KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Eglė Zaksaitė

Informacinei sistemai keliamų reikalavimų apibrėžimo metodika naudojant UML IR OCL

Magistro darbas

Darbo vadovė
doc. dr. L. Nemuraitė

Kaunas
2004

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. R.Butleris
2004 05

Informacinei sistemai keliamų reikalavimų apibrėžimo metodika naudojant UML IR OCL

Magistro darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. Jurgita Mikelionienė
2004 05

Vadovė
doc.dr. L. Nemuraitė
2004 05

Recenzentas
doc.dr. A.Lenkevičius
2004 05

Atliko
IFM-8/1 gr. stud.
E.Zaksaitė
2004 05

Kaunas
2004

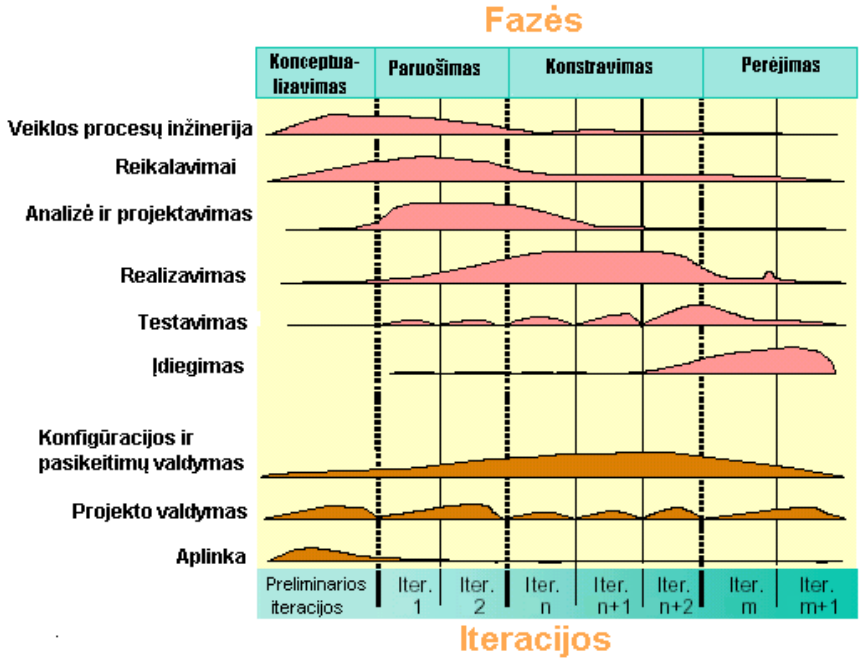
TURINYS

1 ĮVADAS

1.1 UML taikymas IS projektavime

Pastaraisiais metais programinės įrangos, ypač informacinių sistemų, strateginė vertė įmonėms labai išaugo, todėl vis labiau yra siekiama pakelti IS kokybę, sumažinti kainą ir laikotarpį, per kurį reikalingas programinis produktas pasiekia rinką. UML (angl. *Unified Modeling Language*) buvo sukurta patenkinti naujai iškilusius rinkos reikalavimus ir šiuo metu informacinių sistemų (IS) projektavimas neįsivaizduojamas be automatizuoto projektavimo (angl. *Computer Aided Software Engineering* – CASE) įrankių, kurie naudoja universalią modeliavimo kalbą, skirtą įvairiais aspektais aprašyti bei specifikuoti objektines sistemas. UML yra pripažinta informacinių sistemų projektavimo standartu – ji yra taikoma daugelyje paplitusių informacinių sistemų kūrimo procesų (RUP, ICONIX, XP…).

Reikalavimų registravimas, analizė ir projektavimas yra kritinė sistemų kūrimo fazė, kurios metu padarytos klaidos brangiai kainuoja. Pagal vieną iš labiausiai paplitusių projektavimo procesų RUP (angl. *Rational Unified Process)* ši fazė sudaro ne mažiau kaip pusę sistemos kūrimo laikotarpio, taigi UML taikoma būtent šioje fazėje, yra labai svarbi programinės įrangos kūrimo proceso dalis.



1 pav. *RUP fazės ir iteracijos*

UML pagrindas – diagramos. Iš esmės tai yra elementų grupės grafinis vaizdas. Diagramų tikslas – atvaizduoti skirtingus sistemos modelio aspektus, todėl diagrama yra sistemos modelio projekcija ir pasižymi tam tikru abstrakcijos laipsniu. UML turi devynių tipų – klasių, objektų, panaudojimo atvejų, sekų, bendradarbiavimo, būsenų, veiklos, komponentų bei įdiegimo – diagramas.

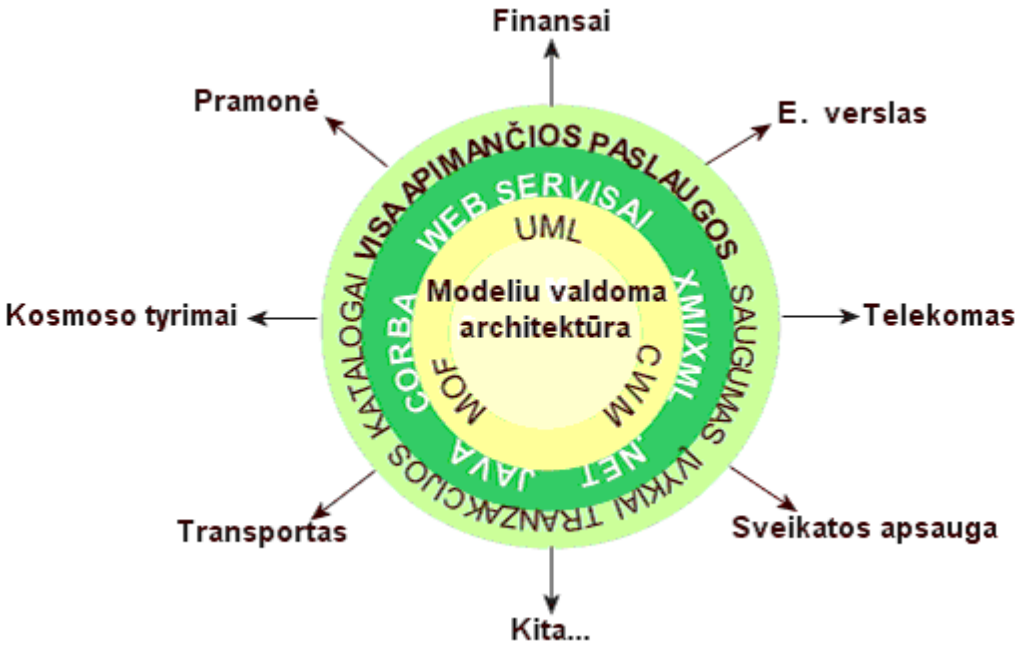
Informacinių sistemų projektavimo metodai nurodo sistemos projektavimo veiksmų seką – kaip, kokia tvarka ir kokias UML diagramas naudoti projektavimo procese bei kaip tą procesą vykdyti. Daugelis jų remiasi kelių tipų diagramomis, aprašančiomis skirtingų sistemos aspektų savybes. Kiekvienos diagramos prasmę galima nusakyti atskirai, bet svarbiau yra tai, kad kiekviena diagrama yra visos sistemos modelio projekcija.

Vis dėlto toks sistemos aprašymas yra gana painus, nes daugumoje diagramų pateikiama informacija iš dalies sutampa ir aprašo tuos pačius dalykus tik skirtingais aspektais. Be to, nepatyrusio specialisto netinkamai naudojamos UML diagramos gali būti nenuoseklios, nepilnos ir nevienareikšmiškai aprašyti sistemą. Dabartiniai UML CASE įrankiai tokiais atvejais labai nedaug kuo tegali padėti projektuotojui. Taigi automatizuoto projektavimo įrankiai labai palengvina projektuotojo darbą, tačiau vis dėlto nepakankamai jį automatizuoja. Didelis dėmesys skiriamas modeliavimui, ir projektuotojas šiai veiklai turi sugaišti nemažai laiko, kai galutinis darbo rezultatas vis dėlto yra programa, o ne modeliai. Yra siekiama kiek įmanoma formalizuoti modelių sudarymą, tam sukurta įvairių metodikų, tačiau dar egzistuoja ir neišnaudotų galimybių, kurios leistų patobulinti CASE įrankiais valdomus informacinių sistemų projektavimo metodus. Formalizavimas atveria naujas programinės įrangos ir informacinių sistemų projektavimo, kūrimo galimybes.

Pasiekus maksimalų UML modelių suderinamumą, kai modeliai tarpusavyje tvirtai susiję pagal aiškiai nusakytas taisykles, griežčiau išreikšti ir pilnesni, labai palengvėtų automatizuoto programinės įrangos kūrimo uždavinio sprendimas.

1.2 UML formalizavimo tikslai ir perspektyvos

Siekis formalizuoti UML, pasiekti maksimalų modelių suderinamumą yra susijęs su nauja programinės įrangos kūrimo idėja, pagal kurią siekiama atskirti verslo arba programos logiką nuo technologinės realizacijos. Šis metodas dar vadinamas modeliu valdoma architektūra arba MDA [1] (angl. *Model Driven Architecture*) ir laikomas ateities programinės įrangos kūrimo pagrindu.

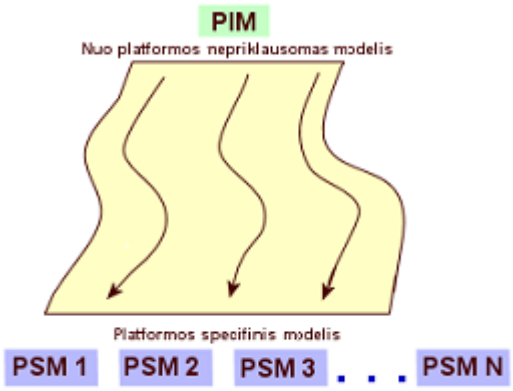


2 pav. *MDA struktūra*

MDA apibrėžia būdą, kaip atskirti sistemos funkcionalumo specifikaciją nuo jos įgyvendinimo tam tikroje platformoje. Tai leidžia automatizuoti informacinių sistemų kūrimo procesą pradedant nuo reikalavimų surinkimo, vėliau juos registruojant modeliuose bei iš jų generuojant būsimos sistemos struktūrą, kuri gali būti realizuota įvairiose platformose. Šio metodo sėkmę iš dalies lemia galimybė generuoti sistemos struktūrą turint UML modelius, taigi šiam uždaviniui reikalingas pilnas UML modelių suderinamumas.

Kuriant MDA metodu pagrįstas sistemas, pirmas žingsnis – nuo platformos nepriklausomo modelio PIM (angl. *Platform Independent Model*) sukūrimas, kuris ir turi būti išreikštas UML. Modelis privalo pilnai išreikšti loginį vaizdą, kuriame vienareikšmiškai specifikuoti komponentai ir jų elgsena. Taip pat reikalaujama, kad nuo platformos nepriklausomas modelis būtų pakankamai pilnas ir specifikuotas, kad būtų galima pradėti anksti vykdyti, testuoti programą, neatsižvelgiant į realizacijos specifines savybes. Šis etapas remiasi prielaida, kad PIM yra sukurtas remiantis griežtai apibrėžtu, formalizuotu UML.

Kadangi PIM apibrėžia tik būsimos sistemos logiką, sekančiame etape PIM logika turi būti įgyvendinta viename ar keliuose specifiniuose platformų modeliuose PSM (angl. *Platform Specific Model*). PSM yra išeities tekstai arba jų grafinis vaizdas išreikštas UML.



3 pav. *MDA etapai*

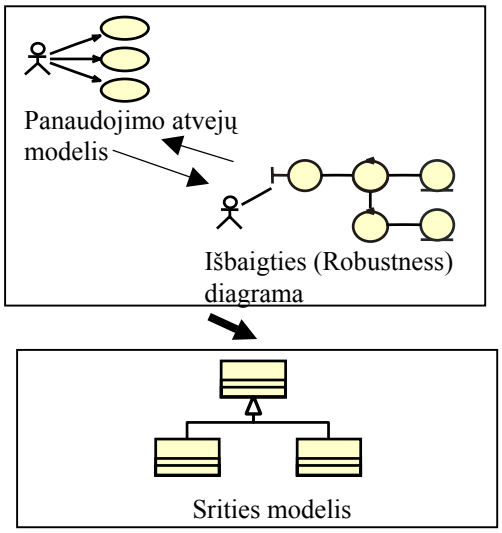
PSM vėliau gali būti naudojamas generuoti konkrečiam programiniam kodui, tačiau viskas priklauso nuo PIM ir PSM kokybės. Kuo aiškesnė, griežtesnė PIM logika, tuo kokybiškesnis gaunamas PSM ir programinis kodas.

Šiame darbe nagrinėjamas projektavimo etapas, kuris turėtų eiti prieš PIM sudarymą – reikalavimų apibrėžimas, kuris turi būti nepriklausomas nuo sistemos projekto, tai yra, jis turi apibrėžti tik dalykinę sritį ir vartotojo reikalavimus būsimai sistemai. Pagal [5], reikalavimų (arba nuo projekto nepriklausomas) modelis turi pilnai aprašyti siekiamos sistemos būsenos ir elgsenos schemą. Tam siūloma naudoti panaudojimo atvejų, klasių, būsenų bei sekų diagramas ir formalizuoti schemą, aprašant panaudojimo atvejus kaip kuriamos sistemos sąsajas. Reikalavimų specifikavime naudojama objektų apribojimų kalba OCL [7]. Darbe buvo stengiamasi ištirti ir naudoti naujos UML versijos savybes [6].

2 REIKALAVIMŲ APIBRĖŽIMO METODŲ ANALIZĖ

2.1 ICONIX procesas

Pagal [8] pateiktą analizę ICONIX programinės įrangos kūrimo metodas, pagrįstas minimaliu UML diagramų kiekiu ir efektyvia metodika, kurios dėka kūrimo procesas „nuo panaudojimo atvejų iki kodo" yra greitas ir efektyvus. ICONIX labai daug dėmesio skiria reikalavimų apibrėžimui. Šio proceso etapai: probleminės srities aprašymas, panaudojimo atvejų aprašymas, išbaigtumo (angl. *Robustness*) analizė ir sekos diagramų sudarymas.



1 pav. *ICONIX reikalavimų specifikacijos struktūra*

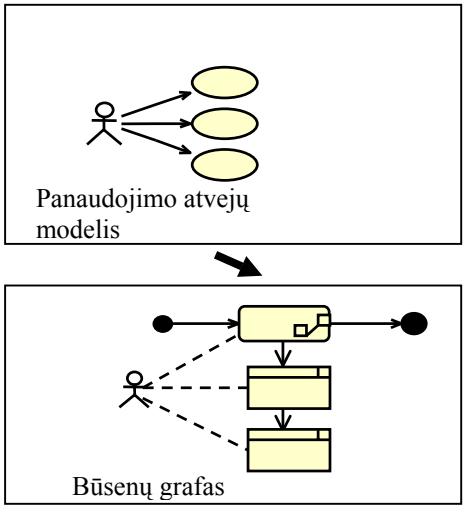
Pagrindinis *ICONIX* proceso rezultatas – sekos diagramos, kurios kartu su statine klasių diagrama toliau panaudojamos kodo rašymui. Siekiant ICONIX procesą padaryti pilnesniu, o projektavimą išsamesniu, šis procesas papildytas būsenų diagramomis. Be to, yra galimybė automatizuotam būsenų diagramos generavimui arba būsenų diagramų suderinimui su sekų diagramomis. Pirmajame etape kiekviena sekos diagrama (scenarijus) aprašoma įvykių seka, o antrajame generuojama būsenų diagrama.

ICONIX labai daug dėmesio skiria reikalavimų apibrėžimui, tačiau suderinamumas tarp panaudojimo atvejų, kuriuose užregistruoti reikalavimai ir kitų modelių, ypač klasių diagramų, nėra užtikrintas.

2.2 Scores reikalavimų inžinerijos metodas

Šiame metode [3] akcentuojama reikalavimų specifikavimo problema, kai reikalavimus atspindi panaudojimo atvejų ir klasių diagramos, tačiau jos sudaromos

naudojant skirtingas technikas bei pasižymi skirtingu abstrakcijos lygiu. Taigi tokiam reikalavimų modeliui labai trūksta suderinamumo ir pilnumo. Metodo autoriai šią problemą sprendžia įvesdami būsenų grafus, kurie lemia vientisą, atsekamą perėjimą nuo panaudojimo atvejų prie klasių diagramų. Be to, naudojant tokį panaudojimo atvejų ir klasių diagramų susiejimą, galima užtikrinti modelių teisingumą. Išskirtinis dėmesys pradinei informacinės sistemos kūrimo fazei leidžia sukurti gerą ir patikrintą specifikaciją, kuri atsiperka jau programos kodo kūrimo ir testavimo fazėje.



2 pav. *SCORES reikalavimų specifikacijos struktūra*

Scores reikalavimų specifikavimo metodo pirmame etape yra patobulinami panaudojimo atvejai, kad būtų galima pasiekti kuo aiškesnę specifikaciją. Įvedami būsenų grafai, atspindintys panaudojimo atvejų elgseną.

Būsenų grafas Scores reikalavimų specifikavimo metode charakterizuojamas:

- netuščia viršūnių (t. y. įvykių) aibe S;
- kryptį turinčių briaunų aibe $E \subseteq S \times S$ (perėjimai);
- pradiniu įvykiu, iš kurio išeina briaunos į kitas gretimas viršūnes (sekančius įvykius);
- netuščia galutinių veiksmų aibė $SF \subseteq S$.

UML panaudojimo atvejų terminologijoje panaudojimo atvejį aprašo aibė UML scenarijų, kurie vaizduoja konkretų atitinkamo veiksmo vykdymą. Scenarijus čia apibrėžiamas kaip įvykių seka, t.y. kelias per būsenų grafą, kuris prasideda pradiniu veiksmu ir tęsiamas atsižvelgiant į sąlygas, kurios tikrinamos įvykdžius būsenų grafo įvykį, kol pasiekiamas paskutinis veiksmas.

