KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Lina Kelmaitė

MagicDraw įrankio praplėtimas sekų ir būsenų diagramų suderinimo galimybe

Magistro darbas

Darbo vadovas
prof. dr. Lina Nemuraitė

KAUNAS, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Lina Kelmaitė

MagicDraw įrankio praplėtimas sekų ir būsenų diagramų suderinimo galimybe

Magistro darbas

Recenzentas
doc. dr. Antanas Lenkevičius
2007-05-28

Darbo vadovas
prof. dr. Lina Nemuraitė
2007-05-28

Atliko
IFM-1/2 gr. stud.
Lina Kelmaitė

KAUNAS, 2007

TURINYS

1. ĮVADAS

Sparčiai tobulėjant informacinėms technologijoms ir didėjant kuriamų sistemų sudėtingumui bei skaičiui, atsirado poreikis kuo daugiau automatizuoti IS kūrimo procesą. Automatizavimo problemai spręsti naudojami modeliais grindžiamo kūrimo (angl. *Model Driven Development*) metodai, kurių esmė – didesnis naudojamų abstrakcijų lygis, kai vietoj programavimo kalbų naudojamos modeliavimo kalbos (UML).

Modeliais grindžiamo kūrimo metoduose pagrindinis dėmesys sutelktas į paskutinius kūrimo proceso etapus. Organizacijos Object Management Group (OMG) sukurta Model Driven Architecture (MDA) sistemų kūrimo technologija koncentruojasi į nuo platformos nepriklausomų modelių PIM (angl. *Platform Independent Model*) transformavimą į konkrečių platformų modelius PSM (angl. *Platform Specific Model*) ir pastarųjų transformavimą į programos kodą. Pradiniai kūrimo proceso etapai – reikalavimai ir jų analizė lieka nepaliesti, projektuotojas lieka atsakingas už PIM sudarymą, korektiškumą bei suderinimą.

Universali modeliavimo kalba UML – tai standartas, skirtas sistemos specifikavimui, vizualizavimui ir dokumentavimui. Modeliai, specifikuoti UML, gali būti vaizduojami įvairiomis diagramomis (klasių, sekų, būsenų, veiklos ir kt.), kurios tą pačią sistemą vaizduoja skirtingais aspektais (statinis, dinaminis, sąveikų). Projektuotojas, kurdamas skirtingas diagramas, gali jų nesuderinti, palikti kokius nors neatitikimus ar prieštaravimus.

Dabartiniuose CASE įrankiuose yra galimybės modelį ne tik pateikti vizualiai, piešiant skirtingas sistemos diagramas, bet ir susieti jas tarpusavyje per tam tikrus elementus. Įrankiai leidžia kurti modelį, kuris atitinka UML specifikaciją, o taip pat susieti diagramas tarpusavyje, t.y. naudoti vienose diagramose sukurtus elementus kuriant kitas diagramas. Tokiu būdu gali būti gaunamas vientisas modelis, tačiau jo suderinamumo užtikrinimas lieka projektuotojo rankose.

Reikalavimus galima apibrėžti naudojant panaudos atvejų (angl. *Use Case*) diagramą. Kiekvienam panaudos atvejui galima sudaryti vieną ar keletą alternatyvių įvykių scenarijų. Įvykių scenarijai UML vaizduojami sekų diagramomis (angl. *Sequence Diagrams*) – parodo kokiais pranešimais keičiasi sąveikaujantys objektai. Vieno objekto pilna elgsena gali būti atvaizduojama būsenų mašina (angl. *State Machine*). Šių abiejų tipų diagramos aprašo sistemą skirtingais aspektais (pirmoji – sąveikų vaizdas, antroji – dinaminis vaizdas). Kad būtų užtikrintas modelio vientisumas, šios diagramos turėtų derėti tarpusavyje, todėl būtų naudinga transformuoti sekų diagramas į būsenų ir atvirkščiai. Tokios transformacijos gali būti automatizuotas, kas palengvintų projektuotojo darbą.

Šio darbo tyrimo sritis – sekų ir būsenų diagramų suderinimas, o tyrimo objektas – elektroninių paslaugų sistemų modeliai. Tai nereiškia, kad modelis yra specifiškai apribotas, tačiau išanalizuota eksperimentinių taikymų aibė yra orientuota paslaugų architektūra paremtoms sistemoms.

Darbo tikslas – pagerinti UML kalbą naudojančius projektavimo procesus, papildant juos sekų ir būsenų diagramų derinimo galimybėmis, praplėsti UML specifikaciją, kad būtų įmanomas abipusis sekų diagramų ir būsenų mašinų transformavimas.

Darbo uždaviniai:

- ištirti sekų diagramų transformavimo į būsenų mašinų algoritmus;
- išanalizuoti abipusio sekų ir būsenų diagramų transformavimo algoritmą;
- eksperimentiškai parodyti transformavimo rezultatus.

Nagrinėti šaltiniai susiję su UML standartu [16] ir keletu pateiktų sekų diagramų transformavimo į būsenų diagramas algoritmų [1], [10], [17], [20]. Abipusio metodo realizacijos galimybių analizė paremta [9] šaltiniu. Darbo tikslai ir uždavinių formuluotė susiję su Informacijos sistemų katedroje sukurtu sekų ir būsenų diagramų suderinimo metodu, kuris išdėstytas [3], [4], [5] šaltiniuose.

Antrame darbo skyriuje pateikti keli literatūroje pasiūlyti sekų diagramų transformavimo į būsenų mašinas algoritmai ir transformacijų pavyzdžiai. Skyriaus gale pateikta šių algoritmų palyginimo lentelė pagal tam tikras savybes bei keleto CASE įrankių, kuriuose galėtų būti įgyvendinti algoritmai, apžvalga. Trečiame skyriuje pateikta abipusės sekų ir būsenų diagramų transformacijos metodika, kuri remiasi dr. L. Čeponienės algoritmu [3], [4], [5]. Šiame skyriuje taip pat pasiūlytas sudėtingesnių diagramų transformavimo galimybė. Ketvirtame skyriuje pateiktas pagal trečio skyriaus metodiką atliktos sekų ir būsenų diagramų transformacijų realizacijos projektas praplečiant MagicDraw UML įrankį. Penktame skyriuje pateiktas sukurto įskiepio efektyvumo tyrimas bei transformacijų pavyzdžiai. Šeštame skyriuje pateiktos bendros darbo išvados.

2. EGZISTUOJANČIŲ SEKŲ DIAGRAMŲ TRANSFORMAVIMO Į BŪSENŲ DIAGRAMAS ALGORITMŲ ANALIZĖ

Šiame skyriuje pateikti keletas literatūroje pasiūlytų metodų, kaip atvaizduoti sekų diagramas į būsenų diagramas. Paskutiniame poskyriuje pateiktas metodų palyginimas pagal kai kurias savybes.

2.1. Makinen ir Systa metodas

Makinen ir Systa pasiūlė interaktyvų būdą [10], kaip atvaizduoti sekų diagramas į būsenų diagramas. Šiame būde interaktyvus algoritmas MAS (Minimally Adequate Synthesizer) konsultuojasi su vartotoju apie galimus pakeitimus.

Sintetiniai algoritmai apibendrina sekų diagramų informaciją, todėl gauta būsenų mašina priima daugiau modeliuojamos sistemos kelių, negu jų galime rasti sekų diagramose, ko dažniausiai ir reikia. Tačiau kai kuriais atvejais sintetinta būsenų mašina priima nepageidaujamus arba klaidingus kelius, tokiu būdu gaunamas pernelyg didelis apibendrinimas. Kad būtų išvengta tokio apibendrinimo, projektuotojas turi valdyti algoritmą sintezės metu. Paprastai projektuotojui užduodami tokie klausimai: (1) jis turi nuspręsti, ar pasiūlyta elgsena tinkama kuriamai sistemai, (2) jis turi priimti arba atmesti sudarytą būsenų diagramą, jeigu atmeta turi būti pateikiamas priešingas pavyzdys. Projektuotojas gali nurodyti teisingus ir neteisingus pavyzdžius. Teisingi pavyzdžiai apibūdina papildomą informaciją, kuri turėtų būti įtraukta į būsenų diagramą, tuo tarpo neteisingi pavyzdžiai apibūdina elgseną, kurią reiktų pašalinti.

Tarkim, kad D yra sekų diagrama, kurioje vaizduojama objekto I sąveika su kitais objektais. Imkime objekto I gyvavimo liniją, pradedant nuo viršaus kiekvieną sekančią pranešimų porą e_i ir e_j, susijusią su I, kur e_i – išsiųstas pranešimas, e_j – gautas pranešimas, dėkime į porų seką. Jei vieno iš pranešimų trūksta, įrašykime vietoj jo *Null*.

Sekos pora (e_i, e_j) reiškia, kad kažkuriuo sistemos vykdymo metu objektas I siunčia pranešimą e_i kuriam nors kitam objektui ir tada reaguoja į pranešimą e_j, kurį atsiuntė kitas objektas. Taigi susiejame kiekviena siųstą pranešimą su veiksmu (elgsena), atliekamu tam tikroje būsenoje, ir kiekvieną gautą pranešimą su įvykiu, kuris sukelia būsenos pasikeitimą (iššaukia naują veiksmą). Pavyzdžiui, porai (e_i, e_j) egzistuoja būsena su veiksmu „*do*: e_i“ ir iš šios būsenos išeinantis perėjimas e_j būsenų mašinoje. Žemiau pateikiamas transformavimo pavyzdys.

Tarkim vartotojas projektuoja žadintuvo elgseną. Pavyzdinė sekų diagrama parodyta 1 paveikslėlyje. Sekų diagramos objektai – vartotojas, kuris gali nustatyti žadintuvą, valdiklis –

mechanizmas, valdantis žadintuvą, žadintuvas – žadintuvo skambėjimo mechanizmas, laikrodis – laiko rodymo mechanizmas. Porų seka valdikliui pateikta žemiau:

(rodyti laiką, nustatyti žadinimo laiką),

(užregistruoti, NULL),

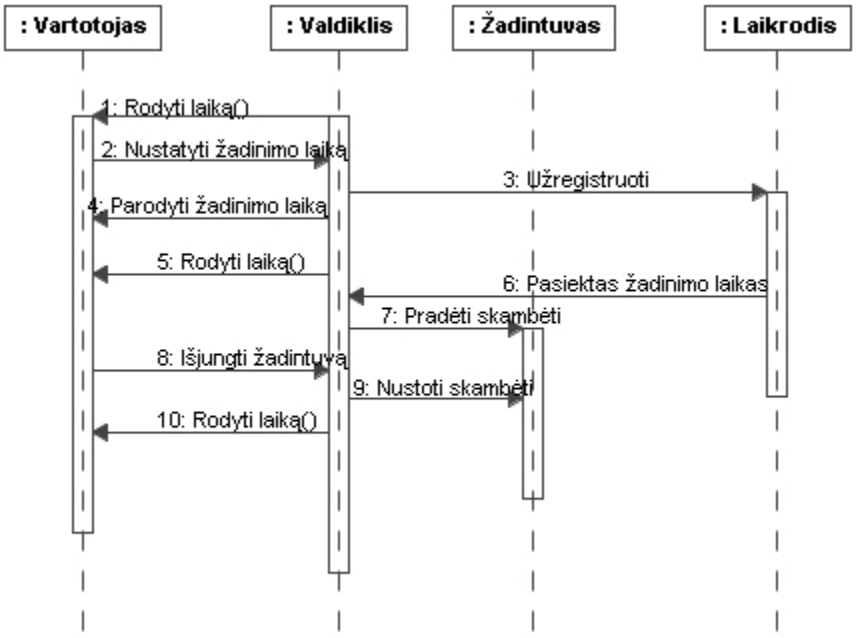
(parodyti žadinimo laiką, NULL),

(rodyti laiką, žadinimo laikas pasiektas),

(pradėti skambėti, išjungti žadintuvą),

(nustoti skambėti, NULL),

(rodyti laiką, VOID).

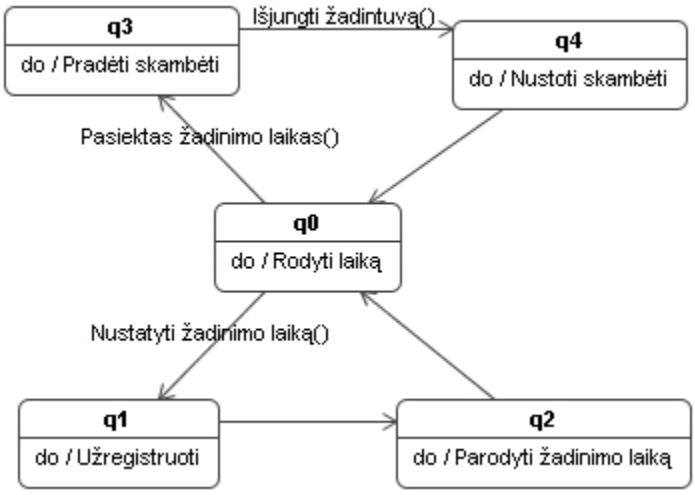


1 pav. Žadintuvo elgseną apibūdinanti sekų diagrama

Kad būtų sudaryta numatyta būsenų diagrama, algoritmas turi konsultuotis su vartotoju keturis kartus. Pateikiamos elgsenos užklausos tokiems keliams:

1) Vartotojas nenurodė jokių įeinamų reikšmių, t.y. žadintuvas tik rodo laiką,
2) nustatytas naujas žadinimo laikas.
3) nustatytas naujas žadinimo laikas ir jis pakeistas prieš tai, kai laikrodis pradeda skambėti,
4) nustatytas naujas žadinimo laikas, žadintuvas pradeda skambėti kai žadinimo laikas pasiektas, nustatytas kitas žadinimo laikas.

Vartotojas priima visus šiuos kelius, algoritmas sudaro būsenų mašiną objektui žadintuvas, kuri pavaizduota 2 paveikslėlyje.



2 pav. Gauta žadintuvo būsenų diagrama

Pradinėje būsenoje q0 žadintuvas rodo laiką. Įvykis „Nustatyti žadinimo laiką“ sąlygoja perėjimą į būseną q1, kurioje užregistruojamas nustatytas laikas. Iš karto po laiko užregistravimo pereinama į q2 būseną, kurioje parodomas žadinimo laikas, kurį nustatė vartotojas ir užregistravimo žadintuvas. Parodžius žadinimo laiką, grįžtama į būseną q0, kurioje rodomas esamas laikas. Kai pasiekiamas žadinimo laikas, iš būsenos q0, pereinama į būseną q3, kurioje žadintuvas pradeda skambėti. Įvykis „Išjungti žadintuvą“ sąlygoja perėjimą iš būsenos q3 į būseną q4, kurioje žadintuvas nustoja skambėti ir pereina į laiko rodymo būseną q0.

Algoritmas yra pakankamai sudėtingas, reikalauja nuolatinio vartotojo įsikišimo. Šis algoritmas įgyvendintas ir integruotas UML modeliavimo įrankyje Nokia TED.

2.2. Whittle J. ir Schumann J. algoritmas

Whittle ir Schumann pateikė algoritmą [20] automatiniam UML būsenų mašinų generavimui iš sekų diagramų. Algoritmą sudaro trys svarbūs žingsniai: pirma, reikia surasti ir išspręsti konfliktines situacijas, susidariusias apjungiant atskiras sekų diagramas; antra, skirtingos sekų diagramos dažnai susideda iš identiškų arba panašių elgsenų, šias elgsenas reikia atpažinti ir apjungti; trečia, sugeneruotos būsenų mašinos paprastai būna tik sistemos aproksimacija, todėl jas rankiniu būdu turi modifikuoti ir tobulinti sistemos projektuotojas. Tam pasiekti gautos būsenų mašinos turi būti struktūriškai apibrėžtos ir suprantamos, t.y. naudojama būsenų hierarchija.

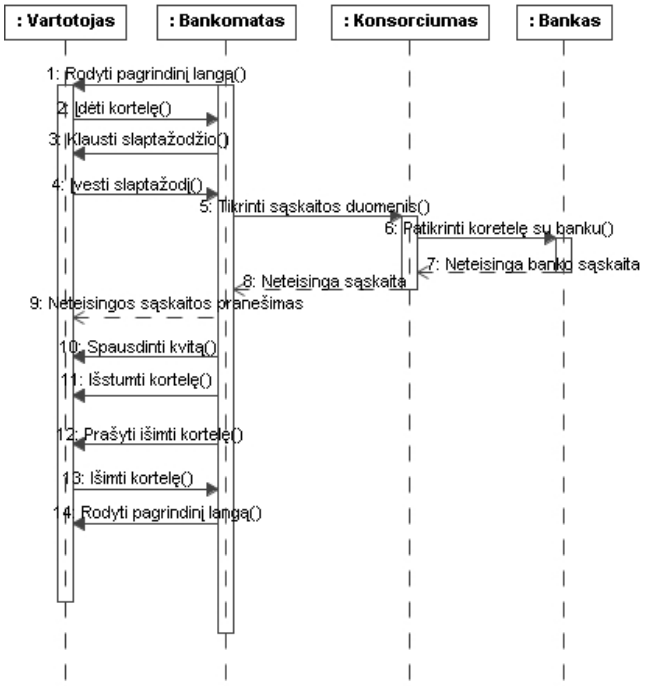
Paprastai sekų diagramų neužtenka tam, kad būtų automatiškai aptiktos konfliktinės situacijos. Trūkstamai informacijai gauti naudojami vartotojo nurodyti apribojimai diagramose. Tokie apribojimai aprašomi objektų apdorojimo kalba OCL [15] (angl. *Object Constraint Language*).

Kad sekų diagramos nebūtų dviprasmiškos ir sunkiai interpretuojamos, kiekvienam pranešimui (sekų diagramoje) sudaromos prieš ir po sąlygos, išreikštos OCL kalba. OCL apribojimai gali būti naudojami ne tik konfliktinių situacijų suradimui, bet ir identiškų būsenų atpažinimui skirtingose sekų diagramose bei šių diagramų apjungimui.

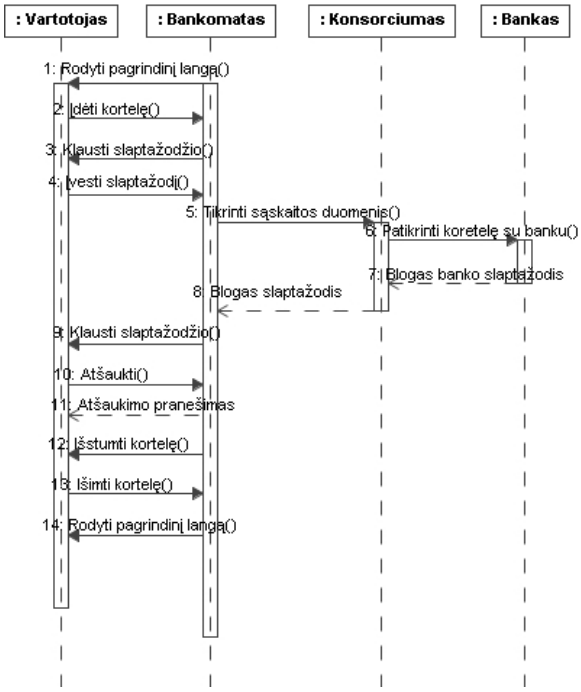
SD gali būti apribota dviem būdais: būsenų vektoriaus apribojimai, aprašyti OCL kalba ir pranešimų tvarkos apribojimai, aprašyti pačios SD. Konfliktinės situacijos reiškia, kad sekų diagrama neatitinka vartotojo numatytos semantikos, arba srities teorija yra neteisinga. Vartotojas turi nuspręsti, kokios modifikacijos turi būti atliktos.

Kai konfliktinės situacijos pašalintos galima generuoti būsenų diagramą. Kiekvienai SD galima sugeneruoti keletą baigtinių būsenų mašinų (FSMs – *finite states machines*), po vieną kiekvienam SD objektui. Kiekviena FSM apibūdina klasės, kuriai atitinkamas objektas priklauso, elgseną. Pranešimai nukreipti į tam tikrą objektą O yra laikomi įvykiais objekto O būsenų mašinoje. Pranešimai nukreipti nuo objekto laikomos veiksmais. Ciklas aptinkamas, jeigu būsenų vektorius, einantis iš karto po to, kai esamas pranešimas įvykdytas, yra toks pats kaip esamos būsenos vektorius ir jeigu tai yra būsenos keitimo pranešimas. Būsenų identifikavimas remiantis tik būsenų vektoriais lems neteisingą ciklo aptikimą. Kad to išvengtumėme, ciklai sukuriami tik tada, kai pranešimas keičia būseną.

Žemiau pateiktos 4 sekų diagramos, vaizduojančios bankomato elgseną (paveikslėliai paimti iš [20] ir išversti į lietuvių kalbą).

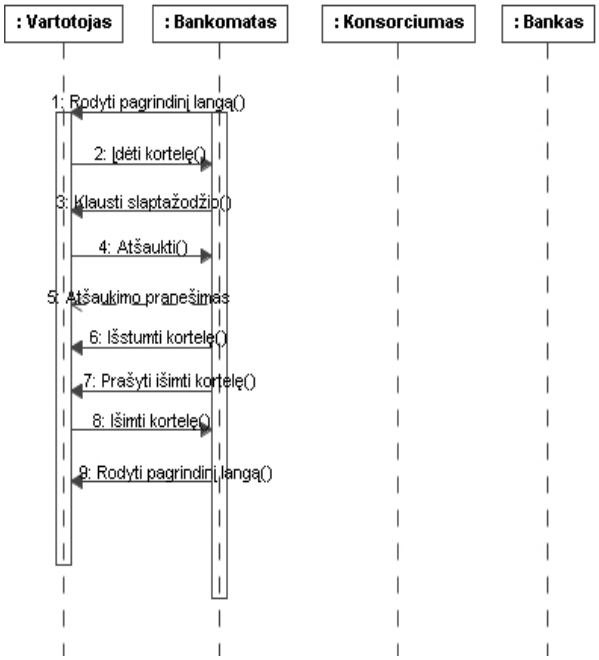


3 pav. Bankomato scenarijus SD 1

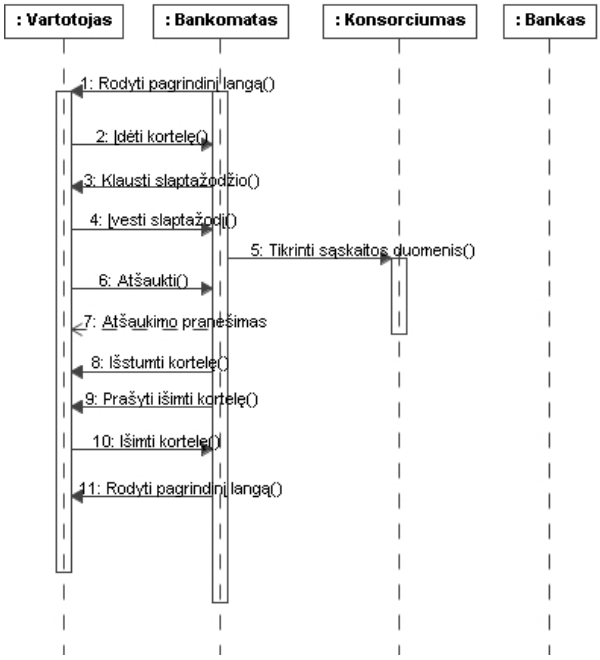


4 pav. Bankomato scenarijus SD 2

Pirmoje sekų diagramoje, pateikiamas variantas, kai banko kortelė neatpažįstama. Sistema išveda pranešimą apie neteisingą sąskaitą ir išstumia iš bankomato kortelę. Tuo tarpu antroje SD įvedamas neteisingas slaptažodis, sistema prašo pakartoti slaptažodį, vartotojas operaciją atšaukia.



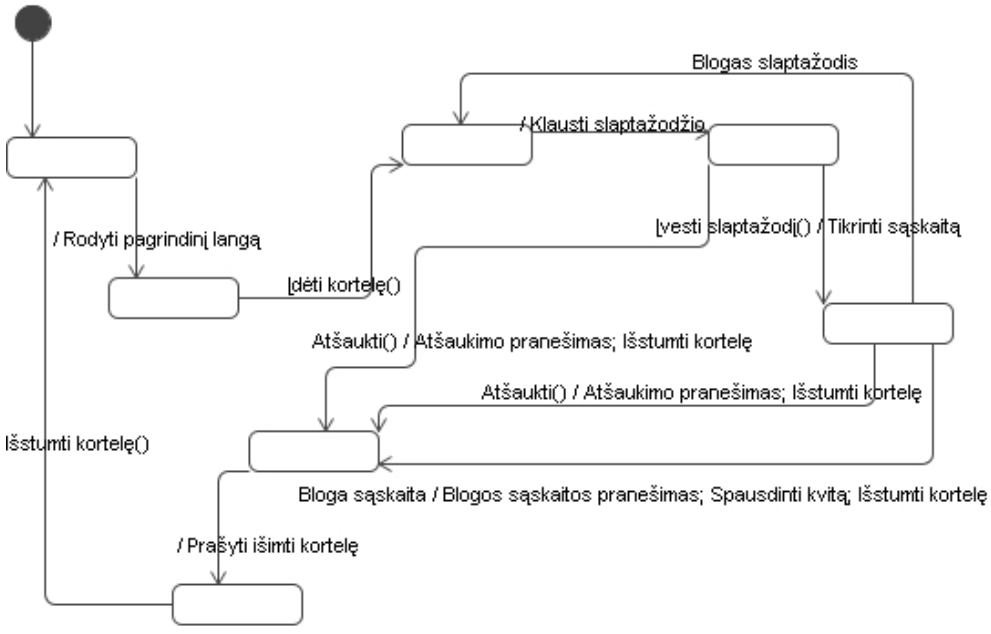
5 pav. Bankomato scenarijus SD 3



6 pav. Bankomato scenarijus SD 4

Trečioje diagramoje atšaukimo veiksmas įvykdomas, kai pirmą kartą užklausiama slaptažodžio. Ketvirtoje SD slaptažodis įvedamas, tačiau atšaukimas įvykdomas dar nespėjus bankomatui patikrinti kortelės.

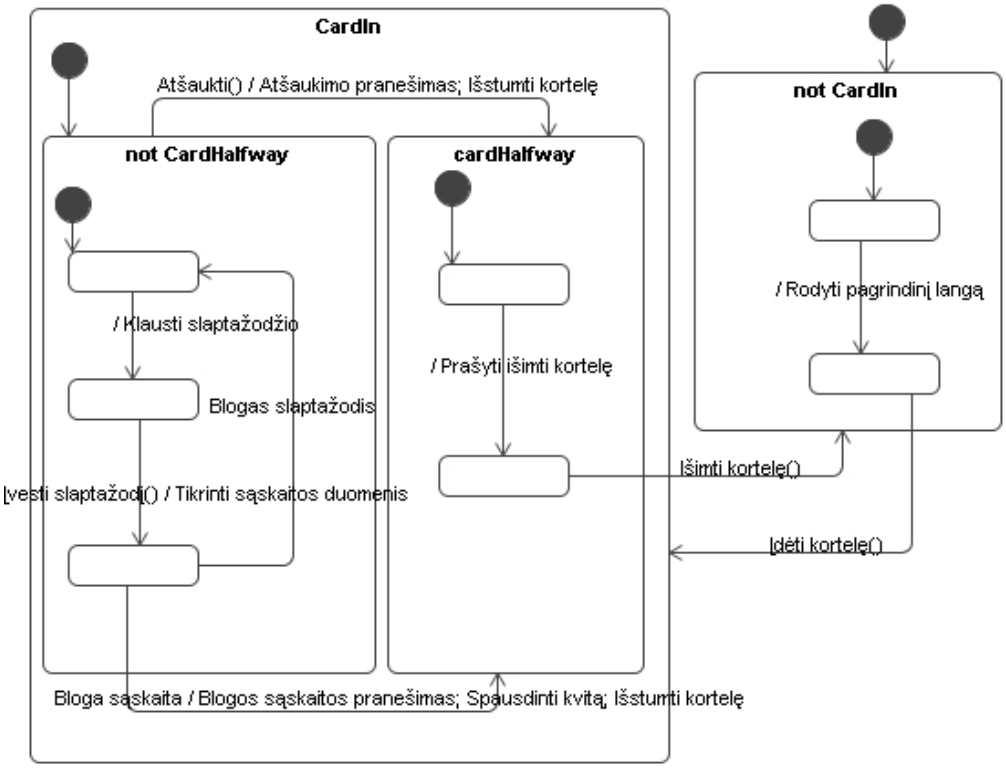
Septintame paveikslėlyje pateikiama pagal algoritmą sugeneruota būsenų diagrama. Ji gauta apjungus visas keturias sekų diagramas.



7 pav. Bankomato objekto būsenų diagrama

Kiekvienas perėjimas iš vienos būsenos į kitą yra susiejamas su gautu įvykiu ir/arba atliekamu veiksmu.

