai surinkimą, sistemos teikiamų servisų apibrėžimą, veikimo prognozavimą ir programinių komponenčių architektūros aprašymą.

Projekto tikslas yra sukurti programinę įrangą, kuri leistų atlikti paskirstytų sistemų integruotą analizę, t.y. atlikti imitacinį modeliavimą bei formalios agregatinės specifikacijos validavimą prieš pradedant kurti realią sistemą. Be to bus sugeneruoti testai, kurie padės išanalizuoti busimą sistemą skirtingais aspektais (specifikacijos pilnumo, neprieštaringumo ir kt.).

3.2. Egzistuojantys sprendimai

3.2.1. SPIN

Specifikavimo, imitacinio modeliavimo ir validavimo sistema SPIN yra plačiai naudojama kompiuterių protokolams tirti. SPIN įtraukia stiprius formalius pagrindus, pastovus, kaip SDL, ir palaiko naudingą modelių tikrinimą. SPIN naudoja C-like specifikacijos notaciją kuri padidina jo pritaikomumą pirmoje kūrimo būsenoje ir daro sekančias kodo diegimo fazes negu kad mechanines kiekybinių detalių užduotis. SPIN leidžia simuliaciją ir validaciją PROMELA specifikacijos faile. Programinės įrangos įrankyje MSC yra pagrindinės funkcijos. Grafinis interfeisas FSM pradeda nuo procesų modelių ir analizės procesų aktyvumo.

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

Sistemai, specifikuotai PROMELA kalboje, SPIN gali atlikti imitacinį modeliavimą tos sistemos vykdymo, arba jis gali generuoti programą C, kuri vykdo pakankamai naudingą sistemos savybių teisingumo patikrinimą realiu laiku. Vykdant imitacinį modeliavimą ir verifikaciją, SPIN patikrina ar nėra mirties taškų, netikėtų gavimų, ir neįvykdomo kodo. Tas tikrintojas taipogi gali būti naudojamas tikrinti sistemos kintamųjų teisingumui, jis gali rasti nereikalingus ciklus, gali patikrinti teisingumą sekančio laiko momento esančių loginių formuluočių teisingumą. Tikrinimas turi būti naudingas ir naudoti minimalų atminties kiekį. Išsamus tikrinimas gali su matematine tikimybe nustatyti ar aprašytas sistemos elgesys yra be klaidų. Labai didelės tikrinimo problemos, kurios negali būti išspręstos su esama kompiuterine technika, gali būti bandomos spręsti su ekonomiška „būsenos saugojimo bitu“ technika, taip pat žinoma kaip „supertrace“. Šiuo metodu būsenos užimama vieta suskirstoma į mažą bitų skaičių pasiekiamoje sistemos būsenoje, su minimaliu „side“ efektu [15].

„SPIN“ yra modelių tikrinimo įrankis, kurtas tikrinti komunikacijos protokolams. Šio įrankio modeliavimo kalba – „Promela“ – leidžia dinamiškai kurti lygiagrečius procesus, sinchroninius bei asinchroninius pranešimų perdavimus bei bendro naudojimo (shared) kintamuosius. Įrankiai duomenų abstrakcijoms labai riboti – tik bitai, baitai, sveikieji skaičiai ir masyvai. Sprendiniai gali būti įsiūti į modelį, o savybės gali būti specifikuojamos pagal LTL (linear-time temporal logic) formules.Be to, „SPIN“ tikrina neteisingas pabaigos būsenas, neprogresuojančius ciklus remiantis (arba nesiremiant) neteisingumo prielaidomis.

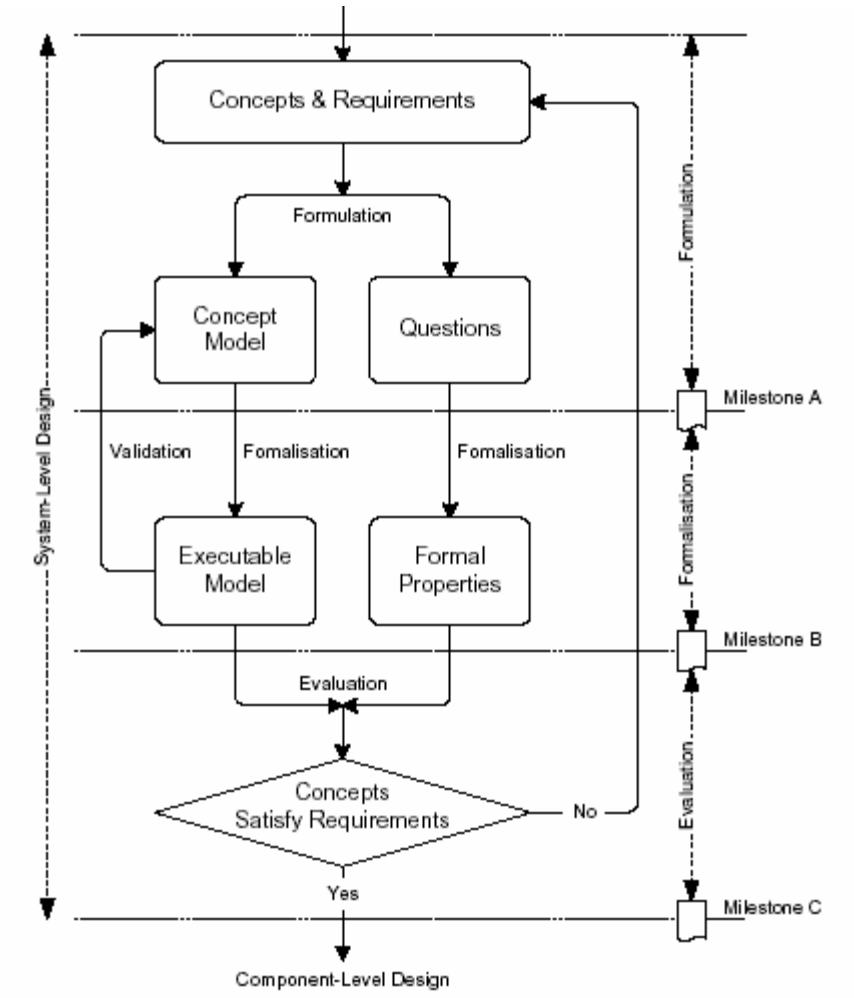
„SPIN“ taip pat turi atsitiktinę arba sąveikaujančią (interaktyvią) simuliacijas ir gali atlikti išsamų būsenų erdvės patikrinimą ir saugumo, ir gyvybingumo požiūriais. „Xspin“ grafinė sąsaja puikiai padidina „SPIN“ naudojamumą, nes tai suteikia standartinę meniu sąsają, ir tai leidžia gyvai (animuotai) vartotojui atlikti imitavimą, klaidų trasavimą keliais skirtingais būdais [3].

3.2.2. Praxis

Tai viena iš CASE priemonių sukurta Verslo Informatikos katedroje. Ji veikė tekstiniame režime ir schematiškai atvaizduodavo vieną agregatą. Deja vaizdavimas buvo gana skurdus, ekrane buvo matomas tik vienas agregatas vienu metu ir buvo terodomi jo įėjimai ir išėjimai. Nebuvo galimybės peržiūrėti visus sudarytos agregatinės specifikacijos kaip visumos.

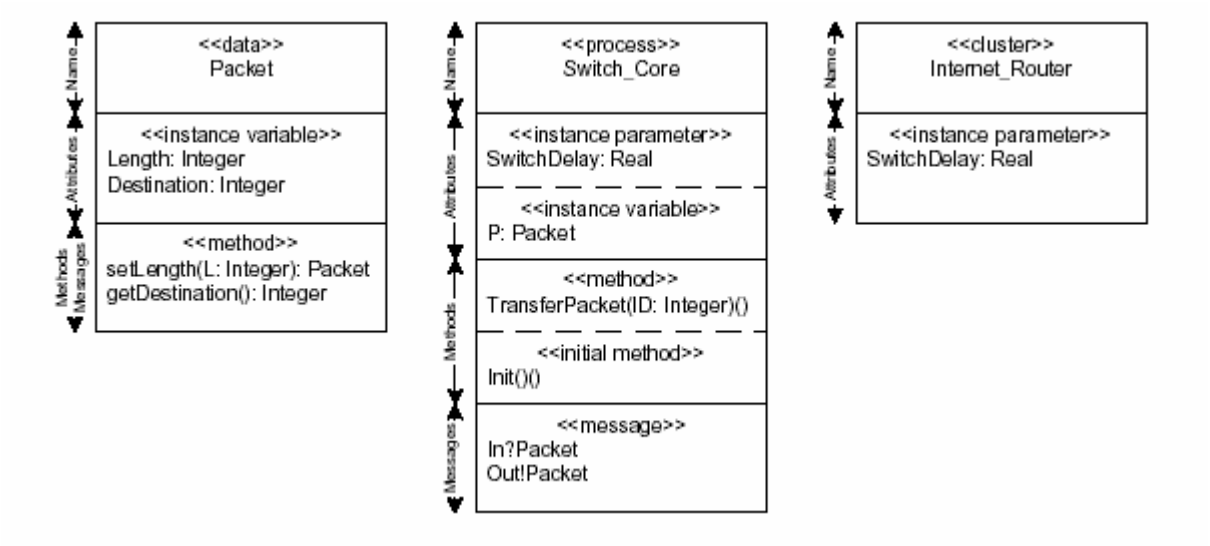
3.2.3. SHE

SHE – Software Hardware Engineering. SHE pagrįstas sistemos lygio srautui, kuris atvaizduotas 3.1 paveikslėlyje:



3.1 pav. Formulavimas, formalizavimas ir evoliucija sistemos lygio kūrimo

SHE įgyvendina euristiką vystant UML modelius, kurie gali būti transformuojami į vykdomuosius modelius specifikuotus POOSL (Parallel Object-Oriented Specification Language) kalboje. POOSL yra išraiškinga modeliavimo kalba su formaliomis semantikomis, skirtomis sudėtingų realaus laiko paskirstytų sistemų analizei. Pagrįsta formalia semantika, POOSL turi ir matematines notacijas, kurios leidžia atlikti matematinę analizę apie sistemos darbo našumą. SHE naudoja trijų stereotipų klases, kad aprašyti skirtingus resursų tipus:



3.2 pav. SHE naudojamos klasės

Duomenų objektai naudojami modeliuojant pasyvius resursus sistemoje. Procesų objektai naudojami aprašyti realaus laiko aktyvių resursų elgesį. Klusteriai yra skirti modeliuoti sudėtingus aktyvius resursus sistemose. Prosesai ir klusteriai yra statiškai sujungiami kanalų. Kanalai modeliuoja bet kokią galimybę apsikeisti informacija tarp komponentų [17].

3.2.4. Murϕ

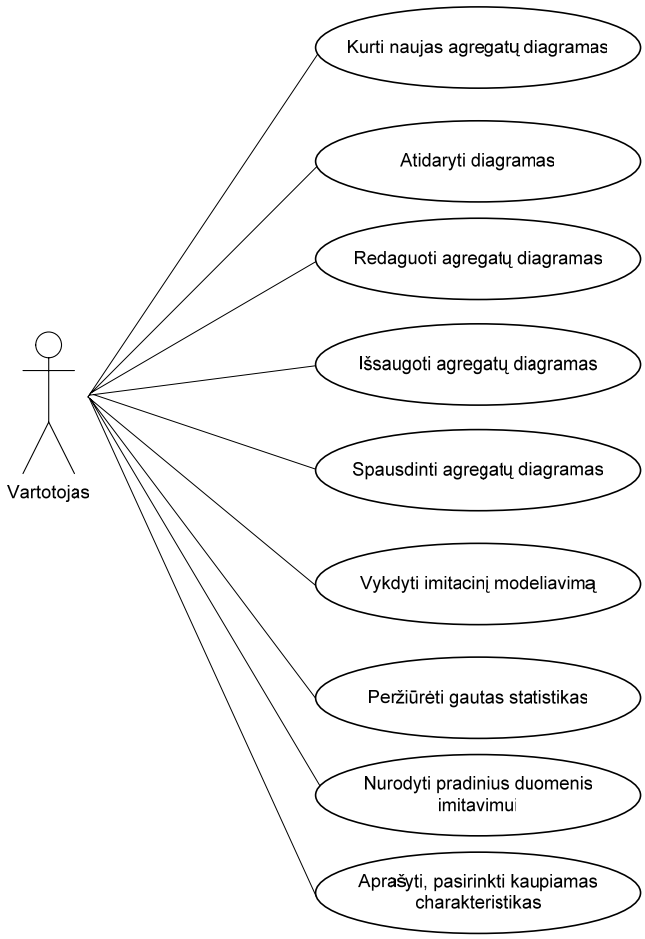
Šis įrankis taip pat iš pradžių buvo kurtas tik komunikacijos protokolams specifikuoti. Šio įrankis susideda iš aibės perėjimo taisyklių sąlygas ir veiksmus. Modelio vykdymas iškviečia pasikartojantį tų taisyklių, kurios yra veikiančios, iškvietimą ir toliau tą taisyklę sekančio veiksmo ar veiksmų sekos įvykdymas. Visi veiksmai dirba su bendro naudojimo kintamaisiais. Sintaksė pagrįsta „Pascal“ kalba. Todėl naudojamos visos „Pascal“ esančios duomenų struktūros ir tipai. Šis įrankis gali tikrinti tik saugumo savybę. Neteisingos pabaigos būsenos čia taip pat gali būti aptinkamos [3].

3.2.5. FDR2

Šis įrankis yra sukurtas konkrečiai darbui su CSP (Comunicating Sequences Processes). Šios įrankio modeliavimo kalba CSP forma, kuri lengvai skaitoma kompiuterio. Duomenų tipai galimi visi standartiniai. „FDR2“ skiriasi nuo panašių įrankių tuo, kad čia naudojamas metodas nustatyti ar sistema atitinka konkrečia savybę pagrįstas patobulinimų supratimu. Šis įrankis taip pat gali patikrinti ar sistema turi mirties taškų, bei amžinų ciklų. Deja, šis įrankis neteikia jokių simuliavimo galimybių. FDR2 yra „windows“ tipo programinė įranga, vartotojas gali pasirinkti procesą ir jį tikrinti. Jei procesas neatliekamas teisingai, įrankis turi „debugerį“, kuris leidžia vaikščioti po sistemą struktūriškai, tam kad surasti kurioje vietoje yra klaida [3].

3.3. Esminiai reikalavimai

Realizuotos programinės įrangos panaudojimo atvejai:

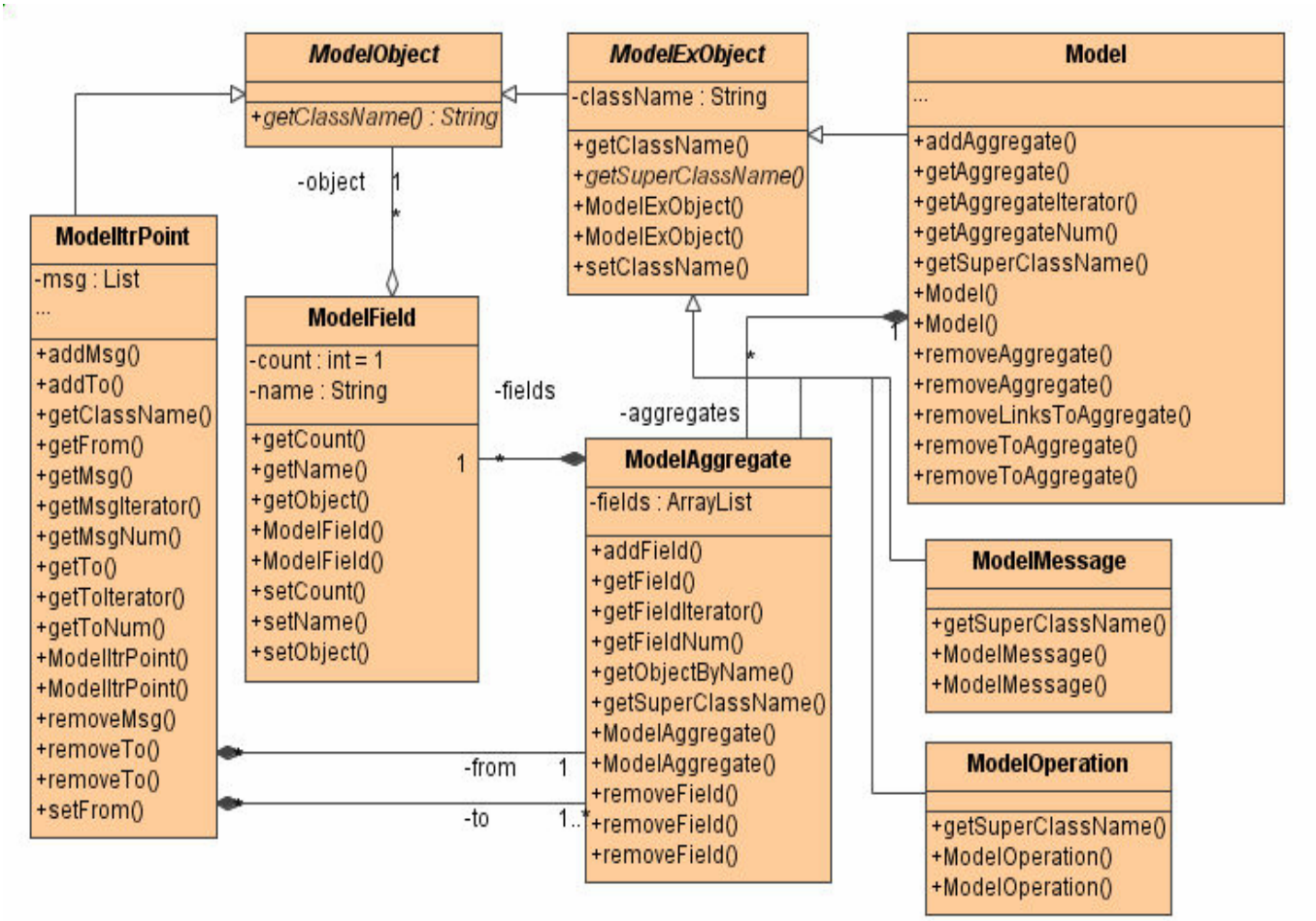


3.3 pav. Panaudojimo atvejai

Pagrindiniai funkciniai reikalavimai:

- Galimybė vartotojui susikurti naują tuščią agregatų diagramą
- Galimybė užkrauti anksčiau išsaugotą diagramą iš disko
- Galimybė naudotis įrankių juosta ir iš jos pelės pagalba sukurti naujus agregatus, sujungti juos sujungimais, ištrinti agregatus
- Peržiūrėti ir nustatyti, keisti agregato specifikaciją
- Galimybė išsaugoti diagramas diske
- Galimybė spausdinti diagramas
- Galimybė generuoti modeliuojamos sistemos kodą JAVA kalboje
- Galimybė užduoti pradinius duomenis imitaciniam modeliavimui
- Galimybė redaguoti renkamas statistikas
- Galimybė peržiūrėti imitavimo rezultatus.

Duomenų modelio schema:



3.4 pav. Agregato duomenų modelio schema

Agregatams saugoti yra skirtos tokios klasės:

- ModelObject - agregatų diagramos modelio bazinė abstrakti klasė apibrėžianti kiekvienos modelyje esančios klasės sąsają. Turi lauką „className“, skirta saugoti diagramos elemento pavadinimui (jei diagramos objektas agregatas tai pavadinimas atitiks agregato pavadinimą, o taip pat generuojamo kodo agregatą atitinkančios klasės vardą).
- ModelExObject - klasė išplečianti ModelObject klasę. Ji prideda naują lauką kaip „superClassName“. Laukas įvestas tuo tikslu, kad agregatų diagrama jau būtų šiek tiek susieta su kokiu programos kodu iš jos bus galima sugeneruoti. Tad šitas laukas nurodo kokią klasę turės paveldėti klasė sugeneruota iš šio diagramos objekto.
- Model - klasė agreguojanti visus agregatus ir atitinkanti visą modeliuojamą sistemą.

