VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Programų sistemų, kuriose yra komponentų su būsenomis, architektūrų tyrimas taikant agentais grįstą modeliavimą.

An agent-based research of software architectures that includes stateful components

Magistro baigiamasis darbas

Atliko:	Audrius Siliūnas	(parašas)
Darbo vadovas:	dr. Karolis Petrauskas	(parašas)
Recenzentas:	doc. dr. Audronė Lupeikienė	(parašas)

Vilnius – 2016

Padėka

Darbo autorius dėkoja darbo vadovui dr. Karoliui Petrauskui už patarimus, pastabas bei nuolatinę pagalbą viso magistro darbo rengimo metu.

Santrauka

Programų sistemos architektūros klaidos, dėl kurių kokybinės charakteristikos netenkina keliamų nefunkcinių reikalavimų yra sudėtingai sprendžiamos vėlyvuose sistemos kūrimo etapuose, todėl svarbu kuo anksčiau kokybines sistemos charakteristikas įvertinti kiekybiškai. Vienas iš vertinimo būdų tai sistemų architektūrų modeliavimas. Šiame darbe aprašoma metodika, kuri nurodo kaip atsižvelgiant į sistemos būsenas modeliuoti programų sistemų architektūras. Remiantis susijusios literatūros apžvalga darbe pateikiama: sistemos būsenų klasifikacija, kitų programų sistemų modeliavimo praktikų apžvalga, siūloma modeliavimo metodika. Taip pat darbe pateikiama pagal siūlomą metodiką sukurta simuliacijos sistema bei atliktas metodikos validavimas. Darbo pabaigoje pateikiami programų sistemų sukurtų pagal „SOA" ir „EDA" architektūrinius šablonus modeliavimo ir simuliavimo pavyzdžiai. Vadovaujantis darbe pateikiama metodika galima palyginti šias programų sistemų architektūrų savybes: greitaveiką, klaidų tikimybę, resursų panaudojimą.

Raktiniai žodžiai: modeliavimas agentais, programų sistemų architektūra, programų sistemų architektūrų vertinimas, programų sistemų kokybinės charakteristikos.

Summary

Quantitive software architecture quality attributes evalation in early software creation stages is important, because problems related to software quality attributes are hard to solve in the late stages. Software modelling is one of the evaluation methods. A technique, that describes how to evaluate software architecture quality attributes using an agent based model, is proposed in this thesis. Based on literature review, a classification of states and a quick review of other software modeling practices, detailed description of proposed technique is present in this paper. Furthermore, it is shown how to build simulation system. Also, a validation based on a simple system is present in this paper. To provide more clear understanding, there are several ”SOA” and ”EDA” software architectures modelling examples. Provided technique allows to evaluate these software architecture characteristics: throughput, error-prone, resources utilization.

Keywords: agent based modelling, software architecture, software architecture evaluation, software quality attributes.

TURINYS

ĮVADAS ... 6

1. PROGRAMŲ SISTEMŲ ARCHITEKTŪRA ... 8
   1.1. Architektūros apibrėžimai ... 8
   1.2. Architektūros aprašymo būdai ... 9
   1.3. Architektūros stiliai ... 10
   1.4. Pavyzdiniai modeliai ... 10
   1.5. Architektūrų šablonai ... 11

2. PROGRAMŲ SISTEMŲ KOMPONENTAI ... 12

3. BŪSENOS ... 14
   3.1. Būsenos apibrėžimas ... 14
   3.2. Komponentą apimanti būsena ... 15
   3.3. Sistemą apimanti būsena ... 15
   3.4. Vartotoją apimanti būsena ... 16

4. PROGRAMŲ SISTEMŲ ARCHITEKTŪROS VERTINIMAS ... 17
   4.1. Petri tinklai ... 17
   4.2. Markovo procesai ... 17
   4.3. Eilių sudarymo teorija ... 18
   4.4. Empiriniais tyrimais paremtas vertinimas ... 18
   4.5. Programų sistemų architektūrų modeliavimas ir simuliavimas ... 20

5. PROGRAMŲ SISTEMŲ KOKYBĖS CHARAKTERISTIKOS ... 22
   5.1. Tiesioginės savybės ... 22
   5.2. Architektūrinės savybės ... 23
   5.3. Gaunamos savybės ... 23
   5.4. Nuo naudojimo priklausančios savybės ... 24
   5.5. Nuo sistemos aplinkos priklausančios savybės ... 24

6. MODELIAVIMAS AGENTAIS ... 25
   6.1. Agento apibrėžimas ... 25
   6.2. Agentų ryšiai ... 25
   6.3. Agentų aplinka ... 26
   6.4. Modelio kūrimas ... 26

7. PROGRAMŲ SISTEMOS MODELIAVIMAS AGENTAIS ... 28
   7.1. Principinis vertinimo modelis ... 28
   7.2. Modelio elementai ... 30
      7.2.1. Komponentai ... 30
      7.2.2. Komponento ryšiai ... 31
      7.2.3. Sistemos resursai ... 32
      7.2.4. Sistemos aplinka ... 33
      7.2.5. Apibdendrinimas ... 35
   7.3. Būsenos modeliavimas ... 35
      7.3.1. Komponentą apimančios būsenos modeliavimas ... 35
      7.3.2. Sistemą apimančios būsenos modeliavimas ... 38
      7.3.3. Vartotoją apimančios būsenos modeliavimas ... 38

8. SIMULIACIJOS ĮGYVENDINIMAS 39
8.1. Simuliacijos sistema 39
8.1.1. Architektūros generavimas 40
8.1.2. Simuliacijos vykdytojai 41
8.2. Simuliacijos vykdymas 42
8.2.1. Simuliacijos sistemos inicializavimas 43
8.2.2. Simuliacijos vykdymas 43
8.2.3. Rezultatų skaičiavimas 44
9. METODIKOS VALIDAVIMAS 45
9.1. Pavyzdinė sistema 45
9.2. Pavyzdinės sistemos modelis 46
9.3. Rezultatų palyginimas 46
10. METODIKOS PANAUDOJIMO PAVYZDŽIAI 48
10.1. Architektūros šablonas „SOA” 48
10.1.1. Architektūros generavimas 49
10.1.2. Rezultatai 50
10.2. Architektūros šablonas „EDA” 53
10.2.1. Architektūros generavimas 53
10.2.2. Rezultatai 55
10.3. Architektūros šablonų „SOA” ir „EDA” palyginimas 57
10.3.1. Architektūrų generavimas 57
10.3.2. Bandymas: kintantis lygiagrečių užklausų kiekis komponente 57
10.3.3. Bandymas: kintanti tikimybė, jog komponentas yra sugedęs 58
10.3.4. Bandymas: kintantis užklausų generavimo dažnis 59
11. REZULTATAI 61
12. IŠVADOS 62
LITERATŪRA 63
PRIEDAI 66
1 priedas. Komponento vieno simuliacijos žingsnio sekų diagrama 67

Įvadas

Bet kokios programų sistemos vienas pirmųjų kūrimo etapų yra architektūros projektavimas. Iki šiol nėra vieningai sutarto programų sistemos architektūros apibrėžimo, akademinėje literatūroje galima rasti įvairių bandymų jį formuluoti [FRF+03]. Juos galima apibendrinti Bass, Clement ir Kazman suformuluotu teiginiu, jog programų sistemos architektūra tai struktūra sudaryta iš smulkesnių struktūrų, kurios susideda iš programinių elementų, bei sąryšių tarp jų [BCK03]. Kuriant programų sistemos architektūrą taikomi įvairūs architektūriniai stiliai. Šiuo metu žinomų stilių pavyzdžiai: klientas - serveris, įvykiais grįstas, segmentinis, komponentinis, sluoksnių, paslaugų ir kiti [GS93]. Kiekvienas architektūrinis stilius pasižymi skirtingomis kokybinėmis savybėmis ir padeda spręsti įvairias problemas. Dažnai programų sistemos architektūra remiasi kelių stilių rinkiniu.

Programų sistema formaliai apibrėžiama reikalavimų specifikacijos dokumente. Jame yra nurodomi sistemos funkciniai ir nefunkciniai reikalavimai. Dažnai techninio išsilavinimo neturintiems užsakovams svarbiausi atrodo funkciniai reikalavimai, nes jais tiesiogiai apibrėžiama kokias užduotis programų sistema galės atlikti [Zsc10]. Tačiau tiek akademinėje visuomenėje, tiek programų sistemų kūrėjų tarpe vis dažniau kalbama apie nefunkcinių reikalavimų svarbą, kurie dažnai ilgoje perspektyvoje tampa net svarbesni nei funkciniai. Nefunkciniai reikalavimai leidžia vertinti visą sistemą bendrai, kitaip negu funkciniai reikalavimai apibūdina tik specifinę sistemos dalį. Nefunkcinius reikalavimus apibrėžia kokybiniai kriterijai, tokie kaip: prieinamumas, greitaveika, saugumas, pasiekiamumas, atsekamumas, plečiamumas ir kiti [BBC+04]. Priimti programų sistemos architektūriniai sprendimai, siekiant tenkinti iškeltus kokybinius kriterijus, dažnai sunkiai keičiami tolimesniuose programų sistemos kūrimo etapuose.

Kiekybiškai įvertinti sistemos kokybinius kriterijus dažniausiai galima tik tuomet, kai sistema jau sukurta. Tam atliekami įvairūs sistemos apkrovimo ir naudojimo simuliacijos testai. Atrastas programų sistemos architektūros problemas sudėtinga išspręsti vėlyvuose sistemos kūrimo etapuose, todėl ieškoma būdų kaip galima pagerinti kokybinių kriterijų įvertinimą dar ankstyvoje sistemos kūrimo stadijoje. Dažnai siūlomi šie būdai: įvertinimas scenarijumi, modeliavimas, matematinis modeliavimas, ekspertinis vertinimas. Šiame darbe naudojamas agentais grįstas programų sistemos modeliavimas.

Modeliuojant programų sistemą agentais siekiama sukurti supaprastintą, tačiau išlaikiusį pagrindines savybes, programų sistemos modelį, kuriame sąveiką tarp sistemos komponentų modeliuojama kaip bendravimas tarp agentų [Nia11]. Toks modelis simuliacijos metu leidžia išmatuoti priklausomybes tarp kuriamos programų sistemos kokybinės charakteristikos ir agentų parametrų, bendravimo intensyvumo bei aplinkos kurioje agentais egzistuoja.

Bendra agentais grįstas modeliavimas plačiai taikomas įvairiuose taikomuosiuose tyrimuose: nusikaltimų analizėje, biomedicinos, žemės ūkio, ekologijos ir kitose srityse [MN05]. Taip pat yra keletas bandymų kurti programų sistemos agentais grįstą modelį. Toks modelis kuriamas kuomet sistema yra projektavimo stadijoje. Tačiau nėra vieningai pripažintos metodikos nusakančios kaip reikėtų programų sistemą modeliuoti, o vėliau ir simuliuoti jos veikimą. Literatūroje randamos metodikos nenusako kokiomis gairėmis reikia remtis modeliuojant programų sistemos būsenas.

Magistro darbo tema – programų sistemų, kuriose yra komponentų su būsenomis, architektūrų tyrimas taikant agentais grįstą modeliavimą.

Darbo tikslas – sukurti metodiką, kuri nusakytų kaip atsižvelgiant į sistemos būsenas modeliuoti ir simuliuoti programų sistemos architektūrą taikant agentais grįstą modeliavimą.

Sukurta metodika suformuluos gaires kaip atsižvelgiant į būsenas kurti programų sistemos modelį, bei simuliuoti tokios sistemos veikimą. Tai leis sistemos projektavimo etape įvertinti sistemos kokybines savybes.

Modeliuojant programų sistemos architektūrą, sunku absoliučiai įvertinti daugumą jos parametrų, nes tai yra tam tikri vertinimai, o ne išmatuotos reikšmės. Todėl architektūrų modeliavimas ir simuliavimas labiau tinka skirtingų architektūrų palyginimui negu vienos architektūros vertinimui.

Pagal sukurtą metodiką sudarytas sistemos modelis simuliacijos metu leis patikrinti šias kokybines sistemos charakteristikas:

- klaidų tikimybė,
- sistemos greitaveika,
- resursų panaudojimas.

Tikslui pasiekti keliami šie **uždaviniai**:

1. Išnagrinėti agentais grįstą modeliavimo metodiką.
2. Apžvelgti programų sistemų modeliavimo praktikas.
3. Išnagrinėti programų sistemų būsenų specifiką.
4. Sukurti metodiką, kuri apibrėš, kaip atsižvelgiant į sistemos būsenas modeliuoti ir simuliuoti programų sistemą.
5. Atlikti metodikos validavimą.
6. Pateikti keletą sukurtos metodikos taikymo pavyzdžių.

Laukiami rezultatai:

1. Sukurta programų sistemų modeliavimo ir simuliavimo agentais metodika, kuri atsižvelgia į sistemos būsenas.
2. Atliktas metodikos validavimas.
3. Pateikti metodikos pritaikymo pavyzdžiai.

1. Programų sistemų architektūra

1.1. Architektūros apibrėžimai

Per paskutinius du dešimtmečius programų sistemų architektūra sistemų inžinerijos srityje tapo labai svarbi. Praktikai pripažino, kad gera architektūra sistemos kūrime yra svarbus sėkmės faktorius, o įvairių architektūrinių stilių taikymas padeda tenkinti nefunkcinius reikalavimus. Dabar išleista daug knygų šia tematika, vyksta reguliarios konferencijos, organizuojami užsiėmimai bei skatinami moksliniai tyrimai. Siekiama sukurti standartinius sprendimus, kurie ateityje padėtų išspręsti pasikartojančias programų sistemų inžinerijos problemas. Sritis, nagrinėjanti architektūrą, nepaisant pastebimo progreso, vis dar nebrandi.

Nors egzistuoja daug įvairių sistemos architektūros apibrėžimų, tačiau nėra vieningai pripažintos nuomonės. Dauguma apibrėžimų pažymi, kad tai programų sistemos struktūra sudaryta iš smulkesnių struktūrų, kurios susideda iš programinių elementų, bei sąryšių tarp jų. Ši struktūra atspindi aukšto lygio dizaino sprendimus, kurie nusako kaip sistemos komponentai sujungti, kaip sąveikauja tarpusavyje bei kokie yra pagrindiniai sąveikos keliai. Sprendimai apibūdina komponentų ir visos sistemos parametrus. Dažnai architektūros aprašas leidžia atlikti aukšto lygio sistemos analizę. Gerai sukurta architektūra gali padėti užtikrinti sistemai keliamus funkcinius ir nefunkcinius reikalavimus, tokius kaip našumas, patikimumas, plečiamumas ir vientisumas. Bloga architektūra gali kliudyti įgyvendinti iškeltus reikalavimus [Gar14].

Egzistuojančius programų sistemos architektūros apibrėžimus galima suskirstyti į tris pagrindines grupes [Sol12] :

- *Aukšto lygio sistemos abstrakcija.* Pavyzdžiui Zhang ir Goddard sistemos architektūrą apibrėžia kaip aukšto abstrakcijos lygio struktūrą, sistemos elgsenos apibūdinimą, bei nefunkcinių reikalavimų įvardinimą [ZG05]. Tokio pobūdžio apibrėžimai iškelia keletą svarbių klausimų: kas yra aukšto lygio abstrakcija, kaip galima užtikrinti nefunkcinių reikalavimų tenkinimą nagrinėjant sistemą tik aukštame abstrakcijos lygmenyje?

- *Sistemos struktūra ir iš išorės matomi sistemos parametrai.* Šie apibrėžimai neįtraukia detalių apie sistemos funkcionalumą. Apibrėžimuose akcentuojama komponentai, ryšiai tarp jų bei komponentų ryšys su sistemos aplinka. Vienas iš pavyzdžių tai Bass, Clements ir Katzman apibrėžimas [BCK03]: programų sistemos architektūra – struktūra ar struktūros kurios apibūdina sistemos komponentus, iš išorės matomus parametrus bei ryšius tarp jų. Šiuose apibrėžimuose taip pat kyla neaiškumų: kas yra sistemos komponentas, kurie sistemos elementai yra architektūriniai, kurie skirti tenkinti dalykinės srities reikalavimus?

- *Koncepciniai sistemos sprendimai ir apribojimai.* Jais remiantis organizuojamas sistemos kūrimo procesas ir programavimo darbai. Labiausiai paplitęs šio tipo apibrėžimas – *IEEE (Institute of Electrical and Electronics Engineers)* pateikiama sąvoka. Ji sistemą apibūdina kaip fundamentalių sprendimų ar sistemos parametrų rinkinį, įgyvendinamą per apibrėžtoje aplinkoje esančius sistemos komponentus ir ryšius tarp jų. Dažnai šiuose apibrėžimuose akcentuojami architektūriniai stiliai, pavyzdžiui: vamzdžių ir filtrų stilius, sluoksniuotos

sistemos stilius ir kiti. Tačiau tokio tipo apibrėžimuose išlieka neapibrėžtas skirtumas tarp architektūrinių, dalykinės srities ar funkcinių sistemos elementų.

Nepaisant apibrėžimų klasių skirtumų, visi apibrėžimai įtraukia sistemos komponentus ir ryšius tarp jų kaip svarbius programų sistemų architektūrinius elementus. Šiame darbe programų sistemų architektūra laikysime: fundamentalių sprendimų ar parametrų rinkinį, kuris yra įgyvendinamas per apibrėžtoje aplinkoje esančius sistemos komponentus bei ryšius tarp jų.

1.2. Architektūros aprašymo būdai

Vienas svarbių klausimų kylančiu programų sistemų architektams – kokiu būdu aprašyti sukurtą sistemą. Aprašymas turi ne tik atspindėti svarbiausius sistemos aspektus, bet būti suprantamas suinteresuotiems asmenims, būti lengvai vertinamas ir analizuojamas, nereikalauti daug kaštų bei būti lengvai keičiamas. Pasak [Gar14] šiuo metu egzistuojančius sistemos apibūdinimo būdus galima suskirstyti į tris pagrindines grupes:

- *Neformalus apibūdinimas*. Tai grafiniai vaizdai skirti apibūdinti sistemą. Dažnai jie sukuriami plačiai naudojamomis taikomosiomis programomis kaip „PowerPoint" ar „Visio". Tokie vaizdai papildomi laisva forma rašomu tekstu. Toks sistemos apibūdinimas nereikalauja specialių ekspertinių žinių, todėl jį galima lengvai ir greitai sukurti. Neformalus apibūdinimas turi nemažai minusų. Pavyzdžiui laisva forma kuriami grafiniai vaizdai neturi apibrėžtos ir bendrai pripažintos semantikos, todėl skirtingų asmenų gali būti skirtingai interpretuojami. Neformaliai apibrėžtai architektūrai negali būti atlikta formali sistemos korektiškumo ir vientisumo analizė, o apibrėžti sistemos apribojimai yra sunkiai kontroliuojami sistemos kūrimo procese.
- *Pusiau formalus apibūdinimas*. Šiuo būdu sistemos aprašomos naudojant bendruosius modeliavimo simbolius. Tokiam aprašymo būdui trūksta detalios semantikos, tačiau palaikomas standartiniuose įrankiuose esantis vaizdinis žodynėlis. Pusiau formalaus apibūdinimo grupei priklauso ir viena populiariausių bendrosios paskirties modeliavimo kalbų – UML. Tai dažniausiai praktikoje naudojama kalba skirta architektūrai apibūdinti. Programuotojams palanku, kad kai kurie UML įrankiai palaiko glaudų ryšį su objektinio programavimo paradigma. Deja, šiuo būdų neįmanoma išreikšti kai kurių architektūrinių principų.
- *Formalus apibūdinimas*. Susidūrus su problemomis, kylančiomis dėl prieš tai išvardintų grupių trūkumų, buvo pasiūlyta architektūrą aprašyti formalia kalba, kuri leistų vykdyti detalią sistemos analizę. Bendrai šios kalbos vadinamos „Architecture Description Language" (ADL). Tokios kalbos turi konkrečią sintaksę, kuri leidžia atvaizduoti, kompiliuoti, analizuoti ir modeliuoti programų sistemų architektūras.

Sudėtingėjant sistemoms, rinkoje matomas vis didesnis formalių architektūros apibūdinimo būdų populiarumas. Tokia tendencija susidarė dėl didėjančio architektūros modeliavimo svarbumo programų sistemos inžinerijoje. Tai indikuoja vis didesnę srities brandą.

Programų sistemų architektūra dažnai aprašoma naudojant tarpusavyje susijusias perspektyvas, kurios nusako funkcines ir nefunkcines savybes. Apibrėžiant architektūras naudojami įvairios perspektyvos. Kruchten 1995 metais pasiūlė naudoti šias „4+1“ perspektyvas [Kru95]: loginį, procesų, sistemos išdėstymo, įgyvendinimo bei naudojimo scenarijų. *IEEE* standartas apibendrino šią idėja pasiūlydamas kitas perspektyvas. Knygoje [RW05] siūlomos šios perspektyvos: įgyvendinimo, informacinė, lygiagrečių procesų, konstravimo, išdėstymo ir operacinė. Modeliuojant dažnai vadovaujamasi įgyvendinimo ir išdėstymo perspektyvomis. Įgyvendinimo perspektyva aprašo sistemos funkcinius elementus, jų atsakomybes, interfeisus ir komunikaciją. Įgyvendinimo perspektyva dažnai yra pagrindinė architektūros aprašo dalis. Šioje perspektyvoje išskirti elementai turi esminę įtaką keičiamumo, našumo, saugumo kokybės charakteristikoms. Išdėstymo perspektyva aprašo aplinką, kurioje sistema bus įdiegta. Ši perspektyva specifikuoja aparatinę įrangą ir jos sąsajas su programų sistemos elementais [RW05].

1.3. Architektūros stiliai

Tai koordinuotas architektūrinių ribojimų rinkinys, nusakantis taisykles kurias turi tenkinti to stiliaus programų sistemos architektūriniai elementai bei ryšiai tarp jų [Fie00]. Kitaip tariant, tai stiliaus apibrėžti ribojimai, kuriais remiantis galima apibrėžti architektūrų šeimas. Viena plačiausiai naudojamų kliento-serverio stiliaus šeima. Šis stilius nusako, kad architektūra turi įtraukti du pagrindinius elementus – serverį ir klientą, bei jų sąveikos ribojimus. Šiuo atveju užklausas gali generuoti klientas, o į jas atsakyti serveris. Stilius nenusako, kokio dydžio sistemos dalys ar konkretūs artefaktai turi būti įtraukti į architektūrą. Tiek klientai, tiek serveriai gali būti sistemos komponentai, posistemės ar maži objektai. Taip pat stilius nenusako kokiu protokolu sistemos dalys bendrauja, tai priklauso nuo konkrečios sistemos realizacijos. Šiuo metu rinkoje yra daug kliento-serverio stiliaus architektūros sistemų, tačiau jos visos skiriasi viena nuo kitos. Svarbu paminėti, jog architektūrinis stilius nėra sistemos architektūra, tai tik ribojimų rinkinys, kuriuos turi tenkinti architektūra.

Kadangi programų sistemų architektūra nusako koncepcinius sprendimus, kurie tenkina tiek funkcinius ir nefunkcinius reikalavimus, sunku tiesiogiai lyginti dviejų sistemų architektūras. Lyginimas sudėtingas netgi tos pačios sistemos architektūros skirtingose aplinkose. Architektūros stiliai vienas iš būdų apibūdinti ir palyginti architektūras. Skirtingi stiliai apibūdina bendruosius sistemos dalių sąveikavimo principus, akcentuoja sąveikos esmę [Sha90].

Kiekvienas architektūrinis stilius pasižymi tam tikromis žinomomis kokybinėmis charakteristikomis [BCK03]. Architektas, kurdamas programų sistemą, būtent dėl atitinkamų stiliaus charakteristikų dažnai pasirenka vieną ar kitą stilių. Pavyzdžiui, vieni stiliai padeda pagerinti našumą, kiti padidina saugumą.

1.4. Pavyzdiniai modeliai

Pavyzdinis modelis nusako kaip funkcionalumas yra padalintas tarp sistemos dalių, bei kokie duomenų srautai šias dalis sieja. Dažnai modeliai naudojami sprendžiant žinomas dalykines prob-

lemas. Kai kurios dalykinės sritys turi žinomus pavyzdinius modelius, kurie padeda projektuojant sistemas. Svarbu paminėti, jog tokie modeliai nenusako kokie konkretūs sistemos elementai turės realizuoti priskirtą funkcionalumą. Gali nutikti taip, jog skirtingą funkcionalumą įgyvendins vienas sistemos komponentas ar atvirkščiai, vieną funkcionalumą įgyvendins keli komponentai [BCK03].

1.5. Architektūrų šablonai

Kaip [BCK03] teigiama, architektūros šablonas – aukščiau minėtas pavyzdinis modelis susietas su architektūriniais elementais. Architektūrinis šablonas pasako, kurie architektūriniai elementai bus atsakingi už konkretų funkcionalumą, kokiais duomenų srautais jie apsikeis. Siejant pavyzdinį modelį su architektūriniu šablonu, galima daryti susiejimą vienas su vienu, bet tai nebūtina. Taigi, architektūrinis šablonas tam tikroje dalykinėje srityje duoda funkcionalumo padalinimą, sistemos padalinimą į architektūrinius elementus bei jų tarpusavio susiejimą. Patys šablonai savyje neturi logikos, tačiau pateikia infrastruktūrą, kurioje logika gali būti sukurta ir vykdoma. Infrastruktūra sukurta pagal tam tikrą architektūrinį šabloną yra vadinama karkasu.

Apibendrinant svarbu paminėti, kad nei architektūriniai stiliai, nei pavyzdiniai modeliai, nei architektūrų šablonai nėra architektūra. Jie yra mažiau detalūs nei architektūros, tačiau naudingi sistemos projektavimo pradžioje, nes nusako funkcionalumo ir struktūros skaidymą į smulkesnes dalis.