Galima sudaryti hierarchinę būsenų mašiną, jeigu paimsime kelias pagrindines bankomato būsenas: kai kortelė neįdėta (būsena „*not CardIn*“), kai kortelė įdėta (būsena „*CardIn*“), kortelė pusiaukelėje (būsena „*cardHalfway*“), kortelė visiškai įdėta (būsena „*not CardHalfway*“). Aštuntame paveikslėlyje pavaizduota gauta hierarchinė būsenų mašina.



8 pav. Hierarchinė bankomato būsenų mašina

Algoritmas yra pakankamai sudėtingas, būsenos nėra įvardinamos.

2.3. Bordeleau ir Corriveau metodas

Bordeleau ir Corriveau pateikė šablonais paremtą metodą [1] hierarchinių būsenų mašinų konstravimui iš sekų diagramų. Jie teigė, kad standartinės (nehierarchinės) būsenų mašinos tinka išsamaus pranešimų kelio analizei, tuo tarpu hierarchinės būsenų mašinos leidžia komponento skirtingų elgsenos aspektų (dažnai vadinamų vaidmenimis) grupavimą į atskiras hierarchines šio komponento būsenas.

Bordeleau ir Corriveau būdas atskiria skirtingus dalykus elgsenoje. Šiame metode nuo panaudojimo atvejų, apibrėžtų reikalavimų lygyje, pereinama prie panaudojimo atvejų plano UCM (angl. *Use Case Map*) modelio, kuris naudojamas kaip sistemos aukšto lygio projektavimo modelis. Tada UCM yra tobulinamas į UML sekų diagramas. UCM modelyje aprašomos sąveikos tarp sekų diagramų. Galiausiai detalizuotos diagramos transformuojamos į hierarchines būsenų mašinas.

Hierarchinių būsenų mašinų generavimas iš sekų diagramų vykdomas tokiais dviem etapais: (1) detali sekų diagramose esanti scenarijų informacija naudojama skirtingų komponento elgsenos aspektų būsenų mašinų sudarymui (kiekvienam scenarijui), (2) tarpscenarinių ryšių informacija (surinkta UCM modelyje) naudojama komponento elgsenos būsenų mašinų integravimui į to komponento hierarchinę būsenų mašiną.

Bordeleau ir Corriveau siūlo būsenų mašinų integravimui naudoti šablonus.

2.4. Vasilache ir Tanaka metodas

Vasilache ir Tanaka [17] pasiūlė naudoti priklausomybės diagramas (angl. *Dependency diagrams*), kuriose nurodomi ryšiai tarp atskirų sekų diagramų. Vienam panaudos atvejui galimi keli alternatyvūs scenarijai (kelios sekų diagramos). Galimi ryšiai tarp scenarijų: eiliškumas (vienas scenarijus eina po kito), disjunkcija (tuo pačiu metu gali vykti tik vienas iš kelių scenarijų), konjunkcija (scenarijai vykdomi vienu metu), kartojimas (scenarijus kartojamas keletą kartų).

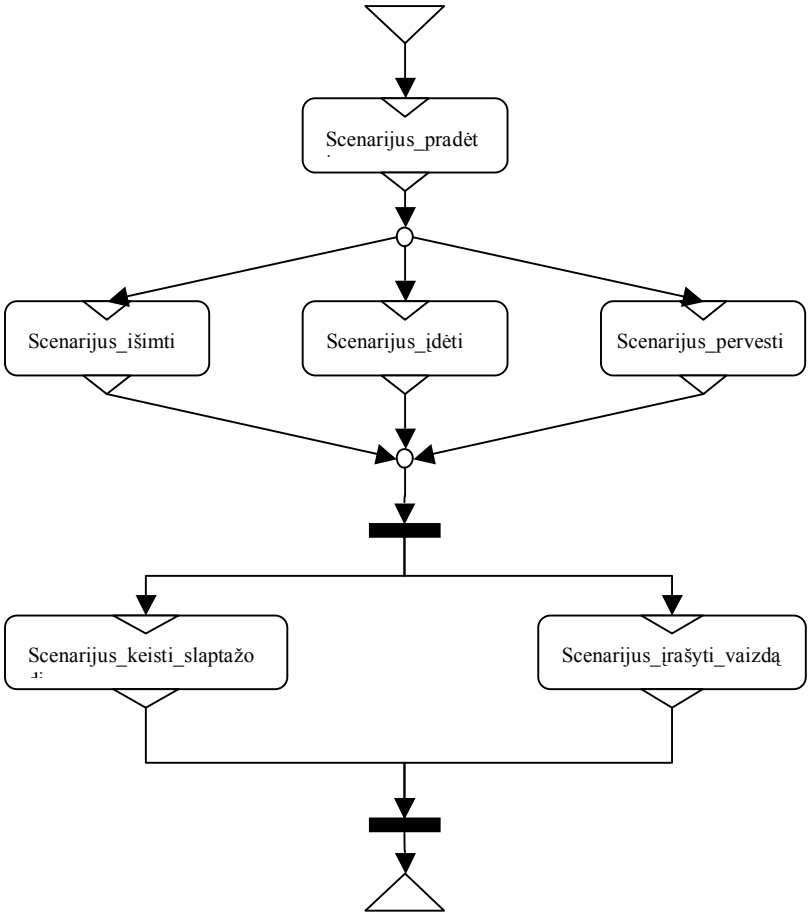
Paprastas priklausomybės diagramos pavyzdys pateiktas 9 paveikslėlyje. Jis paremtas bankomato elgsenos pavyzdžiu. Sekų diagrama „Scenarijus_pradėti“ apima bendrus veiksmus, kuriuose pradžioje turi atlikti vartotojas. Toliau pateikiami trys alternatyvūs scenarijai: „Scenarijus_išimti“, „Scenarijus_įdėti“, „Scenarijus_pervesti“. Vienu metu gali vykti tik vienas iš šių scenarijų, jie susiję disjunkcijos ryšiu. Toliau scenarijai „Scenarijus_keisti_slaptažodį“ ir „Scenarijus_įrašyti_vaizdą“ yra tarpusavyje susiję konjunkcijos ryšiu, t.y. abu vyksta vienu metu.

Naudojant šias priklausomybės diagramas užtikrinamas sistemos reikalavimų pilnumas ir žinant priklausomybes tarp scenarijų galima sukurti būsenų mašinas, kurios parodo pilną objektų elgseną.

Kad būtų galima sudaryti būsenų mašinas visiems objektas, siūloma atlikti tokius veiksmus:

- nustatyti visus atskirus scenarijus (atvaizduotus sekų diagramomis);
- nustatyti ryšius tarp visų scenarijų (atvaizduotus priklausomybės diagramomis);
- sudaryti būsenų mašinas remiantis informacija gauta atlikus pirmus du etapus.

Būsenų mašinų kūrimas susideda iš dviejų etapų: (1) sukuriama viena pradinė būsenų diagrama kiekvienam objekto scenarijui, (2) apjungiamos visos pradinės būsenų diagramos į galutinę objekto būsenų mašiną, priklausomai nuo informacijos, pateiktos priklausomybės diagramoje.



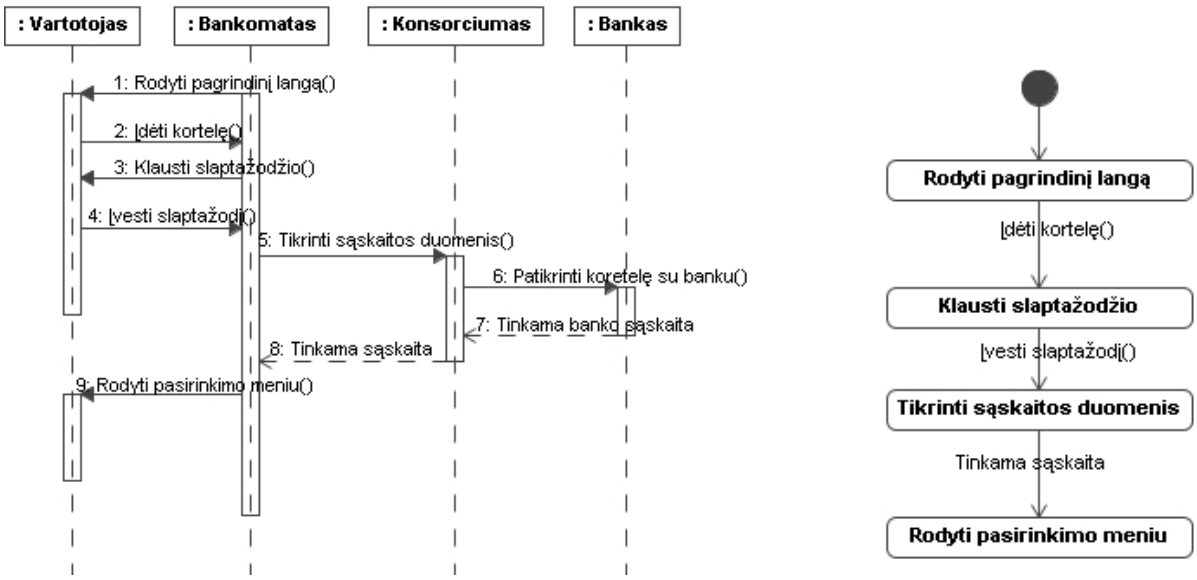
9 pav. Priklausomybės diagrama

Būsenų mašina vienam scenarijui kuriama pagal tokią taisykles:

- į objektą nukreipti pranešimai rodo objekto gautus įvykius – jie susiejami su perėjimais būsenų diagramoje.
- Išeinantys pranešimai yra veiksmai, kurie tampa perėjimų veiksmais vedančiais į būsenas.
- Intervalai tarp įvykių yra būsenos. Prieš gaudamas kokį nors įvykį, objektas yra numatytoje (angl. *default*) būsenoje.

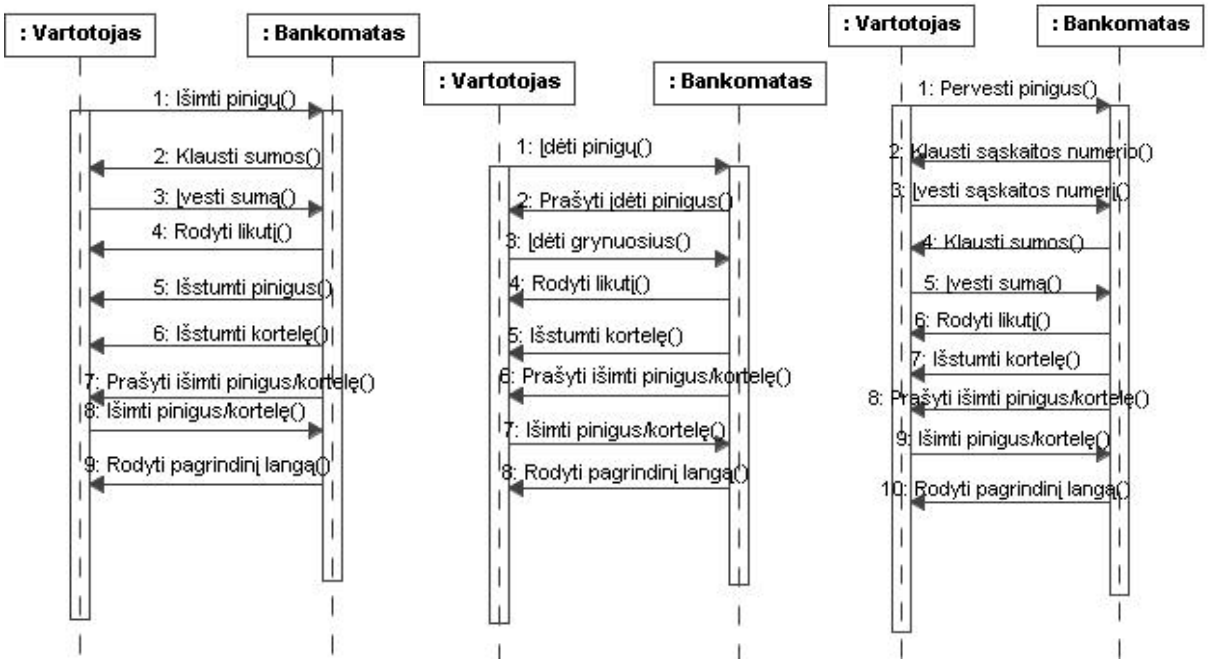
Pradinių būsenų mašinų sukūrimo žingsniai:

1. sukurti tuščias būsenų mašinų diagramas kiekvienam scenarijui, kuriame dalyvauja objektas;
2. kiekvienai būsenų diagramai sukurti visus įvykius (atitinkančius perėjimus nukreiptus į objektą);
3. kiekvienam perėjimui iš objekto sukurti veiksmus, kurie veda į būsenas, ir sukurti atitinkamas būsenas;
4. nustatyti teisingą būsenų perėjimų eiliškumą.

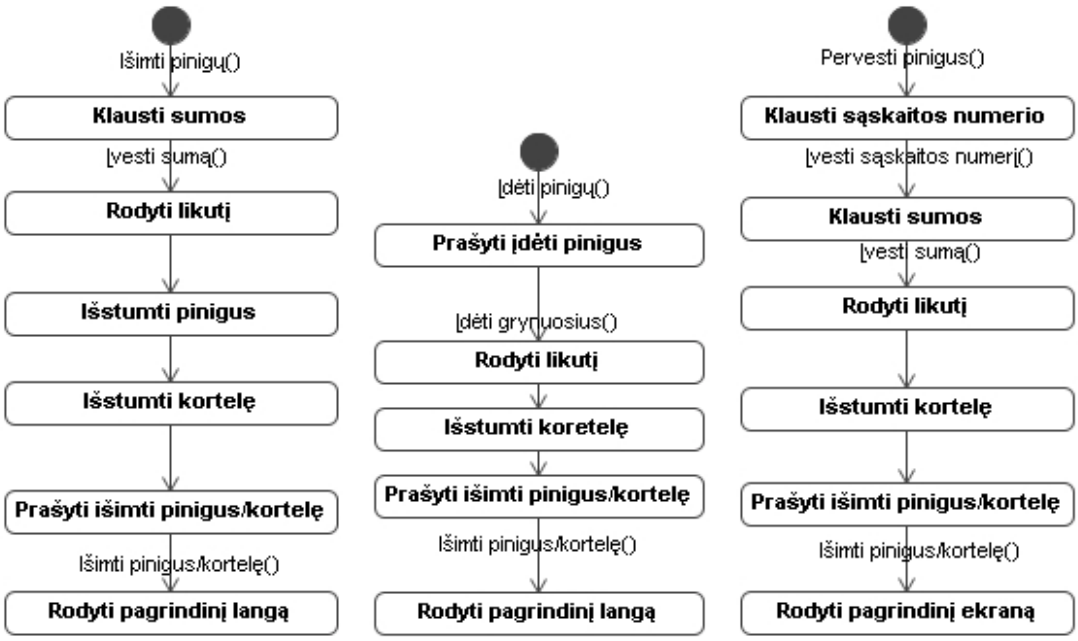


10 pav. Bankomato SD Scenarijus_pradėti ir iš jos gauta būsenų diagrama

10 paveikslėlyje pavaizduota sekų diagrama „Scenarijus_pradėti“ ir iš jos sugeneruota pradinė būsenų diagrama. 11 paveikslėlyje pavaizduotos sekančios sekų diagramos, aprašančios bankomato elgseną po pradinių įvykių, o 12 paveikslėlyje – būsenų diagramos, gautos iš minėtų sekų diagramų.

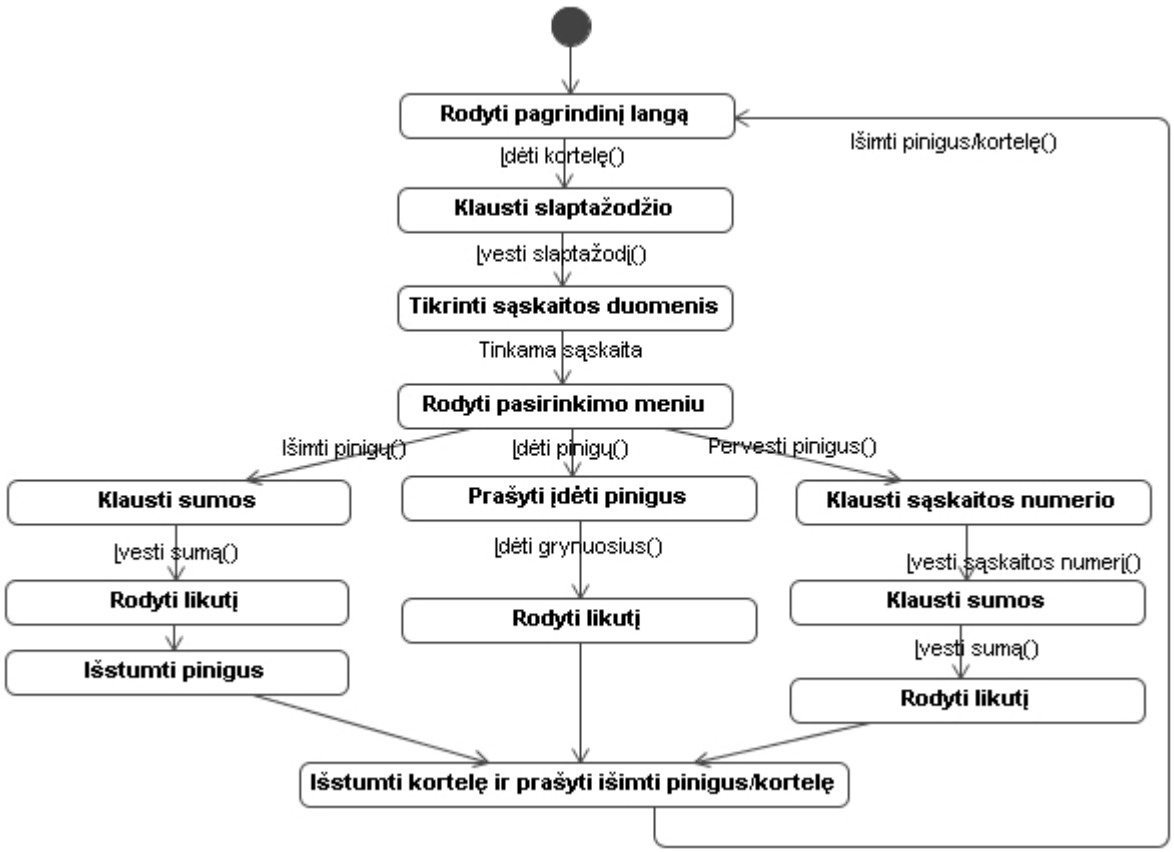


11 pav. Bankomato sekų diagramos „Scenarijus_išimti“, „Scenarijus_įdėti“, „Scenarijus_pervesti“



12 pav. Gautos pradinės būsenų diagramos

13 paveikslėlyje pavaizduota galutinė būsenų mašina, gauta apjungus kelias būsenų mašinas atsižvelgiant į priklausomybės diagramą ir apjungiant vienodas būsenas ir perėjimus.



13 pav. Gauta galutinė būsenų diagrama bankomatui

2.5. Transformacijos metodų palyginimas

Visuose algoritmuose transformacijos paremtos UML specifikacijoje bendrai aprašyta sekų diagramos transformavimo į būsenų diagrama galimybe. Gautas pranešimas sekų diagramoje yra susiejamas su perėjimu iš vienos būsenos į kitą būsenų diagramoje. Išsiųstas pranešimas laikomas veiksmu, jis susiejamas su būsenoje atliekamu veiksmu arba perėjimo metu atliekama veiksmu, kuris veda į būsenos pasikeitimą.

Kai objektas dalyvauja keliuose scenarijuose, buvo pasiūlyta būsenų diagramas kurti apjungus tuos scenarijus. Ryšių tarp scenarijų nustatymui buvo naudojamas panaudos atvejų modelis UCM [1] arba priklausomybės diagrama [17]. Whittle ir Schumann naudoja OCL kalba aprašytus pranešimų apribojimus, kurie naudojami diagramų apjungimui.

Whittle ir Schumann bei Bordeleau ir Corriveau algoritmai nagrinėja hierarchines būsenų mašinas. Būsenų mašinų integravimui Bordeleau ir Corriveau naudoja šablonus. Makinen ir Systa naudoja sugeneruotus teisingus ir neteisingus pavyzdžius būsenų mašinai.

Šie būdai leidžia sekų diagramas atvaizduoti į būsenų mašinų diagramas. Tačiau vartotojui gali prireikti grįžtamojo automatizuoto proceso – sekų diagramų generavimo iš būsenų mašinų. Tokį algoritmą, leidžiantį iš sekų diagramų generuoti būsenų diagramas ir atvirkščiai, pasiūlė dr. Čeponienė L. [4], [5], [6].

Žemiau pateikta lentelė metodų palyginimui pagal tam tikras savybes. Lentelėje lyginami aukščiau apžvelgti metodai ir kitame skyriuje nagrinėjamas dr. L.Čeponienės metodas.

1 lentelė SD į BD transformavimo metodų palyginimas

Savybė	Metodai (pavadinti pagal autorių pavardes)				
	Makinen ir Systa	Whittle ir Schumann	Bordeleau ir Corriveau	Vasilache ir Tanaka	L.Čeponienė
SD į BD transformacija	SD→BD	SD→BD	SD→BD	SD→BD	SD↔BD
Transformavimo pagrindas	Išsiųstų ir gautų pranešimų porų sekos, konsultuojamasi su vartotoju	Gautų ir išsiųstų pranešimų sekos, OCL naudojimas	UCM modelis, aprašantis sąveikas tarp SD, šablonai	Naudojamos priklausomybės diagramos	Įvykių (gauto ir siųsto) porų sekos
UML naudojimas	Sekų ir būsenų diagramos	Sekų ir būsenų diagramos	Sekų ir būsenų diagramos	Sekų ir būsenų diagramos. Įvedamos papildomos priklausomybės diagramos	Sekų ir būsenų diagramos. Būsenų diagramos metamodelis praplėstas

Savybė	Metodai (pavadinti pagal autorių pavardes)				
	Makinen ir Systa	Whittle ir Schumann	Bordeleau ir Corriveau	Vasilache ir Tanaka	L.Čeponienė
Realizavimas įrankyje	Nokia TED	-	-	-	MagicDraw įskiepio pavidalu.
Transformavimas į sudėtingas būsenų mašinas	Nenagrinėjamas.	Yra. Naudojami apribojimai išreikšti OCL kalba	UCM modelio naudojimas	Yra. Naudojamos priklausomybės diagramos	Nagrinėjami ryšiai tarp interfeisų ir esybių būsenų (išreikšti OCL apribojimais)

2.6. CASE įrankių apžvalga

Šiame poskyryje pateikiama kai kurių CASE įrankių galimybių apžvalga. Buvo išbandyti keli gerai žinomi ir plačiai naudojami UML CASE įrankiai. Nagrinėjant įrankius, buvo atsižvelgiama tik į savybes, susijusias su nagrinėjama problema – panagrinėtos įrankių galimybės modelio bei atskirų diagramų transformavimui, kodo generavimui, įrankio funkcionalumui praplėsti, UML ir OCL palaikymas. Palyginimas pateiktas lentelės pavidale. Pasirinktuose įrankiuose naudojamas UML 2.x standartas, palaikomos visos 13 diagramų. Pagal išnagrinėtus algoritmus, buvo lyginama pagal tam tikras savybes – ar yra modelio transformavimas į modelį, ar galima generuoti kodą. Viename pateiktų algoritmų transformavime dalyvauja elementų apribojimai pateikti OCL kalba – ši savybė taip pat įtraukta į palyginimo lentelę. Dar viena svarbi savybė – ar yra diagramų transformavimas, prie šios savybės nurodoma, kokias diagramas galima generuoti iš kitų diagramų. Į lentelę įtraukta įrankio funkcionalumo praplėtimo savybė – ar vartotojas gali kurti įrankiui įskiepius, taip praplėsdamas jį savo sukurtomis funkcijomis.

2 lentelė Keleto CASE įrankių palyginimas

Savybė	Įrankis				
	Rational Rose 2003	Enterprise Architect 6.5	MagicDraw 12.1	Visual Paradigm	UModel 2007
UML palaikymas	UML 2.0	UML 2.1	UML 2.0	UML 2.1	UML 2.1
OCL palaikymas	Yra	Yra	OCL 2.0	Nėra	Nėra
Modelio transformavimas	Yra	Yra	Yra	Yra	Nėra
Kodo generavimas	Yra	Yra	Yra	Yra.	Yra

Savybė	Įrankis				
	Rational Rose 2003	Enterprise Architect 6.5	MagicDraw 12.1	Visual Paradigm	UModel 2007
Diagramų transformavimas	Sekų diagramų į bendradarbiavimo diagramas	Nėra	Nėra	Nėra	Sekų ir bendradarbiavimo diagramų abipusis transformavimas
Įskiepių palaikymas	Yra (Java)	Yra (C#, C++, Delphi)	Yra (Java)	Yra (Java)	Nėra

Iš lentelės matome, kad praktiškai visuose nagrinėtuose įrankiuose galima praplėsti jų funkcionalumą (išskyrus UModel 2007). Rational Rose (RR), Enterprise Architect (EA) ir MagicDraw (MD) įrankiai yra panašiausi nagrinėjamomis savybėmis.

MagicDraw įrankis yra bene sparčiausiai tobulėjantis, kuriamas Lietuvoje UML įrankis. Baltijos programinė įranga, kurianti šį įrankį, bendradarbiauja su Informacijos sistemų katedra, kurioje vykdomi moksliniai tyrimai siejami su įrankio tobulinimu. Įrankis plačiai naudojamas tiek Lietuvoje, tiek pasaulyje. Tai pagrindinės priežastys, dėl ko buvo pasirinktas būtent šis įrankis. Įrankio kūrėjai teikia pagalbą įrankio vartotojams – per kelias dienas atsako į pateiktus klausimus lietuvių arba anglų kalba. Dar viena šio įrankio pasirinkimo priežastis – kitų projektų vykdymas, kuriant būtent MagicDraw funkcionalumo praplėtimą. Kuriamas projektas vėliau galėtų būti integruojamas su jau sukurtais ir dar kuriamais produktais, kurie praplečia MD įrankio funkcionalumą.

2.7. Analizės išvados

Sekų ir būsenų diagramų suderinimas yra aktualus kuriant išsamų sistemą aprašantį modelį. Literatūroje nagrinėjama būsenų diagramų generavimo iš sekų diagramų galimybė.

Nagrinėtuose CASE įrankiuose yra galimybė atlikti MDA transformacijas, tačiau sekų ir būsenų diagramų transformavimo galimybės nei vienas įrankis neturi.

Algoritmų įgyvendinimui naudinga būtų praplėsti esamų CASE įrankių funkcionalumą, o ne kurti naujus įrankius.

Tik vienas nagrinėtas literatūroje algoritmas realizuotas CASE įrankyje, tačiau jis neplatinamas ir nėra galimybės išbandyti įgyvendinto funkcionalumo.

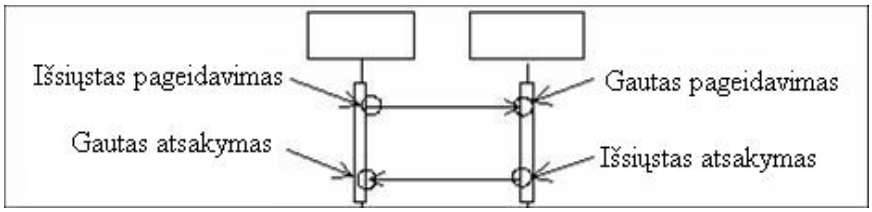
3. ABIPUSIO SEKŲ IR BŪSENŲ DIAGRAMŲ TRANSFORMAVIMO METODAS

Abipusio sekų ir būsenų diagramų transformavimo algoritmą pasiūlė dr. L. Čeponienė [4], [6]. Norint susieti sekų ir būsenų diagramas bei atlikti abipusį diagramų transformavimą, reikia šiek tiek papildyti UML metamodelį. Taip pat sekų ir būsenų diagramos turi tenkinti tam tikrus reikalavimus apžvelgiamus žemiau.

3.1. Sekų diagramos

Sekos diagramos aprašo sistemos interfeisų sąveiką su išoriniu pasauliu bei sistemos objektų tarpusavio sąveiką. Pagrindiniai sekų diagramos elementai yra gyvavimo linijos ir pranešimai.

Kiekviena gyvavimo linija yra susieta su aktoriumi, interfeisu arba klase. Pranešimai gali būti užklausos (call) ir atsako (reply), o įvykiai, susieti su pranešimais, gali būti gauti (recieved) arba išsiųsti (sent). 14 paveikslėlyje pavaizduoti šie įvykių tipai.



14 pav. Pranešimų tipai

Pageidavimo tipo pranešimas yra susietas su atitinkamo interfeiso/klasės operacija.

Norint atlikt transformaciją, reikia patikrinti, ar sekų diagrama tenkina sekančius apribojimus:

- jei sekos diagramoje yra pranešimas, joje turi būti ir bent viena gyvavimo linija;
- pranešimo gavėjo ir siuntėjo gyvavimo linijos priklauso tai pačiai sekos diagramai;
- gyvavimo linija turi būti susieta bent su vienu pranešimu;
- gyvavimo linija turi būti susieta su objektu (klase, interfeisu ar aktoriumi);
- pranešimas turi būti susietas su vienu įvykiu;
- gautas užklausos (*call*) pranešimas turi būti susietas su operacija.