- ModelAggregate – klasė atitinkanti diagramoje vaizduojamą agregatą. Turi ModelField objektų sąrašą, kurie savo ruožtu susiję su agregato atliekamomis operacijomis, pranešimų perdavimo kanalais, siunčiamais/gaunamais pranešimais.
- ModelField - agregate saugomas laukas: operacija, sujungimo kanalas, pranešimas. Pasirinktas dėl to, kad iš agregato sugeneruotoje klasėje atitinka klasės kintamąjį.
- ModelItrPoint - klasė atitinkanti sujungimo kanalą tarp agregatų. Turi laukus: agregatą iš kurio kanalas išeina ir sąrašą agregatų į kuriuos kanalas ateina. Taip pat turi sąrašą juo keliaujančių pranešimų.
- ModelMessage - klasė atitinkanti perduodamą pranešimą tarp agregatų. Ji yra asocijuota su ModelItrPoint klase.
- ModelOperation - klasė atitinkanti agregato atliekamą operaciją.

Pagrindiniai nefunkciniai reikalavimai:

- Įprasta, intuityvi sąsaja ir meniu
- Greito vartojo įrankių „toolbar“
- Sistema turi neleisti vartotojui daryti klaidų
- Sistemos universalumas kalbos atžvilgiu
- Visapusiška pagalba vartotojui
- Sistema turi būti lengvai išplečiama papildomais komponentais, papildomu funkcionalumu

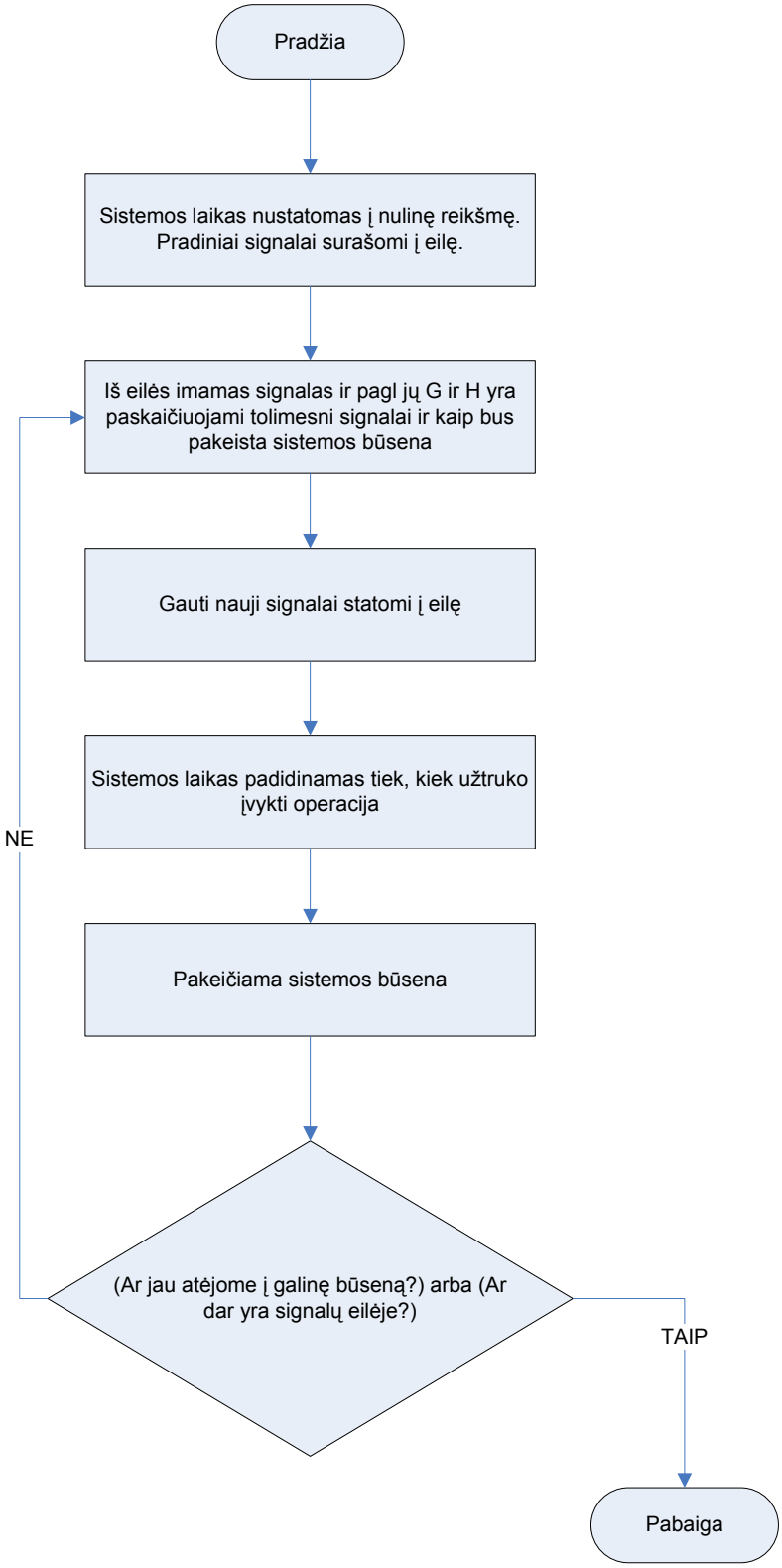
3.4. Imitacinio modeliavimo algoritmo aprašymas objektinio modeliavimo priemonėmis

Algoritmas grindžiamas tuo, kad iš turimos specifikacijos pagal tam tikras taisykles yra generuojamas sistemos programinis kodas ir toliau jis paleidžiamas vykdyti.

Kiekvienas agregatas yra traktuojamas kaip atskiras objektas, todėl jis atitinka atskirą klasę objektinėje kalboje, agregatų sistema taip pat sudaro atskirą klasę. Generuojamas kodas gali būti bet kurios objektinės programavimo kalbos sintakse. Mano darytame projekte yra generuojamas JAVA programavimo kalbos kodas.

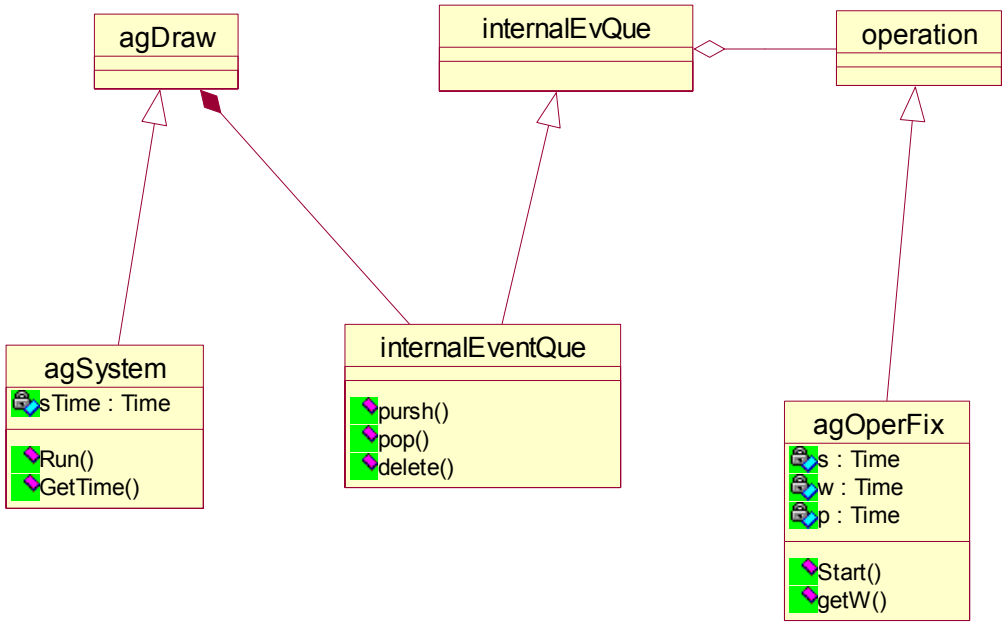
Kadangi agregatai yra traktuojami kaip objektai, o sujungimai atstoja ryšius tarp objektų, tai visa agregatinė specifikacija objektiniu požiūriu veikia kaip objektų visuma. Per sujungimus objektai keičiasi vienas su kitais pranešimais, arba jei reikia perduoda reikalingą informaciją.

Imitacinio modeliavimo algoritmas yra pateikiamas 3.5 paveikslėlyje:



3.5 pav. Imitacinio modeliavimo algoritmo schema

Imitacinio modeliavimo klasės ir jų galimi ryšiai pavaizduoti klasių diagramoje:



3.6 pav. Imitacinio modeliavimo posistemės klasių diagrama

„agSystem“ subklasė aprašo struktūrą imitavimo veiklai. Jos metodai „Run“ – paleidžia patį imitavimo procesą, o „getTime“ – gražina esamą sistemos laiką. Laukas „s“ skirtas sistemos laiko reikšmei saugoti.

Klasė „ahOperFix skirta operacijų imitavimo aprašymui, kintamasis „s“ – operacijos startavimo laikas, „w“ – momentas, kada įvyksta įvykis, „p“ – operacijos trukmė. „Start“ metodas iškviečiamas, kai reikia įvykdyti operaciją.

Vykdant operaciją, yra paleidžiamas metodas „Start“, tada kai tik gaunama esamo laiko reikšmė, „w“ reikšmė yra suskaičiuojama sudedant „s“ ir „p“ reikšmes. Tada įvykis yra pastatomas į įvykių eilę.

Vykdant imitacinį modeliavimą, visą laiką yra stebimos ir skaičiuojamos statistikos. Tos statistikos vartotojui, analizuojančiam specifikuotą sistemą, pateikia labai įvairia informaciją, kurios pagalba analitikas gali susidaryti įspūdį ir nuomonė apie tai kaip veiks realiai ši sistema.

Galimos rinkti statistikos:

- „AgStat“ – atsitiktinio diskretinio dydžio statistika, skirta įvykiams, kurie atsitinka kai kuriais tai laiko momentais.
- „AgStatD“ – atsitiktinio proceso statistika, skirta atsitiktinėms proceso būsenoms. Būsenos keičiasi atsitiktiniais laiko momentais, o visą kitą laiką išlieka pastovios.

- „AgStatXp“ – tolygiai kintanti atsitiktinio proceso statistika. Skirta procesams, kurių būsena tolydžiai kinta tarp atsitiktinių laiko momentų.

Panaudojus šias statistikų klases, jau galimos analizuoti konkrečios statistikos, kaip pvz.:

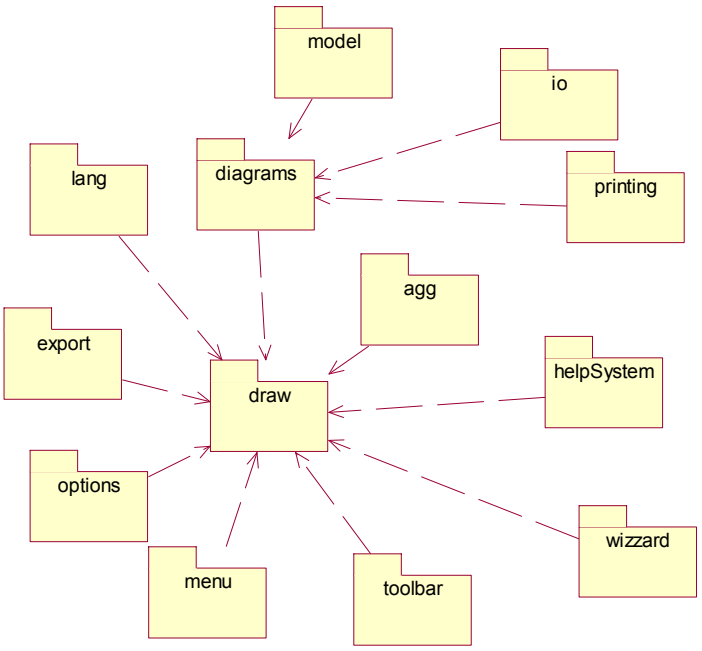
- Vidutinė eilės ilgio reikšmė - kuri rodo, kokio ilgio eilė buvo vidutiniškai per tam tikrą laiko tarpą, kai įvykiai įvyksta visokiais laiko momentais ir nėra spėjami apdoroti, todėl statomi į eilę ir po to imami iš jos apdorojimui.
- Vidutinis laiko tarpas tarp įvykių

3.5. Agregatinės specifikacijos interaktyvaus redaktoriaus struktūros sudarymas

Interaktyviojo redaktoriaus pagrindinis tikslas – sudaryti sąlygas specifikuotojui paprasčiau, greičiau, nuimant pasikartojantį darbą ir be sintaksinių klaidų surinkti sistemos specifikaciją. Detali programinės įrangos architektūra sudaryta iš aibės UMl diagramų, kurios aprašo:

- Panaudojimo atvejų vaizdą.
- Loginį vaizdą.
- Procesų vaizdą.
- Išdėstymo vaizdą.
- Realizacijos vaizdą.

Šiame darbe realizuotos programinės įrangą sudarančius paketus atvaizduoja 3.7 paveikslėlis.



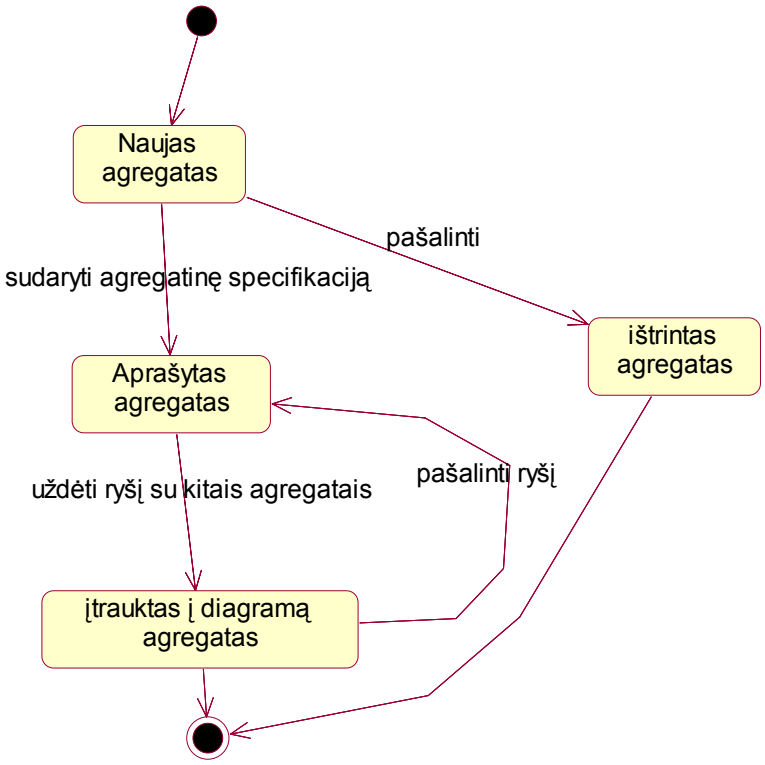
3.7 pav. Interaktyvųjį redaktorių sudarantys paketai

Paketų paskirtis trumpai aprašyta lentelėje:

3.1 lentelė. Interaktyvųjį redaktorių sudarančių paketų aprašymas

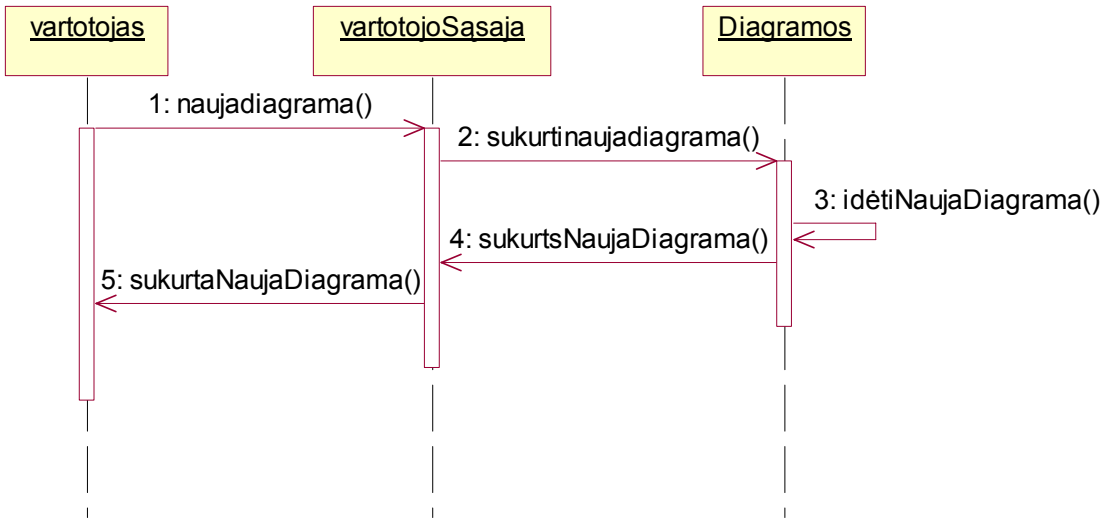
Paketo vardas	Paskirtis
Draw	Paketas skirtas realizuoti visos programos karkasą, t.y. pagrindiniai langai, meniu, įrankių juostos ir pan.
Lang	Paketas skirtas aprašyti kalbos pasirinkimo ar keitimo galimybėms programoje. Programinė įranga suprojektuota taip, kad sąsajos kalbą pakeisti yra labai paprasta. Visa tai aprašyta šiame pakete.
Model	Čia saugomos visos klasės, kurios aprašo agregato modelį.
Io	Aprašo struktūras agregatinių diagramų nuskaitymui iš failo arba išsaugojimui faile diske.
Diagrams	Paketas, kuris apdoroja visų diagramų braižymą interaktyviojo redaktoriaus darbo lange. Čia aprašomos visos agregatines specifikacijas sudarančios struktūros ,kurios skirtos agregatus atvaizduoti ekrane grafiškai.
Printing	Paketas, kuriame realizuota diagramų spausdinimo ant popieriaus funkcija
Agg	Paketas, kuriame saugoma visos agregatinės specifikacijos modelis.
HelpSystem	Pagalbos sistemos realizacija
Menu	Paketas vykdantis meniu užkrovimą iš XML failų ir organizavimą juos vaizduojant ekrane.
Toolbar	Paketas, realizuojantis įrankių juostų aprašymus ir atvaizdavimus programinėje įrangoje.
Wizzard	Paketas, kuriame aprašytos klasės, skirtos vedlio sukūrimui.

Kad geriau įsivaizduoti programinės įrangos funkcionavimą, architektūros specifikacijoje yra pateikiama aibė programinės įrangos objektų būsenų diagramų. Vienos iš tokių diagramų pavyzdys pateiktas 3.8 paveikslėlyje.



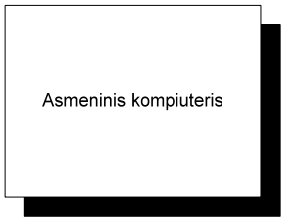
3.8 pav. Agregato būsenų diagrama

Sekančiame etape yra atvaizduotos sekų diagramos. Vienos iš tokių diagramų pavyzdys pateiktas 3.9 paveikslėlyje.



3.9 pav. Sekų diagrama – naujos diagramos sukūrimas

Kadangi programinė įranga nėra paskirstyta per kelis kompiuterius ar tinklą, ji veikia tik viename kompiuteryje. Programinės įrangos išdėstymo diagrama pavaizduota 3.10 paveikslėlyje.



3.10 pav. PĮ išdėstymo diagrama

Vartotojo programinė įranga veikia nepriklausomai nuo platformos asmeniniame kompiuteryje. Programinė įranga veiks 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.