2. Programų sistemų komponentai

Programų sistemų inžinierių tarpe dažnai naudojamas terminas „programinis komponentas“, tačiau ne visuomet jis suprantamas vienodai. Taip yra dėl to, kad vystantis programų sistemų inžinerijos sričiai, buvo sukurta daug skirtingų programinio komponento apibrėžimų ir požiūrių į jį. Apibrėžimus galima apibendrinti Szyperski apibrėžimu: programinis komponentas tai apibrėžtas vienetas turintis įvardintus interfeisus bei apibrėžtas priklausomybes [Szy02]. Toks komponentas gali būti nepriklausomai diegiamas į sistemą, tačiau tai neprivaloma savybė. Kitaip tariant programinis komponentas tai nestipria sankiba pasižyminti sistemos dalis, kuri joje esantį funkcionalumą slepia nuo išorinių komponentų, su jais bendrauja tik per apibrėžtus interfeisus. Komponento priklausomybė nuo išorės išreiškiama per privalomus ir aiškiai apibrėžtus interfeisus. Toks komponentas neatliekant papildomos konfigūracijos gali būti nepriklausomai diegiamas į sistemą.

Bendrai apie komponentus pradėta kalbėti siekiant kurti iš skirtingų savarankiškų programinių komponentų surinktas sistemas. Taip yra daroma elektronikos sistemose. Kuriant tokias sistemas svarbios dvi sritys: universalių ir diegiamų į įvairias sistemas komponentų kūrimas; antroji, sistemų sudarytų iš atskirų komponentų konstravimas. Pasak Szyperski komponentas turi pasižymėti šiomis savybėmis [Szy02]:

- *Tai nepriklausomai kuriamas vienetas.* Komponentą galima perduoti kitai sistemai kaip atskirai supakuotą artefaktą.
- *Komponentą gali naudoti trečios šalys.* Komponentai turi būti kuriami ne konkrečiai sistemai, o skirti perpanaudoti skirtinguose ir nepriklausomose sistemose.
- *Neturi iš išorės matomos būsenos.* Komponentas turi užtikrinti, kad tas pats komponentas, esant tokiomis pačiomis sąlygomis, atliks tokį patį funkcionalumą.
- *Komponentas turi kontraktu apibrėžtus interfeisus ir aiškias priklausomybes.*
- *Komponentas gali būti diegiamas į apibrėžtą aplinką.* Dažnai komponentai naudojasi infrastruktūros paslaugomis, todėl komponentas kuriamas darant prielaidą jog platforma kurioje jis bus įdiegtas komponentui suteiks reikalingas paslaugas.

Be išvardintų savybių komponentas turi gebėti pateikti komponento egzempliorius. Minėtos savybės neužtikrina jog bus sukurtas perpanaudojamas komponentas. Programiniai komponentai atitinkantys šias savybes gali būti naudojami tik specifinėse technologinėse aplinkose. Kuriant komponentines sistemas svarbu užtikrinti dvi praktikas: pirma, turi būti metodas, kuris apibrėžia kokio tipo komponentai kuriami, kaip jie komponuojami, kaip panaudojami; antra, technologija turi palaikyti tokį metodą [SM08].

Komponentas tai abstraktus terminas, nenusakantis komponento dydžio. Vadovaujantis objektinio programavimo paradigma, tai gali būti viena klasė, klasių rinkinys ar klasių paketas. Komponentas gali būti ir mažesnis nei klasė. Komponentu laikysime ir kompleksinį komponentą sudarytą iš mažesnių komponentų. Komponentas komponentais grįstose architektūrose dalinai gali

būti prilyginamas objektui objektinio programavimo paradigmoje. Čia objekto elgsena gali būti atspindėta objekto būsenų modeliu, kur elgsena tai perėjimo nuo vienos prie kitos būsenos atitikmuo. Būsenos keitimas iškviečiamas išorės dirgiklių, šiuo atveju metodų iškvietimu. Kaip ir objektai per savo gyvavimo laikotarpį keičia įvairias būsenas, taip ir komponentai komponentinėse architektūrose keičia būsenas. Į komponentą atkeliaujančios užklausos iškviečia jo metodus, o šie pakeičia komponento būseną. Net jei komponentas sudarytas iš skirtingų savas būsenas turinčių objektų, būsenos modelis apima visą komponentą [SM09].

Šiame darbe naudosime ne tokią griežtą komponento sąvoką kaip aptartosios. Programiniu komponentu laikysime nestipria sankiba pasižyminčią sistemos dalį, kurioje esantis funkcionalumas slepiamas nuo išorinių programinių komponentų, o sistemos komponentai bendrauja tik per apibrėžtus interfeisus.

3. Būsenos

Dėl sistemos veiklos susiformavusios būsenos ir programinės įrangos elgsenos priklausomybė nuo jos konfigūracijos, tai vienos svarbiausių, darančių įtaką modelio tikslumui, savybių. Todėl kuriant sistemos modelį svarbu atsižvelgti į sistemos būseną. Pasiektas didesnis tikslumas paskatintų programinės įrangos kūrėjus modeliavimą naudoti ankstyvoje kūrimo stadijoje, kad nustatyti sistemos nefunkcines savybes. Komponento būsena dažnai yra išreiškiama saugomame atribute, juos galima rasti daugumoje komponentinių sistemų. Komponentai su būsenomis, servisai su būsenomis, ar sistemos su būsenomis tai dažnai naudojami terminai, tačiau iki šiol nėra bendrai pripažinto apibrėžimo nusakančio kokia ir kur saugoma informacija atspindi būseną. Dar daugiau, nėra bendrų metodikų nusakančių kaip reikia modeliuoti būseną kuriamuose sistemos modeliuose.

Komponentinės programinės įrangos inžinerijos srityje kokybinių savybių nustatymo metodikos dažnai remiasi individualių komponentų kokybinėmis savybėmis [BGM+06] [Koz10]. Būsena svarbus, tačiau ne vienintelis, faktorius nulemiantis sistemos kokybinės savybes. Konkreti komponentų implementacija, aplinka kurioje jie egzistuoja, sistemos naudojimo scenarijai, tai keletas kitų faktorių.

Modeliuojant programinės įrangos sistemas, kuriose yra komponentų su būsenomis pirmiausia svarbu identifikuoti ir apibrėžti sistemoje atsirandančias būsenas, išskirti darančias įtaką sistemos kokybinėms savybėms. Čia susiduriama su trimis iššūkiais:

- *Būsenos apibrėžimas*. Būseną atspindinti informacija gali atsirasti įvairiuose sistemos artefaktuose, komponentuose. Tai gali įvykti skirtingose sistemos gyvavimo stadijose. Šiuo metu nėra bendrai pripažintų metodikų kurios nusakytų kaip lokalizuoti ir klasifikuoti sistemoje atsirandančią informaciją [Koz10][BHK06].
- *Įtakos sistemos kokybinėms savybėms nustatymas*. Literatūroje vis dar diskutuojama ir nėra bendros nuomonės kaip reikėtų interpretuoti būsenos įtaką sistemos veikimui [BHK06].
- *Modelio detalumo lygio nustatymas*. Didelėse sistemose yra daug komponentų kuriuose gali susidaryti būsena, todėl modeliuojant dideles sistemas svarbu nustatyti modelio detalizavimo kaštus. Turi būti rastas balansas tarp modelio detalumo ir dėl to pasiekiamos naudos. Rekomenduojama nusistatyti modeliuojamos informacijos abstrakcijos lygį.

3.1. Būsenos apibrėžimas

Būsena – informacija atsiradusi dėl sistemos veiklos, kuri yra saugoma sistemoje bei daro įtaką tolimesnei sistemos veiklai. Dažnai sistemos ateities žingsniai priklauso nuo būsenos, todėl ir resursų poreikis aptarnaujant sistemą priklauso nuo būsenos. Dėl skirtingo resursų panaudojimo skiriasi ir sistemos kokybinės charakteristikos. Svarbu paminėti, jog šiame darbe būsena laikoma tik sistemos atributuose saugoma išreikštinė informacija. Tokią būseną galima išreikštinai nustatyti ir nuskaityti sistemos veikimo metu.

Šiame darbe būsenos apibrėžimas neįtraukia neišreikštos sistemos vykdymo būsenos. Pavyzdžiui, informacija nusakanti konkretų sistemos vykdymo žingsnį algoritme nelaikoma būsena.

Toks požiūris taikytas keliose literatūroje nagrinėtų modeliavimo metodikų [FB] [GGM+08].

Būsenos gali būti skirstomis į įvairias kategorijas ir vertinamos skirtingomis dimensijomis. Darbe [HBR14] būsenos suskirstytos remiantis dvejomis dimensijomis: apimties ir laiko. Apimties dimensijoje išskirtos trys kategorijos: komponento, sistemos ir vartotojo. Laiko dimensijoje išskiriamos taip pat trys kategorijos: Inicijavimo, išdėstymo, ir vykdymo metu atsirandanti būsena. Šiame darbe modeliuojama tik vykdymo metu atsirandanti būsena.

1 lentelė. Komponentų būsenų tipai

Dimensijos	Vykdymas	Išdėstymas	Inicijavimas
Komponentas	a) Protokolo būsena b) Vidinė būsena	c) Priskyrimo būsena (pvž. resursų)	d) Konfigūracija
Sistema	e) Globali būsena	f) Priskyrimo būsena (pvž. resursų)	g) Konfigūracija
Vartotojas	h) Sesijos būsena i) Nuolatinė būsena		

3.2. Komponentą apimanti būsena

Tai informacija saugoma kiekviename komponente, ji naudojama koreguoti komponento atsaką į pateikiamas užklausas. Šią būseną gali koreguoti tik pats komponentas, t.y. jo metodai. Tokios būsenos negali pakeisti kiti komponentai.

a) Protokolo būsena – informacija apie galimybę priimti tam tikro pobūdžio užklausas. Dažnai tai yra komponento interfeiso dalis. Pavyzdžiui: būsena nusako, jog komponentas gali priimti tik skaitymo užklausas, o užklausos reikalaujančios rašymo į duomenų bazę turi būti atmestos. Bendriausiu atveju būsena gali nusakyti, ar komponentas einamuoju momentu iš vis gali priimti užklausas.

b) Vidinė būsena – paties komponento nustatyta viduje saugoma informacija. Ši informacija naudojama komponento veiklai koreguoti. Kitaip tariant kitas komponento žingsnis tampa priklausomas nuo vidinės būsenos. Tokia informacija yra matoma iš išorės, tačiau ją keisti turi teisę tik pats komponentas. Pavyzdžiui: būsena nusakanti ar komponentas veikia suspaustu ar įprastu rėžimu. Suspaustas rėžimas nustatomas kuomet duomenų bazėje lieka nedaug laisvos vietos, tuomet komponentas pateikiamas užklausas duomenų bazei papildomai suspaudžia.

c) Priskyrimo būsena – išdėstymo metu nustatoma informacija apie aplinką (infrastruktūrą). Pavyzdžiui: nustatomas maksimalus lygiagrečių užklausų vykdymo skaičius. Dažnai ši būsena priklauso nuo serverio, kuriame įdiegta sistema, pajėgimų.

d) Konfigūracija – inicijavimo metu priskiriama bei komponento parametrus nusakanti informacija. Pavyzdžiui: lygiagrečių užklausų vykdymo strategija.

3.3. Sistemą apimanti būsena

Tai visiems komponentams prieinama informacija, kuri yra naudojama koordinuoti bendrą sistemos veiklą.

e) Globali būsena – visiems sistemos komponentams vykdymo metu prieinama informacija. Pavyzdžiui: užklausų pateiktų į sistemą skaitkliukas, kuris pasiekus tam tikrą ribą sužadina atsarginės duomenų kopijos darymą.

f) Priskyrimo būsena – išdėstymo metu sukuriama bei visiems sistemos komponentams prieinama informacija. Pavyzdžiui: informacija apie sistemai prieinamus trečių šalių servisus.

g) Konfigūracija – sistemos paleidimo metu nustatomi bei pasiekiami visiems sistemos komponentams parametrai. Pavyzdžiui: maksimalus komponentų egzempliorius skaičius ar sistemos versiją nusakantis parametras.

3.4. Vartotoją apimanti būsena

Tai informacija, kuri sistemoje saugoma kiekvienam vartotojui atskirai, ir naudojama koreguoti sistemos veiklą kiekvienam vartotojui individualiai.

h) Sesijos būsena - vienos sesijos metu vienam vartotojui sukuriama informacija. Ši informacija yra pašalinama kartu su sesija. Pavyzdžiui: vienos sesijos metu elektroninėje parduotuvėje saugoma informacija apie pirkinių krepšelį.

i) Nuolatinė būsena - vartotojo sukurta ir nepriklausomai nuo vartotojo sesijos visą sistemos gyvavimo laiką saugoma informacija. Pavyzdžiui: Konkretaus vartotojo tam tikros elektroninės paslaugos išnaudotas kiekis.

4. Programų sistemų architektūros vertinimas

Programų sistemos architektūros vertinimo metu, sistemos kokybinės charakteristikos yra įvertinamos kiekybine išraiška. Vertinimui naudojami skirtingi būdai, tokie kaip Markovo procesai [WPC06] [ST07], eilių sudarymo teorija [BD98], Petri tinklai [FS02] ir kiti. Bendrai vertinimo būdus galima skirstyti į matematiniu modeliavimu, skaičiavimais, simuliacijomis ar kompiuteriniais eksperimentais grįstas vertinimo technikas.

4.1. Petri tinklai

Tai matematinio modeliavimo kalba skirta paskirstytų sistemų aprašymui. Petri tinklas tai kryptinis dvišalis grafas, kuriame viršūnės vaizduoja perėjimus ir vietas. Perėjimai – sistemoje galimi įvykiai, tinkle jie dažniausiai vaizduojami stulpeliais. Vietos tai sąlygos, dažniausiai vaizduojamos apskritimais. Viršūnes jungia lankai-rodyklės. Jos parodo kurios grafo viršūnės yra išankstinės sąlygos, o kurios yra taikomos po įvykio. Petri tinklai turi tikslius vykdymo semantikos matematinius apibrėžimus, gerai išvystytą matematine procesų analizės teoriją.

K. Jensesn papildydamas standartinius Petri tinklus pasiūlė spalvotus Petri tinklus. Standartiniai elementai, vietos, perėjimai ir rodyklės buvo papildyti spalvų, sargų bei išraiškų koncepcijomis. Taikant spalvotų Petri tinklų modeliavimą straipsnyje [FS02] tiriamos programų sistemos kokybinės savybės: saugumas, našumas ir patikimumas. Autoriai pažymi, jog neturintiems patirties Petri tinklų modeliavime gali būti sudėtinga sukurti tinkamą programų sistemos modelį. Tai specialių įgūdžių ir pasiruošimo reikalaujantis procesas. Modeliuojant sistemą Petri tinklais sunku tirti kompleksiškas ir didelės apimties sistemas. Tokias sistemas atspindintis Petri tinklas sunkiai išsprendžiamas, todėl dažnai modelis turi būti supaprastinamas ir modeliuojamas didesniame abstrakcijos lygmenyje. Tokiu atveju iškyla pavojus į modelį neįtraukti svarbios informacijos apie modeliuojamą sistemą, o tai darytų įtaką galutiniams rezultatams [CH10].

4.2. Markovo procesai

Tai vieni geriausiai ištirtų ir įvairiose mokslo srityse taikomų tikimybių teorijos procesų. Taikant Markovo procesus tiriama sistema aprašoma būsenomis ir galimais atsitiktiniais perėjimais tarp jų. Markovo procesų esminė savybė ta, kad proceso ateitis priklauso tik nuo dabartinės būsenos, bet nepriklauso nuo praeities. Dalyje inžinerinių sistemų vykstantys procesai yra Markovo procesai. Visos Markovo proceso tikimybinės charakteristikos ateityje priklauso tik nuo to, kokioje būsenoje šis procesas yra dabartiniu laiko momentu. Kai laikas per kurį įvyksta procesas yra suskaičiuojamas, kitaip tariant kai tai baigtinis procesas, jis vadinamas Markovo grandine. Informacijos teorijos mokslo pradininkas K.Shanonas teigė, kad daugumą komunikacinių sistemų galima modeliuoti Markovo procesais. Straipsnyje [WPC06] tiriamas programų sistemos patikimumas taikant Markovo procesų modeliavimą, o vėlesniame [ST07] straipsnyje įtraukta ir našumo bei saugumo analizė. Markovo procesai turi keletą trūkumų: pirmiausia norint gauti patikimus rezultatus reikia žinoti specifinę informaciją, kaip perėjimo galimybes ar kiekybines komponentų charakteristikas;

modelis neįtraukia modeliuojamos sistemos aplinkos, modeliuojama tik konkretūs architektūriniai komponentai.

4.3. Eilių sudarymo teorija

Tai matematinis metodas nagrinėjantis įvairiose sistemose susidarančias laukimo eiles. Eilių sudarymo teorija remiasi prielaida, kad eilių ilgis ir laukimo laikas gali būti nuspėjamas ar apskaičiuojamas. Dažnai modeliuojant sistemas kuriami eilių tinklai, kuriais juda įvairūs objektai. Dalis autorių teigia, jog sistemos našumas dažniausiai tiriamas taikant eilių sudarymo teoriją [BD98]. Tikriausiai tokia nuomonė susidariusiu, nes našumas tai akivaizdžiausia sistemos charakteristika, kuri gali būti lengvai modeliuojama naudojant eilių tinklo modelį.

Pavyzdžiui [Het10] remiamasi idėja, jog kiekvienas sistemos komponentas turi individualias našumo charakteristikas. Šios dažniausiai apibrėžiamos matematinėmis funkcijomis. Vienam komponentui siunčiant užklausą ar duomenų srautą į kitą komponentą gali būti daroma įtaka kito komponento įvesties spartai, o tai gali daryti įtaką komponento našumui. Kitaip tariant, komponentų našumas gali priklausyti nuo kitų komponentų našumo. Kadangi kiekvieno komponento našumas yra išreiškiamas matematine funkcija, ją nesunkiai galima įstatyti į kito komponento našumo funkciją, o šią į kitą komponento ir t.t. Todėl galima apskaičiuoti geriausius ir blogiausius sistemos našumo scenarijus. Tačiau autoriai pažymi, jog iškyla sunkumas kuomet turime uždarą priklausomybę, nes nėra būdo apdoroti ciklines priklausomybes. Problemai spręsti pasitelkiama eilių sudarymo teorija, modeliuojamos į komponentus siunčiamų užklausų eilės, bei ieškoma vidutinio užklausos apdorojimo tinkle laiko. Eilių sudarymo teorija leidžia pakankamai lengvai ir gerai įvertinti sistemos našumą, tačiau neturi gerų galimybių vertinti kitų kokybinių kriterijų kaip saugumas, patikimumas.

4.4. Empiriniais tyrimais paremtas vertinimas

Minėti vertinimo būdai kokybinius kriterijus kiekybiškai įvertina remiantis matematiniais skaičiavimais. Tai galutiniuose rezultatuose leidžia pasiekti aukštą tikslumo lygį. Kitų autorių atliktos studijos parodė, jog modeliuojant vadovaujantis aptartais būdais susiduriama su apribojimais [STV11]. Tokiu būdu sukurtuose modeliuose sunku imituoti tikrame gyvenime kylančius sistemos naudojimo scenarijus. Be to, šie vertinimo būdai nėra greitai suprantami taikomosios srities atstovams, kurie dažnai nėra matematinių mokslų specialistai.

Dažni vertinant programų sistemų architektūras remiamasi matematiniais skaičiavimais. Kita vertus, šiuo metu tampa vis svarbesni empiriniais tyrimais paremti būdai. Tokiems tyrimams dažniausiai kuriami sistemos prototipai, modeliai, kuriuose atliekamos simuliacijos, kompiuteriniai eksperimentai ar bandymai, stebimos kokybinės charakteristikos. Tokio tipo technikos reikalauja papildomų pastangų ir laiko kuriant sistemos programinį modelį. Vienas iš vertinimo būdų tai modelių kūrimas RAPID modeliavimo kalba. Tai bendros paskirties architektūros aprašymo kalba, kuri taip pat turi galimybę modeliuoti komponentų interfeisus bei iš išorės matomą elgseną [NHS08] Tai leidžia aptikti pažeidimus interfeisuose, komponentų jungimo korektiškumą ir anali-

zuoti našumą išskirstytose sistemose. RAPID kalba sukurtas modelis yra tinkamas vykdyti įvykiais grįstą simuliaciją. Ši kalba taip pat leidžia modeliuoti vykdymo metu dinamiškai kintančias architektūras.

Modeliuojant programų sistemas plačiai naudojamas diskrečių įvykių simuliavimo būdas. Lyginant su RAPID tai paprastesnis ir labiau plečiamumu pasižymintis būdas. Šis naudojamas jau virš 40 metų įvairiose mokslo srityse [SMG+10]. Tai į procesus orientuotas simuliavimo būdas. Modelis kuriamas remiantis sistema kaip visuma. Modeliuojant ji yra skaidoma į mažesnius komponentus. Sistemos veikla simuliuojama kaip diskrečių įvykių seka laike. Kiekvienas simuliacijos įvykis įvyksta konkrečiu laiko momentu pakeisdamas sistemos būseną. Laikoma, jog tarp konkrečių laiko momentų negali įvykti jokie sistemos pokyčiai, todėl sistemos būsenos pokyčiai yra aiškiai nustatomi laike.

Remiantis diskrečių įvykių simuliavimo sprendimais, ankstyvaisiais 1990 metais pristatytas agentais grįstas modeliavimas. Pastarojo techninis įgyvendinimas panašus į diskrečių įvykių modeliavimo įgyvendinimą. Straipsnio [SMG+10] autoriai rekomenduoja naudoti agentais grįstą modeliavimą kuomet:

- tiriama dinamiška sistema,
- sistemai keliami reikalavimai dažnai kinta,
- agentai dalyvauja bendrame sistemos strateginių veiksmų nustatyme,
- kuomet svarbus modelio plečiamumas.

Autoriai pažymi, jog renkantis tyrimo ir modeliavimo būdą, svarbiau atsižvelgti į nagrinėjimą problemą nei į taikymo sritį. Pažymima, jog agentais grįsto modeliavimo idėja panaši į objektinio programavimo paradigmą, todėl specialistams, dirbantiems objektinio programavimo srityje, greičiau išmokstama.

Straipsnyje [SMG+10] pateikiamas šių dviejų modeliavimo būdų palyginimas, išryškinami pagrindiniai skirtumai.

2 lentelė. Agentais grįsto ir diskrečių įvykių modelių palyginimas

Diskrečių įvykių modelis	**Agentais grįstas modelis**
Taikomas „iš viršaus į apačią” modeliavimas.	Taikomas „iš apačios į viršų” modeliavimas.
Modeliuojama detalizuojant sistemą	Modeliuojama individualios esybės ir jų bendravimas
Už sistemos koordinavimą atsakinga viena gija. Centralizuotas valdymas	Kiekvienas agentas turi savo koordinavimo giją. Decentralizuotas valdymas
Pasyvios esybės. Sprendimų priėmimas modeliuojamas kaip sistemos veiklos dalis	Aktyvios esybės. Kiekviena esybė gali savarankiškai priimti sprendimus ir imtis veiksmų
Eilės yra pagrindinis modelio elementas	Neturi eilių koncepcijos
Elgsena modeliuojama makro lygmenyje	Makro elgsena ne modeliuojama, ji atsiranda priimant sprendimus individualiems agentams

Apibendrinant modeliavimo būdų palyginimą, autoriai pažymi jog nepaisant išskirtų skirtumų jie yra panašūs. Diskrečių įvykių modeliavimą papildžius pro-aktyviomis, autonominėmis

esybėmis gautume labai panašų būdą į agentais grįstą modeliavimą. Literatūroje vis dar trūksta vieningos ir aiškios nuomonės apie šių dviejų modeliavimo būdų skirtumą.

4.5. Programų sistemų architektūrų modeliavimas ir simuliavimas

Straipsnyje [AO99] nagrinėjami įvairūs programų sistemos kokybinių charakteristikų vertinimo būdai, tarp jų ir šiame darbe taikomas agentais grįstas modeliavimas. Autorius pažymi, jog kuriant modelį kuriame bus vykdoma simuliacija, turi būti sumodeliuoti sistemos komponentai ir sistemos veikimo aplinka. Kuriant modelį tik abstrakčiai apibrėžiamas architektūrinių vienetų elgesys ir sąveika, tačiau tai vis vien reikalauja specialios sistemos analizės. Ši analizė inžinieriui kartais gali padėti pastebėti ir funkcinius neatitikimus anksčiau nei tradiciniai funkcinių savybių tikrinimo būdai. Kuomet modelis yra sukurtas, jame yra simuliuojami įvairūs sistemos naudojimo scenarijų profiliai. Simuliacijos metu stebimi apibrėžti kokybiniai atributai. Autorius siūlo šiuos simuliacija grįsto vertinimo žingsnius:

Modeliavimas

- *Apibrėžti ir įgyvendinti aplinką.* Pirmiausia reikia apibrėžti sistemos sąveika su ją supančia aplinka, apibrėžti interfeisus per kuriuos sistema bendrauja su ja. Vėliau reikia nuspręsti kokia sąveika iš apibrėžtų turėtų būti simuliuojama. Čia svarbu pasirinkti tinkamą aplinkos abstrakcijos lygį, nes kuo didesnė abstrakcija, tuo daugiau detalių neįtraukiama į modelį, atsiranda pavojus sukurti sistemos neatspindintį modelį. Pavyzdžiui, inžinierius turi nuspręsti ar į modelį įtraukti laiką, trunkantį nuo sistemai inicijuoto aplinkos impulso iki sistemos reakcijos į šį impulsą. Dažniausiai modelio abstrakcijos lygis nusprendžiamas pagal siekiamą kokybės atributų tikslumo lygį.