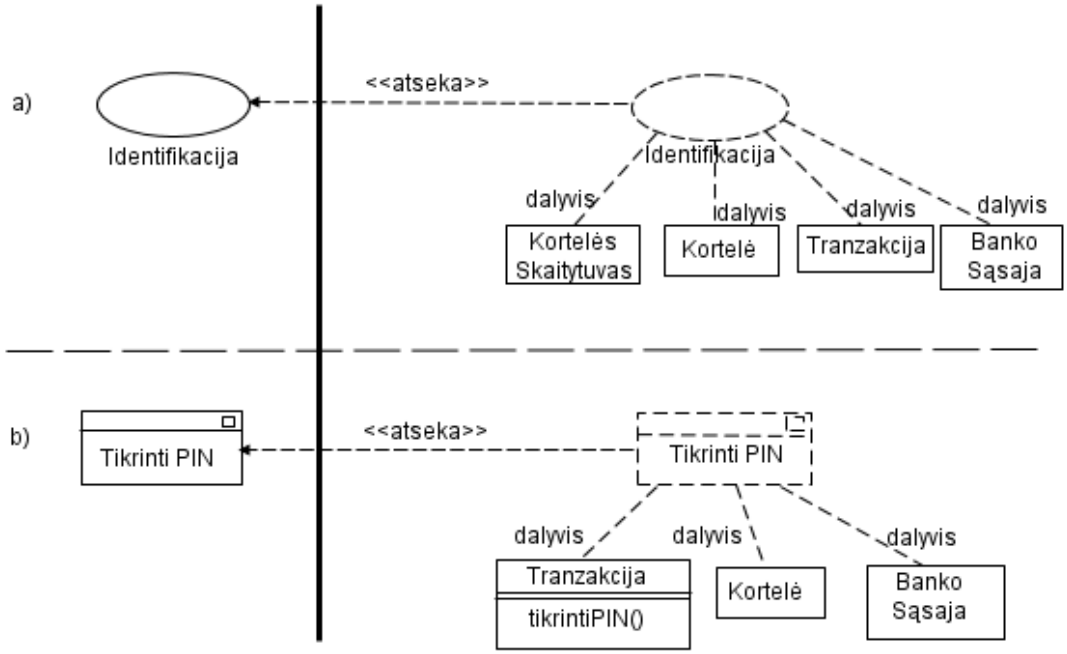
Vėliau būsenų grafais paremta panaudojimo atvejų specifikacija gali būti papildyta. Tam, kad būtų galima naudoti būsenų grafus panaudojimo atvejų modeliavimui, jie yra patobulinami įvedant 3 informacinių srautų, susiejančių skirtingas būsenas, tipus: konteksto ir sąveikos informaciją atspindi stereotipai <<kontekstinis veiksmas>> (angl. *contextual action*) ir <<sąveika>>(angl. *interaction*), aktorių susiejimui su įvykiais naudojamas <<veikiantis aktorius>>(angl. *actor in action*) stereotipas, o <<makro veiksmas>>(angl. *macro action*) atspindi <<apima>>(angl. *include*) ir <<išplečia>>(angl. *extend*) sąryšius būsenų grafuose.

Stereotipas	Aprašymas	Vaizdavimas
<<kontekstinis veiksmas>>	VeiksmoBūsena pažymėta kaip <<kontekstinis veiksmas>> nurodo veiksmo, kurį atlieka aktorius, neįtraukiant sistemos, vykdymą	
<<sąveika>>	IškvietimoBūsena pažymėta <<sąveika>> nurodo veiksmą, kuris yra atliekamas dalyvaujant sistemai	
<<veikiantis aktorius>>	ObjektoGyvavimoBūsena pažymėta stereotipu <<veikiantis aktorius>> nurodo aktorių, kuris gali būti susijęs su tam tikrais veiksmais	 : Aktorius
<<makro veiksmas>>	SubveiksmoBūsena pažymėta <<makro veiksmas>> "iškviečia" subgrafą, vaizduojantį panaudojimo atvejį su stereotipu "išplečia" arba "apima"	

3 pav. *Scores būsenų grafo veiksmai*

Taigi šiame metode kiekvienas būsenų grafe užfiksuotas veiksmas $s \in$ {<<kontekstinis veiksmas>>, <<sąveika>>, <<makro veiksmas>>}.

Sekančiame etape yra sudaromas klasių modelis. Kiekvienam panaudojimo atvejui yra nustatomos klasės, kurios susijusios su jo vykdymu. Tokia klasių aibė yra vadinama konkretaus panaudojimo atvejo klasių vaizdu, kuris gali būti pavaizduotas UML bendradarbiavimo diagrama. Analogiškai klasių vaizdas gaunamas ir būsenų grafo veiksmams, tik čia kiekvieną veiksmą būsenų grafe turi atitikti operacija klasių modelyje. Tokia klasė, kurioje yra operacija atitinkanti veiksmą iš būsenų grafo, vadinama šaknine klase, o jos operacija – šaknine operacija.

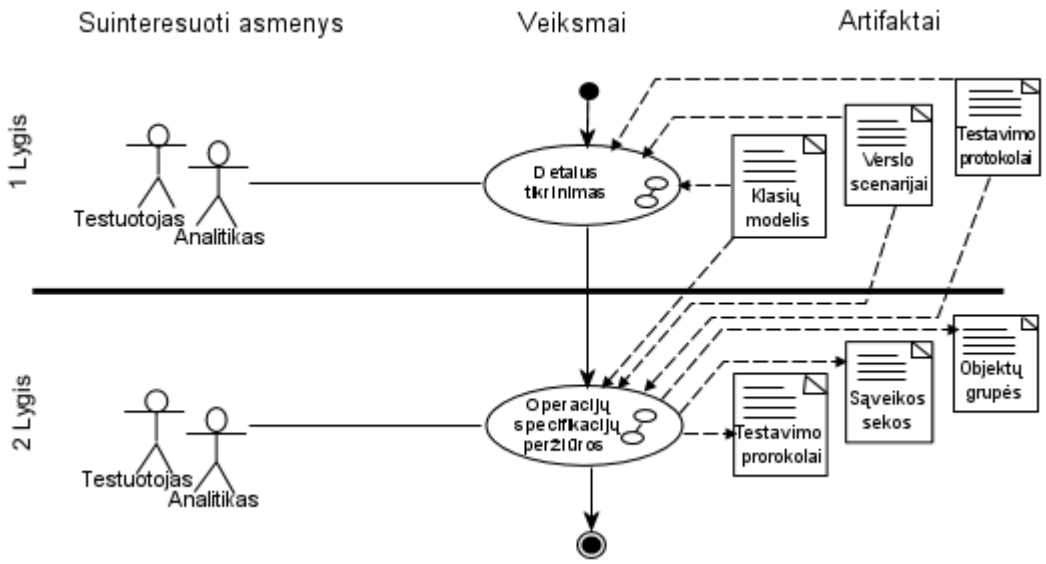


4 pav. *Panaudojimo atvejo ir būsenų grafo veiksmo susiejimo su klasių vaizdu pavyzdys*

Sekančiame etape sudaryti modeliai yra tikrinami – atliekama validacija ir verifikacija. Validacijos metu yra tikrinama, ar visi reikalavimai yra užfiksuoti, ar jie atitinka vartotojų norus ir tikslus. Verifikacijos metu atliekamas suderinamumo tikrinimas tarp panaudojimo atvejų ir klasių modelio. Toks tikrinimas yra dviejų lygių procesas.

Pirmame lygyje verifikacija pradedama nuo formalių panaudojimo atvejų dalių tikrinimo, t.y. žiūrima, ar prieš–sąlygos, po–sąlygos ir kiti elementai pilnai, teisingai aprašyti. Vėliau panašus tikrinimas atliekamas su klasių modeliu, o galutinis etapas – patikrinama, ar kiekvienas <<ąveika>> tipo veiksmas iš būsenų grafo turi atitikmenį klasių vaizde.

Antrajame lygyje atliekamas sekantis tikrinimas: einama per veiklos scenarijus (turi būti sudaryti iš anksto) ir sekamos operacijos, kurios iškviečiamos klasių modelyje priėjus atitinkamą <<sąveika>> tipo veiksmą būsenų grafe. Tokia gauta operacijų seka gali būti vaizduojama UML sekų diagrama. Tikrinant yra randama klaida klasių modelyje, jei tam tikru analizės metu aktyvūs objektai neatitinka sekančios operacijos prieš–sąlygos (kurie objektai turi būti aktyvūs pradedant sekančią operaciją).



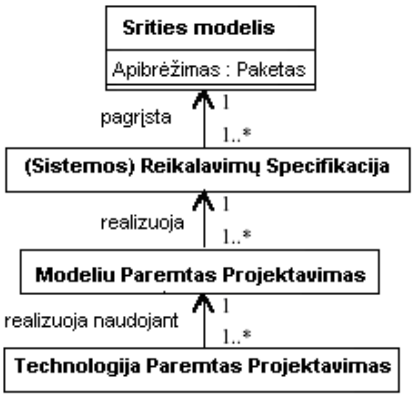
5 pav. *2 lygių verifikacija*

2.3 Reggio reikalavimų inžinerijos metodas

Šio metodo [4] tikslas – sukurti patobulintą ir griežtesnę reikalavimų specifikaciją, kuri leistų lengviau patikrinti įvairių modelių suderinamumą. Vienas esminių bruožų yra tai, jog sudaroma keletas su kuriama sistema susijusių vaizdų, be to, metodas remiasi panaudojimo atvejų diagramomis. Kartais yra nukrypstama ir nuo tradicinių objektinio projektavimo idėjų, bet tuo pačiu apjungiamos kitų metodų mintys, pvz. struktūrinės analizės. Išskiriamos 3 pagrindinės idėjos:

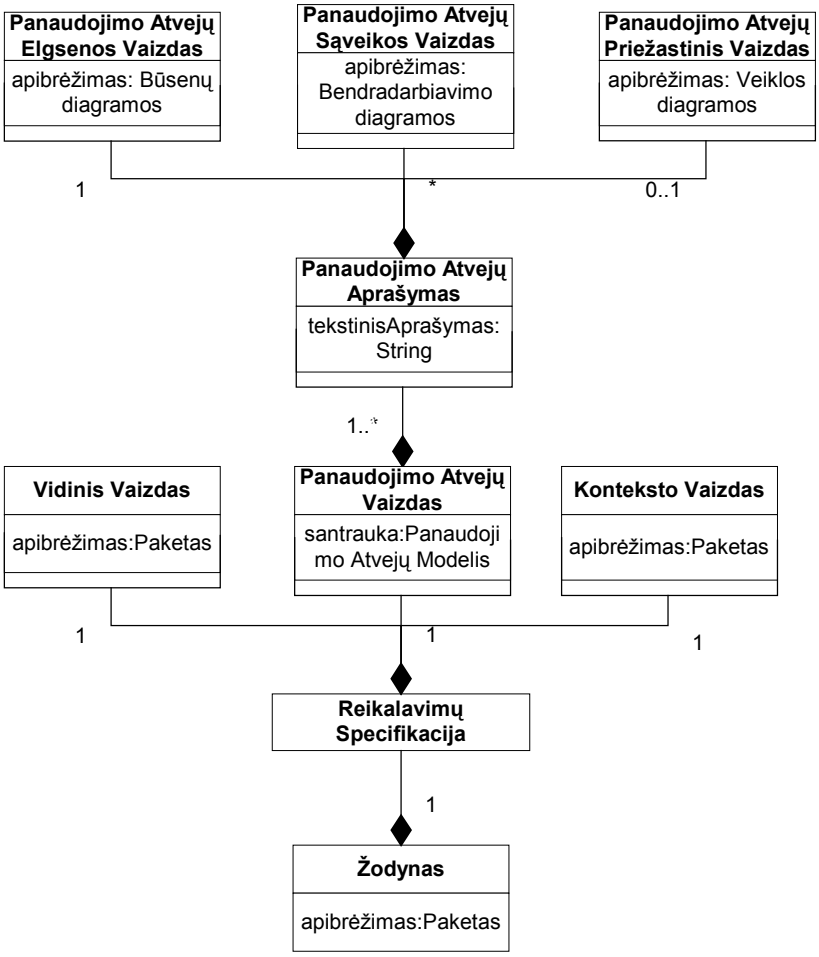
- Visiškas srities modelio (angl. *Domain Model*) atskyrimas nuo sistemos aprašo.
- Sistemos atskyrimas nuo jos aplinkos, kuris formalizuojamas konteksto vaizdu (angl. *Context View*). Konteksto vaizdas kartu su panaudojimo atvejais yra pagrindas reikalavimų apibrėžimui.
- Abstrakčios Būsenos (angl. *Abstract State*) naudojimas. Vietoje idėjos vaizduoti daug neprivalomų panaudojimo atvejų būsenų, yra apibrėžiama abstrakti būsena, kuri leidžia išreikšti abstrakčius sistemos ir jos konteksto sąveikos reikalavimus. Šioje stadijoje nenaudojama objektinė struktūra, kadangi jos dar nereikia. Sekančiuose projektavimo etapuose toks sistemos vaizdavimas išnyks, nes bus pereinama prie modeliais valdomos architektūros bei technologija valdomos architektūros.

6 pav. yra pavaizduoti pagrindiniai šiuolaikinės programinės įrangos kūrimo žingsniai, kuriais ir remiasi šis metodas. Pirmiausia yra atliekamas srities modeliavimas, vėliau seka reikalavimų specifikacija, modeliu valdomos architektūros sudarymas ir konkretaus technologinio taikymo architektūra. Akivaizdu, kad Reggio reikalavimų inžinerijos metodas yra suderinamas su šiuolaikiniu programinės įrangos kūrimo principu atskirti verslo arba programos logiką nuo technologinės realizacijos.



6 pav. Šiuolaikinės programinės įrangos kūrimo procesas

Kuriant reikalavimų specifikaciją pagal šį metodą, yra priimama, kad srities modelis jau yra sukurtas ir pateiktas UML diagramų pavidalu.



7 pav. Reikalavimų Specifikacijos Struktūra

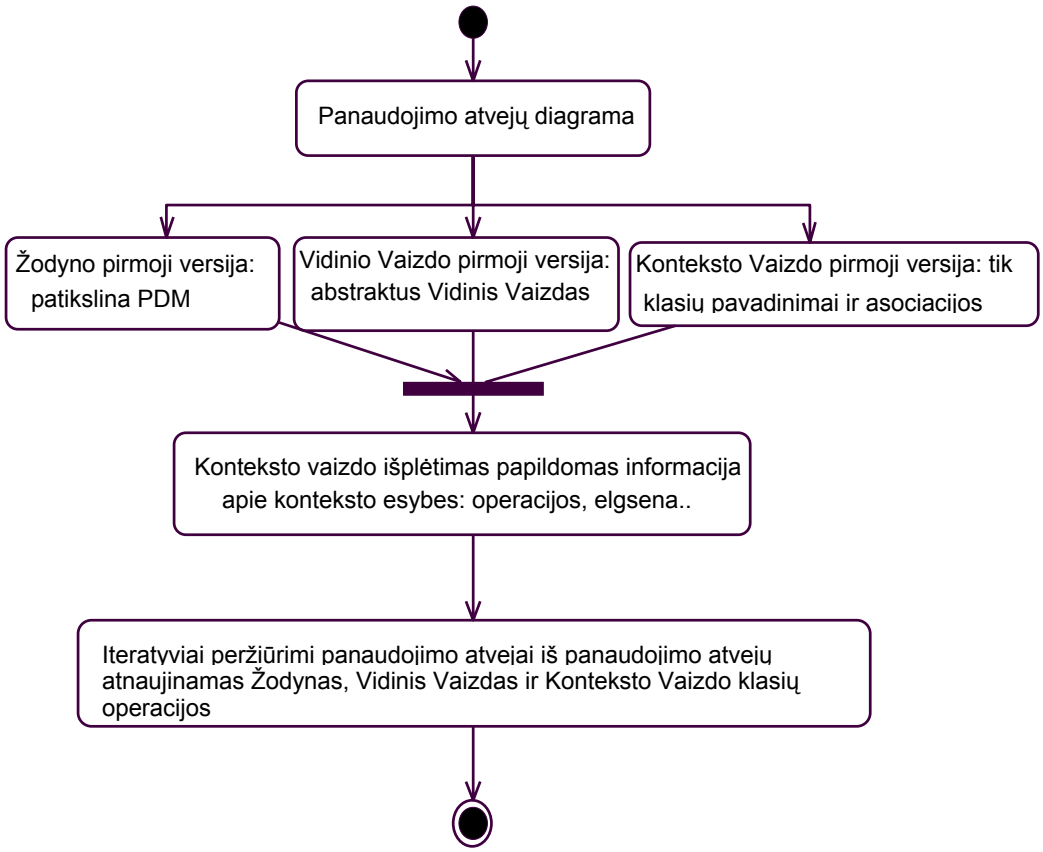
Iš srities modelio sudaroma specifikacija susideda iš skirtingų sistemos vaizdų ir žodyno (7 pav.). Konteksto vaizdas parodo sistemos kontekstą, t.y. esybes, kurios gali sąveikauti su sistema ir būdus, kaip yra sąveikaujama. Toks detalus sistemos ir jos aplinkos atskyrimas padeda išvengti painiavos tarp to, kas jau egzistuoja ir turi būti kuo tiksliau aprašyta (aplinka), ir to, ką reikia sukurti (sistema) bei užregistruoti reikalavimus.

Panaudojimo atvejų vaizdas (angl. *Use Case View*) standartiškai parodo visus sistemos naudojimo būdus ir tikslus.

Vidinis vaizdas (angl. *Internal View*) abstrakčiai aprašo vidinę sistemos struktūrą, kuri dar vadinama abstrakčia sistemos būsena. Ji leidžia labai tiksliai apibrėžti panaudojimo atvejų elgseną pagal tai, kaip kiekvienas iš jų skaito ir atnaujina abstrakčią būseną.

Žodyne yra tiksliai aprašomos visos įvairiuose sistemos vaizduose aptinkamos esybės.

Esminiai ir svarbiausi modeliai yra sistemos vidinis vaizdas ir konteksto vaizdas. Jie padeda užtikrinti panaudojimo atvejų ir visos reikalavimų specifikacijos suderinamumą.