Procesorius: >900 Mhz

Operacinė sistema: nepriklausomai

Atmintis: >= 128 MB

Ekranas: >800*600 dpi, 16-bitų spalvos

3.6. Realizacijos ypatumai

Daugelis kuriamos sistemos realizacijos klausimų buvo galima sakyti apspręsti iš anksto. Kadangi projektas buvo pradėtas vystyti ne nuo pačių pradžių, todėl reikėjo prisitaikyti prie jau esamų realizuotų posistemių. Visų pirma prisitaikymas turi būti prie duomenų formato, nes posistemės keičiasis bendrais duomenimis (šiuo atveju agregatine specifikacija), todėl tas formatas turi būti suderintas tarpusavyje.

Agregatinės specifikacijos duomenys yra saugomi XML faile. Failo struktūra griežtai apibrėžta, todėl tai leidžia tuos pačius duomenis panaudoti ir kitose posistemėse (skaitmeninio modeliavimo, testų generavimo, validavimo). XML failo struktūra atrodo taip:

```
<?xml version=”1.0“ encoding=“UTF-8“?>
<agregate>
      <name>Stotis</name>
      <discretevariables>
            <discrete>
            <name>n</name>
            <value>0.2</value>
            </discrete>
            ...
      </discretevariables>
      <continousvariable>
            <continous>
            <name>e_1</name>
            <value>3.0</value>
            <H>...</H>
            <G>...</G>
            </continous>
```

```
            ...
      </continousvariable>
      <outputs>
            <output>y1</output>
            ...
      </outputs>
      <inputs>
            <input>
            <name>x1</name>
            <G>...</G>
            <H>...</H>
            </input>
            ...
      </inputs>
</agregate>
```

Antras jau apspręstas realizacijos klausimas buvo programavimo kalbos pasirinkimas. Tai lėmė 2 veiksniai: jau buvo parašyta skaitmeninio modeliavimo posistemė JAVA kalboje ir kita vertus, daugelis programinių komponentų buvo pakartotinai panaudota iš ankstesnių projektų ir tie komponentai buvo parašyti JAVA kalboje.

3.7. Vartotojo sąsaja

Vartotojo sąsaja turi būti kuriama taip, kad ji būtų įprasta vartotojui ir panaši į vartotojo naudojamų programų vartotojo sąsajas. Kita vertus ji turi būti sukurta taip, kad neleistų, ar apsaugotų vartotoją nuo galimų klaidų įvėlimo. Viena iš kūrėjų ir projektuotojų problemų ir yra tai, kad jie pamiršta ką žino vartotojas. Pagrindiniai Vartotojo sąsajos kūrimo principai [8]:

- Pažinti programinės įrangos vartotoją – kuo daugiau žinosime apie savo vartotoją, tuo tiksliau galėsime nuspėti jo galimus veiksmus, kurie gali įtakoti sistemos veikimą.
- Kuo daugiau nuspėjamų ir intuityvių sąsajų – tai leidžia vartotojams atlikti intuityvius veiksmus net neatkreipiant dėmesio, nes jie atliekami automatiškai. Pvz., jei įvedimo lauke reikia įvesti skaičių, tai būtinai reikia padaryti, kad naudojant mygtukus į viršų ir žemyn būtų galima tą skaičių didinti ar mažinti.
- Neleisti vartotojui atlikti veiksmų, kurių jie negali atlikti – jei kažkoks veiksmas seka tik po tam tikro veiksmo ar rezultato, tai taip ir turi būti. Pvz., negalima vartotojui leisti pereiti prie tolimesnių specifikacijos pildymo laukų, jei prieš esantys būtini laukai yra neužpildyti.
- Negalima atimti iš vartotojo kokių nors veiksmų ir atlikti juos automatiškai, jei vartotojas nebus informuotas tiksliai kas bus atlikta, jei nėra įsitikinta, kad tas veiksmas pilnai atitiks tai, ką norėjo padaryti vartotojas.
- Sudaryti išimčių sąrašą, kuris neleistų vartotojui atlikti nereikalingų ar negalimų veiksmų, kaip pvz., jei esame sąrašo paskutiniame įraše, tai mygtukas „žemyn“ turi būti neaktyvus.

- Operacijų atlikimo efektyvumas ir greitis. Kai kurias pasikartojančias operacijas būtina realizuoti taip, kad vartotojui nereikėtų kartoti tų pačių veiksmų, norint jas įgyvendinti. Tarkim jei vartotojas nupiešia agregatą, tai sistema turėtų automatiškai atidaryti agregato specifikacijos pildymo langą, kad vartotojui nereiktų to daryti pačiam, nes tai veiksmas, kurį tikrai darys vartotojas po agregato nupiešimo diagramoje.
- „Desing for erorr“ – kuriant vartotojo interfeisą reikia galvoti, kad vartotojas padarys visas galimas klaidas ir kad bandys atlikti visas negalimas operacijų kombinacijas. Tokiu būdu mes išvengsime daugiau galimų GUI klaidų.
- Atsakomoji reakcija – jei vartotojas kažką daro ar padarė, jis turi gauti atsakymą ar tai padaryta, ar tai vyksta ir matomas kažkoks progresas.
- Minimizuoti vartotojui reikalingų atsiminti dalykų sąrašą – jei vartotojas atlikdamas kelis veiksmus vieną po kito turi atsiminti kokią nors informaciją iš prieš tai buvusių veiksmų, tai tikrai sudarys terpę klaidoms.
- Kiekviename žingsnyje vartotojas turi žinoti, kokios operacijos dabar galimos. Negalima projektuoti GUI taip, kad vartotojas mato tuščią langą ir nėra jokių pranešimų ką jis dabar gali daryti.
- Reikia kontroliuoti pateikiamos informacijos kiekį. Jei viename lange iš karto vartotojas gaus per didelį kiekį informacijos, jis paprasčiausiai pasimes joje. Geriau tą kiekį padalinti į kokias tai grupes, suskirstyti į atskirus langus.

Šiame darbe projektuotoje sistemoje renkant vartotojo sąsajos reikalavimus, kai kurie iš jų palietį aukščiau išvardintus principus. Todėl vartotojo sąsaja buvo suprojektuota kaip įmanoma griežčiau prisilaikant šių principų.

4. TYRIMO DALIS

Programinė įranga yra kaip CASE įrankis specifikuotojui, kuris nori nors kiek automatizuoti ir palengvinti agregatinės specifikacijos sudarymo procesą. Tam, kad įvertinti programinės įrangos atliekamą pasikartojančio darbo perėmimą iš vartotojo, programinės įrangos teikiamus palengvinimus ir pagalbą, teikiamą vartotojui, reikia įvesti įvertinimo metrikas.

4.1. Vertinimo rezultatai

4.1 lentelė. Programinės įrangos vertinimo rezultatai

Parametras	Aprašymas
Saugumas	Nėra. Programoje nėra realizuoti vartotojų autorizavimo ar kitos galimybės
Išplečiamumas	Yra. Pateikiamas programinės įrangos kodas ir dokumentacija. Programinė įranga rašyta naudojant šablonus, kad vėliau galima būtų funkcionalumą lengvai plėsti neperrašant kodo.
Pernešamumas	Yra. Programinė įranga sukurta naudojant JAVA programavimo kalbą ,todėl nuo sistemos nepriklauso ir gali būti pernešama.
Sąsajos galimybės	Kol kas nėra. Ateityje žadama praplėsti sistemos funkcionalumą, kad būtų galima importuoti duomenis iš kitų panašių sistemų, pvz SPIN.
Panaudojamumas	Yra. Vartotojo dokumentacija pateikiama.
Patvarumas	Yra. Vartotojo įvedami duomenys tikrinami, pateikiami pranešimai apie neteisingai įvestus duomenis.

4.2. Palyginimo kriterijai

Programinės įrangos įvertinimui gali būti naudojami tokie kriterijai:

- Imitavimo galimybė
- Validavimo galimybė
- Verifikavimo galimybė
- Matematinis modeliavimas
- Grafinis specifikacijos redaktorius
- Vartotojo sąsaja
- Platforma
- Specifikacijos tikrinimas saugumo požiūriu
- Specifikacijos tikrinimas gyvybingumo požiūriu
- Specifikacijos tikrinimas neteisingos pabaigos būsenos požiūriu
- Modeliavimo kalba

- Duomenų tipai
- „Debugeris“

4.3. Formalių metodų CASE įrankių palyginimas

4.2 lentelė. CASE įrankių palyginimas

Kriterijus / Sistema	SPIN	Murϕ	AgDraw su papildomos posistemės	FDR2
Imitacinis modeliavimas	Ne	Ne	Taip	Ne
Validavimas	Taip (reali laiku)	Taip (dalinai)	Taip(statiškai)	Taip
Verifikavimas	Taip	Ne	Ne	Ne
Testų generavimas	Ne	Ne	Taip	Ne
Matematinis modeliavimas	Ne	Ne	Taip	Ne
Grafinis specifikacijos redaktorius	Ne	Ne	Taip	Ne
Platforma	Unix, Linux, cygwin, Plan9, Inferno, Solaris, Mac ir Windows	Windows based	Java (tinka visoms platformoms)	Widnows based
Specifikacijos tikrinimas saugumo požiūriu	Taip	Taip	Taip	Taip
Specifikacijos tikrinimas gyvybingumo požiūriu	Taip	Ne	Taip	Ne
Specifikacijos tikrinimas neteisingos pabaigos būsenos požiūriu	Taip	Taip	Taip	Ne
Modeliavimo kalba	Promella	Pascal based	PLA-CA	CSP form
Duomenų tipai	Bitai, baitai, int ir masyvai	Visi Pascal tipai	Visi standartiniai JAVA tipai ir Galima apsirašyti savo duomenų struktūras	Visi galimi standartiniai tipai
„Debugeris“	Taip	Ne	Ne	Taip
Grafinė vartotojo sąsaja	Taip	Ne	Taip	Taip

5. EKSPERIMENTINĖ DALIS

Šioje dalyve pateikiamas „Duomenų perdavimo protokolo su adaptyviojo komutavimo metodu“ konceptualinis modelis, specifikacija, specifikacijos apdorojimas su programine įranga, ir kaip specifikacija atrodo XML kalboje.

5.1. Konceptualinis modelis [14]

Tokio tipo sistemose galima perdavinėti dviejų tipų srautus:

- Failiniai seansai – tai realaus laiko srautai, kurių perdavimo kokybei stipriai įtakoja vėlinimas.
- Paketai – tai nerealaus laiko srautai, kurių perdavimo kokybei vėlinimas ne toks svarbus.

Šioje sistemoje aptarnavimo kanalai suskirstomi į 3 grupes:

- Failinių seansų aptarnavimo kanalų grupė N_f. Čia gali būti aptarnaujami tik failiniai seansai.
- Paketų aptarnavimo kanalų grupė N_p. Čia gali būti aptarnaujami tik paketai.
- Bendra kanalų grupė N_{fp}, kurioje gali būti aptarnaujami tiek paketai, tiek failiniai seansai.

Dar sistemoje apibrėžiami intensyvumai:

- λ_1 – nurodo, kokiu intensyvumu failiniai seansai patenka į sistemą. μ_1 – nurodo, kokiu intensyvumu failiniai seansai yra apdorojami sistemoje.
- λ_2 – nurodo, kokiu intensyvumu paketai patenka į sistemą. μ_2 – nurodo, kokiu intensyvumu paketai yra apdorojami sistemoje.

Darbas vykdomas tokiu principu:

- Kai ateina realaus laiko failiniai seansai, tai jie gali užimti visus kanalus bendroje aptarnavimo kanalų grupėje N_{fp} (taip sumažinant nejautrių laikui paketinių seansų skaičių) ir visus kanalus savo failinių seansų kanalų grupėje N_f.
- Failinė sesija yra aptarnaujama tik tuo atveju, jei yra laisvų kanalų tose dviejose kanalų grupėse - N_{fp} ir N_f. Jei laisvų kanalų nėra, tada failinė sesija yra praradinėjama.
- Kai pasibaigia failiniai seansai, ir jei atsilaisvina kanalai bendroje kanalų grupėje, tai čia vėl gali būti aptarnaujami paketiniai seansai.

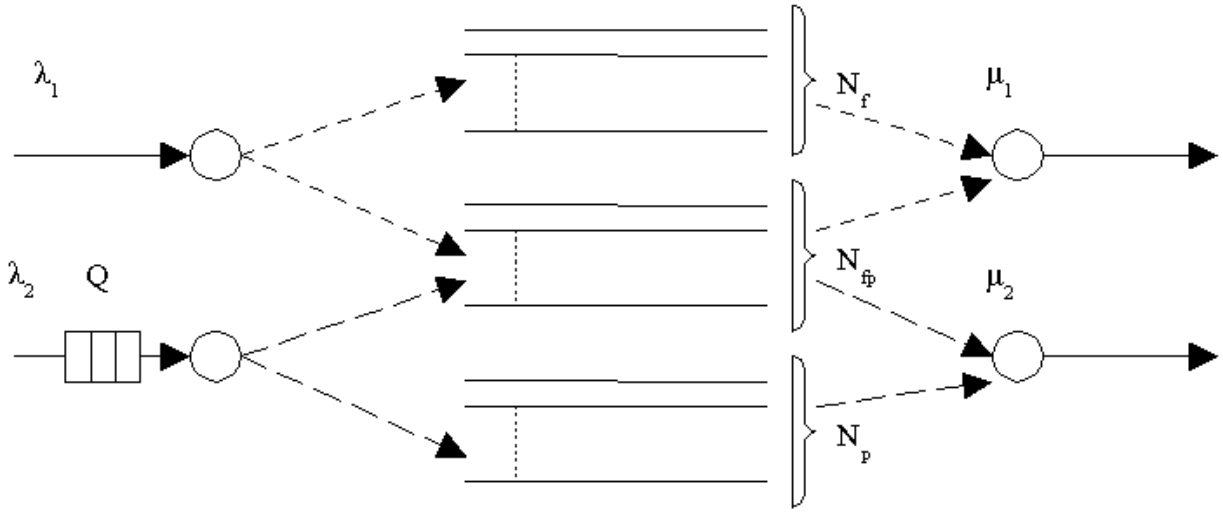
Dar svarbu paminėti, kad failiniai realaus laiko seansai ne visada efektyviai išnaudoja kanalą, nes yra momentai, kai failinių seansų aptarnavime atsiranda pauzės. Dėl to čia įvedamas koeficientas ρ, kuris įvertina failinių seansų aptarnavimo pauzes, ir tada būtent pauzių metu, gali būti perdavinėjami nerealaus laiko srautai taip siekiant efektyviau išnaudoti kanalą.

Kad neprarasti ateinančių paketų, jie yra dedami į eilę Q (buferizuojami). Eilės ilgis yra apribotas ir jo reikšmę nusako dydis l. Kai eilė prisipildo maksimaliai ir norint neprarasti paketų, jų srauto aptarnavimui yra suteikiamas didesnis prioritetas ir jie gali užimti daugiau kanalų bendroje aptarnavimo kanalų grupėje. Todėl gaunasi, kad jei ateina failinė sesija ir visi kanalai grupėje N_f yra užimti, ji gali pereiti į grupę N_{fp} tik tuomet, jei paketų skaičius eilėje Q neviršija dydžio l. Jei paketų eilė Q yra pilna ir ateina dar vienas paketas, tai jis išstumia bendroje kanalų grupėje N_{fp} vieną failinę sesiją.

Galimi sistemos įvykiai:

- Atėjo failinė sesija;
- Atėjo paketas;
- Baigėsi failinės sesijos aptarnavimas N_f kanalų grupėje;
- Baigėsi failinės sesijos aptarnavimas bendroje kanalų grupėje N_{fp}.
- Baigėsi paketo aptarnavimas.

Adaptyvaus trakto sistemos schema:



5.1 pav. Adaptyvaus trakto sistemos schema

5.2. Agregatinė specifikacija

Šio konceptualinio modelio agregatinė specifikacija pagal nurodymus paminėtus skyrelyje „2.4 Agregatinės specifikacijos metodo metamodelis„ aprašoma taip:

1) X = ∅
2) Y = ∅
3) E‘ = ∅
4) E‘‘ = {e_1“, e_2“, e_3“, e_4“, e_5“}

e_1“ – atėjo failinė sesija

e_2“ – atėjo paketas

e_3“ – baigėsi failinės sesijos aptarnavimas N_f kanalų grupėje

e_4“ – baigėsi failinės sesijos aptarnavimas N_{fp} kanalų grupėje

e_5“ – baigėsi paketo aptarnavimas

5) $\upsilon = \{N_f(t), N_{fp}(t), Q(t)\}$

N_f – failų kanalų grupės užimtumas

N_{fp} – failinėmis sesijomis užimtų kanalų skaičius bendrų kanalų grupėje N_{fp}.

Q – paketų eilė

6) $Z_\upsilon(t) = \{ w(e_1“, t), w(e_2“, t), w(e_3“, t), w(e_4“, t), w(e_5“, t) \}$

7) Sistemos būsena:

$Z(t) = \{ N_f(t), N_{fp}(t), Q(t), w(e_1“, t), w(e_2“, t), w(e_3“, t), w(e_4“, t), w(e_5“, t) \}$

8) Sistemos būsena: $Z(t_0) = \{0, 0, 0, \infty, \infty, 0, 0, 0\}$

9) Operatoriai:

H (e_1“):

$N_f(t+0) = \{$ $N_f(t)+1$, jei $N_f(t) < N_{fMAX}$,

$N_f(t)$, kitais atvejais;

$N_{fp}(t+0) = \{$ $N_{fp}(t)+1$, jei $(N_f(t) = Nf_{MAX})$ ir $(N_{fp} < N_{fpMAX})$ ir $(Q>l_1)$,

$N_{fp}(t)$, kitais atvejais;

$Q(t+0) = Q(t)$;

$w(e_1”, t+0) = w(e_1”, t)$;

$w(e_2”, t+0) = w(e_2”, t)$;

$w(e_3”, t+0) = N_f(t) * 1$;

$w(e_4”, t+0) = N_{fp}(t) * 1$;

$w(e_5”, t+0) = (N_p(t) + (N_{fpMAX}(t)-N_{fp}(t)) + (N_{fp}(t) + N_f(t) * \rho) * 2$;

H (e_2“):

$N_f(t+0) = N_f(t)$;

$N_{fp}(t+0) = \{$ $N_{fp}(t)-1$, jei $(N_{fp} > 0)$ ir $(Q>l_2)$,

$N_{fp}(t)$, kitais atvejais;

$Q(t+0) = Q(t) + 1$;

$w(e_1”, t+0) = w(e_1”, t)$;

$w(e_2'', t+0) = w(e_2'', t);$

$w(e_3'', t+0) = w(e_3'', t);$

$w(e_4'', t+0) = w(e_4'', t);$

$w(e_5'', t+0) = (N_p(t) + (N_{fpMAX}(t)-N_{fp}(t)) + (N_{fp}(t) + N_f(t) * \rho) *2;$

H (e_3''):

$N_f(t+0) = N_f(t) - 1;$

$N_{fp}(t+0)= N_{fp}(t);$

$Q(t+0) = Q(t);$

$w(e_1'', t+0) = w(e_1'', t);$

$w(e_2'', t+0) = w(e_2'', t);$

$w(e_3'', t+0) = N_f(t) * 1;$

$w(e_4'', t+0) = w(e_4'', t);$

$w(e_5'', t+0) = (N_p(t) + (N_{fpMAX}(t) - N_{fp}(t)) + (N_{fp}(t) + N_f(t) * \rho) *2;$

H (e_4''):

$N_f(t+0) = N_f(t);$

$N_{fp}(t+0)= N_{fp}(t) - 1;$

$Q(t+0) = Q(t);$

$w(e_1'', t+0) = w(e_1'', t);$

$w(e_2'', t+0) = w(e_2'', t);$

$w(e_3'', t+0) = w(e_3'', t);$

$w(e_4'', t+0) = N_f(t) * 1;$

$w(e_5'', t+0) = (N_p(t) + (N_{fpMAX}(t) - N_{fp}(t)) + (N_{fp}(t) + N_f(t) * \rho) *2;$

