KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Gediminas Šuklevičius

Automatizuotas formalių PLA specifikacijų sudarymas ir interaktyvusis redagavimas

Magistro darbas

Darbo vadovas
prof. habil. dr. H. Pranevičius

KAUNAS 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Automatizuotas formalių PLA specifikacijų sudarymas ir interaktyvusis redagavimas

Magistro darbas

Darbo vadovas
prof. habil. dr. H. Pranevičius

2008-05-26

Recenzentas
doc. dr. T. Blažauskas

2008-05-26

Atliko
IFM-2/2 gr. stud. G. Šuklevičius

2008-05-26

KAUNAS 2008

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLĖLIŲ SĄRAŠAS

AUTOMATED CREATION AND INTERACTIVE EDITING OF FORMAL PLA SPECIFICATIONS

SUMMARY

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 visually and quite easily write systems specifications using *PLA* formalization method. This article also presents a software tool to accomplish this task. That's graphical editor – a subsystem of a formal complex systems integrated analysis automatization system (*FSA*).

1. ĮVADAS

Kompiuteriai šiais laikais tampa dominuojančiais įrengimais valdant įvairias mašinas, nuo kosminių laivų ir atominių elektrinių iki automobilių ir namų apyvokos daiktų. Kuo toliau, tuo labiau reikalaujama, kad įvairios sistemos atliktų kuo sudėtingesnes užduotis, ir tai padarytų reikalaujant kuo mažiau pagalbos iš šalies. Daugelyje sistemų darbas be klaidų yra ypač svarbus, o bent menkiausios klaidos atsiradimas gali sąlygoti įrangos ar net žmogaus gyvybės praradimą.

Būtent šioje vietoje formalusis sistemos aprašymas gali labai smarkiai pasitarnauti užtikrinant, jog sistema veiktų saugiai ir tiksliai esant bet kokiomis aplinkybėmis. Formalios specifikacijos – tai matematinis programinės ar techninės įrangos aprašymas, kurį galima naudoti sistemos realizacijai [1]. Formalios specifikacijos aprašo ką sistema turi daryti, tačiau nebūtinai – kaip tai turi daryti. Pateikta formali sistemos specifikacija dar negarantuoja specifikacijos teisingumo, todėl prieš pradedant sistemos realizaciją reikia verifikuoti specifikaciją. Tam yra naudojami įvairūs korektiškumo tikrinimo metodai, padedantys atskleisti nesuderinamumus, neužbaigtumą ir kitus būsimos sistemos trūkumus. Sudėtingos sistemos analizuojamos dviem požiūriais: elgsenos ir funkcionalumo.

Elgsenos analizės metu tiriamos visos galimos sistemos trajektorijos, o tai leidžia patikrinti, ar sudaryta specifikacija teisinga. Teisingumas tikrinamas įvairiais validavimo bei verifikavimo (teisingumo tikrinimo ir patvirtinimo) metodais. [2]

Funkcionalumo analizės imitacinio modeliavimo priemonėmis metu vykdoma sukurtoji sistemos specifikacija. Specifikacijos teisingumo tikrinime išskiriamos dvi sistemos charakteristikų grupės: saugumo ir gyvybingumo. Saugumo charakteristika rodo, kad sistemoje neįvyksta iš anksto apibrėžtų nepageidaujamų įvykių. Tokių įvykių pavyzdžiai gali būti: statinės ir dinaminės aklavietės, kintamųjų neapibrėžtumas, invariantinės savybės ir pan. Gyvybingumo charakteristika rodo, kad sistemoje įvyksta tam tikrų pageidaujamų įvykių, pavyzdžiui, sistema pereina į pabaigos būseną. [2]

Formalių specifikacijų privalumas yra tai, kad specifikacijų keitimas yra pakankamai paprastas ir nereikalauja jokių papildomų lėšų. Visgi formalių specifikacijų sudarymas – ypač sudėtingoms sistemoms – yra komplikuotas procesas, todėl šis metodas dažnai naudojamas kuriant kritines sistemas, kadangi tokiose sistemose net mažiausias veikimo aprašymo netikslumas gali turėti sunkias pasekmes.

1.1. Dokumento paskirtis

Šis dokumentas apibrėžia problemas, kurios egzistuoja sudarinėjant agregatines specifikacijas ir atliekant jų analizę. Pateikiami egzistuojantys metodai šioms problemoms spręsti ir bei šių metodų realizacijos principai realioje sistemoje.

Šiame dokumente taip pat pateikiami eksperimento su realizuota agregatinių specifikacijų sudarinėjimo programine įranga rezultatai bei atlikto tyrimo išvados.

1.2. Santrauka

PLA (*Atkarpomis tiesinių agregatų*) formalizavimo metodas gali būti naudojamas sistemų formalių specifikacijų sudarinėjimui. Sistemų formalios specifikacijos gali būti užrašomos tekstiniu pavidalu, tačiau tai yra labai varginantis procesas, ko pasėkoje gautas rezultatas yra vaizdiniai neinformatyvus ir vartotojai negali lengvai perprasti sistemos specifikacijos.

Šiame dokumente pateikiamas būdas vizualiai ir pakankamai lengvai užrašyti sistemų formalias specifikacijas naudojant *PLA* formalizavimo metodą. Šiame dokumente taip pat pristatomas grafinis redaktorius – sudėtingų formalių specifikacijų integruotos analizės automatizavimo sistemos posistemė, skirta agregatinių specifikacijų kūrimui ir redagavimui.

2. ANALITINĖ DALIS

2.1. Įvadas

Programinės įrangos kūrimo stadija susideda iš standartinių kūrimo procesų: reikalavimų surinkimas, projektavimas, kūrimas, testavimas, diegimas ir palaikymas [5][8][9]. 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 [3]. 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 Flake [4]. 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 [6] pristatė įrankį, leidžiantį lygiagrečiai manipuliuoti reikalavimais šiose skirtingose notacijose. Vartotojo reikalavimai saugomi abstrakčiame modelyje ir vartotojas gali matyti per grafinę sąsają 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 [7] 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. 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. Tikslai ir uždaviniai

Rašant agregatinę specifikaciją, specifikuotojui palikta didelė erdvė savoms interpretacijoms. Todėl tokiu būdu paliekamos spragos klaidoms ir netikslumams. Šio darbo tikslai yra:

1. pateikti sudėtingų sistemų formalių specifikacijų integruotos analizės automatizavimo sistemos agregatinės specifikacijos interaktyvaus redagavimo posistemės (grafinio redaktoriaus) sukūrimo principus, metodus, kuriais remiantis redaktorius susistemina medžiagą, interaktyviai bendrauja su vartotoju ir nepalieka jam laisvės savaip interpretuoti agregatinės specifikacijos;
2. ištirti ir išanalizuoti grafinio redaktoriaus posistemės panaudojimo efektyvumo galimybes bei pateikti kokybės tobulinimo galimybes, pagrįsti ir realizuoti siūlomus patobulinimus.

2.3. Agregatinė specifikacija

2.3.1. Formalios specifikacijos apibrėžimas

Formali specifikacija – tai matematinis sistemos aprašymas. Ji naudojama specifikuoti sudėtingas, kritines sistemas, kurioms reikalingas ypatingas tikslumas ir klaidų tikimybė turi būti labai maža. Formali specifikacija tinka labai nedideliam sistemų ratui, daugelį sistemų tiesiog nenaudinga formaliai specifikuoti, dėl stipriai išaugančių kaštų. Formalios specifikacijos dėka sumažinamas klaidų skaičius pradiniuose programinės įrangos kūrimo etapuose, kurių taisymas kituose etapuose yra itin brangus. Formalus aprašymas neužtikrina aprašomos sistemos veikimo teisingumo. Tam naudojami įvairūs tikrinimo metodai, kurie padeda atskleisti sistemos trūkumus.

Paprastai formali specifikacija nenaudojama visam projektui, tačiau tikslinga naudoti atskiroms kritinėms dalims, kurios gali būti pažeidžiamos ar jų veikimas turi būti nepriekaištingas. Kaip pavyzdys būtų tinklinės sistemos. Jų tinklo sąsajos dalis yra labiausiai pažeidžiama, todėl formalus šios dalies specifikavimas padėtų ženkliai sumažinti pažeidžiamumą [10].

Pagrindiniai žingsniai kuriant sistemas naudojančias formalius metodus yra šie:

1. *Sistemos specifikavimas.* Formalus požiūris į sistemos reikalavimus leidžia vienareikšmišką pagrindinių sistemos savybių ir elgsenos aprašymą.
2. *Validavimas ir verifikavimas.* Iš aukšto lygio formalių specifikacijų kuriami prototipai, kurie gali būti simuliuojami ir tikrinamas jų korektiškumas.
3. *Funkcinis testavimas.* Pritaikymo ir operacinio suderinamumo testavimas gali būti atliktas naudojant optimalius testų rinkinius, kurie išplaukia iš sistemos elgsenos modeliavimo.
4. *Greitas prototipų kūrimas.* Dėka tikslaus atvaizdavimo galima iš karto iš formalių specifikacijų generuoti programos šaltinio kodą.
5. *Našumo testavimas.* Paprastai našumo matai gaunami atliekant simuliavimą arba teorinius paskaičiavimus, tačiau kiekybinė realizavimo veiksmingumo analizė gali būti atlikta suderinus formalius prototipus su našumo tikrinimo modeliais [11].

2.3.2. Formalių specifikacijų metodai

Formalių specifikacijų metodai skiriasi pagal tai, kokioje paradigmoje specifikacija iš tikrųjų yra [12]:

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.

2.3.3. Formalios specifikacijos savybės

Apskritai, formali specifikacija – tai sistemos savybių rinkinio išraiška kažkokioje formalioje kalboje, kažkokiame abstrakcijos lygyje [12]. Tikslesnis apibrėžimas galimas tik tada, kai tiksliai žinome:

- kas slepiasi po žodžiu „sistema“;
- kokios savybės yra įdomios;
- koks abstrakcijos lygis bus naudojamas;
- 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, visuma 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.

2.3.4. 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 [13][14].

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.

Grafinių specifikacijų sudarymo privalumai [13]:

- Vaizdžiai matoma specifikacija;
- Lengvai skaitoma specifikacija;
- Lengvai sudaroma specifikacija;
- Realizuoti programiniai įrankiai gali panaikinti pasikartojantį, bereikalingą darbą.

Grafinių specifikacijų sudarymo trūkumai [13]:

- Diagramose negalima atvaizduoti visos informacijos, nes tada jos būtų perkrautos (pvz., nesimato įėjimo ir išėjimo funkcijų);
- Diagramose nesimato pradinių būsenų (jas reikia aprašyti atskirai).

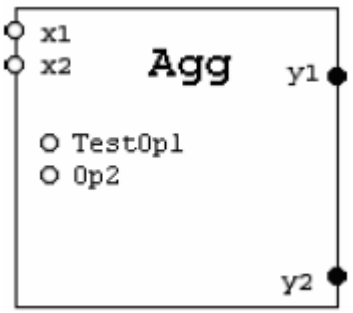
2.3.5. Agregatinės specifikacijos grafinis vaizdavimas

Agregatinės specifikacijos vaizduojamos grafinėmis diagramomis, kurios palengvina jų sudarymą [14]. 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.



Pav. 2. 1. Agregato pavyzdys

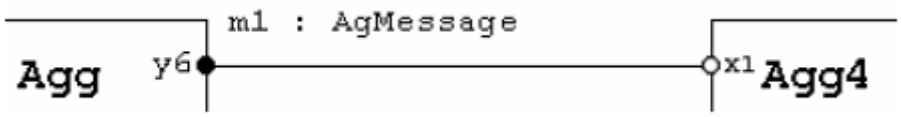
Paveikslėlyje 2.1. pavaizduotame agregate matome, kad jo pavadinimas yra „Agg“, įėjimo signalų aibė yra X={x1, x2}, išėjimo signalų aibė Y={y1, y2} ir vidiniai įvykiai yra „TestOp1“ ir „Op2“.

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* veiklos (angl., 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:

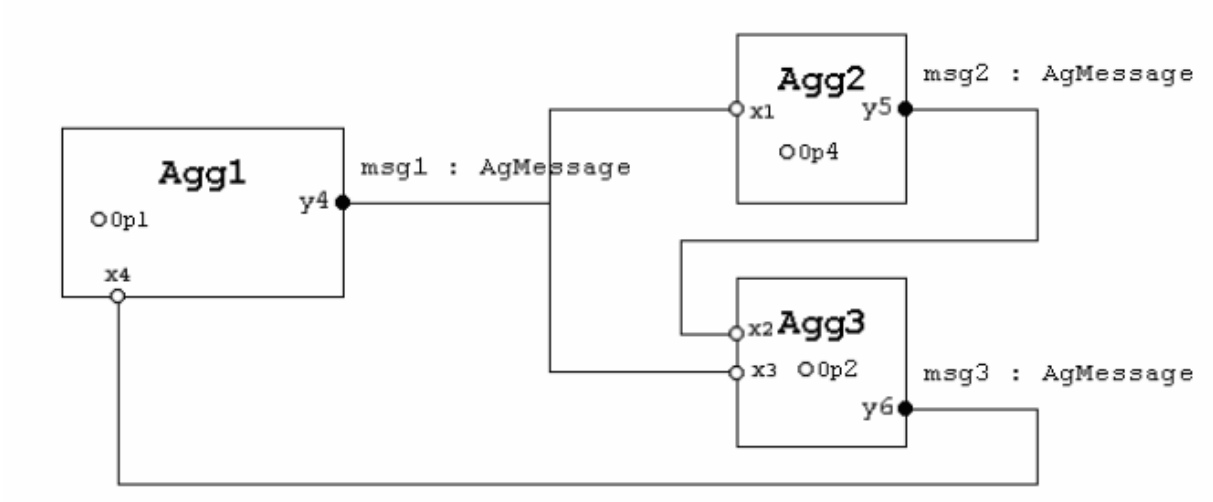
- Sujungiamai agregato išėjimai su kitų agregatų įėjimais - jungiami vieno agregato išėjimai su kito agregato įėjimais linija (esant poreikiui sudarytai iš atkarpų). Linijos jungia agregato tuščiavidurius apskritimus su užpildytais. Linijos neina per agregatą vaizduojančio stačiakampio vidų. Kanalai 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.



Pav. 2. 2. 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.

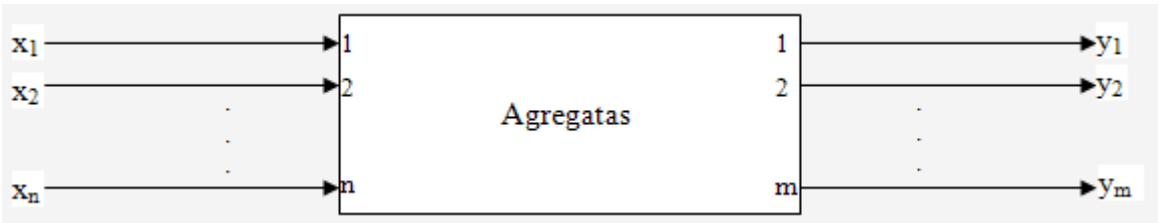


Pav. 2. 3. Agregatų diagrama

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.

2.4. PLA formalizavimo kalbos aprašymas

Aprašinėjant sistemą *PLA* kalba, pirmiausiai sistema suskaidoma į atkarpomis tiesinius agregatus. Atkarpomis tiesiniai agregatai priklauso automatų modelių klasei. Kaip ir automatas, atkarpomis tiesinis agregatas aprašomas nurodant būsenų aibė *Z*, įėjimo signalų aibė *Y* bei perėjimo operatorių (atvaizdavimą) *H* ir išėjimo operatorių *G*. Tačiau agregatas turi nemažai ypatybių, skiriančių šį modelį nuo automatinių modelių. Paveikslėlyje 2.4. pavaizduotas *PLA* agregatas su įėjimų ir išėjimų aibėmis.



Pav. 2. 4. PLA agregatas

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

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

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

$z_v(t)$ - tolydi būsenos dedamoji.

Bendru atveju

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

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

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

Kai nėra įėjimo signalų, agregato būsena kinta taip:

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

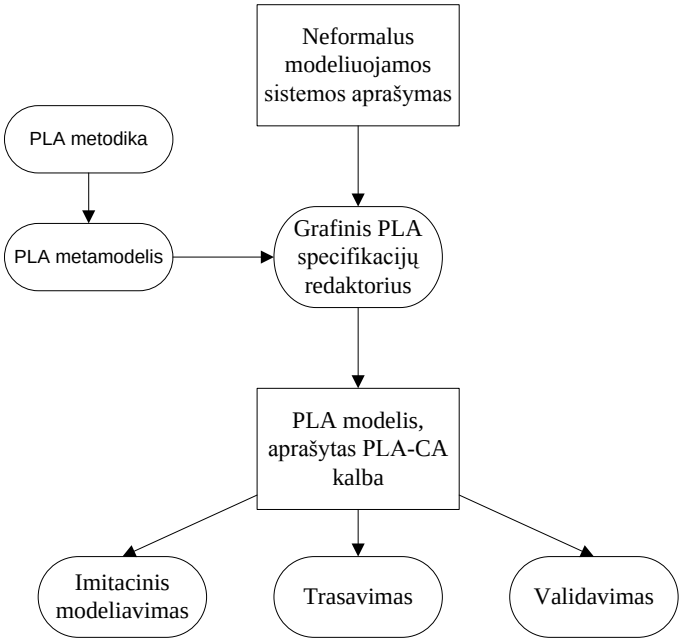
čia $\alpha_v = (\alpha_{v1}, \alpha_{v2}, ..., \alpha_{vk})$ - pastovusis vektorius.