Jeigu visi apribojimai tenkinami, galima vykdyti transformaciją.

3.2. Būsenų diagramos

Būsenų diagrama aprašo dinaminį sistemos aspektą. Būsena suprantama kaip objekto gyvavimo sąlyga ar situacija, kai objektas tenkina tam tikrą sąlygą, vykdo kokį nors veiksmą

arba laukia kokio nors įvykio. Būsenų mašiną sudaro įvykiai, interfeisų reakcijos į įvykius ir būsenų sekos.

Nagrinėjamos interfeisų būsenų mašinos skiriasi nuo UML 2.0 būsenų mašinų:

- laikoma, kad kiekvienas interfeisas visada turi nustatytą būseną vadinamą laukimo būsena;
- būsenų diagramoje yra informacija apie perėjimo metu siųstų ar gautų iš kitų interfeisų įvykių siuntėjus ir gavėjus (to nėra UML 2.0 būsenų mašinose). Tam būsenų diagramoje visiems įvykiams taikomas stereotipas su siuntėjo arba gavėjo žymena (*tag*). Šis stereotipas praplečia UML modelį, kad SD transformavimo į BD metu nebūtų prarasta informacija;
- interfeisas yra Laukimo būsenoje, iš kurios, gavęs užklausą, pereina į tos užklausos vykdymo būseną. Įvykdęs užklausą, interfeisas vėl grįžta į laukimo būseną.

Būsenų mašinose vaizduojama kiekvieno interfeiso sąveikų su kitais interfeisais visuma. Tuo tarpu atskira sekų diagrama vaizduoja vienos sąveikos scenarijų.

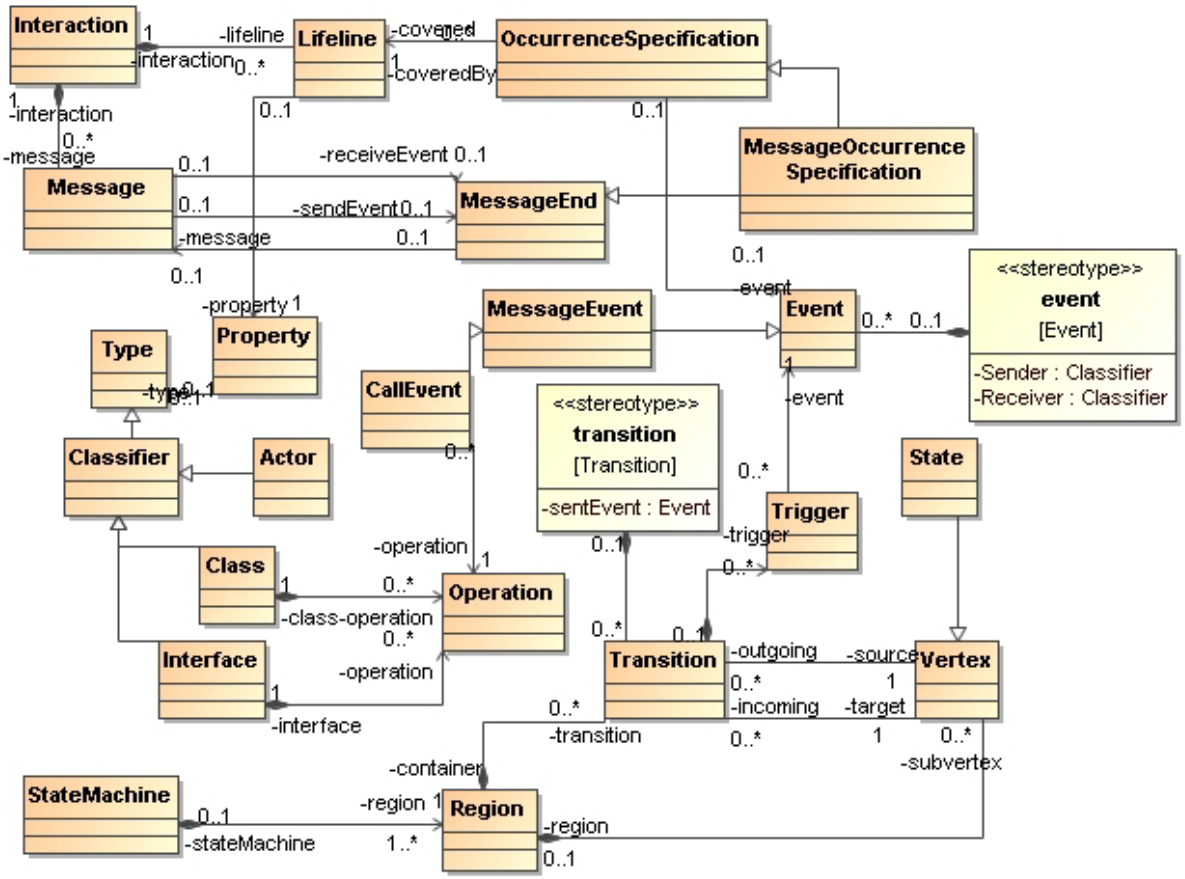
Norint atlikt transformaciją, reikia patikrinti, ar būsenų diagrama tenkina sekančius apribojimus:

- kiekvienas interfeisas būsenų diagramoje (BD) visada privalo turėti nustatytą (*default*) būseną vadinamą laukimo būsena (*WaitState*), kurios metu objektas laukia įvykio, kad pereitų į kitą būseną. Tokia būsena turi būti viena;
- BD laukimo būsena (*WaitState*) turi turėti bent vieną išeinantį perėjimą;
- būsenų diagramoje yra informacija apie perėjimo metu siųstų ar gautų iš kitų interfeisų įvykių siuntėjus ir gavėjus. Visiems įvykiams būsenų diagramoje taikomas stereotipas su siuntėjo arba gavėjo žymena (*tag*);
- interfeisas yra laukimo būsenoje, iš kurios, gavęs užklausą, pereina į tos užklausos vykdymo būseną. Įvykdęs užklausą, interfeisas vėl grįžta į laukimo būseną.
- jei būsenų diagramoje yra perėjimas, jame turi būti ir bent viena būsena;
- perėjimas turi būti susietas su viena arba dviem būsenom – viena, iš kurios išeina (*source*) ir kita, į kurią ateina (*target*), viena būsena gali būti, kai *source* ir *target* sutampa – perėjimas į tą pačią būseną;
- iš laukimo būsenos išeinantis perėjimas privalo turėti trigerį, susietą su gautu įvykiu;
- gauti užklausos įvykiai (*CallEvent*) turi būti susieti su interfeiso operacijomis;
- perėjimams taikomas stereotipas su siunčiamo įvykio žymena (*tag*). Šis stereotipas praplečia UML metamodelį.

3.3. Sekų ir būsenų diagramų susiejimo taisyklės

Pagrindinės taisyklės, taikomos atvaizduojant ryšius tarp sekų ir būsenų diagramų:

- interfeiso gyvavimo linijos gautas užklausos įvykis siejamas su perėjimu į naują interfeiso būseną gavėjo būsenų diagramoje;
- siunčiamas atsakas visada susiejamas su perėjimu į laukimo būseną *WaitState*;
- siunčiama užklausa susiejama su perėjimo metu siunčiamu įvykiu;
- gautas atsakas gali būti susietas arba su perėjimu į laukimo būseną *WaitState* (jei sekantis įvykis yra siunčiamas atsakas) arba su perėjimu toje esamoje būsenoje (jei sekantis įvykis yra siunčiama užklausa).



15 pav. Metamodelis, atspindintis sekos ir būsenų diagramų ryšius

15 paveikslėlyje pateiktame metamodelyje galima pastebėti, kad pagrindinis elementas, siejantis sekų ir būsenų diagramas yra įvykis (klasė *Event*). Įvykis siejasi su pranešimo siuntimu arba gavimu (ryšiai tarp klasių *Message*, *MessageEnd* ir *Event*), taip pat su būsenų diagramos perėjimu (ryšys tarp klasių *Event*, *Trigger* ir *Transition*). Metamodelis papildytas dviem stereotipais *event* ir *transition*. Stereotipas *event* yra *Event* tipo ir gali būti taikomas visiems *Event* tipo elementams, t.y. įvykiams; jis turi dvi žymenas (*tag*) – *Sender* ir *Receiver*, kurios gali būti susietos su konkrečiais objektais (aktoriais, interfeisais arba klasėmis) – įvykių siuntėjais ir gavėjais. Stereotipas transition gali būti taikomas visiems

perėjimams; jis turi žymeną (*tag*) *sentEvent*, kuri gali būti susieta su perėjimo metu siunčiamu įvykiu.

3.4. Būsenų diagramų generavimas

Būsenų mašinų generavimas iš sekų diagramų susideda iš tokių žingsnių:

(1) SD pertvarkoma į kanoninę formą, kur kiekvienas užklausos (*call*) pranešimas papildomas atitinkamu atsakymo (*reply*) pranešimu,

(2) suporojuomi įvykiai kiekvienai gyvavimo linijai, poros pirmas elementas vaizduoja įvykį atitinkantį būseną, o antras – daugiausia du įvykius atitinkančius perėjimą.

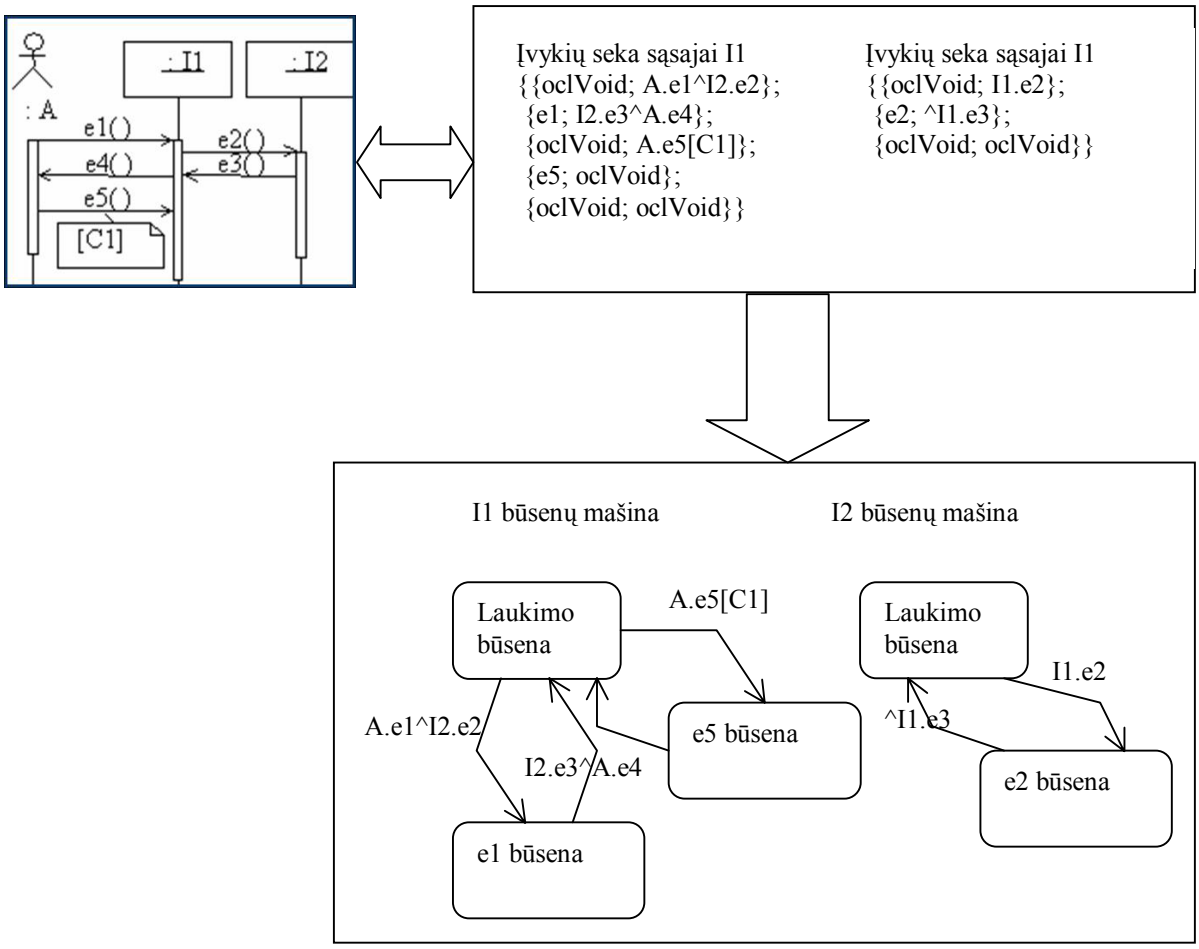
(3) pagal gautas poras sudaromos būsenų mašinos kiekvienam interfeisui.

SD kanoninė forma – kiekvienas sekų diagramos užklausos (*call*) tipo pranešimas yra susiejamas su atsako (*reply*) tipo pranešimu. Jei atsako nėra, pridedamas tuščias atsako pranešimas.

Antrajame etape remiamas sekos diagrama (diagramomis), kurioje dalyvauja interfeisas I. Iš šio interfeiso gyvavimo linijos yra išrenkama porų seka. Kiekvienas gautas užklausos pranešimas sekos diagramoje, atitinka perėjimą būsenų diagramoje. Jei po šio pranešimo seka kitas užklausos pranešimas, kurį siunčia interfeisas I, šis pranešimas prijungiamas prie prieš tai buvusio kaip perėjimo metu siunčiamas įvykis. Šiame etape įvertinamas ir bet kurio pranešimo apribojimas (jei yra) – jis prijungiamas prie pranešimo aprašo. Seką sudaro būsena ir perėjimą atitinkančios poros. Laukimo būseną žymi pirmas poros narys *OclVoid*.

Antrame etape gautos poros naudojamos generuojant būsenų diagramą. Generavimo metu pirmas poros narys tampa būsena, o antrasis – perėjimu iš šios būsenos. Tikrinami pasikartojantys perėjimai ir (ar) būsenos. Jei sekoje randama pasikartojanti būsena, tikrinama ar toks pats yra ir į ją įeinantis perėjimas.

16 paveikslėlyje pavaizduota sekų diagrama, iš jos gautos įvykių poros ir suformuotos būsenų diagramos kiekvienam interfeisui.



16 pav. Sekų diagramos transformavimas į būsenų diagramą

3.5. Sekų diagramų generavimas

Sekų diagramų generavimas iš būsenų diagramų susideda iš tokių etapų:

(1) būsenų diagrama pertvarkoma į kanoninę formą, diagramoje turi būti įvardinti įvykių siuntėjai ir gavėjai,

(2) įvykių poros gaunamos pereinant būsenos diagramoje visus galimus kelius iš laukimo būsenos atgal į laukimo būseną, tuos kelius įsimenant,

(3) iš gautų porų sudaromos sekų diagramas pagal tokias taisykles:

- jeigu pirmas poros elementas yra OclVoid, antras elementas žymi pageidavimo pranešimą; šio antrojo elemento siunčiamas įvykis žymi pageidavimo pranešimą.
- jeigu pirmas poros elementas ir pirmas sekančios poros elementas nėra OclVoid, tada antrasis poros narys žymi atsakymo pranešimą; siunčiamas įvykis žymi pageidavimo pranešimą
- jeigu pirmas poros narys nėra OclVoid, tačiau sekančios poros pirmas narys yra OclVoid, antrasis elementas žymi atsakymo pranešimą ir siunčiamas įvykis taip pat žymi atsakymo pranešimą.

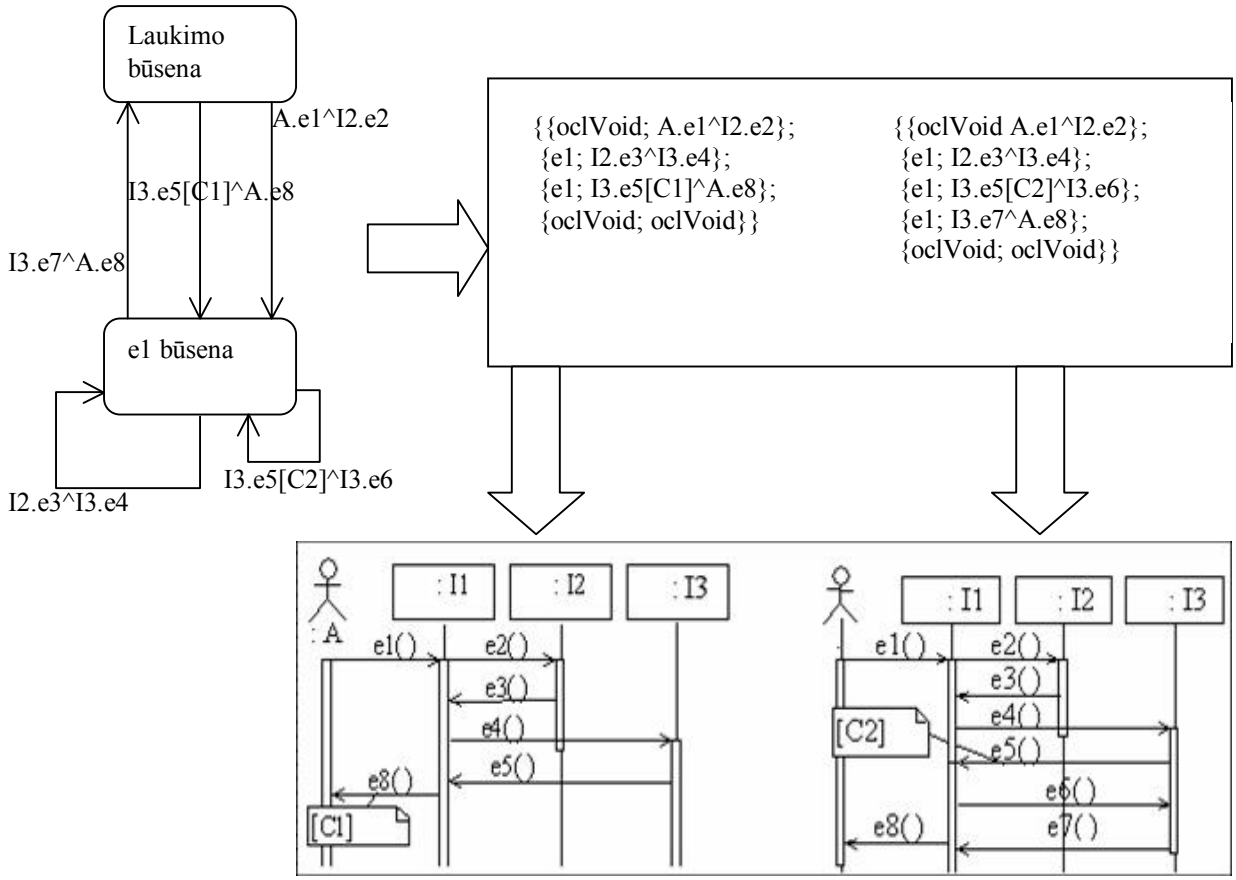
Būsenų diagramų įvykiai atitinka pranešimus sekos diagramose, bet vaizduojant būsenas ir perėjimus standartinėje UML būsenų diagramoje, nėra informacijos apie gautų įvykių siuntėjus ir siunčiamų įvykių gavėjus. Tačiau norint generuoti sekos diagramas, būsenų diagramose būtina nurodyti visų interfeiso (ar kito objekto) siunčiamų įvykių gavėjus ir gaunamų įvykių siuntėjus.

Porų seka yra gaunama einant per būsenų mašiną iš laukimo būsenos į laukimo būseną visais galimais keliais ir atsimenant nueitą kelią, bei užrašant jį būsenų ir iš jų išeinančių perėjimų poromis.

Sekų diagramų generavimas iš porų sekų vykdomas laikantis tokių taisyklių:

- jei būsenų mašinos savininko, nagrinėjamo įvykio siuntėjo ar gavėjo gyvavimo linijų nėra generuojamoje sekų diagramoje, jos pridedamos.;
- jei poros pirmasis narys yra *OclVoid*, antrasis narys atvaizduojamas į būsenų mašinos savininko gautą užklausos pranešimą; šio antrojo nario siunčiamas įvykis atvaizduojamas į būsenų mašinos savininko išsiųstą užklausos pranešimą;
- jei poros pirmasis narys yra ne *OclVoid*, bet sekančios poros pirmasis narys yra *OclVoid*, tuomet nagrinėjamos poros antrasis narys atvaizduojamas į būsenų mašinos savininko gautą atsako pranešimą; šio antrojo nario siunčiamas įvykis atvaizduojamas į būsenų mašinos savininko išsiųstą atsako pranešimą;
- jei poros pirmasis narys yra ne *OclVoid*, bei sekančios poros pirmasis narys yra ne *OclVoid*, tuomet nagrinėjamos poros antrasis narys atvaizduojamas į būsenų mašinos savininko gautą atsako pranešimą; šio antrojo nario siunčiamas įvykis atvaizduojamas į būsenų mašinos savininko išsiųstą užklausos pranešimą .

17 paveikslėlyje pavaizduota būsenų diagrama, iš jos gautos įvykių poros ir suformuotos sekų diagramos kiekvienam ciklui.



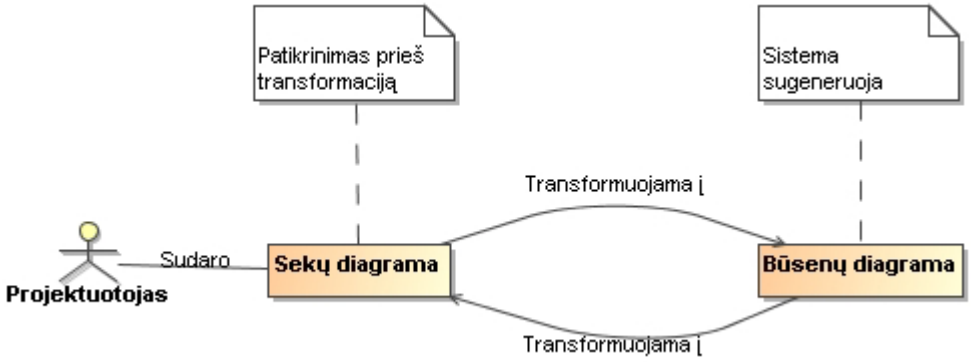
17 pav. Būsenų diagramos transformavimas į sekų diagramas

4. MAGICDRAW ĮRANKIO IŠPLĖTIMO PROJEKTAS

4.1. Funkciniai reikalavimai sistemai

Kuriama sistema turi transformuoti sekų diagramą į būsenų diagramas ir atvirkščiai, patikrinti suderinamumą tarp šių diagramų. Prieš transformavimą turi būti atliekamas sekų ir būsenų diagramų korektiškumo patikrinimas. Diagramos sukuriamos naudojant MagicDraw įrankį arba gali būti importuotos XMI formatu (tai MagicDraw įrankio galimybė). Transformacijos rezultate sukuriami diagramos modelio elementai, bei gauta būsenų diagrama atvaizduojama grafiniu pavidalu. Deja, MagicDraw *openAPI* neleidžia grafiškai atvaizduoti sekų diagramos sugeneruotų pranešimų, todėl sekų diagramos pranešimai vaizduojami tekstiniame pavidale pranešimų lange.

Sistemos koncepcinė schema pavaizduota 18 paveiksle.



18 pav. Sistemos konceptualus modelis

Iš užsakovo pateiktų reikalavimų buvo sudaryti panaudos atvejai pavaizduoti 19 pav. Žemiau pateiktas panaudojimo atvejų sąrašas su jų aprašymais.

19 pav. Sistemos panaudos atvejų diagrama

Panaudojimo atvejų sąrašas

1. PANAUDOJIMO ATVEJIS:	transformuoti modelio sekų diagramas į būsenų diagramas
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	Vykdomas transformavimas iš sekų diagramos į būsenų diagramą
Prieš-sąlyga:	turi būti sukurtas ir pasirinktas transformavimui modelis su sekų diagramomis ir sąveikaujančiomis klasėmis.
Sužadinimo sąlyga:	pasirenkamas SD į SM transformavimo meniu punktas.
Po-sąlyga:	gautos būsenų diagramos

1.1. PANAUDOJIMO ATVEJIS:	patikrinti modelio sekų diagramų sintakse
Vartotojas/Aktorius:	Projektuotojas, sistema
Aprašas:	patikrinama, ar sekų diagramos elementai tenkina apribojimus

1.1. PANAUDOJIMO ATVEJIS:	patikrinti modelio sekų diagramų sintakse
Prieš-sąlyga:	yra nuskaitytas modelis su sekų diagramomis.
Sužadinimo sąlyga:	pasirenkamas SD į SM transformavimo meniu punktas.
Po-sąlyga:	Gaunami tikrinimo rezultatai

1.2. PANAUDOJIMO ATVEJIS:	sudaryti įvykių poras
Vartotojas/Aktorius:	Projektuotojas, sistema
Aprašas:	Iš sekų diagramos elementų sudaromos įvykių poros
Prieš-sąlyga:	turi būti sukurtas ir pasirinktas transformavimui modelis su sekų diagramomis, jis turi atitikti apribojimus.
Sužadinimo sąlyga:	pasirenkamas SD į SM transformavimo meniu punktas.
Po-sąlyga:	Gaunamos įvykių poros

1.3. PANAUDOJIMO ATVEJIS:	sudaryti būsenų diagramas
Vartotojas/Aktorius:	Projektuotojas, sistema
Aprašas:	Iš įvykių porų sudaromos būsenų diagramos
Prieš-sąlyga:	turi būti sudarytos įvykių poros.
Sužadinimo sąlyga:	pasirenkamas SD į SM transformavimo meniu punktas.
Po-sąlyga:	sudarytos būsenų diagramos

2. PANAUDOJIMO ATVEJIS:	transformuoti būsenų diagramas į sekų diagramas
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	Vykdomas transformavimas iš būsenų diagramos į sekų diagramą
Prieš-sąlyga:	turi būti sukurtas ir pasirinktas transformavimui modelis su būsenų diagramomis ir sąveikaujančiomis klasėmis.
Sužadinimo sąlyga:	pasirenkamas SM į SD transformavimo meniu punktas.
Po-sąlyga:	gautos sekų diagramos

2.1. PANAUDOJIMO ATVEJIS:	patikrinti modelio būsenų diagramų sintaksę
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	patikrinama, ar būsenų diagramos elementai tenkina apribojimus
Prieš-sąlyga:	yra nuskaitytas modelis su būsenų diagramomis.
Sužadinimo sąlyga:	pasirenkamas SM į SD transformavimo meniu punktas.

2.1. PANAUDOJIMO ATVEJIS:	patikrinti modelio būsenų diagramų sintaksę
Po-sąlyga:	gaunami tikrinimo rezultatai

2.2. PANAUDOJIMO ATVEJIS:	sudaryti įvykių poras
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	Iš perėjimų sudaromos įvykių poros
Prieš-sąlyga:	turi būti sukurtas ir pasirinktas transformavimui modelis su būsenų diagramomis, jis turi atitikti apribojimus.
Sužadinimo sąlyga:	pasirenkamas SM į SD transformavimo meniu punktas.
Po-sąlyga:	gautos įvykių poros

2.3. PANAUDOJIMO ATVEJIS:	sudaryti sekų diagramas
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	Iš gautų įvykių porų sudaromos sekų diagramos
Prieš-sąlyga:	turi būti sudarytos įvykių poros.
Sužadinimo sąlyga:	pasirenkamas SM į SD transformavimo meniu punktas.
Po-sąlyga:	gautos sekų diagramos

3. PANAUDOJIMO ATVEJIS:	tikrinti diagramų suderinamumą
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	Vykdomas diagramų suderinamumo tikrinimas
Prieš-sąlyga:	turi būti sukurtas ir pasirinktas tikrinimui modelis su sekų ir būsenų diagramomis ir sąveikaujančiomis klasėmis.
Sužadinimo sąlyga:	pasirenkamas suderinamumo tikrinimo meniu punktas.
Po-sąlyga:	gauti suderinamumo tikrinimo rezultatai

3.1. PANAUDOJIMO ATVEJIS:	patikrinti modelio sekų diagramų sintakse (sutampa su 1.1. panaudojimo atveju)
Vartotojas/Aktorius:	Projektuotojas, sistema
Aprašas:	patikrinama, ar sekų diagramos elementai tenkina apribojimus
Prieš-sąlyga:	yra nuskaitytas modelis su sekų ir būsenų diagramomis.
Sužadinimo sąlyga:	pasirenkamas suderinamumo tikrinimo meniu punktas.
Po-sąlyga:	Gaunami tikrinimo rezultatai

3.2. PANAUDOJIMO ATVEJIS:	patikrinti modelio būsenų diagramų sintaksę (sutampa su 2.1. panaudojimo atveju)
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	patikrinama, ar būsenų diagramos elementai tenkina apribojimus
Prieš-sąlyga:	yra nuskaitytas modelis su sekų ir būsenų diagramomis.
Sužadinimo sąlyga:	pasirenkamas suderinamumo tikrinimo meniu punktas.
Po-sąlyga:	gaunami tikrinimo rezultatai

3.3. PANAUDOJIMO ATVEJIS:	sudaryti įvykių poras (susideda iš 1.2. ir 2.2. panaudojimo atvejų)
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	Iš sekų diagramos elementų sudaromos įvykių poros Iš būsenų diagramos perėjimų sudaromos įvykių poros
Prieš-sąlyga:	turi būti sukurtas ir pasirinktas transformavimui modelis su sekų ir būsenų diagramomis, jis turi atitikti apribojimus.
Sužadinimo sąlyga:	pasirenkamas suderinamumo tikrinimo meniu punktas.
Po-sąlyga:	gautos įvykių poros

3.4. PANAUDOJIMO ATVEJIS:	lyginti diagramas
Vartotojas/Aktorius:	Projektuotojas
Aprašas:	Vykdomas gautų įvykių porų palyginimas
Prieš-sąlyga:	turi būti sudarytos įvykių poros sekų ir būsenų diagramoms
Sužadinimo sąlyga:	pasirenkamas suderinamumo tikrinimo meniu punktas.
Po-sąlyga:	gauti palyginimo rezultatai

4.2. Nefunkciniai reikalavimai sistemai

Reikalavimai sistemos išvaizdai

Bendri reikalavimai vartotojo sąsajai:

- sistemos vartotojo sąsaja turėtų derėti su MagicDraw vartotojo sąsaja;
- turi būti paprasta ir suprantama vartotojui;
- sąveikaujanti sąsaja (turi vykti dialogas tarp sistemos ir vartotojo).