H (e_5''):

$N_f(t+0) = N_f(t);$

$N_{fp}(t+0)= N_{fp}(t);$

$Q(t+0) = \{$ $Q(t) -1$, jei $Q(t) > 0$,

0, kitais atvejais;

$w(e_1'', t+0) = w(e_1'', t);$

$w(e_2'', t+0) = w(e_2'', t);$

$w(e_3'', t+0) = w(e_3'', t);$

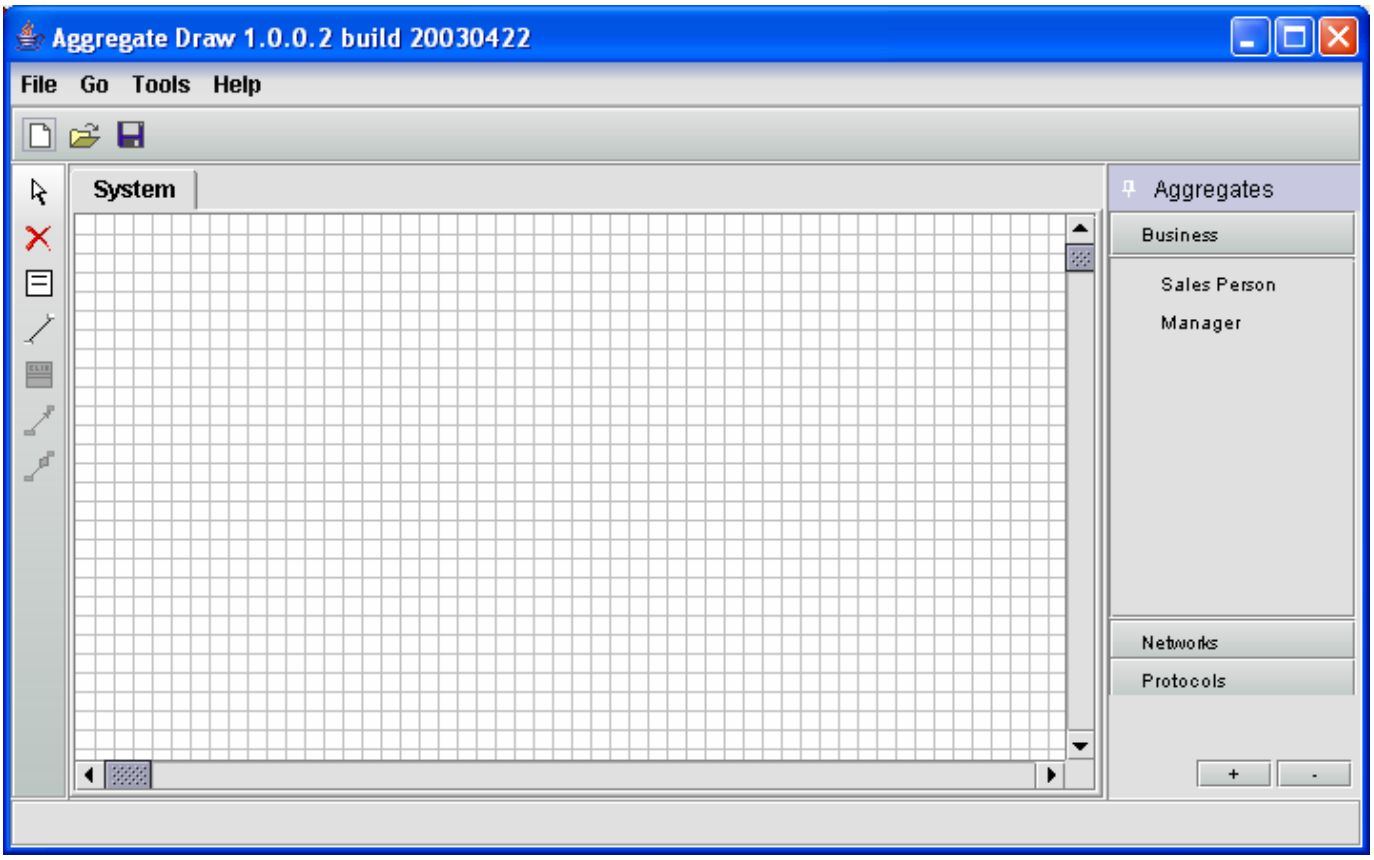
$w(e_4'', t+0) = w(e_4'', t);$

$w(e_5'', t+0) = \{ \quad (N_p(t) + (N_{fpMAX}(t) - N_{fp}(t)) + (N_{fp}(t) + N_f(t) * \rho) * 2$, jei $Q(t) > 0$,

0, kitais atvejais;

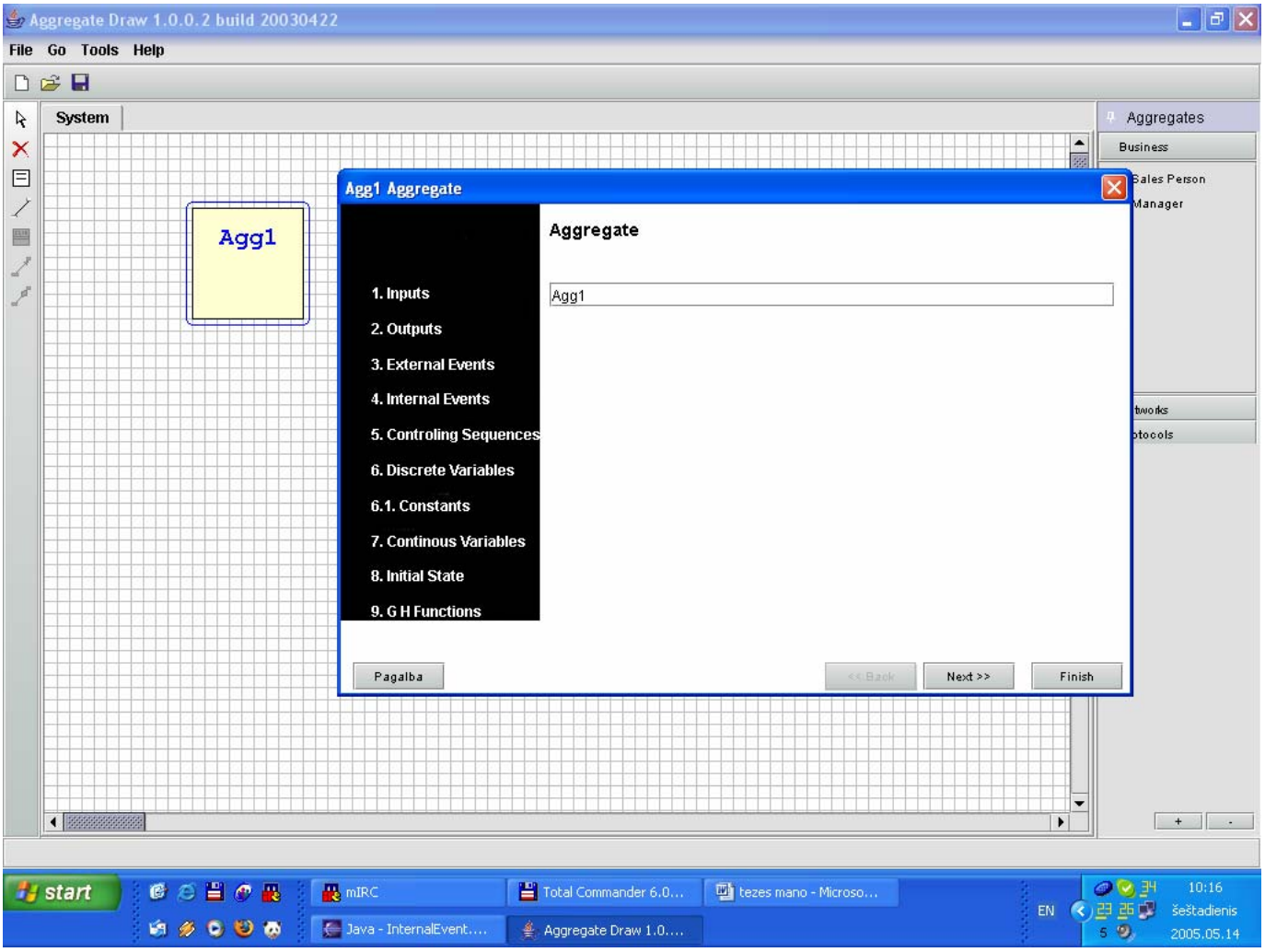
5.3. Agregatinė specifikacijos apdorojimas programine priemone „AgDraw“

Pasileidžiame intelektualųjį redaktorių „Agdraw“:



5.2 pav. Intelektualiojo redaktoriaus „Agdraw“ programos langas

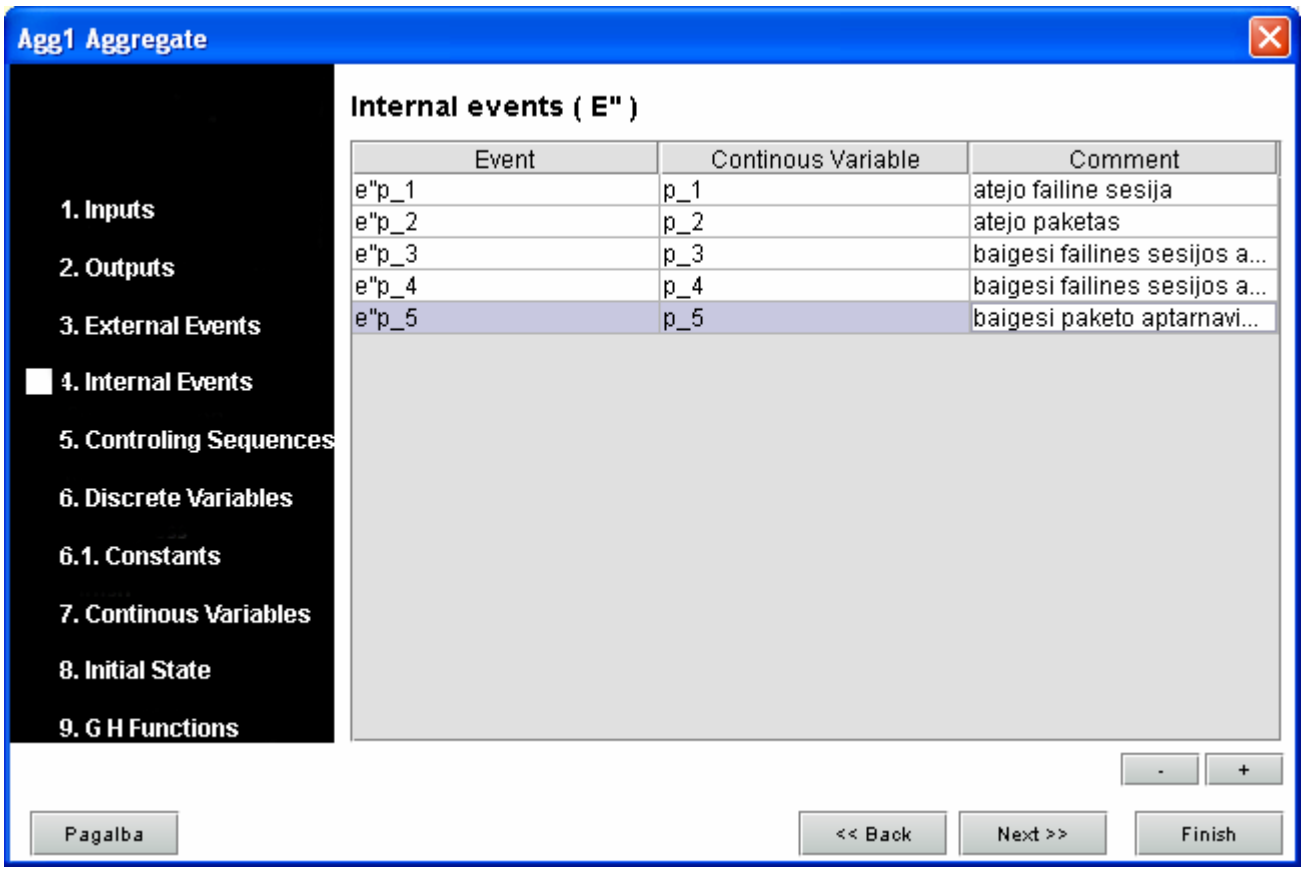
Su įrankiu „Agregatas“ iš kairėje esančios įrankių juostos nupaišome vieną agregatą, ir paspaudę ant agregato dešinį pelės mygtuką, iš „pop-up“ meniu iškviečiame komandą „Properties“, kuri leis mums pradėti specifikuoti agregatą.



5.3 pav. Agregato specifikacijos aprašymo vedlys

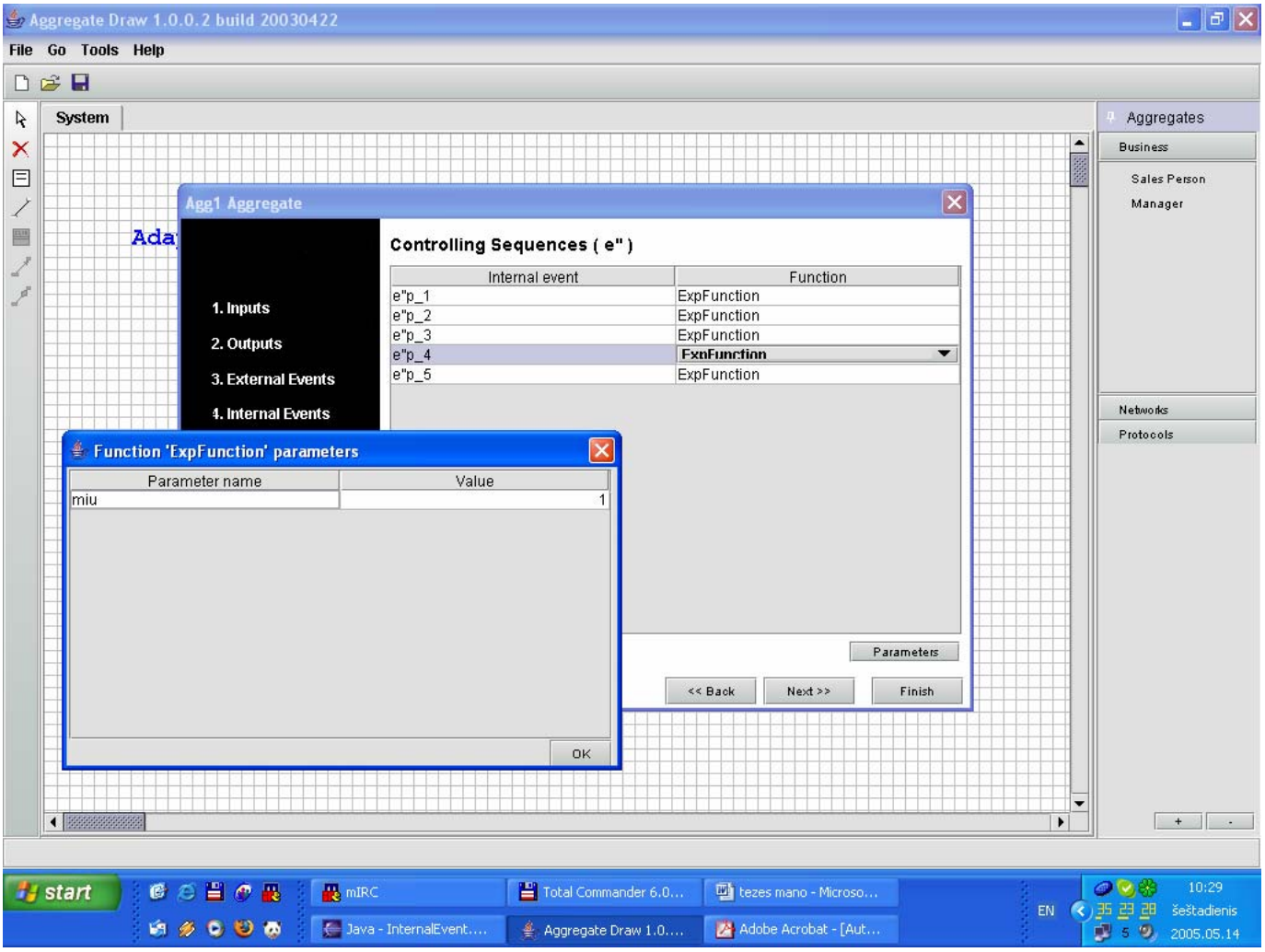
Iš eilės einame per specifikacijos vedlio punktus ir įvedame prašomą informaciją. Pirmajame punkte nurodome agregato vardą „AdaptyvusTraktas“. Antrajame lange reikia nurodyti agregato įėjimus, tačiau kadangi šiame pavyzdyje įėjimų nėra, tiesiog pereiname prie sekančio punkto. Su agregato išėjimais bei išoriniais įvykiais situacija lygiai tokia pati kaip prieš tai.

Vidinių įvykių lange nurodome kokie bus vidiniai įvykiai:



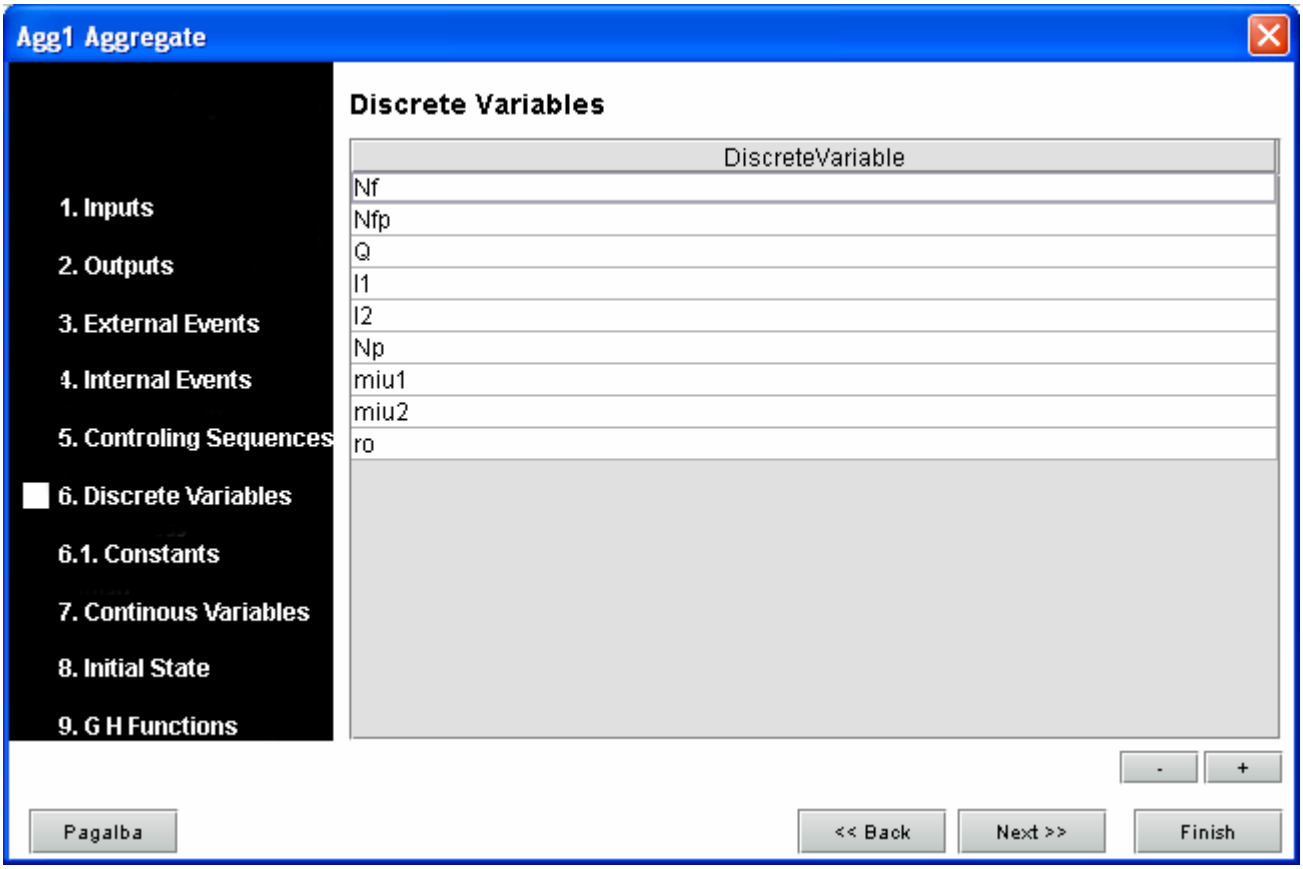
5.4 pav. Agregato vidiniai įvykiai

Kontrolinių sekų lange, nurodome, nuo kokios funkcijos priklausys vidinių įvykių intensyvumas. Kiekvienai parinktai funkcijai taip pat nurodomos jos parametro (-ų) reikšmės parametrų lange.