Agregato būsena gali pakisti tik dviem atvejais: kai į agregatą siunčiamas įėjimo signalas arba kai viena iš tolydžiųjų dedamosios koordinačių įgyja tam tikrą reikšmę [15][16].

2.5. 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 pateiktas 2.5. paveikslėlyje:



Pav. 2. 5. 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 imitacinis modeliavimas, trasavimas ar specifikacijų validavimas.

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

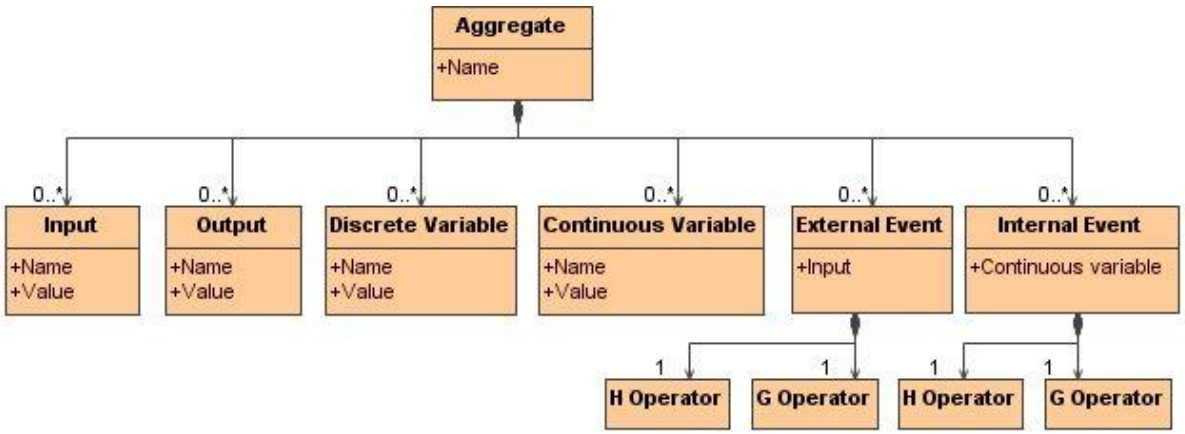
1. Įėjimų signalų aibė *X*;
2. Išėjimo signalų aibė *Y*;
3. Išorinių įvykių aibė *E′*;
4. Vidinių įvykių aibė *E″*;
5. Valdymo sekos;
6. Diskrečiosios agregato būsenų dedamosios;
7. Tolydžiosios agregato būsenų dedamosios;
8. Pradinė agregato būsena;
9. Perėjimų ir išėjimų operatoriai.

2.6. Agregato metamodelis

Agregatas susideda iš:

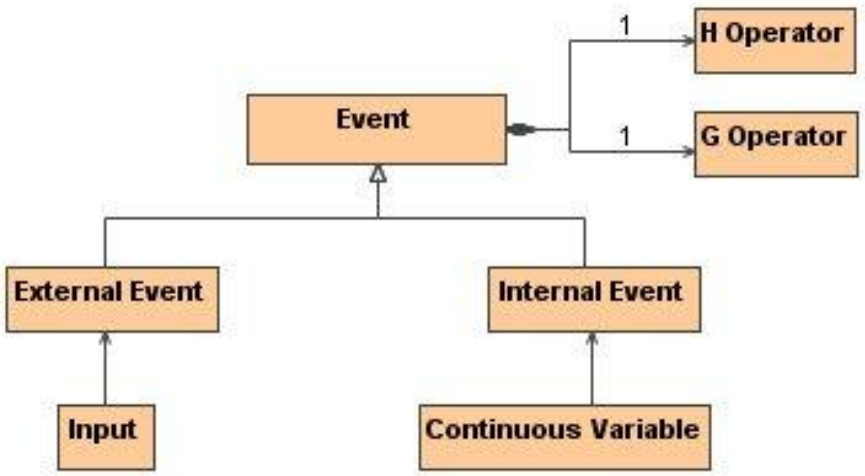
1. Įėjimų aibės, kurioje gali būti neribotas kiekis įėjimų;
2. Išėjimo aibės, kurio gali būti neribotas kiekis išėjimų;
3. Diskrečiųjų kintamųjų aibės, kuriai gali priklausyti neribotas kiekis diskrečiųjų kintamųjų. Diskretūs kintamieji naudojami aprašyti agregato būsenos dedamąsias.
4. Tolydžiųjų kintamųjų aibės, kuriai gali priklausyti neribotas kiekis tolydžiųjų kintamųjų. Tolydūs kintamieji yra naudojami aprašyti laiko momentus kuriais įvyksta vidiniai agregato įvykiai pasibaigus operacijoms;
5. Vidinių įvykių aibės, kurioje aprašomi įvykiai, vykstantys agregato viduje;
6. Išorinių įvykių aibės, kurioje aprašomi įvykiai, vykstantys agregatui įėjime gavus signalą.

Paveikslėlyje 2.6. pavaizduotas agregato metamodelis *UML* notacijoje.



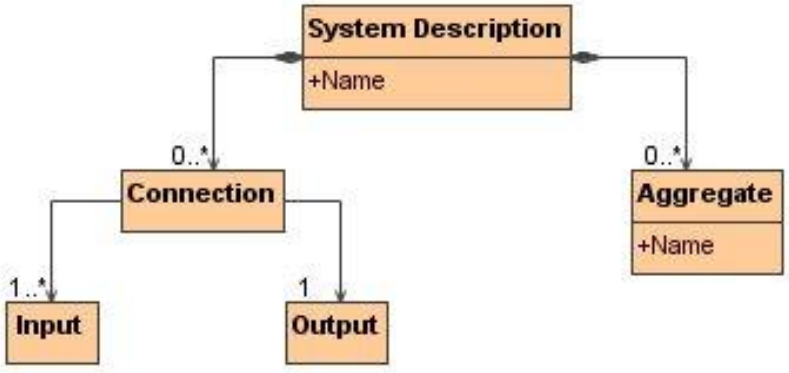
Pav. 2. 6. Agregato metamodelis UML notacijoje

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. H ir G operatoriai *UML* notacijoje pavaizduoti 2.7 paveikslėlyje.



Pav. 2. 7. H ir G Operatoriai UML notacijoje

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 2.8. paveikslėlyje.



Pav. 2. 8. Sistemos metamodelis UML notacijoje

2.7. Formalių specifikacijų integruotos analizės sprendimai pasaulyje

Šiuo metu rinkoje egzistuoja keletas sistemų, atliekančių formalių specifikacijų integruotą analizę. Labiausiai paplitusios ir plačiausiai naudojamos yra šios sistemos: *SPIN, VIS, SMV, NuSMV, Uppaal, PRANAS-2, VALSYS* [17][18].

2.7.1. SPIN sistema

Specifikavimo, imitacinio modeliavimo ir validavimo sistema *SPIN* yra plačiai naudojama kompiuterių protokolams tirti. SPIN buvo sukurtas kompanijos „Bell Labs“ dar 1980 metais. 2002 metais įrankis gavo prestižinį „System Software Award“ apdovanojimą. *SPIN* naudoja *C* specifikacijos notaciją, kuri padidina jo pritaikomumą pirmoje kūrimo stadijoje [19]. *SPIN* leidžia simuliavimą ir validavimą *PROMELA* kalba parašytai specifikacijai.

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

Sistemai, specifikuotai *PROMELA* kalboje, *SPIN* gali atlikti imitacinį modeliavimą tos sistemos vykdymo, arba jis gali generuoti programą *C* kalba, kuri vykdo sistemos savybių teisingumo patikrinimą realiu laiku. Tikrintojas taipogi gali būti naudojamas tikrinti sistemos kintamųjų teisingumui, jis gali rasti nereikalingus ciklus, gali patikrinti sekančio laiko momento loginių formuluočių teisingumą. Tikrinimas turi būti naudingas ir naudoti minimalų atminties kiekį. Išsamus tikrinimas gali su matematine tikimybe nustatyti ar aprašytas sistemos elgesys yra be klaidų. Labai didelės tikrinimo problemos, kurios negali būti išspręstos su esama kompiuterine technika, gali būti bandomos spręsti su ekonomiška „būsenos saugojimo bitu“ technika. Šiuo metodu būsenos užima vieta suskirstoma į mažą bitų skaičių pasiekiamoje sistemos būsenoje, su minimaliu pašaliniu efektu [19].

SPIN programos kalba *PROMELA* susideda iš procesų, pranešimų kanalų bei kintamųjų. Procesai yra globalūs objektai kurie atvaizduoja paskirstytos sistemos esybes. Pranešimų kanalai ir kintamieji gali būti aprašyti kaip globalūs arba pačiame procese. Procesai aprašo elgseną, kanalai ir globalūs kintamieji aprašo aplinką, kurioje procesas veikia. [19][20].

SPIN gali būti naudojamas kaip:

- Imitatorius, leidžiantis greitą analizę naudojant atsitiktinę, valdomą arba dialoginę simuliaciją.

- Nuodugnus verifikatorius, galintis kruopščiai įrodyti vartotojo aprašytų reikalavimų specifikacijos teisėtumą.
- Apytikrio skaičiavimo sistema, leidžianti validuoti net labai didelius sistemų modelius maksimaliai išnaudojant būsenos vietą [19].

2.7.2. VIS sistema

VIS (*Verification Interacting with Synthesis*) yra sistema, leidžianti atlikti sistemų, aprašytų formaliomis specifikacijomis, verifikavimą, sintezę ir simuliaciją. Pagrindinis reikalavimas aprašomai sistemai – ji turi turėti baigtinį skaičių būsenų.

VIS gali susintetinti baigtinio būsenų skaičiaus sistemą ir/arba verifikuoti sistemos formaliai aprašytų savybių teisingumą.

Pagrindinės *VIS* sistemos atliekamos funkcijos:

- Loginių ciklų simuliavimas ir aklaviečių bei uždarumo tikrinimas;
- Sudėtingų ir nuoseklių ciklų verifikavimas [21].

2.7.3. SMV sistema

SMV yra formaliai aprašytų sistemų validavimo ir verifikavimo įrankis. Pasinaudojant šiuo įrankiu galima verifikuoti ir validuoti sistemas, aprašytas išplėsta *SMV* formalizavimo kalba. *SMV* turi lengvai naudojamą ir nesunkiai įsimenamą grafinę vartotojo sąsają, bei leidžia dalinį aprašytos sistemos trasavimą [22].

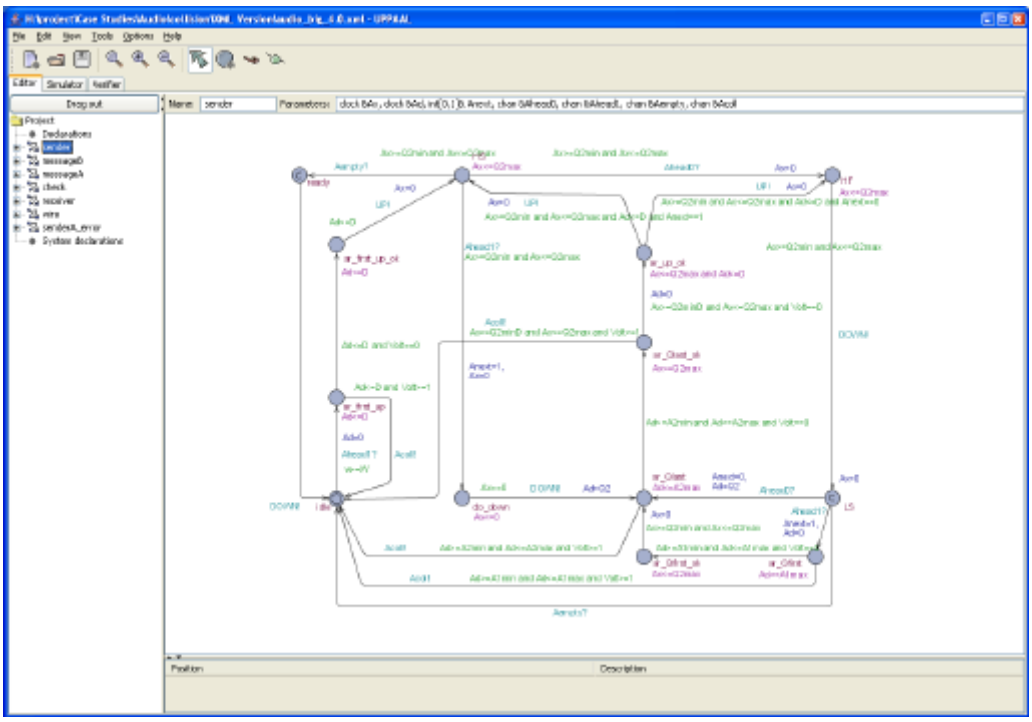
2.7.4. NuSMV sistema

NuSMV yra patobulintas ir pagerintas *SMV* įrankis. *NuSMV* buvo sukurtas kaip atviros architektūros įrankis formaliai aprašytų sistemų verifikavimui, validavimui. Taip pat šį įrankį galima naudoti kaip branduolį kuriant savo verifikavimo ir validavimo sistemas [23].

2.7.5. Uppaal sistema

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

Uppaal susideda iš trijų pagrindinių dalių: kalbos aprašymo, simuliatoriaus ir modelio tikrintojo. Aprašymo kalba yra nedeterminuota kalba su paprastais duomenų tipais (sveiki skaičiai, masyvai ir kt.). Ji naudojama formaliai aprašant tiriamą sistemą. *Uppaal* sistemos specifikacijų grafinis redaktorius pavaizduotas 2.9. paveikslėlyje.



Pav. 2. 9. UPPAAL specifikacijų redaktorius

Imitatorius leidžia tikrinti sistemos veikimą, tikrinant visus galimus veikimo kelius. Imitacinis modeliavimas gali būti atliekamas jau ankstyvojo modeliavimo etapuose ir tokiu būdu suteikia galimybę aptikti ir ištaisyti galimas klaidas pačioje pradžioje [24].

Validatorius gali tikrinti ar sistema nepatenka į aklavietes, ar nesusidaro amžinų ciklų ir pan.

Pagrindinės *UPPAAL* savybės:

- Grafinis redaktorius sistemos aprašymui įvesti.
- Grafinis imitatorius vizualiai pateikia galimas sistemos veikimo kryptis. Taip pat gali būti vizualizuojama trasavimo informacija gauta iš validatoriaus.
- Reikalavimų specifikacijos redaktorius.
- Validatorius sistemos gyvybingumui tikrinti.
- Trasavimo informacijos generavimas, su galimybe grafiškai atvaizduoti.

2.7.6. Pranas-2 sistema

Pranas-2 – tai KTU Verslo informatikos katedroje sukurta protokolų modeliavimo ir validavimo sistema. Ši sistema sukurta agregatinio metodo pagrindu ir ji naudojama *ESTELLE/Ag* kalbą. *ESTELLE/Ag* kalba yra *ESTELLE* ISO standarto modifikacija, specialiai pritaikyta *PLA* formalizavimui. Naudojant tokią kalbos modifikaciją galima sukurti formalias specifikacijas, kurios tinka validavimui ir imitaciniam modeliavimui.

Pranas-2 analizės sistemą sudaro tokios dalys:

- Specifikacijų redaktorius;
- Validavimo posistemis;

- Imitacinio modeliavimo posistemis.

Validavimo posistemis leidžia sugeneruoti pasiekiamumo grafą ir atlikti tokius patikrinimus:

- Pasiekiamumą – kokiu būdu iš pradinės būsenos galima pasiekti galinę būseną;
- Būsenų koordinačių apribojimus – galima nustatyti, ar būsenos koordinatės neišeina už iš anksto nustatytų ribų;
- Tolydžiųjų koordinačių reikalingumą – reikia nustatyti, ar specifikacijoje nėra aprašytų tolydžiųjų dedamųjų, kurios niekada negeneruoja vidinių įvykių;
- Aklaviečių paiešką – galima nustatyti, ar yra būsenų, iš kurių niekur neišeinama;
- Ciklų paiešką – galima nustatyti, ar nepatenkama į uždarą ciklą, kuriame kartojasi tam tikra įvykių seka [25].

2.7.7. VALSYS sistema

VALSYS – tai KTU Verslo informatikos katedros magistranto V. Astrovo sukurta agregatinių specifikacijų saugumo ir gyvybingumo tyrimo sistema. Sistemos paskirtis yra pasinaudojant pagal agregatines specifikacijas gautais tiriamos sistemos būsenų grafais ištirti jos veikimo korektiškumą bei specifikacijos pilnumą. Tai automatinio tikrinimo priemonė, turinti užtikrinti, jog kuriama agregatinėmis specifikacijomis aprašyta sistema nepateks į aklavietes, uždarus ciklus ar iš anksto nenumatytas, neapibrėžtas būsenas. Tai įgyvendinama analizuojant būsenų grafą ir ieškant būsenų, netenkinančių specifikacijos.