- *Įgyvendinti architektūrinius komponentus.* Kuomet sistemos aplinka apibrėžta ir įgyvendinta, modeliuojami architektūriniai komponentai. Šiame etape svarbiausia apibrėžti komponentų sąveiką, interfeisus ir jungimą. Dažnai šios savybės yra aprašomos architektūriniame apraše. Komponentų elgsena ir atsakas į užklausas gautas per interfeisus gali būti apibrėžti ne taip detaliai kaip patys interfeisai. Remiantis tiriamais kokybės atributais, architektas nusprendžia kokiame abstrakcijos lygyje modeliuojama komponento elgsena. Pavyzdžiui, tiriant sistemos našumą svarbu nuspręsti kokių operacijų vykdymo laikas įtraukiamas į modelį.

- *Įgyvendinti scenarijų profilį.* Remiantis tiriamais kokybės atributais sukuriamas sistemos naudojimo profilis. Lyginant su aplinkos ir architektūrinių komponentų modeliavimu, tai lengvesnis etapas. Architektas turi sumodeliuoti atskirus sistemos naudojimo profilius ir turėti galimybę kiekvieną iš jų simuliacijos metu atskirai aktyvuoti. Taip pat gali prireikti vykdyti atsitiktinius naudojimo scenarijus neapibrėžtą laiko tarpą.

Simuliavimas

- *Simuliuoti sistemą ir inicijuoti naudojimo profilį.* Šiame etape architektas jau turi turėti parengta sistemos modelį, kuriame bus vykdoma simuliacija. Priklausomai nuo tiriamo koky-

bės atributo vykdomi konkretūs arba atsitiktiniai naudojimo scenarijai. Šie gali būti aktyvuojami rankiniu arba automatiniu būdu. Straipsnio autorius pažymi, jog kiekvienam tiriamam kokybės atributai vykdomi skirtingi specialiai tam sukurti naudojimo scenarijai. Kai kuriems atributams gali tekti naudoti keletą scenarijų vienu metu.

- *Įvertinti kuriamos sistemos kokybės charakteristikas.* Paskutiniame etape analizuojami duomenys surinkti simuliacijos metu, remiantis jais įvertinamos kuriamos sistemos kokybinės charakteristikos. Priklausomai nuo charakteristikos architektas turi nuspręsti koks duomenų kiekis reikalingas analizei. Dažnai dideli duomenų kiekiai surinkti simuliacijos metu yra apibendrinami pačiame simuliacijos įrankyje. Pavyzdžiui išvedamas komponento ar sistemos atsako laiko vidurkis.

Rezultatų tikslumas priklauso nuo kelių skirtingų veiksnių: pirmiausia nuo modelio abstrakcijos lygio; antra, nuo naudojimo scenarijų profilių tikslumo; galiausiai, nuo to kiek sumodeliuota aplinka tiksliai atitinka realaus pasaulio aplinką. Bendrai, simuliacija grįstas vertinamas leidžia atlikti naudojimo scenarijus kuriamos sistemos modelyje, surinkti duomenis apie sistemos veiklą bei remiantis jais įvertinti sistemos kokybės charakteristikas.

5. Programų sistemų kokybės charakteristikos

Vertinant programų sistemų kokybę įtraukiami įvairūs kokybės atributai. Iki šiol nėra vieningai pripažinto šių atributų klasifikavimo. Dažnai remiamasi programų inžinerijos produkto kokybės modeliu ISO/IEC 9126, jame išskiriamos šešios pagrindinės kokybės charakteristikos: funkcionalumas, patikimumas, panaudojamumas, efektyvumas, palaikomumas, perkeliamumas. Šios vėliau skaidomos į sub-charakteristikas. Šiame darbe siekiama ištirti šias sub-charakteristikas:

- klaidų tikimybę (patikimumo charakteristika),
- greitaveiką,
- resursų panaudojimą (našumo charakteristikos).

Kokybės modelyje pažymima, jog kiekviena sub-charakteristika priklausomai nuo programų sistemos technologijos gali būti apskaičiuojama skirtingai.

M. Larsson savo darbe [Lar04] siūlo kokybės kriterijus klasifikuoti pagal jų sudedamąsias dalis. Klasifikuojant atsižvelgiama kaip apskaičiuojamas bendras sistemos kokybės atributas ir kokie komponentų kokybės atributai įtraukiami. Charakterizuojant sistemą atskiriamos sąvokos „sistema“ ir „komponentų rinkinys“. Pastaroji atspindi sujungtų tarpusavyje komponentų rinkinį. Kai kuriais atvejais atributai apibūdina ne tik „komponentų rinkinį“, bet ir dėl savitos architektūros bei aplinkos kurioje ji yra atsirandančias savybes. Tokiu atveju kokybės atributas apibūdina sistemą.

Išskiriami penki kokybės atributų tipai:

- *Tiesioginės savybės.* Komponentų rinkinio savybė priklausanti tik nuo vienodų atskirų komponentų savybių.
- *Architektūrinės savybės.* Komponentų rinkinio savybė, kuri priklauso nuo sistemos architektūros ir vienodų atskirų komponentų savybių.
- *Gaunamos savybės.* Komponentų rinkinio savybė, kuri priklauso nuo kelių skirtingų komponentų savybių.
- *Nuo naudojimo priklausančios savybės.* Komponentų rinkinio savybė, kuri priklauso nuo naudojimo profilio.
- *Nuo sistemos aplinkos priklausančios savybės.* Savybė, kuri priklauso nuo kitų kokybės atributų bei sistemos aplinkos.

5.1. Tiesioginės savybės

Tai savybės, kurios išreiškiamos priklausomybe nuo tų pačių atskirų komponentų parametrų.

$$R = \{k_i | 1 \leq i \leq n\}, \tag{1}$$
$$S(R) = f(S(k_1), S(k_2),...,S(k_n)) \tag{2}$$

S - savybė, R - komponentų rinkinys, k - komponentas.

Šio tipo savybės gali būti gaunamos tiesiogiai iš atskirų komponentų parametrų. Savybė matoma komponento rinkinio, sistemos bei individualaus komponento lygyje. Kuriant sistemos modelį savybę galima apibrėžti pakankamai tiksliai.

5.2. Architektūrinės savybės

Tai savybės, kurios išreiškiamos priklausomybe nuo sistemos architektūros ir vienodų atskirų komponentų parametrų.

$$R = \{k_i | 1 \leq i \leq n\},$$
$$S(R) = f(S(k_1), S(k_2),...,S(k_n, SA), \tag{3}$$
$$SA \subseteq RxR \tag{4}$$

SA - sistemos architektūra.

Skirtingai nei tiesioginė savybė, ši priklauso ne tik nuo atskirų komponentų vienodų savybių, bet ir nuo sistemos architektūros. Dažnai architektūriniai sprendimai leidžia pagerinti atskirų komponentų savybes nekeičiant pačio komponento.

5.3. Gaunamos savybės

Tai komponentų rinkinio savybė, kuri priklauso nuo kelių skirtingų komponentų parametrų.

$$R = \{k_i | 1 \leq i \leq n\},$$
$$S(R) = f \begin{cases} S_1(k_1), S_1(k_2),...,S_1(k_n), \\ S_2(k_1), S_2(k_2),...,S_2(k_n), \\ \vdots \\ S_m(k_1), S_m(k_2),...,S_m(k_n) \end{cases} \tag{5}$$

S - rinkinio savybė, S_1... S_{m} - komponento savybė.

Taip kaip komponentų rinkinio tiesioginė savybė yra kompleksiškesnė nei paprasčiausia pavienių komponentų parametrų suma, taip ir kai kurios savybes yra priklausomos nuo kelių skirtingų komponento parametrų. Į šią kategoriją patenka savybės, kurios yra matomos sistemos ar

komponento rinkinio lygyje, bet nėra matomos pavienių komponentų lygyje. Modeliuojant, sunku nustatyti tokių savybių priklausomybę nuo atskirų komponentų parametrų.

5.4. Nuo naudojimo priklausančios savybės

Esant tam tikriems naudojimo profiliams pasireiškiančios komponentų rinkinio savybės.

$$S(R,U_k) = f(S(c_i,U`_{i,k})) : i,k \in N \tag{6}$$

S - savybė prie konkretaus naudojimo profilio, U_k - Sistemos rinkinio naudojimo profilis,, $U`_{i,k}$ - c_i komponento naudojimo profilis esant U_k.

Šios savybės priklauso ne tik nuo atskirų komponentų, komponentų rinkinio parametrų, bet ir nuo sistemos naudojimo profilio. Sistemos naudojimo profilis turi būti detalizuojamas iki komponentų naudojimo profilio, dažnai tai sunkiai apibrėžiama aiškia priklausomybės funkcija.

5.5. Nuo sistemos aplinkos priklausančios savybės

Ši savybė priklauso nuo aukščiau išvardintų savybių bei nuo aplinkos, kurioje sistema egzistuoja. Tokios savybės pavyzdys - saugumas.

$$\begin{gathered} R = \{k_i | 1 \leq i \leq n\}, \\ S_k(R,U_k,E_l) = f(S_k(k_1,U`_{1,k}),S_k(k_2,U`_2,k),...,S_k(k_i,U`_i,k),S_l) : i,k,l \in N \end{gathered} \tag{7}$$

U_k - Sistemos rinkinio naudojimo profilis, E_l - Aplinka, $U`_{i,k}$ - Komponento naudojimo profilis.

M. Larsson [Lar04] pažymi, jog tokias savybes modeliuoti ar įvertinti konkrečiais įverčiais beveik neįmanoma.

6. Modeliavimas agentais

6.1. Agento apibrėžimas

Agentais grįsto modeliavimo srityje vis dar nėra vieningos nuomonės kaip turėtų būti apibrėžiamas agentas. Tai moksliniuose straipsniuose vis dar aptarinėjama tema. Galima rasti daug agento apibrėžimų, daugumoje jų išskiriamos panašios savybes. Vis dar nesutariama kokiomis išskirtinėmis savybėmis turėtų pasižymėti agentas modelyje. Vieni autoriai teigia, jog agentu galima laikyti bet kokią agentais grįsto modelio dalį, kiti teigia jog agentu galime vadinti tas modelio dalis, kurios reaguoja į aplinką. Tačiau apibendrinant daugumos autorių nuomonę galime išskirti keletą agentams būdingų savybių [MN05]:

- *Autonomiškumas*. Agentas yra autonomiškas ir pats save valdo. Tai yra bendraujanti, savarankiška esybė, kuri funkcionuoja nepriklausomai ją supančioje aplinkoje. Agento elgseną galima apibrėžti kaip procesą. Jo metu, per agento jutiklius iš aplinkos gauta informacija yra apdorojama, o vėliau remiantis ja agentas priima sprendimus apie tolimesnę savo veiklą.

- *Moduliariškumas*. Agentas yra moduliarus. Agentas yra identifikuojama, diskreti ir gebanti priimti sprendimus esybė. Ji pasižymi savitu elgesiu, turi charakteristikų ar atributų rinkinį.

- *Sociališkumas*. Agentas yra socialus. Tai su kitais agentais sąveikaujanti esybė. Modeliuojant programų sistemas, dažnai agentų sąveikavimo protokolai nusako resursų prašymo, trikių išvengimo, kitų agentų atpažinimo, sąveikos ir informacijos apsikeitimo taisykles bei kitus modeliuojamai sričiai būdingus mechanizmus.

- *Sąlygiškumas*. Agentas turi laike kintančią būseną. Būsena susidaro dėl agento praeities veiksmų bei aplinkos poveikio jam. Agento būsena išreiškiama jame saugomais atributais ar atributų rinkiniais. Agento veiksmai priklauso nuo būsenos, taigi kuo būsena yra kompleksiškesnė ir detalesnė tuo platesnis agento elgsenos spektras.

Agentai gali turėti ir papildomų savybių, tačiau jos nėra būtinos. Agentai dažnai turi tikslus. Siekdami įgyvendinti juos, agentai atitinkamai koreguoja ir savo veiksmus. Agentas gali gebėti mokytis, remtis įgyta patirtimi, todėl jis privalo turėti atmintį. Ji dažniausiai išreiškiama dinamiškai keičiamais kintamaisiais.

6.2. Agentų ryšiai

Vienas pagrindinių elementų agentais grįstame modelyje yra ryšiai tarp agentų. Pirmiausia reikia nustatyti agentų tarpusavio ryšius bei jų kitimą laike. Svarbu identifikuoti draudžiamus ryšius. Visi modelio agentai sudaro tam tikras topologijas. Dažniausiai sutinkamos topologijos [MN05]:

- *Sriuba*. Modelis be erdvės, jame agentas neturi atributų išreiškiančių jo padėti erdvėje.

- *Tinklelis*. Agentų ryšiai vaizduojami kaip tinklelis. Tokiame modelyje agentai gali bendrauti tik su savo kaimynais.

- *Euklidinė erdvė*. Agentai išdėstomi plokštumoje arba trimatėje erdvėje.
- *Geografinės informacinės sistemos*. Agentai susieti su realistiniu geografiniu reljefu ir juda juo.
- *Tinklai*. Tinklai gali būti statiniai ir dinaminiai. Statiniai tinklai, tai tokie tinklai kuriems ryšiai nustatomi iš anksto, o dinaminių tinklų ryšiai yra kintantys laike.

Nesvarbu kokia agentų topologija panaudota modelyje, idėja, jog duotuoju laikotarpiu agentai gali bendrauti tik su ribotu kiekiu kitų agentų yra svarbiausia. Šiame darbe ryšiai tarp modeliuojamos sistemos dalių modeliuojami taikant tinklų topologiją.

6.3. Agentų aplinka

Vienas iš privalomu agentais grįsto modelio elementų tai agentų aplinka. Dažniausiai ji modeliuojama kaip vienas ar keli savarankiški agentai [HCW+12]. Iš aplinkos gauta informacija daro įtaką agento veiksmams, todėl visiems svarbiems aplinkos objektams turi būti parinktas atitinkamas detalumo lygis. Aplinka kaip ir agentai gali turėti laike kintančią būseną. Pavyzdžiui serverių, kuriuose veikia programinė įranga, apkrovos pasikeitimai, arba tinklo srauto intensyvumo pokyčiai.

6.4. Modelio kūrimas

Straipsnyje [MN05] nagrinėjama kaip reikėtų kurti agentais grįstus modelius. Pažymima, jog bendrai modeliuojant agentais reikėtų remtis tais pačiais principais kaip ir kuriant modelius kitais būdais. Pirmiausia identifikuojamas modelio tikslas, kitaip tariant apibrėžiami klausimai į kuriuos kuriamas modelis turėtų padėti atsakyti. Toliau, atliekama tiriamosios srities sisteminė analizė, kurios metu identifikuojamos sistemos dalys, sąveika tarp jų, informacijos šaltiniai ir srautai.

Agentais grįstas modeliavimas skirtingai nei tradiciniai modeliavimo būdai akcentuoja modeliuojamą esybę. Be tradicinių modeliavimo žingsnių modeliuojant agentais papildomai reikia:

- Identifikuoti agentus ir nustatyti jų elgseną.
- Identifikuoti agentų ryšius ir nustatyti vykstančias sąveikas.
- Pasirinkti modeliavimo platformą ir strategiją.
- Atlikti papildomą agentų elgsenos validaciją.
- Nustatyti ryšius tarp pavienių agentų ir bendros sistemos elgsenos.

Tikslus veiklų ir sąveikų nustatymas, atidus agentų identifikavimas leidžia sukurti detalų modelį. Dažnai sprendimus gebantys priimti sistemos vienetai yra modeliuojami kaip atskiri agentai. Tai gali būti trivialios sistemos dalys ar savo sprendimų priėmimo strategiją turinčios kompleksinės posistemės. Svarbu tiksliai aprašyti sistemos dalių elgseną, todėl reikia atlikti dalykinės srities analizę ir pasirinkti tinkamą elgsenos aprašymo būdą. Dalykinės analizės metu identifikuojama ryšiai ir vykstančios sąveikos.

Sukurtą teorinį modelį galima realizuoti naudojant įvairius įrankius. [MN05]:

- Įvairiuose įrankiuose įdiegta *Visual basic for application* programavimo kalba. Pavyzdžiui „Microsoft Excel”.
- Specialiai agentais grįstam modeliavimui skirtomis programomis.
- Programomis skirtomis atlikti bendruosius matematinius skaičiavimus.
- Specializuotos agentų kūrimo programomis. Pavyzdžiui: „Repast”, „Swarn”, „MASON”, „AnyLogic”.
- Bendromis programavimo kalbomis.

Visual basic for application programavimo kalba įgalina kasdienius įrankius, tokius kaip „Microsoft Excel”, naudoti modeliavimui. „Excel” darbo knyga gali tapti paprasčiausia ir greičiausiai išmokstama modeliavimo aplinka. Tačiau tokiu būdu sunku modeliuoti sudėtingas, turinčias daug skirtingomis savybėmis pasižyminčių savarankiškų dalių, sistemas. Tokiais atvejais patariama naudoti specialiai agentų modeliavimui skirtas aplinkas, tokias kaip „NetLogo”. Tai viena iš greičiausiai matomų rezultatų pasiekti leidžiančių aplinkų. „NetLogo” yra patobulinta „Logo” programavimo kalbos versija. Nors „NetLogo” pagrindinė paskirtis yra edukacinė, tačiau įrankiu galima kurti plataus spektro programas.

Agentais grįstus modelius taip pat galima kurti ir bendrosios paskirties matematinių skaičiavimų taikomosiomis programomis, tokiomis kaip „MATLAB” ar „Mathematica”. Tačiau šios programos neturi integruotų, specialiai agentų modeliavimui skirtų bibliotekų. Modelio kūrėjas turi savarankiškai suprogramuoti agentams būdingas savybes. Tam pakanka bazinių skriptų rašymo žinių. Naudojantis tokio tipo programomis galima greitai modeliuoti sudėtingai apskaičiuojamas agentų priklausomybes ar kompleksinius agentų atributus. Tai galima atlikti pasinaudojant integruotomis matematinių skaičiavimų bibliotekomis. Turint programavimo patirties, bendrosios paskirties matematinių skaičiavimų programos gali būti gera pradžia modeliuojant agentais.

Dažnai agentų modeliai yra kuriami specializuotomis programomis. Šiuo metu dauguma tokių programų yra nemokomos ir laisvai prieinamos. Pavyzdžiui: „Repast”, „Swarn”, „NetLogo” ar „MASON”. Išsamus programų palyginimas atliktas [SM08] straipsnyje. Jame lyginamos „Java” technologijomis sukurtos agentų modeliavimo programos.

Šiame darbe naudojama nemokama atviro kodo „Repast Simphony” programinė įranga. Ji sukurta bendradarbiaujant Čikagos universitetui ir Oregono nacionalinei laboratorijai. Be standartinio agentų modeliavimo funkcionalumo „Repast Simphony” leidžia vizualiai atvaizduoti ir specifikuoti modelio struktūrą, o gautus rezultatus atvaizduoti grafiškai.

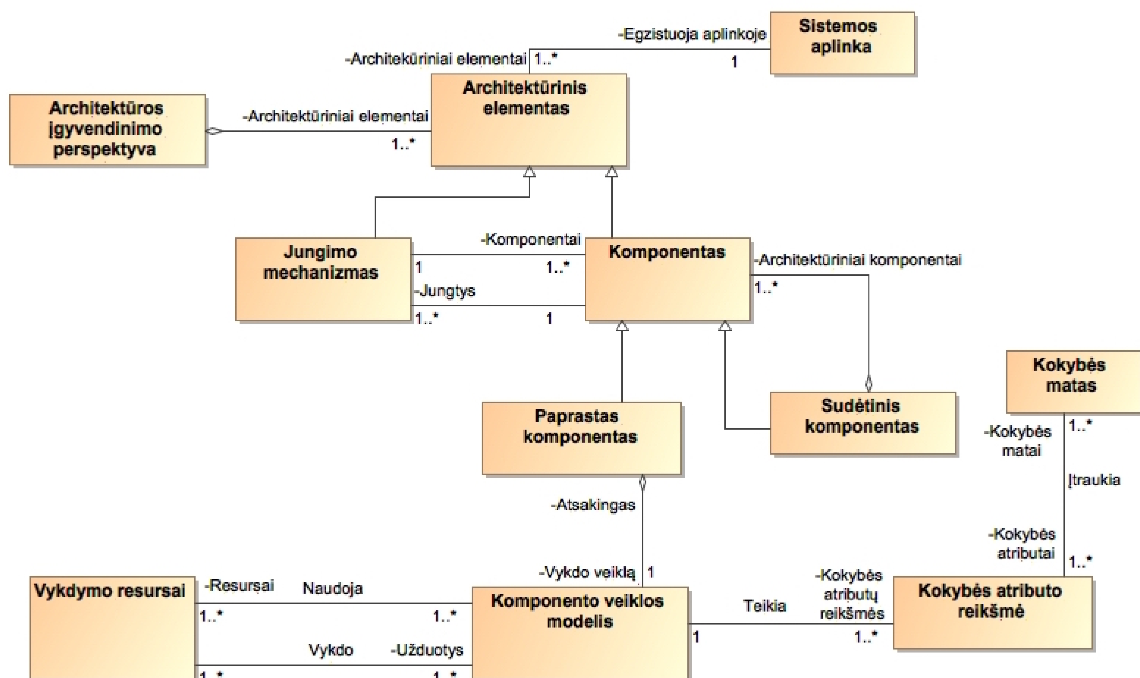
7. Programų sistemos modeliavimas agentais

7.1. Principinis vertinimo modelis

Šiame darbe aptariama metodika skirta vertinti programų sistemos architektūrą projektavimo etape. Šiame etape svarbiausi aukšto lygio architektūriniai sprendimai, dažnai net nežinoma detali sistemos bei jos dalių veikimo logika. Taikant šią metodiką architektas turi galimybę palyginti keletą skirtingų projektinių architektūros sprendimų. Svarbu paminėti, jog gauti absoliutūs kokybinių charakteristikų įverčiai gali ir neatspindėti tikros situacijos, nes metodika skirta palyginti keletą architektūrų.

Programų sistemų architektūros kūrimas tai kompleksiška veikla. Ji reikalauja žinių skirtingais architektūros aspektais, tokiais, kaip: architektūriniai požiūriai, architektūriniai elementai ir ryšiai tarp jų, sistemos kokybės atributai ir jų matai, vykdymo scenarijai. Kuriant architektūrą svarbu žinoti metodikas, kurios padeda atskirose sistemos dalyse suprojektuoti planuojamas vartotojų veiklas. Tokios metodikos aptartos straipsnyje [AM11], viena iš jų – vartotojų panaudos atvejų žemėlapiai (*angl. UCM - Use Case Maps*). Pastaroji, naudojant veiklos modelių ir ryšių sampratas, padeda architektūrinius elementus susieti su funkciniais reikalavimais.

Dažnai, metodikos, skirtos lokalizuoti funkcinius reikalavimus programų sistemose, naudoja bendrinius architektūrinius elementus, tokius kaip: jungimo mechanizmas (ryšys), komponentas, veiklos modelis [Buh99]. Šiais elementais galima aprašyti daugumą programų sistemų architektūrų. Vertinant architektūros kokybines charakteristikas, straipsnyje [BGL14] patariama į šiais elementais aprašytą architektūros modelį įtraukti ir kokybės atributo bei kokybės mato elementus. Remiantis išskirtais elementais sukurtas principinis programų sistemos architektūros modelis skirtas tirti kokybines charakteristikas.



1 pav. Principinis modelis skirtas tirti programų sistemų architektūros kokybines charakteristikas

Prieš kuriant agentais grįstą modelį, modeliuojamą sistemą reikia aprašyti aukščiau minėtais elementais. Tai geriausia daryti remiantis sistemos įgyvendinimo perspektyva (1 pav. Architektūros įgyvendinimo perspektyva). Joje galima identifikuoti pagrindinius architektūrinius elementus. Architektūrinis elementas – abstrakti esybė, kuri atspindi architektūros įgyvendinimo perspektyvoje išskirtas sistemos dalis. Prisilaikant šiame darbe įvestos komponento sąvokos, toliau minėtas sistemos dalis laikysime sistemos komponentais. Modelyje architektūrinį elementą sudaro komponentas ir jungimo mechanizmas. Čia komponentas – abstrakti esybė, ji apibendrina dviejų tipų komponentus: vienas vaizduoja paprastas struktūras (1 pav. Paprastas komponentas), o kitas kompleksiškas struktūras (1 pav. Sudėtinis komponentas). Kiekvieno komponento dalimi laikomas ir jo veiklos modelis. Komponento veikos modelis – tai taisyklių ir funkcijų rinkinys nusakantis kaip komponentas atlieka jam pavestas užduotis. Veiklos modelis įtraukia užklausos apdorojimo veiksmų aprašą, komponento būseną, bei apibrėžia veiklos priklausomybę nuo būsenos. Ryšiai tarp komponentų aprašomi jungimo mechanizme (1 pav. Jungimo mechanizmas). Jei ryšys yra kompleksiškas, jungimo mechanizmas gali turėti savo veiklos modelį. Paprastas komponentas vaizduoja sistemos dalį, kuri egzistuoja programos vykdymo metu ir atlieka gan paprastus veiksmus. Pavyzdžiui, tai gali būti vienas procesas, objektas ar metodas. Sistemos dalis atliekanti sudėtingus veiksmus ir apjungianti kitas sistemos dalis vaizduojama kaip sudėtinis komponentas. Jis gali apjungti tiek paprastus, tiek kitus sudėtinius komponentus. Apibrėžtomis sąvokomis galime aprašyti architektūrą, bei apibrėžti sistemos veiklos principus.