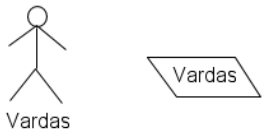
8 pav. Reikalavimų specifikacijos sudarymo uždaviniai

Reikalavimų specifikacijos sudarymo procesą galima suskirstyti į uždavinius:

- Panaudojimo atvejų diagramos sudarymas
- Žodyno pradinės versijos sukūrimas
- Pradinio konteksto vaizdo sudarymas
- Konteksto vaizdo išplėtimas
- Pradinio sistemos vaizdo sudarymas
- Atskirų vaizdų kiekvienam panaudojimo atvejui sukūrimas ir žodyno, vidinio bei konteksto vaizdų atnaujinimas.

Sudarant panaudojimo atvejų diagramas, yra įvedamas aktorių suskirstymas, t.y. sistemos ir jos teikiamų paslaugų vartotojas yra žymimas skirtingai nei paslaugų,

kurių reikia sistemai funkcionuoti, tiekėjai. Sistemos paslaugų vartotojas žymimas standartiškai kaip aktorius, o paslaugų sistemai tiekėjai žymimi rombu.



9 pav. Panaudojimo atvejų aktorių tipai

Žodynas susideda iš UML paketo, kur apibrėžiamos visos esybės, reikalingos reikalavimų specifikacijoje. Sudarant žodyną, yra naudojamas anksčiau sudarytas srities modelis, todėl nebūtinai visos klasės, pavaizduotos žodyne, vėliau bus aprašytos reikalavimų specifikacijoje.

Konteksto vaizdas susideda bent iš vienos klasių diagramos, kurioje gali būti 3 tipų klases apibrėžiantys stereotipai: sistema, sistemos vartotojas ir sistemai reikalingų paslaugų tiekėjas. Tokioje klasių diagramoje visi ryšiai tarp klasių yra asociacijos iš sistemą vaizduojančios klasės į kiekvieną kitą klasę.

Pirmojoje vidinio sistemos vaizdo versijoje yra pateikiama labai abstrakti sistemos architektūra. Čia sistemą sudaro vienas aktyvus objektas, atliekantis sistemos veiksmus ir daug pasyvių objektų, apibūdinančių abstrakčią sistemos būseną.

Sekančiame etape išplečiamas konteksto vaizdas, tam yra panaudojamos žinios apie konteksto esybes, t.y. apibrėžiamos pradinio konteksto vaizdo klasių operacijos ir savybės.

Paskutiniame etape pateikiamas tekstinis panaudojimo atvejų aprašas, o taip pat ir grafiniai jų vaizdai, kurie gali atspindėti elgseną, sąveiką arba priežastingumą.

Elgsenos panaudojimo atvejų vaizdas yra privalomas visiems panaudojimo atvejams ir apibrėžiamas būsenų diagrama, kurioje vaizduojamas sistemą apibūdinančios klasės elgesys tam tikro panaudojimo atvejo atžvilgiu.

Panaudojimo atvejų sąveikos vaizdas yra apibrėžiamas sekų arba bendradarbiavimo diagrama, kurioje vaizduojama sąveika tarp konteksto esybių, sistemos ir vidinio sistemos vaizdo sudedamųjų dalių (anksčiau aprašytų aktyvaus ir pasyvių objektų). Sąveiką atspindinčios diagramos turi būti suderinamos su elgsenos vaizdu, t.y. turi atvaizduoti tam tikro panaudojimo atvejo vykdyme atsirandančias sąveikas.

Panaudojimo atvejų priežastinis vaizdas yra apibrėžiamas veiklos diagrama ir parodo įvairius svarbius įvykius, faktus būdingus panaudojimo atvejams ir jų

priežastiniams ryšiams. Priežastinis vaizdas turi būti suderintas su elgsenos vaizdu, t.y. priežastiniai ryšiai tarp „faktų" gali turėti atitikmenis elgsenos diagramose.

2.4 Išnagrinėtų metodų tinkamumas pasirinktam uždaviniui

Apžvelgtuose programinės įrangos kūrimo metoduose įvairiai sprendžiamas UML diagramų suderinamumas. Nors ne visas išnagrinėtų metodų idėjas eina panaudoti mano sprendžiamam uždaviniui, tačiau atskirus jų elementus galima pritaikyti sudarant geresnę reikalavimų apibrėžimo metodiką.

ICONIX programinės įrangos kūrimo metodui būdingos sekų ir išbaigties diagramos kiekvienam panaudojimo atvejui padeda užtikrinti reikalavimų specifikacijos pilnumą, išbaigtumą. Tačiau toks reikalavimų apibrėžimas nėra griežtai atskirtas nuo projekto modelio (reikalavimų specifikacijoje įvedami tik vėliau projekto modelyje reikalingi valdikliai), o tai nesuderinama su modeliu valdomos architektūros idėjomis.

Scores reikalavimų inžinerijos metodas dėl reikalavimų specifikavimo metu nuolat vykdomų tikrinimo ir atestacijos procesų pasižymi aukštu pradinių vartotojo reikalavimų išpildymo laipsniu, tačiau yra gana sudėtingas ir reikalauja nemažai darbo, be to, neatsižvelgiama į gauto projekto klasių modelio kokybę (pvz. klasių tarpusavio susietumo pobūdį), nuo kurios labai priklauso praktinė realizacija. Tačiau tinkama pasirodė panaudojimo atvejų ir klasių modelio susiejimo idėja pagal panaudojimo atvejų žingsnius (čia yra įvestas patobulinimas – būsenų grafai).

Reggio reikalavimų inžinerijos metodas neapibrėžia aiškaus būdo, kaip pereinama nuo panaudojimo atvejų prie projekto klasių diagramų, taip pat yra gana sudėtingas ir reikalaujantis nemažai darbo. Tačiau projektavimą, remiantis panaudojimo atvejais, duomenų žodynu, dalykinės srities modeliu galima taikyti, siekiant geresnio modelių suderinamumo.

Lentelė Nr. 1. Panaudotos metodų savybės

Metodas	**Kaip panaudotos metodo savybės**
ICONIX	Sekų diagramos ir išbaigties diagramos sudaromos kiekvienam panaudojimo atvejui, bet atsisakoma valdiklių (t.y., priešlaikinių projektinių sprendimų).
Scores	Papildomos diagramos panaudojimo atvejų ir klasių diagramų susiejimui ir modelių suderinamumui – sąsajų diagramos (Scores – būsenų grafai).
Reggio	Bendra reikalavimų modelio struktūra, tačiau modifikuotas jo elementų turinys.

Lentelė Nr. 2. Išnagrinėtų metodų trūkumai

Metodas	**Trūkumai**
ICONIX	Nėra užtikrintas suderinamumas tarp panaudojimo atvejų ir kitų modelių, ypač klasių diagramų.
Scores	Sudėtingas, neatsižvelgiama į gauto klasių modelio kokybę.
Reggio	Neapibrėžiamas aiškus būdas, kaip pereiti nuo panaudojimo atvejų prie klasių diagramų, aprašančių kontekstą; sistema aprašoma viena klase, todėl prarandama informacija apie sistemos operacijų grupavimą.

3 REIKALAVIMŲ APIBRĖŽIMO METODIKA

Šioje dalyje aptariamas siūlomas reikalavimų apibrėžimo metamodelis, kuris išsamiai aprašo kuriamos sistemos elgseną nepriklausomai nuo projekto modelio. Metamodelis atitinka MDA idėjas, jame yra integruoti kai kurie anksčiau išnagrinėtų metodų komponentai. Taip pat detaliau analizuojami sudaryto modelio elementų tarpusavio ryšiai, kurie užtikrina diagramų tarpusavio suderinamumą. Pateiktas reikalavimų apibrėžimo ir suderinimo procesas padeda sudaryti reikalavimų specifikaciją, atitinkančią gautą metamodelį.

3.1 Reikalavimų metamodelis

Šiame darbe siekiant sudaryti patobulintą reikalavimų apibrėžimo metodiką, didelis dėmesys buvo skiriamas praktiniam sistemos projektavimui, kuris vykdomas iteratyviu būdu, stengiantis pasiekti vis didesnį naudojamų modelių tarpusavio atitikimą. Tokiu būdu gavus pakankamai gerą, išsamiai apibrėžtą ir suderintą reikalavimų modelį, buvo galima sudaryti metamodelį, kuris apibendrintų, kokios diagramos, jų elementai turėtų būti naudojamos ir kaip jos turėtų būti susietos tarpusavyje.

3.1.1 Reikalavimų apibrėžimo struktūra

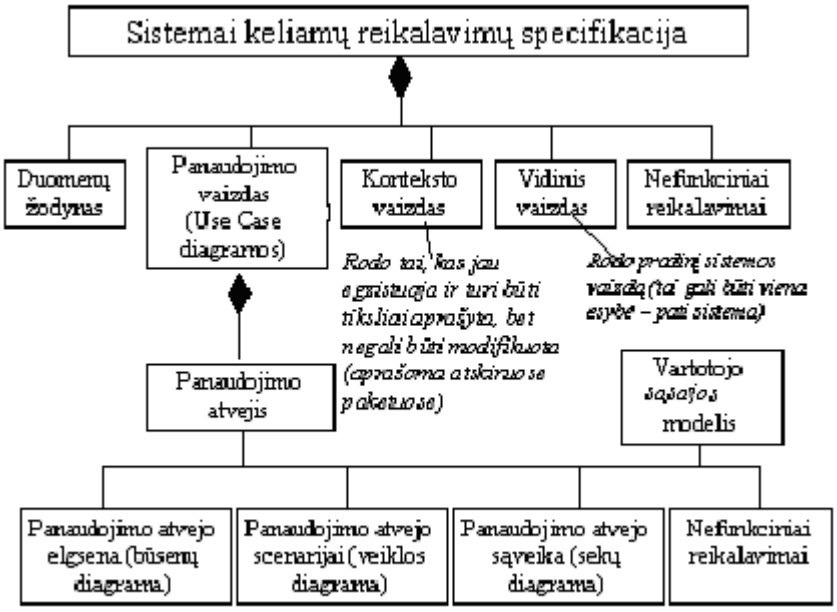
Pirmas žingsnis sudarant reikalavimų metamodelį, yra reikalavimų apibrėžimo struktūros pasirinkimas. Išnagrinėjus analizės dalyje pateiktus egzistuojančius metodus, kaip tinkamiausia buvo pasirinkta Reggio reikalavimų apibrėžimo struktūra. Projektavimo eigoje ji buvo papildyta trūkstamais elementais (1 pav.).

Reikalavimų modelį sudaro duomenų žodynas, kurį galima apibrėžti kaip dalykinės srities klasių diagramą, nusakančią pagrindines, su projektuojama sistema susijusias esybes.

Labai svarbus yra panaudojimo vaizdas, susidedantis iš panaudojimo atvejų. Panaudojimo atvejį šioje reikalavimų specifikacijoje pilnai nusako panaudojimo atvejo elgsena (jo būsenų diagrama), panaudojimo atvejo scenarijai (veiklos diagrama), panaudojimo atvejo sąveika (sekų diagrama) ir papildomai įvesti atskiro panaudojimo atvejo nefunkciniai reikalavimai bei vartotojo sąsajos modelis. Nefunkciniai reikalavimai yra įvedami, siekiant specifikacijos pilnumo, o vartotojo sąsajos modelis yra naudojamas siekiant formalizuoti panaudojimo atvejus – neformaliai aprašytą panaudojimo atvejį turi atitikti formalus sąsajos aprašas.

Reikalavimų specifikacijos konteksto vaizdas aprašo kuriamos sistemos kontekstą, aplinką, t.y. tai, kas jau egzistuoja ir kas padeda geriau suvokti sistemai keliamus reikalavimus. Šis vaizdas aprašomas atskiruose paketuose.

Vidinis vaizdas nusako pradinį sistemos vaizdą, kurį pradžioje dažniausiai sudaro tik viena esybė – kuriama sistema.

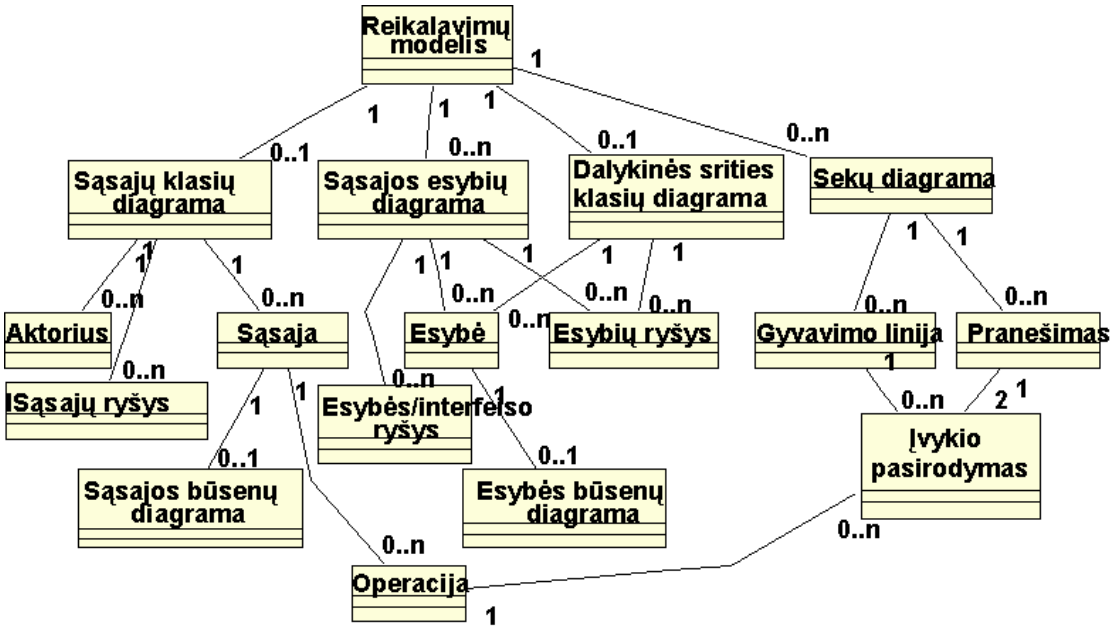


1 pav. Reikalavimų specifikacijos struktūra

3.1.2 Reikalavimų metamodelio struktūra

Pagal pasirinktą reikalavimų specifikacijos struktūrą, kurioje yra nurodytos diagramos, kurias reikėtų sukurti, buvo praktiškai sudarytas reikalavimų modelis. Toks praktinis sprendimas padėjo tiksliau nustatyti, kokia diagramų ir jų elementų paskirtis, kokie sistemos funkcionavimo aspektai kokiomis diagramomis atvaizduoti ir kokia informacija modeliuose iš dalies persidengia. Toks tyrimas buvo apibendrintas reikalavimų metamodelyje (2 pav.). Šis eskizas parodo, kokios diagramos ir tų diagramų elementai yra naudotini reikalavimų modelyje, taip pat parodyti sąryšiai tarp diagramų ir jų atskirų elementų. Sąryšių informacija yra būtina, siekiant tiek sudaryti teisingą reikalavimų modelį, tiek patikrinti, ar sudarytas reikalavimų modelis yra teisingas. Buvo siekiama rasti kuo daugiau atitikimų tarp elementų ir taip dar labiau formalizuoti modelį.

Gautas reikalavimų modelis skiriasi nuo neformalių panaudojimo atvejų aprašymo natūralia kalba arba eskizinio pobūdžio diagramomis – tai yra išsamiai apibrėžtas reikalavimų modelis.



2 pav. Reikalavimų apibrėžimo metamodelis

Aprašomi skirtingi elgsenos aspektai, naudojami duomenys, būsenų kaita, sąveikos – ir jie visi tarpusavyje suderinami. Be to, kartu toks metamodelis parodo, kad nors UML diagramos iš dalies persidengia, kiekviena jų atskleidžia skirtingas savybes ir, norint gerai išnagrinėti reikalavimus, reikia sudaryti visas šias diagramas.

Eskize nėra parodyta panaudojimo atvejų diagramos, kadangi ji laikoma įėjimo duomenimis, kuriais remiantis yra sudaromas reikalavimų modelis.

3.2 Reikalavimų metamodelio diagramų tarpusavio ryšiai

Pateiktas reikalavimų metamodelis pasižymi diagramų tarpusavio suderinamumu, kurį atspindi elementų tarpusavio sąryšiai. Galima detaliau apibūdinti kiekvieną ryšį:

Reikalavimų modelis (1) – Sąsajų klasių diagrama(0..1)

Šis ryšys parodo, jog reikalavimų modelį sudaro sąsajų klasių diagrama. Panaudojimo atvejai sudaromi viename iš pradinių reikalavimų apibrėžimo etapų, jie aprašomi neformaliai, dėl ko atsiranda problemų vėliau pereinant prie projekto modelio, o formali sąsajų klasių diagrama palengvina šį perėjimą, padeda užtikrinti reikalavimų specifikacijos pilnumą bei teisingumą.

Sąsajų klasių diagrama(1) – Aktorius(0..n)

Ryšys parodo, jog sąsajų klasių diagramą sudaro aktoriai, kurie atitinka panaudojimo atvejų diagramos aktorius (kadangi sąsajomis formalizuojami panaudojimo atvejai).

Sąsajų klasių diagrama(1) – Sąsaja(0..n)

Šis ryšys taip pat parodo, kad sąsajų klasių diagrama susideda iš sąsajų, kurios gautos iš neformalių panaudojimo atvejų, ir bus panaudotos gauti projekto modelio klasėms.

Sąsajų klasių diagrama(1) – Sąsajų ryšys(0..n)

Sąsajų klasių diagramoje yra tarpusavyje susietos ryšiais, būdingais klasių diagramoms ir nusakančiais sąsajų valdymo struktūrą.

Sąsaja(1) – Sąsajų būsenų diagrama(0..1)

Kadangi sąsaja atitinka panaudojimo atvejį, jis nurodo kokį nors sistemos atliekamą veiksmą, o jo vykdymas keičia sąsajos būseną. Sąsajos būsenų diagrama parodo visas būsenas, į kurias sąsaja patenka.

Sąsaja (1) – Operacija(0..n)

Šis ryšys parodo, jog sąsaja paprastai turi operacijas, kurios keičia sąsajos būseną ir tokiu būdu realizuojama sąsajai priskirta užduotis. Operacijos yra specifikuojamos OCL kalba, kartu nusakant jų „prieš" ir „po" sąlygas bei operacijų argumentus.