Pagrindinės šios sistemos funkcijos:

- Aklaviečių paieška;
- Uždarų ciklų paieška;
- Būsenos koordinačių ribų patikrinimas;
- Invarianto tikrinimas;
- Būsenų pasiekiamu tikrinimas [26].

2.7.8. AgDraw sistema

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

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 [14].

2.8. Įgyvendinimo problemos

Pilnai išanalizavus *PLA* kalbą ir egzistuojančius sprendimus, galima teigti, kad pagrindinės sudėtingų sistemų formalių specifikacijų integruotos analizės automatizavimo sistemos bei grafinio redaktoriaus posistemės įgyvendinimo problemos yra šios:

1. Analizuojamos sistemos specifikacijos įvedimas;
2. Analizuojamos sistemos objektinio modelio sukūrimas;
3. Tolydinių dydžių interpretavimas;
4. Vartotojo sąsajos aiškumas

2.8.1. Analizuojamos sistemos specifikacijos įvedimas

Integruotos analizės sistemai reikalingas formalios specifikacijos įvedimas. Jis gali būti įgyvendinamas, pateikiant specifikaciją, paruoštą *XML* formatu. *XML* formatas pasirinktas dėl griežtos notacijos ir palaikomumo tarp įvairių sistemų. Duomenys pateikti tokiu formatu gali būti efektyviai pervedami į objektinį modelį. Pastarasis bus naudojamas sistemoje, kaip pagrindinis duomenų šaltinis. Jo pagrindu bus realizuojamos kitos posistemės kaip imitacinis modeliavimas, trasavimas ir validavimas.

Nuskaitomi duomenys turi būti teisingai aprašyti *XML* pagalba ir suformuotas objektinis modelis. Šioje vietoje gali būti susiduriama su eile problemų, kurios atsiranda dėl sudėtingų sistemų specifikacijų apimties. Tai nėra lengvas darbas, be to didelė klaidų tikimybė. Įvedant specifikaciją rankiniu būdu *XML* formate sudėtinga aprašyti matematinius reiškinius. Tai turi būti iš anksto tiksliai specifikuota ir vienareikšmiškai interpretuojama. Aprašant matematinius reiškinius minėtu būdu atskiri matematiniai reiškiniai ir operacijos gali būti pernelyg sudėtingai užrašomi, o specifikacijos apimtis išaugti. Įvedant formalią specifikaciją kintamųjų žymėjimas turi būti supaprastintas, nes tokie žymėjimai, kaip esamo ir sekančio laiko momento, yra pernelyg sudėtingi ir neužrašomi be specialių redaktorių.

2.8.2. Analizuojamos sistemos objektinio modelio sukūrimas

Analizuojamą sistemą įvedus į kompiuterį, reikia išsaugoti taip, kad bet kuriuo laiko momentu būtų galima gauti einamąją sistemos būseną, taip pat būtų galima nesunkiai

apskaičiuoti sekančią sistemos būseną. Tam geriausia tiktų naudoti šiuolaikinę objektiškai orientuotą programavimo kalbą, pvz., *JAVA, C/C++, C#.*

2.8.3. Tolydinių dydžių interpretavimas

Kiekviena analizuojama sistema, aprašyta *PLA-CA* formalizavimo kalba, naudoja tolydžiuosius kintamuosius. Deja, bet kompiuteris tolydžiųjų kintamųjų taip lengvai nesupranta, todėl teks naudoti tam tikrus reikšmių rėžius. Jei aprašant tolydžius kintamuosius, naudojami palyginimai su begalybe, tai paprastai aprašyti kompiuteriui suprantamam formate neįmanoma, todėl vietoj begalybės bus naudojamas sutartinis dydis.

2.8.4. Vartotojo sąsajos aiškumas

Sudėtingų sistemų formalių specifikacijų integruotos analizės sistema dirba su dideliais kiekiais įvairių duomenų, tokių kaip pasiekiamumo grafas, informacija apie kiekvieną sistemos būseną ir kt. Kad vartotojas galėtų nesunkiai orientuotis pateiktoje informacijoje, reikia, kad sistemos vartotojo sąsaja būtų neperpildyta, bet tuo pačiu pakankamai informatyvi ir lengva naudoti.

Vartotojo sąsaja kiekvienoje posistemėje turi padėti vartotojui teisingai interpretuoti sistemos veikimo rezultatų duomenis, kad vartotojas galėtų juos analizuoti ir priimti sprendimus. Tokiu būdu užtikrinant formalios specifikacijos teisingumo patikrinimą.

Vartotojo sąsajoje turi atsispindėti vykdymo metu esanti objektinio modelio būsena, tarpiniai rezultatai ir sistemos veikimo kryptis. Taip pat analogiškose sistemose naudojamas grafinis rezultatų pateikimas.

3. PROJEKTINĖ DALIS

3.1. Sistemos paskirtis

Formalios specifikacijos – tai matematinis programinės ar techninės įrangos aprašymas, kurį galima naudoti sistemos realizacijai. Formalios specifikacijos yra naudojamos kaip vienas iš būdų aprašyti vartotojų reikalavimams.

Formalių specifikacijų sudarymas nėra paprastas procesas, nes ypač sunku sudarinėjant specifikaciją patikrinti jos teisingumą.

Specifikacijų sudarytojas neturi galimybių patikrinti, ar sudaryta sistemos specifikacija yra teisinga, dar sunkiau įrodyti specifikacijos teisingumą užsakovams ar kitiems suinteresuotiems asmenims.

Projekto tikslas yra sukurti automatizuotą sudėtingų sistemų formalių specifikacijų integruotos analizės automatizavimo sistemą. Sistema leis ne tik patikrinti specifikacijos teisingumą atliekant sistemos validavimą, verifikavimą, imitacinį modeliavimą ar pažingsninį imitavimą, tačiau kartu leis sudarinėti ir interaktyviai redaguoti pačias formalias specifikacijas.

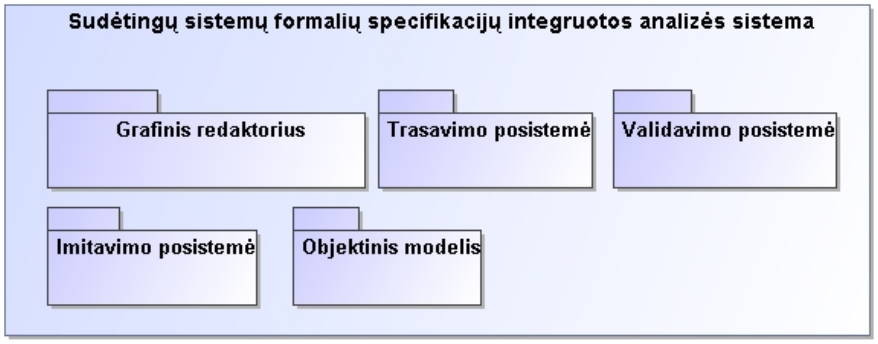
3.1.1. Projekto kūrimo pagrindas

Analogiškos sistemos, t.y. tokios, kuri kaip įėjimo kalbą naudoja *PLA* (*Piece Linear Aggregates*) nėra žinoma. Poreikis kurti sistemą atsirado tada, kai prireikė sistemos, kuri leistų nustatyti formalios specifikacijos, aprašytos *PLA* kalba, teisingumą. Sukūrus sistemą būtų išplėstas formalių specifikacijų analizės mechanizmas ir tai leistų žymiai efektyviau sudarinėti formalius sistemų aprašus.

3.1.2. Sistemos tikslai

Darbo tikslas yra sukurti sudėtingų formalių sistemų modelių kūrimo automatizavimo sistemą. Kuriama sistema yra grupinis projektas, kuriam turi būti sukurtos formalios specifikacijos transformavimo į objektinį modelį, grafinio redaktoriaus, imitacinio modeliavimo, trasavimo bei validavimo posistemės. Sistemos įėjimo kalba yra *PLA* (*Piece Linear Aggregates*). Formalios specifikacijos transformavimo į objektinį modelį posistemė transformuoja formalų sistemos aprašą į sistemai suprantamą formą. Grafinio redaktoriaus posistemės paskirtis yra formalių specifikacijų sudarymas ir redagavimas. Validavimo posistemės paskirtis yra panaudojant formalų sistemos aprašą *PLA* įėjimo kalba ištirti aprašo korektiškumą, bei formalios specifikacijos pilnumą. Imitacinio modeliavimo posistemė leidžia imituoti sistemos veikimą, peržiūrėti ir analizuoti galutinius rezultatus. Trasavimo posistemė leidžia imituoti sistemos veikimą pažingsniui, bet kuriuo metu nurodant sistemos parametrus.

3.2. Sistemos sudėtis



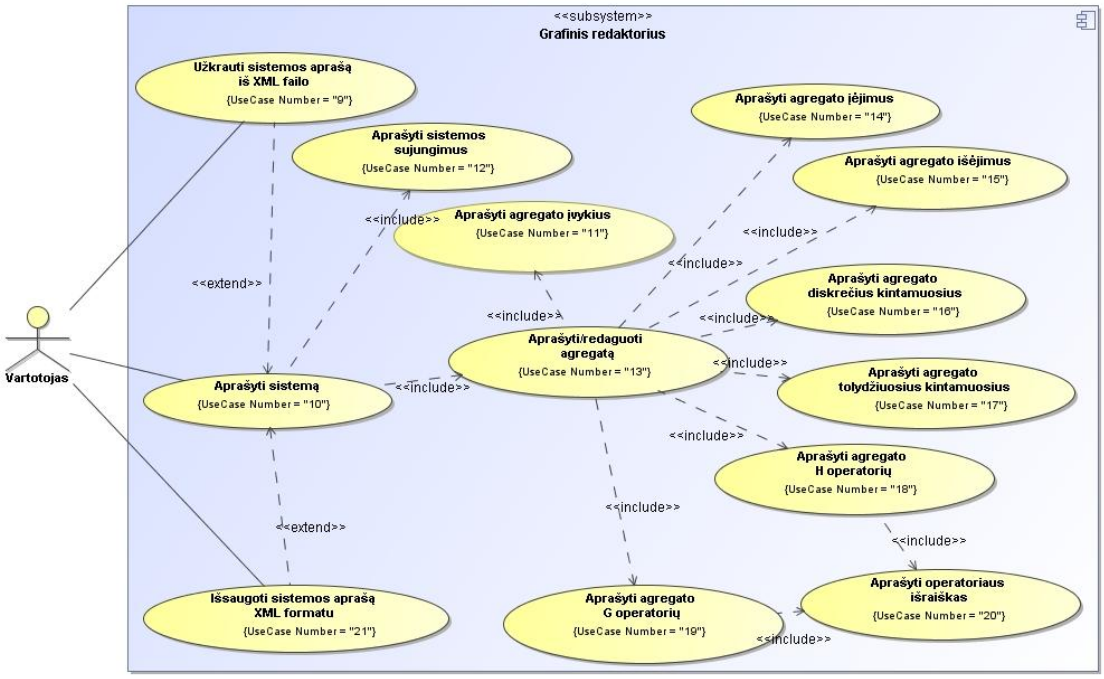
Pav. 3. 1. Sistemos sudėtis

Sudėtingų sistemų formalių specifikacijų integruotos analizės sistemos sudėtis pavaizduota 3.1. paveikslėlyje. Sistema susideda iš penkių pagrindinių dalių:

- Grafinis redaktorius – sistemos dalis, skirta analizuojamų sistemų aprašų kūrimui, redagavimui ir saugojimui *XML* formatu.
- Objektinis modelis – sistemos dalis, skirta analizuojamos sistemos aprašo saugojimui kompiuterio atmintyje darbo metu.
- Imitavimo posistemė – sistemos dalis, skirta analizuojamos sistemos veikimo imitavimui.
- Trasavimo posistemė – sistemos dalis, skirta analizuojamos sistemos trasavimui.
- Validavimo posistemė – sistemos dalis, skirta analizuojamos sistemos aprašo validavimui.

3.3. Panaudojimo atvejai

Grafinio redaktoriaus posistemės panaudojimo atvejai pavaizduoti 3.2. paveikslėlyje.



Pav. 3. 2. Grafinio redaktoriaus panaudojimo atvejai

Trumpas panaudojimo atvejų aprašymas:

1. Užkrauti sistemos aprašą iš *XML* failo - Apima procesą, kurio metu sistemos vartotojas atidaro analizuojamos sistemos aprašą, saugomą kompiuterio kietajame diske *XML* formatu;
2. Aprašyti sistemą - apima procesą, kurio metu sistemos vartotojas pilnai aprašo analizuojamą sudėtingą sistemą *PLA* formalizavimo kalba;
3. Aprašyti agregato įvykius - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sudėtingos sistemos agregato įvykius;
4. Aprašyti sistemos sujungimus - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sistemos sujungimus – ryšius tarp konkrečių agregatų įėjimų ir išėjimų;
5. Aprašyti/redaguoti agregatą - apima procesą, kurio metu sistemos vartotojas aprašo arba redaguoja analizuojamos sistemos agregatus;
6. Aprašyti agregato įėjimus - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sudėtingos sistemos agregato įėjimus;
7. Aprašyti agregato išėjimus - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sudėtingos sistemos agregato išėjimus;
8. Aprašyti agregato diskrečius kintamuosius - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sudėtingos sistemos agregato diskrečius kintamuosius;
9. Aprašyti agregato tolydžius kintamuosius - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sudėtingos sistemos agregato tolydžiuosius kintamuosius;
10. Aprašyti agregato H operatorių - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sudėtingos sistemos agregato H operatorius;
11. Aprašyti agregato G operatorių - apima procesą, kurio metu sistemos vartotojas aprašo analizuojamos sudėtingos sistemos agregato G operatorius;
12. Aprašyti operatoriaus išraiškas - apima procesą, kurio metu sistemos vartotojas aprašo konkretaus operatoriaus matematines išraiškas;
13. Išsaugoti sistemos aprašą *XML* formatu - apima procesą, kurio metu sistemos vartotojas išsaugo sistemos aprašą kompiuterio kietajame diske *XML* formatu.

3.4. Reikalavimai sistemai

Pagrindiniai funkciniai reikalavimai grafinio redaktoriaus posistemei:

- Sistemos formalaus aprašo saugojimas *XML* formatu kompiuterio kietajame diske;
- Sistemos formalaus aprašo užkrovimas iš *XML* failų;
- Sistemos sujungimų aprašymas ir redagavimas;

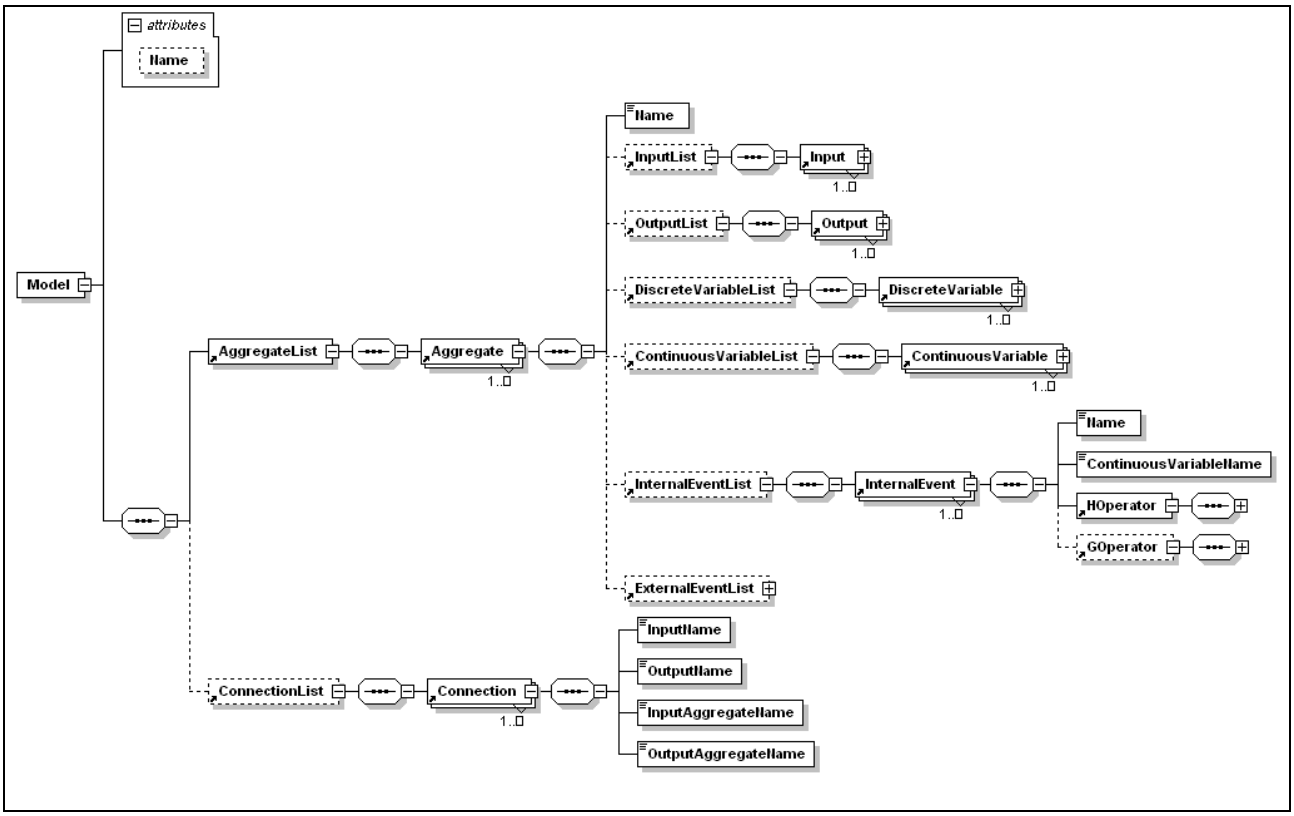
- Sistemos agregatų aprašymas ir redagavimas;
- Agregato operatorių išraiškų aprašymas ir redagavimas;
- Sistemos aprašo validavimas.