Architektūriniai elementai egzistuoja apibrėžtoje aplinkoje, todėl į modelį įtraukiamas ir sistemos aplinkos elementas. Tai abstrakti samprata, nusakanti kaip programų sistema vykdymo metu yra susijusi su aplinka kurioje egzistuoja. Nors dažnai programų sistemai prieinami vykdymo resursai yra laikomi aplinkos dalimi, šiame principiniame modelyje jie išskiriami kaip atskira dalis. Tokiu būdu siekiama parodyti, jog komponentų veiklą riboja turimi vykdymo resursai.

Galiausiai, siekiant įvertinti kokybines charakteristikas kiekybine išraiška įvedami kokybės atributo ir mato elementai. Kiekvienas kokybės atributas yra tiesiogiai susijęs su vienu ar daugiau kokybės matų. Kokybės matai nustatomi remiantis tiriamomis kokybės charakteristikomis.

Apibrėžus programų sistemos architektūrą aukščiau minėtais konceptualiais architektūriniais elementais, reikalinga juos susieti su elementais naudojamais agentais grįstame modelyje. Remiantis anksčiau darbe įvesta komponento sąvoka, išskirtus paprastus ir sudėtinius komponentus modeliuosime tokia pat viena komponento esybe.

3 lentelė. Ryšys tarp konceptualių architektūros elementų ir agentais grįsto modelio elementų

Architektūros elementai	**Modelio elementai**
Komponentas	Agentas
Komponento veiklos modelis	Agento elgsena
Jungimo mechanizmas	Agentų ryšys
Vykdymo resursai	Agentų aplinka
Sistemos aplinka	Agentų aplinka

7.2. Modelio elementai

Darbe [Mik14] kaip ir šioje metodikoje programų sistema modeliuojama agentais grįstu modeliu. Minėtame darbe pristatoma metodika nusako kokius agentų parametrus reikia naudoti modeliuojant programų sistemą. Žemiau pateikiami agentų parametrai pristatyti minėtame darbe. Modeliavimo metodiniai nurodymai taip pat suformuluoti remiantis pristatyta metodika. Tačiau, darbe [Mik14] neatsižvelgiama į sistemoje atsirandančias būsenas. Jų modeliavimo metodiniai nurodymai pateikiami skyrelyje „Būsenos modeliavimas”.

7.2.1. Komponentai

Aptarti programinio komponento apibrėžimai griežtai nenusako komponento dydžio, todėl ir komponento dydis gali būti pasirenkamas modeliavimo metu. Atsižvelgiant į siekiamus modeliavimo tikslus komponentu galima laikyti tiek visą programų sistemą, tiek vieną kodo eilutę. Bendrai, tai išskirta sistemos dalis. Kuriant sistemos modelį galima pasirinkti sudėtingesnes sistemos dalis modeliuoti detaliau, paprastesnes – ne taip detaliai. Pavyzdžiui standartinė biblioteka kurios nefunkcinės savybės jau yra žinomos gali būti modeliuojama kaip vienas komponentas, o kitos sistemos dalys – atskirais komponentais. Norint gauti kuo tikslesnius rezultatus rekomenduojama sistemą modeliuoti mažesniais komponentais. Bendrai, rekomenduojama kiekvieną išskirtą sistemos dalį modeliuoti atskiru komponentu modelyje.

Komponento naudojami ir pateikiami interfeisai modeliuojami įeinančių ir išeinančių agentų ryšiais. Ryšiai tarp komponentų modeliuojami kaip agento parametrai. Jiems apibrėžti sukuriamas specialus tipas „Ryšys”. Jame apibrėžiamos savitos bendravimo taisyklės, kuriomis vadovaujasi sujungti komponentai. Modeliuojant programinio komponento atliekamą darbą, modelyje komponentui nurodomas resursų kiekis, kuris yra reikalingas modeliuojamą darbą atlikti. Atliekamas darbas nedetalizuojamas, tokiu atveju sistema panašėtų į įgyvendintą modeliuojamą sistemą. Taigi modeliuojant pakanka žinoti darbui atlikti reikalingus resursus.

Dažnai sistemose komponentai vienu metu gali apdoroti lygiagrečiai keletą užklausų, todėl šią savybė taip pat svarbu įtraukti ir į kuriamą modelį. Modeliuojant, komponentui nurodomas maksimalus lygiagrečiai apdorojamų užklausų kiekis ir įvedama taisyklė, jog prieš komponentui pradedant apdoroti užklausą turi būti patikrinta ar maksimalus lygiagrečių užklausų limitas nėra pasiektas.

Darbe [Mik14] pateikiamas agento, vaizduojančio modeliuojamą komponentą, parametrų sąrašas:

- K_k – kviečiamų komponentų kvietimo tipas: *ALL* – kviesti visus sujungtus komponentus, *RANDOM* – kviesti atsitiktinį skaičių komponentų, tačiau maksimaliai K_{kmax}, minimaliai K_{kmin}
- K_{kmax} – maksimalus kviečiamų komponentų skaičius (Jei K_k = *RANDOM*, ne didesnis už vaikinių komponentų skaičių).
- K_{kmin} – minimalus kviečiamų komponentų skaičius (Jei K_k = *RANDOM*).

- K_w – užklausos apdorojimo užlaikymas milisekundėmis. Nurodytą laiko tarpą komponentas lauks nuo užklausos priėmimo iki apdorojimo pradžios. Gali būti reikalinga vaizduojant laukimą iš kitos sistemos ar vartotojo interfeise atliekamo veiksmo laukimą.
- K_l – maksimalus lygiagrečiai apdorojamų užklausų kiekis komponente. Nurodžius 0, lygiagrečių užklausų kiekis neribojamas.
- K_e – klaidos tikimybė apdorojant užklausą. Įvykūs klaidai užklausa pažymima kaip nesėkminga. Reikšmė nurodoma intervale [0 - 1]. Tikimybė 1, reiškia jog visos užklausos bus įvykdytos su klaida ir pažymėtos kaip nesėkmingos.
- K_{dw} – tikimybė, jog komponentas yra išjungtas dėl atliekamų priežiūros darbų, arba tiesiog yra sugedęs. Reikšmė nurodoma intervale [0 - 1]. Tikimybė 1, reiškia, kad komponentas visuomet bus išjungtas ir nepriims jokių užklausų (bus pažymėtos kaip nesėkmingos).

Komponento veiklos modelis yra susietas agregacijos ryšiu su pačiu komponentu. Tai reiškia, jog veiklos modelis yra neatsiejama komponento dalis. Jei modeliuojant yra reikalinga apibrėžti sudėtingesnį veiklos modelį nei standartinį, jis gali būti pakeistas atsižvelgiant į modeliuojamą situaciją. Perrašyti veiklos modelį gali būti reikalinga norint apibrėžti būsenos įtaką komponento veiklai.

7.2.2. Komponento ryšiai

Apibrėžti komponentai yra sujungiami jungimo mechanizmais – ryšiais. Straipsnyje [BGL14] ryšiai yra modeliuojami kaip savarankiškos esybės, tačiau toks būdas neatitinka modeliavimo agentais principų. Todėl darbe [Mik14] nuspręsta sukurti parametro tipą „Ryšys". Jame saugoma nuoroda į komponentą, kuris yra sujungtas su modeliuojamu komponentu. Tipe taip pat apibrėžiami resursai reikalingi perduoti informaciją modeliuojamu ryšiu. Komponento ryšiai modeliuojami kaip tipo „Ryšys" parametrų rinkinys. Šį rinkinį sudaro nuorodos į prijungtus komponentus bei ryšio su jais tipai (asinchroniai ar sinchroniai). Rinkinys yra saugomas modeliuojamame komponente (iš kurio išeina sąryšiai).

Galima modeliuoti dviejų tipų ryšius: sinchroninį ir asinchroninį. Sujungus komponentus sinchroninių ryšiu, agentas išsiunčia užklausą į ryšio gavėją ir laukia atsakymo. Kol atsakymas negaunamas, laikoma, jog užklausa yra neužbaigta. Gavus atsakymą iš kviesto komponento, užklausa pažymima kaip sėkmingai įvykdyta. Asinchroninio ryšio atveju, agentas išsiuntęs užklausą į ryšio gavėją, ją iš kart pažymi kaip įvykdytą ir toliau tęsią savo darbą. Norint modeliuoti situaciją, kuomet išsiuntus asinchroninę užklausą, komponentas turi sulaukti šios užklausos vykdymo pabaigos, įvestas parametras K_{lw}. Reikšmė „true", nurodo, jog, komponentas turi laukti iki bus apdorotos visos jo sukurtos asinchroninės užklausos. Šiuo parametru, galima modeliuoti gijų sujungimą, kuomet komponentas asinchroniškai sukuria keletą užduočių ir turi sulaukti jų pabaigos.

Darbe [Mik14] patariama ryšiams modeliuoti nekurti atskirų esybių, o papildyti agentų, vaizduojančių komponentus, parametrus:

- K_r – komponentų, kurie yra kviečiami vykdant užklausas, sąrašas. Šiame sąraše taip pat nurodoma ir ryšio tipas. Reikšmė „*ASYNC*” reiškia asinchroninį ryšį, o „*SYNC*” sinchroninį ryšį.
- K_{lw} – nurodoma, ar komponentas laukia kol pasibaigs visų asinchroninių užklausų vykdymas. Reikšmė „true”, nurodo, kad komponentas turi laukti visų asinchroninių užklausų pabaigos. Parametras naudojamas, tik jei K_r = ASYNC.

7.2.3. Sistemos resursai

Programų sistemos veiklai reikalingi vykdymo resursai, dažniausiai šiuos resursus suteikia serveris. Dalis programų sistemų kokybinių charakteristikų, tokių kaip našumas ar klaidų tikimybė priklauso nuo turimų serverio resursų. Metodiniai nurodymai pateikiami darbe [Mik14] pataria serverį modeliuoti kaip atskirą agentą. Modeliuojamas komponentas vykdydamas veiklą turi atlikti užduotis, kurioms reikalingi resursai. Taigi, komponentas siunčia užduotis vykdyti į serverį ir laukia kol šios bus įvykdytos. Taip pat, serverį galima laikyti ir sistemos aplinkos dalimi. Tokiu atveju, resursai turėtų būti modeliuojami kaip visiems komponentus vaizduojantiems agentams prieinami parametrai. Agentas vaizduojantis komponentą prieš atlikdamas užduotį, turėtų patikrinti ar pakanka resursų, jei taip sunaudoti reikalingą resursų kiekį. Pirmasis būdas labiau atspindi modeliuojamos programų sistemos veikimą, o resursus modeliuojant atskirame agente lengviau įgyvendinti įvairias užduočių atlikimo tvarkos strategijas.

Serveris atlieka ne tik komponentų sukurtas užduotis, bet ir siunčia pranešimus tarp jų. Parametro tipas skirtas modeliuoti ryšius, saugo ir pranešimui perduoti reikalingą serverio resursų kiekį. Modeliuojant pranešimo perdavimą, sukuriamos užklausos perdavimo užduotys agente vaizduojančiame serverį. Rekomenduojama modeliuoti keturis resursus: procesorių, operatyviąją atmintį, disko operacijų kiekį bei duomenų perdavimo kiekį tinkle.

Kadangi sistemoje vienu metu veikia ne vienas komponentas, tai ir užduočių į serverį atkeliauja ne viena. Serveris vienu metu gali atlikinėti daug užduočių. Šios užduotys gali būti atliktos įvairia tvarka. Pavyzdžiui, eilės strategijos atveju, užduotys sustatomos į eilę ir atliekamos tokia tvarka, kokia buvo atsiųstos į serverį. Kita atlikimo strategija – atvirkštinė eilė. Šios atveju, paskutinė į serverį atkeliavusi užduotis vykdoma pirma. Taip pat galima vykdyti atsitiktinės tvarkos strategiją, tuomet užduotys vykdomos atsitiktine tvarka.

Darbe [Mik14] pateikiami serverį vaizduojančio agento parametrai:

- S_p – procesoriaus resurso kiekis;
- S_a – operatyviosios atminties resurso kiekis;
- S_d – disko operacijų resurso kiekis;
- S_t – duomenų perdavimo resurso kiekis.
- S_s – užduočių atlikimo strategija: 1 – eilė, 2 – atvirkštinė eilė, 3 – atsitiktinė tvarka.

Jei komponento būsena nepasikeitusi, laikoma, jog komponentas užklausą apdoroja visuomet per tiek pat laiko. Programų sistema dažniausiai yra išdėstoma keliuose serveriuose, todėl ir modeliuojant komponentą, jį vaizduojančiame agente yra parametras nurodantis, kuriame serveryje komponentas įdiegtas.

Resursų kiekis reikalingas atlikti komponento kuriamoms užduotims saugomas komponentą vaizduojančio agento parametruose. Kiekvieno resurso poreikis modeliuojamas atskiru parametru. Šie resursų parametrai nurodo, kiek komponentas atlikdamas užduotį turi rezervuoti serverio resursų. Atskiru parametru nurodoma, kiek laiko turi būti rezervuoti serverio resursai iki komponentas įvykdys užduotis.

Modeliuojant resursus, agentas vaizduojantis komponentą papildomas šiais parametrais:

- K_s – nuoroda į serverį, kuriame komponentas yra įdiegtas;
- K_{cpu} – procesoriaus resursų kiekis procentais, reikalingas atlikti užklausos apdorojimo užduotis;
- K_o – operatyviosios atminties resursų kiekis procentais, reikalingas atlikti užklausos apdorojimo užduotis;
- K_d – disko operacijų resursų kiekis procentais, reikalingas atlikti užklausos apdorojimo užduotis;
- K_s – tinklo resursų kiekis procentais, reikalingas atlikti užklausos perdavimą;
- K_t – užduotis atlikti reikalingas vykdymo laikas milisekundėmis;

7.2.4. Sistemos aplinka

Visos programų sistemos egzistuoja apibrėžtoje aplinkoje. Modeliuojant sistemos aplinką, darbe [Mik14] patariama į programų sistemą žvelgti kaip į „juodąją dėžę". Žvelgiant į sistema kaip į „juodąją dėžę" sistemos aplinkoje galima išskirti jos vartotojus. Tai gali būti tiek žmonės tiek kitos sistemos. Užklausos į modeliuojamą sistemą gali būti siunčiamos ir gaunamos iš jos aplinkos. Kitaip tariant, sistema ne tik gali priimti užklausas iš vartotojų ar kitų sistemų, bet ir pati jas siųsti aplinkai. Siekiant nenutolti nuo modeliuojamos aplinkos, patariama kiekvieną aplinkos elementą – sistemą ar vartotoją modeliuoti atskiru agentu. Rekomenduojama aplinką modeliuoti dviejų tipų agentais: vaizduojančiais naudotojus (tiek vartotojus tiek kitas sistemas) bei vaizduojančiais kviečiamas sistemas.

Modeliuojant vartotoją, svarbu atsižvelgti į keletą aspektų. Pirmiausia, agentas vaizduojantis vartotoją turi turėti ryšį su agentu vaizduojančiu sistemos įeities komponentą. Modeliuojant vartotojo elgseną išskiriama keletas aspektų: užklausų generavimo strategija, užklausų tipas (sinchroninės ar asinchroninės) bei vartotojo reakcija į nepavykusias užklausas. Modeliuojant užklausų generavimo strategiją svarbu apibrėžti laiko intervalus kuriais užklausos yra generuojamos bei generuojamų užklausų kiekį. Modeliuojant sinchroninės užklausos siuntimą svarbu atsižvelgti į

tai, jog vartotojas prieš vykdydamas tolimesnę veiklą privalo laukti sugeneruotos sinchroninės užklausos atsako. Asinchroninės atveju – atsakas nebūtinas. Modeliuojant reakciją į nepavykusias užklausas, svarbūs keli aspektai: galimybė pažymėti užklausą kaip nepavykusią jei ji yra neapdorojamą ilgą laiką bei naudotojo elgsena tokiu atveju. Darbe [Mik14] rekomenduojama modeliuoti dvi strategijas: naudotojas nieko nedaro arba vartotojas pakartoja nepavykusią užklausą.

Darbe [Mik14] pateikiamas agento, vaizduojančio vartotoją parametrų sąrašas:

- V_i – nuorodą į agentą, kuris vaizduoją komponentą priimanti vartotojo užklausas;
- V_{ut} – generuojamų užklausų tipas: „ASYNC" – asinchroninės, „SYNC" – sinchroninės;
- V_s – užklausų generavimo strategija: 0 – užklausos generuojamos atsitiktinai, 1 – užklausos generuojamos reguliariai;
- V_a – užklausų sugeneravimo tikimybė per vieną laiko vienetą. Naudojama jei $V_s = 0$. Sugeneruojama užklausų ne daugiau nei V_{umax} ir ne mažiau V_{umin}. Reikšmė nurodoma intervale [0,1];
- V_t – laiko vienetų kiekis tarp užklausų generavimo. Naudojamas jei $V_s = 1$. Reikšmė nurodo laiko tarpo dydį milisekundėmis tarp užklausų generavimo. Sugeneruojama užklausų ne daugiau nei V_{umax} ir ne mažiau V_{umin};
- V_{umax} – maksimalus sugeneruojamų užklausų kiekis per užklausų generavimo laiko vienetą;
- V_{umin} – minimalus sugeneruojamų užklausų kiekis per užklausų generavimo laiko vienetą;
- V_k – reagavimo į nepavykusią užklausą strategija. 0 – nieko nedaryti, 1 – kartoti užklausą;
- V_{uk} – maksimalus užklausų kartojimų kiekis. Jei reikšmė 0 – kartojama neribotą kiekį. Naudojamas jei $V_k = 1$;
- V_{ul} – laikas milisekundėmis, per kurį neišsiuntus užklausos ji yra pažymima kaip nepavykusi;

Modeliuojant agentą, vaizduojantį kviečiamą sistemą, į ją žvelgiamą kaip į vientisą vienetą. Jos struktūra nedetalizuojama. Nors galima modeliuoti įvairius šios sistemos parametrus, tačiau darbe [Mik14] rekomenduojama į modelį įtraukti: sistemos atsako laiką, užklausos apdorojimo tikimybę bei planuotą sistemos neveikimo laiką. Darbe [Mik14] pateikiamas agento, vaizduojančio kviečiamą sistemą, parametrų sąrašas:

- R_a – sistemos atsako laikas milisekundėmis;
- R_k – sistemos klaidos tikimybė. Reikšmė nurodoma intervale [0,1];
- R_a – tikimybė, jog kviečiame sistema yra išjungta. Reikšmė nurodoma intervale [0,1];

Čia pateikiama tik vienas iš galimų, kviečiamą sistemą vaizduojančio agento, parametrų rinkinys. Prireikus, kviečiamą sistemą galima detalizuoti – modeliuoti ne kaip vieną sudėtinį komponentą, o kaip keleto komponentų junginį. Tokiu atveju, reikia vadovautis komponento modeliavimo nurodymais.

7.2.5. Apibdendrinimas

Atsižvelgiant į ryšius tarp modeliuojamos architektūros principinių elementų ir agentais grįsto modelio elementų, bei atsižvelgiant į darbe [Mik14] pateikiamus nurodymus, šioje metodikoje programų sistemą siūloma modeliuoti keturių tipų agentais: atspindintį sistemos naudotoją, komponentą, kviečiamą sistemą ir serverį. Kiekvienam agentui, vaizduojančiam komponentą, reikalinga priskirti agentą, vaizduojantį serverį. Reikalinga sukurti ryšį tarp agentų, vaizduojančių naudotojus ir įeities komponentus. Remiantis sistemos funkciniu vaizdu sukuriami ryšiai tarp komponentus vaizduojančių agentų.

Sistemos naudojimas modeliuojamas vartotojus vaizduojančio agento siunčiamomis užklausomis. Užklausos siunčiamos į vartotojui priskirtą įeities komponentą. Modeliuojamoje sistemoje, tiek užklausų siuntimui, tiek komponento veiklai yra naudojami serverio resursai. Jų naudojimas modeliuojamas kaip komponentų sukuriami darbai serveryje. Į modeliuojamo komponento veiklos modelį įtraukiama ir užklausos apdorojimo klaidos tikimybė. Jei modeliuojamas komponentas kviečia kitus komponentus, agentas, vaizduojantis komponentą, taip pat turi kviesti kitus agentus. Pasibaigus užklausos vykdymui, atsakymas turi būti siunčiamas į naudotoją vaizduojantį agentą.

7.3. Būsenos modeliavimas

Dažnai sistemos veikla priklauso nuo joje saugomų būsenų. Keičiantis sistemos veiklai keičiasi ir vykdymo resursų poreikis, o tai gali daryti įtaką sistemos kokybinėms charakteristikoms. Pavyzdžiui, atsižvelgiant į vidinę būseną, komponentas pradeda vykdyti sudėtingesnį užklausų apdorojimo algoritmą. Tokiu atveju yra sunaudojama daugiau serverio resursų. Sudėtingesnis algoritmas taip pat gali būti vykdomas ilgiau nei įprastas, todėl tikėtina jog sistemos greitaveika sumažės.

Būsenų pasikeitimas sistemoje gali turėti ir teigiamą įtaką kokybinėms charakteristikoms. Pavyzdžiui, komponentas, atsižvelgiant į vidinę būseną, pradeda vykdyti paprastesnį užklausų apdorojimo algoritmą. Tokiu atveju komponentas užklausas apdoroja greičiau ir sunaudoja mažiau serverio resursų. Tikėtina, jog šiuo atveju sistemos greitaveika padidėtų. Modeliuojant būseną svarbu atsižvelgti tiek į teigiamą, tiek į neigiamą įtaką komponento veiklai.

Žvelgiant iš modeliavimo agentais paradigmos, agentai modelyje taip pat turi būseną. Ji yra vaizduojama agento parametru ar parametrų rinkiniu. Lygiai kaip ir sistema veiklą koreguoja atsižvelgdama į esamas būsenas, taip ir agentais grįstame modelyje, agentai priima sprendimus apie kitus vykdymo žingsnius, atsižvelgdami į esamą būseną – turimą informaciją. Atsižvelgiant į šį panašumą, metodika, sistemos būsenas pataria modeliuoti agento parametru ar parametrų rinkiniu, kurių reikšmės kinta priklausomai nuo modeliuojamos sistemos veiklos. Šią priklausomybę patariama apibrėžti agento elgsenoje.

7.3.1. Komponentą apimančios būsenos modeliavimas

Protokolo būsena. Protokolo būsena yra apibrėžiama architektūros modelio kūrimo metu. Ji gali būti modeliuojama kaip agento, vaizduojančio komponentą parametras. Protokolo būsena daro įtaka užklausų apdorojimui, modeliuojant ši įtaka yra apibrėžiama agento elgsenoje. Modeliuojant,

sukuriamas tarsi papildomas tarpininkas tarp užklausų priėmimo ir komponento funkcionalumo vykdymo, kuris filtruoja įeinančias užklausas. Būsenai modeliuoti įvedamas agento parametras K_{bp}. Jo reikšmė priklauso nuo modeliuojamos būsenos pobūdžio. Pateikiama keletas šios būsenos modeliavimo pavyzdžių:

- *Užklausos, išsiųstos seniau nei prieš x laiko, turi būti atmestos.* Modeliuojant šią situaciją svarbūs du aspektai: saugoti parametrą, nusakantį maksimalų leistiną laiko tarpą nuo užklausų išsiuntimo iki priėmimo į komponentą; prieš apdorojant užklausą patikrinti ar šis laikas neviršytas. Šiuo atveju maksimalus laiko tarpas gali būti saugomas agento, vaizduojančio komponentą, parametre K_{bp}. Parametrui priskiriant milisekundėmis išreikštą reikšmę. Agentas, prieš atlikdamas užklausos apdorojimą, turėtų patikrinti ar einamojo ir užklausos išsiuntimo laikų skirtumas nėra didesnis nei K_{bp}. Jei taip, užklausa turėtų būti atmesta.

- *Užklausos, išsiųstos iš x komponento, turi būti atmestos.* Čia, svarbu saugoti komponento, iš kurio išsiųstos užklausos turi būti atmestos, vardą ir įgyvendinti užklausų patikrinimą. Šiuo atveju, protokolo būsena tai saugomas komponento vardas. Ji gali būti saugoma agento, vaizduojančio komponentą, parametre K_{bp}. Taip, kaip ir ankstesniu atveju, taip ir čia užklausų patikrinimas turėtų būti aprašytas agento elgsenoje. Pirmiausia turėtų būti patikrinta, ar užklausa buvo išsiųsta iš kito sistemos komponento. Jei taip, toliau turėtų būti patikrinta ar užklausą sugeneravusio komponento vardas nesutampa su saugomu parametre K_{bp}. Jei sutampa, užklausą turėtų būti atmesta.

- *Žinoma x tikimybė, jog įeinanti užklausa neatitinka protokolo, todėl ji turi būti atmesta.* Šioje situacijoje, svarbu saugoti tikimybę, jog užklausa neatitinka protokolo bei įgyvendinti užklausos patikrinimą. Minėtąją tikimybę laikysime protokolo būsena. Ji gali būti saugoma agento, vaizduojančio komponentą, parametre K_{bp}. Tikimybė nurodoma intervale [0,1]. Agento elgsenoje, turėtų būti aprašyta protokolo neatitikimo tikimybės patikrinimas. Jei tikimybė patenkinama, tuomet užklausa turėtų būti atmesta. Tokią situaciją patogu modeliuoti tada, kai nežinomas modeliuojamos sistemos užklausų pobūdis, tačiau galima įvertinti užklausos neatitikimo protokolui tikimybę.