Reikalavimai panaudojamumui

- Vartotojui turėtų būti lengva naudotis sistema, sąsaja suderinta su MagicDraw sąsaja.
- Anglų kalbos panaudojimas.

Reikalavimai vykdymo charakteristikoms

Sistema turi efektyviai išnaudoti jai prieinamus resursus ir skaičiavimai turi būti įvykdomi per vartotojui priimtiną laiką.

Reikalavimai sistemos priežiūrai

- Sistema yra ne kritinė, tačiau klaidų taisymas turėtų būti vykdomas pakankamai greitai.
- Sistemos priežiūra turėtų būti paprasta.

Reikalavimai standartams

Sistema turi atitikti MDA standartus (UML, XMI, JMI, MOF).

Reikalavimai saugumui

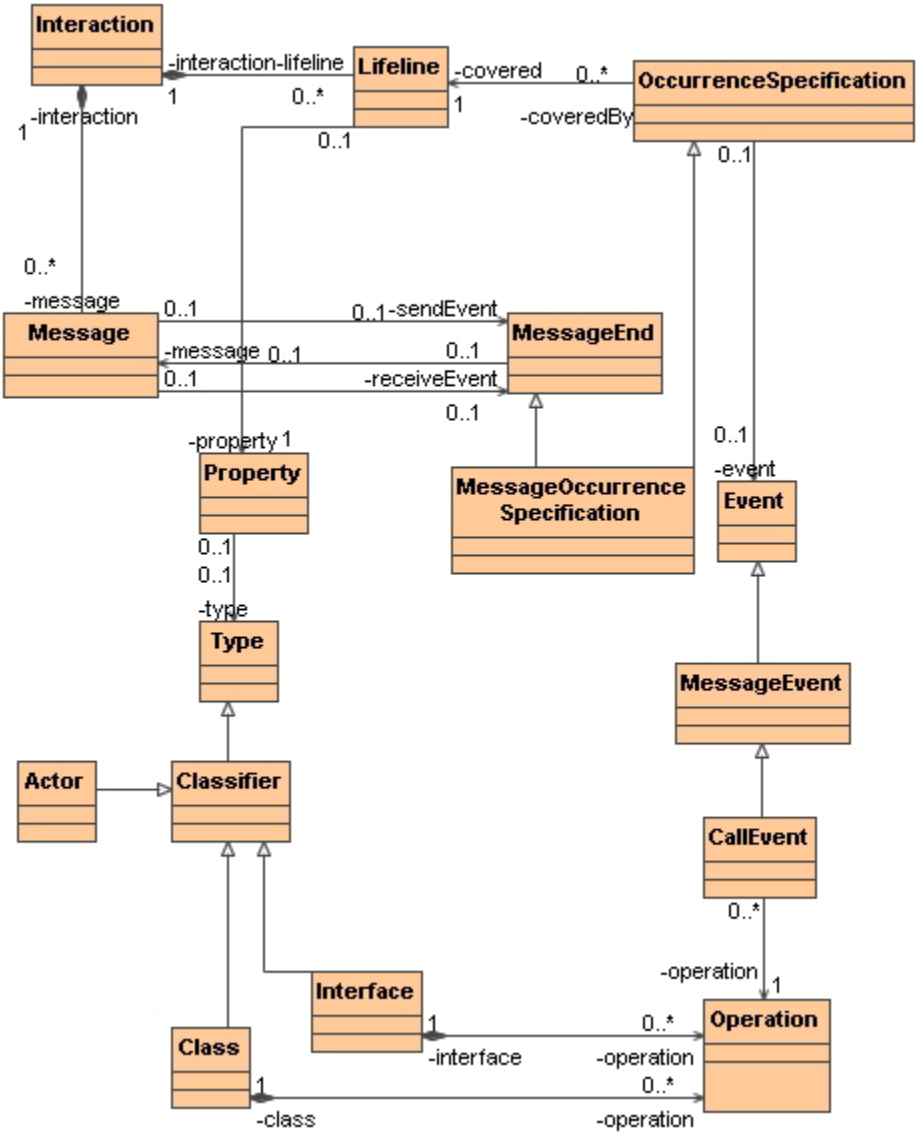
Sistemos vientisumas – sistema neturėtų iškraipyti ar sugadinti vartotojo sukurtų diagramų.

Kultūriniai-politiniai reikalavimai

- Turi būti naudojama taisyklinga anglų kalba.
- Negalima naudoti įžeidžiančių frazių ar paveiksliukų.

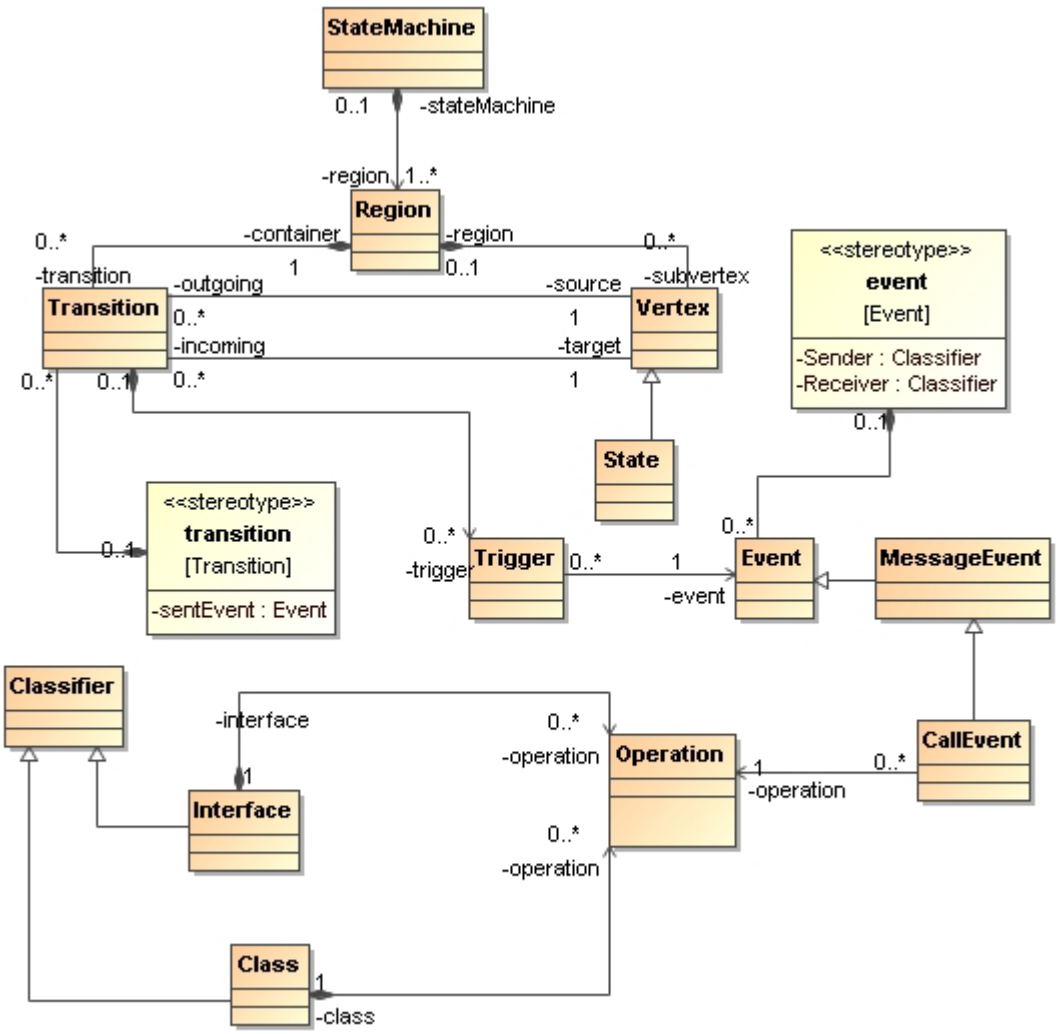
4.3. Duomenų struktūra

Sistemoje naudojami OMG UML specifikacijoje aprašomi sekų ir būsenų diagramų metamodeliai. Šie metamodeliai pavaizduoti 20 ir 21 paveikslėliuose.



20 pav. Sekų diagramos duomenų modelis

Sekų diagramą vaizduojantis elementas yra *Interaction*. Jis gali turėti daug pranešimų (*Message*) ir daug gyvavimo linijų (*Lifeline*). Kiekviena gyvavimo linija yra susieta su vienu klasifikatoriumi (ryšys su klase *Property*, kurios tipas (*Type*) gali būti aktorius (*Actor*), klasė (*Class*) arba interfeisas (*Interface*). Kiekvienas pranešimas turi du galus (*sendEvent* ir *receiveEvent*). Jie susieti su gyvavimo linija per klasę *MessageOccurrenceSpecification*. Kai pranešimas yra užklausos (*call*), jo *receiveEvent* susietas su įvykiu *Event*, kai pranešimas yra atsako (*reply*) *sendEvent* susietas su įvykiu *Event*. Pranešimas visada susietas tik su vienu įvykiu iš *Event* klasės. Užklausos tipo įvykis (CallEvent) gali būti susietas su viena operacija. MagicDraw kuriant užklausos pranešimą jam galima priskirti interfeiso, į kurį nukreiptas pranešimas, operaciją.



21 pav. Būsenų diagramos duomenų modelis

Būsenų mašiną vaizduojantis elementas (*StateMachine*) gali turėti daug sričių (*Region*). Srityje gali būti daug būsenų (klasė State) ir perėjimų (Transition). Perėjimas susietas su viena šaltinio (*source*) būsena ir viena būsena, į kurią yra nukreiptas (target). Tuo tarpu būsena gali turėti daug išeinančių ir įeinančių perėjimų. Perėjimas susietas su įvykiu per trigerį. Perėjimui naudojamas stereotipas *transition* papildo UML metamodelį, perėjimui galima nurodyti siųstą įvykį naudojant žymeną (*tag*) *sentEvent*.

4.4. Sistemos architektūra

Kuriamos sistemos architektūros tikslai:

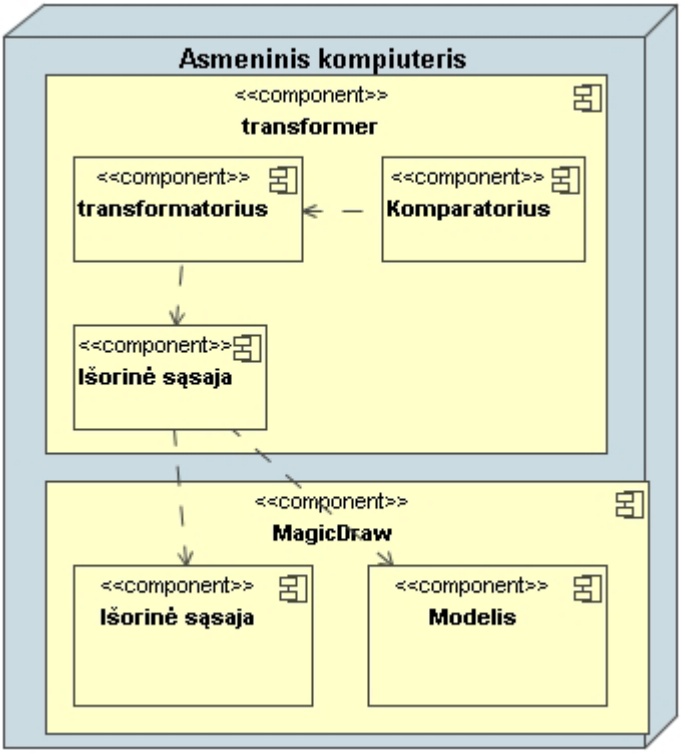
- Nepriklausomumas nuo platformos – sistema turi veikti keliose plačiai
- naudojamose platformose.
- Daugiakalbiškumas -sistemoje galima naudoti skirtingų kalbų simbolius
- duomenų įvedimui/išvedimui (tą užtikrina MagicDraw įrankis).
- Sistemos integravimas į MagicDraw. Sistema turi būti MagicDraw CASE įrankio dalis.

Kuriamai sistemai taikomi architektūros apribojimai:

- Veiklos logika turi būti realizuota remiantis užsakovo pateiktu duomenų struktūros šablonu.
- Kuriama sistema turi būti integruota i MagicDraw UML CASE įrankį.

Paveiksle 22 pavaizduotas sistemos išdėstymo vaizdas atitinkantis sistemos architektūros reikalavimus.

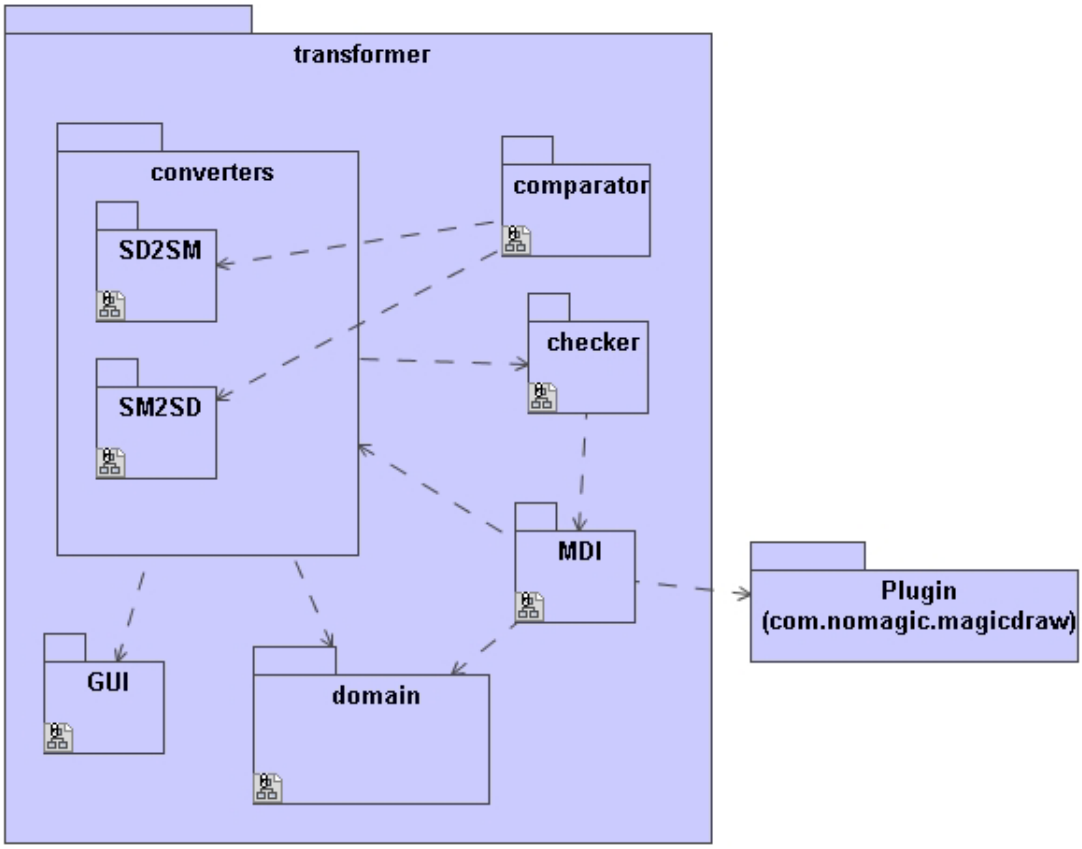
Kuriamą sistemą sudaro tokie komponentai:



22 pav. Sukurtos sistemos komponentai

4.5. Paketų struktūra

Programinė įranga sekų ir būsenų suderinamumui tikrinti kuriama kaip CASE įrankio (MagicDraw UML įrankio) įskiepis. Kuriamos sistemos realizacijai reikalinga paketų struktūra pavaizduota 23 paveiksle.



23 pav. Sistemos paketų struktūra

4.5.1. Paketų aprašymas

Transformer.converters.SD2SM – sekų diagramų transformavimo į būsenų diagramas paketas.

Transformer.converters.SM2SD – būsenų diagramų transformavimo į sekų diagramas paketas.

Transformer.comparator – sekų ir būsenų diagramų palyginimo paketas.

Transformer.domain – paketas skirtas sistemoje naudojamų duomenų struktūroms ir tipams saugoti.

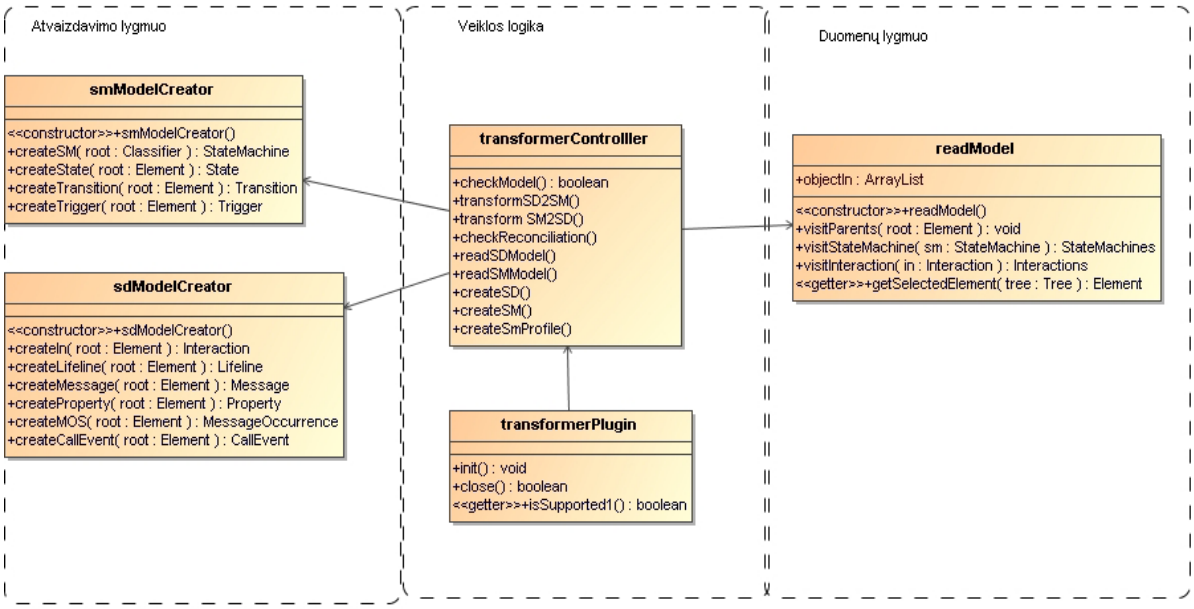
Transformer.MDI – sąsajų su MagicDraw įrankiu paketas.

Com.nomagic.magicdraw.plugins – MagicDraw modeliavimo įrankio paketas, realizuojantis išorinę sąsają, leidžiančią įskiepiams naudotis įrankio funkcionalumu.

Transformer.GUI – vartotojo sąsajos klasių paketas.

4.6. Esminiai projektavimo sprendimai

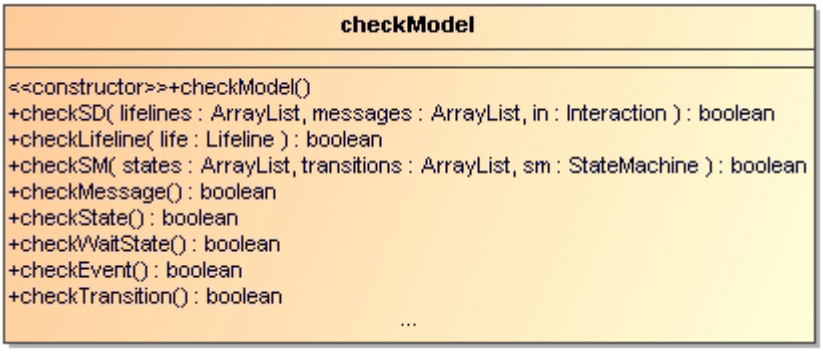
Sistema buvo sukurta naudojant trijų lygimų architektūrą. *transformerController* yra pagrindinė valdymo klasė, kuri pasiekia duomenų lygmenį per *readModel* klasę, bei atvaizduoja modelį vaizdavimo lygmenyje esančių klasių pagalba (24 pav.)



24 pav. Sistemos trijų lygių architektūros realizavimas

4.6.1. Modelio patikrinimo realizavimas

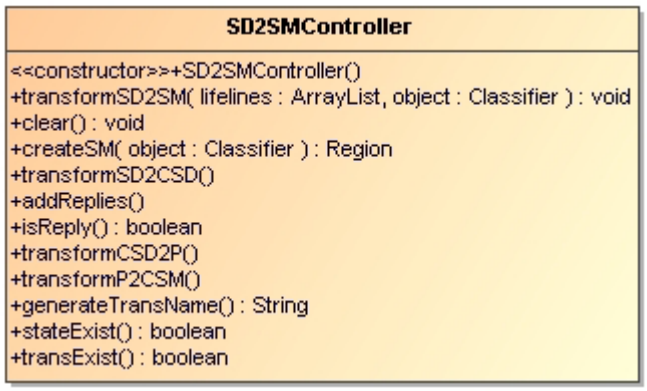
Modelio tikrinimo funkcijos *checkSD()* ir *checkSM()* (25 pav.) iškviečiamos prieš atitinkamų diagramų transformavimą. Tikrinami apribojimai sekų diagramos gyvavimo linijoms ir pranešimams bei įvykiams pagal apribojimus aprašytus 3.1 skyrelyje „Sekų diagramos“. Atitinkamai Būsenų diagramos būsenos, perėjimai yra tikrinami pagal 3.2 skyrelyje „Būsenų diagramos“ aprašytus apribojimus.



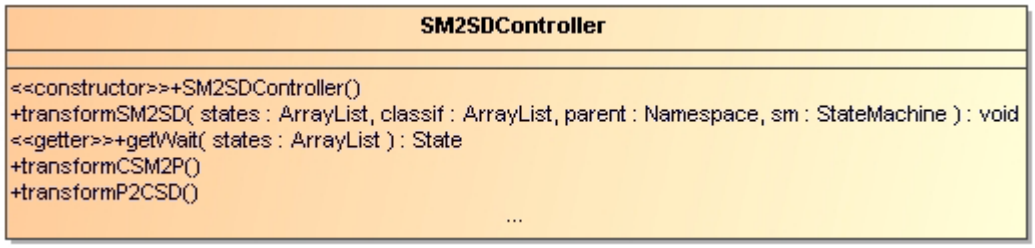
25 pav. Diagramų apribojimo tikrinimą realizuojanti klasė

4.6.2. Transformavimo realizavimas

Transformavimą iš sekų diagramos į būsenų diagramą realizuojanti klasė pavaizduota 26 paveiksle. Atvirkštinį transformavimą vykdanti klasė pavaizduota 27 paveiksle



26 pav. Sekų diagramos transformavimo į būsenų diagramą klasė



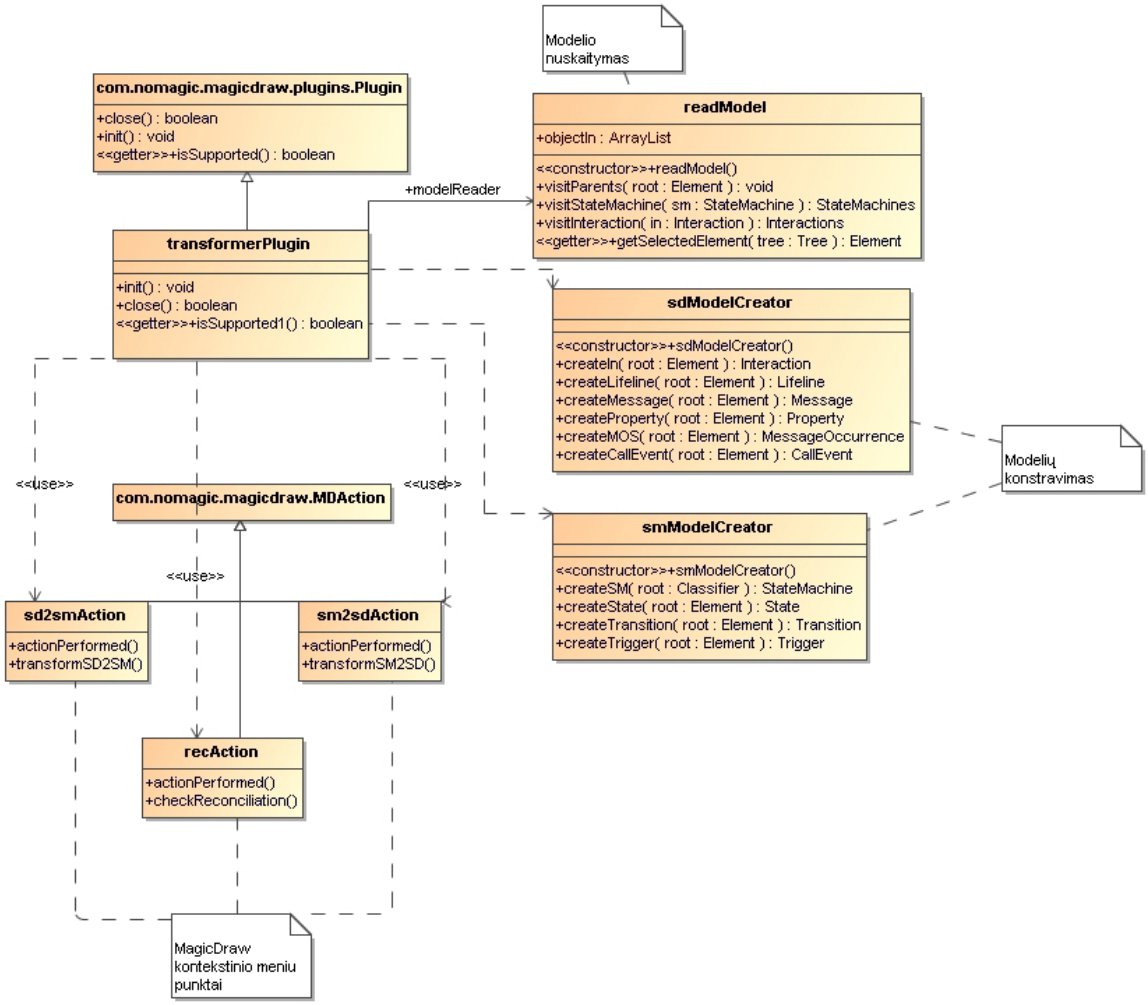
27 pav. Būsenų diagramos transformavimo į sekų diagramą klasė

4.6.3. Sąsaja su MagicDraw

MagicDraw įrankio funkcionalumas išplečiamas įskiepių pagalba.Visi MagicDraw įskiepiai realizuoja com.nomagic.magicdraw.plugins pakete esančią bendrą sąsają *Plugin* (28 pav.). Ši sąsaja aprašo *init()* ir *close()* metodus, kurie yra iškviečiami MagicDraw įrankio darbo pradžios ir pabaigos metu. Šiuose metoduose aprašomi įskiepio inicializavimui reikalingi veiksmai ir panaudotų resursų atlaisvinimo operacijos, kurios iškviečiamos įskiepio darbo pabaigoje.

Įskiepio funkcionalumas valdomas MagicDraw grafine aplinka. Tam tikslui į kontekstinį meniu buvo įterpti transformavimo funkcijų iškvietimo meniu punktai. *MDAction* sąsaja aprašo *actionPerformed()* metodą,kuris yra iššaukiamas kiekvieną kartą vartotojui pasirinkus atitinkamą meniu punktą (28 pav.).

MagicDraw būsenų diagramos generavimo metu sukuriamas *SM_profile* – profilis, skirtas naudojamiems stereotipams saugoti. Klasė *SMProfileManager* praplečia naudojamą UML modelį.