Reikalavimų modelis (1) – Dalykinės srities klasių diagrama(0..1)

Šis ryšys parodo, kad laikantis reikalavimų apibrėžimo metodikos, turi būti sudaryta dalykinės srities diagrama, kurioje parodomi pagrindiniai objektai, susiję su kuriama sistema. Siekiant formalesnio aprašymo, dalykinės srities klasių apribojimai užrašomi OCL kalba.

Dalykinės srities klasių diagrama(1) – Esybė(0..n)

Dalykinės srities klasių diagramą dažniausiai sudaro keletas esybių, iš esmės apibūdinančių tiriamą sritį.

Esybė(1) – Esybės būsenų diagrama(0..1)

Esybė, apibūdinanti tam tikrą dalykinės srities objektą, kuris susijęs su kuriama sistema, gali turėti keletą būsenų, kurios kinta, vykdant tam tikras sąsajų operacijas. Šios būsenos parodomos esybės būsenų diagramoje ir padeda pilnai aprašyti būsimą sistemą.

Dalykinės srities klasių diagrama(1) – Esybių ryšys(0..n)

Esybės dalykinės srities klasių diagramoje yra tarpusavyje susijusios ryšiais, o ryšius patogu apibrėžti ir formalizuoti OCL kalba.

Reikalavimų modelis (1) – Sąsajų esybių diagrama(0..n)

Sąsaja, kuri apibūdina tam tikrą sistemos atliekamą veiksmą yra susijusi su tam tikromis esybėmis iš dalykinės srities klasių diagramos. Sąsajų esybių diagrama konkretizuoja su kokiomis esybėmis ir kaip sąsaja susijusi.

Sąsajos esybių diagrama(1) – Esybė(0..n)

Sąsaja gali būti susijusis su keliomis dalykinės srities esybėmis, kurias parodo sąsajos operacijų argumentai.

Sąsajos esybių diagrama(1) – Esybių ryšys(0..n)

Šioje diagramoje esybių tarpusavio ryšiai atitinka tuos pačius ryšius dalykinės srities klasių diagramoje, tik čia yra vaizduojama ta dalykinės srities klasių diagramos dalis, kuri yra susijusi su nagrinėjama sąsaja.

Sąsajos esybių diagrama(1) – Esybės/sąsajos ryšys(0..n)

Sąsajos esybių diagramoje yra ryšys tarp sąsajos ir esybių, jei tik sąsaja yra susijusi bent su viena esybe.

Reikalavimų modelis (1) – Sekų diagrama(0..n)

Sekų diagramos yra skirtos detalizuoti panaudojimo atvejams, kadangi jie suskaidomi į atliekamas operacijas, parodoma jų vykdymo seka, alternatyvūs vykdymo variantai.

Sekų diagrama(1) – Gyvavimo linija(0..n)

Šis ryšys rodo, jog sekų diagramos vienas iš elementų yra gyvavivo linijos, kurių pagalba galime suprasti, kiek ir kada objektas yra aktyvus, t.y. siunčia ar gauna pranešimus.

Sekų diagrama(1) – Pranešimas(0..n)

Sekų diagramoje labai svarbūs pranešimai – jie įvairiai keičia vykdomų veiksmų eigą, pvz. iššaukia perėjimą į kitą būseną, parodo kad tam tikra būsena baigėsi ir pereinama į laukimo būseną ir pan.

Pranešimas(1) – Įvykio pasirodymas(2)

Šis ryšys parodo, kad vienas pranešimas sekų diagramoje turi iššaukti 2 įvykių pasirodymus („siųsti užklausą operacijai” ir „gauti užklausą operacijai” arba „siųsti atsakymą” ir „gauti atsakymą”)

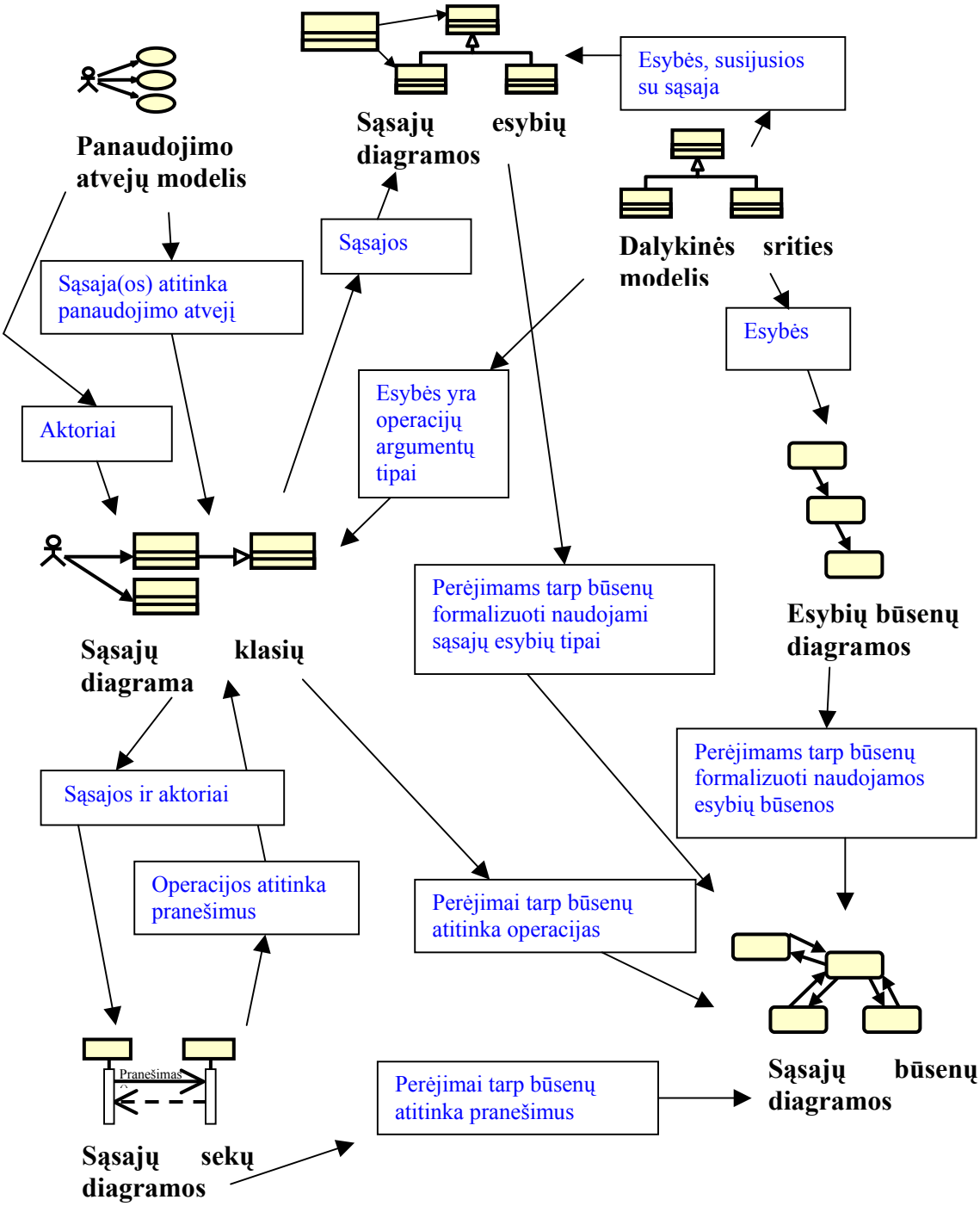
Gyvavimo linija(1) – Įvykio pasirodymas(0..n)

Sekų diagramos gyvavimo liniją gali atitikti daug įvykio pasirodymų, kadangi tiek aktorius, tiek sąsaja gali siųsti ir gauti įvairiuss pranešimus.

Operacija(1) – Įvykio pasirodymas(0..n)

Ryšys parodo, jog sąsajos operaciją atitinka daug įvykių pasirodymų sekų diagramoje.

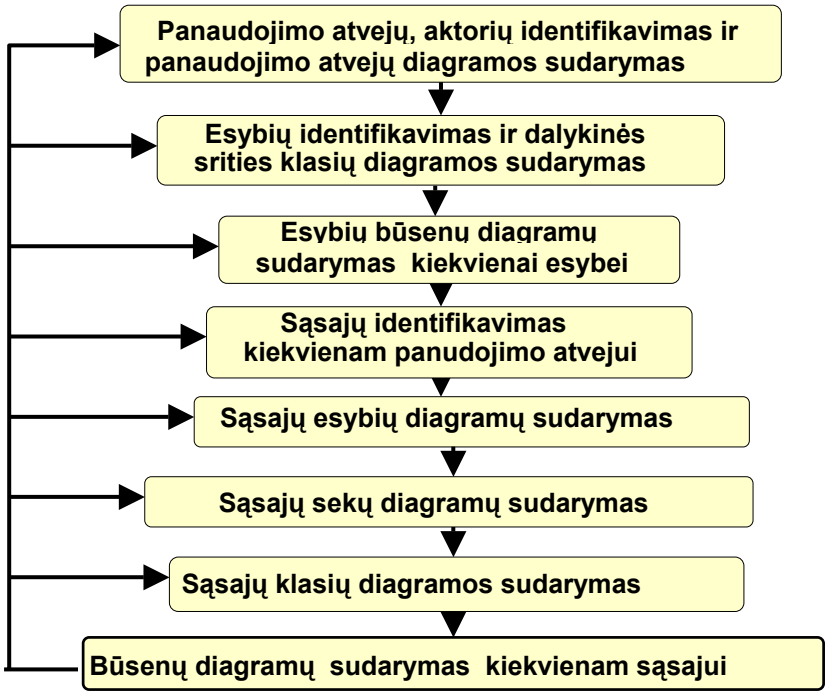
Diagramų tarpusavio sąryšius galime pavaizduoti diagrama, kurioje parodomos reikalavimų apibrėžimui naudojamas diagramas ir kaip (per kokius elementus) jos tarpusavyje susijusios.



3 pav. Diagramų tarpusavio sąryšiai

3.3 Reikalavimų apibrėžimo ir suderinimo procesas

Reikalavimų metamodelis parodo, koks turėtų būti reikalavimų apibrėžimo proceso rezultatas, tačiau nėra aišku, kaip tokį rezultatą pasiekti ir kokia veiksmų eiga tam turi būti atlikta. Todėl siekiant pilnai nusakyti siūlomą reikalavimų apibrėžimų metodiką, čia yra pateikiamas reikalavimų apibrėžimo ir suderinimo procesas (4 pav.).



4 pav. Perėjimo nuo panaudojimo atvejų prie projekto klasių diagramos algoritmas

Šis algoritmas gautas išnagrinėjus keletą ankščiau aprašytų esamų reikalavimų apibrėžimo metodų, taip pat buvo atliktas realios sistemos išsamus reikalavimų aprašymas, kuris leido pasirinkti tinkamiausią reikalavimų apibrėžimo ir suderinimo proceso variantą.

Algoritmas remiasi panaudojimo atvejų diagrama, todėl pirmiausia yra identifikuojami panaudojimo atvejai, aktoriai ir sukuriama panaudojimo atvejų diagrama. Šis pirmasis etapas yra vienodas daugeliui reikalavimo apibrėžimo metodų ir gerai suprantamas. Sekančiame etape reikalavimų modelio pilnumui būtina sudaryti dalykinės srities klasių diagramą, kurios sudarymo metu yra išskiriamos esybės, susijusios su kuriamos sistemos veikla. Taip pat nubraižomos būsenų diagramos kiekvienai esybei. Po to, kiekvienam panaudojimo atvejui sukuriamos sąsajos. Sąsajos padeda formaliai aprašyti panaudojimo atvejus, o su dalykinės srities klasių

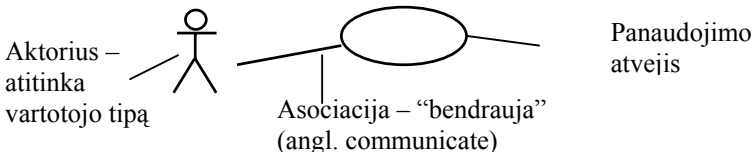
diagramos esybėmis sąsajos susiejamos sąsajų esybių diagramose. Vėliau, siekiant geriau ir tiksliau identifikuoti sąsajų operacijas, yra nubraižomos sekų diagramos kiekvienai sąsajai. Po sekų diagramų jau galima nubraižyti detalią sąsajų klasių diagramą ir sukurti būsenų diagramas kiekvienai sąsajai.

Reikia pabrėžti, kad šis reikalavimų apibrėžimo procesas yra iteratyvus ir gali būti sugrįžtama į bet kurį ankstesnį etapą bet kuriuo proceso metu. Kai sistemos elgsena visais aspektais jau tenkina iškeltus poreikius, galima sakyti, kad reikalavimai suformuluoti.

3.3.1 Panaudojimo atvejų, aktorių identifikavimas. Panaudojimo atvejų diagramos sudarymas

Reikalavimų apibrėžimo procesas prasideda nuo panaudojimo atvejų modelio sudarymo. Įvairių projektuotojų daugelio metų patirtis ir įvairių technikų naudojimas parodė, jog tai yra vienas iš geriausių būdų, padedančių apibrėžti reikalavimus.

Panaudojimo atvejis aprašo su vartotoju bendraujančios sistemos ar jos dalies elgesį veiksmų sekų aibe. Šio elgesio rezultate vartotojas gauna tam tikrą jam reikšmingą rezultatą.



5 pav. Panaudojimo atvejo struktūra

Sistemos reikalavimai, taip pat ir panaudojimo atvejai visada yra išgaunami ir sudaromi galvojant apie būsimą sistemą iš verslo, vartotojo ir techninės pusių. Analizuojant iš verslo perspektyvos yra identifikuojami suinteresuoti asmenys, tiesioginiai vartotojai arba panaudojimo atvejų aktoriai. Nagrinėjant sistemą iš vartotojo pusės, iškeliami visi vartotojo poreikiai, užduotys, iš kurių identifikuojami panaudojimo atvejai. Iš sistemos nagrinėjimo techniniame lygyje kyla reikalavimai, kurie gali koreguoti jau sudarytus panaudojimo atvejus. Tačiau panaudojimo atvejai turi aprašyti sistemos elgesį, bet ne jos realizaciją.

Panaudojimo atvejus išskirti padeda atsakymai į tokius klausimus:

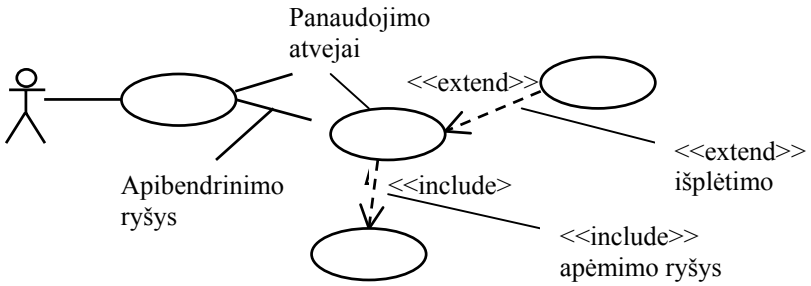
- ***Kiekvienam išskirtam aktoriui – kokias užduotis aktorius turi atlikti su sistema?***

- ***Apie ką sistema turi pranešti vartotojui?***
- ***Apie ką aktorius turi pranešti sistemai?***
- ***Ar panaudojimo atvejai atitinka visus veiklos poreikius?***
- ***Kokie panaudojimo atvejai bus skirti sistemos palaikymui?***
- ***Kokią informaciją reikia sukurti ar modifikuoti sistemoje?***

Reikia sekti, kad būtų užfiksuoti visi panaudojimo atvejai, netgi tie, kurie nusako netipines sistemos funkcijas:

- ***Sistemos paleidimas ir sustabdymas***
- ***Sistemos palaikymas, pvz., naujų vartotojų įvedimas ir modifikavimas***
- ***Duomenų palaikymas, pvz., sinchronizavimas su sena sistema***
- ***Sistemos elgesio modifikavimas***

Panaudojimo atvejai yra identifikuojami ir detalizuojami palaipsniui, pradedant nuo keleto pagrindinių ir juos skaidant į smulkesnius, išplečiant. Detalizavimui ir smulkesnių panaudojimo atvejų įvedimui gali būti naudojami **<<include>>**, **<<extend>>** ir **apibendrinimo** ryšiai.



6 pav. Panaudojimo atvejų tarpusavio sąryšiai

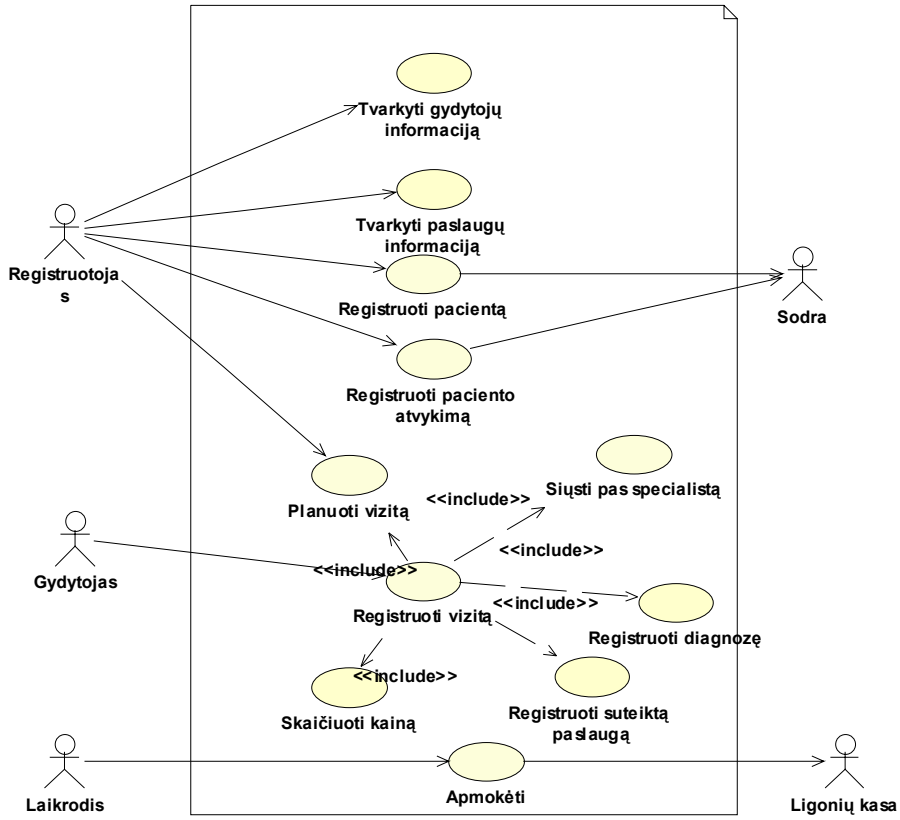
Smulkesni panaudojimo atvejai, kurie „įeina" į vieną panaudojimo atvejį ir turi būti visi įvykdyti, kad jį atitiktų, yra įtraukiami ryšiu <<include>>.