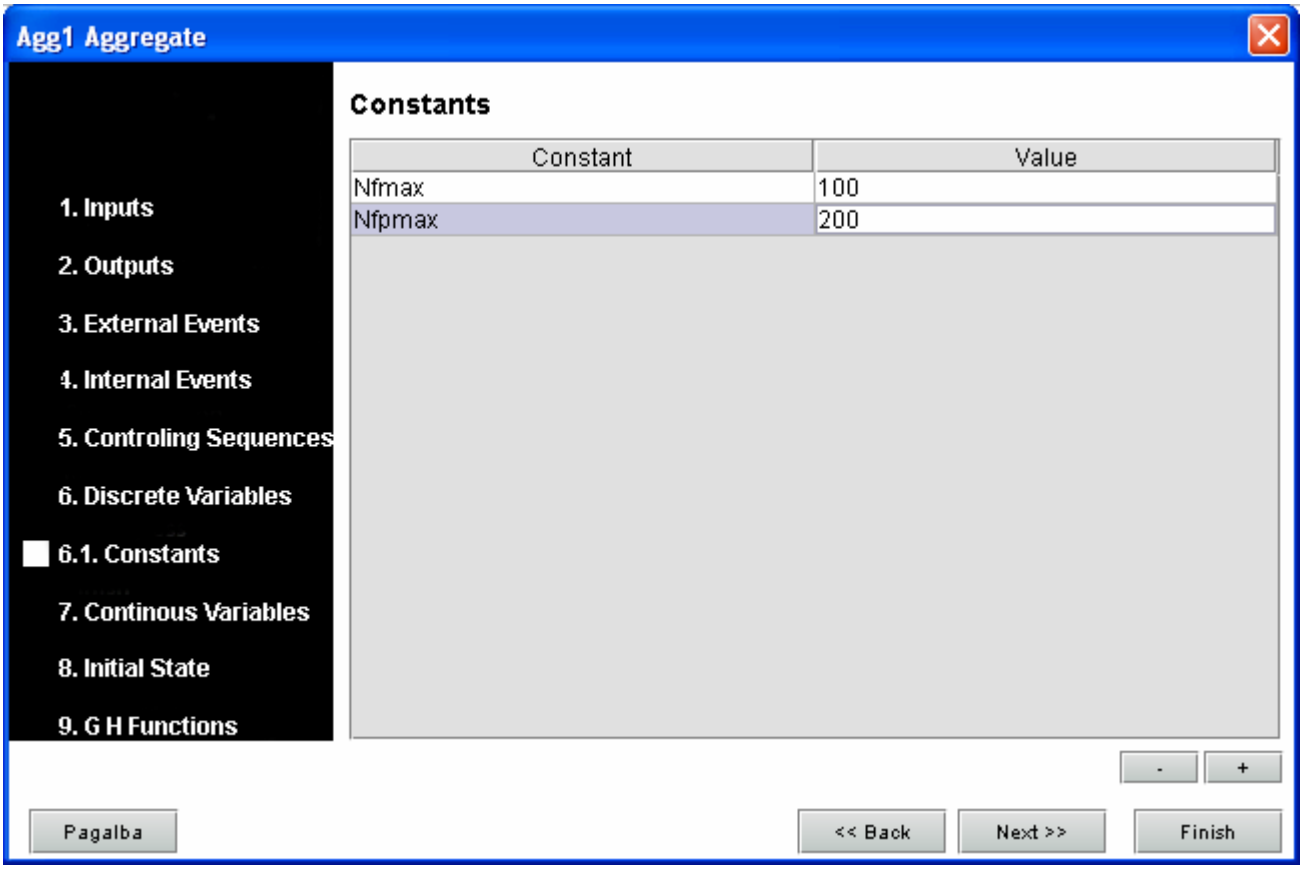
5.5 pav. Agregato valdančiosios sekos

Nurodome diskretinius kintamuosius:



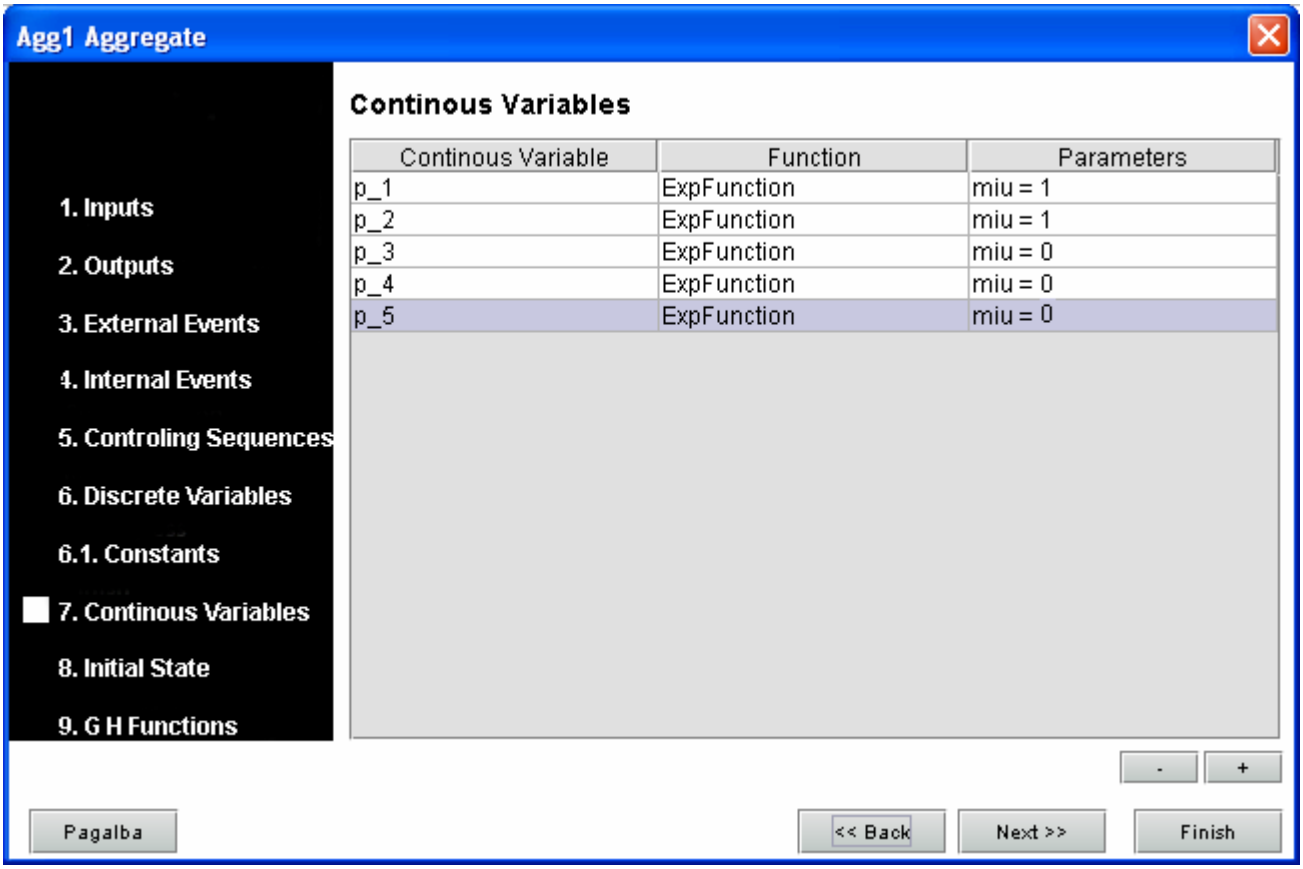
5.6 pav. Agregato diskretiniai kintamieji

Sekančiame lange nurodome naudojamas konstantas:



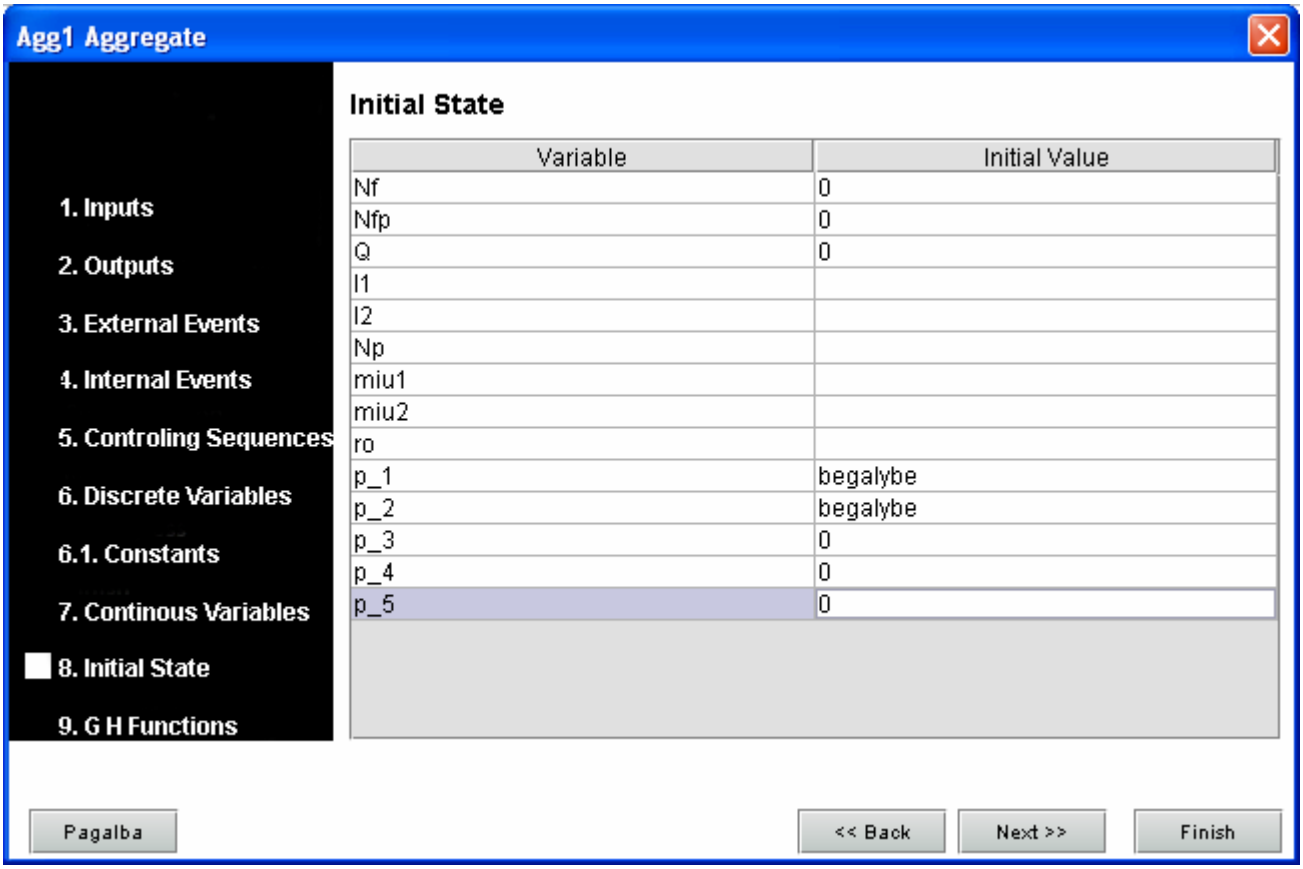
5.7 pav. Agregato konstantos

Toliau pateikiami tolydieji kintamieji, kurie kinta pagal nurodytas funkcijas. Šalia pateikiamos funkcijų parametrų reikšmės.



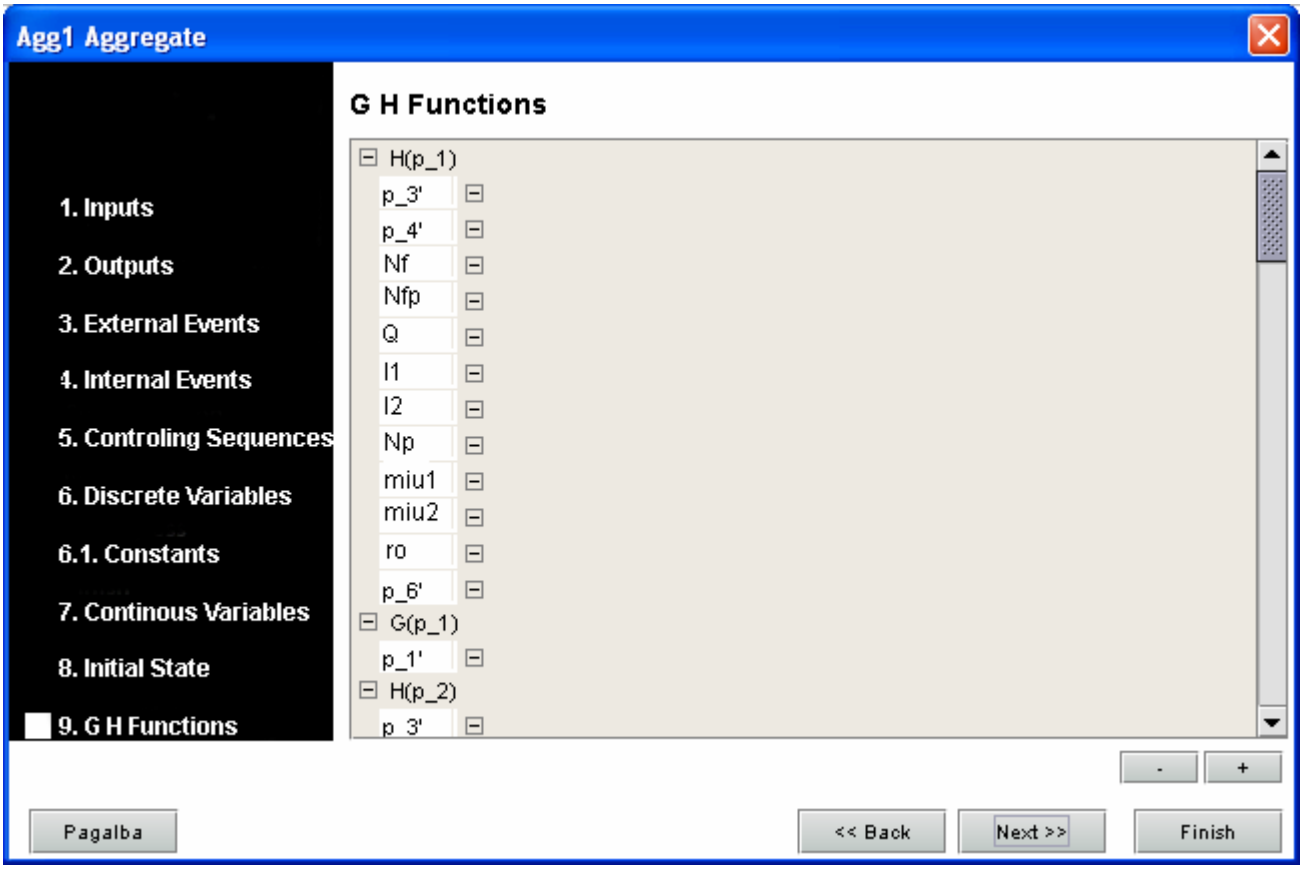
5.8 pav. Agregato tolydieji kintamieji

Sekančiame lange nurodoma pradinė agregato būsena.



5.9 pav. Agregato pradinė būsena

Ir galiausiai belieka aprašyti operatorius G ir H.



5.10 pav. Agregato G ir H operatorių aprašymo langas

Taigi, tarkime šis specifikacijos fragmentas:

H (e_1“):

$N_f(t+0) = \{$ $N_f(t)+1$, jei $N_f(t) < N_{fMAX}$,

$N_f(t)$, kitais atvejais;

$N_{fp}(t+0) = \{$ $N_{fp}(t)+1$, jei $(N_f(t) = Nf_{MAX})$ ir $(N_{fp} < N_{fpMAX})$ ir $(Q>l_1)$,

$N_{fp}(t)$, kitais atvejais;

$Q(t+0) = Q(t)$;

$w(e_1”, t+0) = w(e_1”, t)$;

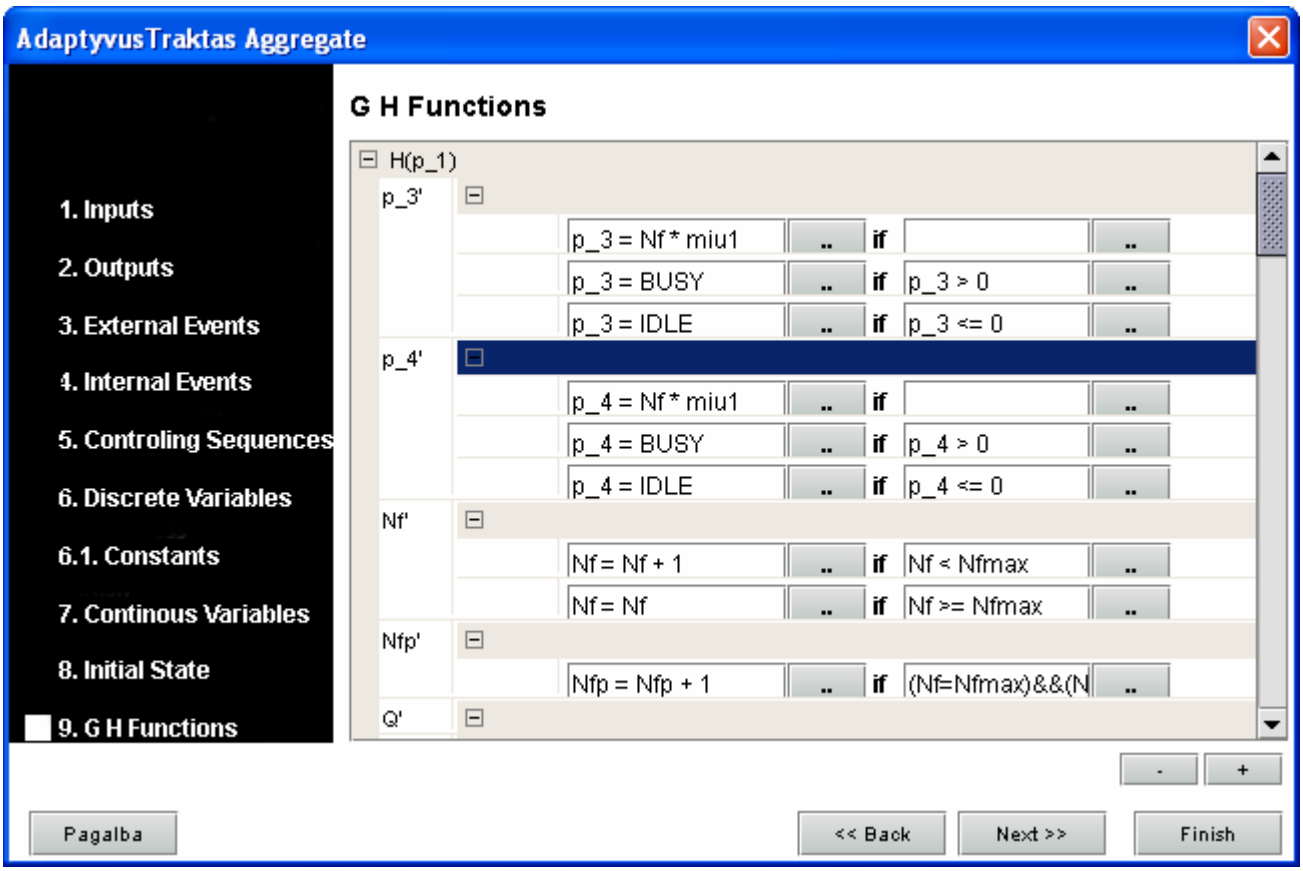
$w(e_2”, t+0) = w(e_2”, t)$;