28 pav. Sąsajos su MagicDraw realizavimas

4.7. Sukurtos programų sistemos charakteristikos

Sukurta sistema gali būti praplėsta, pakartotinai panaudota. Sistema gali būti praplėsta naujais diagramų tikrinimo ir transformavimo algoritmais (pvz., sudėtingesnių diagramų transformavimas). 3 lentelėje pateikti kai kurie sistemos parametrų įverčiai.

3 lentelė Sistemos parametrų įverčiai

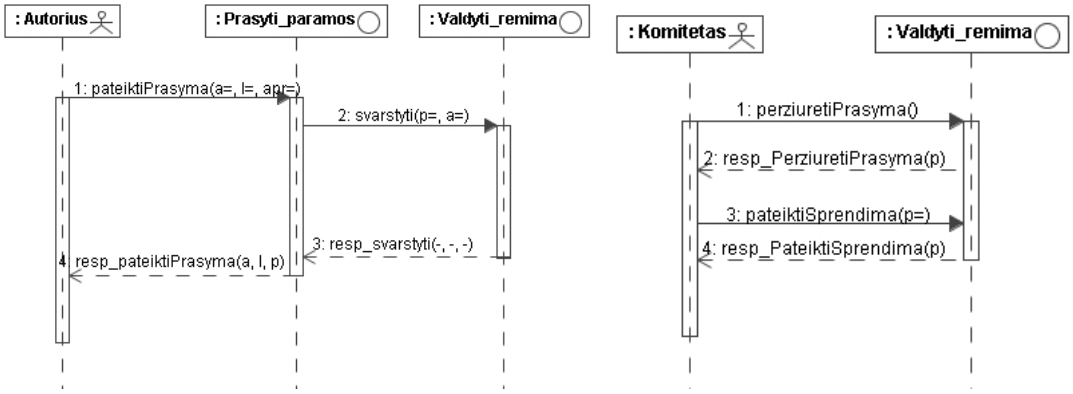
Parametro pavadinimas	Dydis
Statinių metodų kiekis	2
Klasių kiekis	29
Atributų kiekis	59
Metodų kiekis	151
Max parametrų metode kiekis	4
Paveldėjimų kiekis	4
Paketų kiekis	7
Kodo eilučių kiekis	4025

4.8. Sistemos ateities tobulinimo darbai

Naujų sistemos algoritmų kūrimą gali įgyvendinti programuotojas. Ateityje būtų naudinga sistemą praplėsti nagrinėjant sudėtingesnes sekų diagramas, kuriose yra nuorodos į kitas sekų diagramas. UML 2.0 leidžia nupiešti sekų diagramą, kurioje atsispindėtų ryšiai tarp kitų sekų diagramų. Tokiu būdu būtų galima kurti hierarchines būsenų mašinas.

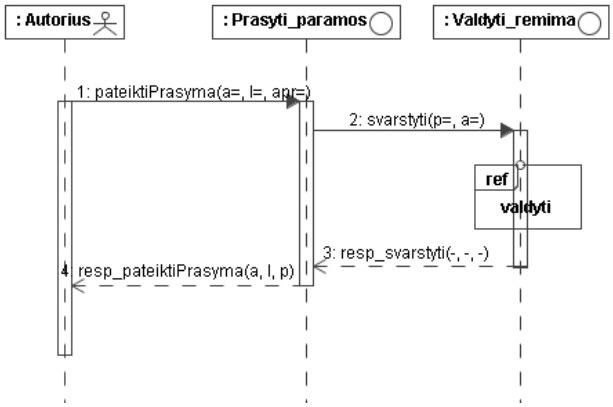
Sistema galėtų būti integruojama su panašiomis kuriamomis sistemomis, kaip būsenų ir klasių, sekų ir klasių suderinimo užtikrinimas. Taip būtų užtikrinamas pilnas DIM [6] modelio suderinimas, jis būtų parengtas transformacijai į PIM modelį.

Sistema galėtų būti praplėsta sudėtingesnių sekų diagramų transformavimo į būsenų mašinas galimybe. Sukurta sistema atlieka sekų diagramų transformavimą į būsenų mašinas. UML 2.0 leidžia kurti sudėtingas sekų diagramas, kuriose gali būti nuorodų į kitas diagramas. Panagrinėkime dvi sekų diagramas pateiktas 29 paveiksle.



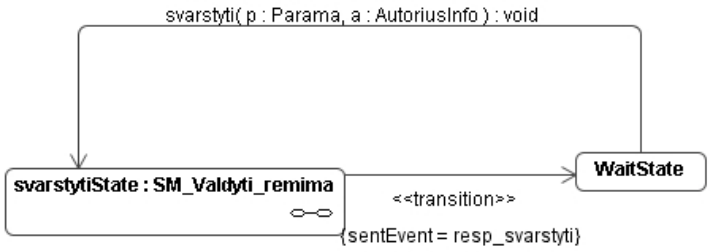
29 pav. Dvi tarpusavyje susiję sekų diagramos: SD „Prašyti“ ir SD „Valdyti“

Pirmoje sekų diagramoje interfeisas „Prašyti_paramos“ gauna pranešimą ir siunčia užklausą interfeisui „Valdyti_rėmimą“. Po šio pranešimo gavimo, interfeisas pereina į jo vykdymo būseną ir prieš siųsdamas atsakymą vykdo veiksmus, nurodytus SD „Valdyti“. Šias dvi sekų diagramas galima atvaizduoti viena sekų diagrama, kurioje atsispindėtų tarp jų esantys ryšiai. 19 paveikslėlyje pateikta sekų diagrama, kurioje po nusiųsto pranešimo „svarstyti“, turėtų būti vykdoma sekų diagrama „valdyti“ (30 pav.)



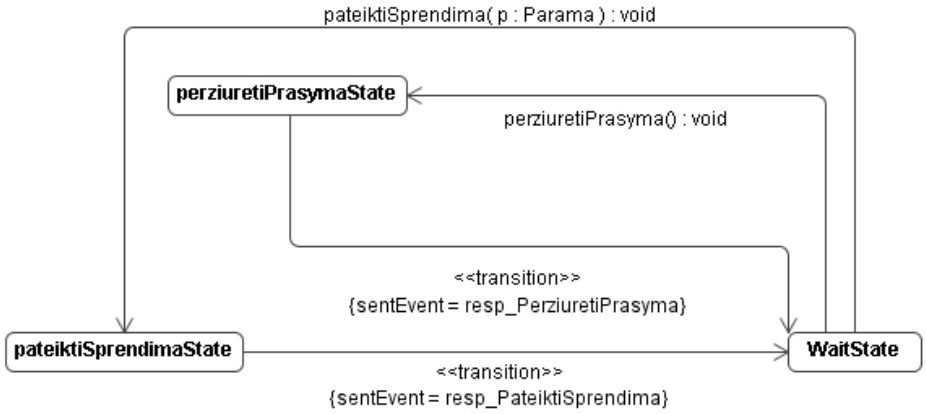
30 pav. Sekų diagrama „Prašyti“ su nuoroda į sekų diagramą „Valdyti“

Interfeisui „Valdyti_remima“ sugeneruojama būsenų diagrama turinti sudėtinę būsena, kaip pavaizduota 31 paveikslėlyje.



31 pav. Interfeiso „Valdyti_remima“ būsenų mašina

Interfeiso „Valdyti_remima“ būsenų mašinos sudėtinė būsena „svarstytiState“ yra susieta su būsenų mašina „SM_Valdyti_remima“ (32 pav.).



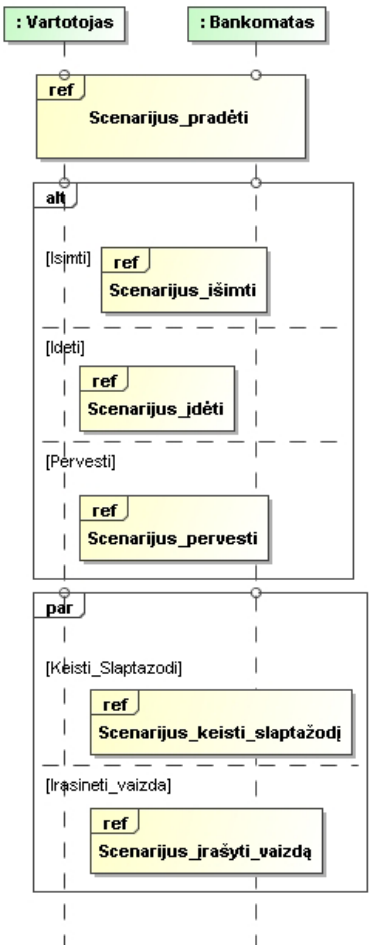
32 pav. Būsenų mašina „SM_Valdyti_remima“

Tarp sekų diagramų gali būti įvairūs ryšiai, Tanaka ir Vasilache pasiūlė priklausomybės diagramas naudoti tų ryšių aprašymui. UML 2.0 leidžia naudoti bendrą sekų diagramą, kurioje galima nurodyti ryšius tarp sekų diagramų naudojant sąveikų fragmentus. 33 paveikslėlyje pateikta sekų diagrama, kurioje atsispindi tie patys ryšiai, kurie pavaizduoti priklausomybės diagramoje (9 pav.) ir aprašyti 2.4. skyrelyje.

Sekų diagramoje gali būti naudojami tokie sąveikų fragmentai:

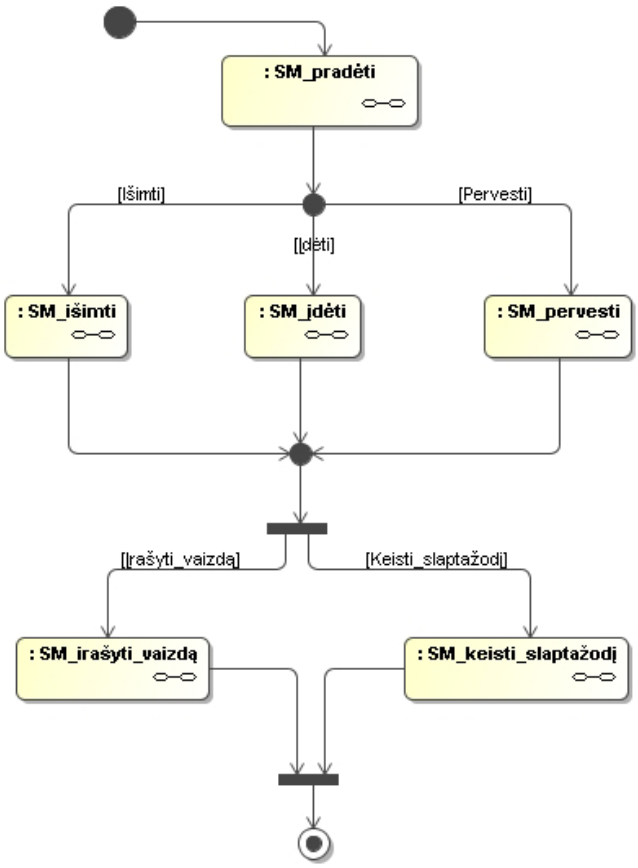
- Jau nagrinėtas nuorodos fragmentas (*ref*) – tai nuorodą į kokią nors sekų diagramą.
- Alternatyvų fragmentas (*alt*) – sąveikoje nurodomi alternatyvios elgsenos, kurios sekų diagramoje gali būti pasirenkamos pagal nurodytus kriterijus. Vienu metu gali būti vykdoma tik vienas scenarijus, aprašomas alternatyvių komponentų. Gali būti nurodytas bet koks alternatyvų skaičius, kiekvienai alternatyvai nurodomi apribojimai (diagramoje jie pateikti laužtiniuose skliausteliuose).
- Lygiagrečių veiksmų fragmentas (*par*) – veiksmai sąveikoje vykdomi lygiagrečiai, t.y. vykdomi vienu metu, kiekvienam lygiagrečiam veiksmui gali būti naudojami apribojimai.

– Ciklas (*loop*) – sąveikoje nurodyta kiek kartų fragmentas gali būti kartojamas. Parametruose nurodoma kiek kartų ciklas gali būti vykdomas (gali būti išreikšta apribojimais).



33 pav. Sekų diagrama aprašanti ryšius tarp kitų sekų diagramų.

33 paveiksle pateiktoje sekų diagramoje interfeisui „Bankomatas“ galima sugeneruoti būsenų mašiną, kuri susideda iš sudėtinių būsenų, kurių kiekviena susieta su būsenų mašina, sugeneruota iš konkrečios sekų diagramos. Tokia būsenų diagrama pateikta 34 paveiksle.



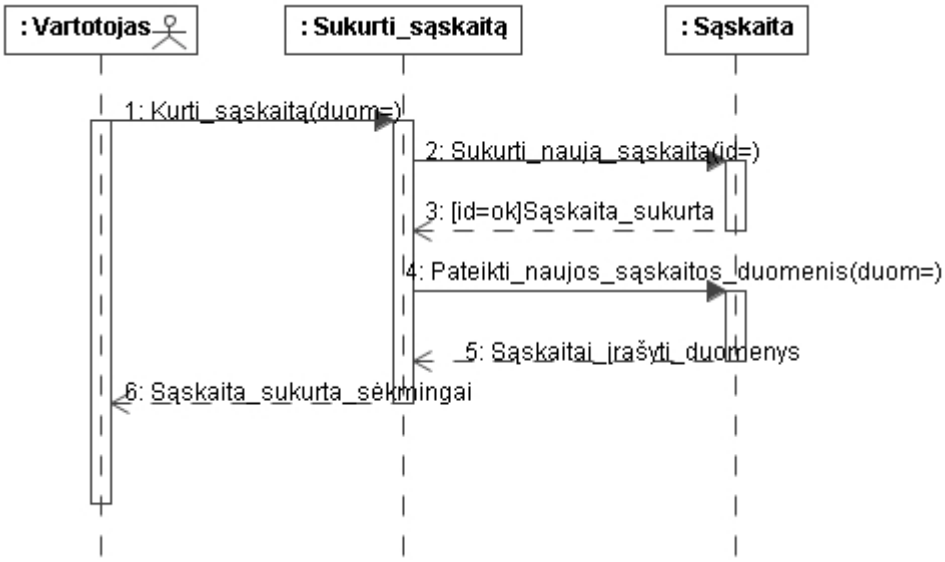
34 pav. Bankomato būsenų mašina

34 paveikslėlyje vaizduojamas būsenų mašinas galima sugeneruoti iš sekų diagramų pagal 3.4. skyrelyje pateiktą algoritmą.

5. SUKURTO ĮSKIEPIO ĮVERTINIMAS

5.1. Sekų ir būsenų diagramų transformacijos pavyzdys

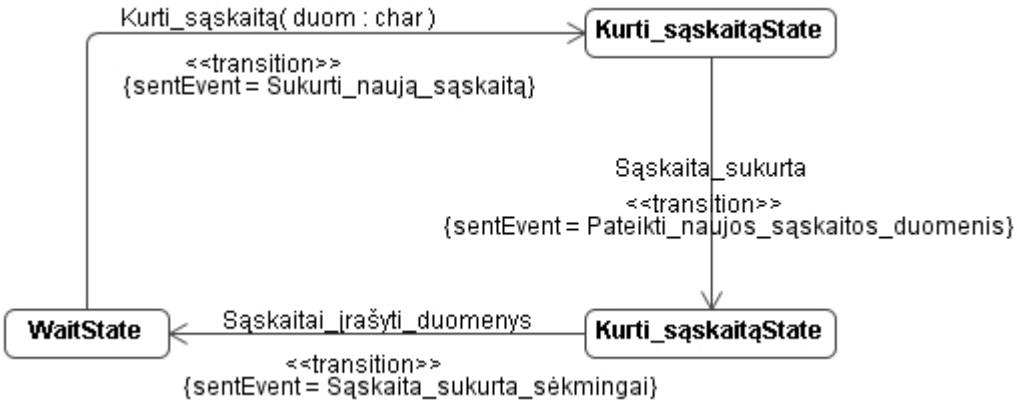
Sukurtas įskiepis buvo testuojamas kuriant sekų diagramas MagicDraw įrankio pagalba, jas automatiškai transformuojant į būsenų mašinas įskiepio pagalba. Gautos būsenų mašinos transformuojamos į sekų diagramas ir patikrinama, ar sutampa gauta sekų diagrama su pradine sekų diagrama. 35 paveikslėlyje pateikta sekų diagrama, vaizduojanti sąskaitos (angl. *account*) sukūrimo scenarijų. Vartotojas, kurdamas naują sąskaitą, įveda duomenis ir siunčia užklausą interfeisui „Sukurti_sąskaitą“. Interfeisas siunčia užklausą klasei „Sąskaita“ su vartotojo nurodytu id (identifikacijos kodu). Sąskaitos klasė grąžina pranešimą, kad sąskaita sėkmingai sukurta (id nenaudojamas kitų vartotojų sąskaitoms). Tuomet interfeisas siunčia užklausą klasei su įvestais duomenimis, kuriuos klasė sąskaita priskiria nurodytam id. Vartotojas per interfeisą „Sukurti_sąskaitą“ gauna atsako pranešimą, kad sąskaita sėkmingai sukurta.



35 pav. Sekų diagramos pavyzdys

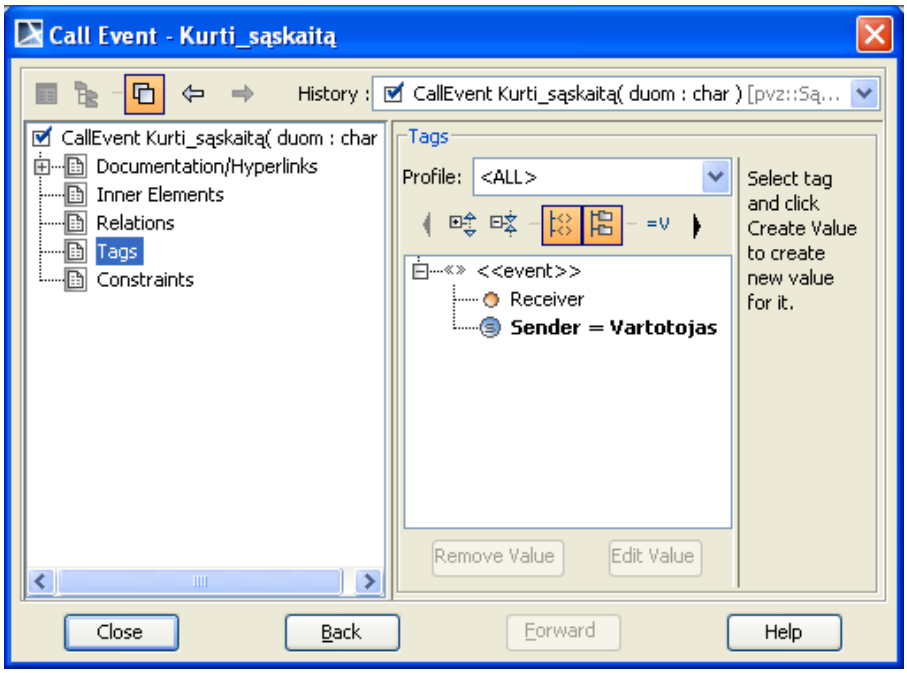
36 paveikslėlyje pavaizduota transformacijos metu gauta būsenų diagrama interfeisui „Sukurti_sąskaitą“. Interfeisas iš laukimo būsenos *WaitState* pereina į būseną *Kurti_sąskaitąState*, gavęs įvykį *Kurti_sąskaitą()*. Su perėjimu susieto stereotipo <<transition>> žymena (angl. *tag*) rodo, koks įvykis siunčiamas perėjimo metu – įvykis *Sukurti_naują_sąskaitą*. Būdamas *Kurti_sąskaitąState* būsenoje interfeisas gauna įvykį *Sąskaita_sukurta*, kuris lemia perėjimą į tą pačią būseną *Kurti_saskaitąState* (ši būsena yra viena, paveikslėlyje tik atvaizduojama atskiromis būsenomis dėl aiškumo). Gavęs atsaką, interfeisas „Sukurti_sąskaitą“ siunčia užklausos įvykį *Pateikti_naujos_sąskaitos_duomenis*,

kuris susiejamas su perėjimo metu siunčiamu įvykiu (naudojantis pritaikyto stereotipo žymena). Toliau gautas atsakas *Sąskaitai_įrašyti_duomenys* ir siunčiamas atsakas *Sąskaita_sėkmingai_sukurta* sąlygoja perėjimą atgal į laukimo būseną *WaitState*.



36 pav. *Sugeneruotos būsenų diagramų pavyzdys*

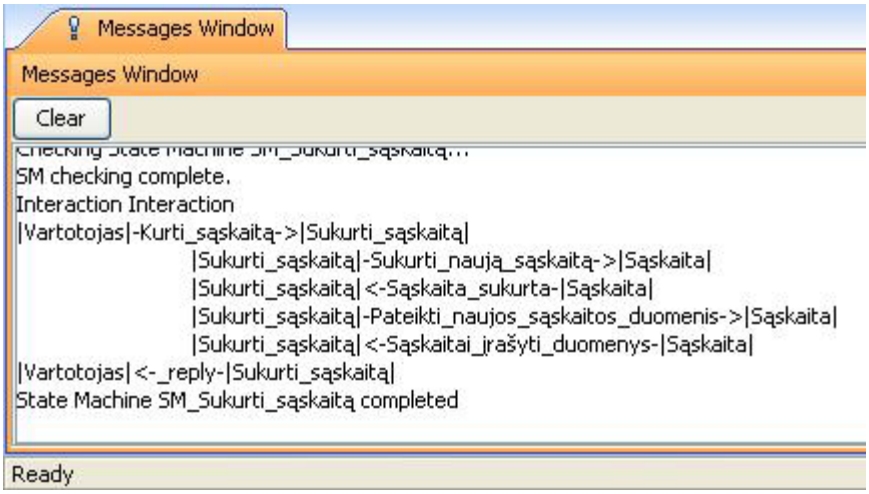
Gautoje būsenų diagramoje kiekvienas perėjimas yra susietas su gautais ir siunčiamais įvykiais, o kiekvienas įvykis – su siuntėju arba gavėju. 37 paveikslėlyje pavaizduota su įvykio susieto stereotipo žymena *Sender*, kurioje nurodytas įvykio siuntėjas *Vartotojas*.



37 pav. *Įvykis ir su juo susieto stereotipo žymena*

Žemiau pateikta iš gautos būsenų diagramos sugeneruota sekų diagrama. Kadangi MagicDraw nėra galimybės atvaizduot sugeneruotos diagramos grafiškai, ji vaizduojama tekstiniame pavidale.

Palyginus gautą sekų diagramą (38 pav.) su pradine (35 pav.) matome, kad jos sutampa. Vadinasi, transformacija atlikta teisingai.



38 pav. Sugeneruota sekų diagrama tekstiniame pavidale

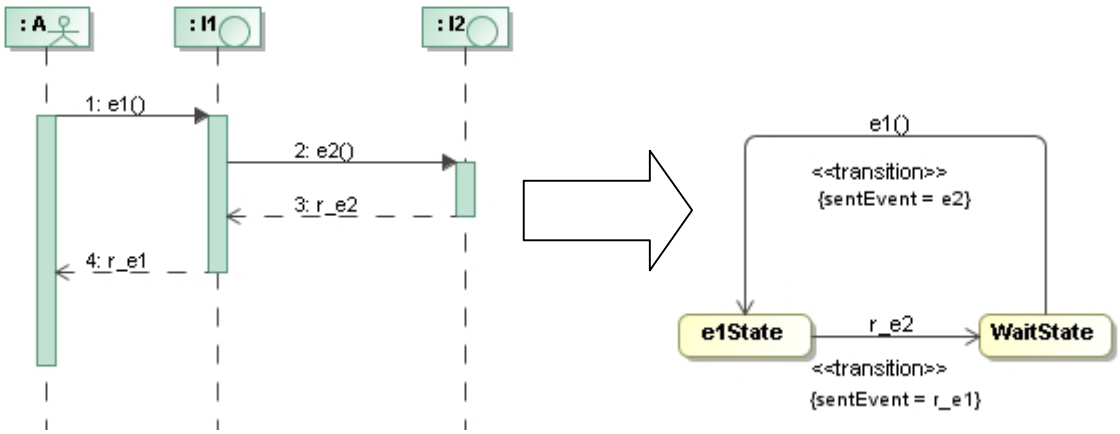
Daugiau tirtų sekų ir būsenų diagramos pavyzdžių pateikta 1 priede.

5.2. Sukurto įskiepio efektyvumo tyrimas

Įrankio efektyvumui tirti buvo nuspręsta sukurti įvairias sekų diagramas ir iš jų pagal algoritmą generuoti būsenų diagramas rankiniu būdu ir naudojant įrankį. Taip pat eksperimentui buvo paimtas konkretus modelis su sekų diagramomis ir atliktos transformacijos į būsenų diagramas.

Efektyvumui tirti buvo papildytas programos vykdymas laiko skaičiavimu. Laikas pradedamas skaičiuoti paspaudus transformacijos mygtuką ir baigiamas, kai transformacija baigta. Rankiniu būdu atliktų transformacijų laikui skaičiuoti buvo naudojamas chronometras. Chronometras matuoja 10ms tikslumu, tuo tarpu programoje laikas matuojamas 1ms tikslumu. Efektyvumui tirti buvo sukurtos sekų diagramos su skirtingais gautų pranešimų skaičiais. Kiekviena sekų diagrama buvo transformuojama ir fiksuojamas jos transformavimo laikas (transformacijos vykdomos vienam konkrečiam objektui).

Elementarus sekų diagramos ir iš jos gautos būsenų diagramos pavyzdys pavaizduotas 39 paveikslėlyje.

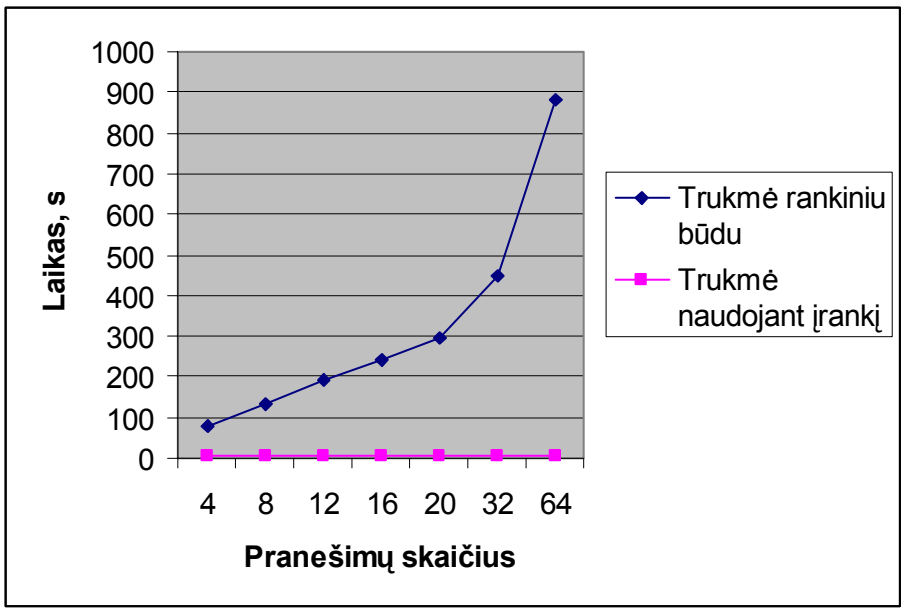


39 pav. Elementarus sekų diagramos ir iš jos gautos būsenos diagramos pavyzdys

Būsenų diagramoje ant perėjimo iš *WaitState* į *e1State* vaizduojamas gautas įvykis *e1()*, kuris susietas su perėjima sužadinančiu trigeriu. <<*transition*>> elementas parodo su kokiu stereotipu yra susietas perėjimas, o skliausteliuose esantis elementas {*sentEvent* = e2} – žymenos *sentEvent* reikšmė *e2* (kitaip tariant siųstas įvykis *e2*).

4 lentelė Transformacijos trukmės rankiniu ir automatiniu būdu palyginimas

Diagrama	Pranešimų skaičius	Trukmė rankiniu būdu		Trukmė automatizavus	Skirtumas kartais
		(min:s)	(s)	(s)	
SD1	4	1:20	80	2,563	31
SD2	8	2:13	133	2,625	51
SD3	12	3:12	192	2,844	68
SD4	16	4:03	243	2,921	83
SD5	20	4:56	296	3,360	88
SD8	32	7:30	450	3,844	120
SD16	64	14:40	880	5,406	165



40 pav. Transformacijos vykdymo trukmės priklausomybės nuo pranešimų skaičiaus sekų diagramoje grafikas

Iš grafiko (40 pav.) matyti, kad rankiniu būdu transformavimo laikas didėja greičiau negu naudojant automatinį transformavimą, todėl galime teigti, kad esant didelėms sekų diagramoms labai pravartu naudoti realizuotą įrankį – projektuotojas sutaupytų daug laiko.