Ypatingiems, netipiniams atvejams yra naudojamas <<extend>> ryšys, toks panaudojimo atvejis išplečia kitą panaudojimo atvejį.

Apibendrinimo ryšys nusako paveldėjimą. Panaudojimo atvejis, susietas su kitu tokiu ryšiu, paveldi visą veiksmų seką iš kito panaudojimo atvejo, bet gali turėti ir savo papildomą elgseną.

Panaudojimo atvejai turi apibrėžti sistemai keliamas užduotis, tačiau tik esminį elgesį ir negali būti nei per daug dideli, nei perdaug specializuoti. Taip pat reikia sekti, kad panaudojimo atvejai atitiktų tik kuriamos sistemos reikalavimus.

Sudarant panaudojimo atvejus, juos detalizuojant, būtina nuolat tikrinti, ar jie neišeina už nagrinėjamos srities ribų, kad panaudojimo atvejai ir visas projektas per daug neišsiplėstų ir netaptų nebekontroliuojami. 7 pav. yra pateiktas panaudojimo atvejų modelio pavyzdys.



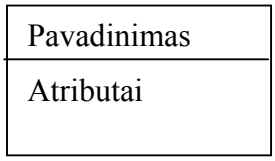
7 pav. Panaudojimo atvejų modelis

3.3.2 Esybių identifikavimas ir dalykinės srities klasių diagramos sudarymas

Dalykinės srities klasių diagramos tikslas – parodyti, kokia informacija yra manipuliuojama, vykdant verslo procesą. Toks modelis padeda geriau suprasti dalykinę sritį, pastebėti, kokie poreikiai dar netenkinami bei tarnauja kaip žodynas su dalykine sritimi nesusijusiems asmenims. Pradinis dalykinės srities modelis nusako tik pagrindines sąvokas ir jų tarpusavio ryšius, bet dar nedetalizuojami atributai, ryšių kardinalumai.

Dalykinės srities esybės gali būti identifikuojamos įvairiai, bet visada nagrinėjama, kokia informacija yra registruojama, išsaugoma, modifikuojama, pateikiama. Paprasčiausias radimo būdas – išrinkti daiktavardžius, kurie yra

naudojami dalykinės srities verslo kalboje. Galima pasinaudoti jau sudarytais panaudojimo atvejais. Kiekviena sąvoka (esybė) vaizduojama UML klase (8 pav.).



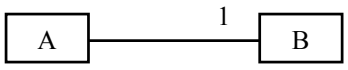
8 pav. Esybės vaizdavimas UML klase

Veiklos esybės gali būti asmenys, vietovės, įvykiai, daiktai ir sąvokos, kurie atspindi tiriamą dalykinę sritį. Dalykinė srities modelis apibrėžia esybes ir jų tarpusavio ryšius, kurios yra susijusios su nagrinėjamu verslo procesu ar kuriama sistema.

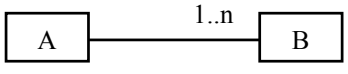
Kiekviena esybė yra abstrakti, t.y. ji aprašo tam tikrą egzempliorių aibę, o ne konkretų egzempliorių. Pvz. sistemoje bus daug gydytojų, pacientų, vizitų, apie kuriuos reikės registruoti informaciją. Visi klasės egzemplioriai turės tas pačias savybes – atributus ir ryšius, tačiau kiekvienas egzempliorius taip pat turi unikalų identifikatorių, kuris išskiria jį iš kitų.

Jeigu siekiama sudaryti detalesnį modelį, nustatomi esybių atributai – savybės. Tam užduodami paprasti klausimai: ką reikia žinoti apie kiekvieną esybę, kokias savybes reikia užregistruoti ir pan.

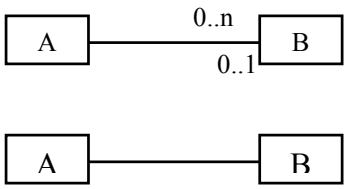
Dalykinės srities klasių diagramoje esybės tarpusavyje yra susijusios ryšiais. Ryšiai identifikuojami panašiai kaip atributai. Pvz., registruojant vizitą pas gydytoją, reikia žinoti, koks pacientas užsiregistravo vizitui. Bet pacientą netinka vaizduoti atributu, nes tai yra savarankiška sąvoka, kuri pati turi atributus. Atributais vaizduojamos tik tokios savybės, kurios vienos pačios neegzistuoja, jos būtinai turi priklausyti kokiam nors kitam objektui. Pvz., vardas, adresas, kodas, spalva, kiekis, kaina. Jei reikia žinoti informaciją apie kitą savarankišką esybę, tokia informacija išreiškiama ryšiu. Ryšiai turi kardinalumus. Kardinalumai išreiškia, kiek vieno tipo esybių gali turėti ryšius su kito tipo esybėmis. Gali egzistuoti 4 tipų ryšiai tarp esybių(9 pav.).



Kiekviena A esybė privalo turėti vieną ryšį su B esybe (privalomas ryšys).



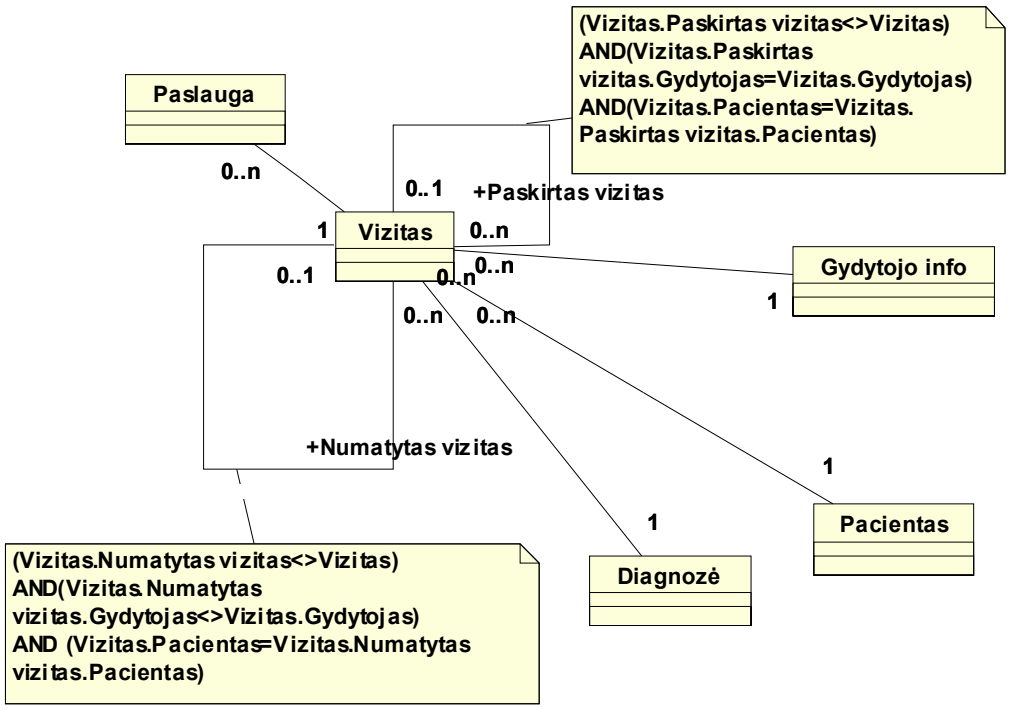
Kiekviena A esybė privalo turėti vieną ar daugiau ryšių su B esybe.



9 pav. Esybių tarpusavio sąryšiai

Kiekviena A esybė gali turėti nulį ar daugiau ryšių su B esybe.

Kiekviena A esybė gali turėti vieną ryšį su B esybe (bet neprivalo)



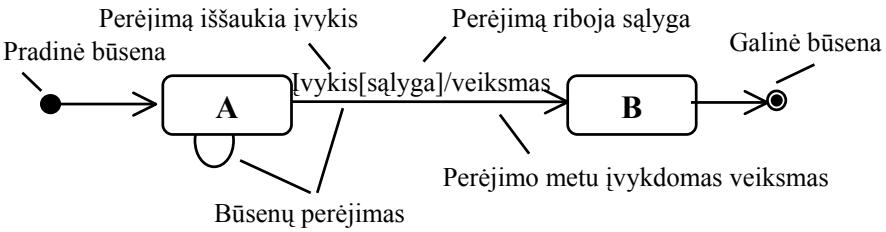
10 pav. Dalykinės srities klasių diagrama

3.3.3 Esybių būsenų diagramų sudarymas kiekvienai esybei

Esybių būsenų diagramos vaizduojamos UML būsenų diagramomis. Tokios diagramos parodo būsenas, kurias esybė gali turėti per savo gyvavimo laikotarpį. Įvairūs procesai ir įvykiai, būdingi nagrinėjamai dalykinei sričiai, laikui bėgant keičia esybių būsenas. Būsenų diagramomis nustatomos ir parodomos galimos esybių būsenos. Tai padeda išskirti procesus, kuriuose dalyvauja vienos ar kitos esybės arba nustatyti dar neidentifikuotus procesus. Be to, tokia dalykinės srities esybių analizė gali išryškinti ir anksčiau nepastebėtą procesų vykdymo nuoseklumą.

Būsenų diagrama (11 pav.) susideda iš būsenų, vaizduojamų stačiakampiais ir perėjimų, kurie vaizduojami rodyklėmis. Ant perėjimo taip pat galima žymėti įvykius,

sąlygas, kurie iššaukia perėjimą. Pradinė ir galinė būsenos taip pat turi specifinius pažymėjimus.

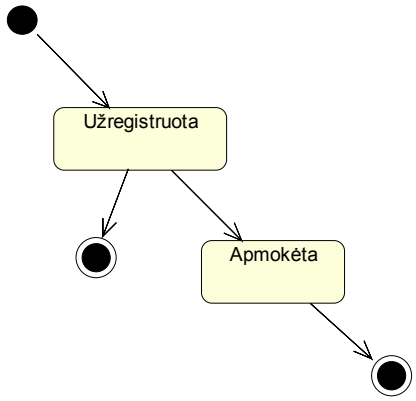


11 pav. Būsenų diagramos elementai

Objektas, būdamas tam tikroje būsenoje, tenkina kokią nors sąlygą, vykdo kokį nors veiksmą arba laukia kokio nors įvykio.

Esybės būsena nusakoma tam tikru jos atributų reikšmių rinkiniu. Vykdant veiklos procesus, daugelio dalykinės srities esybių būsenos modifikuojamos, bet kartais vienos esybės būsenos gali atspindėti visą proceso vykdymo eigą. Nors dažnai tokios esybės negalima išskirti, nes veiklos proceso metu modifikuojamos daugelio esybių būsenos. Į veiksmą įeinantis objektų srautas išreiškia esybės būseną prieš veiksmą, iš veiksmo išeinantis srautas rodo esybės būseną po veiksmo. Veiksmo metu gali keistis kelių esybių būsenos, atitinkamai gali būti keli įėjimo ir išėjimo srautai.

12 pav. dalykinės srities esybės „Paslauga “ būsenų diagramoje rodomos visos galimos esybės būsenos, pradedant nuo pradinės būsenos, kuri yra esybės egzemplioriaus sukūrimas, baigiant galine būsena. Ryšiai tarp būsenų rodo, į kokias būsenas galima pereiti iš kiekvienos būsenos. Pvz., jei paslauga yra būsenoje „Užregistruota”, po to ji gali būti apmokėta arba likti neapmokėta. Galinių būsenų gali būti kelios, nes esybės gyvavimas gali baigtis įvairiais žingsniais.

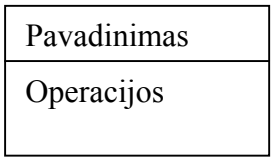


12 pav. Esybės „Paslauga” būsenų diagrama

3.3.4 Sąsajų identifikavimas kiekvienam panaudojimo atvejui

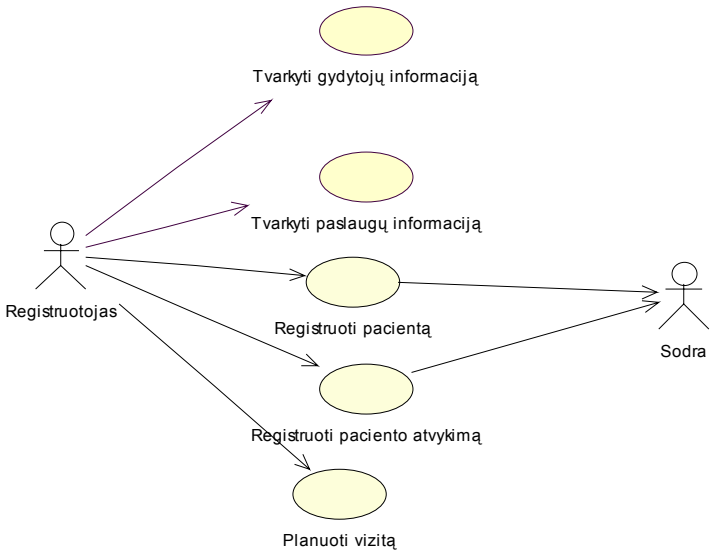
Panaudojimo atvejų diagrama yra neformali specifikacija, kokias užduotis sistema turėtų atlikti ir kaip vartotojas galėtų ja naudotis. Toks būdas apibrėžti pagrindines kuriamos sistemos galimybes yra praktiškas bendraujant su užsakovais, būsimais vartotojais ypač reikalavimų apibrėžimo proceso pradžioje, kai tokios specifikacijos pakanka. Tačiau siekiant sudaryti formalią ir nepriklausomą nuo sistemos projekto reikalavimų specifikaciją, būtina formalizuoti schemą, aprašant panaudojimo atvejus kaip kuriamos sistemos sąsajas. Išskiriamos sąsajos – klasės beveik atitinka panaudojimo atvejus. Galima sakyti, kad kiekvieno aktoriaus kiekvienam panaudojimo atvejui būna bent po vieną sąsają.

Sąsajos žymėjimas atitinka UML klasės žymėjimą.



13 pav. Sąsajos vaizdavimas UML klase

Sąsajos pavadinimas parenkamas panašus į panaudojimo atvejo, kuris yra formalizuojamas, pavadinimą. Patogiausia yra naudoti tą patį pavadinimą kaip ir panaudojimo atvejo, tik pridėti pažymėjimą „I". Pvz., jei turime panaudojimo atvejį „Registruoti Pacientą", sąsają galime pavadinti „IPacientoRegistravimas". Gali būti, kad vieną panaudojimo atvejį yra tikslinga vaizduoti keliomis sąsajomis. Taip būna pvz. tada, kai aktorius su sistemos pagalba atlieka tam tikrą užduotį, bet jos atlikimas taip pat yra susijęs su kitu aktoriumi, t.y. užduoties atlikimui būtina gauti informaciją iš kito aktoriaus. Tokiu atveju yra sukuriama sąsaja pagrindiniam panaudojimo atvejui ir dar viena sąsaja tai panaudojimo atvejo daliai, kurią atlieka kitas aktorius. Pvz. jei yra panaudojimo atvejis „Registruoti pacientą" (14 pav.), sudaromos 2 sąsajos – „IPacientoRegistravimas (nusako paciento registravimą, kurį atlieka registratorius) ir „ITikrinimasSodroje" (nusako paciento duomenų tikrinimą Sodroje, kuris susijęs su aktoriumi „Sodra").



14 pav. Panaudojimo atvejų diagrama

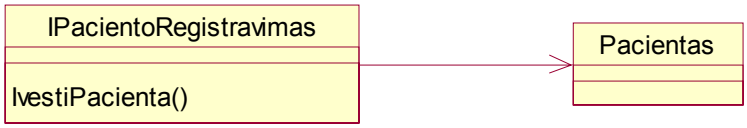
Jei panaudojimo atvejų diagramoje yra panaudojimo atvejai, susieti su kitais panaudojimo atvejais <<include>> ir <<extend>> sąryšiais, tokių detalizuotų panaudojimo atvejų atskirai sąsajomis išskirti nereikia. Tada panaudojimo atvejai, susieti anksčiau minėtais sąryšiais, tampa pagrindinio panaudojimo atvejo operacijomis.

Tačiau, jei panaudojimo atvejų modelyje yra panaudojimo atvejai, susieti <<include>> ar <<extend>> ryšiais su kitu panaudojimo atveju, bet kartu yra ir tiesiogiai susieti su aktoriais, juos būtina išskirti atskiromis sąsajomis. Tam galima vienam panaudojimo atvejui išskirti keletą sąsajų ir naudoti apibendrinimo ryšius.

3.3.5 Sąsajų esybių diagramų sudarymas

Sąsajų esybių diagramos yra sudaromos kiekvienai sąsajai ir parodo su kuriomis esybėmis sąsaja yra susijusi. Sąsaja yra susijusi su esybe, jeigu sąsajos operacijų vykdymui būtinas kurios nors esybės egzempliorius. Kol nėra išskirtos sąsajų operacijos, yra tikrinama, ar sąsaja (arba panaudojimo atvejis) yra susijusi su kuria nors dalykinės srities esybe. Tokiu būdu ieškant su sąsaja susijusių esybių ir naudojantis jau sudarytais modeliais, galima lengviau nustatyti ir sąsajų operacijas. Galutinėse sąsajų esybių diagramose (jos nuolat tobulinamos ir papildomos reikalavimų apibrėžimo proceso metu) sąsajų operacijose dalyvaujančios esybės yra vaizduojamos kaip operacijų argumentai.