$w(e_3”, t+0) = N_f(t) * 1$;

$w(e_4”, t+0) = N_{fp}(t) * 1$;

$w(e_5”, t+0) = (N_p(t) + (N_{fpMAX}(t)-N_{fp}(t)) + (N_{fp}(t) + N_f(t) * \rho) *2$;

Intelektualiojo redaktoriaus specifikavimo lange aprašomas taip:



5.11 pav. Agregato G ir H operatorių aprašymo langas su aprašyta vienu iš operatorių

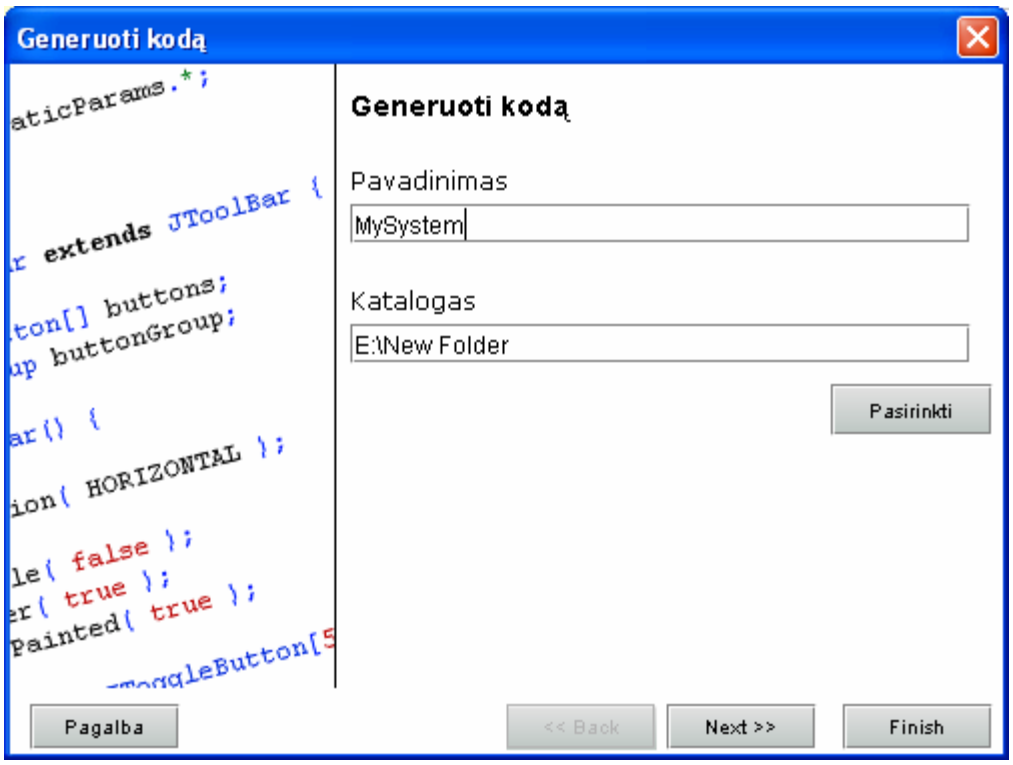
Kai visa specifikacija supildoma pilnai, diagramą galima užsaugoti į kompiuterio diską. Gauti išsaugoti failai: „AdaptyvusTraktas.agg“, „sistema.sys“, ir kiekvienam aprašytam tipui ar duomenų struktūrai po atskirą failą su galūne *.sig.

Šios specifikacijos failų XML turinį galite rasti 1 priede .

5.4. Kodo generavimas iš aprašytos specifikacijos imitacinio modeliavimo posistemei

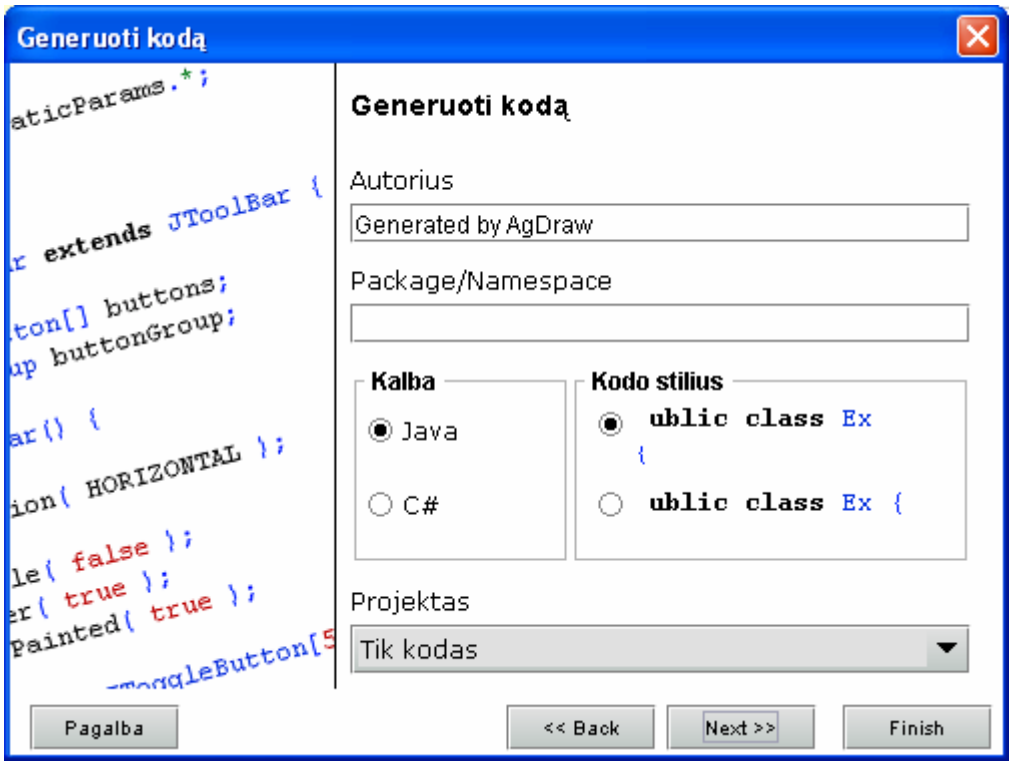
Norint atlikti specifikuojamos sistemos imitacinį modeliavimą, reikia iš turimos specifikacijos sugeneruoti sistemos karkasą, kurį vėliau papildžius renkamomis statistikomis galima vykdyti ir tokiu būdu kaupti statistikas apie imituojamą sistemą.

Programinėje įrangoje tai atliekama pasirinkus iš meniu „Go“ -> „Imitation“. Pradiniame lange nurodome kur saugoti kodą ir koks generuojamo kodo projekto pavadinimas.



5.12 pav. Kodo generavimo vedlio pradinis langas

Sekančiame lange nurodome koks generuojamo kodo kodavimo stilius ir kokia kalba generuoti kodą. Kol kas programinėje įrangoje realizuota tik JAVA kalbos kodo generavimas.



5.13 pav. Kodo generavimo vedlys – kalbos pasirinkimo langas

Sekančiame lange pranešama, kad nustatymai baigti ir kad galima pradėti generuoti kodą. Apie kodo sugeneravimo sėkmę pranešama sekančiame lange. Jei kodas sugeneruotas sėkmingai, tai nurodytoje direktorijoje gauname tokius failus:

- Kiekvieną agregatą aprašantis failas „Agregatovardas.java“
- Sistemą aprašantis failas „MyAgSystem.java“
- Imitacinį modeliavimą aprašantis failas „Simulation.java“

Sugeneruoto kodo pavyzdžius galima rasti 4 priede.

6. IŠVADOS

6.1. Išvados

- Imitacinis modeliavimas leidžia įvertinti modeliuojamos sistemos darbą ir ją charakterizuojančius parametrus dar prieš pradedant sistemos projektavimą – tokiu būdu galima sutaupyti didelius sistemos kūrimo kaštus.
- Grafinis interaktyvus agregatinės specifikacijos redaktorius padeda specifikuotojui (vartotojui) daug greičiau, su mažesne klaidų tikimybe suspecifikuoti sistemą.
- Išsaugotą specifikaciją XML formatu galima lengvai panaudoti kitoms posistemėms – matematiniam modeliavimui, testų generavimui ir validavimui. Be to, lengvai galima atlikti transformacijas ir pritaikyti specifikaciją kitoms sistemoms ar poreikiams.
- Jei specifikuojama panašios sistemos, pasikartojantį tokių pačių agregatų specifikavimą visiškai panaikina interaktyvaus redaktoriaus agregatų šablonų aprašymo galimybė.
- Šiame darbe aprašyta programinė įranga „Agdraw“ suprojektuota taip, kad lengvai galima praplėsti jos funkcionalumą papildomais komponentais ar posistemėmis. Tai labai svarbu ateityje plėtojant šio CASE įrankio funkcionalumą.
- PLA-CA metamodelio sukūrimas leidžia „priversti“ kompiuterį suprasti PLA modelį ir tada kompiuterizuoti darbą su juo.
- Kodo generavimo galimybė prieš atliekant imitacinį modeliavimą, gali būti panaudojama tolimesniems sistemos kodavimo darbams, kaip sistemos karkasas, pagreitinantis sistemos kodavimą.

6.2. Tolimesni darbai

- Norint, kad kuriama Agregatinių specifikacijų redagavimo sistema būtų naudojama plačiau, reikalinga įvesti modelių transformacijas iš PLA į kitas formalias kalbas.
- Taip pat reikalinga sudaryti galimybę, keistis duomenimis su jau dabar pasaulyje žinomomis panašaus tipo programomis (kaip pvz. SPIN)
- Imitacinio modeliavimo naudota biblioteka „JPranas“ nelabai atitinka visus reikalavimus atlikti pilnos agregatines specifikacijos (pagal tuos 9 punktus) imitacijai, todėl reikia pakeisti/perrašyti minėtą biblioteką.

7. LITERATŪRA

1. Bakanas, A., Packevičius, Š., Pranevičius, H. Agregatinių specifikacijų grafinis vaizdavimas// Informacinė visuomenė ir universitetinės studijos – 2004: tarptautinės konferencijos pranešimų medžiaga [Kaunas, 2004]. Kaunas 2004. p. 164 – 167.
2. Bakanas, A., Packevičius, Š., Pranevičius, H. Intelektualus agregatinių specifikacijų redaktorius// Informacinės technologijos 2005: tarptautinės konferencijos pranešimo medžiaga [Kaunas, 2005]. Kaunas 2005.
3. Curie, A., J. A Comparison of Three Model Checkers Applied to a Distributed Database Problem// 4th Irish Workshop on Formal Methods: tarptautinės konferencijos pranešimo medžiaga [5th-6th July, 2000]. Maynooth, Ireland, 2000.
4. Fernandes, R., Alex J. Cowie. Capturing informal Requirements as Formal Models// AWRE'04 9th Australian Workshop on Requirements Engineering: tarptautinės konferencijos pranešimų medžiaga [South Australia, 2004]. 2004.
5. Flake, S., Muller, W., Ruf, J. Structured English for Model Checking specification// GI/ITG/GMM Workshop [Frankfurt/M., Germany]. 2000
6. Garlan, D. Software Architecture: a Roadmap// The Future of Software Engineering: tarptautinės konferencijos pranešimo medžiaga [Ireland, 2000]. ACM Press, p.91–101. USA, 2000.
7. Hahnle, R., Johannisson, K., Ranta, A. An authoring tool for informal and formal requirements specifications// Fundamental Approaches to Software Engineering - 2002: tarptautinės konferencijos pranešimų medžiaga [France, 2002] p. 233-248. 2002
8. Japenga, R. Principles of software Driven User interface Desing for Business and Industrials Applications. [žiūrėta 2005-05-15] prieiga internete http://www.microtoolsinc.com/articles.php
9. Lamsweerde, V., A. Formal specification: a Roadmap// The Future of Software Engineering – 2000: tarptautinės konferencijos pranešimo medžiaga [Ireland, 2000] p. ACM Press, 147-159, USA, 2000.
10. Miller, S., P., Tribble, A., C., Heimdahl, M., P., E. Proving the shalls// FME2003 the 12th International FME Symposium [Italy, 2003].
11. Pranevičius, H. Aggregate approach for specification, validation, simulation and implementation of computer network protocols. Lecture Notes in Computer Science, 1991, No 502, Springer-Verlag, 433 – 477.
12. Pranevičius, H. Formal specification and analysis of distributed systems. Applications of AI to Production Engineering, Kaunas University of Technology Press, 1997, 269 - 322.

13. Pranevičius, H. Kompiuterių tinklų protokolų formalusis specifikavimas ir analize: agregatinis metodas. Kaunas University of Technology Press, 2003, 48 - 62.
14. Pranevičius, H., Germanavičius, V., Tumelis, G., Bakanas, A. PLA sistemų sudarymo automatizavimas. Kauno technologijos universitetas, Verslo informatikos katedra. 2005
15. Ravindran, K., Kwiat, K., A., Ding, G. Simulation-based Validation of Protocols for Distributed Systems// 38th Hawaii International Conference on System Sciences: tarptautinės konferencijos pranešimo medžiaga [Hawaii, January 03-06, 2005]. Hawaii, 2005.
16. Rumbaught, J., Jacobson, I., Booch, G. The Unified Modeling Language Reference Manual. [Boston, July 19, 2004], 2 edition. Addison – Wesley, Amsterdam (The Netherlands), 2004.
17. Theelen, B., D., Putten, V., D., P., H., A., Voeten, J., P., M. Using the SHE Method for UML-based Performance Modelling// System Specification and Design Languages: tarptautinės konferencijos pranešimų medžiaga [Dordrecht, 2003]. Kluwer Academic Publishers, chapter 12, p. 143-160, The Netherlands, 2003.
18. Vanderhaeghen, D., Zang, S., Hofer, A., Adam, O. XML-based Transformation of business Process Models – Enabler for Collaborative Business Process Management// Business, Technology, and Web (BTW 2005): tarptautinės konferencijos pranešimo medžiaga [Karlsruhe, Germany, March 2005]. Germany, 2005.

8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

8.1 lentelė. Terminai ir santrumpos

Sutrumpinimas/Terminas	Paaiškinimas
Agregatas	Modeliuojamos, specifikuojamos sistemos objektas agregatinėje specifikacijoje. Pvz., jei turime sistemą „traukinių stotis“, tai agregatas gali būti „traukinys“.
Agregatų diagrama	Ją sudaro agregatai ir ryšiai tarp jų.
CSP	DCommunivating Sequential Processes
GUI	Grafinė vartotojo sąsaja (Graphical User Interface)
JVM	Java virtuali mašina, tai sluoksnis tarp operacinės sistemos ir programinės įrangos, leidžiantis tą pačia programą vykdyti skirtingose operacinėse sistemose.
Neprogresyviniai ciklai	Tai tokie ciklai grafe, į kuriuos įėjus nėra išėjimo iš ciklo
Paskirstytoji sistema	Kompiuterių moksle, tai sistema, kurios objektai pasiskirstę per kelius kompiuterius tinkle. Formaliosiose specifikacijose tokia sistema supranta kaip sistema sudaryta iš kelių objektų ir tie objektai susiję tarpusavyje, bei priklausomi vienas nuo kito.
PLA	Piece Linear Aggregate
PLA-CA	PLA Computer Adopted
Specifikacija	Sistemos funkcionalumo aprašymas formaliais metodais
SPIN	Simple Promela Interpreter
UML	Unifikuota modeliavimo kalba (Unified Modeling Language)
XML	eXtended Markup Languade
XSL	eXtensible Stylesheet Language

9. PRIEDAI

9.1. 1 PRIEDAS. Agregatinės specifikacijos XML failų pavyzdžiai

Gauta XML ištrauka iš failo „AdaptyvusTraktas.agg“:

```
<?xml version='1.0' encoding='UTF-8'?>
<Aggregate name="AdaptyvusTraktas" >
 <Inputs />
 <Outputs />
 <DiscreteVariables >
       <DiscreteVariable name="Nf" value="0" />
       <DiscreteVariable name="Nfp" value="0" />
       <DiscreteVariable name="Q" value="0" />
       <DiscreteVariable name="l1" value="null" />
       <DiscreteVariable name="l2" value="null" />
       <DiscreteVariable name="Np" value="null" />
       <DiscreteVariable name="miu1" value="null" />
       <DiscreteVariable name="miu2" value="null" />
       <DiscreteVariable name="ro" value="null" />
 </DiscreteVariables>
 <ContinousVariables >
       <ContinuousVariable name="p_1" value="begalybe" >
             <H >
                   <Expression for="p_3" />
                     <Value exp="p_3 = p_3 * miu1" >
                         <apply>
                           <eq/>
                           <ci>p_3</ci>
                           <apply>
                             <times/>
                             <cn>Nf</cn>
                             <cn>miu1</cn>
                           </apply>
                         </apply>
                     </Value>
                         <Case >
                               <Condition exp="p_3 > 0" >
                                     <mathml>
                                           <cn>
```

```
   <apply>
     <lt/>
     <ci>p_3</ci>
     <ci>0</ci>
   </apply>
</cn>
                                    </mathml>
                              </Condition>
                              <Value exp="p_3 = BUSY" >
                                    <mathml>
                                          <apply>
                                            <eq/>
                                            <ci>p_3</ci>
                                            <apply>
                                               <ci>BUSY</ci>
                                            </apply>
                                          </apply>
                                    </mathml>
                              </Value>
                        </Case>

                  <Expression for="p_4" />
                  <Expression for="Nf" >
                        <Case >
                              <Value exp="Nf < Nfmax" >
                                    <mathml>
                                          <cn>
   <apply>
     <lt/>
     <ci>Nf</ci>
     <ci>Nfmax</ci>
   </apply>
</cn>
                                    </mathml>
                              </Value>
                              <Condition exp="Nf = Nf + 1" >
                                    <mathml>
                                          <cn>1</cn>
                                    </mathml>
                              </Condition>
```

```
                        </Case>
                        <Case >
                              <Value exp="Nf >= Nfmax" >
                                    <mathml>
                                          <cn>
   <apply>
     <geq/>
     <ci>Nf</ci>
     <ci>Nfmax</ci>
   </apply>
</cn>
                                    </mathml>
                              </Value>
                              <Condition exp="Nf = Nf" >
                                    <mathml>
                                          <cn>
   <apply>
     <eq/>
     <ci>Nf</ci>
     <ci>Nf</ci>
   </apply>
</cn>
                                    </mathml>
                              </Condition>
                        </Case>
                  </Expression>
                  <Expression for="Nfp" />
                  <Expression for="Q" />
                  <Expression for="l1" />
                  <Expression for="l2" />
                  <Expression for="Np" />
                  <Expression for="miu1" />
                  <Expression for="miu2" />
                  <Expression for="ro" />
            </H>
            <G >
                  <Expression for="p_1" />
            </G>
      </ContinuousVariable>
```

Gauta XML ištrauka iš failo „sistema.sys“:

```
<?xml version='1.0' encoding='UTF-8'?>
<System name="traktas1" >
 <SignalStructures />
 <Aggregates >
       <Aggregate name="AdaptyvusTraktas" />
 </Aggregates>
 <Connections />
 <SystemConstants />
</System>
```

9.2. 2 PRIEDAS. Konferencijos „Informacinė visuomenė ir universitetinės studijos 2004“ pranešimo medžiaga

Agregatinių specifikacijų grafinis vaizdavimas

Aivaras Bakanas, Šarūnas Packevičius, Henrikas Pranevičius

Kauno Technologijos universitetas, Informatikos fakultetas, Studentų g. 50, LT - 3031 Kaunas

Formaliai aprašant sistemas naudojamas PLA metodas. Sistemas galima aprašyti tekstu, kuris yra išsamus ir suprantamas, bet nėra labai vaizdus, reikia įdėmiau pasigilinti, norint suprasti specifikaciją.