Pagrindiniai nefunkciniai reikalavimai grafinio redaktoriaus posistemei:

- Paprastas panaudojamumas, nereikalaujantis specialaus vartotojų apmokymo;
- Lengvas pernešamumas;
- Išplečiamumas: esant poreikiui, posistemė turi būti papildyta naujomis funkcijomis.

Pagrindiniai reikalavimai duomenims:

- Sistemos duomenys bus saugomi *XML* failuose. Paveikslėlyje 3.3. pateikta pradinė XML schema, kurioje saugomi sistemos duomenys.

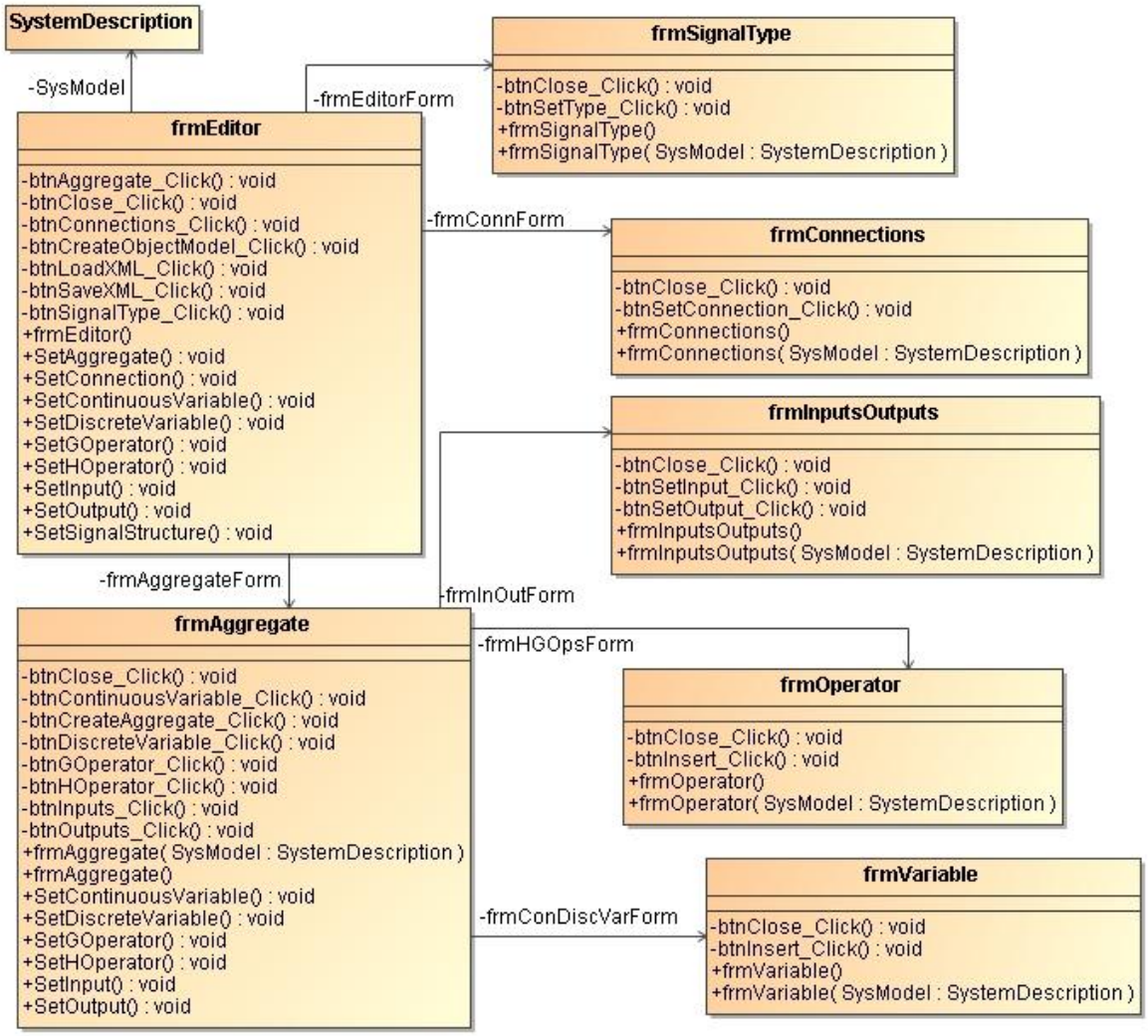


Pav. 3. 3. Sistemos duomenų struktūra

3.5. Grafinio redaktoriaus detali specifikacija

3.5.1. Klasių diagrama

Grafinis redaktorius skirtas analizuojamos sistemos aprašų sudarymui, redagavimui ir išsaugojimui *XML* failuose. Posistemę sudaro septynios *Windows* formos, kurių kiekviena atsakinga už konkrečios aprašomos sistemos informacijos įvedimą ir redagavimą. Grafinio redaktoriaus klasių diagrama pateikta 3.4. paveikslėlyje.

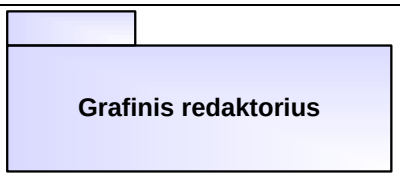


Pav. 3. 4. Grafinio redaktoriaus klasių diagrama

3.5.2. Detalus aprašymas

Grafinio redaktoriaus posistemės klasių diagramos detalus aprašymas pateiktas 3.1. lentelėje.

Lentelė 3. 1. Grafinio redaktoriaus posistemės detalus aprašymas

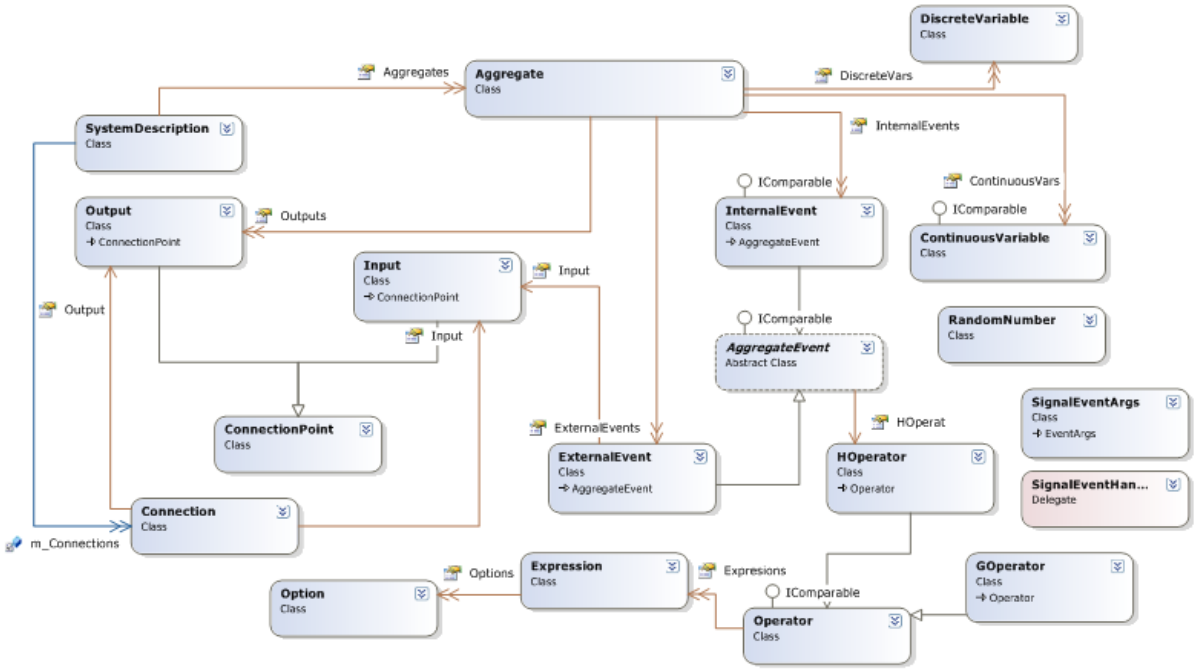
	
Klasifikacija	Posistemė
Apibrėžimas	Grafinis redaktorius skirtas analizuojamos sistemos aprašų sudarymui, redagavimui ir išsaugojimui *XML* failuose.
Atsakomybės	Posistemės pagrindinis panaudojimo tikslas yra formaliai aprašyti sudėtingą analizuojamą sistemą taip, kad aprašą būtų galima nesunkiai sutrasuoti į objektinį modelį. Formalizavimo kalba – *PLA*. Posistemės pagalba aprašomi tokie analizuojamos sistemos komponentai – agregatai, jų įėjimai, išėjimai, tolydūs, diskretūs kintamieji, H ir G operatoriai,

	sujungimai tarp agregatų ir agregatų signalų struktūra. Šio komponento pagalba galima sukurti naują sistemos aprašą, redaguoti ir validuoti jau esamus sistemų aprašus.
Apribojimai	Grafinio redaktoriaus posistemė yra glaudžiai susijusi su objektinio modelio posisteme, todėl redaguojami ir kuriami sistemų aprašai tinkami naudoti tik šios sistemos ribose. Sistemų aprašai saugomi *XML* failuose.
Struktūra	Posistemę sudaro septynios *Windows* formos, kurių kiekviena atsakinga už konkrečios aprašomos sistemos informacijos įvedimą ir redagavimą: • *frmEditor* – pagrindinė forma; • *frmAggregate* – forma, kurios pagalba galima aprašinėti agregatus; • *frmSignalType* – forma, kurios pagalba galima aprašinėti signalų struktūrą; • *frmConnections* – forma, kurios pagalba galima aprašinėti sujungimus tarp agregatų; • *frmInputsOutputs* – forma, kurios pagalba galima aprašinėti agregatų įėjimus ir išėjimus; • *frmVariable* – forma, kurios pagalba galima aprašinėti agregatų diskrečiuosius ir tolydžiuosius kintamuosius. • *frmOperator* – forma, kurios pagalba galima aprašinėti agregatų H ir G operatorius.
Sąveikavimas	Posistemė sąveikauja tik su objektinio modelio posisteme. Šios posistemės pagalba aprašyta analizuojama sistema konvertuojama į objektinį modelį, kuris saugomas objektinio modelio posistemėje.
Resursai	Posistemė nenaudoja jokių specialių resursų.
Skaičiavimai	Skaičiavimams nenaudoji jokie specialūs algoritmai, visi veiksmai atliekami nuosekliai.
Sąsaja/eksportas	Posistemė bendrauja tik su objektinio modelio posisteme. Bendravimui naudojamos objektinio modelio klasė *SystemDescription* ir grafinio redaktoriaus klasė *frmEditor*.

3.6. Objektinio modelio posistemės detali specifikacija

3.6.1. Klasių diagrama

Objektinio modelio klasių diagrama pavaizduota paveikslėlyje 3.5.



Pav. 3. 5. Objektinio modelio posistemės klasių diagrama

3.6.2. Detalus aprašymas

Objektinio modelio posistemės klasių diagramos detalus aprašymas pateiktas 3.2. lentelėje.

Lentelė 3. 2. Objektinio modelio posistemės detalus aprašymas

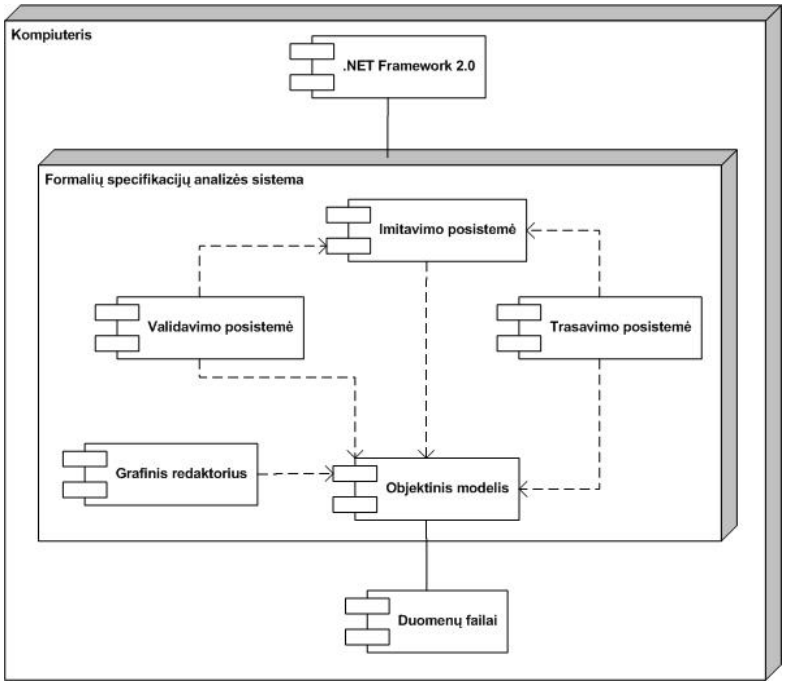
	
Klasifikacija	Posistemė
Apibrėžimas	Objektinis modelis skirtas saugoti formalų sistemos aprašą, sutransliuotą į kalbą, kurią supranta ir gali naudoti sistema (PLA-CA).
Atsakomybės	Posistemės pagrindinis panaudojimo tikslas yra saugoti formalų sistemos aprašą, kurį gali naudoti visos kitos sistemos posistemės. Sistemai aprašyti naudojama PLA-CA formalizavimo kalba. Posistemė saugo analizuojamos sistemos komponentus – agregatus, jų įėjimus, išėjimus, sujungimus tarp agregatų, tolydžiuosius bei diskrečiuosius kintamuosius, H ir G operatorius, valdymo sekas, ir imitacinio modeliavimo duomenis.
Apribojimai	Objektinis modelis transliuoja tik griežtos struktūros sistemos aprašą. Kitokios

	struktūros sistemos aprašai nėra transliuojami į objektinį modelį. Sistemų aprašai saugomi *XML* formatu.
Struktūra	Posistemę sudaro 20 klasių: • *Aggregate* – klasė aprašanti agregatą; • *AggregateEvent* – abstrakti klasė aprašanti agregato įvykį; • *Connection* – klasė aprašanti sujungimą, kuris susideda iš įėjimo ir išėjimo taškų; • *ConnectionPoint* –klasė aprašanti sujungimą; • *ContinuousVariable* – klasė aprašanti tolydų kintamąjį; • *DiscreteVariable* – klasė aprašanti diskretų kintamąjį; • *Expression* – klasė aprašanti išraišką; • *ExternalEvent* – klasė aprašanti išorinį įvykį, paveldi klasę *AggregateEvent*; • *GOperator* – klasė aprašanti G operatorių, paveldi Operator klasę; • *HOperator* – klasė aprašanti H operatorių, paveldi Operator klasę; • *Input* – klasė aprašanti įėjimą, paveldi *ConnectionPoint* klasę; • *InternalEvent* – klasė aprašanti vidinį įvykį, paveldi klasę *AggregateEvent*; • *ModelResults* – klasė atsakinga už imitacinio modeliavimo rezultatų saugojimą, eksportavimą; • *Operator* – klasė aprašanti visus operatorius; • *Option* – klasė aprašanti išraiškos alternatyvą; • *Output* – klasė aprašanti išėjimą, paveldi *ConnectionPoint* klasę; • *RandomNumber* – klasė generuojanti atsitiktinius skaičius; • *SystemDescritpion* – sistemos aprašas.
Sąveikavimas	Posistemė sąveikauja su visomis kitomis posistemėmis – objektiniame modelyje saugomas sistemos aprašas yra visų skaičiavimų pagrindas.
Resursai	Posistemė nenaudoja jokių specialių resursų.
Sąsaja/eksportas	Posistemė yra visų sistemos skaičiavimų pagrindas, todėl ją naudoja visos kitos posistemės. Posistemė naudojama turint klasės *SystemDescription* objektą.