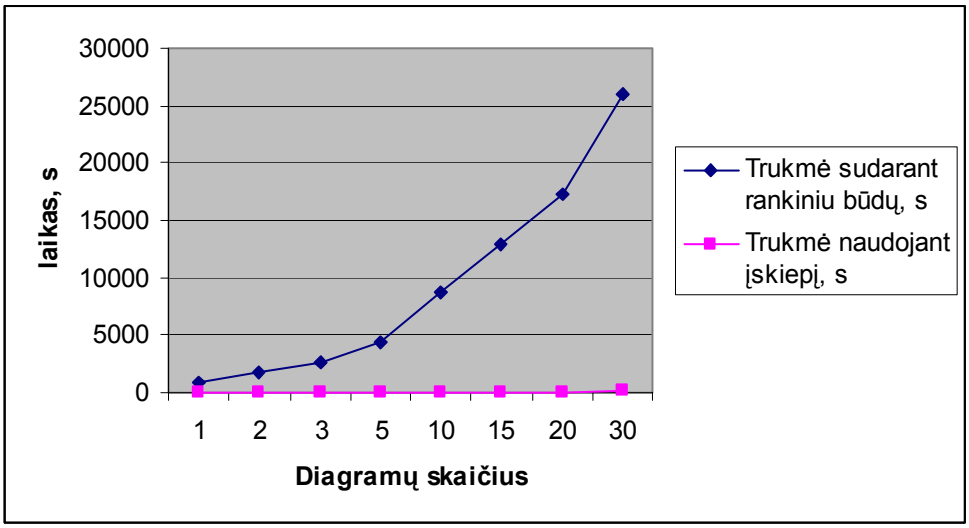
Žemiau pateikta lentelė, kur nagrinėjami skirtingi sekų diagramų kiekiai, pagal kurias reikia atlikti transformaciją į būsenų diagramą konkrečiam objektui, kiekvienoj diagramoj po

64 pranešimus. Rankiniu būdu transformacijos nebuvo vykdomos, jos buvo apskaičiuotos remiantis duomenimis pateiktais aukščiau (4 lentelė).

5 lentelėje pateiktas palyginimas, kai diagramų skaičius yra didelis. Jeigu būtų 30 diagramų, naudojantis įskiepiu transformaciją galima atlikti beveik 200 kartų greičiau negu rankiniu būdu.

5 lentelė Transformacijos trukmės palyginimas esant dideliam diagramų skaičiui

Diagramų skaičius	Trukmė rankiniu būdu		Trukmė automatizavus	Skirtumas kartais
	(h:min:s)	(s)	(s)	
1	0:14:52	892	5,0	177
2	0:29:16	1756	7,4	237
3	0:43:40	2620	8,0	328
5	1:12:28	4348	11,0	394
10	2:24:28	8668	17,7	489
15	3:36:28	12988	26,7	487
20	4:48:28	17308	60,8	285
30	7:12:28	25948	132,5	196



41 pav. Trukmės palyginimo grafikas esant dideliam diagramų skaičiui.

Tyrimo rezultatai rodo, kad rankiniu būdu transformuojant diagramas užtrunkama žymiai ilgiau, negu transformuojant naudojant sukurtą įskiepį. Vadinasi, projektuotojas norėdamas sukurti suderintą sistemą, be įskiepio pagalbos užtruktų žymiai ilgiau.

Jeigu projektuotojas nenaudotų transformavimo algoritmo, sukurtos būsenų mašinos gali būti netikslios, likti nesusietos (per įvykius) su sekų diagramomis. Be to buvo imami

tokie pavyzdžiai, kur transformavimo algoritmo pritaikymas praktiškai nesiskiria nuo būsenų diagramų sudarymo netaikant algoritmo.

Sugeneruotų būsenų diagramų pavyzdžiai ir sekų diagramos pateiktos 1 priede „DIM modelio pavyzdys“.

6. IŠVADOS

1. MDA tikslas yra automatizuoti programų sistemų kodo generavimą. Magistriniame darbe nagrinėjama ankstesnė sistemos gyvavimo ciklo stadija – reikalavimų modelio suderinamumo tikrinimas, modelį aprašančių sekų ir būsenų diagramų tarpusavio transformacijos.

2. Sukurta sekų ir būsenų diagramų abipusio transformavimo programinė realizacija MagicDraw įskiepio pavidalu. Šios programinės realizacijos sukūrimas rodo, kad galima automatizuoti ir taip palengvinti bei paspartinti informacinių sistemų projektavimą, programiškai įgyvendinant sekų ir būsenų diagramų tarpusavio transformacijas. Naudojant transformacijas išvengiama nesuderinamumų modelyje.

3. Sekų ir būsenų diagramų abipusis transformavimas atliekamas naudojant tam tikrą algoritmą. Kad būtų galima transformacija iš būsenų diagramos, buvo papildytas UML naudojamų būsenų mašinų metamodelis.

4. Diagramų suderinimas reikalingas, kad būtų gaunamas kuo tikslesnis modelis, o iš tikslaus modelio galima generuoti programos kodą.

5. Abipusių transformacijų nėra realizuotų esamuose CASE įrankiuose. Nagrinėtuose CASE įrankiuose nebuvo galimybės transformuoti sekų diagramos į būsenų. Įrankiuose galima naudoti MDA transformacijas – transformaciją tarp modelių ir kodo generavimą.

6. Eksperimentai parodė, kad automatinis sekų ir būsenų diagramų transformavimo būdas sumažina diagramai sukurti reikalingo laiko sąnaudas ir pagerina modelio kokybę.

7. Programinę realizaciją galima naudoti tolesniuose eksperimentuose bei praplėsti esamų CASE įrankių galimybes. Pateikta sukurto produkto praplėtimo galimybė (4.8 skyrelyje).

8. Šio darbo pagrindu buvo paruoštas straipsnis ir skaitytas pranešimas 12-ojoje doktorantų ir magistrantų konferencijoje „Informacinė visuomenė ir universitetinės studijos“.

LITERATŪRA

[1] F.Bordeleau, J.-P.Corriveau. From Scenarios to Hierarchical State Machines: A Pattern-Based Approuch. Iš: OOPSLA 2000 Workshop: Scenario-based round-trip engineering, Tampere University of Technology, Software Systems Laboratory, Report 20, 2000, p.13-18.

[2] L.Ceponiene, L.Nemuraite. 2004. Design Independent Modeling of Information System Using UML and OCL. Sixth International Baltic Conference on Data Bases and Information Systems (DB&IS'2004), Vol. 672: 357-372.

[3] L.Ceponiene, L.Nemuraite. Reconcilation of UML Models for Development of Information Systems. Rigas Tehniskas Universitates Zinatniskie Raksti. Ser. 5. Datorzinatne. ISSN 1407-7493. 2005, sejums 22. p. 9-17.

[4] L.Ceponiene, L.Nemuraite. Transformation of UML Diagrams for Reconcilation of Requirements. Information Systems Development: Advances in Theory, Practice, and Education: proceedings of 13th international conference, Vilnius, Lithuania, September 9-11, 2004. Vol. 28. New York: Springer Science+Business Media., 2005. ISBN 0-387-25026-3. p. 289-301.

[5] L.Čeponienė, L.Nemuraitė. UML klasių, būsenų ir sekos diagramų suderinimas. Informacinės technologijos '2003. ISBN 9955-09-335-8, Kaunas, Technologija, 2003, Sekcija XIV p.62-67.

[6] L.Čeponienė. Informacijos sistemų reikalavimų analizės ir transformavimo į projektą metodas. Daktaro disertacija.

[7] Enterprise Architect – UML design tools. [Žiūrėta 2007 03 15], prieiga internete <http://www.sparxsystems.com.au/products/ea.html>.

[8] MagicDraw product feature overview. [Žiūrėta 2007 03 15], prieiga internete <http://www.magicdraw.com/main.php?ts=navig&NMSESSID=706771c1badd1c157bb2eaff1b07a8be&cmd_show=1&menu=feature_list&NMSESSID=706771c1badd1c157bb2eaff1b07a8be>.

[9] MagicDraw OpenAPI User's Guide. NoMagic, Inc, 2006. [Žiūrėta 2005 10 15], prieiga internete <http://www.magicdraw.com/files/manuals/12.5/MagicDraw%20OpenAPI%20UserGuide.pdf?NMSESSID=469bba9a3c2fadebc22b3d0490befba6>.

[10] E.Makinen, T.Systa. An Interactive Approach for Synthesizing UML Statechart Diagrams from Sequence Diagrams. Iš: OOPSLA 2000 Workshop: Scenario-based round-trip engineering, Tampere University of Technology, Software Systems Laboratory, Report 20, 2000, p. 7-12.

[11] OMG 2003. MDA Guide Version 1.0.1. [Žiūrėta 2005 10 15], prieiga internete <http://www.omg.org/docs/omg/03-06-01.pdf>.

[12] T.Pender. UML Bible. ISBN:0764526049. John Wiley & Sons, 2003, 940p.

[13] T.Systa, R.K.Keller, K.Koskimies. Summary Report of the OOPSLA 2000 Workshop on Scenario-Based Round-Trip Engineering. OOPSLA 2000 Workshop: Scenario-based round-trip engineering, Tampere University of Technology, Software Systems Laboratory, 2000.

[14] UModel-UML 2.1 software modeling and application development tool. [Žiūrėta 2007 04 11], prieiga internete <http://www.altova.com/products/umodel/uml_tool.html>.

[15] Unified Modeling Language: OCL Version 2.0, 2003 OMG document ptc/03-08-08 <http://www.omg.org/docs/ptc/03-08-08.pdf>.

[16] Unified Modeling Language. Superstructure Specification. Version 2.0. OMG document formal/05-07-04, 2005. [Žiūrėta 2005 10 11], prieiga internete <http://www.omg.org/cgi-bin/doc?formal/05-07-04>

[17] S.Vasilache, J. Tanaka. Synthesis of State Machines from Multiple Interrelated Scenarios Using Dependency Diagrams. Journal of Systemics, Cybernetics and Informatics, Vol.3, No.3, 2006, pp. 18-25. [Žiūrėta 2007 02 25], prieiga internete <http://www.iplab.cs.tsukuba.ac.jp/paper/journal/si_jsci2006.pdf>.

[18] Visual Paradigm for UML. [Žiūrėta 2007 02 30], prieiga internete <http://www.visual-paradigm.com/news/vpsuite30sp2/vpuml60sp2.jsp>.

[19] What is MagicDraw? No Magic Inc. [Žiūrėta 2006 04 20], prieiga internete <http://www.magicdraw.com/main.php?ts=navig&NMSESSID=4db1e1632f98b380cbbd74cce71ff725&cmd_show=1&menu=what_is>.

[20] J.Whittle, J.Schumann. Generating Statechart Designs From Scenarios. International Conference on Software Engineering, Ireland, 2000, pp.314-323.

SANTRUMPŲ IR TERMINŲ ŽODYNAS

CASE įrankis – įrankis, skirtas programinės įrangos kūrimo automatizavimui (angl. *Computer-Aided Software Engineering*).

DIM – nuo projektavimo metodo nepriklausantis reikalavimų modelis (angl. *Design Independent Model*).

IS – informacijos sistema (angl. *Information System*).

JMI – java kalbos meta duomenų sąsaja (angl. *Java Metadata Interface*).

MAS –būsenų diagramų sintetinimo algoritmas (ang. *Minimally Adequate Synthesizer*).

MDA – modeliu paremta architektūra (angl. *Model Driven Architecture*).

OCL – objekto apribojimų kalba (angl. *Object Constraint Language*).

OMG – Objektų valdymo grupė (angl. *Object Management group*).

PIM – nuo platformos nepriklausomas modelis (angl. *Platform Indemependent Model*).

PSM – platformai specifinis modelis (angl. *Platform Specific Model*).

UCM – panaudos atvejų planas (angl. *Use Case Map*).

UML – unifikuota modeliavimo kalba (angl. *Unified Modeling Language*).

XMI – formatas, skirtas keitimuisi UML modeliais tarp CASE įrankių, pagrįstų XML (angl. *XML Metadata Interchange*).

XML – bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba (angl. *Extensible Markup Language*).

MagicDraw UML tool extension for reconciliation of sequence diagrams and state machines

SUMMARY

In this master thesis the transformation from sequence diagrams to statemachines and vice versa is presented.

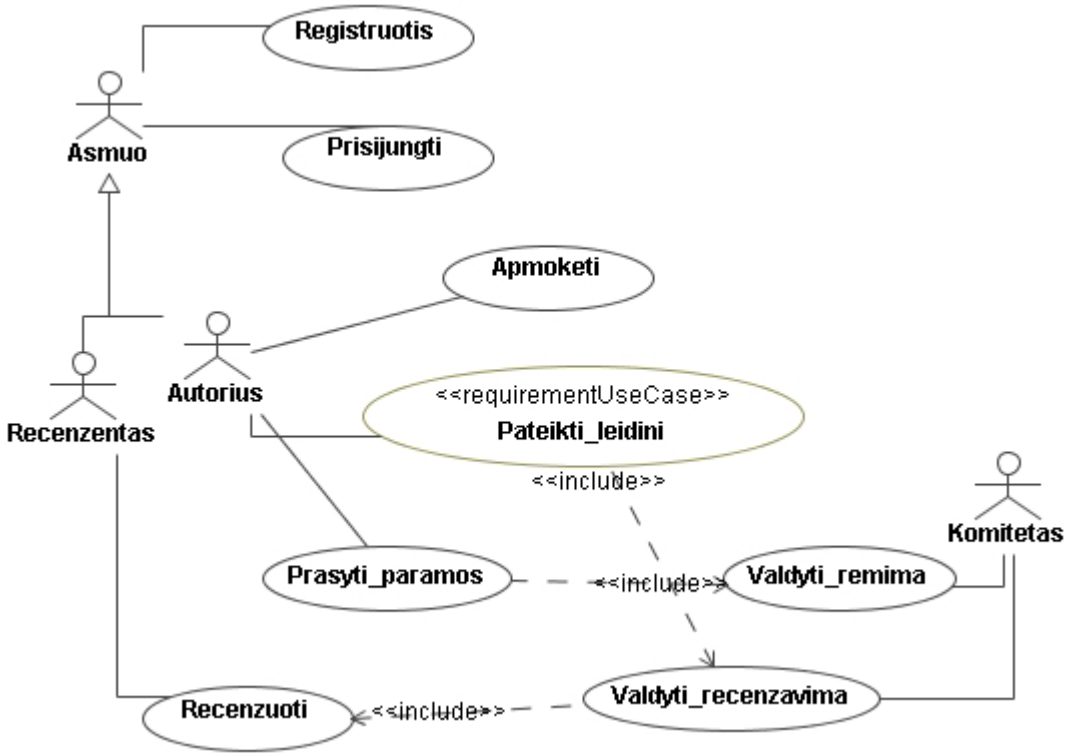
The first section describes a research of four existing algorithms of generating state machines from sequence diagrams. For diagrams transformation plug-in for CASE tool MagicDraw is created according MDA standards. Transformation plug-in takes sequence (state) diagram model as input and generates state (sequence) diagrams according to transformation sules. Created plug-in requirements,functional specification and architecture described in Project section.

The investigation section describes investigation of the developed plug-in. In this section were investigate the working efficiency of designer trying to reconcile model diagrams.

1 PRIEDAS. SEKŲ IR BŪSENŲ DIAGRAMŲ TRANSFORMACIJOS PAVYZDYS

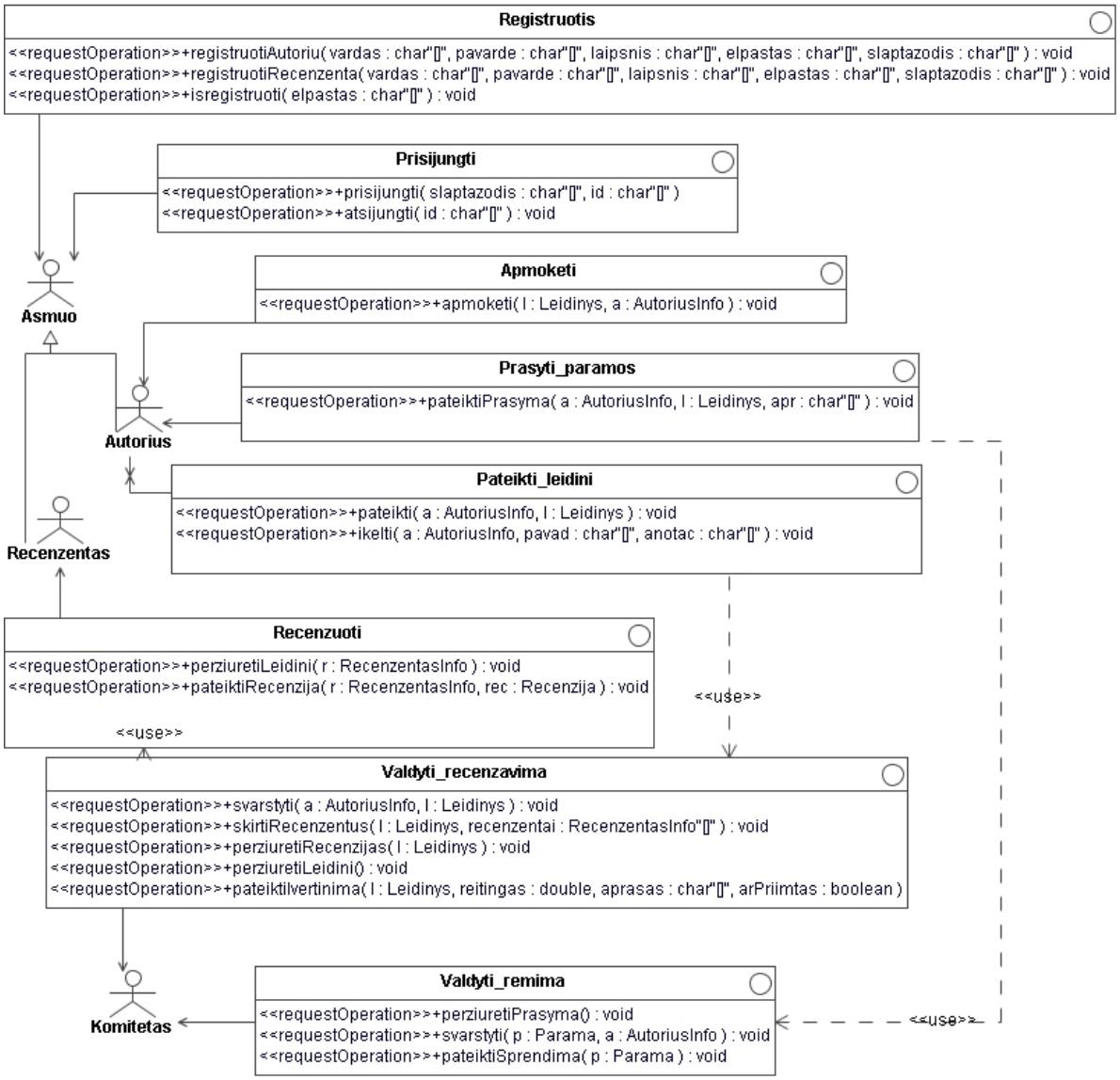
Paimtas IS katedroje nagrinėjamas DIM modelio pavyzdys, šiame priede pateikta pavyzdžio dalis – panaudojimo atvejai, interfeisai ir interfeisų veiklos, išreikštos sekų diagramomis. Taip pat pateikiamos būsenų diagramos, sugeneruotos naudojant sukurtą produktą.

Panaudojimo atvejų diagrama:



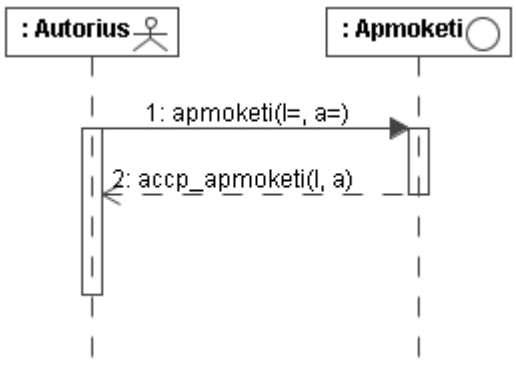
42 pav. Panaudojimo atvejų diagrama

Interfeisų diagrama

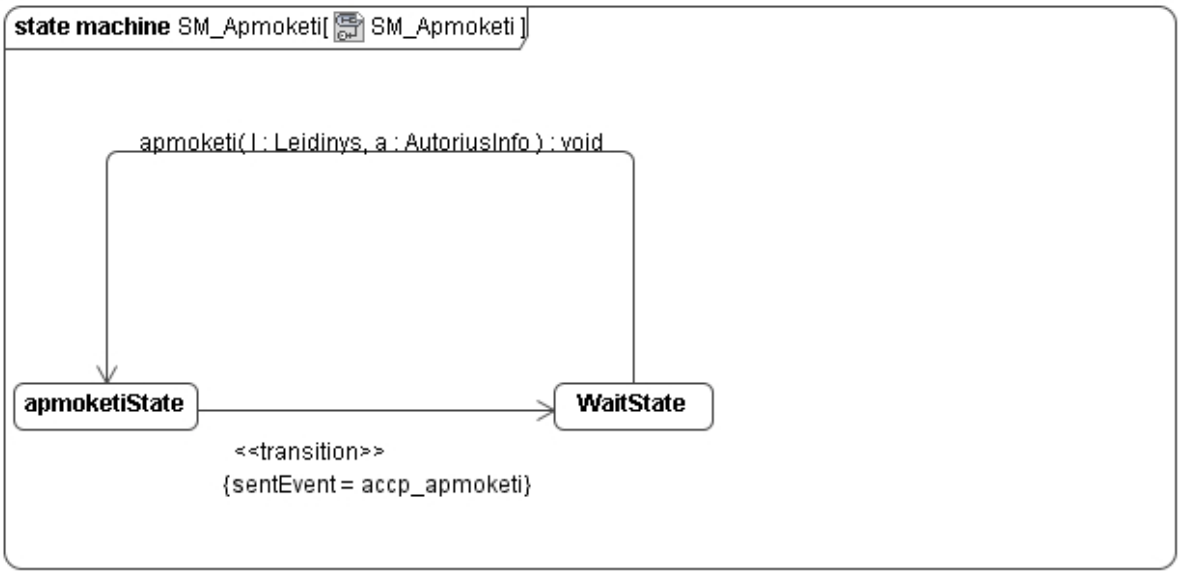


43 pav. Sistemoje naudojamų interfeisų diagrama

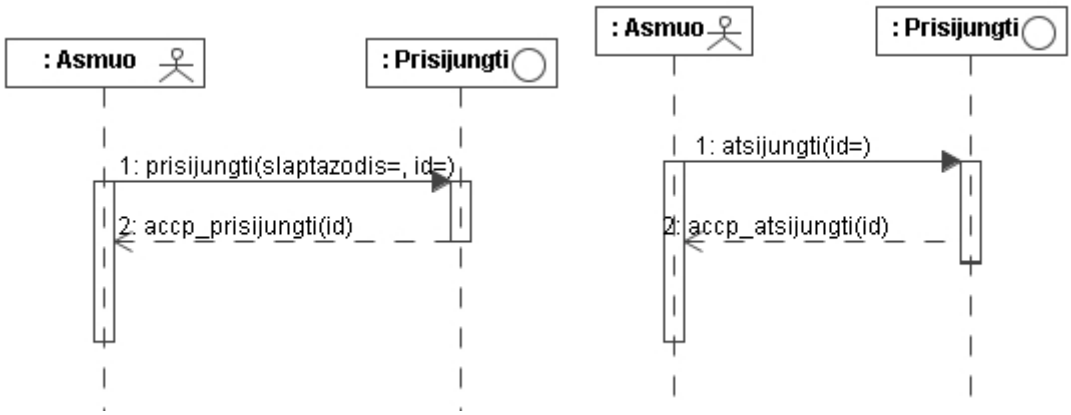
Sekų diagramos ir iš jų sugeneruotos būsenų mašinos



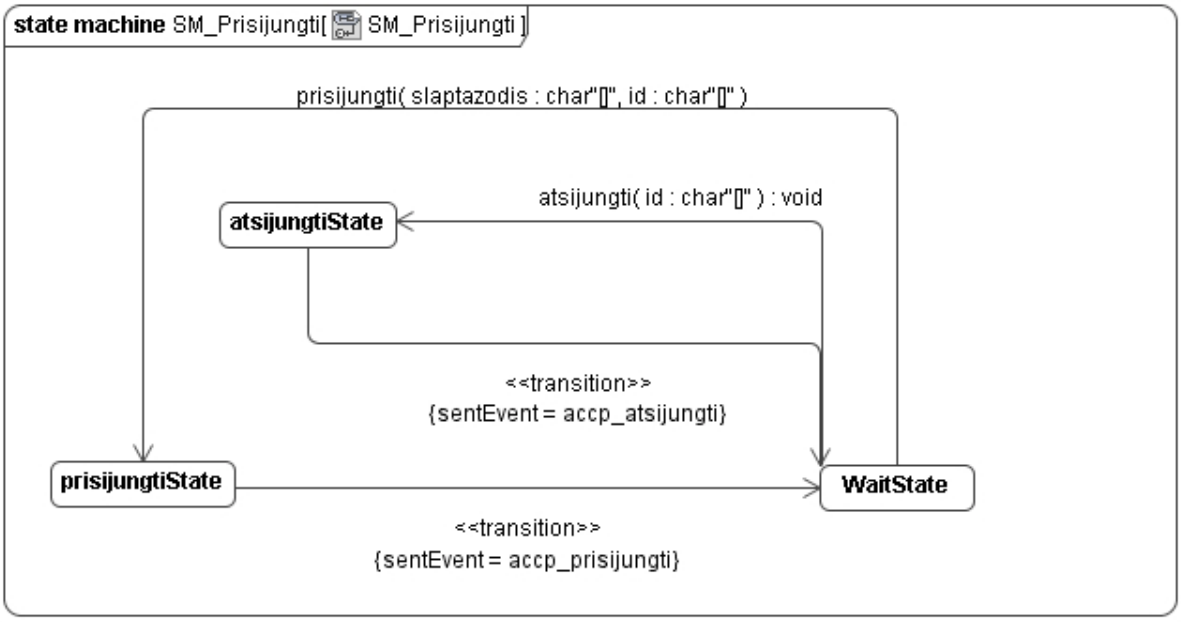
44 pav. Sekų diagrama panaudos atvejui „Apmokėti“



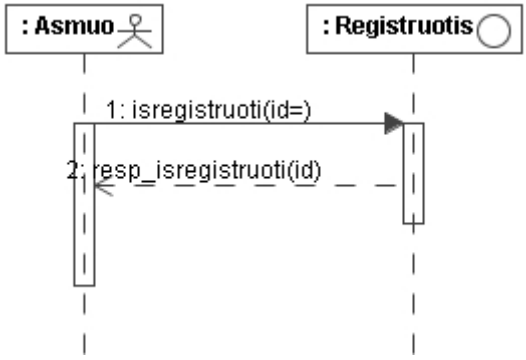
45 pav. Sugeneruota būsenų mašina interfeisui „Apmokėti“



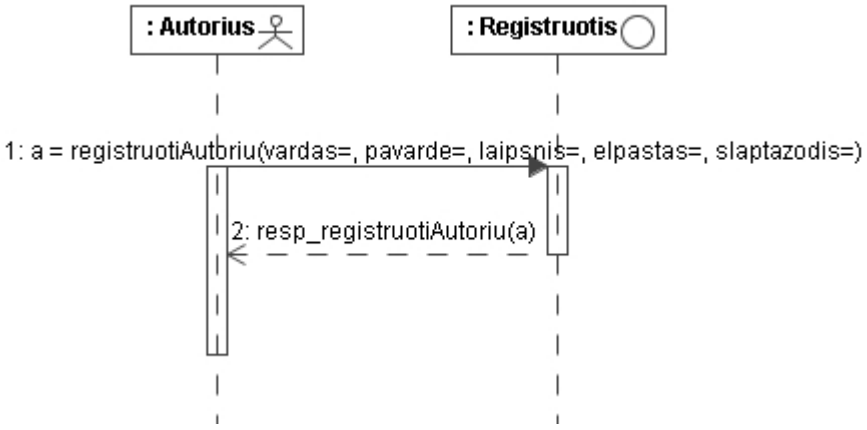
46 pav. Sekų diagramos panaudojimo atvejui „Prisijungti“



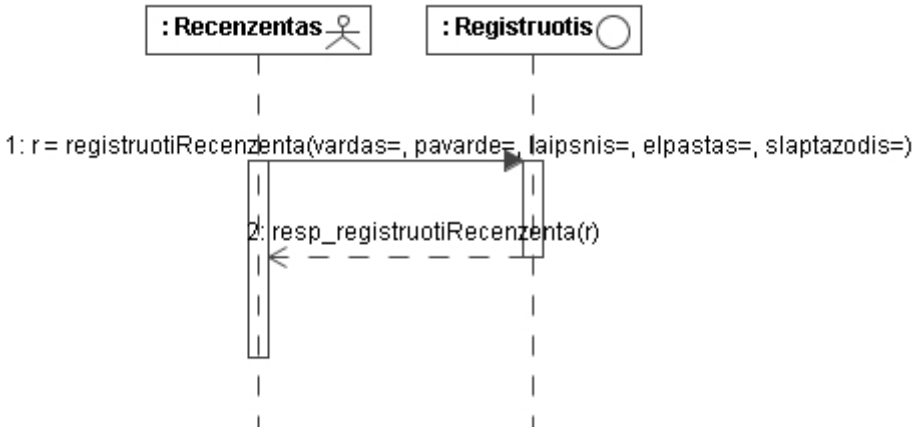
47 pav. Būsenų mašina interfeisui „Prisijungti“