Vidinė komponento būsena. Ši būsena modeliuojama, kaip agento, vaizduojančio komponentą, parametras. Ji yra apibrėžiama tiems agentams, kurie vaizduoja komponentus su vidine būsena. Komponento vidinė būsena gali būti nustatyta modelio inicijavimo metu arba priskirta vykdymo metu. Svarbu, jog agentas vykdymo metu turėtų galimybę keisti ir nuskaityti būseną. Vidinei komponento būsenai modeliuoti, įvedamas agento parametras K_{bv}. Būsena gali būti apibrėžiama ir keletu skirtingų agento parametrų, tai priklauso nuo modeliuojamos būsenos pobūdžio. Komponento veiklos priklausomybė nuo vidinės būsenos modeliuojama agento elgsenoje. Pateikiamas šios būsenos modeliavimo pavyzdys:

- *Komponentas, apdorojęs x užklausų, įjungia lėtesnio ar greitesnio veikimo rėžimą.* Tokios situacijos modeliavimo poreikis gali kilti norint pavaizduoti riboto kiekio paslaugą teikiantį komponentą. Pavyzdžiui, yra ribojamas nemokamas užklausų apdorojimo kiekis. Viršijus

jį, paslaugos kokybė prastėja – veikia lėčiau. Arba atvirkščiai, dažnai besinaudojantiems vartotojams teikiama geresnės kokybės paslauga – veikianti greičiau. Šiuo atveju, reikalinga saugoti komponente apdorotų užklausų kiekį ir po x užklausų pakeisti agento, vaizduojančio komponentą, elgseną. Apdorotų užklausų kiekis, turėtų būti saugomas agento, vaizduojančio komponentą, parametre K_{bv1}. Tai vidinė komponento būsena. Užklausų kiekis x, po kurio komponento veikla pagreitėja arba palėtėja, turėtų būti saugomas parametre K_{bv2}. Šis parametras atspindi ne vidinę komponento būseną, bet konfigūracijos. Ji plačiau aptariama žemiau. Modeliuojama situacija tiesiogiai susijusi su komponento greitaveika, todėl verta įvesti ir papildomą parametrą nusakantį būsenos įtakos stiprumą komponento veiklai. T.y. koeficientą, kuris nurodytų kiek kartų komponento darbas sulėtėja ar pagreitėja po x užklausų. Koeficientui saugoti, gali būti įvestas papildomas agento parametras K_{bv3}, jam priskiriant reikšmes intervale $(-\infty; +\infty)$. Čia neigiamos reikšmės atspindi modeliuojamo komponento pagreitėjimą, o teigiamos – sulėtėjimą. Modeliuojant komponento veiklą, yra kuriamos vykdymo užduotys, o jose nurodomas vienos užklausos apdorojimo komponente laikas. Jis yra nustatomas atsižvelgiant į agento parametrą K_t. Taigi, modeliuojant sulėtėjusį ar pagreitėjusį komponentą, reikia pakeisti agento elgseną taip, kad prieš kurdamas vykdymo užduotis, šis patikrintų ar $K_{bv1} > K_{bv2}$. Jei taip, kuriamoms vykdymo užduotims nurodytų vykdymo komponente laiką: $K_t * K_{bv3}$.

Priskyrimo būsena. Komponentui būdinga priskyrimo būsena gali būti apibrėžta statiniu agento, vaizduojančio komponentą, parametru. Jis turėtų būti priskiriamas modelio inicializavimo metu. Parametro reikšmę nustato pats modelio kūrėjas. Svarbu tai, jog ši būsena yra nekeičiama modelio vykdymo metu. Tokios būsenos pavyzdys:

- *Maksimalus lygiagrečiai apdorojamų užklausų kiekis komponente*. Šiai būsenai apibrėžti darbe [Mik14] jau įvestas agento parametras K_l, tačiau darbe neapibrėžta, jog tai yra priskyrimo būsena. Modeliuojant šios būsenos įtaką sistemai, agento, vaizduojančio komponentą, elgsena pakoreguojama taip, jog agentas prieš pradėdamas vykdyti modeliuojamas užklausas, patikrintų ar einamuoju momentu komponente vykdomų užklausų skaičius nėra didesnis už K_l. Jei taip, užklausos turėtų būti nepriimamos vykdymui.

Konfigūracijos būsena. Taip kaip ir priskyrimo būsena, taip ir konfigūracijos būsena gali būti modeliuojama statiniu agento parametru. Jis turėtų būti nustatomas modelio kūrėjo dar prieš modelio inicializaciją. Kaip ir priskyrimo būsena, taip ir konfigūracijos būsena vykdymo metu yra nekeičiama. Bendrai, priskyrimo ir konfigūracijos būsenos yra modeliuojamos tokiu pat būdu. Konfigūracijos būsenos modeliavimo pavyzdys:

- *Apibrėžtas greičiau apdorojamų užklausų kiekis*. Šis būsenos pavyzdys aptartas vidinės būsenos modeliavimo pavyzdyje. Ten būsena apibrėžta agento parametru K_{bv2}. Kadangi tai dar prieš sistemos inicijavimą žinoma būsena, ji yra laikoma konfigūracijos būsena.

7.3.2. Sistemą apimančios būsenos modeliavimas

Globali būsena. Tai visiems, sistemos komponentus vaizduojantiems, agentams vykdymo metu prieinama informacija. Kadangi sistemos aplinka nėra modeliuojama kaip atskiras agentas, metodika pataria šia būseną modeliuoti kaip agento, turinčio ryšius su visais kitais agentais, parametrą. Svarbu, jog šis parametras būtų prieinamas visiems modelio agentams. Modeliuojant sistemą, kuri yra išdėstyta viename serveryje, tokios būsenos pavyzdys yra agento vaizduojančio serverį parametrai: S_p. S_a, S_d, S_t. Čia jie atspindi likusių resursų kiekį, kurį gali sužinoti visi jame esantys modeliuojamos sistemos komponentai.

7.3.3. Vartotoją apimančios būsenos modeliavimas

Sesijos būsena. Ši būsena dažniausiai yra apibrėžiama remiantis dalykine sritimi, modeliuojama kaip agento, vaizduojančio vartotoją, parametras. Būsenai saugoti įvedamas agento parametras V_b. Jei reikalinga galima įvesti ir papildomus būseną vaizduojančius agento parametrus. Dažnai vartotojo veiksmai priklauso nuo saugomos būsenos, todėl ir modeliuojant, ši priklausomybė turėtų būti apibrėžta vartotoją vaizduojančio agento elgsenoje. Tokios būsenos modeliavimo pavyzdys:

- *Vartotojas, sulaukęs n nesėkmingų užklausų, nebegeneruoja naujų užklausų*. Šioje situacijoje svarbu saugoti nesėkmingai įvykdytų užklausų kiekį ir maksimalų leistiną nesėkmingų užklausų kiekį. Tai yra vartotojo sesijos būsena. Jai saugoti naudojami atitinkamai agento parametrai V_{b1} ir V_{b2}. Parametras V_{b2}, saugantis maksimaliai leistiną nesėkmingų užklausų kiekį, yra nustatomas modelio inicializavimo metu. Parametras V_{b1} yra keičiamas kas kart užklausai pasibaigus nesėkme. Modeliuojant šią situaciją, agento elgsena pakoreguojama, taip jog, prieš naujų užklausų generavimą patikrintų ar V_{b1} daugiau už V_{b2}, jei taip, nebegeneruotų užklausų. Taip pat, kaskart užklausai pasibaigus nesėkme, agentas turi pakeisti V_{b1} reikšmę į $V_{b1} + 1$.

Nuolatinė būsena. Lygiai kaip ir sesijos būsena, ši turėtų būti modeliuojama kaip agento, vaizduojančio vartotoją, parametras. Jei norima modeliuoti iš anksto nustatytą būseną, jos reikšmę turėtų nustatyti modelio kūrėjas dar prieš modelio inicializavimą. Tokios būsenos pavyzdys yra aukščiau aprašytas, vartotoją vaizduojančio, agento parametras V_k. Šis nusako vartotojo reagavimo į nepavykusias užklausas strategiją.

8. Simuliacijos įgyvendinimas

Remiantis metodiniais nurodymais, pateikiamais skyriuje „Programų sistemos modeliavimas agentais”, sukuriamas sistemos modelis. Tai tik statinis modelis, norint įvertinti sistemos kokybines charakteristikas reikalinga įvykdyti simuliaciją. Tam pasirinktas modeliavimo ir simuliavimo agentais karkasas „Repast Simphony”. Vykdant simuliaciją imituojama tiriamos sistemos veikla, matuojami apibrėžti kokybiniai matai. Pasirinktas karkasas leidžia kurti agentų esybes, jas sujungti ryšiais, prireikus ir vizualizuoti. Šis karkasas paremtas „Java” programavimo kalba, todėl ir jame kuriamos esybės aprašomos šia programavimo kalba.

Simuliacija vykdoma žingsniais. Vieną laiko vienetą atstoja vienas žingsnis, jo metu iškviečiamos agentų žingsnio vykdymo funkcijos. Simuliacijos metu, be metodikoje aptartų agentų tipų, sukuriamas ir architektūros generatorius. Jis inicializuoja modelį, kitaip tariant sukuria atitinkamus agentus ir atsižvelgdamas į modeliuojamą sistemą sujungia juos ryšiais.

8.1. Simuliacijos sistema

Pasirinktas programavimo karkasas neturi griežtų apribojimų, todėl konstruojant simuliacijos sistemą suteikiama daug laisvės. Jog karkasas gebėtų vykdyti simuliaciją, pakanka įgyvendinti agentų kūrimo funkciją. Simuliacijos valdiklio klasė yra apibrėžiama karkaso konfigūracijos faile, šiuo atveju tai klasė „Simuliacijos valdiklis”. Ji yra atsakinga už modelio inicializavimą bei statistinių matų išvedimą. Inicializavimo metu sukuriami agentai ir agentų ryšiai. Remiantis darbe [Mik14] suformuota simuliacijos sistema bei šio darbo metodika, pateikiama simuliacijos sistemos struktūra.

2 pav. Simuliacijos sistemos struktūra

Simuliacijos valdiklis kurdamas komponentus atspindinčius agentus, jiems nustato konfigūracijos ir priskyrimo būsenas. Taip pat nurodo komponento veiklai vykdyti reikalingą resursų kiekį. Valdiklis kurdamas agentus, vaizduojančius serverius, jiems nustato turimų resursų parametrus. Kuriant agentus, vaizduojančius vartotojus, nustatoma užklausų generavimo ir reagavimo į nesėkmingas užklausas strategijos. Jei reikalinga, nustatoma vartotojo nuolatinė būsena. Valdikliui kuriant kviečiamas sistemas vaizduojančius agentus, nustatomi atsako laiko, klaidos bei išjungimo tikimybių parametrai.

Priklausomai nuo architektūros generavimo realizacijos, agentai gali būti sujungiami ryšiais agentų kūrimo metu arba vėliau kaip atskiras žingsnis.

8.1.1. Architektūros generavimas

Architektūros generavimą atlieka esybė „Architektūros generatorius". Joje, specialiai šiam tikslui, sukurtas metodas „generuotiArchitektūrą". Šiame metode aprašomos taisyklės, kuriomis remiantis konstruojamas modelis. Generatoriui taip pat gali būti įvedami įvairūs papildomi parametrai.

Pavyzdžiui, šiame darbe pateikiama „SOA" ir „EDA" architektūriniais šablonais paremtų programų sistemų modeliavimo pavyzdžiai. Šių modelių generatoriams buvo įvesta keletas papil-

domų parametrų: tikimybė, jog sugeneruotas komponentas bus sugedęs bei komponentų skaičių nusakantys parametrai. Bendrai architektūros generatorius kuria agentus, simbolizuojančius trijų tipų komponentus: į kuriuos yra siunčiamos užklausos, atsakingus už sistemos funkcijas, bei šias funkcijas įgyvendinančius komponentus. Pirmiausia, generatorius sukuria agentus, vaizduojančius įeities komponentus ir juos sujungia su vartotojus atspindinčiais agentais. Kitu žingsniu generuojami agentai, vaizduojantys visus kitus sistemos komponentus. Jie taip pat sujungiami atitinkamais ryšiais su įeities komponento agentu. Kiekvienam komponentą vaizduojančiam agentui sukuriamas ryšys su agentu, vaizduojančiu serverį.

Norint modeliuoti tiksliai apibrėžtos architektūros programų sistemą ar modeliuojant kitokio architektūrinio stiliaus programų sistemas reikia pakeisti architektūros generatoriaus veikimą. Tam reikalinga perrašyti generatoriaus klasės metodą „generuotiArchitektūrą".

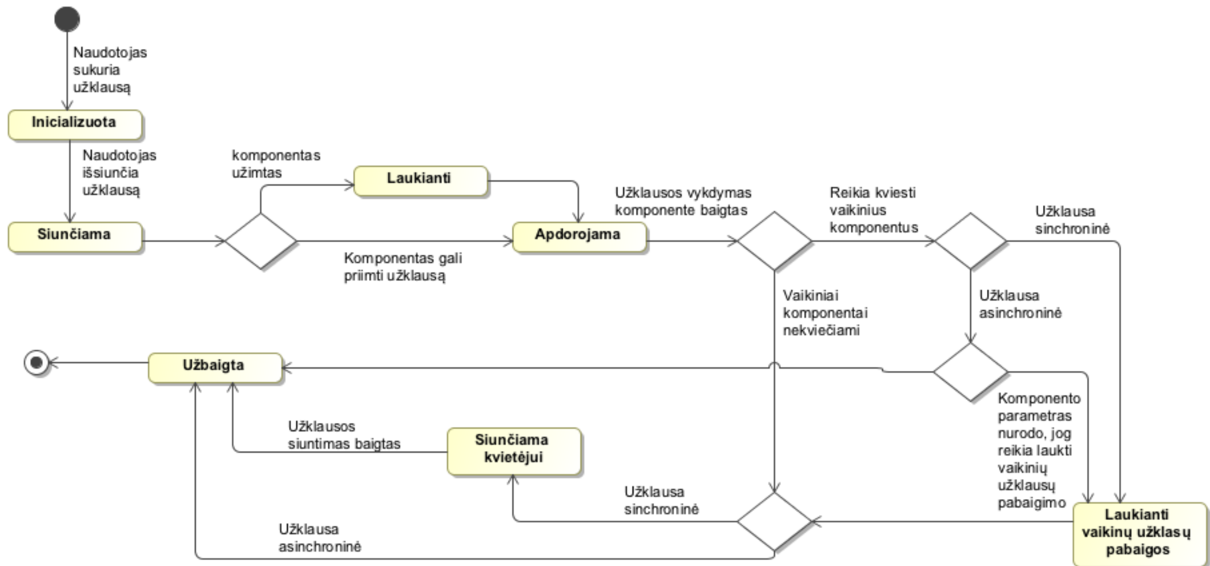
8.1.2. Simuliacijos vykdytojai

Simuliaciją vykdo agentai, atspindintys šias esybes: komponentas, naudotojas, kviečiama sistema bei serveris.

Komponentus vaizduojantys agentai imituoja tiriamos sistemos komponentus. Kiekvienas toks agentas susietas ryšiu su serverį vaizduojančiu agentu. Pastarasis atlieka sistemoje formuojamų užduočių vykdytojo rolę. Atsižvelgiant į ryšį agentas-serveris, visos konkretaus komponento užduotys siunčiamos į vieną serverį vaizduojantį agentą. Serveris vykdo tiek užduočių, kiek pakanka turimų resursų. Žemiau pateikiamuose pavyzdžiuose modeliuojamas serveris, vykdantis užduotis remiantis eilės strategija.

Agentai, vaizduojantys sistemos naudotojus, simuliuoja sistemos apkrovimą. Agentas, pagal nustatytus parametrus sugeneruoja užklausas ir pagal atitinkamus ryšius jas išsiunčia į agentus, vaizduojančius komponentus. Priklausomai nuo modeliuojamos naudotojo elgsenos, užklausos, atmestos įeities komponente, gali būti pakartotos arba pažymėtos kaip nesėkmingos.

Sugeneruotos užklausos modeliuojamos esybe „Užklausa" *(žr. 2 pav.)*. Priklausomai nuo užklausos vykdymo, ji gali būti šių būsenų: inicializuota, siunčiama, laukianti, apdorojama, kviečianti vaikines užklausas, siunčiama atgal bei pabaigta. Jei komponentas turi kviesti kitus komponentus, suformuojamos vaikinės užklausos. Jos turi ryšį su tėvine užklausa, todėl esant sinchroniniui ryšiui tarp komponentų, nesunku stebėti kada yra pabaigiamos visos vaikinės užklausos. Pasibaigus vaikinių užklausų vykdymui, tėvinė užklausa yra pažymima kaip baigta. Jei vaikinės užklausos yra asinchroninės, tuomet tėvinė užklausa iškart pažymima kaip baigta. Užklausos esybėje saugoma ir statistiniai duomenys apie jos vykdymo pradžios bei pabaigos laikus.

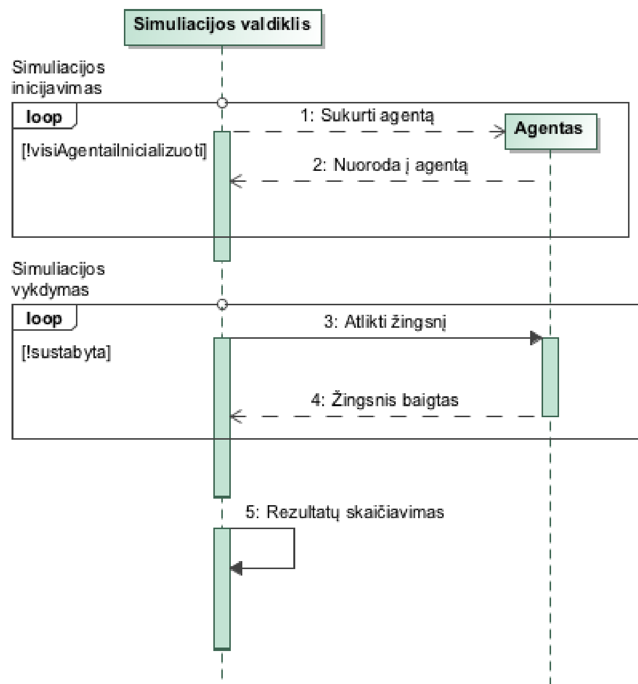


3 pav. Užklausos būsenų diagrama

8.2. Simuliacijos vykdymas

Simuliacija vykdoma trimis etapais:

1. Simuliacijos sistemos inicializavimas;
2. Simuliacijos vykdymas;
3. Rezultatų skaičiavimas;



4 pav. Bendroji simuliacijos vykdymo sekų diagrama

8.2.1. Simuliacijos sistemos inicializavimas

Šiame etape kuriami agentai ir ryšiai tarp jų. Pirmiausia, sukuriami agentai, vaizduojantys serverius. Kūrimo metu nustatomas modeliuojamų resursų kiekis. Kitu etapu inicializuojami agentai, vaizduojantys komponentus. Kūrimo metu jie priskiriami konkretiems serverius vaizduojantiems agentams. Agento parametrai nustatomi jo kūrimo metu. Kiekvienam agentui priskiriami unikalus vardai, tokiu būdu juos lengva identifikuoti analizuojant gautus rezultatus. Komponentus vaizduojantys agentai kuriami trimis etapais: pirmiausia sukuriami įeities komponentai, vėliau komponentai atsakingi už sistemos funkcionalumo koordinavimą, galiausiai sukuriami agentai vaizduojantys funkcionalumo įgyvendinimą. Kiekvieno agento kūrimo metu sukuriami ir jo ryšiai.

Modeliuojant kitokia architektūra paremtą sistemą reikia naudoti kitokį architektūros generatorių, kuris inicializuotų ir sujungtu agentus pagal modeliuojamą sistemą.

8.2.2. Simuliacijos vykdymas

Pasirinktas karkasas simuliaciją vykdo žingsniais. Kiekvieno žingsnio metu yra kviečiami visų agentų žingsnio vykdymo metodai. Šiame metode aprašyti veiksmai kuriuos turi atlikti agentas.

Agentai, vaizduojantys naudotojus, pagal aprašytas taisykles sugeneruoja užklausą ir siunčia ją į sistemą per įeities komponentą. Jei modeliuojami serveriai neturi užklausai apdoroti reikalingų laisvų resursų, naudotojas, pagal aprašyta strategiją, pakartoja užklausos siuntimą arba tęsia tolimesnę savo veiklą. Agentas, vaizduojantis serverį, žingsnio metu vykdo jam sugeneruotas užduotis. Vykdymo strategija aprašoma agente, tačiau standartiškai taikoma eilės strategija. Vieno žingsnio metu vykdoma tiek užduočių, kiek turima laisvų resursų. Užduočių, kurioms vykdyti nepakanka resursų, vykdymas yra atidedamas kitam žingsniui.

Užklausų apdorojimas modeliuojamas agentuose, kurie vaizduoja komponentus. Komponento vieno simuliacijos žingsnio sekų diagrama pateikiame priede nr.1. Agentas žingsnio pradžioje patikrina protokolo būseną ir ar yra įeinančių užklausų. Jei yra įeinančių užklausų, tuomet atsižvelgiant į protokolo būseną patikrinama ar užklausos gali būti priimtos. Jei taip, agente, vaizduojančiame serverį, užklausų siuntimui sukuriamos užduotys, o užklausa pažymima kaip siunčiama. Serveriui įvykdžius sukurtas užduotis, užklausa pažymima kaip atsiųsta į komponentą. Tuomet agentas, vaizduojantis komponentą, atsižvelgdamas į vidinę būseną, sukuria užklausos apdorojimo užduotis ir jas išsiunčia į agentą, vaizduojantį serverį. Standartinis resursų kiekis reikalingas apdoroti užklausą yra nurodytas agento, vaizduojančio komponentą, parametruose. Priklausomai nuo komponento būsenos, šis resursų kiekis gali būti keičiamas komponento veiklos metu. Užduotyse be resurso kiekio nurodoma ir užduoties vykdymo laikas. Kuomet užduotys yra įvykdomos, laikoma jog komponento funkcionalumas yra įvykdytas. Kitu žingsniu patikrinama ar vykdant užklausą neįvyko klaida, jei taip, ji pažymima kaip sugadinta. Jei klaida neįvyko, patikrinama ar komponento funkcionalumui užbaigti reikalinga kviesti vaikinius komponentus. Jei taip, remiantis apibrėžtais komponento parametrais, sugeneruojamos užklausos į vaikinius komponentus. Jei ryšys su vaikiniu komponentu yra sinchroninis arba komponento parametruose yra nurodyta jog privaloma laukti asinchroninių užklausų pabaigos, agentas vaizduojantis komponentą, laukia kol

bus baigtos vaikinės užklausos. Pasibaigus joms, užklausa yra pažymima kaip baigta. Jei užklausa buvo atsiųsta sinchroniniu būdu, ji išsiunčiama atgal ją atsiuntusiam komponentą vaizduojančiam agentui.

8.2.3. Rezultatų skaičiavimas

Simuliacija užbaigiama vykdymo rezultatų skaičiavimu. Visos simuliacijos metu valdiklis kaupia informaciją apie vykdomas užklausas. Pasibaigus simuliacijai, karkasas „Repast Simphony" apibrėžtus matus išveda į rezultatų failą. Šiame darbe pateikiamuose pavyzdžiuose išvedama užklausų vykdymo laikas, nesėkmingai pasibaigusių užklausų kiekis bei laisvas serverio procesoriaus resurso kiekis.

9. Metodikos validavimas

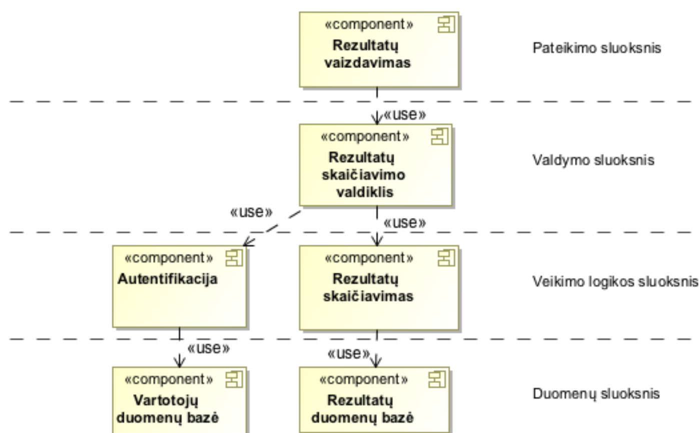
Straipsnyje [Sar11] aptariami įvairūs validavimo būdai. Vienas iš jų tai rezultatų gautų modeliuojamoje sistemoje bei tos sistemos modelyje įvykdytos simuliacijos metu gautų rezultatų palyginimas. Šis validavimo būdas pritaikytas ir šiame darbe. Nefunkcinių savybių matai gauti pavyzdinėje sistemoje palyginami su simuliacijos metu gautais nefunkcinių savybių matais. Simuliacija atliekama pavyzdinės sistemos modelyje. Validuojant metodiką pasirinkta gan paprasta pavyzdinė sistema.

9.1. Pavyzdinė sistema

Pavyzdinė sistema, tai žiniatinklio paslauga, teikianti balsavimo suvestinę. Sistemoje esančias balsavimo anketas kuria administratoriai, o įprasti vartotojai gali jas pildyti bei peržiūrėti rezultatų suvestinę. Ši sistema pasiekiama HTTP protokolu. Modeliuoti pasirinkta tik ta sistemos dalis, kuri leidžia peržiūrėti balsavimo suvestinę. Vartotojas, norėdamas pamatyti balsavimo suvestinę, užklausoje pateikia savo prisijungimo duomenis ir norimos apklausos identifikatorių. Sistema, pagal pateiktą identifikatorių, duomenų bazėje išrenka balsus ir juos susumuoja. Rezultatai pateikiami mažėjimo tvarka. Modeliuojamoje sistemoje tyrimo metu buvo apie 30 000 balsavimo įrašų.