3.7. Sistemos išdėstymo vaizdas

Sistema yra pritaikyta dirbti viename kompiuteryje. Nėra tikslinga kurti sistemą kaip paskirstytą sistemą, kadangi skaičiavimai vykdomi daugiausia nuosekliai, ir naudojant paskirstytą sistemą nebus pasiekta našumo padidėjimo. Dėl tos pačios priežasties sistemoje nėra naudojamas kelių gijų palaikymas (multithreading). Nors nemažai sistemų atskirų gijų naudojimas galėtų padėti, esant nuosekliems skaičiavimams tai neduotų jokios apčiuopiamos naudos.

Sistemos išdėstymo vaizdą (3.6. paveikslėlis) galima pavaizduoti taip:



Pav. 3. 6. Sistemos išdėstymo vaizdas

4. TYRIMO DALIS

Programinė įranga yra sudėtingų formalių specifikacijų integruotos analizės automatizavimo sistema (*FSA*), kuri skirta atlikti visą analizuojamos sistemos analizę: specifikacijos įvedimas ir redagavimas, statistinis imitavimas, trasavimas ir validavimas. Tam, kad įvertinti programinės įrangos teikiamas paslaugas, palengvinimus ir pagalbą vartotojui, reikia įvesti įvertinimo metrikas.

4.1. Vertinimo rezultatai

Formalių specifikacijų integruotos analizės automatizavimo sistemos vertinimo metrikos ir jų aprašymas pateikti lentelėje 4.1.

Lentelė 4. 1. Programinės įrangos vertinimo rezultatai

Parametras	Aprašymas
Saugumas	Nėra. Programoje nėra realizuoti jokios vartotojų autorizavimo ar kitos priemonės.
Išplečiamumas	Yra. Pateikiamas programinės įrangos kodas ir pilna techninė dokumentacijas. Programinė įranga rašyta vadovaujantis objektiškai orientuoto programavimo principais, kad vėliau funkcionalumą galima būtų papildyti neperrašant egzistuojančio kodo.
Pernešamumas	Dalinai yra. Programinė įranga sukurta naudojant *Microsoft C#* programavimo kalbą, todėl funkcionuoti gali ten, kur veikia *Microsoft .NET Framework 2.0* karkasas.
Sąsajos galimybės	Nėra. Sistema dirba tik su jai skirtais duomenimis. Ateityje bus galima praplėsti sistemos funkcionalumą, kad būtų galima dirbti ir su kitų panašių sistemų duomenimis, pvz., *SPIN*.
Panaudojamumas	Yra. Pateikiama pilna vartotojo dokumentacija.
Patvarumas	Yra. Vartotojo įvesti duomenys tikrinami, pateikiami pranešimai esant kokiems nors netikslumams.
Išbaigtumas	Yra. Realizuoti visi vartotojo pageidavimai ir reikalavimai. Visa sistema ar atskiri jos posistemiai gali būti naudojami praktiniams darbams.
Efektyvumas	Yra. *Microsoft C#* kalbos panaudojimas užtikrina minimalius reikalavimus techninei bei programinei įrangai vartotojo darbo vietoje.
Ištestavimo lygis	Yra. Atsižvelgiant į resursų bei laiko trūkumą, sistema ištestuota maksimaliai, gauti vartotojų bei užsakovų atsiliepimai, realizuoti pataisymai.
Lankstumas	Nėra. Vartotojo aplinka negali būtų tinkinama. Vartotojas mažai gali įtakoti posistemio funkcionalumą.

4.2. Palyginimo kriterijai

Kad geriau suprasti programinės įrangos teikiamas paslaugas ir jų privalumas, reikia programinę įrangą (*FSA* sistemą) palyginti su jos tiesioginiais konkurentais pasaulyje (skyrius 2.7.).

Programinės įrangos įvertinimui gali būti naudojami tokie kriterijai:

- Grafinis specifikacijos redaktorius (1);
- Imitavimo galimybė (2);
- Trasavimo galimybė (3);
- Validavimo galimybė (4);
- Vartotojo sąsaja (5);
- Įrankio pernešamumas (6);
- Sistemos saugumo savybių tikrinimas (7);
- Sistemos gyvybingumo savybių tikrinimas (8);
- Specifikacijos nepilnumo tikrinimas (9);
- Specifikacijos perpildymo tikrinimas (10);
- Specifikacijos tikrinimas neteisingos pabaigos būsenos požiūriu (11);
- Darbo realiu laiku galimybė (12);
- Būsenų grafo saugojimas (13).

Lentelė 4. 2. Egzistuojančių sistemų savybių palyginimas

Sistema / Savybė	Spin	VIS	SMV	NuSMV	Uppaal	Pranas-2	ValSYS	AgDraw	FSA
1	-	-	-	-	-	+	+	+	+
2	-	-	-	-	+	+-	-	+	+
3	+-	-	-	-	+-	-	+-	-	+
4	+	+	+	+	+	+-	+	+	+
5	+	-	+	+	+	+-	+	+	+
6	+	+	+	+	+	+	+	+	+
7	+	+	+	+	+	+-	+	+	+
8	+	+	+	+	+	+	+	+	+
9	+	+	+	+	+	+	+	+	+
10	+	-	-	+	+	-	-	-	-
11	+	+	+	+	+	+	+	+	+
12	+	-	-	-	+	-	-	-	-
13	-	-	-	-	-	+	+	-	+

Egzistuojančių sistemų ir *FSA* sistemos savybių palyginimo lentelėje (lentelė 4.2.) pateiktų simbolių paaiškinimas:

„+” - savybė sistemoje įdiegta pilnai;

„-“ - savybė sistemoje neįdiegta;

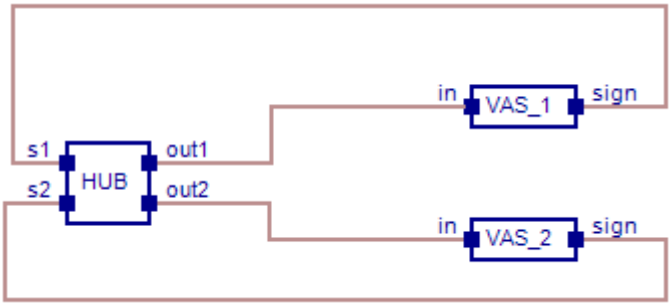
„+-” - savybė sistemoje įdiegta dalinai.

5. EKSPERIMENTINĖ DALIS

Šioje dalyje pateikiamas „Dviejų vienkanalių sistemų su maršrutizatoriumi“ konceptualinis modelis, specifikacija, specifikacijos apdorojimas su programine įranga. Pilnas specifikacijos vaizdas *XML* kalboje pateikiamas 1 priede „Agregatinės specifikacijos XML failo pavyzdys“.

5.1. Konceptualinis modelis

Šioje dalyje analizuojamą sistemą sudaro dvi vienkanalės sistemos ir maršrutizatorius. Bendras sistemos vaizdas pateiktas 5.1. paveikslėlyje



Pav. 5. 1. Bendras konceptualinio modelio vaizdas

Maršrutizatoriaus „HUB“ paskirtis:

- Maršrutizatorius paskirto į jį atėjusias paraiškas atitinkamai vienkanalei sistemai. Jei sistema „VAS_1“ užimta, paraiška perduodama sistemai „VAS_2“. Jei abi sistemos užimtos, paraiška ignoruojama.
- Maršrutizatoriaus įėjimai „s1“ ir „s2“ naudojami nustatyti, kuri aptarnaujanti sistema gali priimti atėjusią paraišką. Jei „s1 = 1“, tai sistema „VAS_1“ užimta ir aptarnauti paraiškos negali. Atitinkamai viskas vyksta ir su sistema „VAS_2“.
- Maršrutizatoriaus išėjimai siunčiamos paraiškos aptarnaujančioms sistemoms. Jei paraiška siunčiama sistemai VAS_1, tai naudojamas išėjimas „out1“, jei siunčiama VAS_2 – naudojamas išėjimas „out2“.

Abi vienkanalės sistemos yra identiškos, todėl norint pilnai aprašyti abi sistemas, užtenka paaiškinti tik vienos sistemos veikimą. Vienkanalės sistemos paskirtis:

- Vienkanalė sistema aptarnauja į ją patekusias paraiškas. Sistema turi fiksuoto dydžio buferį, į kurį talpinamos paraiškos, jei atėjus naujai paraiška jau yra aptarnaujama egzistuojanti paraiška. Jei buferis užpildomas maksimaliai, naujai atėjusi paraiška ignoruojama.
- Vienkanalės aptarnavimo sistemos iėjimas „in“ naudojamas paraiškai priimti.
- Vienkanalės aptarnavimo sistemos išėjimas „sign“ naudojamas informacijai apie sistemos užimtumą į maršrutizatorių pasiųsti. Jei „sign = 1“, tai sistema yra užimta ir paraiškų aptarnauti negali. Jei „sign = 0“, tai sistema paraiškas priimti gali.

5.2. Agregatinė specifikacija

Kadangi šis konceptualinis modelis susideda iš trijų agregatų, tai kiekvienas agregatas turi būti aprašomas atskirai pagal skyrelyje 2.5. aprašytus punktus. Abi vienkanalės sistemos yra identiškos, todėl pilnai aprašyti užtenka tik vieną sistemą, o kitą galima tiesiog nukopijuoti.

Agregato „HUB“ specifikacija:

1. Įėjimų aibė $X = \{s1, s2\}$
2. Išėjimų aibė $Y = \{out1, out2\}$
3. Diskrečiųjų kintamųjų aibė $D = \{state1, state2\}$
4. Tolydžiųjų kintamųjų aibė $T = \{par\_gen\}$
5. Išorinių įvykių aibė $E' = \{atejo\_s1, atejo\_s2\}$
6. Vidinių įvykių aibė $E'' = \{gen\_par\}$
7. H operatoriai

   H(atejo_s1): $state1 = s1$

   H(atejo_s2): $state2 = s2$

   H(gen_par): *par_gen = random();*
8. G operatoriai

   G(gen_par):

$$out1 = \begin{cases} 1, kai \quad state1 = 0 \\ 0, kai \quad state1 = 1 \end{cases}$$

$$out2 = \begin{cases} 1, \quad kai \quad state1 \neq 0 \quad \&\& \quad state2 = 0 \\ 0, \quad kai \quad state2 = 1 \quad \| \quad state1 = 0 \end{cases}$$

9. Pradinė būsena $Z(t_0) = \{0, 0, 0, 0, 0, 0, 0.1\}$

Agregato „Vienkanalė aptarnavimo sistema“ specifikacija:

1. Įėjimų aibė $X = \{in\}$
2. Išėjimų aibė $Y = \{sign\}$
3. Diskrečiųjų kintamųjų aibė $D = \{buf, buf_{max}, dirba\}$
4. Tolydžiųjų kintamųjų aibė $T = \{par\_apt\}$
5. Išorinių įvykių aibė $E' = \{atejo\_par\}$

6. Vidinių įvykių aibė $E'' = \{apt\_par\}$

7. H operatoriai

   H(atejo_par):

$$par\_apt = \begin{cases} random(), & kai \quad buf = 0 \quad \&\& \quad sign = 0 \quad \&\& \quad dirba = 0 \quad \&\& \quad in = 1 \\ par\_apt, & kai \quad buf \neq 0 \quad \| \quad sign = 1 \quad \| \quad dirba = 1 \quad \| \quad in = 0 \end{cases}$$

$$buf = \begin{cases} buf, & kai \quad buf = buf_{\max} \quad \| \quad in = 0 \quad \| \quad dirba = 0 \quad \| \quad sign = 1 \\ buf + 1, & kai \quad buf \neq buf_{\max} \quad \&\& \quad in = 1 \quad \&\& \quad sign = 0 \quad \&\& \quad dirba = 1 \end{cases}$$

$$dirba = \begin{cases} dirba, & kai \quad in = 0 \\ 1, & kai \quad in = 1 \end{cases}$$

   H(apt_par):

$$par\_apt = \begin{cases} random(), & kai \quad buf \neq 0 \\ \infty, & kai \quad buf = 0 \end{cases}$$

$$dirba = \begin{cases} 0, & kai \quad buf = 0 \\ 1, & kai \quad buf \neq 0 \end{cases}$$

$$buf = \begin{cases} buf, & kai \quad buf = 0 \\ buf - 1, & kai \quad buf \neq 0 \end{cases}$$

8. G operatoriai

   G(atejo_par):

$$sign = \begin{cases} 1, & kai \quad sign = 0 \quad \&\& \quad in = 1 \quad \&\& \quad buf = buf_{\max} \\ sign, & kai \quad sign \neq 0 \quad \| \quad in = 0 \quad \| \quad buf \neq buf_{\max} \end{cases}$$

   G(apt_par):

$$sign = \begin{cases} 0, & kai \quad buf \neq buf_{\max} \\ 1, & kai \quad buf = buf_{\max} \end{cases}$$

9. Pradinė būsena $Z(t_0) = \{0, 0, 0, 3, 0, \infty\}$

5.3. Agregatinės specifikacijos apdorojimas programine priemone FSA

Sudėtingų formalių specifikacijų integruotos analizės automatizavimo sistemą sudaro keturi įrankiai:

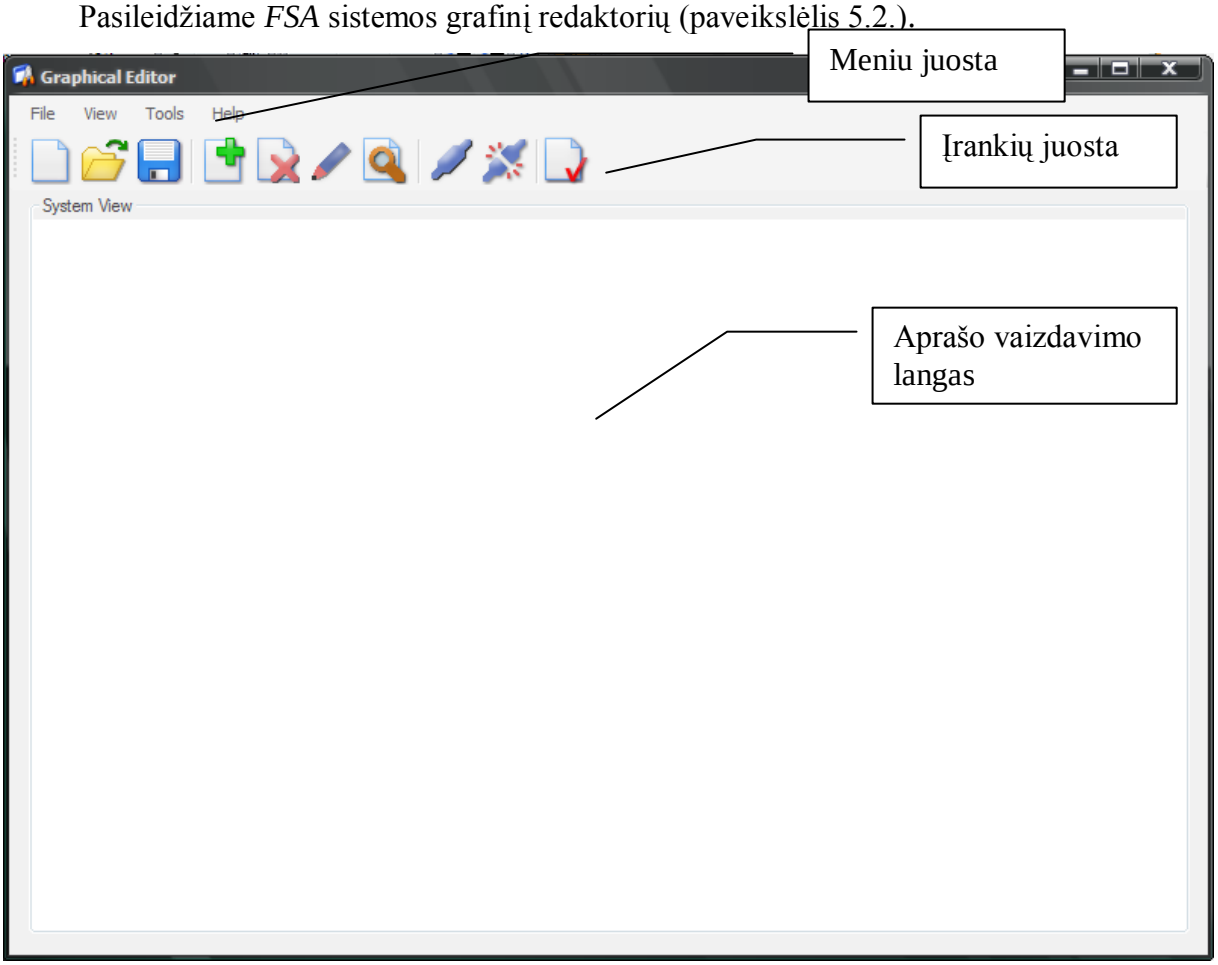
1. Grafinis redaktorius;
2. Imitavimo įrankis;
3. Trasavimo įrankis;
4. Validavimo įrankis.

Eksperimento metu mes pirmiausia grafinio redaktoriaus pagalba aprašysim analizuojamą sistemą. Aprašę sistemą, patikrinsim formalios specifikacijos teisingumą.