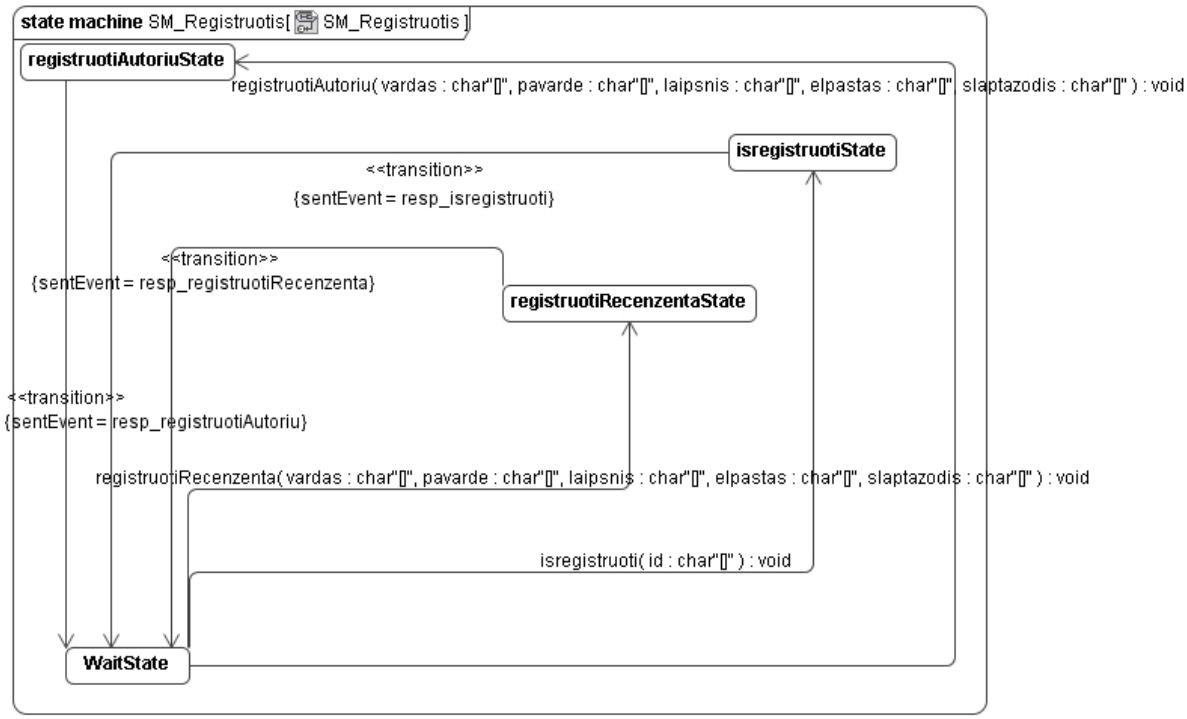
48 pav. Sekos diagrama panaudos atvejui „Registruotis“



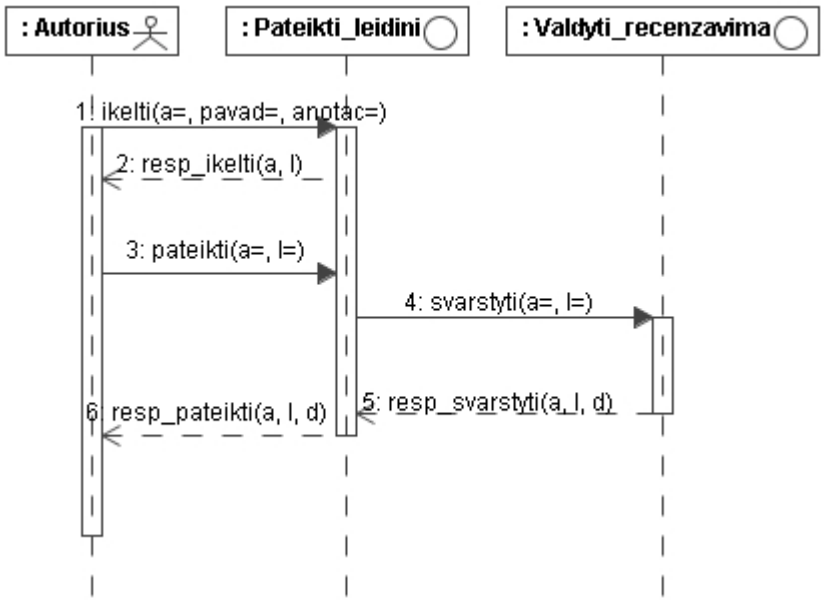
49 pav. Sekos diagrama panaudos atvejui „Registruotis“



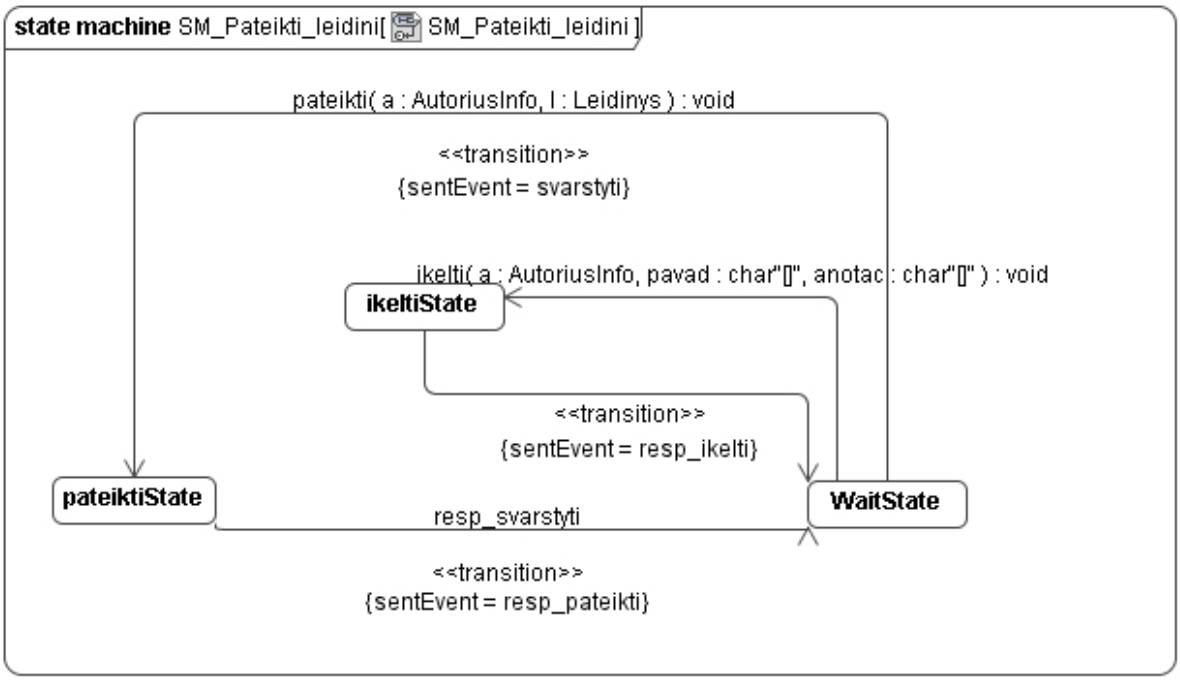
50 pav. Sekos diagrama panaudos atvejui „Registruotis“



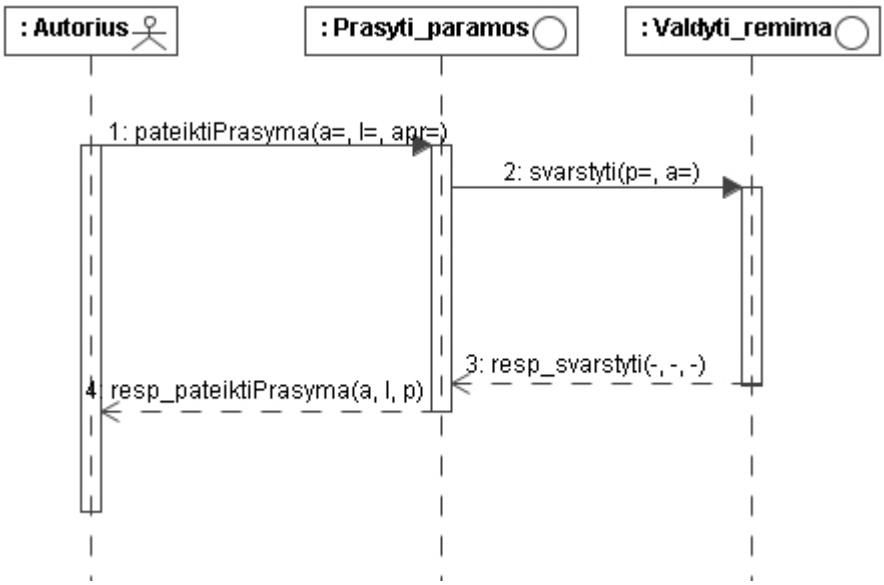
51 pav. Sugeneruota būsenų mašina interfeisui „Registruotis“



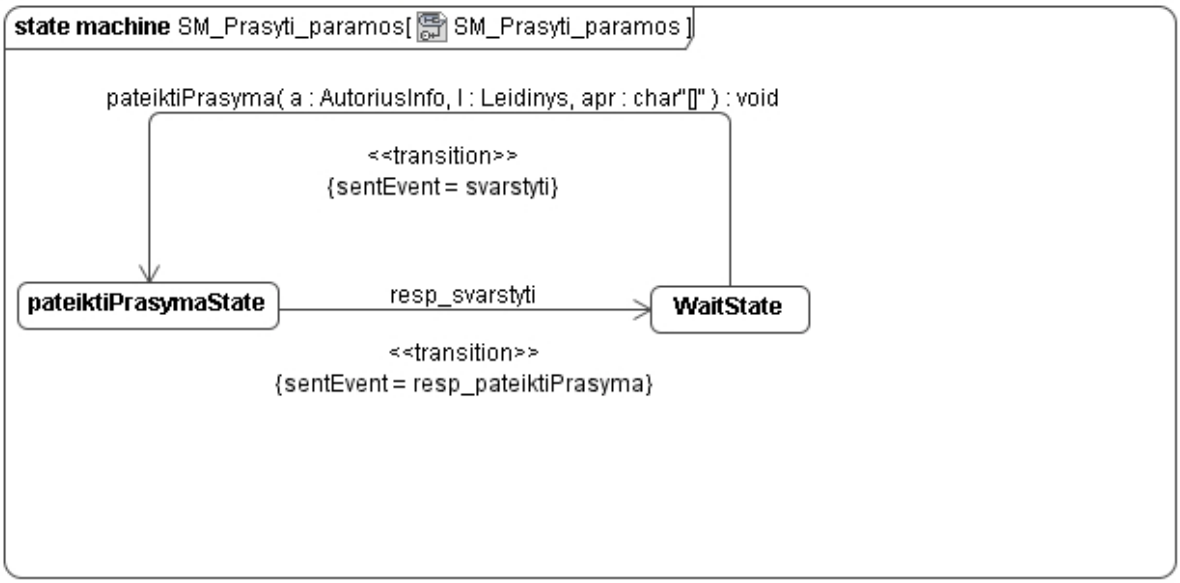
52 pav. Sekų diagrama panaudos atvejui „Pateikti leidinį“



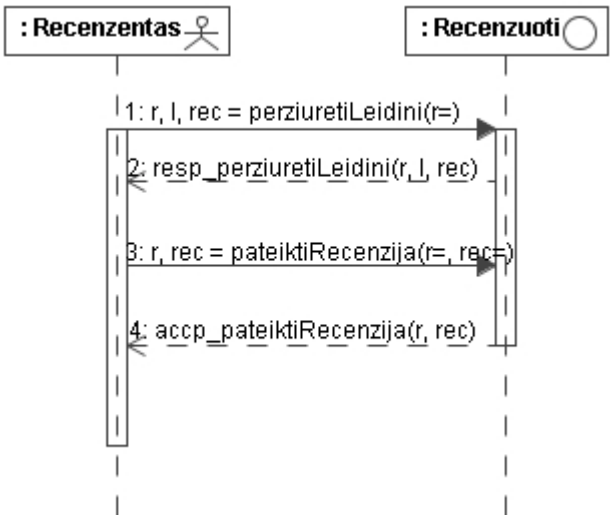
53 pav. Sugeneruota būsenų mašina interfeisui „Pateikti leidinį“



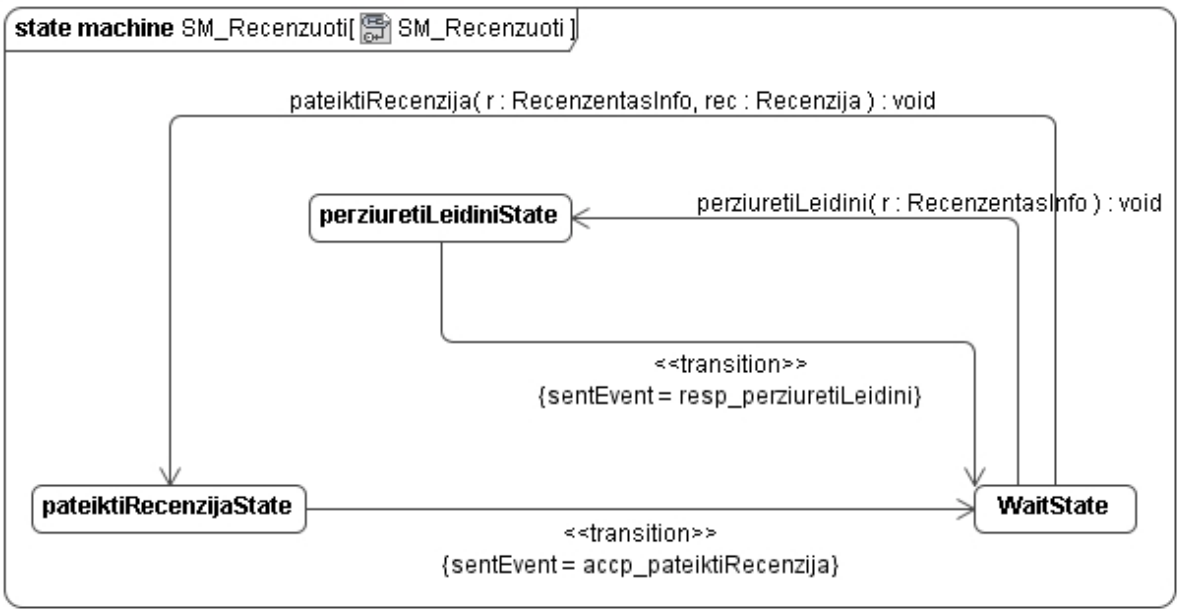
54 pav. Sekų diagrama panaudos atvejui „Prašyti paramos“



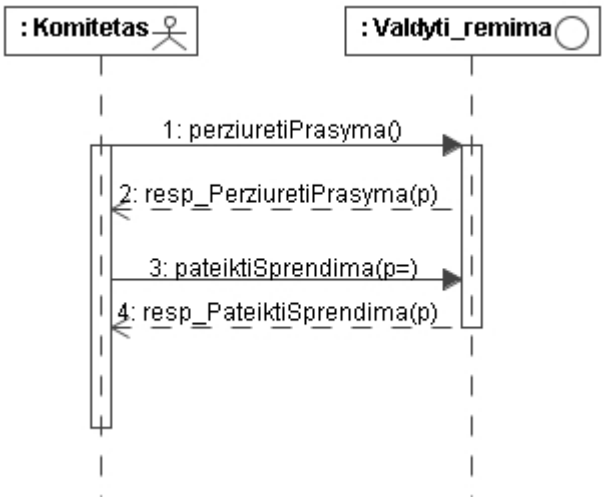
55 pav. Būsenų mašina interfeisui „Prašyti paramos“



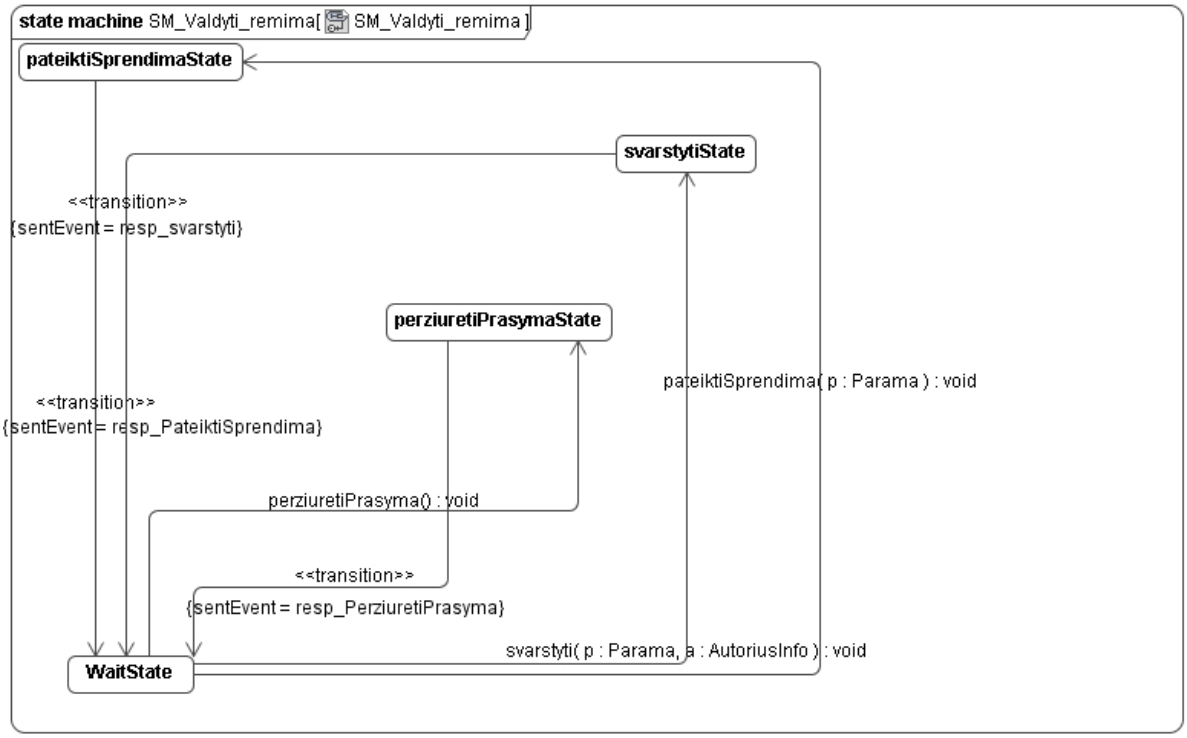
56 pav. Sekų diagrama panaudos atvejui „Recenzuoti“



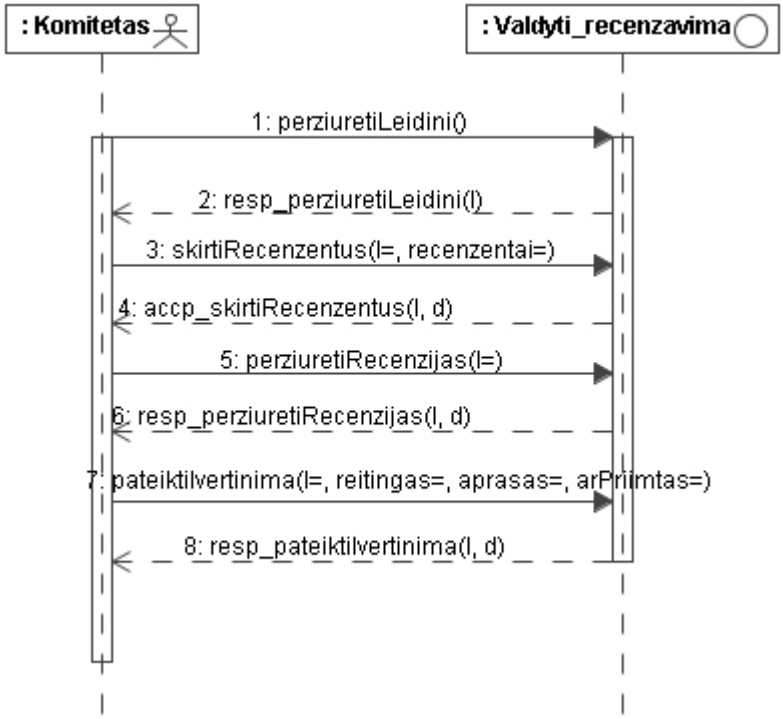
57 pav. Sugeneruota būsenų mašina interfeisui „Recenzuoti“



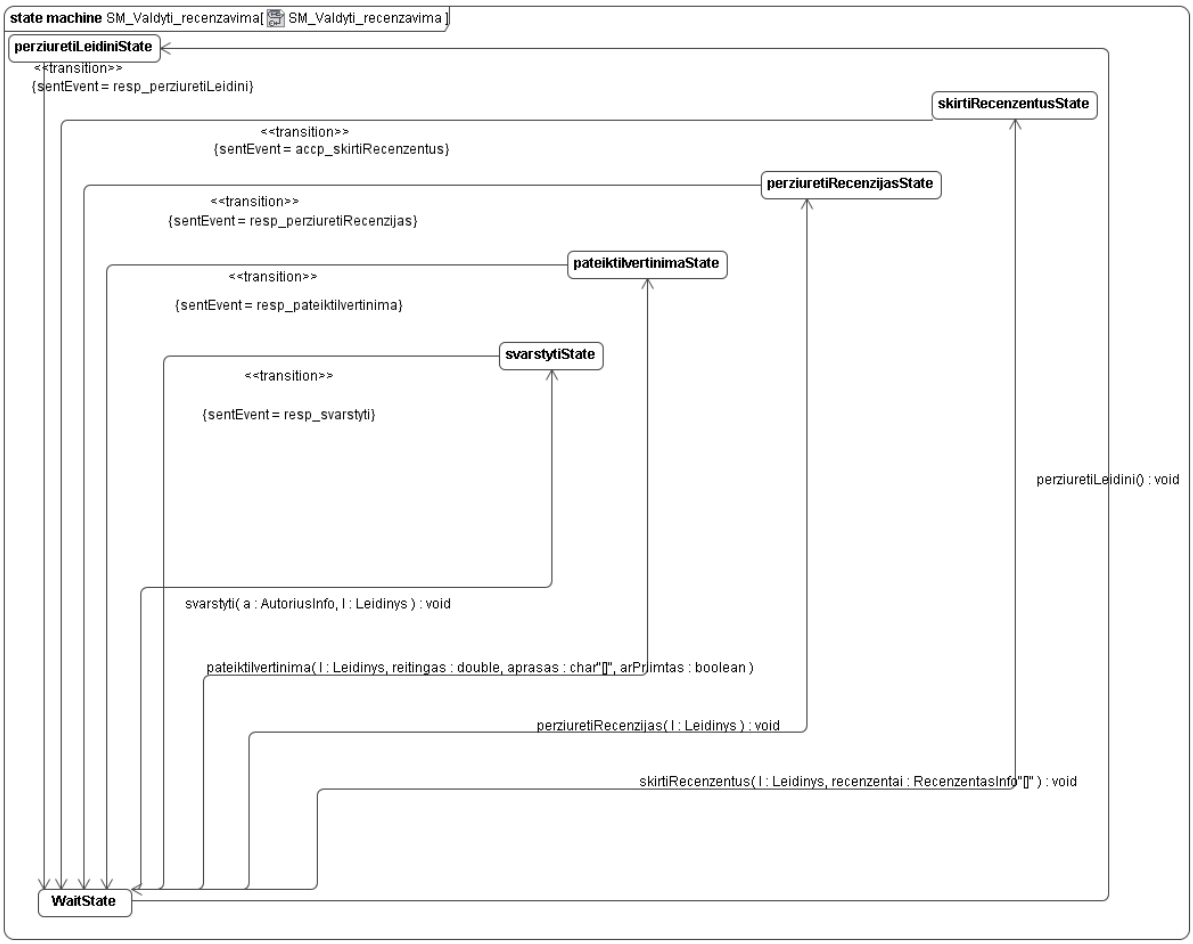
58 pav. Sekų diagrama panaudos atvejui „Valdyti rėmimą“



59 pav. Sugeneruota būsenų mašina interfeisui „Valdyti rėmimą“



Sekų diagrama panaudos atvejų „Valdyti recenzavimą“



Sugeneruota būsenų mašina interfeisui „Valdyti recenzavimą“

2 PRIEDAS. KONFERENCIJOS „INFORMACINĖ VISUOMENĖ IR UNIVERSITETINĖS STUDIJOS“ STRAIPSNIS

MAGIC DRAW ĮRANKIO IŠPLĖTIMAS SEKŲ IR BŪSENŲ DIAGRAMŲ DERINIMO GALIMYBE

Lina Kelmaitė, Lina Čeponienė, Lina Nemuraitė
Kauno technologijos universitetas
Informacijos sistemų katedra

Straipsnyje pateikiamas nuo platformos nepriklausomo modelio korektiškumo tikrinimas, kuris paremtas UML būsenų ir sekos diagramų suderinimu. Šiam tikslui sukurtas Magic Draw UML įrankio įskiepis, kuris leidžia projektuotojui susieti būsenų mašinų ir sekų diagramų rinkinius, patikrinti jų tarpusavio suderinamumą, atvaizduoti sekų diagramų aibes į būsenų diagramų aibes ir atvirkščiai pagal tam tikrą algoritmą. Sukurtas produktas padidina projektuotojų darbo našumą ir užtikrina geresnę modelių kokybę.

1. Įžanga

Taip sparčiai tobulėjant informacinėms technologijoms ir didėjant kuriamų sistemų sudėtingumui bei skaičiui, atsirado poreikis kuo daugiau automatizuoti IS kūrimo procesą. Automatizavimo problemai spręsti naudojami modeliais grindžiamo kūrimo (angl. *Model Driven Development*) metodai, kurių esmė – didesnis naudojamų abstrakcijų lygis, kai vietoj programavimo kalbų naudojamos modeliavimo kalbos (UML).

Modeliais grindžiamo kūrimo metoduose pagrindinis dėmesys sutelktas į paskutinius kūrimo proceso etapus. Organizacijos Object Management Group (OMG) sukurta Model Driven Architecture (MDA) [7] sistemų kūrimo technologija koncentruojasi į nuo platformos nepriklausomų modelių PIM (angl. *Platform Independent Model*) transformavimą į konkrečių platformų modelius PSM (angl. *Platform Specific Model*) ir pastarųjų transformavimą į programos kodą. Pradiniai kūrimo proceso etapai – reikalavimai ir jų analizė lieka nepaliesti, projektuotojas lieka atsakingas už PIM sudarymą, korektiškumą bei suderinimą. Perėjimas nuo reikalavimų prie projektinio modelio gali būti automatizuotas, kas palengvintų projektuotojo darbą.

Reikalavimus galima apibrėžti naudojant panaudos atvejų (angl. *Use Case*) diagramą. Kiekvienam panaudos atvejui galima sudaryti vieną ar keletą alternatyvių įvykių scenarijų. Įvykių scenarijai UML vaizduojami sekų diagramomis (angl. *Sequence Diagrams*) – parodo kokiais pranešimais keičiasi sąveikaujantys objektai. Vieno objekto pilna elgsena gali būti atvaizduojama būsenų mašina (angl. *State Machine*). Šių abiejų tipų diagramos aprašo sistemą skirtingais aspektais (pirmoji – sąveikų vaizdas, antroji – dinaminis vaizdas). Kad būtų užtikrintas modelio vientisumas, šios diagramos turėtų derėti tarpusavyje, todėl būtų naudinga transformuoti sekų diagramas į būsenų ir atvirkščiai.

Šiame straipsnyje aprašoma L. Čeponienės pasiūlytų sekų ir būsenų diagramų abipusio transformavimo algoritmų [3], [4] programinė realizacija. Atlikta programinė realizacija – tai MagicDraw [6] UML CASE įrankio įskiepis.

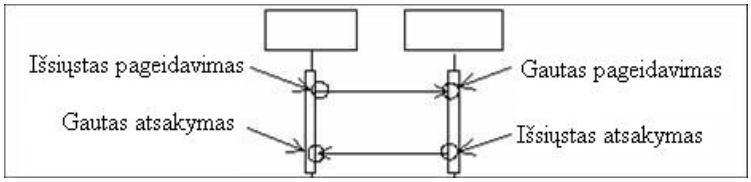
2. Transformavimo aprašymas

Sekų diagramos

Sekos diagramos aprašo sistemos interfeisų sąveiką su išoriniu pasauliu bei sistemos objektų tarpusavio sąveiką. Pagrindiniai sekų diagramos elementai yra gyvavimo linijos ir pranešimai.