Sistema sukurta remiantis sluoksnių architektūriniu stiliumi, „Groovy" programavimo kalba. Lygiagrečiai sistema gali apdoroti 100 užklausų. Duomenys saugomi „MySQL" duomenų bazių valdymo sistemoje. Tiek programa tiek duomenų bazių valdymo sistema įdiegta į tą patį serverį, turintį „Intel Core i7", 2,7 GHz spartos procesorių, 4 GB operatyviosios atminties bei 12MB/s disko skaitymo ir rašymo greičiu pasižymintį kietąjį diską.

Naudojama duomenų bazių valdymo sistema, turi integruotą spartinančiąją atmintį (*angl. caching*). Taigi, nuolat pateikiant tokias pat užklausas į duomenų bazę, dalis informacijos yra padedama į podėlį (*angl. cache*), todėl pakartotinos užklausos yra įvykdomos greičiau. Jei visą duomenų bazių valdymo sistemą laikyti kompleksiniu komponentu, podėlis gali būti laikomas vidine komponento būsena. Kadangi, ši būsena daro įtaką greitaveikai, ji įtraukta ir į sistemos modelį.



5 pav. Pavyzdinės sistemos įgyvendinimo perspektyva

9.2. Pavyzdinės sistemos modelis

Modeliuojant sistemą, kiekvienas komponentas modeliuojamas atskiru agentu. Duomenų bazių valdymo sistemoje saugomas podėlis modeliuojamas kaip ją atspindinčio agento būsena. Šio agento elgsena atsižvelgia į tai, jog podėlis spartina duomenų bazių valdymo sistemos veiklą. Apdorojant kas 100-ąją užklausą agentas sutrumpina modeliuojamą duomenų bazių valdymo sistemos užklausos apdorojimo laiką 0.05 karto, iki pasiekiama 0.3 karto riba. Sistemoje visi komponentai sujungti sinchroniniais ryšiais, taigi ir modelyje suformuojami atitinkami ryšiai tarp agentų. Visa modeliuojama sistema įdiegta į tą patį serverį, todėl ir visiems komponentus vaizduojantiems agentams priskiriamas tas pats, serverį vaizduojantis, agentas.

Modeliuojant komponentus, nustatomi šie agentų parametrai:

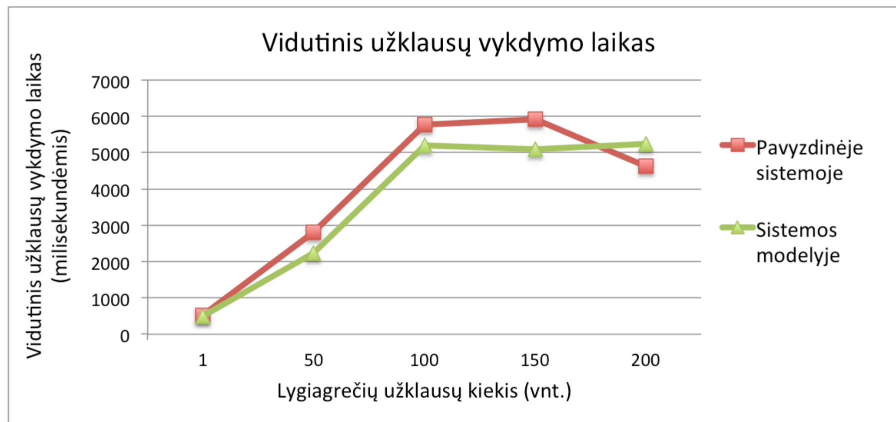
- „Rezultatų vaizdavimas": K_k = ALL, K_w = 0, K_l = 100, K_e = 0,0001, K_{dw} = 0, K_{cpu} = 1%, K_o = 1%, K_d = 0%, K_s = 1%, K_{kt} = 15;
- „Rezultatų skaičiavimo valdiklis": K_k = ALL, K_w = 0, K_l = 100, K_e = 0,001, K_{dw} = 0, K_{cpu} = 2%, K_o = 1%, K_d = 0%, K_s = 1%, K_{kt} = 20;
- „Autentifikacija": K_k = ALL, K_w = 0, K_l = 100, K_e = 0,001, K_{dw} = 0, K_{cpu} = 3%, K_o = 2%, K_d = 0%, K_s = 3%, K_{kt} = 100;
- „Vartotojų duomenų bazė": K_w = 0, K_l = 100, K_e = 0,05, K_{dw} = 0, K_{cpu} = 5%, K_o = 5%, K_d = 5%, K_s = 2%, K_{kt} = 300;
- „Rezultatų skaičiavimas": K_w = 0, K_l = 100, K_e = 0,005, K_{dw} = 0, K_{cpu} = 7%, K_o = 5%, K_d = 0%, K_s = 2%, K_{kt} = 100;
- „Rezultatų duomenų bazė": K_w = 0, K_l = 100, K_e = 0,05, K_{dw} = 0, K_{cpu} = 5%, K_o = 5%, K_d = 5%, K_s = 2%, K_{kt} = 350.

Modeliuojant skirtingą sistemos apkrovimą atlikta keletas bandymų. Kiekviename bandyme buvo keičiami šie, agento, vaizduojančio vartotoją, parametrai: V_t, V_{umax}, V_{umin}. Kitos parametrų reikšmės nebuvo keičiamos: V_{ut} = ASYNC, V_s = 1, V_k = 0, V_{uk} = 500.

Modeliuojant serverį, nustatyti šie agento parametrai: S_p = 100, S_a = 100, S_d = 100, S_t = 100, S_s = 1.

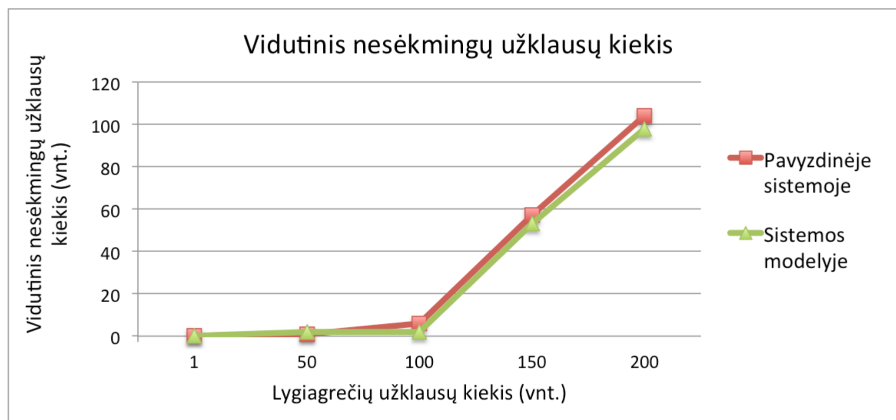
9.3. Rezultatų palyginimas

Sukurtame modelyje bei sistemoje atlikti 5 bandymai. Kiekvieno bandymo metu buvo keičiamas sistemos apkrovimas, keičiant į sistemą lygiagrečiai pateikiamų užklausų kiekį. Kiekvieno bandymo metu gauti užklausų vykdymo laikai bei laisvi serverio resursai, buvo suvidurkini. Taigi, į grafiką įtrauktas kiekvieno bandymu metu apdorotų užklausų vykdymo laiko vidurkis bei laisvo serverio resurso vidurkis.



6 pav. Vidutinio užklausų vykdymo laikų palyginimas

Esant nedideliam sistemos apkrovimui, tiek sistemoje, tiek modelyje atlikti bandymai parodė, tiesišką vykdymo laiko didėjimą. Kadangi, modeliuojama sistema gali lygiagrečiai apdoroti 100 užklausų, pasiekus šį apkrovimą, atsako laikas stabilizuojasi. Visos perteklinės užklausos yra statomos į eilę prie įeities komponento, o jų nepriėmus apdorojimui per V_{uk} laiką, jos pažymimos kaip nesėkmingos. Grafiko pabaigoje matomas pavyzdinės sistemos užklausų vykdymo laiko sumažėjimas. Taip yra nes, pavyzdinėje sistemoje, didinant lygiagrečių užklausų kiekį sistema nebe sugeba tinkamai administruoti lygiagrečias užklausas, ir vis daugiau jų atmeta. Į sistemą priimama apdoroti mažiau nei 100 užklausų, todėl jos apdorojamos greičiau. Tuo tarpu simuliacijos metu, pasiekus 100 užklausų ribą, apdorojama apytiksliai toks pat kiekis užklausų.



7 pav. Nesėkmingų užklausų kiekių palyginimas

Tiek sistemoje, tiek modelyje esant mažesniam nei 100 lygiagrečių užklausų apkrovimui klaidų kiekis gan stabilus ir nedidelis. Pasiekus maksimalią lygiagrečiai apdorojamų užklausų ribą, klaidų kiekis didėja tiesiškai. Pavyzdinėje sistemoje pastebima, jog ir esant 100 lygiagrečių užklausų apkrovimui, šiek tiek daugiau užklausų pasibaigia nesėkmingai, nei modelyje. Taip yra, nes sistemoje yra daugiau faktorių įtakojančių užklausų apdorojimą, kurių neįtraukia modelis.

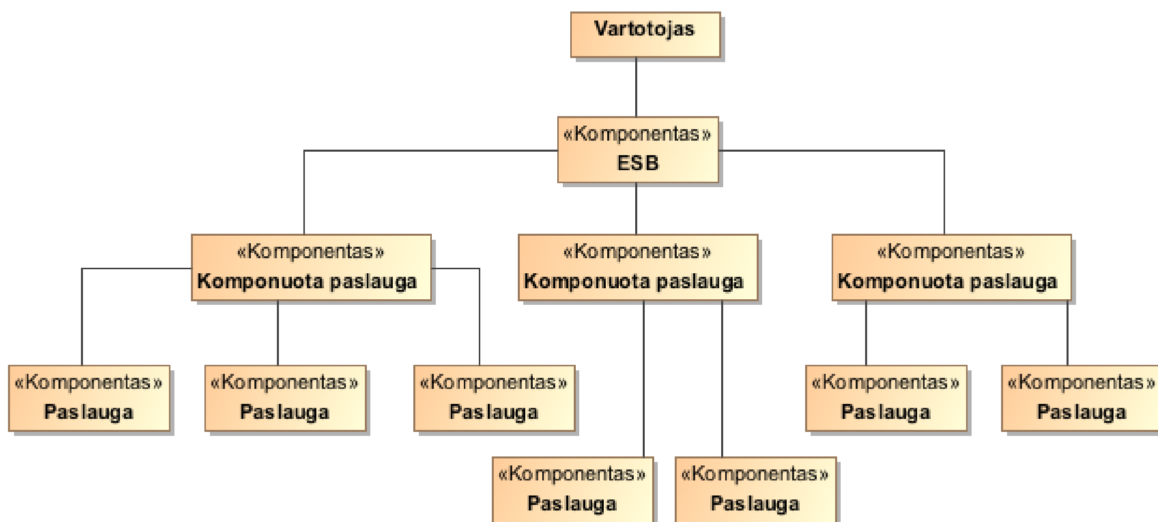
Atlikus rezultatų, gautų simuliacijos bei sistemos bandymų metu, palyginimą, pastebėta jog jie yra labai panašūs. Galime daryti prielaidą, jog modelis sukurtas vadovaujantis šia metodika gali atspindėti modeliuojamą sistemą.

10. Metodikos panaudojimo pavyzdžiai

Šiame skyriuje pateikiami, pagal „SOA” ir „EDA” architektūrinių šablonus, kuriamų sistemų modeliavimo pavyzdžiai. Šie architektūriniai šablonai pasirinkti, nes tai vieni populiariausių šablonų naudojamų šiuolaikinėse sistemose. Juose skiriasi sąryšių tipas, užklausos apdorojimo eiga.

10.1. Architektūros šablonas „SOA”

„SOA” (*angl. Service-Oriented Architecture*) – tai į paslaugas orientuotas architektūrinis šablonas. Šio šablono kontekste, paslauga – tai standartizuota pasikartojanti vartotojo atliekama užduotis. Pavyzdžiui, prekės likučio sandėlyje patikrinimo procedūra. „SOA” šablonas leidžia įvairių paslaugų rinkinį apjungti į vieną sistemą. Tokio tipo sistemose didelis dėmesys skiriamas integruojamumui ir modifikuojamumui. Sistemose, kuriamose pagal šį šabloną, veikimo logika yra paskirstoma atskirose paslaugose [Mik14]. Dažnai paslauga susideda iš kitų mažesnių paslaugų. Tokiu atveju ji laikoma komponuota paslauga. Visos komponuotos paslaugos apjungiamos vieno centrinio „ESB” (*angl. Enterprise Service bus*) komponento. Jis atlieka orkestravimo funkciją. Šis komponentas užtikrina, kad visos paslaugos naudotų vienodus protokolo standartus, prireikus jungia ir transformuoja skirtingus standartus. Tokioje sistemoje visos paslaugos yra derinamos tik prie centrinio komponento, todėl jas lengva pakeisti kitomis. Žemiau pateikiamas principinis tokios sistemos architektūros modelis.



8 pav. Principinis sistemos, įgyvendintos pagal „SOA” architektūrinį šabloną, modelis

Šiame modelyje išskiriami pagrindiniai keturi komponentai: „ESB”, „Paslauga”, „Komponuota paslauga” ir „Vartotojas”. Čia „Vartotojas” atspindi naudotojus, kurie reguliariai generuoja užklausas į sistemą. Vartotojai, nesvarbu ar tai asmenys, ar kitos sistemos turi ryšį tik su „ESB” komponentu. Tai atspindi vieną iš „SOA” architektūros principų: užklausos į sistemą gali būti pateikiamos tik per „ESB” komponentą. Šis komponentas sujungtas sinchroniniais ryšiais su komponentais „Komponuota paslauga”. „Komponuota paslauga” sujungta taip pat sinchroniniais ryšiais

ir su komponentais „Paslauga”. Tai reiškia, jog įvykdyti komponuota paslaugą reikia sulaukti visų ją sudarančių paslaugų atsako. Šiame modelyje komponuotos paslaugos atitinka „SOA” verslo procesų lygmenį.

10.1.1. Architektūros generavimas

Bendrai architektūros generavimą atlieka klasė „Architektūros generatorius”, ji privalo įgyvendinti metodą „gneruotiArchitektūrą”. Atsižvelgiant į aukščiau pateiktą principinį sistemos modelį, sukurtas šio metodo įgyvendinimas. Jis turi sugeneruoti agentus, vaizduojančius šiuos komponentus: „ESB”, komponuota paslauga, paslauga. Taip pat turi būti sugeneruoti agentai, vaizduojantys naudotoją ir serverį.

Siekiant pavaizduoti būsenos modeliavimą, nuspręsta atsižvelgti į situaciją, jog vykdymo metu komponentas gali būti sugadintas ir todėl užklausas apdoroti lėčiau. Modeliuojamoje situacijoje yra tiksliai nežinoma kokie komponentai yra sugadinti, todėl reikalinga įvesti į modelio konstravimą tikimybę, jog komponentas yra sugadintas. Taip pat nežinoma, kuomet sugadintas komponentas pradeda lėčiau apdoroti užklausas, todėl reikalinga sugadintiems komponentams įvesti tikimybę, jog nuo einamojo momento jie pradeda veikti lėčiau.

Architektūros generatorius turi atsižvelgti į galimybę sugeneruoti sugedusį komponentą, todėl įvesti šie architektūros generatoriaus parametrai:

- G_{skp} – tikimybė, jog komponuotos paslaugos komponentas yra sugedęs. Parametro reikšmė nurodoma intervale [0,1];
- G_{spp} – tikimybė, jog paprastos paslaugos komponentas yra sugedęs. Parametro reikšmė nurodoma intervale [0,1].

Siekiant neprisirišti prie konkrečios sistemos modeliavimo, generatoriui įvesta keletas papildomų parametrų:

- G_{kp} – komponuotų paslaugų kiekis;
- G_{pmin} – minimalus komponuotą paslaugą sudarančių paprastų paslaugų kiekis;
- G_{pmax} – maksimalus komponuotą paslaugą sudarančių paprastų paslaugų kiekis.

Šiems parametrams nustatytos reikšmės: $G_{kp} = 50$, $G_{pmin} = 1$, $G_{pmax} = 5$.

Atsižvelgiant į „SOA” architektūros šabloną bei šio darbo metodiką, agentams priskirti šie parametrai:

- agentas, vaizduojantis „ESB” komponentą: K_k = RANDOM, $K_{kmax} = 5$, $K_{kmin} = 1$, $K_w = 0$, $K_l = 0$, $K_e = 0{,}0001$, $K_{dw} = 0$, $K_{cpu} = 10\%$, $K_o = 5\%$, $K_d = 0.1\%$, $K_s = 1\%$, $K_{kt} = 2$;
- agentas, vaizduojantis komponuotos paslaugos komponentą: K_k = ALL, $K_w = 0$, $K_l = 0$, $K_e = 0{,}0001$, $K_{dw} = 0$, $K_{cpu} = 2\%$, $K_o = 2\%$, $K_d = 2\%$, $K_s = 2\%$, $K_{kt} = 5$;
- agentas, vaizduojantis paprastos paslaugos komponentą: $K_w = 0$, $K_l = 0$, $K_e = 0{,}002$, $K_{dw} = 0$, $K_{cpu} = 5\%$, $K_o = 2\%$, $K_d = 3\%$, $K_s = 0{,}5\%$, $K_{kt} = 10$;

- agentas, vaizduojantis vartotoją: V_{ut} = ASYNC, V_s = 1, V_t = 10, V_{umax} = 3, V_{umin} = 1, V_k = 0, V_{uk} = 500;
- agentas, vaizduojantis serverį: S_p = 100, S_a = 100, S_d = 100, S_t = 100, S_s = 1.

Modeliuojant reikalinga saugoti vidinę komponento būseną, nusakančią ar komponentas veikia „sugadintu" rėžimu ar įprastu. Būsenai saugoti naudojamas agento parametras K_{bv1}. Jei modeliuojamas komponentas yra sugedęs, modelio inicializavimo metu, jį vaizduojančiam agentui nustatoma šio parametro reikšmė „false". Tai nurodo, jog komponentas pradžioje veiks įprastu rėžimu. Šiame agente taip pat turi būti saugoma sugedimo tikimybė, tam naudojamas agento parametras K_{bv2}. Agentuose, vaizduojančiuose komponuotas paslaugas, modelio inicializavimo metu, šio parametro reikšmė nustatoma į K_{bv2} = 0.2, o agentuose, vaizduojančiuose paprastas paslaugas, K_{bv2} = 0.05. Tai reiškia, kad sugedę komponentai, kaskart apdorodami užklausas turi atitinkamai 0,2 ir 0,05 tikimybę sugesti ir todėl užklausas pradėti vykdyti ilgiau ir sunaudoti daugiau serverio resursų. Siekiant apibrėžti būsenos įtakos stiprumą komponento veiklai, įvedamas ir sulėtėjimo koeficientas. Jis nurodo, kiek lėčiau veiks sugedęs komponentas ir kiek daugiau serverio resursų reikės darbui atlikti. Koeficientui saugoti naudojamas agento parametras K_{bv3}. Serverio resursų sunaudojimas padidinamas 0,1 karto nuo sulėtėjimo koeficiento. Agentams, vaizduojantiems komponuotas paslaugas ir paprastas paslaugas, atitinkamai nustatytos šios parametro reikšmės: K_{bv3} = 20 ir K_{bv3} = 100. Modeliuojant būsenos įtaką komponento veiklai, pakoreguota agento elgsena taip, jog prieš apdorojant užklausą šis įvertintų tikimybę sugesti. Jei komponentas sugenda, pakeičiama komponento vidinė būsena, kurią atspindi agento parametras K_{bv1} = „true". Taip pat, agento elgsena pakoreguojama taip, jog jei komponentas veikia sugedusiu rėžimu, jis kurdamas užklausos apdorojimo užduotis, jų vykdymo laiką ir reikalingą serverių resursų kiekį padidina K_{bv3} karto. Kitaip tariant, modeliuojama situacija, jog sugedęs komponentas užklausas apdoroja K_{bv3} kartų ilgiau, bei reikalauja K_{bv3} kartų daugiau serverio resursų.

Generuojant modelį, pirmiausia inicializuojami vartotoją ir serverį vaizduojantys agentai. Vėliau, kuriamas agentas, vaizduojantis „ESB" komponentą, jis yra įeities į sistemą komponentas. Tuo pat metu sukuriamas ir asinchronins ryšys tarp agentų, vaizduojančių vartotoją ir „ESB" komponentą. Kitu etapu, inicializuojami agentai, vaizduojantys komponuotas paslaugas, šie sujungiami sinchroniniais ryšiais su agentu, vaizduojančiu „ESB" komponentą. Paskutiniu žingsniu kuriami agentai, vaizduojantys paprastas paslaugas. Jie taip pat sujungiami sinchroniniais ryšiais su agentais, vaizduojančiais komponuotas paslaugas. Kiekvieno agento inicializavimo metu sukuriamas ryšys su agentu, vaizduojančiu serverį. Inicializuotame modelyje vykdoma simuliacija.

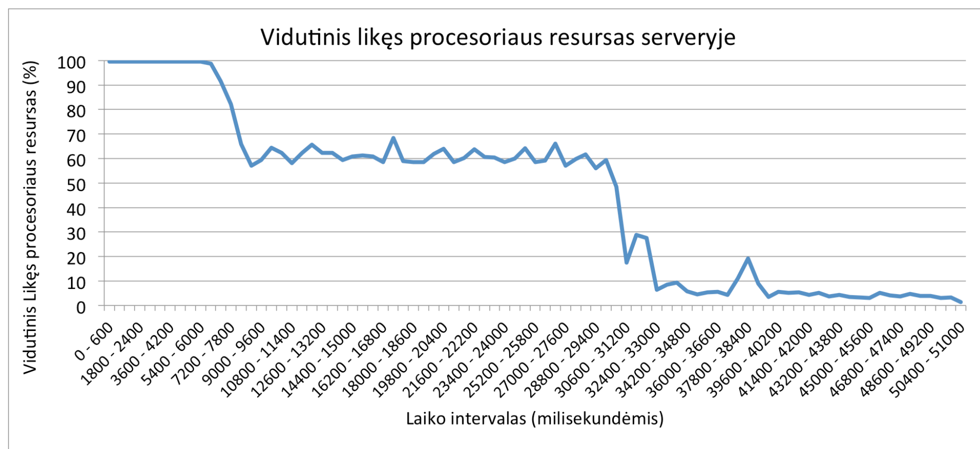
10.1.2. Rezultatai

Simuliacijos metu buvo renkami šie matai: užklausos vykdymo laikas, likęs procesoriaus resursas ir nesėkmingų užklausų kiekis. Atitinkamai kiekvienas iš jų leidžia įvertinti modeliuojamos sistemos kokybės atributus: sistemos greitaveiką, resursų panaudojimą bei klaidų tikimybę. Rezultatai pateikiami laiko perspektyvoje – x ašyje pateikiami laiko intervalai. Y ašyje pateikiami reikšmių vidurkiai. Toks vaizdavimas leidžia matyti kaip kito matai, kintant komponentų vidinei

būsenai. Užklausos vykdymo laikas ir jos statusas buvo registruojamas tik jai pasibaigus, nebuvo atsižvelgiama kuriuo laiku ji išsiųsta. Pavyzdžiui, užklausa, kuri išsiųsta laiko intervale [0;600) ir pasibaigusi intervale [600;1200), buvo įtraukta į intervalo [600;1200) rezultatus.

9 pav. Vidutinis užklausos vykdymo laikas „SOA" architektūros modelyje

Vidutinio užklausų vykdymo laiko grafike matomi du lūžiai: pirmasis ties laiko intervalu [7200;7800), antrasis ties laiko intervalu [2700;27600). Pirmasis lūžis įvyko sugedus komponuotų paslaugų komponentams, o antrasis pradėjus gesti paprastų paslaugų komponentams. Komponuotų paslaugų sulėtėjimo koeficientas gerokai mažesnis, negu paprastų paslaugų, todėl ir užklausų apdorojimo laikas kito ne taip žymiai kaip sugedus paprastų paslaugų komponentams. Komponuotų paslaugų sulėtėjimo koeficientas 20, o paprastų paslaugų 100. Galima pastebėti, jog sugedus komponuotų paslaugų komponentams užklausų vykdymo laikų svyravimai nebuvo tokie dideli kaip sugedus paprastų paslaugų komponentams. Mažesnieji svyravimai atsiranda, nes „ESB" komponentas kviečia atsitiktinį skaičių komponuotų paslaugų, tarp kurių pasitaiko ir nesugedusių komponentų, kurie apdoroja užklausas įprastu greičiu. Taigi, užklausos vykdymo laiko padidėjimas rodo, kad tame laiko intervale buvo kviesta daugiau kartų sugedęs komponuotos paslaugos komponentas. Sugedus paprastų paslaugų komponentams, svyravimai matomi didesni, dėl to, jog sulėtėjimo koeficientas yra didesnis. Kadangi, paprasta paslauga yra sujungta sinchroniniu ryšiu su komponuota paslauga, atvejais, kuomet kviečiama komponuota paslauga įtraukianti sugedusią paprastą paslaugą, vykdymo laikas žymiai išauga.