Sąsajos esybių modelis susideda iš nagrinėjamos sąsajos (vaizduojamo UML klase) ir dalykinės srities esybių, kurios yra vaizduojamos su tarpusavio ryšiais. Jei sąsaja yra susijusi su visomis esybėmis, vaizduojama visa dalykinės srities klasių diagrama. Sąsajos sąryšis su esybe vaizduojamas rodykle (15 pav.).



15 pav. Sąsajos ir esybės sąryšis

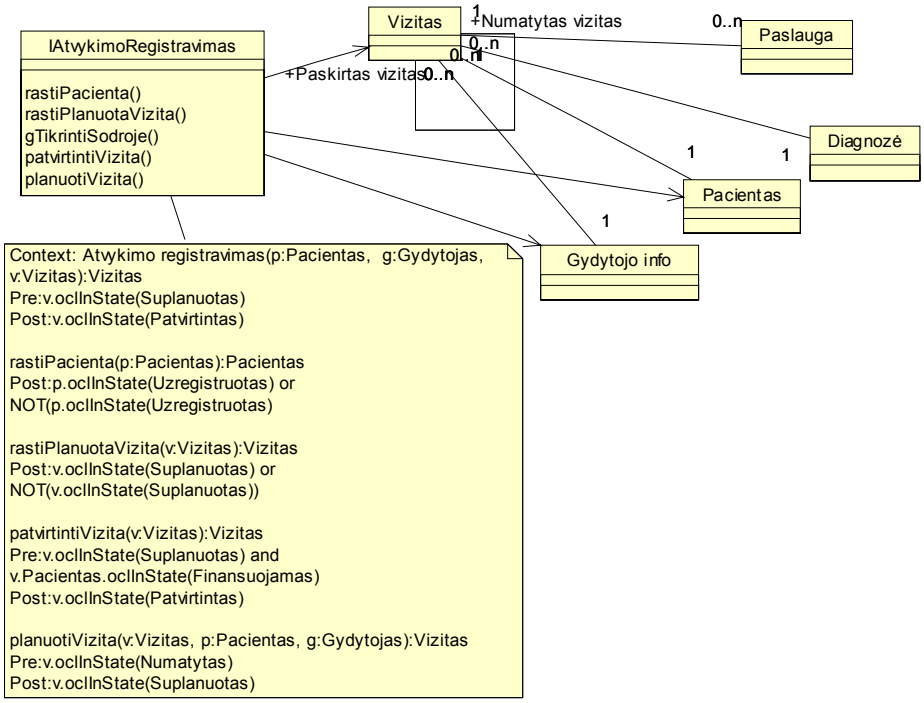
Sudarius pilną reikalavimų specifikaciją pagal siūlomą reikalavimų apibrėžimo metodiką, sąsajų esybių diagramoje būtina pateikti sąsajų, jų operacijų aprašus bei „prieš" ir „po" sąlygas OCL kalba. Sąsajų operacijų argumentai yra dalykinės srities esybių tipai, jų reikšmės – konkretūs objektai, su kuriais atliekamos operacijos. Apribojimuose naudojamos dalykinės srities esybių būsenų reikšmės, pavyzdžiui, *v.OclInState(Patvirtintas)* reiškia, kad vizitas turi būti patvirtintas. Taigi visas panaudojimo atvejis formalizuojamas sąsaja, kurią galima aprašyti kaip operaciją, kurios argumentai – atitinkami dalykinės srities objektų tipai:

Context: IAtvykimo registravimas (p:Pacientas, g:Gydytojas, v:Vizitas,d:Date):Vizitas
Post: v.OclInState(Patvirtintas) or Vizitas → not(exists (p:Vizitas) = v)

Sąsajos operacijos specifikuojamos, naudojant dalykinės srities objektų tipus. Čia taip pat nurodomos operacijų „prieš" ir „po" sąlygos:

Operations
rastiPacienta(p:Pacientas):Pacientas
planuotiVizita(v:Vizitas, p:Pacientas, g:Gydytojas, d:Date):Vizitas
Context planuotiVizita(p:Pacientas,g:Gydytojas,v:Vizitas,d:Date):Vizitas
Pre
v.OclInState(Numatytas) or Vizitas → not(exists(p:Vizitas | p.Gydytojas=g and p.Date=d)
Post
v.oclInState(Suplanuotas)

Pradinėje sąsajų esybių modelio versijoje sąsajos dar nedetalizuojamos OCL kalba. Kadangi reikalavimų apibrėžimo metodika yra iteratyvus procesas, prie sąsajų operacijų yra grįžtama, sudarinėjant kitus modelius, ypač sekų diagramos suteikia papildomos informacijos, reikalingos identifikuoti sąsajų operacijoms ir jas formalizuoti OCL kalba.



16 pav. Sąsajos esybių diagrama

3.3.6 Sąsajų sekų diagramų sudarymas

Sekų diagramos sudaromos kiekvienam panaudojimo atvejui arba kiekvienam panaudojimo atvejo vykdymo scenarijui. Jos vaizduoja objektų (šiuo atveju sąsajų) sąveikas laike panaudojimo atvejo žingsnių vykdymo metu. Sekų diagramos labai svarbios tuo, jog parodo, sukonkretina sąsajų atsakomybę už tam tikrą sistemos vykdomą užduotį ir taip įgalina išskirti sąsajų operacijas. Jos taip pat padeda nustatyti, ar teisingai vykdomi verslo procesai ir juos patobulinti.

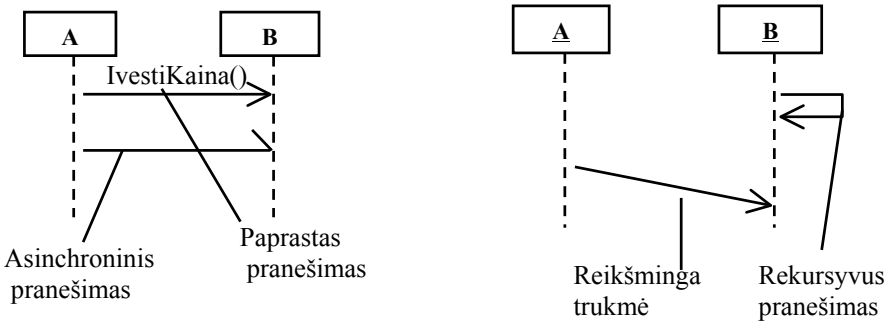
Sekų diagramą sudaro sąsajų ir aktorių gyvavimo linijos, pranešimai, įvykių pasirodymai. Pranešimai parodo, kaip sąsajos ir aktoriai sąveikauja tarpusavyje.

Sąsajos sekų diagramoje yra vaizduojamos kaip vertikalios linijos (dar vadinamos gyvavimo linijos) su sąsajos simboliu linijos viršuje.

Aktorių paprastai vaizduoja pirmoji gyvavimo linija sekų diagramoje (labiausiai nutolusi į kairę).

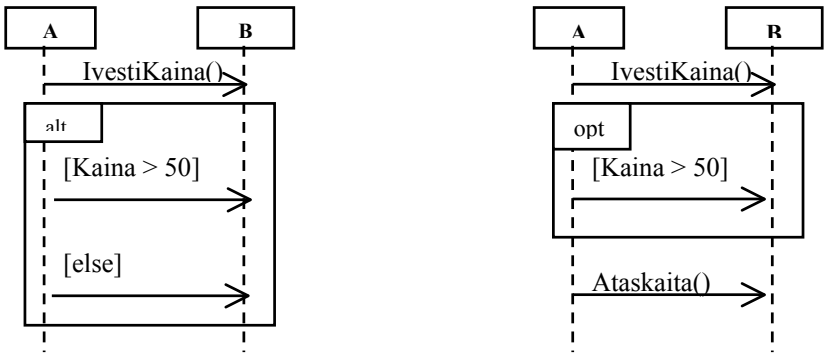
Pranešimai sekų diagramoje atspindi sąsajų ir aktorių tarpusavio sąveiką. Jie vaizduojami linijomis su rodyklėmis, kurios brėžiamos nuo vienos sąsajos (aktoriaus) prie kitos. Taip pat gali būti naudojamas asinchroninis pranešimas (suprantamas kaip signalas – objektas nelaukia atsakymo), rekursyvus pranešimas ar pranešimas, kuriuo

norima akcentuoti pranešimo trukmę. Ant linijos užrašomas pranešimo pavadinimas ir reikalingi parametrai.



17 pav. Sekų diagramų elementai

Pagal UML 2.0 versiją sekų diagramos buvo papildytos naujais elementais, tarp jų ir kombinuotais fragmentais (angl. *combined fragments*). Jie leidžia diagramose pavaizduti alternatyvas, pvz. **„if...then…else"** logika (18 pav.), ciklus ar scenarijų, kai išpildoma tam tikra sąlyga (18 pav.).

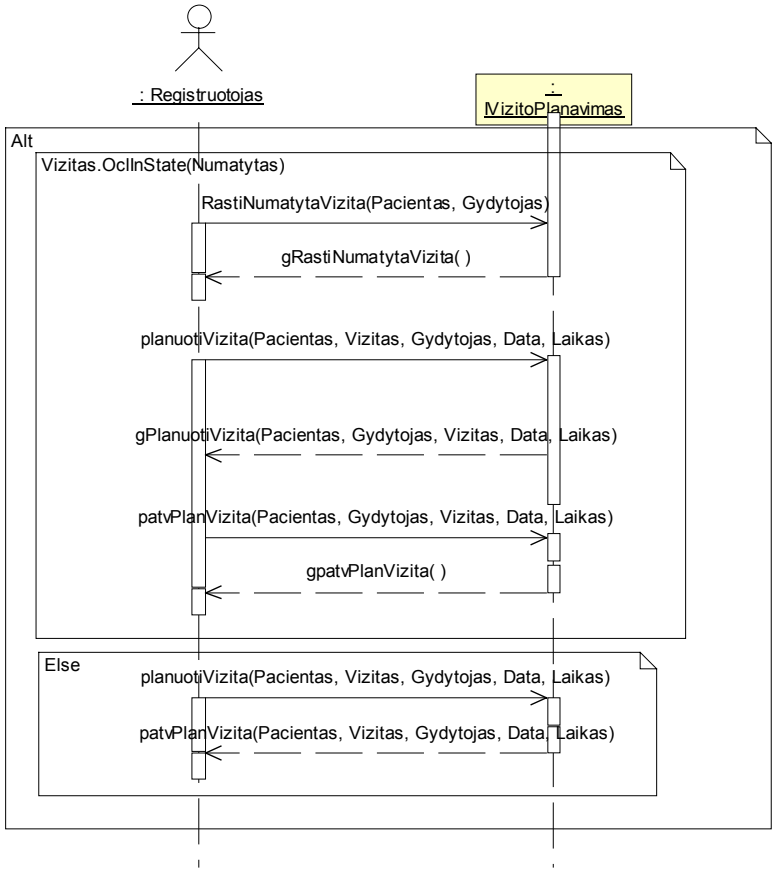


18 pav. Sekų diagramų kombinuoti fragmentai

Sekų diagramos paprastai sudaromos kiekvienam panaudojimo atvejo vykdymo variantui. Panaudojimo atvejų scenarijai nagrinėjami ir žymima pranešimais kokia informacija yra perduodama tarp aktorių ir sąsajų. Pradžioje sekų diagrama turi nurodyti tik nesudėtingą, nedetalizuotą vykdymo seką, kur būtų pavaizduoti pranešimai, bet nebūtinai jų parametrai. Vėliau sekų diagramos yra plečiamos, detalizuojamos.

Operacijos identifikuojamos, analizuojant pranešimus sekų diagramose. Kiekvienas panaudojimo atvejo įvykis arba operacija sekų diagramoje gali būti susiję su 4 įvykiais: siųsti „request" tipo pranešimą (kuris iškviečia operaciją), gauti

„request” tipo pranešimą, siųsti „response” pranešimą (atsakymas į operacijos iškvietimą), gauti „response” pranešimą.

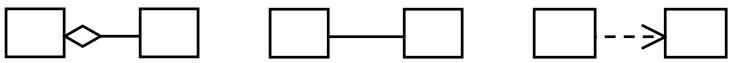


19 pav. Sekų diagrama

3.3.7 Sąsajų klasių diagramų sudarymas

Sąsajų klasių diagrama sudaroma sujungiant anksčiau identifikuotas sąsajas ryšiais į bendrą struktūrą. Kadangi panaudojimo atvejai formalizuojami sąsajomis, kurių žymėjimas atitinka UML klasių diagramų žymėjimus, ryšiai tarp sąsajų taip pat atitinka UML klasių diagramų ryšius.

Ryšiai gali būti identifikuojami, analizuojant sekų diagramas – jei objektai siunčia vienas kitam pranešimus, tarp atitinkamų sąsajų turi būti ryšys: asociacija, agregavimas arba priklausomybė.



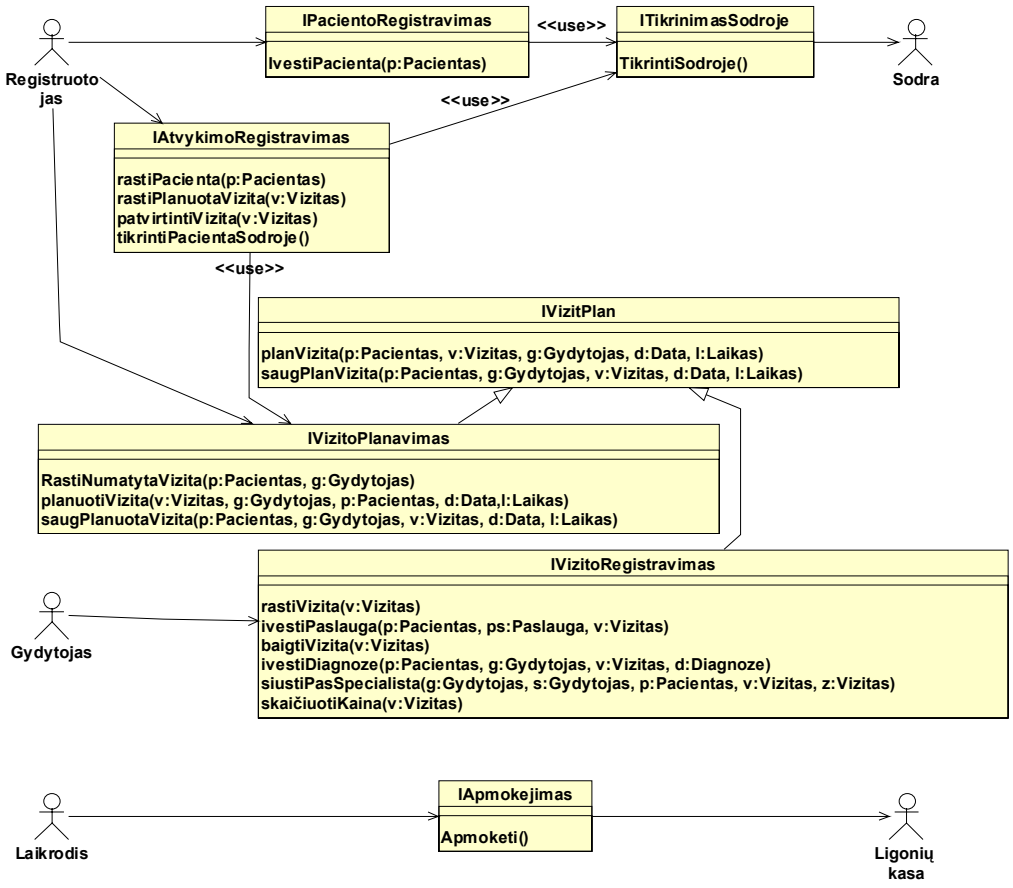
a) agregavimas b) asociacija c) priklausomybė

e) kompozicija f) apibendrinimas

20 *pav. Sąsajų tarpusavio sąryšiai*

Priklausomybės tarp klasių dažniausiai vaizduojamos tada, kai vienos klasės operacija naudoja kitą klasę kaip argumentą.

Struktūriniai ryšiai – **asociacijos** – sukuriami tarp savarankiškų klasių, jei tarp šių klasių objektų vyksta navigavimas duomenų požiūriu. Jei vienos klasės objektai susideda iš kitos klasės objektų – **agregavimas**, o jei agreguotų objektų gyvavimo ciklas priklauso nuo agregato gyvavimo ciklo – **kompozicija**. Jei vienos klasės objektai turi dalį tų pačių savybių, kaip kitos klasės objektai, galima ieškoti **apibendrinimo.**



21 pav. Sąsajų klasių diagrama

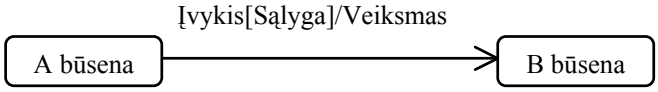
Dažniausiai sąsajų klasių diagramoje yra sutinkami apibendrinimo, asociacijų ryšiai. Apibendrinimo ryšiai yra dažni, kadangi jais yra keičiami panaudojimo atvejų diagramoje esantys <<include>> ar <<extend>> ryšiai.

3.3.8 Būsenų diagramų sudarymas kiekvienai sąsajai

Siekiant sudaryti pilną reikalavimų specifikaciją, būtina pateikti būsenų diagramas, sudarytas kiekvienai sąsajai. Būsenų diagrama aprašo visą galimą klasės, panaudojimo atvejo, posistemio, sąsajos ar kito modelio elemento elgesį.