Patikrinę ir ištaisę visas klaidas, pamėginsim aprašytą sistemą paleisti imitavimo, trasavimo ir validavimo įrankiuose tam, kad įsitikinti, jog formalus sistemos aprašas yra korektiškas.

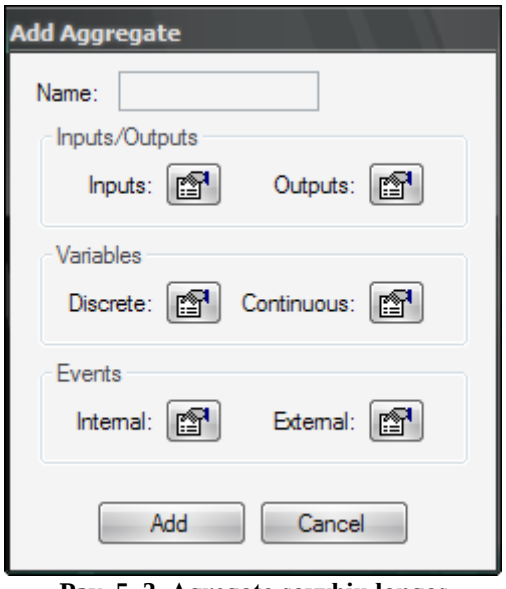
5.3.1. Agregatinės specifikacijos apdorojimas grafiniu redaktoriumi

Pasileidžiame *FSA* sistemos grafinį redaktorių (paveikslėlis 5.2.).



Pav. 5. 2. Grafinio redaktoriaus langas

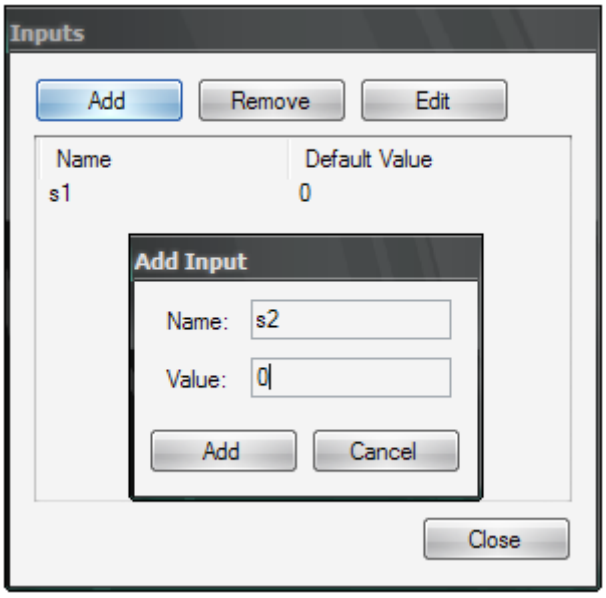
Iš įrankių juostos pasirenkam įrankį „Add Aggregate“  . Atidaromas agregato savybių langas (paveikslėlis 5.3.).



Pav. 5. 3. Agregato savybių langas

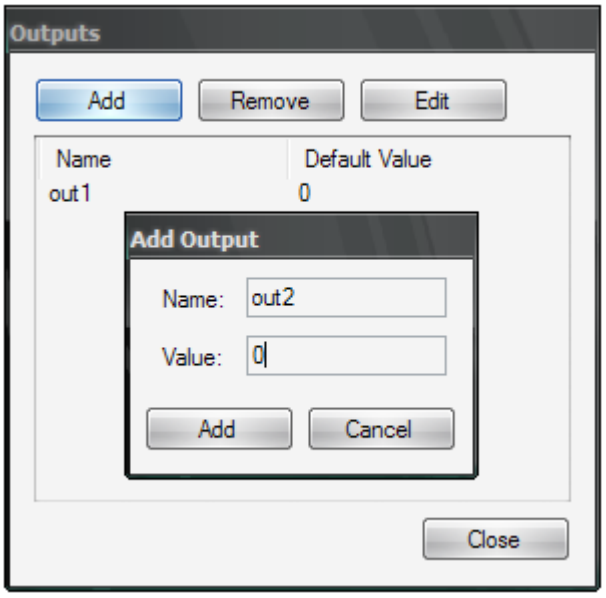
Atsidariusiame agregato savybių lange pirmiausia nurodom agregato vardą. Pirmiausia mes aprašinėsim maršrutizatoriaus agregatą, todėl agregato vardo įvedimo laukelyje „Name“ įvedam „HUB“.

Toliau reikia nurodyti visus agregato įėjimus ir jų pradines reikšmes. . Paspaudus mygtuką  prie “Inputs”, atidaromas agregato įėjimų įvedimo langas (paveikslėlis 5.4.).



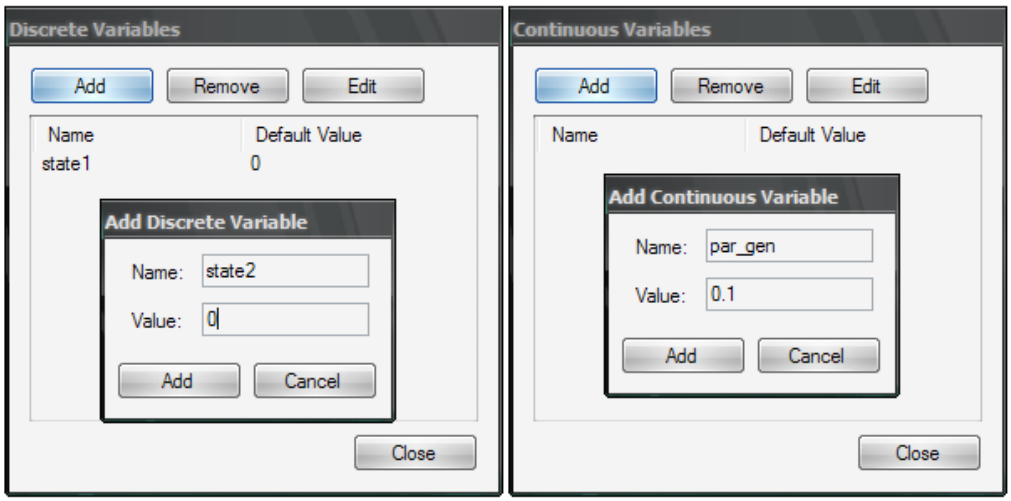
Pav. 5. 4. Agregato įėjimų įvedimo langas

Agregato išėjimai įvedami analogiškai kaip įėjimai (paveikslėlis 5.5.).



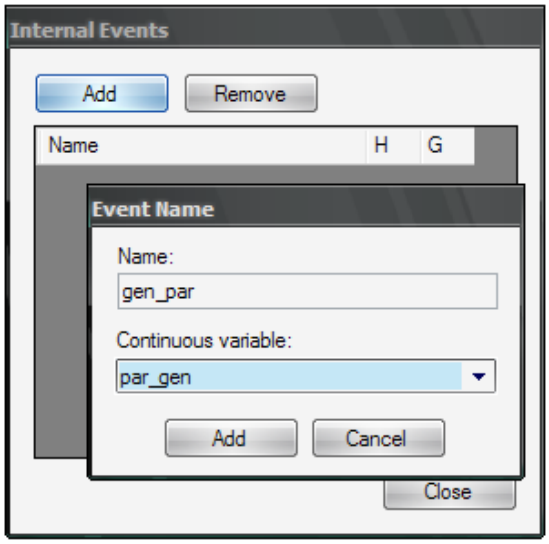
Pav. 5. 5. Agregato išėjimų įvedimo langas

Įvedus agregato įėjimus ir išėjimus, reikia nurodyti jo tolydžiuosius ir diskrečiuosius kintamuosius. Tolydūs ir diskretūs kintamieji įvedami nurodant jų vardus ir pradines reikšmes (paveikslėlis 5.6.).



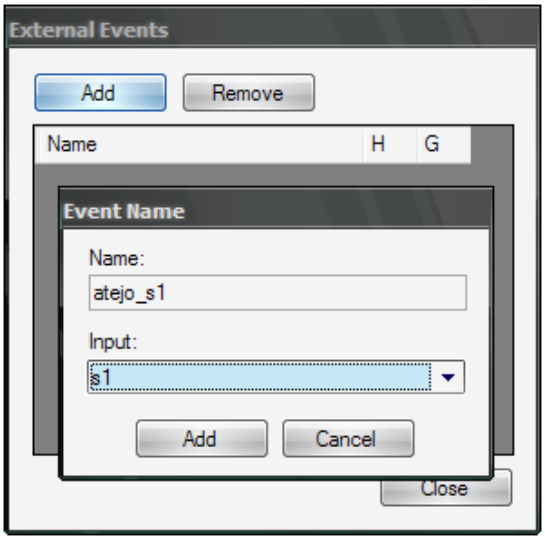
Pav. 5. 6. Agregato tolydžiųjų ir diskrečiųjų kintamųjų įvedimo langai

Toliau reikia įvesti agregato vidinius įvykius. Vidiniai įvykiai įvedami nurodant įvykio vardą ir su juo susietą tolydųjį kintamąjį (paveikslėlis 5.7.).



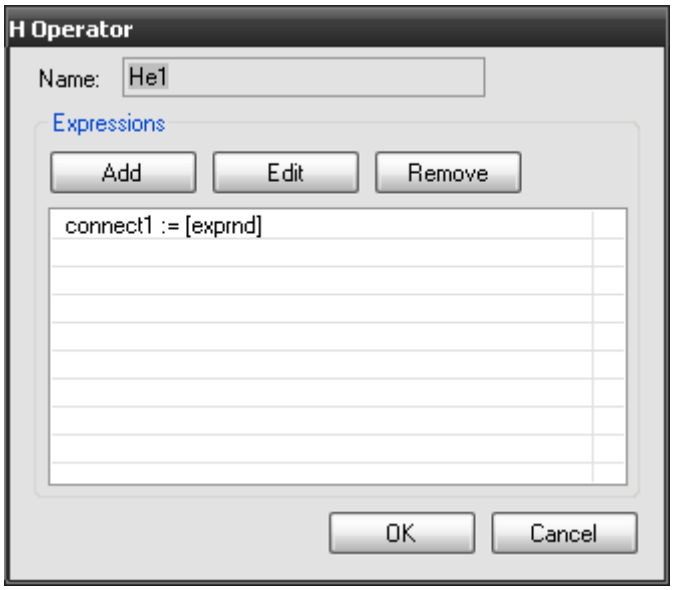
Pav. 5. 7. Agregato vidinių įvykių įvedimo langas

Agregato išoriniai įvykiai nurodomi panašiai kaip ir vidiniai, tik vietoj susieto tolydaus kintamojo reikia nurodyti susietą įėjimą (paveikslėlis 5.8.).



Pav. 5. 8. Agregato išorinių įvykių įvedimo langas

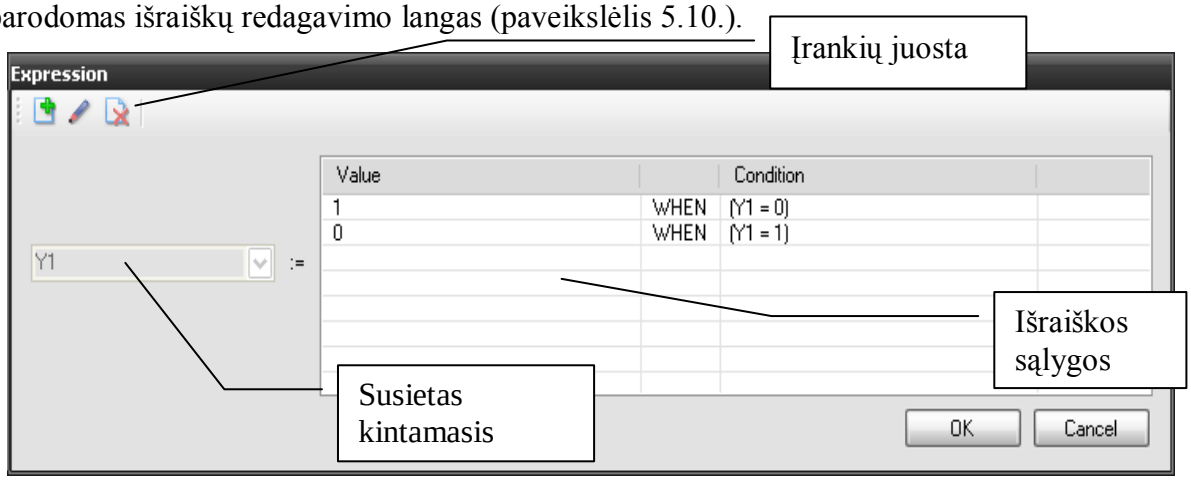
Norint redaguoti vidinio ar išorinio agregato įvykio H arba G operatorius, reikia paspausti atitinkamo operatoriaus redagavimo mygtuką  įvykių įvedimo lange. Vartotojui bus parodytas operatoriaus redagavimo langas (paveikslėlis 5.9.).



Pav. 5. 9. Operatoriaus redagavimo langas

Operatoriaus redagavimo lange rodomas operatoriaus vardas (laukas "Name"). Vartotojas vardo keisti negali. Kiekvienas agregato H ir G operatorius aprašomas naudojant išraiškas, angl. "Expressions". Vartotojas gali kurti, šalinti, redaguoti operatoriaus išraiškas.

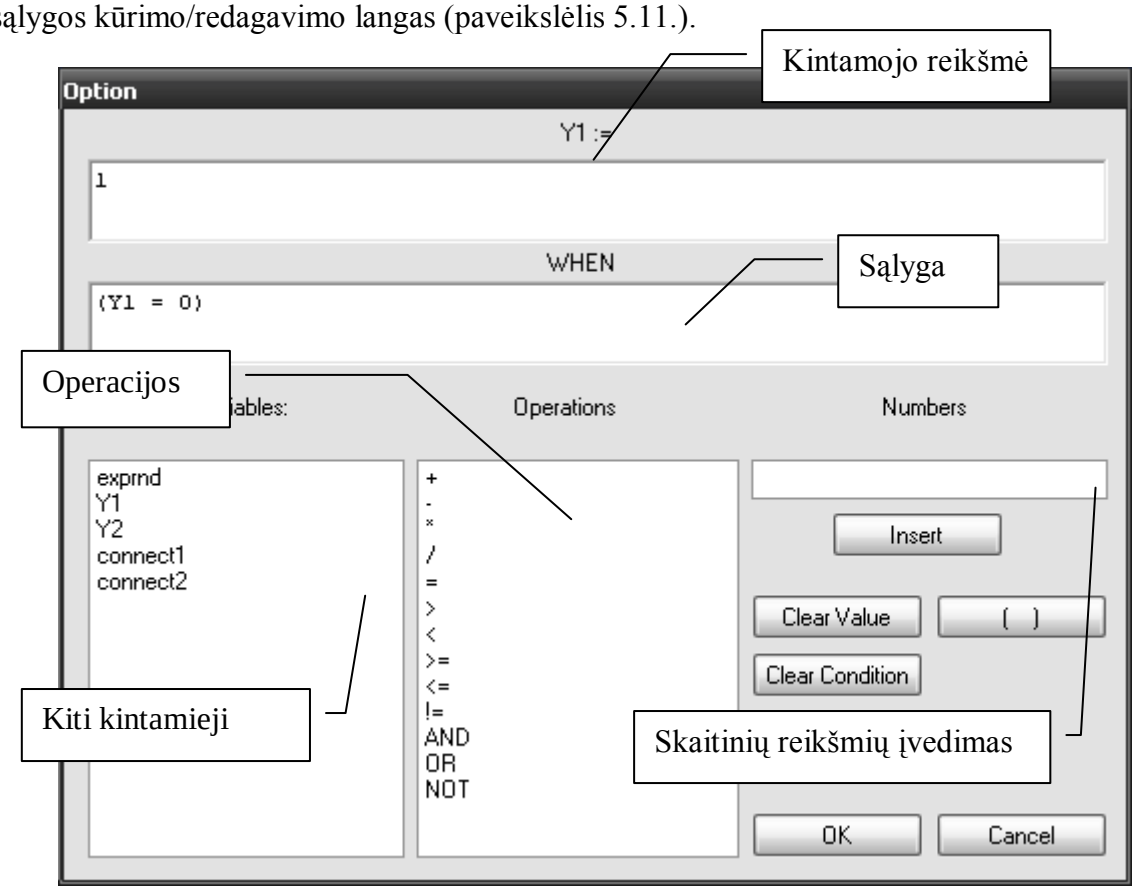
Norint sukurti naują operatoriaus išraišką, reikia spausti mygtuką "Add". Vartotojui parodomas išraiškų redagavimo langas (paveikslėlis 5.10.).



Pav. 5. 10. Operatoriaus išraiškos kūrimo langas

Kuriant naują operatoriaus sąlygą, pirmiausia reikia nurodyti susietą kintamąjį. Tai gali būti agregato diskretus kintamasis, tolydus kintamasis arba išėjimas. Visos išraiškos sąlygos vaizduojamos išraiškos sąlygų sąraše. Sąlygai kurti, redaguoti ir šalinti galima naudojantis įrankių juostoje esančiais mygtukais. Norint pašalinti sąlygą, reikia pažymėti šalinamas sąlygas ir paspausti mygtuką  . Norint sukurti naują arba redaguoti jau sukurtą sąlygą,

reikia atitinkamai spausti mygtukus  ir  . Paspaudus minėtus mygtukus, atidaromas sąlygos kūrimo/redagavimo langas (paveikslėlis 5.11.).