Kiekviena gyvavimo linija yra susieta su aktoriumi, interfeisu arba klase. Pranešimai gali būti užklausos (*call*) ir atsako (*reply*), o įvykiai, susieti su pranešimais, gali būti gauti (*recieved*) arba išsiųsti (*sent*).

1 paveikslėlyje pavaizduoti šie įvykių tipai.



1 pav. Pranešimų tipai

Pageidavimo tipo pranešimas yra susietas su atitinkamo interfeiso/klasės operacija.

Norint atlikt transformavimą, sekų diagrama turėtų tenkinti tokius apribojimus:

- Jei sekos diagramoje yra pranešimas, joje turi būti ir bent viena gyvavimo linija.
- Pranešimo gavėjo ir siuntėjo gyvavimo linijos priklauso tai pačiai sekos diagramai.
- Gyvavimo linija turi būti susieta bent su vienu pranešimu.

- Gyvavimo linija turi būti susieta su objektu (klase, interfeisu ar aktoriumi).
- Pranešimas turi būti susietas su vienu įvykiu.
- Gautas užklausos (*call*) pranešimas turi būti susietas su operacija.

Būsenų diagramos

Būsenų diagrama aprašo dinaminį sistemos aspektą. Būsena suprantama kaip objekto gyvavimo sąlyga ar situacija, kai objektas tenkina tam tikrą sąlygą, vykdo kokį nors veiksmą arba laukia kokio nors įvykio. Būsenų mašiną sudaro įvykiai, interfeisų reakcijos į įvykius ir būsenų sekos.

Nagrinėjamos interfeisų būsenų mašinos skiriasi nuo UML 2.0 būsenų mašinų, noritn atlikt transformaciją, jos turi tenkinti tokius apribojimus:

- laikoma, kad kiekvienas interfeisas būsenų diagramoje (BD) visada turi nustatytą (*default*) būseną vadinamą laukimo būsena (*WaitState*), kurios metu objektas laukia įvykio, kad pereitų į kitą būseną. Tokia būsena turėtų būti viena;
- BD laukimo būsena (WaitState) turi turėti bent vieną išeinantį perėjimą;
- būsenų diagramoje yra informacija apie perėjimo metu siųstų ar gautų iš kitų interfeisų įvykių siuntėjus ir gavėjus (to nėra UML 2.0 būsenų mašinose). Visiems įvykiams būsenų diagramoje taikomas stereotipas su siuntėjo arba gavėjo žymena (*tag*). Šis stereotipas praplečia UML modelį, kad SD transformavimo į BD metu nebūtų prarasta informacija;
- interfeisas yra laukimo būsenoje, iš kurios, gavęs užklausą, pereina į tos užklausos vykdymo būseną. Įvykdęs užklausą, interfeisas vėl grįžta į laukimo būseną.
- jei būsenų diagramoje yra perėjimas, jame turi būti ir bent viena būsena;
- jei perėjimo metu gautas įvykis yra susietas su užklausos tipo pranešimu, tai ir to paties perėjimo metu siunčiamas įvykis turi būti susietas su užklausos tipo pranešimu;
- perėjimas turi būti susietas su dviem būsenom – viena, iš kurios išeina (*source*) ir kita, į kurią ateina (*target*);
- iš laukimo būsenos išeinantis perėjimas privalo turėti trigerį, susietą su gautu įvykiu;
- užklausos įvykiai (*CallEvent*) turi būti susieti su operacijomis;
- perėjimams taikomas stereotipas su siunčiamo įvykio žymena (*tag*). Šis stereotipas praplečia UML metamodelį.

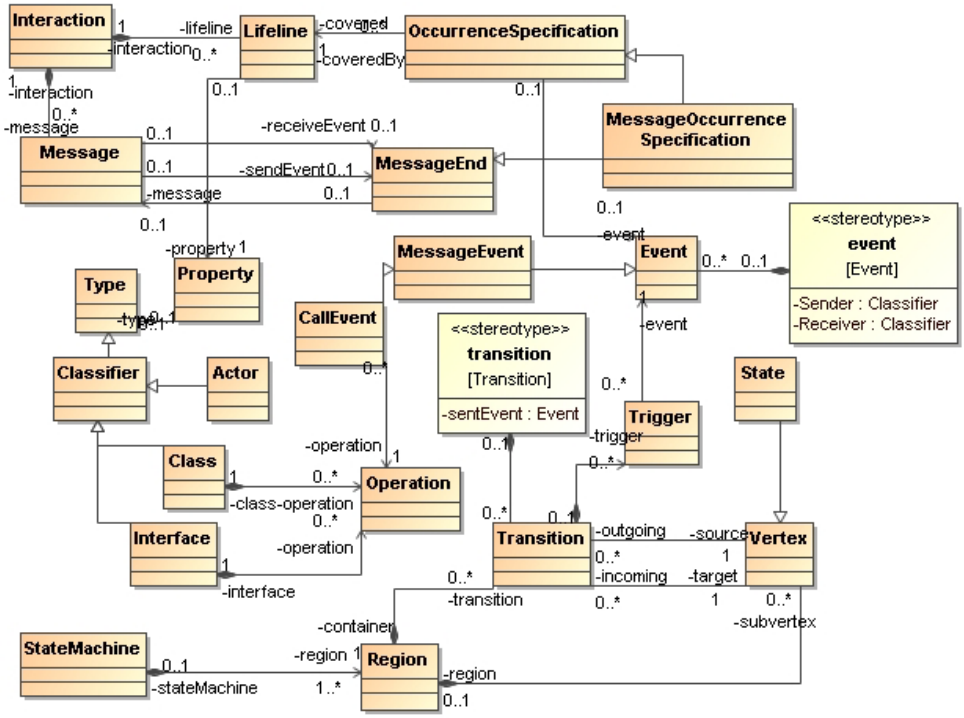
Būsenų mašinose vaizduojama kiekvieno interfeiso sąveikų su kitais interfeisais visuma. Tuo tarpu atskira sekų diagrama vaizduoja vienos sąveikos scenarijų.

Sekų ir būsenų diagramų susiejimo taisyklės

Pagrindinės taisyklės, taikomos atvaizduojant ryšius tarp sekų ir būsenų diagramų:

- interfeiso gyvavimo linijos gautas užklausos įvykis siejamas su perėjimu į naują interfeiso būseną gavėjo būsenų diagramoje;
- siunčiamas atsakas visada susiejamas su perėjimu į laukimo būseną;
- siunčiama užklausa susiejama su perėjimo metu siunčiamu įvykiu;
- gautas atsakas gali būti susietas arba su perėjimu į laukimo būseną (jei sekantis įvykis yra siunčiamas atsakas) arba su perėjimu toje esamoje būsenoje (jei sekantis įvykis yra siunčiama užklausa).

2 paveikslėlyje pateiktame metamodelyje galima pastebėti, kad pagrindinis elementas, siejantis sekų ir būsenų diagramas yra įvykis (klasė *Event*). Įvykis siejasi su pranešimo siuntimu arba gavimu (ryšiai tarp klasių *Message*, *MessageEnd* ir *Event*), taip pat su būsenų diagramos perėjimu (ryšys tarp klasių *Event*, *Trigger* ir *Transition*). Metamodelis papildytas stereotipais – *event*, kuris skirtas su įvykiu susieto siuntėjo arba gavėjo saugojimui bei *transition*, kuris skirtas su perėjimu susieto siunčiamo įvykio saugojimui.



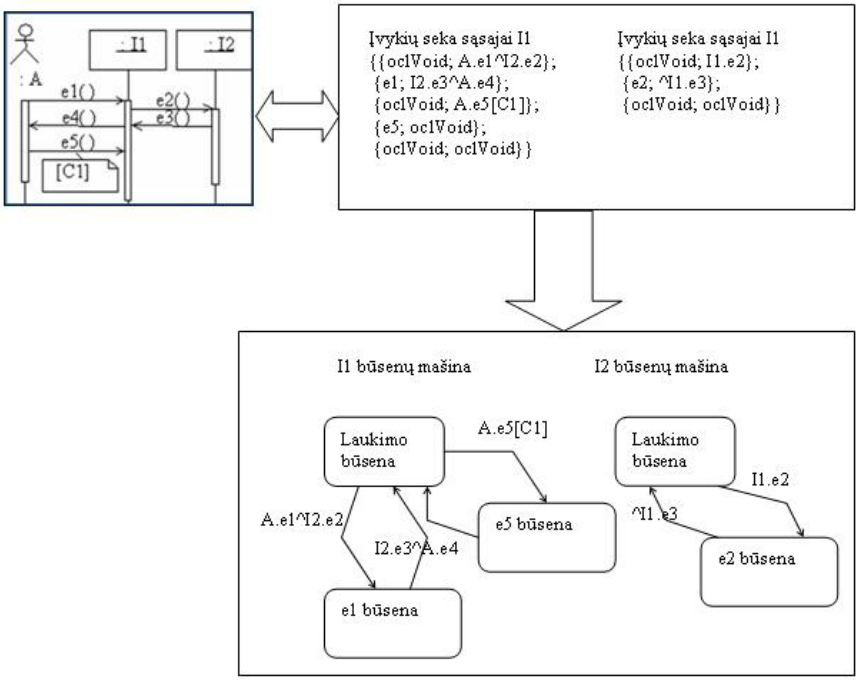
2 pav. Metamodelis, atspindintis sekos ir būsenų diagramų ryšius

Būsenų diagramų generavimas

Būsenų mašinų generavimas iš sekų diagramų susideda iš tokių žingsnių:

1. SD pertvarkoma į kanoninę formą, kur kiekvienas užklausos (*call*) pranešimas papildomas atitinkamu atsakymo (r*eply*) pranešimu,
2. suporojuomi įvykiai kiekvienai gyvavimo linijai, poros pirmas elementas vaizduoja įvykį atitinkantį būseną, o antras – daugiausia du įvykius atitinkančius perėjimą.
3. pagal gautas poras sudaromos būsenų mašinos kiekvienam interfeisui.

3 paveikslėlyje pavaizduota sekų diagrama, iš jos gautos įvykių poros ir suformuotos būsenų diagramos kiekvienam interfeisui.



3 pav. Sekų diagramos transformavimas į būsenų diagramą

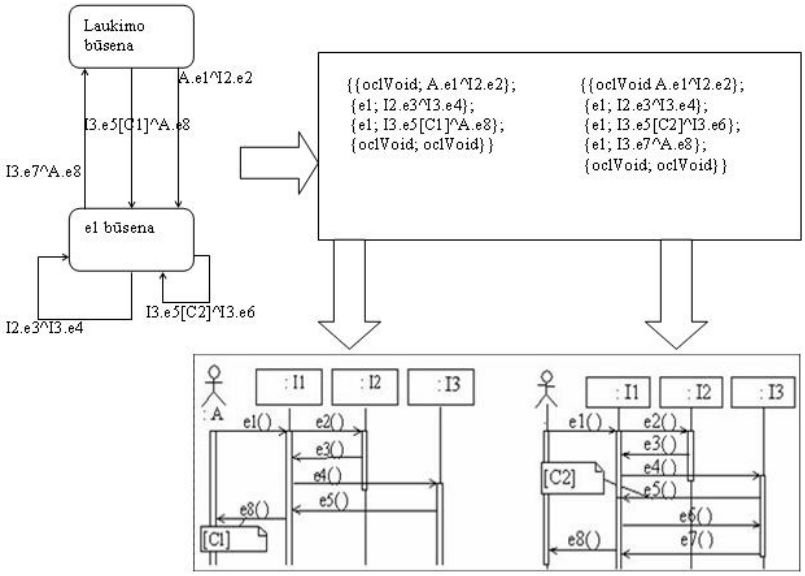
Sekų diagramų generavimas

Sekų diagramų generavimas iš būsenų diagramų susideda iš tokių etapų:

1. būsenų diagrama pertvarkoma į kanoninę formą, diagramoje turi būti įvardinti įvykių siuntėjai ir gavėjai,
2. įvykių poros gaunamos pereinant būsenos diagramoje visus galimus kelius iš laukimo būsenos atgal į laukimo būseną, tuos kelius įsimenant,
3. iš gautų porų sudaromos sekų diagramos pagal tokias taisykles:

– jeigu pirmas poros elementas yra *OclVoid*, antras elementas žymi pageidavimo pranešimą; šio antrojo elemento siunčiamas įvykis žymi pageidavimo pranešimą.
– jeigu pirmas poros elementas ir pirmas sekančios poros elementas nėra *OclVoid*, tada antrasis poros narys žymi atsakymo pranešimą; siunčiamas įvykis žymi pageidavimo pranešimą
– jeigu pirmas poros narys nėra *OclVoid*, tačiau sekančios poros pirmas narys yra *OclVoid*, antrasis elementas žymi atsakymo pranešimą ir siunčiamas įvykis taip pat žymi atsakymo pranešimą.

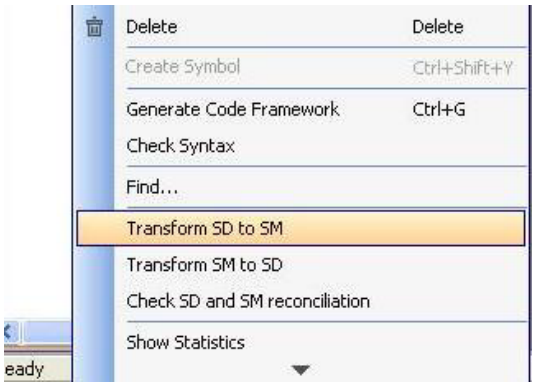
4 paveikslėlyje pavaizduota būsenų diagrama, iš jos gautos įvykių poros ir suformuotos sekų diagramos kiekvienam ciklui.



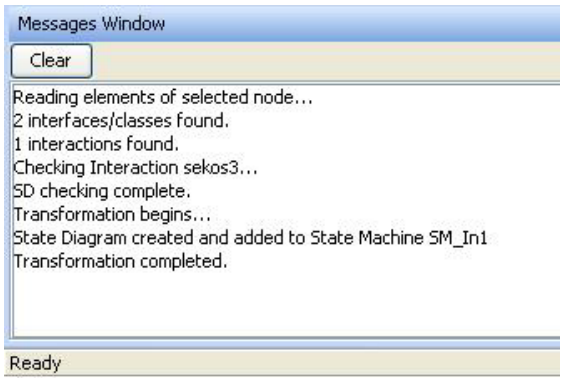
4 pav. Būsenų diagramos transformavimas į sekų diagramas

3. Transformavimo programos funkcionalumas

Sukurta programinė realizacija veikia kaip MagicDraw CASE įrankio įskiepis (angl. *plug-in*). Įskiepis atlieka sekų ir būsenų diagramų korektiškumo patikrinimą bei transformavimą MagicDraw aplinkoje. Norimas transformuoti modelis (paketas, kuriame yra sekų ar būsenų diagramos elementai) pasirenkamas MagicDraw modelio elementų lange. Dešinių pelės klavišu iškviečiamas kontekstinis meniu, kuriuo pažymėtam modeliui galima pasirinkti įskiepio funkcionalumo vykdymą. Kontekstinio meniu (5pav.) naudojimas supaprastina bei pagreitina projektuotojo darbą su įskiepiu, neapkraunant ir be to sudėtingos CASE įrankio aplinkos.



5 pav. Įskiepio valdymas kontekstiniu meniu

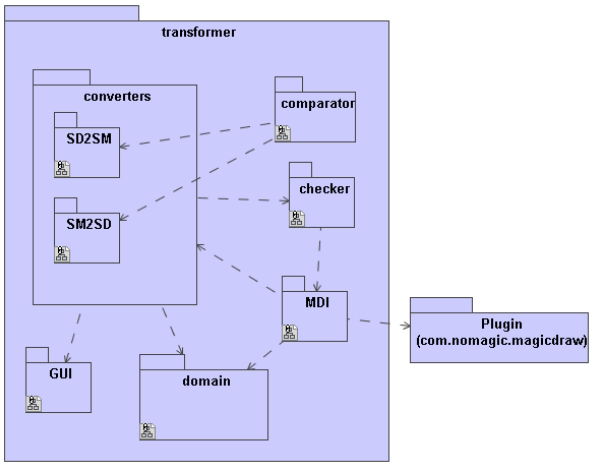


6 pav. MagicDraw pranešimų langas

Prieš transformavimą yra atliekamas sekų ir būsenų diagramų patikrinimas, ar yra tenkinami 2.1 ir 2.2 skyreliuose minėti apribojimai. Jei šie apribojimai netenkinami, transformavimas nevykdomas. Elementų tikrinimo metu rastoms klaidoms bei transformavimo proceso eigos pranešimams išvesti naudojamas MagicDraw įrankio pranešimų langas (6 pav.), kurį galima paslėpti. Pranešimo lango turinį galima išvalyti valymo mygtuku *Clear* arba kopijuoti į laikinąją atmintį (angl. *ClipBoard*).

Iškviesdamas pasirinkto modelio sekų diagramų transformavimo į būsenų diagramas funkciją projektuotojas turi nurodyti kuriam objektui bus generuojama būsenų diagrama, išsirinkus objektą, reikia pasirinkti sekų diagramas, iš kurių bus generuojama būsenų diagrama (pateikiamas sekos diagramų, kuriose dalyvauja pasirinktas objektas, sąrašas). Pasirinkus būsenų diagramų transformavimo į sekų diagramas funkciją, projektuotojas turi išsirinkti būsenų diagramą iš pateikto sąrašo. Taip pat projektuotojas gali patikrinti jau sukurtų sekų diagramų ir būsenų diagramų suderinamumą. Jis turi išsirinkti būsenų diagramą ir sekų diagramas pagal kuriuos bus tikrinamas suderinimas. Jei randama neatitikimų, apie tai pranešame pranešimų lange. Programiškai sukurtų sekų arba būsenų diagramų elementai gali būti tvarkomi standartinėmis MagicDraw priemonėmis kaip ir kiti MagicDraw modelių elementai.

Sistemos architektūrinis modelis pavaizduotas 7 paveikslėlyje. Pakete *checker* yra klasės, skirtos diagramų patikrinimui (ar tenkinami atitinkami apribojimai), pakete *converters* yra transformavimui skirtų algoritmų klasės (*SD2SM* – sekų diagramos transformavimui į būsenų mašinas, *SM2SD* – būsenų mašinų transformavimui į sekų diagramas). *MDI* yra sąsajos su MagicDraw paketas, jame yra klasės skirtos elementų nuskaitymui iš modelio, jų sukūrimui ir atvaizdavimui. *Domain* paketas skirtas sistemoje naudojamų duomenų struktūroms ir tipams saugoti. *Comparator* – sekų ir būsenų diagramų palyginimo paketas. *GUI* – vartotojo sąsajos klasių paketas.

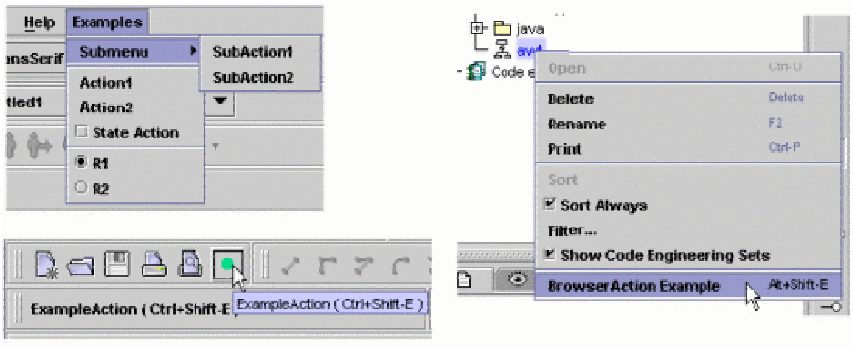


7 pav. Programinės realizacijos architektūra

4. Programinės realizacijos aprašymas

Sukurta programinė realizacija atlikta Java kalba. Šiuo metu Java yra viena populiariausių programavimo kalbų pasaulyje, kurios dėka galima kurti objektines, nepriklausančias nuo platformos, dinamiškai plečiamas programas.

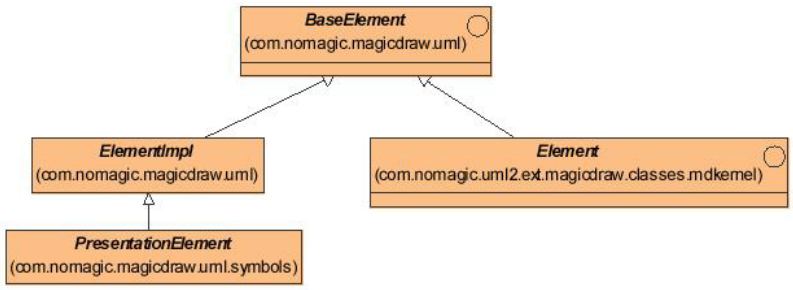
MagicDraw UML įrankis palaiko įskiepius [5] sukurtus Java kalba. Įskiepiai naudojami įrankio funkcionalumui išplėsti. Įskiepis turėtų susidėti iš tokių resursų: katalogas, sukompiliuoti java failai, supakuoti į jar (java archyvinis failas) failą, įskiepį aprašantis failas (XML formatu), papildomi failai, kuriuos naudos įskiepis. MagicDraw įskiepiai realizuoja *com.nomagic.magicdraw.plugins* pakete esančią bendrą sąsają *Plugin*. Šioje sąsajoje aprašomi *init()* ir *close()* metodus, kurie yra iškviečiami MagicDraw įrankio darbo pradžios ir pabaigos metu. Šiuose metoduose aprašomi įskiepio inicializavimui reikalingi veiksmai bei panaudotų resursų atlaisvinimo veiksmai įskiepio darbo pabaigoje.



8 pav. MagicDraw grafinės aplinkos išplėtimas

Įskiepio funkcionalumą galima valdyti įterpiant į pagrindinį arba kontekstinį meniu norimus meniu punktus. Taip pat galima papildyti MagicDraw įrankių juostą savo sukurtu mygtuku (8 pav.). *MDAction* sąsajoje aprašomas *actionPerformed()* metodą, kuris yra iššaukiamas kiekvieną kartą vartotojui pasirinkus atitinkamą menių punktą arba paspaudus mygtuką įrankių juostoje.

Visi MagicDraw įrankyje naudojami elementai [5] priklauso vienam šakniniam UML modeliui, kuris atitinka MOF UML 2.0 modelį. Kiekvienas MagicDraw modelio elementas turi dvi poklases (9 pav.).

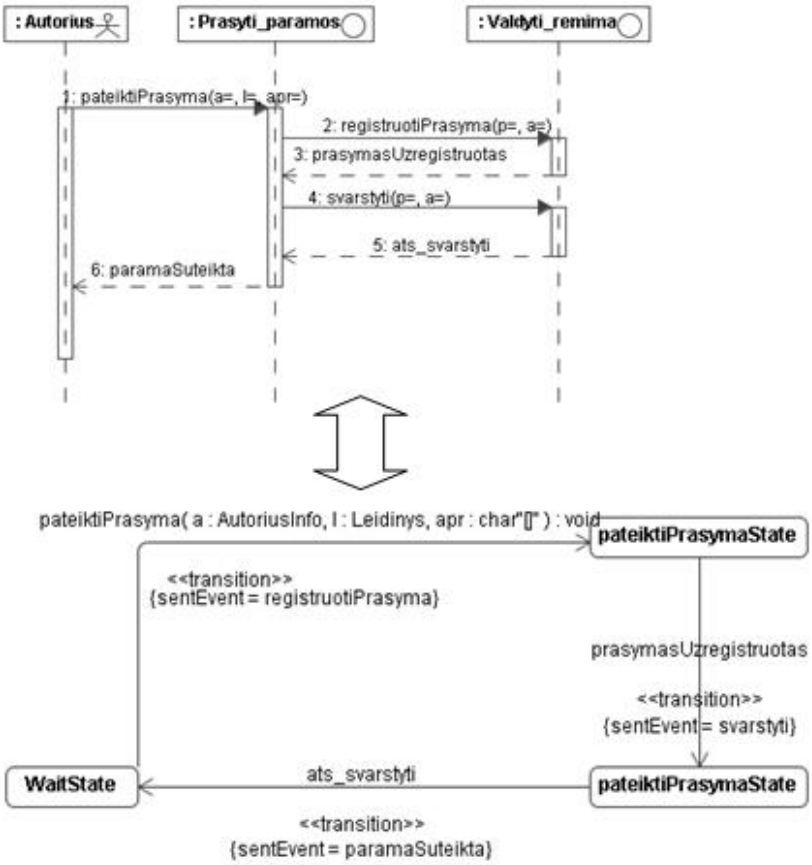


9 pav. MagicDraw elemento poklasės

Element saugo modelio elemento atributus: pavadinimą, priskirtus stereotipus, tėvinį bei vaikinius elementus. *Element* poklasių struktūra atitinka MOF UML 2.0 specifikacijoje aprašytą struktūrą. Modelio elementams valdyti MagicDraw aplinkoje skirta *ModelElementsManager* klasė. Šios klasės pagalba galima papildyti modelį savo sukurtu elementu, keisti modelio elementų atributus, pašalinti elementus iš modelio.

PresentationElement poklasė aprašo grafinį elemento atvaizdavimą. Šią poklasę galima naudoti MagicDraw diagramose elemento atvaizdavimo duomenims gauti bei keisti. Tai duomenys apie elemento išdėstymo diagramoje koordinates, dydį, spalvą bei kitus grafinius atributus, būdingus tam tikriems elementams. Diagramų elementų valdymą atlieka *PresentationElementsManager* klasė. Įskiepio kūrėjas, naudodamas *PresentatioElementsManager* klasę, gali kurti naujus, šalinti bei modifikuoti esamus diagramų elementus.

Kiekvienas sukurto modelio elementas yra vaizduojamas modelio medyje ir gali būti atvaizduojamas grafiškai atitinkamoje diagramoje. MagicDraw leidžia atvaizduoti daugumą modelyje esančių elementų, kurie reikalingi piešiant diagramas. MagicDraw nėra įgyvendintas modelyje sukurtų pranešimų atvaizdavimas diagramoje, todėl transformacijos metu gautų elementų seka vaizduojama pranešimų lange. 10 paveikslėlyje pateiktas transformacijos, gautos naudojant realizuotą įskiepį, pavyzdys.



10 pav. Transformacijos pavyzdys

5. Išvados

MDA tikslas yra automatizuoti programų sistemų kodo generavimą. Straipsnyje nagrinėjama ankstesnė sistemos gyvavimo ciklo stadija – reikalavimų modelio suderinamumo tikrinimas, modelį aprašančių sekų ir būsenų diagramų tarpusavio transformacijos.

Sukurta sekų ir būsenų diagramų abipusio transformavimo programinė realizacija MagicDraw įskiepio pavidalu. Šios programinės realizacijos sukūrimas rodo, kad galima automatizuoti ir taip palengvinti informacinių sistemų projektavimą, programiškai įgyvendinant sekų ir būsenų diagramų tarpusavio transformacijas. Naudojant transformacijas išvengiama nesuderinamumų modelyje.

Sekų ir būsenų diagramų abipusis transformavimas atliekamas naudojant tam tikrą algoritmą. Kad būtų galima transformacija iš būsenų diagramos, buvo papildytas UML naudojamą būsenų mašinų metamodelis.

Diagramų suderinimas reikalingas, kad būtų gaunamas kuo tikslesnis modelis, o iš tikslaus modelio galima generuoti programos kodą.

Literatūra

[1] **L.Ceponiene, L.Nemuraite**. 2004. Design Independent Modeling of Information System Using UML and OCL. Sixth International Baltic Conference on Data Bases and Information Systems (DB&IS'2004), Vol. 672: 357-372.

[2] **L.Ceponiene, L.Nemuraite**. Reconcilation of UML Models for Development of Information Systems. Kaunas University of Technology.

[3] **L.Čeponienė, L.Nemuraitė**. UML klasių, būsenų ir sekos diagramų suderinimas. Informacinės Technologijos'2003. - Kaunas: Technologija, 2003, p. XIV-62 – XIV-67

[4] **L.Čeponienė, L.Nemuraitė**. Transformations of UML Diagrams for Reconciliation of Requirements. In: Vasilecas, O. et al. (Eds.) Information Systems Development. Advances in Theory, Practice, and Education. Springer, 2005, XXVIII, ISBN: 0-387-25026-3, p. 289 – 301

[5] MagicDraw OpenAPI User‘s Guide, 2006

[6] MagicDraw product feature overview. www.magicdraw.com

[7] OMG 2003. MDA Guide Version 1.0.1. <http://www.omg.org/docs/omg/03-06-01.pdf>

[8] **T.Pender**. UML Bible. ISBN:0764526049. John Wiley & Sons, 2003, 940p.

MAGICDRAW TOOL EXTENSION WITH RECONCILIATION OF SEQUENCE DIAGRAMAS AND STATE MACHINES

In the paper, the platform independent model reconciliation verification is presented. This verification is based on reconciliation of UML sequence diagrams and state machines. For this purpose, plug-in for CASE tool MagicDraw is created. This plug-in helps designer to associate the collections of state machines and sequence diagrams, to check their reconciliation, to generate state machines from sequence diagrams and vice versa according to certain algorithm. The developed plug-in increases the working efficiency of designer and ensures the quality of design models.