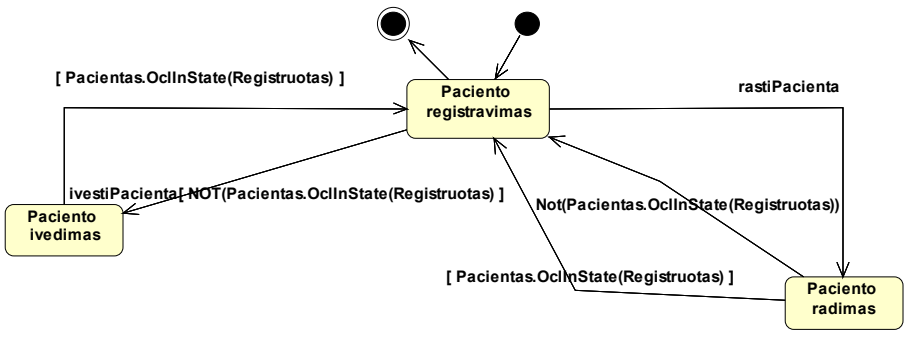
Sąsaja yra formalizuotas panaudojimo atvejis, todėl ji nusako tam tikrą veiksmų seką, kuri turi būti atlikta vykdant panaudojimo atvejį. Būsenų diagrama šiuo atveju padeda detaliai nusakyti atliekamą veiksmų seką, kurią galima formalizuoti perėjimus tarp būsenų aprašius OCL kalba.

Sudarant būsenų diagramas, galioja taisyklė: kiekvienas perėjimo veiksmas aprašo operaciją, įvykis – operacijos sužadinimo įvykį, pradinė perėjimo būsena ir saugumo sąlyga – operacijos prieš–sąlygą, galinė perėjimo būsena – operacijos po–sąlygą.



22 pav. Perėjimas tarp būsenų

Sąsajos būsenų diagramoje nurodomas tik veiksmas, kuris atitinka sąsajos operaciją, ir sąlyga, kuri nusako operacijos įvykdymo prieš–sąlygą. Būsenų diagramos sudaromos naudojantis anksčiau sudarytomis sąsajų klasių, sekų ir kitomis diagramomis. Būsenų diagramos elementai turi būti suderinti su kitų diagramų elementais (pvz. sąsajų operacijos atitinka būsenų perėjimų veiksmus, sąsajų prieš–sąlygos – būsenų perėjimų sąlygas).



23 pav. Sąsajos būsenų diagrama

4 PAVYZDYS, SUDARYTAS NAUDOJANT SIŪLOMĄ METODIKĄ

Dalykine sritimi, kuriai bus taikoma siūloma metodika, pasirinkta medicinos paslaugas pacientams teikianti poliklinika.

Poliklinikoje saugoma informacija apie gydytojus, jų specialybes, poliklinikos teikiamas paslaugas pacientams ir jų kainas bei pačius pacientus. Pirmiausia pacientai turi užsiregistruoti poliklinikoje, t.y. pateikti savo duomenis. Vėliau pacientas, norėdamas apsilankyti pas gydytoją, kreipiasi į registruotoją, kuris nurodo laiką ir datą, kada atvykti. Sekančiam vizitui pas gydytoją arba siuntimui pas naują specialistą gali užregistruoti ir pats gydytojas paciento vizito metu. Jei pacientas ateina neužsiregistravęs vizitui, registracija vykdoma vietoje ir gydytojas gali jį priimti, jei nėra kitų pacientų. Atvykus užsiregistravusiam pacientui, registruojamas paciento atvykimas, o vizito metu gydytojas užrašo diagnozę, suteiktą paslaugą, apskaičiuojama vizito kaina. Nustatytais laiko momentais (pvz., paskutinę mėnesio dieną) Sodros finansuojamiems pacientams suteiktas paslaugas apmoka ligonių kasos.

Šiame darbe naudojama trečios dalies 1 paveiksle pavaizduota reikalavimų struktūros dalis, kuri reikalinga, aprašant statines elgsenos savybes, tai yra, nenagrinėjami galimi dinaminiai sąsajų ir operacijų tarpusavio apribojimai ir nenaudojamos veiklos diagramos. Esamos OCL kalbos galimybės šiuo metu neleidžia aprašyti tokio tipo apribojimų. Taigi laikomasi tradicinio sąsajų specifikavimo būdo, pagal kurį sąsajos atvaizduojamos operacijų aibe, kiekvienai operacijai aprašoma signatūra, „prieš” ir „po” sąlygos.

Atvaizduojant panaudojimo atvejus sąsajomis, gali kilti klausimų, kaip elgtis su panaudojimo atvejų ryšių <<*include*>> ir <<*extend*>> ryšių stereotipais. UML 2.0 versijoje, kaip ir ankstesnėse, nėra apibrėžta, kaip įtraukti priklausomus panaudojimo atvejus į viršesniųjų panaudojimo atvejų elgsenos modelius. [5] buvo priimta, kad specifikuojat panaudojimo atvejus, susietus <<*include*>> ir <<*extend*>> ryšiais, bus naudojami sąsajų apibendrinimo ryšiai, griežtai laikantis *Liskov* substitucijos principo.

Šis darbas taip pat neapima nefunkcinių reikalavimų ir vartotojo sąsajos modelio, kurie neįeina į informacinės sistemos schemą. Toliau pateikiami reikalavimų specifikacijos fragmentai, iliustruojantys siūlomą metodiką.

4.1 Panaudojimo atvejų modelis

Panaudojimo atvejų modelis nusako pagrindines kuriamos sistemos galimybes (1 pav.). Sistemą naudoja registruotojas ir gydytojas. Registruotojas registruoja pacientus ir tvarko įvairią sistemoje saugomą informaciją, gydytojas registruoja konkretaus vizito duomenis. Ligonių kasos apmoka suteiktas paslaugas Sodros finansuojamiems pacientams, kurių finansavimas tikrinamas, kreipiantis į Sodros informacinę sistemą.

Buvo išskirti tokie aktoriai ir panaudojimo atvejai:

Registruotojas:

- Tvarkyti gydytojų informaciją
- Tvarkyti paslaugų informaciją
- Registruoti pacientą
- Registruoti paciento atvykimą
- Planuoti vizitą

Gydytojas:

- Registruoti vizitą:

Planuoti vizitą

Siųsti pas specialistą

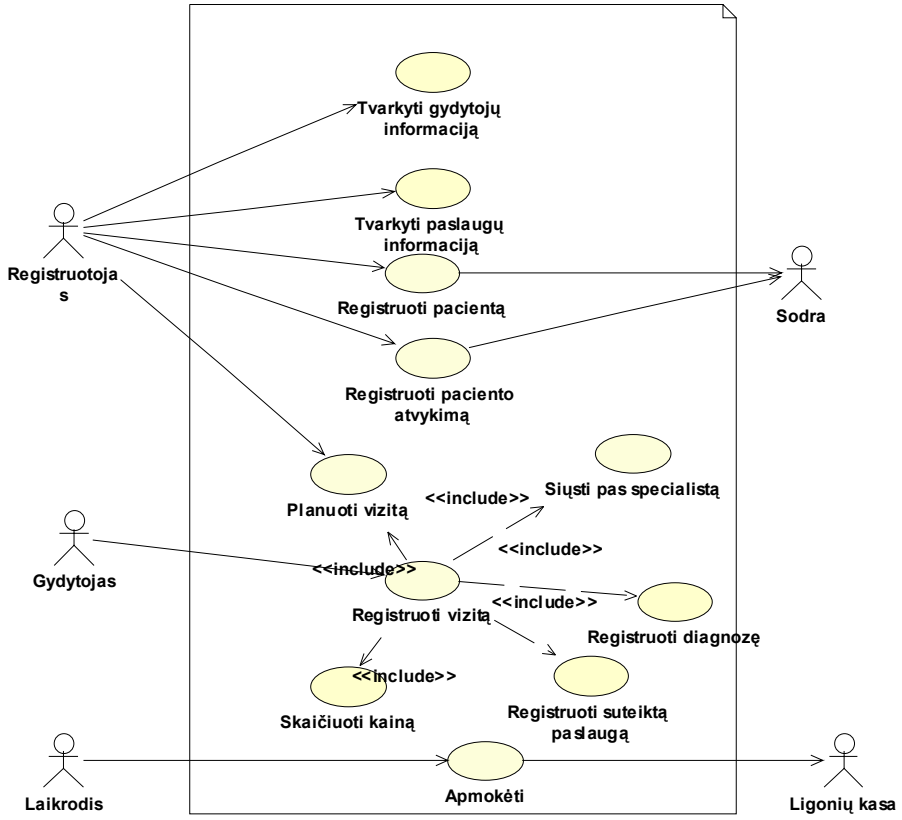
Registruoti diagnozę

Registruoti suteiktą paslaugą

Skaičiuoti kainą

Laikrodis:

- Apmokėti

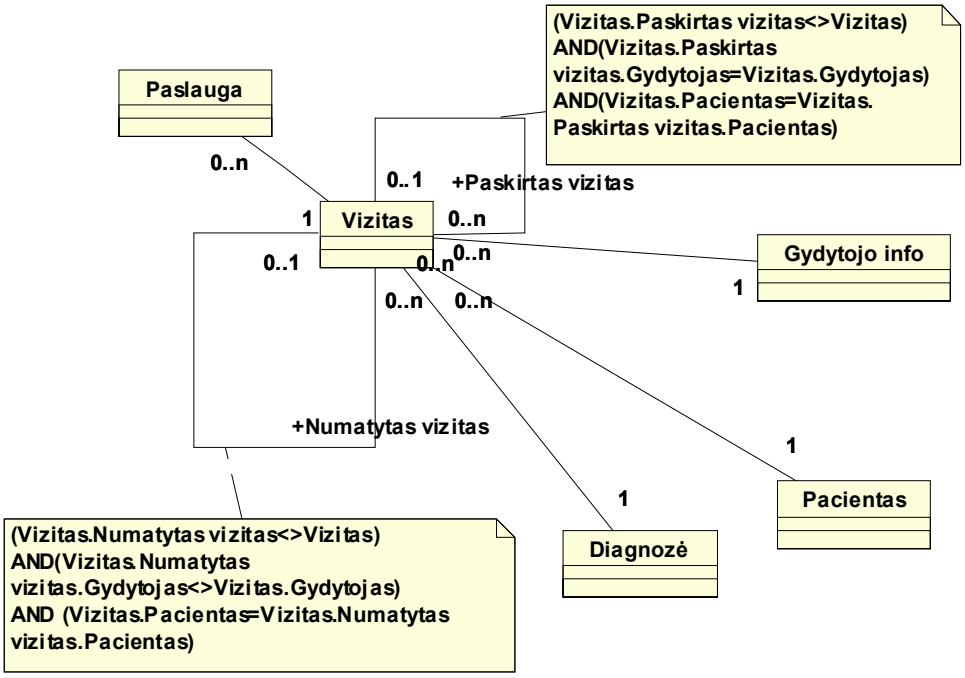


1 pav. Panaudojimo atvejų diagrama

4.2 Dalykinės srities klasių diagrama

2 paveiksle pateikiama dalykinės srities klasių diagrama rodo pagrindines klases, susijusias su tiriama sistema. Klasių apribojimai (invariantai) užrašyti OCL kalba.

Esybių sąraše yra vizitas, gydytojo info, paslauga, pacientas, diagnozė.



2 pav. Dalykinės srities klasių diagrama

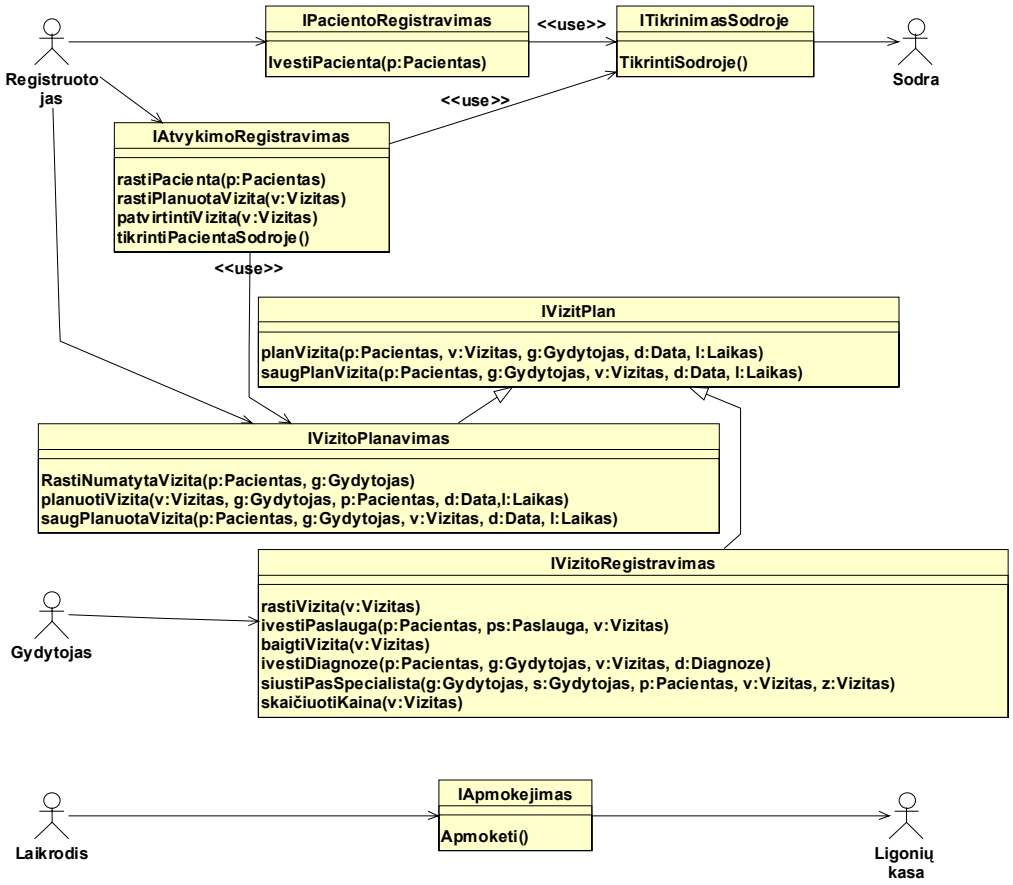
4.3 Sąsajų klasių diagrama

Sąsajų klasių diagrama projektavimo pradžioje atstoja projekto klasių diagramą. Čia yra išskiriamos sąsajos – klasės beveik atitinkančios panaudojimo atvejus ir identifikuojamos sąsajų operacijos.

Buvo išskirtos sąsajos:

- IPacientoRegistravimas
- ITikrinimasSodroje
- IAtvykimoRegistravimas
- IVizitoPlanavimas
- IVizitPlan
- IVizitoRegistravimas
- IApmokėjimas

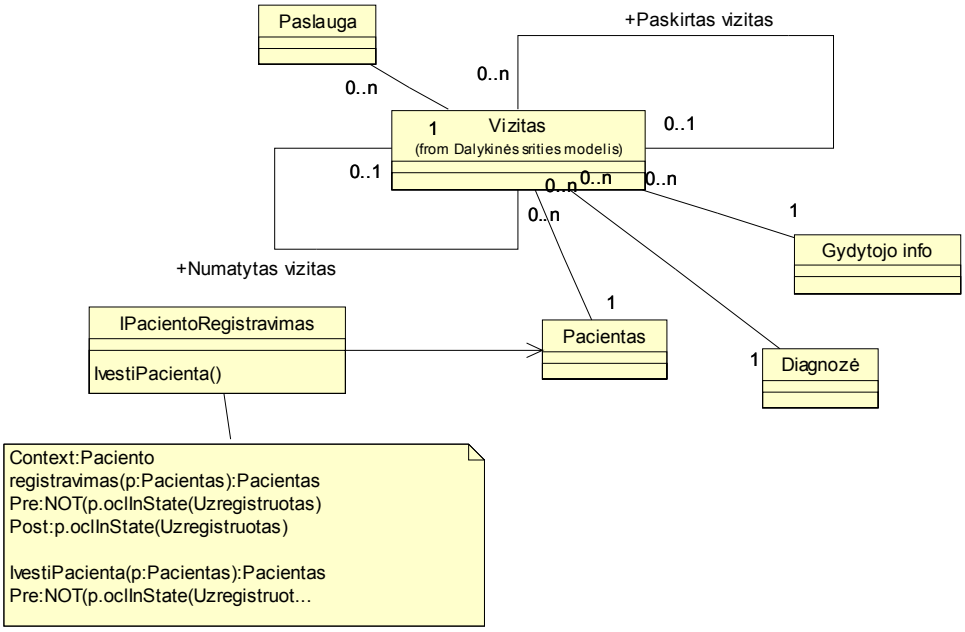
Taip pat nurodytos sąsajų operacijos.