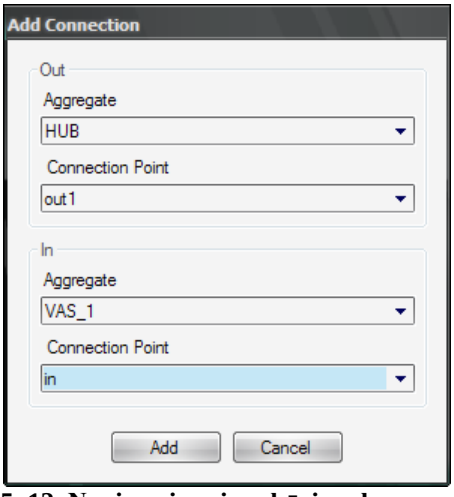
Pav. 5. 11. Operatoriaus išraiškos kūrimo/redagavimo langas

Norint sukurti naują išraiškos sąlygą, pirmiausia reikia nurodyti susieto kintamojo reikšmę. Reikšmę gali būti tiek skaitinė, tiek lygi kitam kintamajam. Norint nurodyti kito kintamojo reikšmę, reikia kursorių (raudono spalvos!) pastatyti kintamojo reikšmės lauke ir dukart spustelti su pelės kairiuoju klavišu ant kintamojo iš kitų kintamųjų sąrašo. Norint nurodyti skaitinę reikšmę, reikia įvesti norimą skaičių į skaitinių reikšmių įvedimo lauką, pastatyti raudoną kursorių kintamojo reikšmės lauke ir spausti mygtuką “Insert”.

Išraiškos sąlygai taip pat reikia nurodyti sąlygą. Sąlyga nurodoma analogiškai kaip ir kintamojo reikšmė – pele spaudinėjant ant kintamųjų, sąlygos operatorių ir įvedinėjant skaitines reikšmes. Svarbiausia – raudoną kursorių pastatyti reikiamam lauke. Įvedinėjant sudėtingesnes sąlygas, galima naudoti skliaustų įterpimo mygtuką “( )”, kuris automatiškai įterpia skliaustus.

Įvedus visą agregato informaciją, pagrindiniame agregato savybių lange spaudžiame mygtuką „Add“ ir agregatas bus sukurtas. Analogiškai kaip maršrutizatoriaus agregatą „HUB“, reikia įvesti ir vienkanalių aptarnavimo sistemų „VAS_1“ ir „VAS_2“ agregatus.

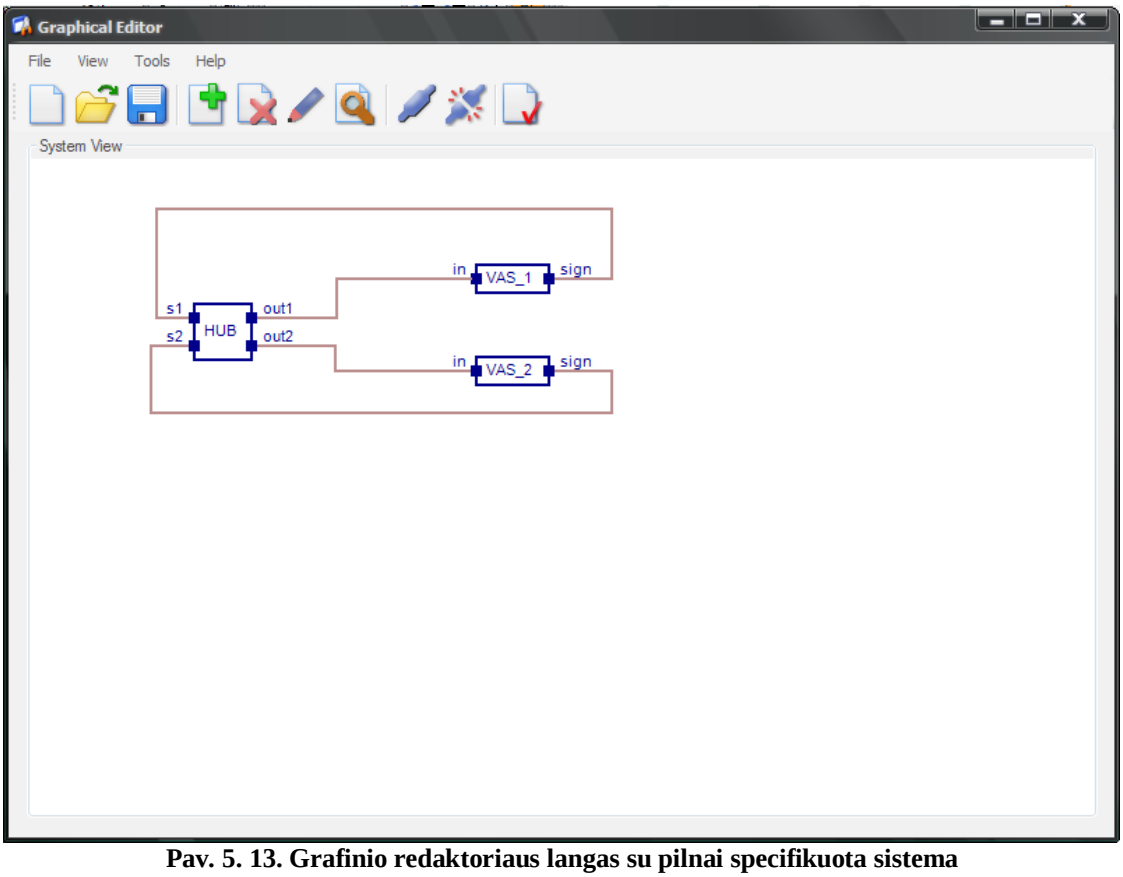
Turint visus sistemos agregatus, reikia įvesti sujungimus tarp jų. Tam naudojamas mygtukas „Add Connection“  iš grafinio redaktoriaus (paveikslėlis 5.2.) įrankių juostos. Atidaromas naujo sujungimo kūrimo langas (paveikslėlis 5.12.).



Pav. 5. 12. Naujo sujungimo kūrimo langas

Norint sukurti naują sujungimą, reikia dalyje “Out” nurodyti išeinantį agregatą (laukas “Aggregate”) ir jo išėjimą (laukas “Connection Point”), bei dalyje “In” įeinantį agregatą (laukas “Aggregate”) ir jo įėjimą (laukas “Connection Point”). Nurodžius visą reikalingą informaciją, spausti mygtuką “Add”.

Aprašius visą specifikuojamą sistemą, pagrindiniame grafinio redaktoriaus lange turėtų būti toks sistemos vaizdas (paveikslėlis 5.13.):

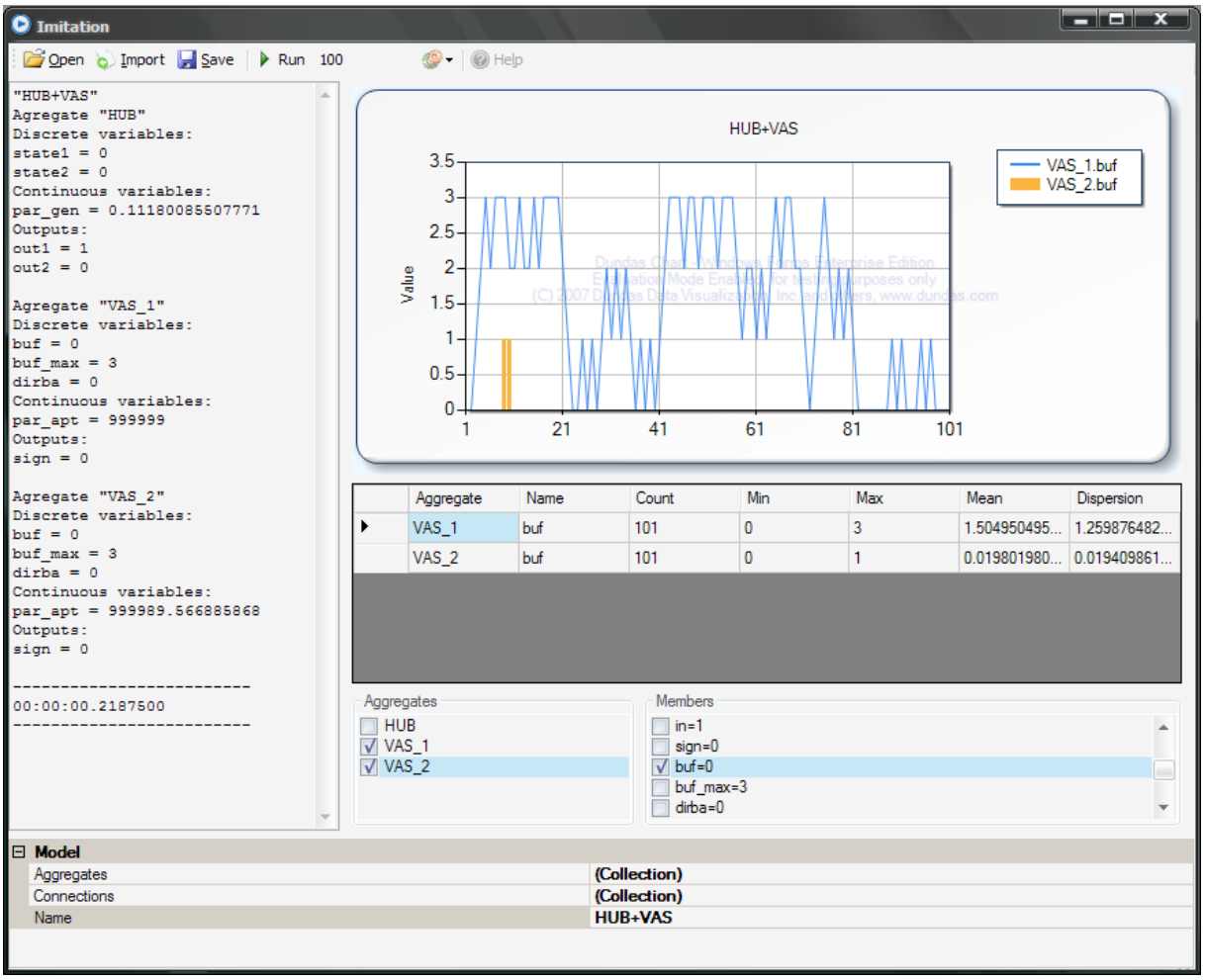


Pav. 5. 13. Grafinio redaktoriaus langas su pilnai specifikuota sistema

Norint įsitikinti, ar sistema specifikuota bei paprastų sintaksės klaidų, galima paspausti įrankių juostos mygtuką „Check System Description“  . Jei rasta sintaksės klaidų – rodomas pranešimas vartotojui su visomis rastomis klaidomis, priešingu atveju rodomas pranešimas, kad validavimas buvo sėkmingas.

5.3.2. Agregatinės specifikacijos apdorojimas imitavimo įrankiu

Kad įsitikintume, jog aprašyta agregatinė specifikacija tikrai veikia ir yra korektiška, galime apdoroti ją imitavimo įrankiu (paveikslėlis 5.14.).

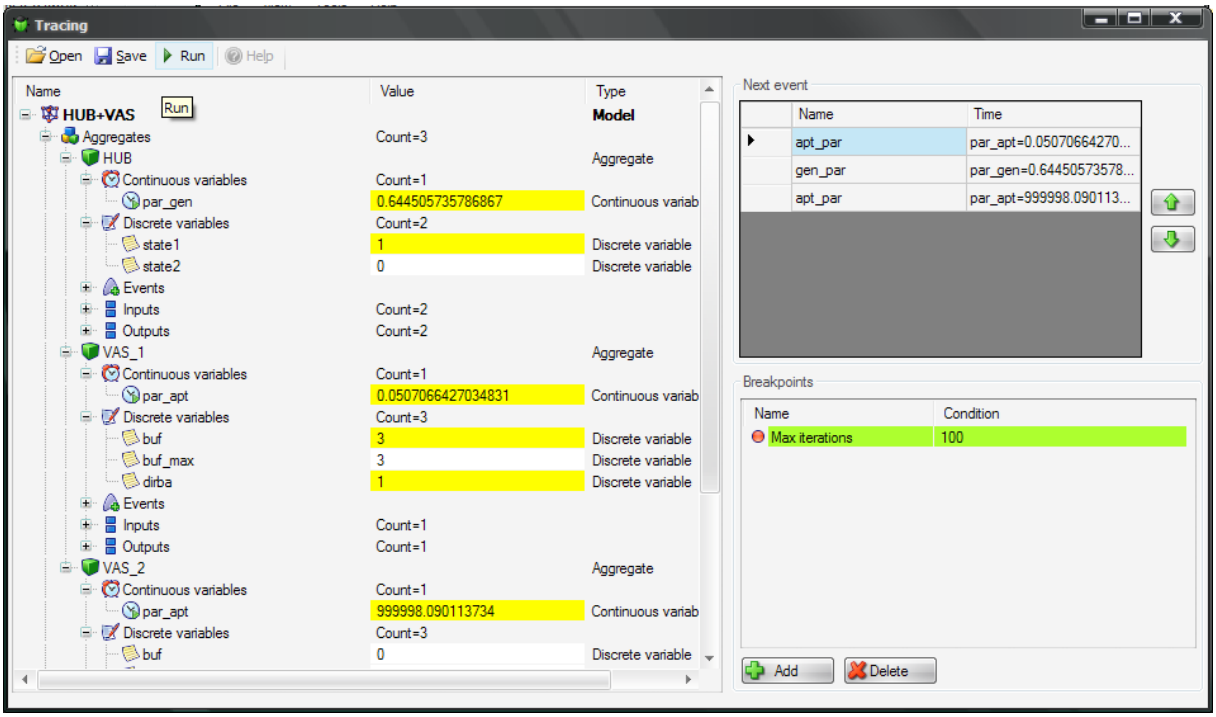


Pav. 5. 14. Agregatinės specifikacijos apdorojimas imitavimo įrankiu

Kaip matyti, aprašyta agregatinė specifikacija yra korektiška ir galima atlikti statistinį imitavimą naudojantis *FSA* sistemos imitavimo įrankiu.

5.3.3. Agregatinės specifikacijos apdorojimas trasavimo įrankiu

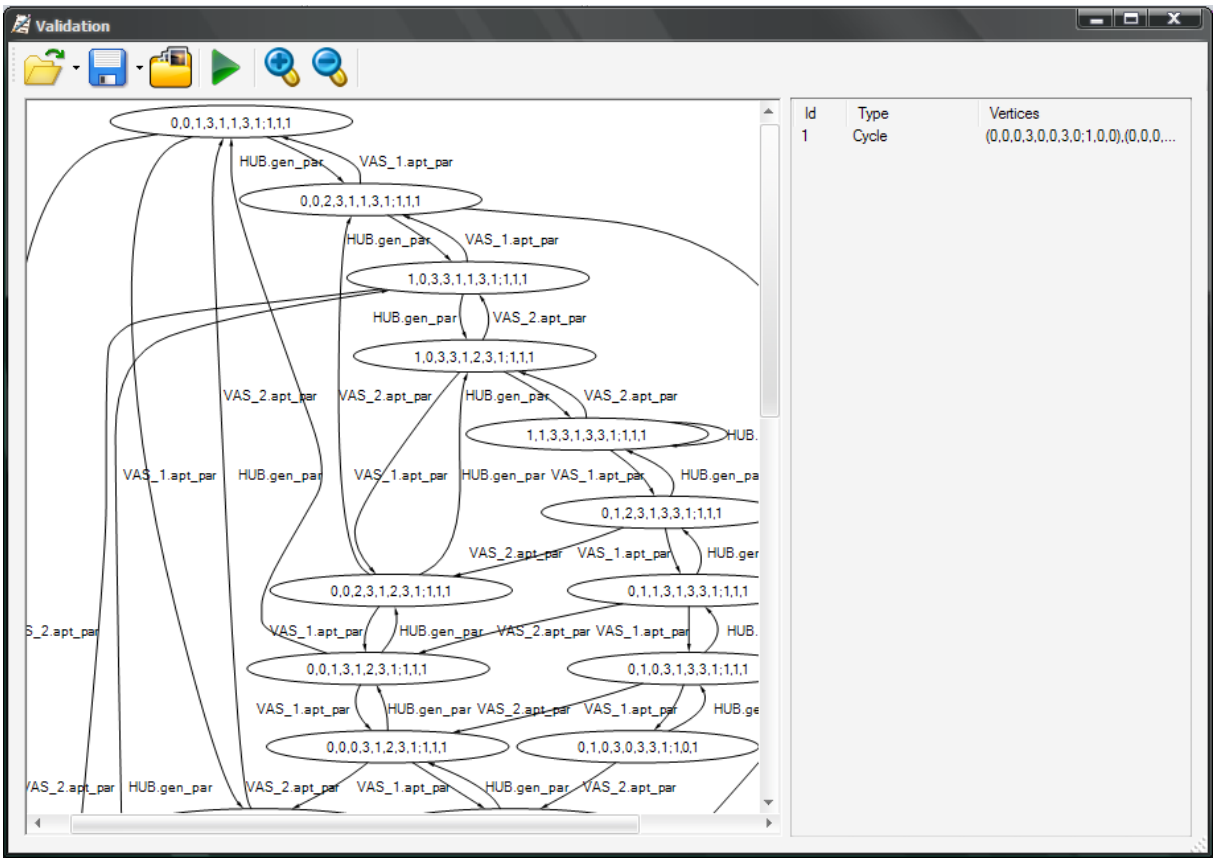
Jei reikia atlikti aprašytos agregatinės specifikacijos trasavimą, patikrinti kintamųjų reikšmes įvairiais laiko momentais, galima apdoroti ją *FSA* sistemos trasavimo įrankiu (paveikslėlis 5.15.).