10 pav. Vidutinis likęs procesoriaus resursas serveryje, „SOA" architektūros modelyje

Procesoriaus resursų grafike matomi taip pat du lūžiai, kurie atspindi tą pačią situaciją: komponuotų paslaugų komponentų sugedimą, o vėliau ir paprastų paslaugų komponentų sugedimą. Pradžioje, kol visi komponentai veikė įprastu rėžimu, procesoriaus resursas naudojamas labai nežymiai. Sugedus komponuotų paslaugų komponentams, nuo intervalo [7200;7800), matomas stiprus procesoriaus resurso sunaudojimo šuolis. Toliau grafike matomi svyravimai iki intervalo [27000;27600). Taip yra, nes „ESB" komponentas kviečia atsitiktines komponuotas paslaugas, tarp kurių yra sugedusių komponentų. Toliau matomas dar stipresnis laisvo procesoriaus resurso kritimas, kaip ir minėta tai nutinka sugedus paprastų paslaugų komponentams. Nuo šio taško svyravimų beveik nėra, nes beveik visas procesoriaus resursas būna išnaudotas. Neapdorotos vartotojo užklausos statomos į eilę prie „ESB" komponento.

11 pav. Nesėkmingų užklausų skaičius „SOA" architektūros modelyje

Nesėkmingų užklausų grafike matomas vienas lūžis, kuris atspindi sumažėjusį apdorojamų užklausų kiekį. Kadangi modeliuojamas komponentų sugedimas neturi įtakos užklausų apdorojimo klaidoms, sugedus komponentui nesėkmingų užklausų skaičius nekinta. Kitaip tariant komponento vidinė būsena, nusakanti ar komponentas yra sugedęs neturi įtakos parametrui K_e. Tikimybė jog apdorojama užklausa pasibaigs klaida išlieka tokia pat nepaisant ar komponentas sugedęs ar ne. Grafike matomas lūžis atsirado, nes sugedus paprastoms paslaugoms, procesoriaus resursas buvo

visiškai išnaudotas, o dėl to sistema apdorojo mažiau užklausų. Tuo pačiu ir nesėkmingų užklausų kiekis sumažėjo.

10.2. Architektūros šablonas „EDA"

„EDA" (*angl. Event-driven architecture*) – tai įvykiais grįstas architektūrinis šablonas. Sistemos, kurios kuriamos remiantis šiuo šablonu, veikia kurdamos įvykius, bei į juos reaguodamos. Kitaip tariant sistemoje vykstantys procesai pagrindinai tarpusavyje komunikuoja per įvykių propagavimą. Dažnai įvykių sklaida vykdoma pasitelkiant publikavimo/prenumeratos architektūrinį stilių. Tokiu atveju, sistemoje nutikęs įvykis yra pateikiamas tik tiems procesams, kurie jo laukia. Šiame architektūros šablone galima išskirti tris pagrindines dalis: įvykius, įvykių magistrales bei įvykių jutiklius.

Įvykis – tai nutikimas sistemoje arba jos išorėje, apie kurį yra pranešama sistemai. Įvykus įvykiui, jis yra publikuojamas sistemos komponentams, pastarųjų atpažįstamas bei apdorojamas. Formaliai, patys įvykiai nėra siunčiami sistemai, siunčiamas tik pranešimas apie įvykusį įvykį. Dėl patogumo, šios žinutės dažnai vadinamos tiesiog įvykiais. Dažniausiai tai yra asinchroninės žinutės. Pavyzdžiui, tai gali būti žinutė apie vartotojo klaviatūra paspaustą mygtuką. Įvykių siuntėjai nežino informacijos apie gavėją, netgi nežino ar toks egzistuoja. Įvykius įvykiui šis yra perduodamas į įvykių magistralę.

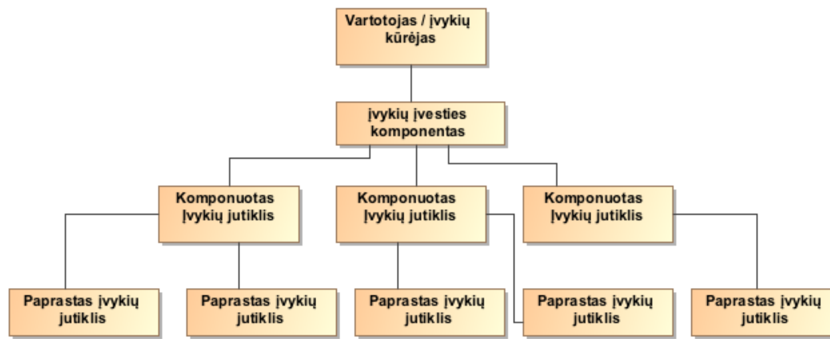
Įvykių magistralė – tai mediumas, kuriuo yra siunčiamas įvykis tarp įvykio publikuotojo ir įvykio jutiklio. Jis yra atsakingas, jog publikuotas įvykis būtų korektiškai išsiunčiamas kitiems sistemos komponentams. Įvykio prenumeratos atveju, magistralė atsakinga, jog prenumeratoriai gautų jiems skirtus įvykius. Įvykių magistralė turi informaciją apie sistemoje egzistuojančius jutiklius, todėl išsiųstą įvyki gali toliau perduoti jį priimantiems jutikliams.

Įvykio jutiklis – tai sistemos dalis, kuri atsakinga už reagavimą į įvykusius įvykius. Reaguojant dažniausiai yra iškviečiama nurodyta funkcija. Atsiradus naujam įvykių jutikliui sistemoje, jis priregistruojamas įvykių magistralėje. Tik taip jis gali gauti sistemoje vykstančius įvykius.

Sistemoje, sukurtoje remiantis „EDA" šablonu, komponentai neturi tiesioginio tarpusavio ryšio, tai leidžia į sistemą lengviau integruoti naujus komponentus, ją plėsti. Kitaip tariant, tokioje sistemoje nenutinka taip, jog komponentas tiesiogiai kvies kito komponento funkcionalumą. Bendravimas tarp komponentų vyksta įvykių pagalba. Tačiau tokią sistemą suvokti yra sudėtingiau, nes ne visuomet lengva suprasti aiškią veiksmų seka vykstančią sistemoje. Kuriant tokią sistemą, labai svarbu teisingai išskaidyti sistemoje atliekamų veiksmų seką.

10.2.1. Architektūros generavimas

Generuojant „EDA" architektūrą nuspręsta sukurti agentus vaizduojančius šiuos elementus: įvykių įvesties komponentas, komponuoto įvykių jutiklio komponentas, paprasto įvykių jutiklio komponentas, vartotojas ir serveris.



12 pav. Principinis sistemos, įgyvendintos pagal „EDA" architektūrinį šabloną, modelis

Generuojamame modelyje, įvykių įvesties komponentas sujungtas asinchroniniais ryšiais su komponuotais įvykių jutikliais. Pastarieji sujungti taip pat asinchroniniais ryšiais su paprastais įvykių jutikliais.

Kaip ir „SOA" architektūros atveju, siekiant pavaizduoti būsenos modeliavimą, nuspręsta atsižvelgti į situaciją, jog komponentas vykdymo metu gali būti sugadintas ir pradėti užklausas apdoroti lėčiau. Čia architektūros generatoriui taip pat įvedami papildomi parametrai:

- G_{skj} – tikimybė, jog komponuotas įvykių jutiklis yra sugedęs. Parametro reikšmė nurodoma intervale [0,1];

- G_{spj} – tikimybė, jog paprastas įvykių jutiklis yra sugedęs. Parametro reikšmė nurodoma intervale [0,1].

Siekiant neprisirišti prie konkrečios sistemos modeliavimo, generatoriui įvesta keletas papildomų parametrų:

- G_{kj} – komponuotų jutiklių kiekis;

- G_{jmin} – minimalus komponuoto jutiklių generuojamų įvykių prenumeratorių kiekis (paprastų jutiklių);

- G_{jmax} – maksimalus komponuoto jutikliu generuojamų įvykių prenumeratorių kiekis (paprastų jutiklių).

Šiems parametrams, kaip ir „SOA" atveju, nustatytos reikšmės: G_{kp} = 50, G_{pmin} = 1, G_{pmax} = 5.

Generuojamiems agentams nustatyti atitinkami parametrai:

- agentui, vaizduojančiam įvykių įvesties komponentą: K_k = RANDOM, K_{kmax} = 5, K_{kmin} = 1, K_w = 0, K_l = 0, K_e = 0,01, K_{dw} = 0, K_{cpu} = 10%, K_o = 5%, K_d = 0.1%, K_s = 1%, K_{kt} = 2;

- agentui, vaizduojančiam komponuotų įvykių jutiklių komponentą: K_k = RANDOM, K_w = 0, K_l = 0, K_e = 0,0001, K_{dw} = 0, K_{cpu} = 2%, K_o = 2%, K_d = 2%, K_s = 2%, K_{kt} = 5;

- agentui, vaizduojančiam paprastų įvykių jutiklių komponentą: K_w = 0, K_l = 0, K_e = 0,002, K_{dw} = 0, K_{cpu} = 5%, K_o = 2%, K_d = 3%, K_s = 0,5%, K_{kt} = 10;

- agentui, vaizduojančiam vartotoją: V_{ut} = ASYNC, $V_s = 1$, $V_t = 10$, $V_{umax} = 3$, $V_{umin} = 1$, $V_k = 0$, $V_{uk} = 500$;
- agentui, vaizduojančiam serverį: $S_p = 100$, $S_a = 100$, $S_d = 100$, $S_t = 100$, $S_s = 1$.

Kaip ir „SOA" architektūros modelyje, čia gali būti sugeneruoti sugedę komponentai. Todėl modelio inicijavimo metu, agente, vaizduojančiame komponuotą jutiklio komponentą, nustatomas parametras $K_{bv2} = 0.2$, o agente, vaizduojančiame paprasto jutiklio komponentą, $K_{bv2} = 0.05$. Tai reiškia, kad sugedę komponentai, kaskart apdorodami įvykius turi atitinkamai 0,2 ir 0,05 tikimybę sugesti ir pradėti užklausas vykdyti ilgiau pareikalaudami daugiau serverio resursų. Agentams, vaizduojantiems komponuotų jutiklių komponentus ir paprastų jutiklių komponentus, atitinkamai nustatyti šie sulėtėjimo koeficientai: $K_{bv3} = 20$ ir $K_{bv3} = 100$.

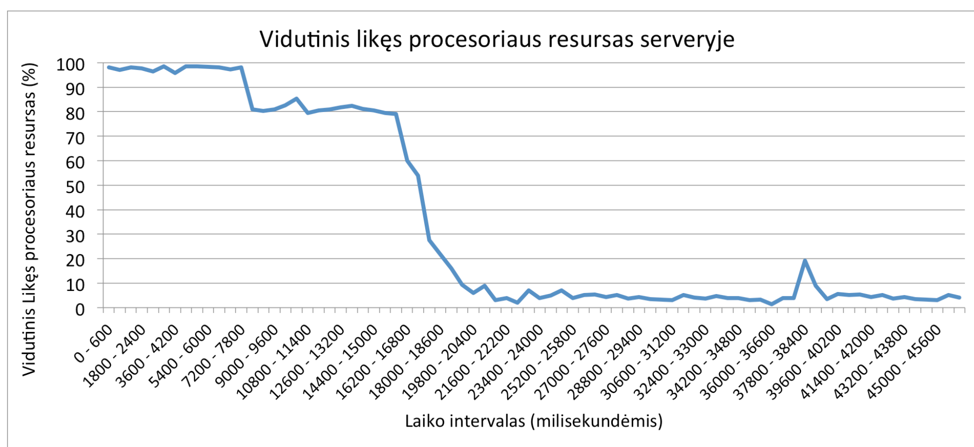
Generuojant modelį pirmiausia sukuriami agentai, vaizduojantys vartotoją ir serverį. Vėliau, kuriamas agentas, vaizduojantis įvykių įvesties komponentą. Jis yra vartotojų užklausų įėjimo į sistemą taškas. Sukuriamas asinchronins ryšys tarp agentų, vaizduojančių vartotoją ir įvesties komponentą. Kitu etapu inicializuojami agentai, vaizduojantys komponuotų jutiklių komponentus. Pastarieji sujungiami asinchroniniais ryšiais su įvesties komponentu. Vėliau kuriami agentai, vaizduojantys paprastų jutiklių komponentus, jie asinchroniniais ryšiais sujungiami su komponuotų jutiklių agentais. Kuriant kiekvieną agentą vaizduojantį komponentą sukuriamas ryšys su agentu vaizduojančiu serverį. Kitu etapu vykdoma simuliacija.

10.2.2. Rezultatai

Simuliacijos metu buvo renkami užklausos vykdymo laiko, nesėkmingų užklausų kiekio bei likusio procesoriaus resurso matai. Rezultatai pateikiami laiko perspektyvoje. Grafiko X ašyje vaizduojami laiko intervalai, o Y ašyje pateikiami matų reikšmių vidurkiai. Užklausų apdorojimo laikas buvo registruojamas laikantis tokių pat principų kaip ir „SOA" architektūros modelio bandymo atveju.

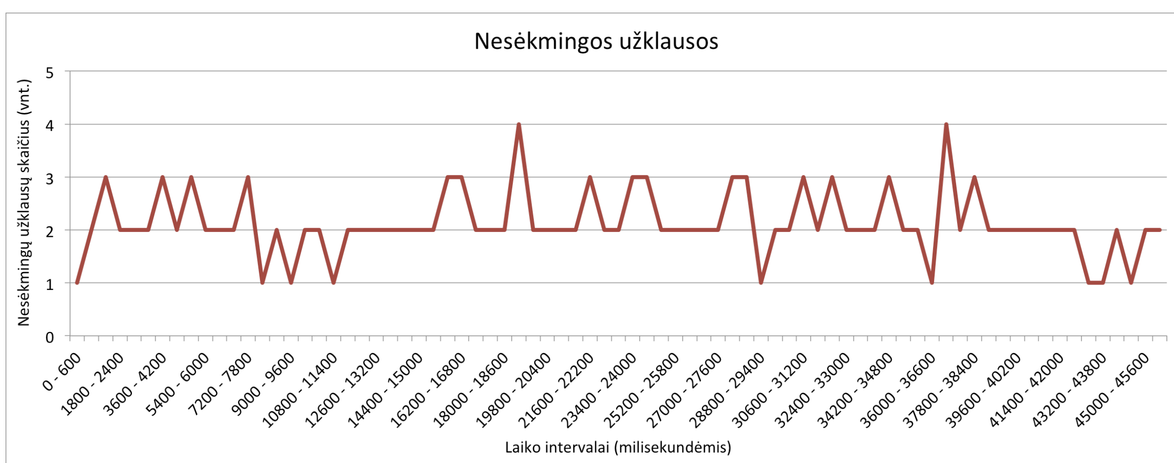
13 pav. Vidutinis užklausos vykdymo laikas „EDA" architektūros modelyje

Grafike matomas lūžis ties intervalu [16200; 16800), jis atspindi paprastų įvykių jutiklių komponentų sugedimą. Kadangi šio tipo komponentų sistemoje daugiausia ir jų sulėtėjimo koeficientas gerokai didesnis, jų sulėtėjęs darbas daro didesnę įtaką užklausų apdorojimo laikui nei komponuotų jutiklių komponentų sugedimas. Kuris šiame grafike beveik net nepastebimas. Iki sugendant komponentams užklausų apdorojimo laikas buvo stabilus, o daliai komponentų sugedus pastebima vykdymo laiko šuoliai. Taip yra dėl to, jog įvykiai yra siunčiami atsitiktine tvarka paprastiems jutiklių komponentams, o dalis iš jų yra nesugedę ir veikai įprastu rėžimu. Vykdymo laiko pakilimai rodo, jog tuo laiko intervalu buvo daugiau kartų kreipiamasi į sugedusį komponentą. Verta pastebėti, jog vykdymo laikas neišaugo akimirksniu, o didėjo gan tolygiai. Taip yra, nes komponentai sugedo ne visi iškart, o vienas po kito.



14 pav. Vidutinis likęs procesoriaus resursas serveryje, „EDA" architektūros modelyje

Grafike matomi du lūžiai, pirmasis lūžis įvyksta sugedus komponuotų įvykių jutiklių komponentams, jie pradeda reikalauti daugiau sistemos resursų. Antrasis lūžis įvyksta kuomet sugenda paprastų įvykių jutiklių komponentai. Grafike matome, jog procesoriaus resursas išnaudojamas beveik visiškai. Tai atsispindi ir vidutinio užklausos vykdymo grafike, matome, jog įvykus lūžiui, vykdymo laikas nors ir svyruoja bet po truputi nuolat auga. Taip yra, nes neužtenkant procesoriaus resurso, užklausos yra statomos į eiles prie jutiklių, kol atsilaisvina procesoriaus resursas. Dėl atsirandančių eilių ir bendras užklausos vykdymo laikas tampa ilgesnis.



15 pav. Nesėkmingų užklausų skaičius „EDA" architektūros modelyje

Grafike nepastebima ryškių pokyčių, nesėkmingų užklausų skaičius nuolat svyruoja, tačiau išlieka panašus. Tokia situacija matoma, nes komponentai yra sujungti asinchroniniais ryšiais, o tai reiškia, jog tik vartotojui išsiuntus užklausą į įvesties komponentą, užklausa pažymima kaip sėkminga ir toliau kuriamos užklausos iš šio komponento. Taigi, užklausa gali tapti nesėkminga tik įvesties komponente. Šiuo atveju šis modeliuojamas komponentas turi klaidos tikimybę K_e = 0.01.

10.3. Architektūros šablonų „SOA" ir „EDA" palyginimas

Šiame skyriuje pateikiami bandymai, kurie leidžia palyginti pagal „SOA" ir „EDA" architektūrinius šablonus sukurtas sistemas. Bandymai atlikti remiantis šiame darbe apibrėžta metodika. Siekiant neprisirišti prie konkrečių sistemų modeliavimo, naudojami architektūros generatoriai. Jie atsižvelgdami į nurodytus parametrus bei naudojamą architektūrinį šabloną sugeneruoja modelį, vaizduojantį sistemą sukurtą pagal pasirinktą architektūrinį šabloną. Siekiant palyginti sistemas sukurtas pagal „SOA" ir „EDA" architektūrinius šablonus, abiem atvejais kuriamas panašios topologijos sistemos modelis. Pagrindinių „SOA" architektūros sistemos modelių elementų išeities kodas pateikimas priede nr.2.

10.3.1. Architektūrų generavimas

Abiejuose architektūros modeliuose generuojama toks pat kiekis agentų vaizduojančių serverį ir naudotoją: 1 naudotojas, 1 serveris. Komponentus vaizduojančių agentų generuojama tiek, jog tiek viename tiek kitame modelyje vienai užklausai apdoroti būtų iškviečiamas panašus kiekis agentų vaizduojančių komponentus. „SOA" atveju sugeneruojamas 1 „ESB" komponentą vaizduojantis agentas bei 20 komponuotas paslaugas modeliuojančių agentų. Kiekvieną modeliuojamą komponuotą paslaugą sudaro nuo 3 iki 20 paprastų paslaugų, sugeneruojami ir šias paslaugas vaizduojantys agentai. „EDA" atveju sugeneruojama agentai vaizduojantys 1 įvykių įvesties komponentą, 20 komponuoto įvykių jutiklių komponentų (atitinka „SOA" komponuotą paslaugą). Kiekvieną iš jų sudaro nuo 3 iki 20 paprastų įvykių jutiklių vaizduojančių agentų (atitinka „SOA" paprastą paslaugą).

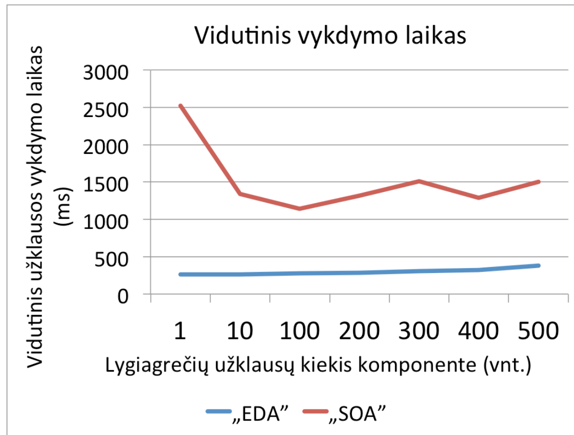
Atliekant bandymus keičiamas tik vienas pasirinktas parametras. Pradiniai parametrai:

- agentas, vaizduojantis komponentą: K_k = RANDOM, K_{kmax} = 20, K_{kmin} = 2, K_w = 0, K_l = 100, K_e = 0,001, K_{dw} = 0, K_{cpu} = 0.2%, K_o = 0.1%, K_d = 0.1%, K_s = 0.2%, K_{kt} = 5;
- agentas, vaizduojantis vartotoją: V_{ut} = ASYNC, V_s = 1, V_t = 10, V_{umax} = 1, V_{umin} = 1, V_k = 0, V_{uk} = 0, V_{ul} = 1000;
- agentas, vaizduojantis serverį: S_p = 1000, S_a = 1000, S_d = 1000, S_t = 1000, S_s = 1.

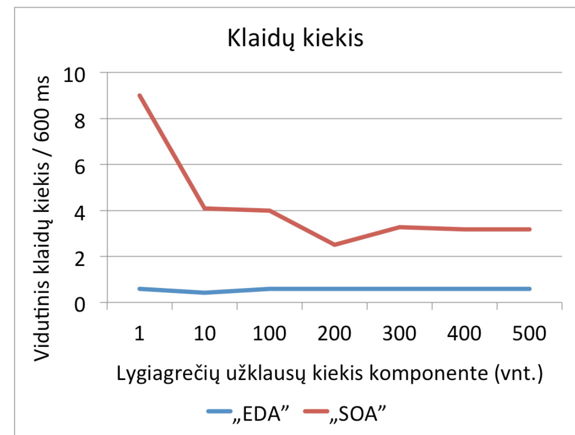
10.3.2. Bandymas: kintantis lygiagrečių užklausų kiekis komponente

Vienas iš „SOA" ir „EDA" architektūrinių šablonų skirtumų, tai ryšiai tarp komponentų: „EDA" aveju – asinchroniniai, o „SOA" – sinchroniniai. Antruoju atveju, komponentas gavęs už-

klausą ir išsiuntęs vaikines užklausas, privalo sulaukti jų atsako prieš priimdamas naują užklausą (jei yra pasiektas maksimalus lygiagrečiai apdorojamų užklausų kiekis). Vienu metu komponentas gali apdoroti užklausų K_l. Esant asinchroniniam ryšiui komponentas nelaukia vaikinių užklausų pabaigos ir priima naujas užklausas. Šis bandymas skirtas ištirti kaip kinta užklausų apdorojimo laikas ir klaidų kiekis kintant lygiagrečių užklausų kiekiui komponente K_l.



16 pav. Vidutinis užklausos vykdymo lakas



17 pav. Klaidų kiekis

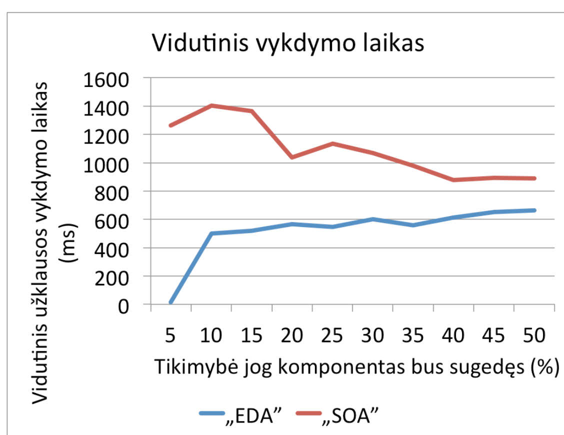
Grafike matyti, jog esant mažam lygiagrečių užklausų kiekiui komponente, „SOA" architektūros modelyje vykdymo laikas ir klaidų kiekis yra stipriai padidėjęs lyginant su „EDA" architektūros modeliu. Taip yra todėl, nes „SOA" atveju visos užklausos į sistemą patenka per „ESB" komponentą. Šiame komponente pasiekus lygiagrečiai apdorojamų užklausų limitą, naujos užklausos turi laukti kol nors viena jau pradėta užklausa bus baigta. Laukiančioms užklausoms pasiekus laukimo limitą V_{ul}, jos pasibaigia klaida.

„EDA" architektūros modelyje vykdymo laikai ir klaidų kiekis yra mažesni, dėl asinchroninių ryšių tarp komponentų. Čia nesusidaro didelės užklausų eilės prie komponentų, nes nereikia laukti kol bus baigtos vykdyti vaikinės užklausos. „EDA" architektūros modelyje užklausos yra apdorojamos lygiagrečiai.

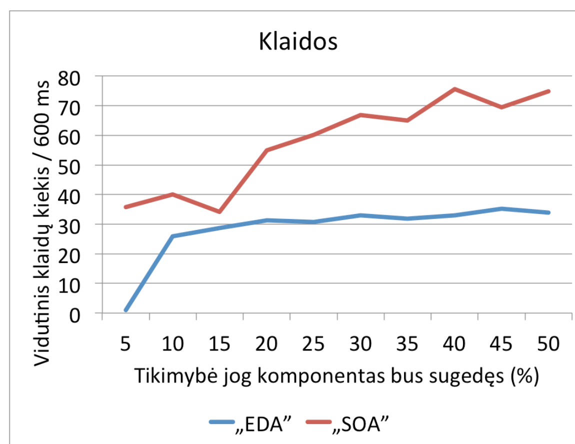
10.3.3. Bandymas: kintanti tikimybė, jog komponentas yra sugedęs

Dėl skirtingų ryšių abiejų tipų architektūros skirtingai reaguoja į būsenų sistemoje pokyčius. Vienas iš galimų komponentų būsenos pavyzdžių tai būsena nusakanti jo veikimo rėžimą. Šiame bandyme modeliuojami komponentai gali veikti įprastu rėžimu arba sugadintu. Bandyme laikoma, jog sugedęs komponentas užklausas apdoroja 20 kartų lėčiau. Bandymo metu keičiamas architektūros generatoriaus parametras, nusakantis, ar generuojamas komponentas veikia įprastu ar sugedusiu rėžimu. Stebimos kokybinės charakteristikos: vykdymo laikas ir klaidų kiekis.