Pranešime pateikiamas būdas atvaizduoti specifikacijas grafiškai, diagramomis tuo leidžiant greičiau suprasti kas jose pateikta. Taip pat pristatoma priemonė skirta specifikacijai sudaryti grafiniu būdu.

9.2.1. Įžanga

Sudarant sistemos agregatines specifikacijas, reikia sistemas aprašyti formalia kalba. Vienas iš aprašymo būdų yra aprašymas formaliu tekstu. Toks aprašymas nebūna vaizdus, norint suprasti apie ką yra aprašyta sistema, tenka atidžiai įsigilinti į specifikaciją. Daug laiko sugaištama bandant nustatyti sistemos dalis ir jų santykius, bendradarbiavimus. Kad palengvinti formalios agregatinės specifikacijos sudarymą ir jos peržiūrėjimą galima naudoti grafinius būdus. Agregatinės specifikacijos būtų vaizduojamos grafinėmis diagramomis. Taip pat jos būtų sudaromos grafiniu būdu. Turint galimybes vaizduoti specifikacijas grafiniu būdų, galima sukurti CASE priemones leidžiančias lengvai, kompiuteriu sudaryti norimas specifikacijas.

Grafinio specifikacijų sudarymo privalumai

- Vaizdžiai matosi specifikacija

  Vartotojas gali greitai susipažinti su pateikta specifikaciją, sistemoje egzistuojančiais elementais, jų savybėmis.

- Lengvai skaitoma
- Lengvai sudaroma

  Programinėje įrangoje realizuoti įrankiai vartotojui leidžia greičiau ir paprasčiau sudaryti specifikaciją. Jie palengvina arba nuo vartotojo nuima bereikalingą, besikartojantį darbą.

Trūkumai

- Diagramoje neatvaizduojamos išėjimo ir įėjimo funkcijos (jas reikia aprašyti atskirai)
- Nesimato pradinių būsenų (jas reikia aprašyti atskirai)

9.2.2. Programinės priemonės

Buvo kuriamos kelios CASE priemonės leidžiančios sudaryti agregatines specifikacijas grafiškai. Viena iš jų buvo pavadinta „Praxis“ vardu. Ji veikė tekstiniame rėžime ir schematiškai leisdavo pavaizduoti agregatus. Deja joje agregatinių specifikacijų vaizdavimas buvo gana skurdus, buvo galima

matyti ekrane tik vieną agregatą vienu metu, ir matyti tik jo įėjimus ir išėjimus. Nebuvo galima peržiūrėti visos sudarytos agregatinės specifikacijos vienoje diagramoje.

Kita agregatines specifikacijos sudarinėti priemone yra mūsų kuriama „AgDraw“ . Kurioje jau grafiniu rėžiu galima sudarinėti specifikaciją. Galima apsirašyti agregatus, kurtį jų šablonus, juos naudoti sudarant sistemos agregatinę specifikaciją. Taip pat sistema yra integruota su tos specifikacijos validavimo, verifikavimo ir pagal agregatinę specifikaciją aprašytos sistemos imitavimo moduliais.

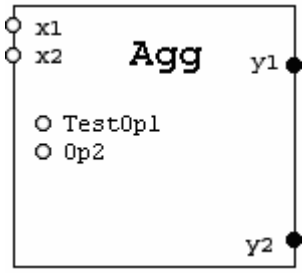
9.2.3. Agregatinės specifikacijų grafinis vaizdavimas

Agregatinės specifikacijos vaizduojamos grafinėmis diagramomis, kurios palengvina jų sudarymą. Agregatinės specifikacijos diagramose vaizduojami tik dviejų tipų elementai: agregatai ir ryšiai tarp jų.

Agregatas Vaizduojamas:

- Stačiakampiu
  Agregatą diagramoje atitinka stačiakampis su jame pažymėta kita informacija. Stačiakampis gali būtį įvairaus dydžio.
- Agregato vardu stačiakampyje
  Stačiakampyje vaizduojančiame agregatą, viršuje, viduryje užrašomas agregato vardas.
- Vidiniai įvykai atvaizduojami apskritimu ir operacijos pavadinimu
  Kiekvienas agregate vykstantis vidinis įvykis atvaizduojamas apskritimu ir šalia jo užrašomas jo pavadinimas. Vienas įvykis užrašomas žemyn į apačia po kito įvykio. Užrašai išlygiuojami pagal dešinį agregato kraštą. Vidiniai įvykiai vaizduojami agregato stačiakampio viduje, pradedami žymėti iš kart po pavadinimo.
- Išėjimai žymimi tamsiu apskritimu
  Agregato išėjimai pažymimi užtamsintu apskritimu ant agregato krašto, šalia užrašomas išėjimo signalas. Užrašas daromas agregato stačiakampio vidinėje dalyje.
- Įėjimai žymimi tuščiaviduriu apskritimu.
  Agregato įėjimai pažymimi tuščiaviduriu apskritimu ant agregato krašto, šalia užrašomas įėjimo signalas. Užrašas daromas agregato stačiakampio vidinėje dalyje.

Agregato pavyzdys pateiktas 9.1 paveikslėlyje.



9.1 pav. Agregatas

9.1 paveikslėlyje pateiktame agregate matome:

Pavadinimas: Agg

Įėjimo signalų aibė: X = { x1, x2 };

Išėjimo signalų aibe: Y = { y1, y2 };

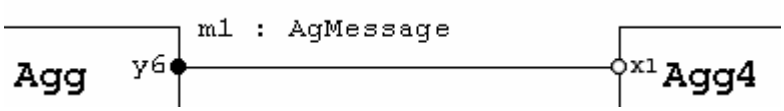
Vidiniai įvykai: TestOp1, Op2

Tačiau grafinėje agregato vaizdavime neatsispindi visos jo savybės. Diagramoje nematome pavaizduotos išorinių įvykių aibės, bet juos galima numatyti pagal įeinančius signalus. Taip pat nematome valdančiųjų sekų, tolydinės ir diskrečios agregato būsenos, pradinių būsenų. Diagramoje taip pat neatsispindi perėjimo ir išėjimo operatoriai. Juos galima atvaizduoti atskirose diagramose vaizduojant UML „activity“ diagramas. Trūkstama informacija pateikiama priede prie specifikacijos. Diskrečias ir tolydžias agregato būsenos dedamąsias galima vaizduoti po vidiniu operacijų, išorinius įvykius po vidinių įvykiu kokiu nors kitokiu žymėjimu, pavyzdžiui stačiakampiais. Tačiau tai apkrautų per daug diagramą. Programinėje įrangoje būtų galima realizuoti paslėpimą arba dalinį rodymą laukų.

Kanalai tarp agregatų vaizduojami grafiškai diagramoje sujungiant agregatus diagramoje. Tai yra daug patogiau negu naudojant lenteles, kuriose pažymėta koks agregatas su kuriuo agregatu sujungtas. Kanalai tarp agregatų vaizduojami:

- Sujungiamai agregato išėjimai su kitų agregatų įėjimais.
  Jungiami vieno agregato išėjimai su kito agregato įėjimais linija (esant poreikiui sudarytai iš atkarpų). Linijos jungia agregato tuščiavidurius apskritimus su užpildytais. Linijos neina per agregatą vaizduojančio stačiakampio vidų. Kanalai taip pat gali išsišakoti ir tas pats kanalas išeidamas iš vieno agregato gali būti prijunktas prie kelių agregatų įėjimų.
- Ant kanalo vaizduojami agregato siunčiami pranešimai
  Ant kanalo, jungiančio agregatus, rašomi juo perduodami pranešimai. Nurodoma pranešimo pavadinimas ir tipas.

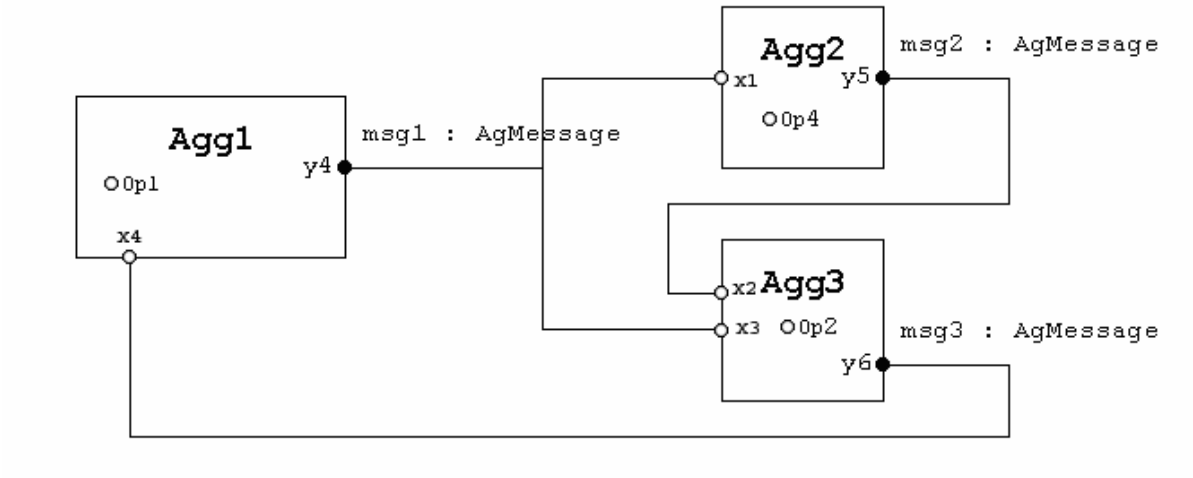
Kanalo pavyzdys pateiktas 9.2 paveikslėlyje.



9.2 pav. Kanalas tarp agregatų

Paveikslėlyje 9.2 pateiktame agregatų sujungime vaizduojami sujungtu kanalu du agregatai Agg ir Agg4, agregatas Agg išduoda išėjimo signalą y6, agregatas Agg4 prima įėjimo signalą x1 ir kanalu jungiančiu agregatus perduodamas pranešimas m1.

Apjungus agregatus ir kanalus tarp jų gauname agregatų diagramą. Diagramos pavyzdys pateiktas paveikslėlyje 9.3.

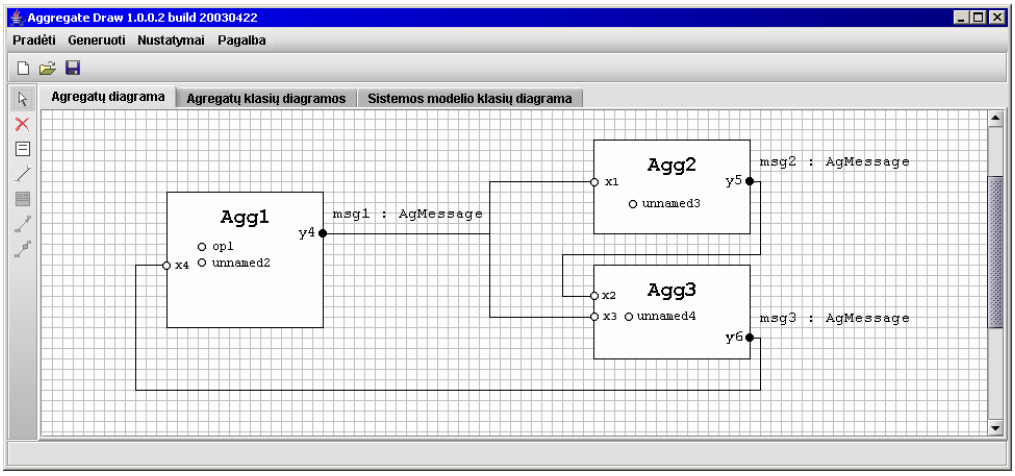


9.3 pav. Agregatų diagrama

Paveikslėlyje 9.3 pavaizduota agregatų diagrama. Joje yra aprašyti 3 agregatai Agg1, Agg2, Agg3 jų įėjimo ir išėjimo signalai, kanalai jungiantys agregatus, bei jais perduodami pranešimai. Tai papt pavaizduotas kanalas jungiantis iškarto tris agregatus.

9.2.4. Agregatų braižyklė

Agregatų braižyklė „AgDraw“ yra priemonė skirta grafiškai sudaryti agregatines specifikacijas. Ji palengvina agregatinės specifikacijos sudarymą vartotojui. Programos vaizdas pateiktas 9.4 paveikslėlyje.



9.4 pav. Programinės įrangos pagrindinis langas

Programinė įranga teikia vartotojui tokias funkcijas:

- Agregatinės specifikacijos sudarymas
  Vartotojas grafiškai sudaro specifikaciją. Naudojat pelę ekrane paišomi agregatai, sudaromi ryšiai tarp jų.
- Agregatinės specifikacijos verifikaviams ir validavimas
  Grafiškai sudarytą specifikaciją galima verifikuoti ir validuoti naudojant programinę įrangą.
- Agregatine specifikacija aprašytos sistemos imitavimas
  Pagal sudarytą specifikaciją programinė įranga leidžia atlikti sistemos imitacinį modeliavimą.

Programinėje įrangoje agregatą galima aprašyti vedlio lange įvedant nuosekliai visus 9 agregato aprašymo punktus, taip susikuriant agregato šabloną, kurį vėliau bus galima iš įrankių juostos įtraukti į diagramą. Kadangi kai kuriose diagramose reikia naudoti tuos pačius agregatus kelis kartus, tai programinė įranga leidžia susikurti agregatų šablonus, specifikuojant visą agregatą ir vėliau prireikus įtraukti paruoštus agregatus į diagramą taip sutaupant specifikacijos paruošimo laiką. Taip pat lengvai galima įtraukti ir sekantį tokį patį agregatą. Kanalai tarp agregatų sukuriame tiesiog pele ekrane brėžiant linijas nuo vieno agregato išėjimo iki kito agregato įėjimų.

Vaizduojant agregatą grafiškai, gali susidaryti situacija, kad jame yra aprašyta daug vidinių įvykių ir agregato grafinis vaizdas tampa gana gremėzdiškas, o tuo pačiu prarandamas vaizdumas. Šiai problemai spręsti grafinėje agregatinės specifikacijos sudarymo priemonėje numatyta funkcija paslėpti, t.y. nerodyti visų vidinių įvykių, o tik dalį jų ir pažymėti, kad jų yra daugiau daugtaškio ženklu.

Vartotojas naudodamasis programine įranga galės greičiau ir lengviau sudaryti agregatines specifikacijas, jas verifikuoti ir validuoti, ir taip pat atlikti sistemos imitavimo eksperimentus.

9.2.5. Literatūros sąrašas

[1] **F.Babich, L.Deotto**. Formal Methods for Specification and Analysis of Communication Protocols. 2002. *IEEE Communications Surveys*, 2002.

[2] **Q.Gao, R.Groz, G.v.Bochmann, J.Dargham, E.H.Htite**. Validation of Distributed Algorithms and Protocols. *ICNP-95,* 1995.

[3] **H.Pranevicius**. Aggregate approach for specification, validation, simulation and implementation of computer network protocols. *Lecture Notes in Computer Science*, 1991, No 502, Springer-Verlag, 433 – 477.

[4] **H. Pranevičius**. Formal specification and analysis of distributed systems. *Applications of AI to Production Engineering, Kaunas University of Technology Press*, 1997, 269 - 322.

[5] **H.Pranevičius**. Kompiuterių tinklų protokolų formalusis specifikavimas ir analize: agregatinis metodas. *Kaunas University of Technology Press*, 2003, 48 - 62.

Graphic notation of aggregates specification

PLA method can be used to formally specify systems. Systems specifications are written as text, what is exhaustive and understandable, but unfortunately are not visual, and user can't quickly acquaintance with the formalized system.

This article presents a way to represent aggregates specifications visually, which eases an acquaintance with specification. This article also presents a software tool to visually design formal specifications.

9.3. 3 PRIEDAS. Konferencijos „Informacinės technologijos 2005“pranešimo medžiaga

Intelektualus agregatinių specifikacijų redaktorius

Aivaras Bakanas, Šarunas Packevičius, Prof.habil.Dr Henrikas Pranevičius

Kauno Technologijos universitetas, Informatikos fakultetas Studentu g. 50, LT - 3031 Kaunas

Vienas iš budu aprašyti sistemas formaliais metodais yra naudojant PLA metodą. Sistema aprašoma matematinemis išraiškomis. Ju specifikacija yra gana išsami ir suprantama, bet nera labai vaizdi, reikia idemiau pasigilinti, norint suprasti specifikaciją.

Šiame pranešime pateikiamas budas atvaizduoti PLA specifikacijas grafiškai, diagramomis tuo leidžiant greičiau suprasti kas jose pateikta. Pranešime pristatoma programine priemone leidžianti vizualiai sudaryti PLA formalias specifikacijas, taip pat atlikti validavimą, verifikavimą bei sistemos imitavimo eksperimentus.

9.3.1. Įžanga

Sistemos formali specifikacija gali būti išreikšta PLA metodu. Specifikacija būna pateikta tekstu, matematinėmis išraiškomis. Tokia specifikacija nėra vaizdi, norint suprasti apie ką yra aprašyta sistema, tenka atidžiai įsigilinti į specifikaciją. Tenka daug laiko sugaišti bandant nustatyti sistemos dalis ir jų santykius, bendradarbiavimus. Kad palengvinti formalios agregatinės specifikacijos sudarymą ir jos peržiūrėjimą galima naudoti grafinius budus. Agregatines specifikacijos būtų vaizduojamos grafinėmis diagramomis. Taip pat jos būtų sudaromos grafiniu būdu. Turint galimybes vaizduoti specifikacijas grafiniu būdu, galima sukurti CASE priemones leidžiančias lengvai, kompiuteriu sudaryti norimas specifikacijas.

9.3.2. Programines priemones

Buvo kuriamos kelios CASE priemonės leidžiančios sudaryti agregatines specifikacijas grafiškai. Viena iš jų buvo pavadinta „Praxis“ vardu. Ji veikė tekstiniame režime ir schematiškai leisdavo pavaizduoti agregatus. Deja joje agregatinių specifikacijų vaizdavimas buvo gana skurdus, buvo galima matyti ekrane tik vieną agregatą vienu metu, ir matyti tik jo įėjimus ir išėjimus. Nebuvo galima peržiūrėti visos sudarytos agregatinės specifikacijos vienoje diagramoje. Pagal sudarytą specifikaciją, buvo generuojamas imituojamos sistemos programos kodas aprašytas „Estela“ kalba ir buvo galima atlikti sistemos verifikavimą, validavimą, o taip pat ir imitavimo eksperimentus.

Kita agregatinės specifikacijos sudarinėjimo priemonė yra mūsų kuriama „AgDraw“ . Kurioje jau grafiniu rėžiu galima sudarinėti specifikaciją. Galima apsirašyti agregatus, kurti jų šablonus, juos naudoti sudarant sistemos agregatinę specifikaciją. Taip pat sistema yra integruota su tos specifikacijos validavimo, verifikavimo ir pagal agregatinę specifikaciją aprašytos sistemos imitavimo moduliais.

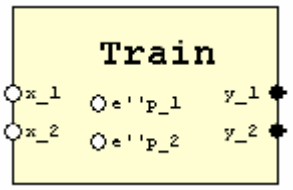
9.3.3. Specifikacijų grafinis vaizdavimas

Agregatinės specifikacijos vaizduojamos grafinėmis diagramomis, kas palengvina jų sudarymą. Agregatinės specifikacijos diagramose vaizduojami tik dviejų tipų elementai: agregatai ir ryšiai tarp jų.