Pav. 5. 15. Agregatinės specifikacijos apdorojimas trasavimo įrankiu

5.3.4. Agregatinės specifikacijos apdorojimas validavimo įrankiu

Norint patikrinti specifikacijos teisingumą, galima ją apdoroti *FSA* sistemos validavimo įrankiu (paveikslėlis 5.16.).



Pav. 5. 16. Agregatinės specifikacijos apdorojimas validavimo įrankiu

6. IŠVADOS

6.1. Išvados

- Sukurtos grafinio redaktoriaus ir objektinio modelio posistemės yra sudėtingų formalių specifikacijų integruotos analizės sistemos sudedamosios dalys. Visą sistemą sudaro šios posistemės: objektinio modelio, grafinio redaktoriaus, imitavimo, trasavimo ir validavimo.
- Grafinio redaktoriaus posistemė leidžia specifikuotojui daug greičiau, su mažesne klaidų tikimybe suspecifikuoti sistemą, nepaliekant jam laisvės savaip interpretuoti agregatinės specifikacijos. Specifikuotojas privalo pilnai aprašyti kiekvieną sistemos agregatą ir ryšius tarp jų. Agregatai aprašomi nurodant jo įėjimų ir išėjimų, tolydžiųjų ir diskrečiųjų kintamųjų, išorinių ir vidinių įvykių aibes bei H ir G operatorius.
- Sukurta objektinio modelio posistemė leidžia išsaugoti *PLA* modelį kompiuteryje bei panaudoti jį kitose sudėtingų formalių specifikacijų integruotos analizės automatizavimo posistemėse, sprendžiant imitacinio modeliavimo, trasavimo ir validavimo uždavinius.
- Išsaugotą analizuojamos sistemos specifikaciją *XML* formatu galima lengvai panaudoti kitoms posistemėms – imitacinio modeliavimo, trasavimo ir validavimo.
- Atlikus visos sudėtingų sistemų formalių specifikacijų integruotos analizės automatizavimo sistemos ir analogiškų produktų pasaulyje palyginimą, galima teigti, kad magistro studijų metu sukurta sistema savo galimybėmis nenusileidžia daugeliui konkurentų. Tai yra viena iš nedaugelio sistemų, kuri apima specifikacijų sudarymą, imitacinį modeliavimą, trasavimą ir validavimą kartu.
- Grafinio redaktoriaus funkcionalumas ištestuotas su realiu „dviejų vienkanalių sistemų su maršrutizatoriumi“ uždaviniu. Aprašyta sistemos specifikacija išmėginta su kitomis posistemis.

6.2. Tolimesni darbai

- Realizuoti dinaminio *PLA* modelio (dynPLA) palaikymą. Dinaminis *PLA* modelis – tai *PLA* papildymas ir pritaikymas dinaminėms sistemoms, kurios keičia savo struktūrą ir/arba elgseną laike [27]. Dinaminis *PLA* įgalintų modeliuoti sistemas, gebančias evoliucionuoti laike, o tai atvertų plačias *PLA* taikymo galimybes. Kadangi esamam sprendime yra galimybė keisti sistemos būseną ir dalinai keisti struktūrą, reikėtų papildyti *PLA* modelį, ko pasėkoje keistųsi sistemos formaliojo aprašo struktūra ir jį interpretuojančios kitos *FSA* dalys.
- Siekiant, kad sukurtas įrankis ir *PLA* formalizmas būtų plačiai naudojamas ir žinomas, reikalinga sukurti ir realizuoti modelių transformacijas iš ir į *PLA* modelį. Tokiu atveju

bus galima panaudoti įrankius, kurie yra realizuoti kitiems modeliams ir kitus modelius transformavus į *PLA*, jų analizei naudoti *FSA* funkcionalumą.

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

7. LITERATŪRA

[1] *Computation and Formal Systems* [interaktyvus][žiūrėta 2007-11-20]. Prieiga per internetą: <http://www.cs.rochester.edu/~nelson/courses/csc_173/overview.html>

[2] H. Pranevičius. *„Kompiuterių tinklų protokolų formalusis specifikavimas ir analizė: agregatinis metodas“*. Monografija. Kaunas: Technologija, 2005.

[3] 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.

[4] Flake, S., Muller, W., Ruf, J. *Structured English for Model Checking specification*// GI/ITG/GMM Workshop [Frankfurt/M., Germany]. 2000.

[5] Garlan, D. *Software Architecture: a Roadmap*// The Future of Software Engineering: tarptautinės konferencijos pranešimo medžiaga [Ireland, 2000]. ACM Press, p.91–101. USA, 2000.

[6] 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.

[7] Miller, S., P., Tribble, A., C., Heimdahl, M., P., E. *Proving the shalls*// FME2003 the 12th International FME Symposium [Italy, 2003].

[8] 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.

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

[10] „*A Case for Formal Specification*“ [interaktyvus] [žiūrėta 2007-11-12]. Prieiga per internetą <http://www.kuro5hin.org/story/2005/7/29/04553/9714>

[11] Babich F.; Deotto L.; *Formal Methods for Specification and Analysis of Communication Protocols* [interaktyvus] [žiūrėta 2007-11-22]. Prieiga per internetą: <http://www.comsoc.org/livepubs/surveys/Public/2002/Dec/babich.html>

[12] 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.

[13] 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.

[14] Bakanas, A., Packevičius, Š., Pranevičius, H. *Intelektualus agregatinių specifikacijų redaktorius*// Informacinės technologijos 2005: tarptautinės konferencijos pranešimo medžiaga [Kaunas, 2005]. Kaunas 2005.

[15] H. Pranevičius, J. A. G. Knight, I. Pranevičienė. *Formal Specification and Analysis of Distributed Systems Used Aggregate Approach*. Information Technology and Control. No.22(1). ISSN 1392-124 X. Kaunas: Technologija, 2002. P.31–38.

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

[17] *Formal Systems Verification LAB* [interaktyvus] [žiūrėta 2008-05-14]. Prieiga per internetą <http://fsv.dimi.uniud.it/software>.

[18] *Carnegie Mellon Software Engineering Institute* [interaktyvus] [žiūrėta 2008-05-14]. Prieiga per internetą <http://www.sei.cmu.edu/vtu/tools.html>.

[19] *„ON-THE-FLY, LTL MODEL CHECKING with SPIN“* [interaktyvus] [žiūrėta 2007-11-28]. Prieiga internete <http://spinroot.com/spin/whatispin.html>.

[20] The PROMELA Language [interkatyvus] [žiūrėta 2006-11-26]. Prieiga per internetą <http://www.dai-arc.polito.it/dai-arc/manual/tools/jcat/main/node168.html>.

[21] *VIS (Verification Interacting with Synthesis)* [interaktyvus] [žiūrėta 2007-11-14]. Prieiga per internetą <http://vlsi.colorado.edu/~vis/>.

[22] *SMV* [interaktyvus] [žiūrėta 2006-11-10]. Prieiga per internetą <http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/>.

[23] *NuSMV* [interaktyvus] [žiūrėta 2006-11-14]. Prieiga per internetą <http://nusmv.irst.itc.it/NuSMV/index.html>.

[24] *Uppaal* [interaktyvus] [žiūrėta 2006-11-14]. Prieiga per internetą <http://www.uppaal.com/>.

[25] H. Pranevičius, I. Pranevičienė, R. Benkunskis. *Formalization and Simulation Telecommunication Protocols and Systems using PLA Method*. Kaunas: Technologija, 2005. – Nr. 4(60).P. 5–10.

[26] V. Astrovas. Magistrinis darbas „*Agregatinių specifikacijų saugumo ir gyvybingumo tyrimo sistema“*. Kaunas, 2006.

[27] H.Pranevičius, Š.Packevičius, A.Kazla. *PLA Specifikacijų išplėtimas dinaminių sistemų formalizavimui*. Informacinės technologijos 2006. Konferencijų pranešimų medžiaga. II tomas. ISBN 9955-09993-3. Kaunas, Technologija. 2006. – p.475-478.

8. TERMINŲ IR SANTRAUKŲ ŽODYNAS

FSA – sudėtingų sistemų integruotos analizės automatizavimo sistema (***F****ormal Complex* ***S****ystems Integrated* ***A****nalysis System*).

UML – Unifikuota modeliavimo kalba (***U****nified* ***M****odeling* ***L****anguage*).

RSML – Reikalavimų išdėstymo mašininė kalba (***R****equirements* ***S****tate* ***M****achine* ***L****anguage*).

CASE – Kompiuterių padėtas programinės įrangos projektavimas (***C****omputer-****A****ided* ***S****oftware* ***E****ngineering*).

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

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

JAVA, C/C++ - objektiškai orientuotos programavimo kalbos.

C# - *Microsoft* kompanijos sukurta programavimo kalba.

SIMAS – ***SIM****ulation of the* ***A****ggregate* ***S****ystems.*

UPAAL – ***UPP****sala* ***AAL****borg*.

SPIN - įrankis, skirtas analizuoti loginį paskirstytų sistemų nuoseklumą.

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

ADPRO – ***A****utomatic* ***D****ifferentiation* ***PRO****gram.*

KTU – Kauno Technologijos universitetas.

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

ISO – tarptautinė standartų organizacija.

XML – praplečiama žymėjimo kalba (e**X**tensible **M**arkup **L**anguage).

9. PRIEDAI

9.1. 1. PRIEDAS. Agregatinės specifikacijos XML failo pavyzdys

```
<Model Name="Egzample" xmlns="http://tempuri.org/ObjectModel.xsd">
  <AggregateList>
    <Aggregate>
      <Name>HUB</Name>
      <InputList>
        <Input>
          <Name>s1</Name>
          <Value>0</Value>
        </Input>
        <Input>
          <Name>s2</Name>
          <Value>0</Value>
        </Input>
      </InputList>
      <OutputList>
        <Output>
          <Name>out1</Name>
          <Value>0</Value>
        </Output>
        <Output>
          <Name>out2</Name>
          <Value>0</Value>
        </Output>
      </OutputList>
      <DiscreteVariableList>
        <DiscreteVariable>
          <Name>state1</Name>
          <Value>0</Value>
        </DiscreteVariable>
        <DiscreteVariable>
          <Name>state2</Name>
          <Value>0</Value>
        </DiscreteVariable>
      </DiscreteVariableList>
      <ContinuousVariableList>
        <ContinuousVariable>
          <Name>par_gen</Name>
          <Value>0.1</Value>
        </ContinuousVariable>
      </ContinuousVariableList>
      <InternalEventList>
        <InternalEvent>
          <Name>gen_par</Name>
          <ContinuousVariableName>par_gen</ContinuousVariableName>
          <HOperator>
            <Name>H_gen_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>par_gen := [exprnd]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_2</Name>
                <Express>state1 := [state1]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_3</Name>
                <Express>state2 := [state2]</Express>
              </Expression>
```

```
            </ExpressionList>
          </HOperator>
          <GOperator>
            <Name>G_gen_par_int</Name>
            <ExpressionList>
              <Expression>
                <Name>G_exp_1</Name>
                <Express>out2 := [1 WHEN ( ( state1 != 0 ) AND ( state2 = 0
) ); 0 WHEN ( ( state2 = 1 ) OR ( state1 = 0  ) )]</Express>
              </Expression>
              <Expression>
                <Name>G_exp_2</Name>
                <Express>out1 := [1 WHEN (state1 = 0 ); 0 WHEN (state1 = 1
)]</Express>
              </Expression>
            </ExpressionList>
          </GOperator>
        </InternalEvent>
      </InternalEventList>
      <ExternalEventList>
        <ExternalEvent>
          <Name>atejo_s1</Name>
          <InputName>s1</InputName>
          <HOperator>
            <Name>H_atejo_s1_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>state1 := [s1]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
        </ExternalEvent>
        <ExternalEvent>
          <Name>atejo_s2</Name>
          <InputName>s2</InputName>
          <HOperator>
            <Name>H_atejo_s2_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>H_exp_1</Name>
                <Express>state2 := [s2]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
        </ExternalEvent>
      </ExternalEventList>
    </Aggregate>
    <Aggregate>
      <Name>VAS_1</Name>
      <InputList>
        <Input>
          <Name>in</Name>
          <Value>0</Value>
        </Input>
      </InputList>
      <OutputList>
        <Output>
          <Name>sign</Name>
          <Value>0</Value>
        </Output>
      </OutputList>
      <DiscreteVariableList>
        <DiscreteVariable>
```

```
            <Name>buf</Name>
            <Value>0</Value>
          </DiscreteVariable>
          <DiscreteVariable>
            <Name>buf_max</Name>
            <Value>3</Value>
          </DiscreteVariable>
          <DiscreteVariable>
            <Name>dirba</Name>
            <Value>0</Value>
          </DiscreteVariable>
        </DiscreteVariableList>
        <ContinuousVariableList>
          <ContinuousVariable>
            <Name>par_apt</Name>
            <Value>999998</Value>
          </ContinuousVariable>
        </ContinuousVariableList>
        <InternalEventList>
          <InternalEvent>
            <Name>apt_par</Name>
            <ContinuousVariableName>par_apt</ContinuousVariableName>
            <HOperator>
              <Name>H_apt_par_int</Name>
              <ExpressionList>
                <Expression>
                  <Name>expr1</Name>
                  <Express>par_apt := [exprnd WHEN (buf != 0 ); 999999 WHEN
(buf = 0 )]</Express>
                </Expression>
                <Expression>
                  <Name>H_exp_3</Name>
                  <Express>dirba := [0 WHEN (buf = 0 ); 1 WHEN (buf != 0
)]</Express>
                </Expression>
                <Expression>
                  <Name>H_exp_2</Name>
                  <Express>buf := [buf WHEN (buf = 0 ); buf - 1 WHEN (buf !=
0 )]</Express>
                </Expression>
              </ExpressionList>
            </HOperator>
            <GOperator>
              <Name>G_apt_par_int</Name>
              <ExpressionList>
                <Expression>
                  <Name>G_exp_1</Name>
                  <Express>sign := [0 WHEN (buf != buf_max ); 1 WHEN (buf =
buf_max )]</Express>
                </Expression>
              </ExpressionList>
            </GOperator>
          </InternalEvent>
        </InternalEventList>
        <ExternalEventList>
          <ExternalEvent>
            <Name>atejo_par</Name>
            <InputName>in</InputName>
            <HOperator>
              <Name>H_atejo_par_ext</Name>
              <ExpressionList>
                <Expression>
                  <Name>H_exp_1</Name>
```

```
                <Express>par_apt := [exprnd WHEN ( ( buf = 0 ) AND ( sign =
0  ) AND ( dirba = 0 ) AND ( in = 1 ) ); par_apt WHEN ( ( buf != 0 ) OR (
sign = 1 ) OR ( dirba = 1 ) OR ( in = 0) )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_2</Name>
                <Express>buf := [buf WHEN ( ( buf = buf_max ) OR ( in = 0 )
OR ( dirba = 0  ) OR ( sign = 1  ) ); buf + 1 WHEN ( ( buf != buf_max ) AND
( in = 1 ) AND ( sign = 0  ) AND ( dirba = 1 ) )]</Express>
              </Expression>
              <Expression>
                <Name>H_exp_3</Name>
                <Express>dirba := [dirba WHEN (in = 0 ); 1 WHEN (in = 1
)]</Express>
              </Expression>
            </ExpressionList>
          </HOperator>
          <GOperator>
            <Name>G_atejo_par_ext</Name>
            <ExpressionList>
              <Expression>
                <Name>G_exp_1</Name>
                <Express>sign := [1 WHEN ( ( sign = 0 ) AND ( in = 1 ) AND
( buf = buf_max  ) ); sign WHEN ( ( sign != 0 ) OR ( in = 0 ) OR ( buf !=
buf_max  ) )]</Express>
              </Expression>
            </ExpressionList>
          </GOperator>
        </ExternalEvent>
      </ExternalEventList>
    </Aggregate>
  </AggregateList>
  <ConnectionList>
    <Connection>
      <InputAggregateName>VAS_1</InputAggregateName>
      <InputName>in</InputName>
      <OutputAggregateName>HUB</OutputAggregateName>
      <OutputName>out1</OutputName>
    </Connection>
  </ConnectionList>
</Model>
```