Atlikus bandymą pastebėta, jog klaidų kiekis didėjant tikimybei jog komponentas veikia sugedusiu rėžimu, „SOA" architektūros modelyje augo sparčiau nei „EDA" architektūros modelyje. Tokia situacija susidarė, kaip ir praeitame bandyme dėl susidarančių užklausų eilių sistemoje. Užklausos nepriimtos į komponentą apdoroti greičiau nei V_{ul} pasibaigia nesėkme. Kadangi lygiagrečiai apdorojamų užklausų kiekis komponente nekinta, komponentui pasiekus maksimalų lygia-



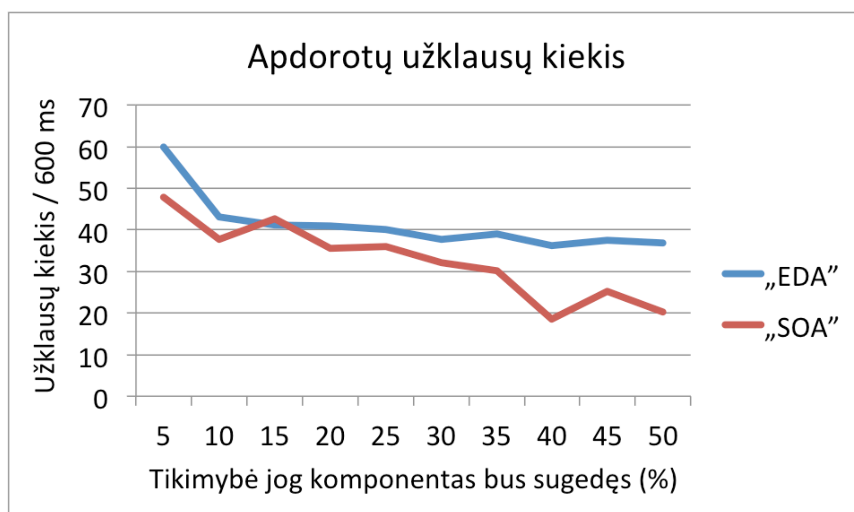
18 pav. Vidutinis užklausos vykdymo lakas



19 pav. Klaidų kiekis

grečiai apdorojamų užklausų kiekį naujos užklausos statomos į eilę. Grafike matome, jog „EDA” architektūros modelyje tiek klaidų kiekis tiek vidutinis vykdymo laikas augo stabiliai.

Įdomu pastebėti, didėjant tikimybei jog komponentas veikia sugedusiu rėžimu, „SOA” architektūros modelyje vidutinis užklausos vykdymo laikas mažėjo. Ši situacija susidaro, nes vis mažiau užklausų patenka į sistemą, o patekusios dėl mažesnio apkrovimo apdorojamos greičiau.

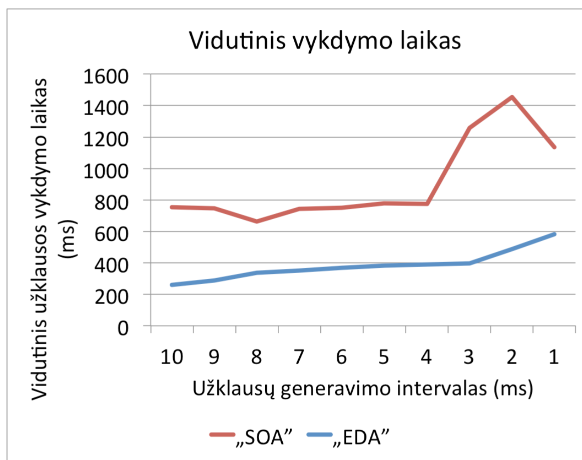


20 pav. Užklausų kiekis

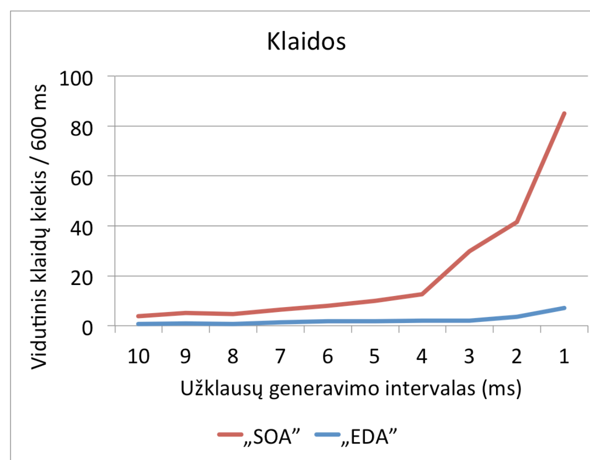
10.3.4. Bandymas: kintantis užklausų generavimo dažnis

Šis bandymas skirtas ištirti kaip „EDA” ir „SOA” architektūrų modeliuose kinta užklausų vykdymo laikas, klaidų skaičius, sunaudojamų serverio resursai bei apdorotų užklausų kiekis keičiant vartotojo generuojamų užklausų dažnį V_t. Vienu užklausų generavimo momentu sugeneruojama 1 užklausa. Taigi, V_t nusako kas kiek laiko bus generuojama užklausa į sistemą. Kiti modelio parametrai lieka nekeičiami.

Rezultatų grafikuose matoma, jog „SOA” architektūros modelyje tiek vykdymo laikas, tiek klaidų skaičius didėja, tačiau išlieka gan stabilus iki 4ms užklausų generavimo dažnio. Toliau pastebimas ryškus vykdymo laiko bei klaidų šuolis. Toks pokytis įvyksta, nes sistema nebespėja apdoroti užklausų ir pradeda susidaryti užklausų prieš komponentus eilės. Įdomu tai, jog užklausų

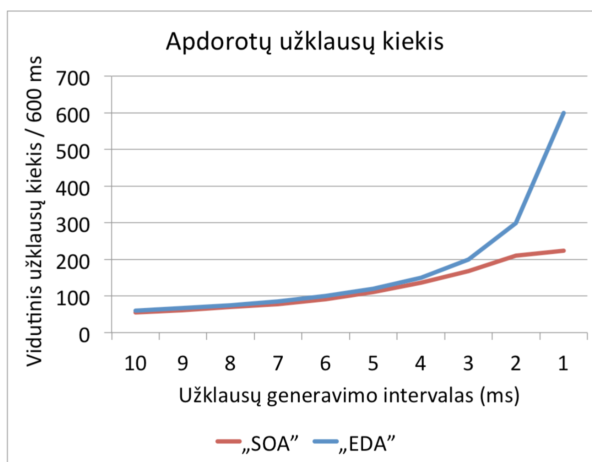


21 pav. Vidutinis užklausos vykdymo lakas

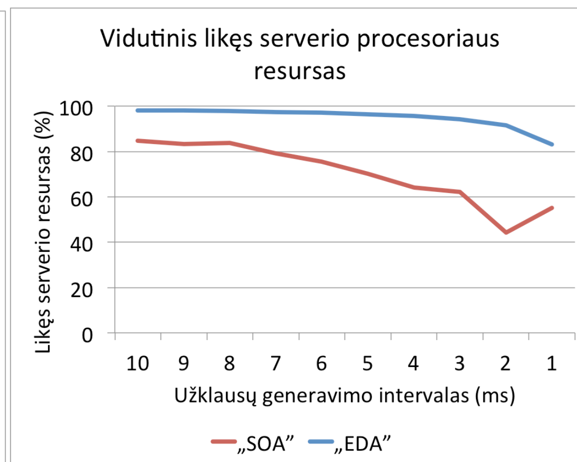


22 pav. Klaidų kiekis

generavimo dažniui pasiekus 1 ms vykdymo laikas sumažėja, o klaidų kiekis dar labiau padidėja. Serverio resursų grafike matoma, jog ir resursų minėtame taške lieka daugiau. Kaip ir kituose tyrimuose čia yra pasiekiamas taškas, kuomet daug užklausų yra atmetama dar prieš priimant jas į sistemą apdoroti. Tai matoma ir apdorotų užklausų grafike: apdorotų užklausų kiekis nepadidėja esant generavimo dažniui 1ms, lyginant su 2ms dažniu. Taigi, dažniui sumažėjus nuo 2ms iki 1ms, dauguma papildomai sugeneruotų užklausų yra atmetamos dar nepradėjus jų apdoroti.



23 pav. Vidutinis užklausų kiekis



24 pav. Likęs serverio procesoriaus kiekis

Kadangi „EDA” architektūros modelyje užklausos apdorojamos greičiau, tai ir resursų sunaudojama mažiau. Grafike matomas didesnis likęs procesoriaus resurso kiekis nei „SOA” architektūros modelyje.

11. Rezultatai

Šiuo magistriniu darbu buvo siekiama sukurti metodiką, kuri nusakytų kaip reikia modeliuoti programų sistemas, kuriose yra komponentų su būsenomis, taikant agentais grįstą modeliavimą. Siekiant tikslo gauti šie rezultatai:

- Išskirti specifiniai agentais grįsto modelio kūrimo žingsniai.
- Pateikti literatūroje aptariamų programų sistemų modeliavimo praktikų privalumai ir trūkumai.
- Remiantis literatūros apžvalga išskirtos programų sistemų būsenos ir jų įtaka sistemos veiklai, bei atsižvelgiant į apimties ir laiko dimensijas pateikta šių būsenų klasifikacija.
- Suformuluota agentais grįsto modeliavimo metodika, nusakanti, kaip reikia modeliuoti programų sistemas, kuriose yra komponentų su būsenomis.
- Naudojant pavyzdinę sistemą ir jos modelį atliktas metodikos validavimas.
- Pateikti programų sistemų, sukurtų pagal „SOA" ir „EDA" architektūrinius šablonus, modeliavimo ir simuliavimo pavyzdžiai naudojant „Repast Simphony" programavimo karkasą.

12. Išvados

Šiuolaikinės programų sistemos susideda iš skirtingas funkcijas atliekančių sudėtinių dalių. Šių dalių veikla dažnai tarpusavyje susijusi. Deja, vis dar nėra vieningai pripažinto programų sistemos architektūros apibrėžimo, tačiau dauguma autorių komponentus išskiria kaip svarbias sistemos dalis. Kuriant sistemos architektūra atsižvelgiama ne tik į funkcinius bet ir nefunkcinius reikalavimus. Pastarieji apibrėžiami architektūrų kokybinėmis charakteristikomis. Projektavimo etape programų sistemų architektūrų kokybinės charakteristikos gali būti vertinamos matematiniais skaičiavimais ar empiriniais tyrimais. Vertinant programų sistemas matematiniais skaičiavimais grįstais būdais, galima pasiekti aukštą rezultatų tikslumą, tačiau tokie būdai turi nemažai apribojimų, pavyzdžiui sunku imituoti tikrame gyvenime kylančius naudojimo scenarijus. Taikant empiriniais tyrimais paremtus vertinimo būdus kuriami sistemos modeliai, juose atliekamos simuliacijos. Sėkmingas dinamiškų ir kompleksiškų programų sistemose vykstančių procesų modeliavimas dažnai tampa sudėtingas, nes sunku numatyti visus sistemos veikimo variantus. Šiame darbe naudojamas agentais grįstas modeliavimas leidžia modelį pradėti kurti nuo mažų nesudėtingų elementų ir tik vėliau juos sujungti į vientisą sistemą, todėl modeliavimo procesas tampa lengvesnis.

Remiantis atlikta literatūros apžvalga, sukurta agentais grįsta programų sistemos modeliavimo metodika, skirta dar sistemos projektavimo etape palyginti keletą programų sistemos architektūrų. Metodikoje pateikiami patarimai kokiais agentais grįsto modelio elementais reikia modeliuoti sistemos architektūrinius elementus.

Darbo metu nustatytos šios išvados:

- Pagal pateikiamą metodiką sukonstruoto modelio simuliacija leidžia dar sistemos projektavimo etape ištirti šias programų sistemos kokybines savybes: greitaveika, klaidų tikimybė, resursų panaudojimas.
- Dėl sistemos veiklos susiformavusi būsena dažnai daro įtaką tolimesniems sistemos veiksmams, todėl būseną svarbu įtraukti ir sistemos modelį.
- Komponento būsena gali būti adekvačiai modeliuojama kaip agento, vaizduojančio komponentą, atributai, kurių reikšmės kinta priklausomai nuo komponento aptarnaujamų užklausų, komponento kokybines savybes apibrėžiant kaip funkciją nuo šių atributų.
- Vartotoją apimanti būseną gali būti adekvačiai modeliuojama kaip agento, vaizduojančio vartotoją, atributai, kurių reikšmės kinta priklausomai nuo generuojamų užklausų.

Darbe pateikiamą metodiką galima pagerinti išnagrinėjant sistemos sunaudojamų resursų kiekio įvertinimo metodikas bei pateikiant patarimus kaip šios metodikos atveju reikėtų vertinti sunaudojamus resursus. Giliau išnagrinėti sistemos naudotojo elgsenos modeliavimą ir pateikti patarimus, kaip galėtų būti modeliuojama skirtinga naudotojų elgsena.

Literatūra

[AM11] Daniel Amyot ir Gunter Mussbacher. User requirements notation: the first ten years, the next ten years (invited paper). *Journal of software*, 6(5), 2011.

[AO99] *Tools'99: proceedings of the technology of object-oriented languages and systems*. IEEE Computer Society, Washington, DC, USA, 1999. ISBN: 0-7695-0278-4.

[BBC+04] P Botella, X Burgues, JP Carvallo, X Franch, G Grau, J Marco ir C Quer. Iso iec 9126 in practice: what do we need to know? *Proceedings of the first software measurement european forum (smef)*, 2004.

[BCK03] Len Bass, Paul Clements ir Rick Kazman. *Software architecture in practice (2nd edition)*. Addison-Wesley Professional, 2003.

[BD98] Spitznagel B ir Garlan D. Architecture-based performance analysis. *Proceedings of the tenth international conference on software engineering and knowledge engineering*. San Francisco Bay, USA, 1998.

[BGL14] Verónica Bogado, Silvio Gonnet ir Horacio Leone. Modeling and simulation of software architecture in discrete event system specification for quality evaluation. *Simulation*, 90(3):290–319, 2014-03. ISSN: 0037-5497. DOI: `10.1177/0037549713518586`. URL: `http://dx.doi.org/10.1177/0037549713518586`.

[BGM+06] Steffen Becker, Lars Grunske, Raffaela Mirandola ir Sven Overhage. Performance prediction of component-based systems: a survey from an engineering perspective. *Architecting systems with trustworthy components, volume 3938 of lncs*. Springer, 2006, p. 169–192.

[BHK06] Steffen Becker, Jens Happe ir Heiko Koziolek. Putting Components into Context: Supporting QoS-Predictions with an explicit Context Model. *Proc. 11th international workshop on component oriented programming (wcop'06)*. Ralf Reussner, Clemens Szyperski ir Wolfgang Weck, redaktoriai, 2006-07, p. 1–6.

[Buh99] R. J. A. Buhr. Making behaviour a concrete architectural concept. *Proceedings of the thirty-second annual hawaii international conference on system sciences-volume 8 - volume 8*. HICSS '99. IEEE Computer Society, Washington, DC, USA, 1999, p. 8065–. ISBN: 0-7695-0001-3. URL: `http://dl.acm.org/citation.cfm?id=874074.876295`.

[CH10] Henrik Bærbak Christensen ir Klaus Marius Hansen. An empirical investigation of architectural prototyping. *Journal of systems and software*, 83(1):133–142, 2010. ISSN: 0164-1212.

[FB] Viktoria Firus ir Steffen Becker. Abstract fesca 2005 preliminary version parametric performance contracts for qml-specified software components.

[Fie00] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. AAI9980887. Disertacija. 2000. ISBN: 0-599-87118-0.

[FRF+03] Martin Fowler, Dave Rice, Matthew Foemmel, Edward Hieatt, Robert Mee ir Randy Stafford. *Patterns of enterprise application architecture*. Addison-Wesley Professional, 2003.

[FS02] Kimiyuki Fukuzawa ir Motoshi Saeki. Evaluating software architectures by coloured petri nets. *Proceedings of the 14th international conference on software engineering and knowledge engineering*. SEKE '02. ACM, Ischia, Italy, 2002, p. 263–270. ISBN: 1-58113-556-4.

[Gar14] David Garlan. Software architecture: a travelogue. *Proceedings of the on future of software engineering*. FOSE 2014. ACM, Hyderabad, India, 2014, p. 29–39. ISBN: 978-1-4503-2865-4.

[GGM+08] Stefano Gallotti, Carlo Ghezzi, Raffaela Mirandola ir Giordano Tamburrelli. Quality prediction of service compositions through probabilistic model checking. *Qosa*. Steffen Becker, Frantisek Plasil ir Ralf Reussner, redaktoriai. Tom. 5281. Lecture Notes in Computer Science. Springer, 2008-10-22, p. 119–134. ISBN: 978-3-540-87878-0.

[GS93] David Garlan ir Mary Shaw. An introduction to software architecture. *Advances in software engineeringand knowledge engineering*, I:5–15, 1993.

[HBR14] Lucia Happe, Barbora Buhnova ir Ralf Reussner. Stateful component-based performance models. English. *Software and systems modeling*, 13(4):1319–1343, 2014. ISSN: 1619-1366.

[HCW+12] Glenn I. Hawe, Graham Coates, Duncan T. Wilson ir Roger S. Crouch. Agent-based simulation for large-scale emergency response: a survey of usage and implementation. *Acm comput. surv.*, 45(1):8:1–8:51, 2012-12. ISSN: 0360-0300.

[Het10] Sjors Hettinga. Performance analysis for embedded software design. Disertacija. University of Twente, 2010.

[Koz10] Heiko Koziolek. Performance evaluation of component-based software systems: a survey. *Perform. eval.*, 67(8):634–658, 2010-08. ISSN: 0166-5316.

[Kru95] Philippe B. Kruchten. The 4+1 view model of architecture. *Ieee software*, 12:42–50, 1995.

[Lar04] Magnus Larsson. Predicting quality attributes in component-based software systems. Disertacija. Mälardalen University, 2004-03.

[Mik14] Algirdas Mikoliūnas. Research of software architectures using agent-based modelling. Magistrinis darbas. Vilniaus Universitetas, 2014.

[MN05] Charles M. Macal ir Michael J. North. Tutorial on agent-based modeling and simulation. *Proceedings of the 37th conference on winter simulation*. WSC '05. Winter Simulation Conference, Orlando, Florida, 2005, p. 2–15. ISBN: 0-7803-9519-0.

[NHS08] Lewis Ntaimo, Xiaolin Hu ir Yi Sun. Devs-fire: towards an integrated simulation environment for surface wildfire spread and containment. *Simulation*, 84(4):137–155, 2008-04. ISSN: 0037-5497.

[Nia11] Muaz Ahmed Khan Niazi. Towards a novel unified framework for developing formal, network and validated agent-based simulation models of complex adaptive systems. Disertacija. School of Natural Sciences University of Stirling Scotland UK, 2011.

[RW05] Nick Rozanski ir Eóin Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley Professional, 2005. ISBN: 0321112296.

[Sar11] Robert G. Sargent. Verification and validation of simulation models. *Proceedings of the winter simulation conference*. WSC '11. Winter Simulation Conference, Phoenix, Arizona, 2011, p. 183–198. URL: http://dl.acm.org/citation.cfm?id=2431518.2431538.

[Sha90] Mary Shaw. Toward higher-level abstractions for software systems. *Data knowl. eng.*, 5:119–128, 1990.

[SM08] Paulo Salem da Silva ir Ana C. V. de Melo. Reusing models in multi-agent simulation with software components. *Aamas '08: proceedings of the 7th international joint conference on autonomous agents and multiagent systems - volume 2*. International Foundation for Autonomous Agents ir Multiagent Systems, Estoril, Portugal, 2008. ISBN: 978-0-9817381-1-6.

[SM09] Rajiv Ranjan Suman ir Rajib Mall. State model extraction of a software component by observing its behavior. *Sigsoft softw. eng. notes*, 34(1):1–7, 2009-01. ISSN: 0163-5948.

[SMG+10] Peer-Olaf Siebers, Charles M. Macal, Jeremy Garnett, D. Buxton ir Michael Pidd. Discrete-event simulation is dead, long live agent-based simulation! *J. simulation*, 4(3):204–210, 2010.

[Sol12] Fritz Solms. What is software architecture? *Proceedings of the south african institute for computer scientists and information technologists conference*. SAICSIT '12. ACM, Pretoria, South Africa, 2012, p. 363–373. ISBN: 978-1-4503-1308-7.

[ST07] Vibhu Saujanya Sharma ir Kishor S. Trivedi. Quantifying software performance, reliability and security. *J. syst. softw.*, 80(4):493–509, 2007-04. ISSN: 0164-1212.

[STV11] Lalit Kumar Singh, Anil Kumar Tripathi ir Gopika Vinod. Software reliability early prediction in architectural design phase: overview and limitations. *JSEA*, 4(3):181–186, 2011.

[Szy02] Clemens Szyperski. *Component software: beyond object-oriented programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd leid., 2002. ISBN: 0201745720.

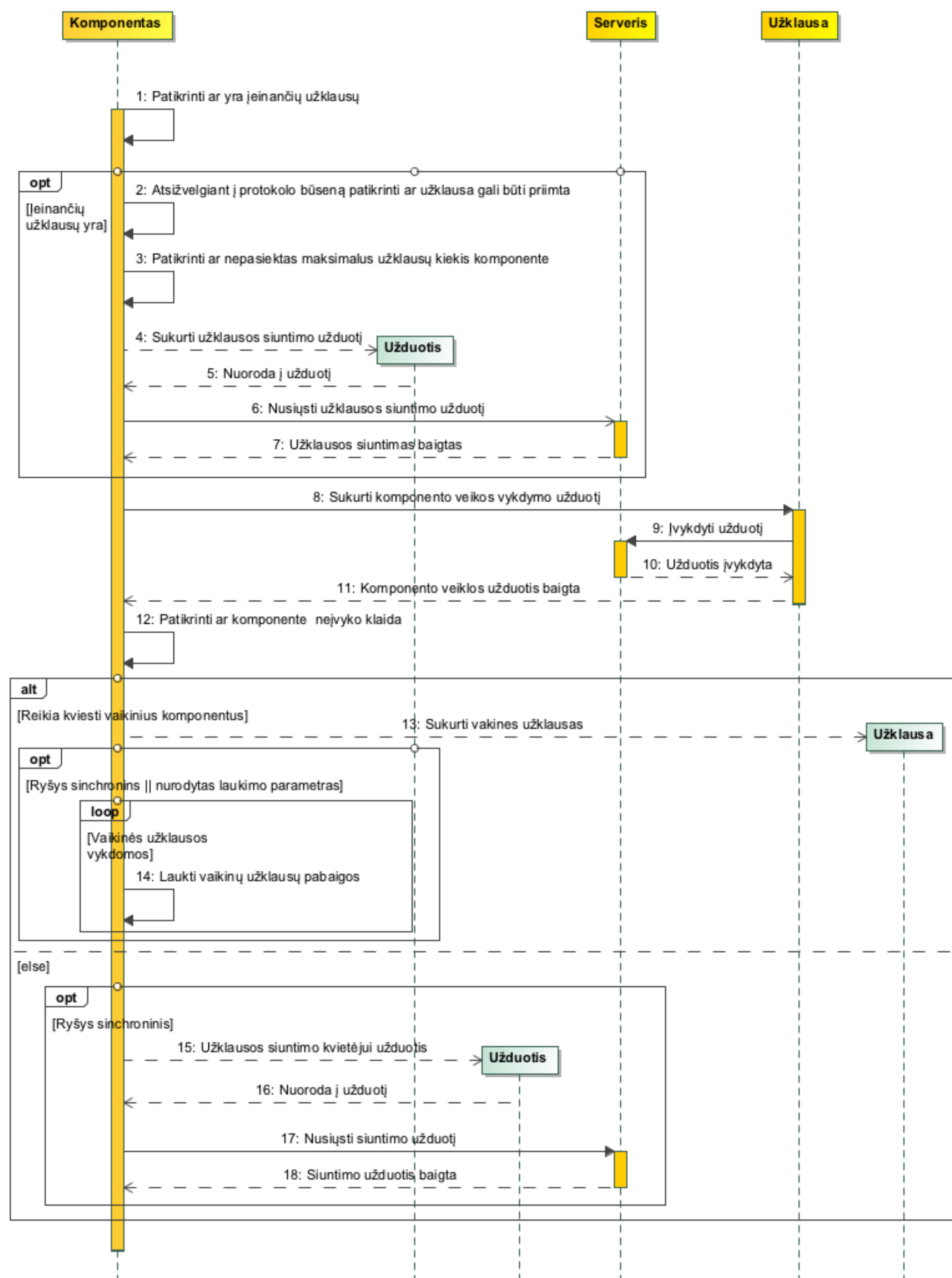
[WPC06] Wen-Li Wang, Dai Pan ir Mei-Hwa Chen. Architecture-based software reliability modeling. *J. syst. softw.*, 79(1):132–146, 2006-01. ISSN: 0164-1212.

[ZG05] Shifeng Zhang ir Steve Goddard. Xsadl: an architecture description language to specify component-based systems. *Proceedings of the international conference on information technology: coding and computing (itcc'05) - volume ii - volume 02*. ITCC '05. IEEE Computer Society, Washington, DC, USA, 2005, p. 443–448. ISBN: 0-7695-2315-3. DOI: 10.1109/ITCC.2005.303.

[Zsc10] Steffen Zschaler. Formal specification of non-functional properties of component-based software systems. *Software and systems modeling*, IX:161–201, 2010.

Priedas nr. 1

Komponento vieno simuliacijos žingsnio sekų diagrama



25 pav. Vieno simuliacijos žingsnio komponento veiklos sekų diagrama