Agregatas vaizduojamas:

- Stačiakampiu. Agregatą diagramoje atitinka stačiakampis su jame pažymėta kita informacija. Stačiakampis gali būti įvairaus dydžio.
- Agregato vardu stačiakampyje. Stačiakampyje vaizduojančiame agregatą, viršuje, viduryje užrašomas agregato vardas.
- Vidiniai įvykiai atvaizduojami apskritimu ir operacijos pavadinimu. • Kiekvienas agregate vykstantis vidinis įvykis atvaizduojamas apskritimu ir šalia jo užrašomas jo pavadinimas. Vienas įvykis užrašomas žemyn į apačią po kito įvykio. Užrašai išlygiuojami pagal dešinį agregato kraštą. Vidiniai įvykiai vaizduojami agregato stačiakampio viduje, pradedami žymėti iš karto po pavadinimo.
- Išėjimai žymimi tamsiu apskritimu. Agregato išėjimai pažymimi užtamsintu apskritimu ant agregato krašto, šalia užrašomas išėjimo signalas. Užrašas daromas agregato stačiakampio vidinėje dalyje.
- Įėjimai žymimi tuščiaviduriu apskritimu. Agregato įėjimai pažymimi tuščiaviduriu apritimu ant agregato krašto, šalia užrašomas įėjimo signalas. Užrašas daromas agregato stačiakampio vidinėje dalyje.

Agregato pavyzdys:



9.5 pav. Agregatas

Paveikslėlyje pateiktame agregate matome:

Pavadinimas: Train

Įėjimo signalų aibė: X = { x_1, x_2 };

Išėjimo signalų aibė: Y = { y_1, y_2 };

Vidiniai įvykai: e‘‘p_1, e‘‘p_2

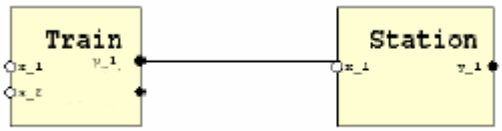
Tačiau grafiniame agregato vaizdavime neatsispindi visos jo savybės. Diagramoje nematome pavaizduotos išorinių įvykių aibės, bet juos galima numatyti pagal įeinančius signalus. Taip pat nematome valdančiųjų sekų, tolydinės ir diskrečios agregato būsenos, pradinių būsenų. Diagramoje taip pat neatsispindi perėjimo ir išėjimo operatoriai. Juos galima atvaizduoti atskirose diagramose vaizduojant „UML activity“ diagramas. Trūkstama informacija pateikiama priede prie specifikacijos. Diskrečias ir tolydžias agregato būsenos dedamąsias galima vaizduoti po vidinių operacijų, išorinius įvykius po vidinių įvykių kokiu nors kitokiu žymėjimu, pavyzdžiui stačiakampiais. Tačiau tai per daug apkrautų diagramą. Programinėje įrangoje būtų galima realizuoti paslėpimą arba dalinį laukų rodymą.

Kanalai tarp agregatų vaizduojami grafiškai diagramoje sujungiant agregatus. Tai yra daug patogiau negu naudojant lenteles, kuriose pažymėta, koks agregatas su kuriuo agregatu sujungtas. Kanalai tarp agregatų vaizduojami:

Sujungiamai agregato išėjimai su kitų agregatų įėjimais.

Jungiami vieno agregato išėjimai su kito agregato įėjimais linija (esant poreikiui sudarytai iš atkarpų). Linijos jungia agregato tuščiavidurius apskritimus su užpildytais. Linijos neina per agregatą vaizduojančio stačiakampio vidų. Kanalai taip pat gali išsišakoti ir tas pats kanalas išeidamas iš vieno agregato gali būti prijungtas prie kelių agregatų įėjimų.

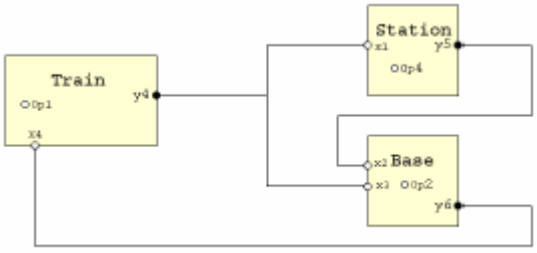
Kanalo pavyzdys:



9.6 pav. Sujungimo pavyzdys

Paveikslėlyje pateiktame agregatų sujungime vaizduojami sujungti kanalu du agregatai „Train“ ir „Station“, agregatas „Train“ išduoda išėjimo signalą „y_1“, agregatas „Station“ prima įėjimo signalą „x_1“.

Apjungus agregatus kanalais gauname agregatų diagramą. Diagramos pavyzdys:

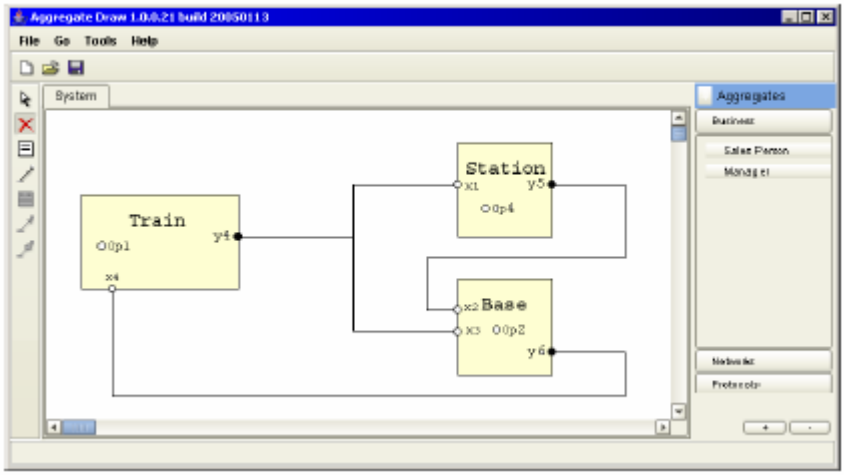


9.7 pav. Agregatų diagrama

Paveikslėlyje pavaizduota agregatų diagrama. Joje yra aprašyti 3 agregatai „Train“, „Station“, „Base“; jų įėjimo ir išėjimo signalai, kanalai jungiantys agregatus, tai pat pavaizduotas kanalas jungiantis iškarto tris agregatus.

9.3.4. Agregatų braižyklė

Agregatu braižyklė „AgDraw“ yra priemonė, skirta grafiškai sudaryti agregatines specifikacijas. Ji palengvina agregatinės specifikacijos sudarymą vartotojui. Programos vaizdas:

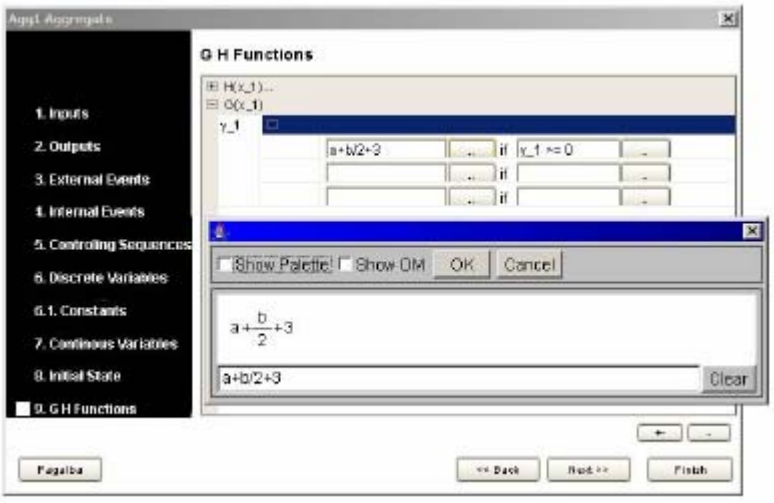


9.8 pav. Agregatų braižyklė „AgDraw“

Programinė įranga teikia vartotojui tokias funkcijas:

- Agregatinės specifikacijos sudarymas.
- Vartotojas grafiškai sudaro specifikaciją. Naudojat pelę ekrane paišomi agregatai, sudaromi ryšiai tarp jų.
- Agregatinės specifikacijos verifikavimas ir validavimas.
- Grafiškai sudarytą specifikaciją galima verifikuoti ir validuoti naudojant programinę įrangą.
- Agregatine specifikacija aprašytos sistemos imitavimas .
- Pagal sudarytą specifikaciją programinė įranga leidžia atlikti sistemos imitacini modeliavimą.
- Aprašytos sistemos matematinis modeliavimas.

Programinėje įrangoje agregatą galima aprašyti vedlio lange įvedant nuosekliai visus 9 agregato aprašymo punktus, taip susikuriant agregato šabloną, kurį vėliau bus galima iš įrankių juostos įtraukti į diagramą. Vedlio lango pavyzdys:

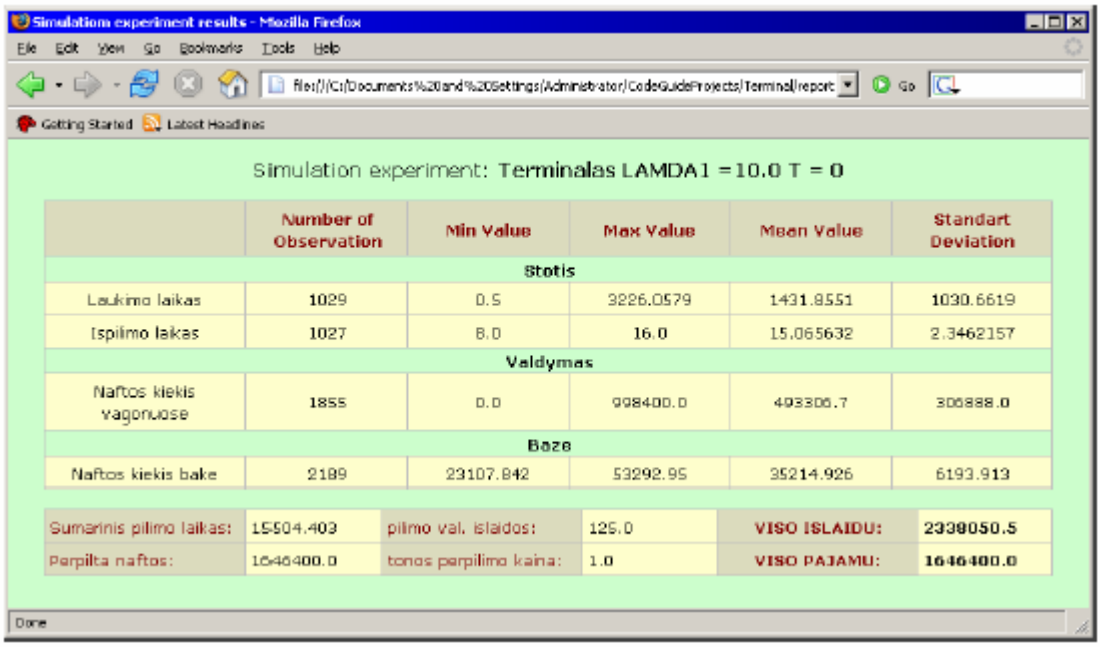


9.9 pav. Agregato specifikacijos detalizavimas (G ir H operatoriai)

Kadangi kai kuriose diagramose reikia naudoti tuos pačius agregatus kelis kartus, tai programinė įranga leidžia susikurti agregatų šablonus, specifikuojant visą agregatą ir vėliau prireikus įtraukti paruoštus agregatus į diagramą taip sutaupant specifikacijos paruošimo laiką. Taip pat lengvai galima įtraukti ir sekantį tokį patį agregatą. Kanalai tarp agregatų sukuriami tiesiog pele ekrane brėžiant linijas nuo vieno agregato išėjimo iki kitų agregatų įėjimų.

Vaizduojant agregatą grafiškai, gali susidaryti situacija, kad jame yra aprašyta daug vidinių įvykių ir agregato grafinis vaizdas tampa gana gremėzdiškas, o tuo pačiu prarandamas vaizdumas. Šiai problemai spręsti grafinėje agregatines specifikacijos sudarymo priemonėje numatyta funkcija paslėpti, t.y. nerodyti visų vidinių įvykių, o tik dalį jų ir pažymėti, kad jų yra daugiau daugtaškio ženklu.

Vartotojas naudodamasis programine įranga galės greičiau ir lengviau sudaryti agregatines specifikacijas, jas verifikuoti ir validuoti, ir taip pat atlikti sistemos imitavimo eksperimentus. Imitavimo eksperimentai generuoja ataskaitas, kurias galima išsaugoti archyvuoti, imitavimo eksperimento rezultatu pavyzdys:



9.10 pav. Imitavimo rezultatai

9.3.5. Panašūs produktai

Egzistuoja ir keletas analogiškų produktų skirtų sudaryti formalioms specifikacijoms, atlikti sistemų imitavimus, validavimus, verifikavimus. Vienas iš jų yra „SPIN“ sistema[6].

„SPIN“ sistema yra atviro kodo programinė įranga pagrinde skirta atlikti sistemų modelių verifikavimą. Sistema aprašoma „PROMELA“ kalba. “SPIN“ sistema gali dirbti trim režimais:

- Stimuliatorius, leidžiantis dažną prototipų sudarymą su atsitiktinai valdomu ar interaktyviu verifikavimu.
- Išsamus verifikatorius, galintis griežtai įrodyti griežtai apibrėžtų vartotojo teisingumo reikalavimų teisingumą.
- Teisingumo aproksimavimo sistema, kuri gali validuoti netgi labai didelių sistemų modelius atliekant maksimalų būsenų padengimą.

„SPIN“ sistema parašyta „ANSI C“ kalba ir yra pritaikyta „Unix“, „Linux“, „cygwin“, „Plan9“, „Inferno“, „Solaris“, „Mac“ ir „Windows“ sistemoms.

Žemiau pateiktoje lentelėje pateikiama „SPIN“ ir „AgDraw‘ sistemų palyginimas.

9.1 lentelė. „AgDraw“ ir „SPIN“ palyginimas

Kriterijus Sistema	AgDraw	SPIN
Imitavimas	Taip	Ne
Validavimas	Taip	Ne
Verifikavimas	Taip	Taip
Matematinis modeliavimas	Taip	Ne
Grafinis redaktorius	Taip	Ne
Vartotojo sąsaja	Taip	Taip
Platforma	Java	Unix, Linux, cygwin, Plan9, Inferno, Solaris, Mac ir Windows

9.3.6. Išvados

- Grafinių specifikacijų sudarymo privalumai:
  - Vaizdžiai matosi specifikacija.
  - Vartotojas gali greitai susipažinti su pateikta specifikacija, sistemoje egzistuojančiais elementais, jų savybėmis.
  - Lengvai skaitoma.
  - Lengvai sudaroma.
  - Programinėje įrangoje realizuoti įrankiai vartotojui leidžia greičiau ir paprasčiau sudaryti specifikaciją. Jie palengvina arba nuo vartotojo nuima bereikalingą, besikartojanti darbą.
- Grafinių specifikacijų sudarymo trūkumai:
  - Diagramoje neatvaizduojamos išėjimo ir įėjimo funkcijos (jas reikia aprašyti atskirai).
  - Nesimato pradinių būsenų (jas reikia aprašyti atskirai).
- Agregatinių specifikacijų sudarymo įrankis padės greičiau ir lengviau įvesti sistemos specifikacijas. Tai leis su minimaliomis pastangomis atlikti sistemų validavimą, verifikavimą, matematinį modeliavimą ir imitavimo eksperimentus.

9.3.7. Literatūros sąrašas

[1] **F.Babich, L.Deotto**. Formal Methods for Specification and Analysis of Communication Protocols. 2002. *IEEE Communications Surveys*, 2002.

[2] **Q.Gao, R.Groz, G.v.Bochmann, J.Dargham, E.H.Htite**. Validation of Distributed Algorithms and Protocols. *ICNP-95,* 1995.

[3] **H.Pranevicius**. Aggregate approach for specification, validation, simulation and implementation of computer network protocols. *Lecture Notes in Computer Science*, 1991, No 502, Springer-Verlag, 433 – 477.

[4] **H. Pranevičius**. Formal specification and analysis of distributed systems. *Applications of AI to Production Engineering, Kaunas University of Technology Press*, 1997, 269 - 322.

[5] **H.Pranevičius**. Kompiuteriu tinklu protokolu formalusis specifikavimas ir analize: agregatinis metodas. *Kaunas University of Technology Press*, 2003, 48 - 62.

[6] Spin - Formal Verification [žiureta 2005 m. sausis 13 d.], prieiga internete http://spinroot.com/spin/whatispin.html

9.3.8. Intelligent Aggregate Specifications Editor

PLA method can be used to formally specify systems. Systems specifications are written as text, what is exhaustive and understandable, but unfortunately are not visual, and user can't quickly acquaintance with the formalized system.

This article presents a way to represent aggregates specifications visually, which eases an acquaintance with specification. This article also presents a software tool to visually design formal specifications and perform verification, validation, mathematical modeling and imitation experiments of modeled system.

9.4. 4 PRIEDAS. Sugeneruoto JAVA kodo pavyzdžiai

Failas „MyAgSystem.java“:

```
package AdaptyvusTraktas;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;
public class MyAgSystem extends AgSystem
{
   private AdaptyvusTraktas adaptyvustraktas;


   public MyAgSystem()
   {
        super();
        adaptyvustraktas = new AdaptyvusTraktas(this);
   };
};
```

Failas „AdaptyvusTraktas.java“:

```
package Traktas;
import ktu.vik.jpranas.*;
import ktu.vik.jpranas.jsimulation.*;
public class AdaptyvusTraktas extends agModule
{
 private final static int IDLE=0;
 private final static int BUSY=1;
 private final static int Nfmax=100;
 private final static int Nfpmax=200;
 private agOpInt p_1;    //atejo failine sesija
 private agOpInt p_2;    //atejo paketas
 private agOpInt p_3;    //baigesi failines apdorojimas Nf kanalu grupeje
 private agOpInt p_4;    //baigesi failines apdorojimas Nfp kanalu grupeje
 private agOpInt p_5;    //baigesi paketo aptarnavimas

 //creates new AdaptyvusTraktas
 public AdaptyvusTraktas(agSystem sm)
 {
       super (sm, "AdaptyvusTraktas");
       p_1 = new ahInternalExtEvent(this,0.0, IDLE);
```

```
      p_2 = new ahInternalExtEvent(this,0.0, IDLE);
      p_3 = new ahInternalExtEvent(this,0.0, IDLE);
      p_4 = new ahInternalExtEvent(this,0.0, IDLE);
      p_5 = new ahInternalExtEvent(this,0.0, IDLE);
}

public void Initialize()
{
      p_1.Initialize();
      p_2.Initialize();
      p_3.Initialize();
      p_4.Initialize();
      p_5.Initialize();
      //TDO: add statistics counters
}
public void InternalEvent(agInternalEvent ev)
{
      switch (ev.getName()){
            case p_1:
                  p_1.Start();
                  if (Nf < Nfmax) {Nf := Nf + 1;}
                        else {if ((Nfp < Nfpmax)and(q < l1)) {Nfp := Nfp +1;}}
                  p_3 := Nf * miu1;
                  if (p_3 > 0) {p_3 := BUSY;} else {p_3:= IDLE;}
                  p_4 := Nfp * miu1;
                  if (p_4 > 0) {p_4 := BUSY;} else {p_4:= IDLE;}
                  p_5 := (Np + (Nfpmax-Nfp)+(Nfp + Nf)*ro)*miu2;
                  if (p_5 > 0) {p_5 := BUSY;} else {p_5:= IDLE;}
                  break;
            case p_2:
                  P_2.Start();
                  //.....
                  break;
            case p_3:
                  P_3.Start();
                  //.....
                  break;
            case p_4:
                  P_4.Start();
                  //.....
```

```
                  break;
            case p_5:
                  P_5.Start();
                  //.....
                  break;
      }
 }
 public void ExternalEvent(agExternalEvent ev)
 {
 }
}
```