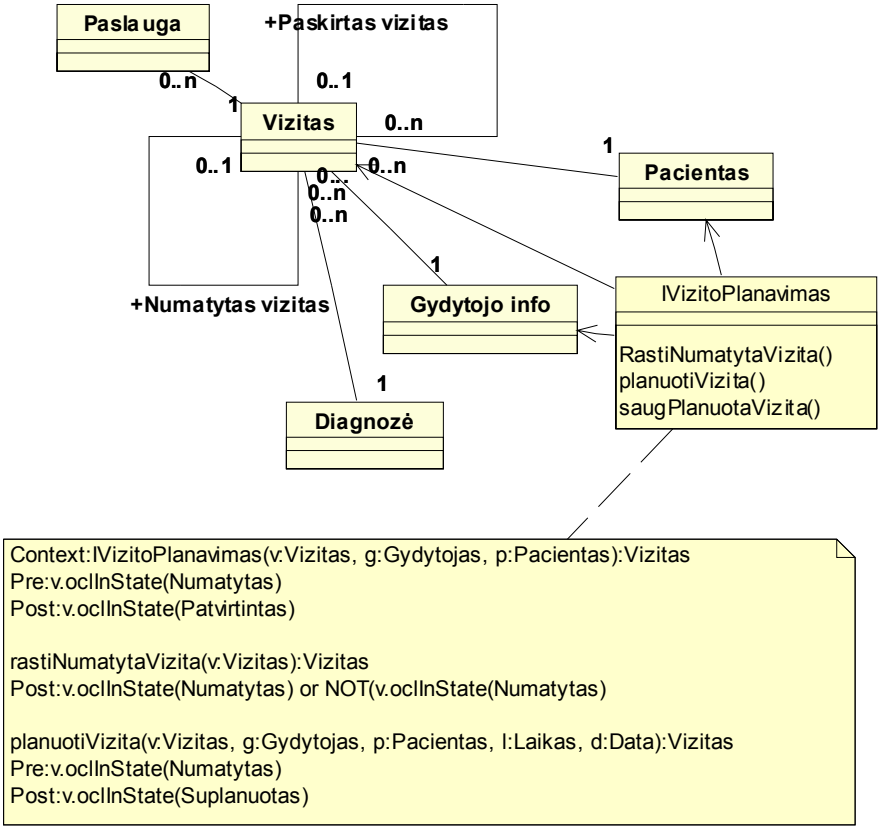
3 pav. Sąsajų klasių diagrama

4.4 Sąsajų esybių diagramos ir operacijų specifikacijos OCL kalba

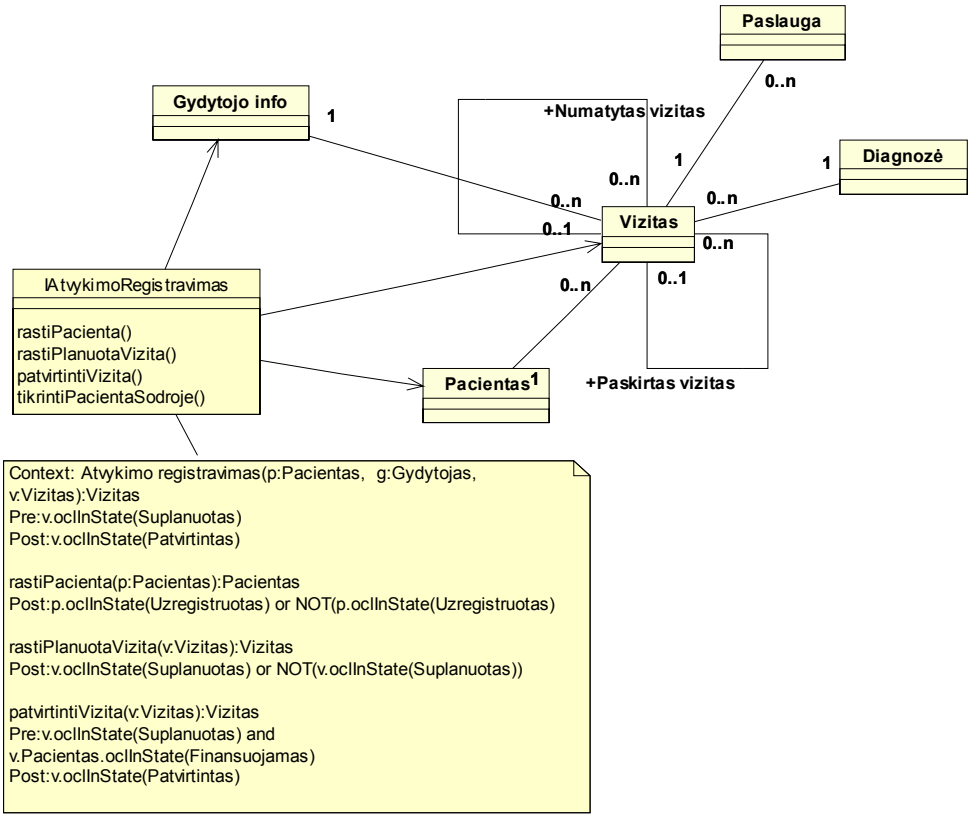
Praktinį pavyzdį taip pat sudaro sąsajų esybių diagramos kiekvienai sąsajai, kurios parodo, su kokiomis esybėmis yra sąsajos susijusios. Taip pat yra aprašytos sąsajų operacijos ir jų „prieš" ir „po" sąlygos OCL kalba.



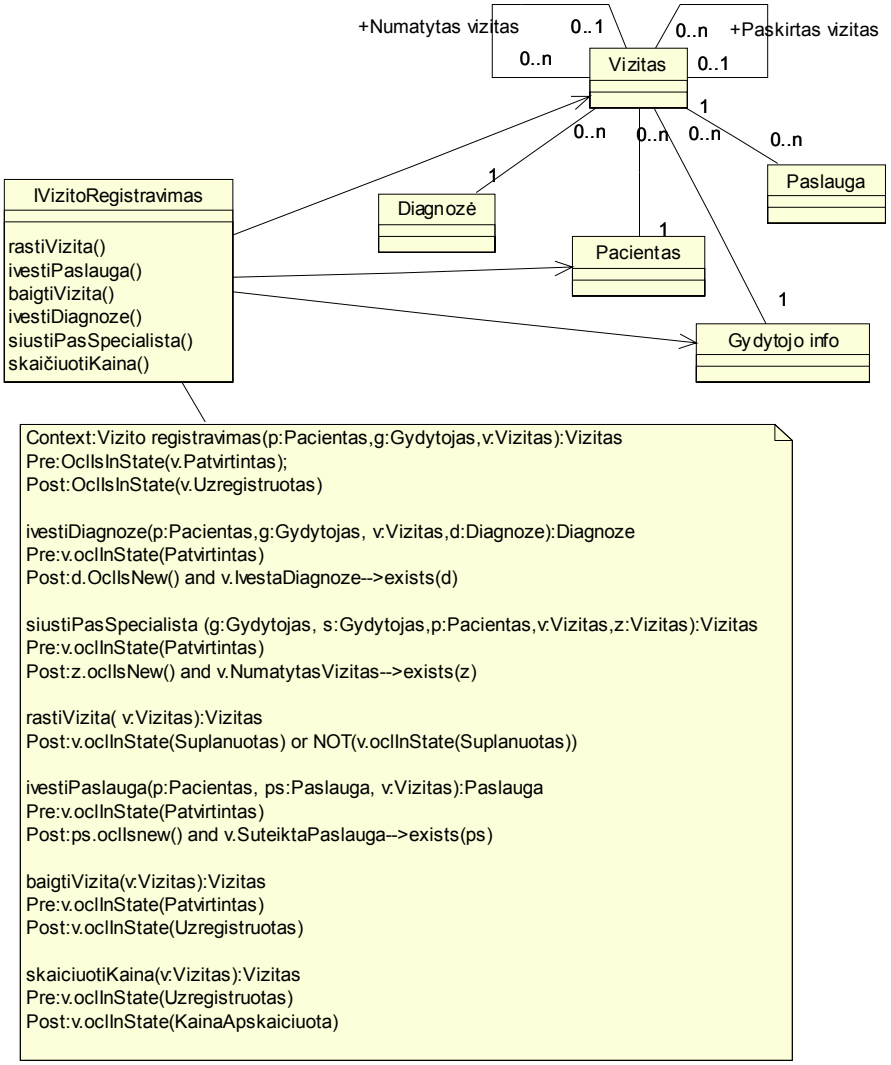
4 pav. Sąsajos „IPacientoRegistravimas" esybių diagrama ir operacijų specifikacijos OCL kalba



5 pav. Sąsajos „IVizitoPlanavimas" esybių diagrama ir operacijų specifikacijos OCL kalba



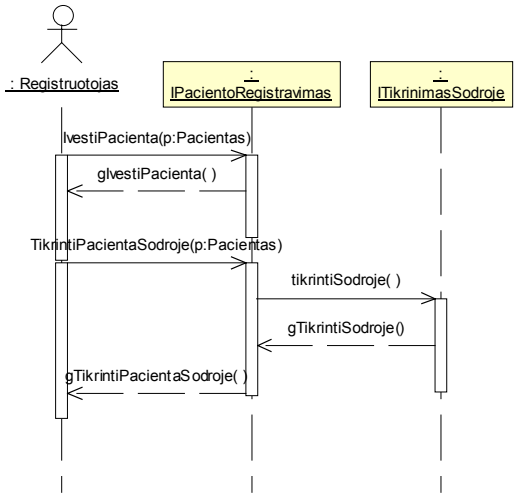
6 pav. Sąsajos „IatvykimoRegistravimas" esybių diagrama ir operacijų specifikacijos OCL kalba



7 pav. Sąsajos „IVizitoRegistravimas" esybių diagrama ir operacijų specifikacijos OCL kalba

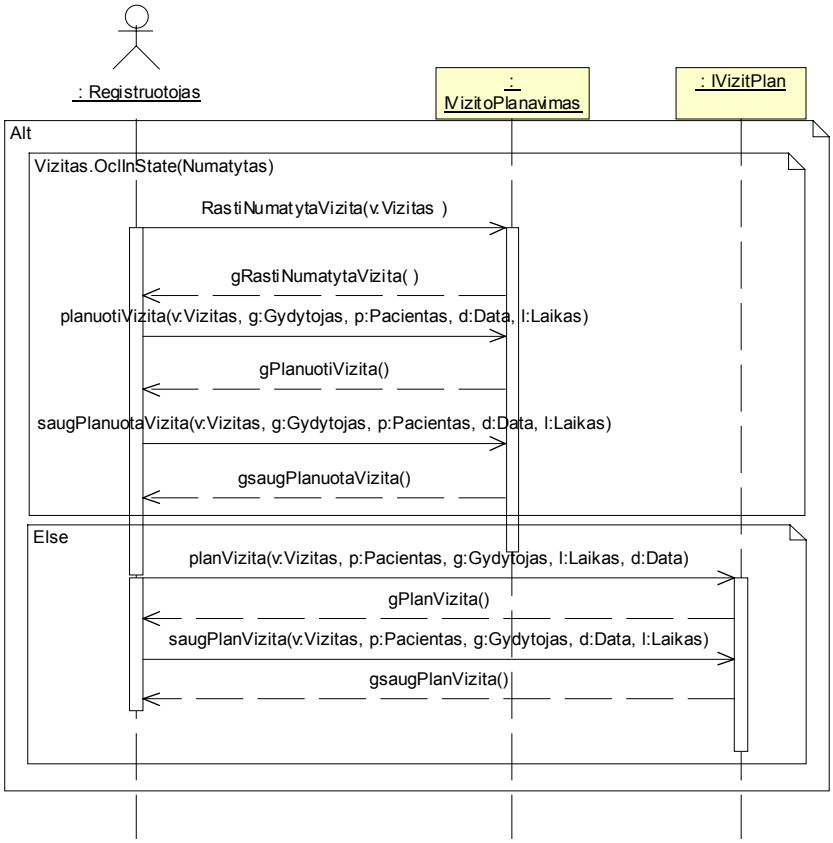
4.5 Sąsajų sekų diagramos

Nagrinėjant panaudojimo atvejus gautos sekų diagramos. Jos yra skirtos detalizuoti panaudojimo atvejus, išskirti operacijas ir objektus, kurie yra susiję su tomis operacijomis. Buvo sukurtos sekų diagramos pagrindiniams panaudojimo atvejams, kurių vykdymo eiga yra sudėtingesnė.



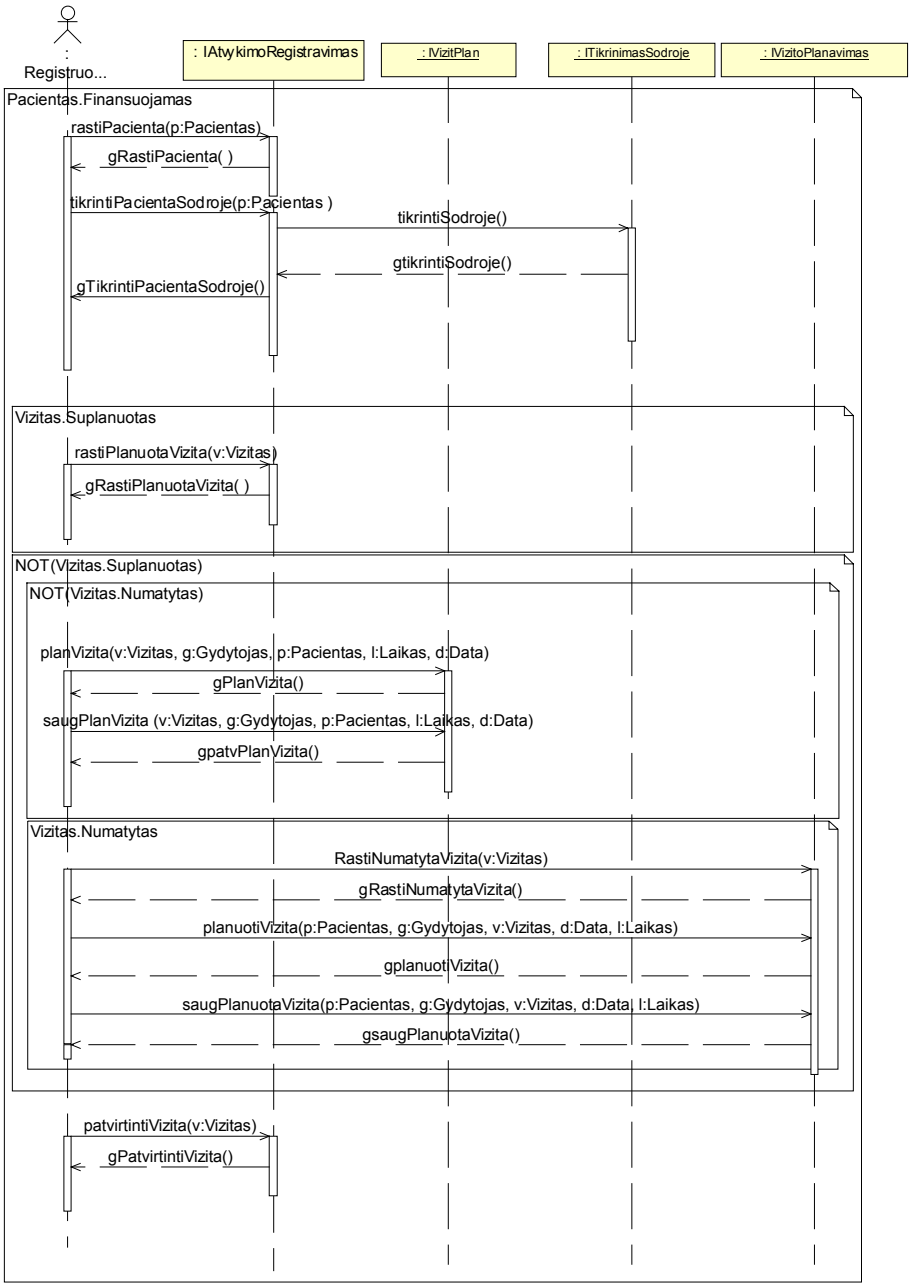
8 pav. Paciento registravimo sekų diagrama

8 pav. pavaizduotoje sekų diagramoje parodoma veiksmų seka, kaip turėtų būti registruojamas pacientas gydymo įstaigoje. Pirmiausia yra įvedami paciento duomenys, po to tikrinama, ar pacientas yra valstybės finansuojamas. Nepriklausomai nuo gauto rezultato, paciento duomenys yra saugomi sistemoje.



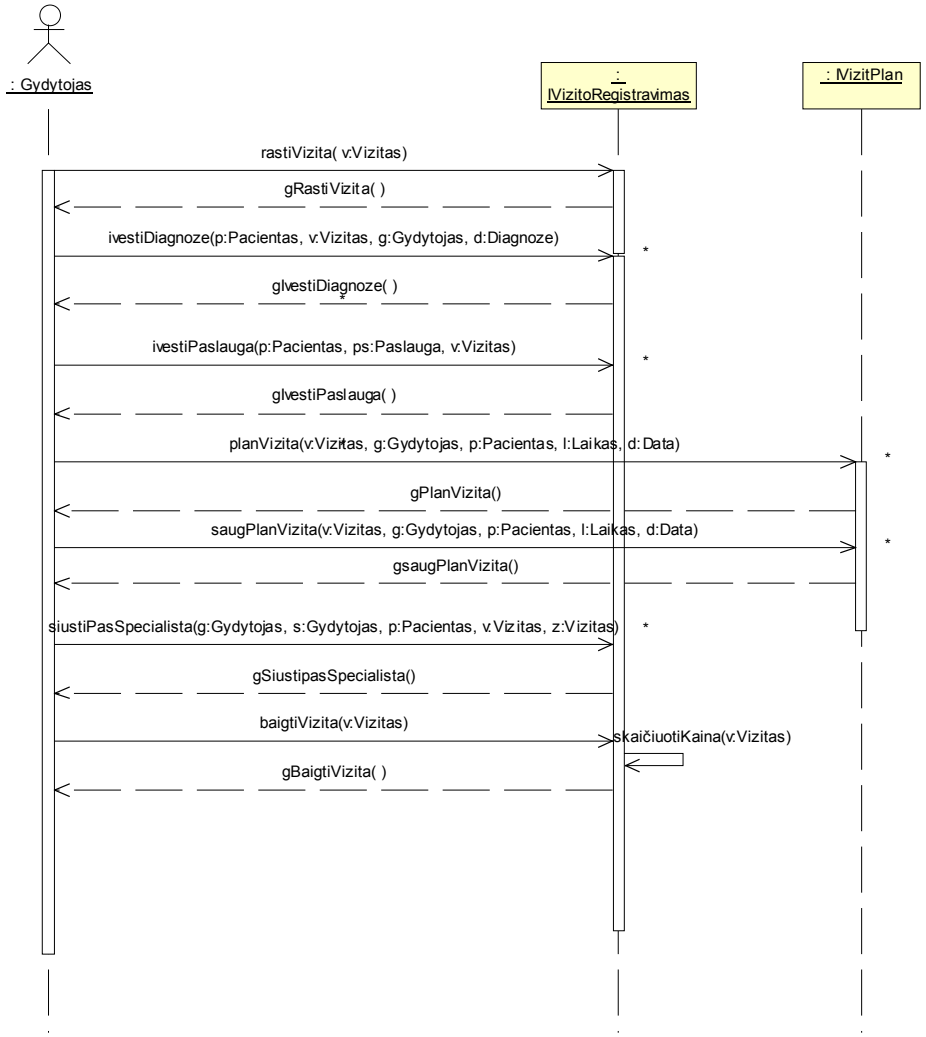
9 pav. Vizito planavimo sekų diagrama

Ši sekų diagrama (9 pav.) nurodo, kaip turėtų būti planuojamas vizitas. Priklausomai nuo to, ar vizitas yra jau numatytas, t.y. pacientas vizitą numatė telefonu ar lankydamasis pas kurį nors gydytoją, yra vykdomi atitinkami veiksmai. Jei vizitas yra numatytas, tada ieškoma numatyto vizito, t.y. pas kurį gydytoją ir kada numatytas vizitas, jei pacientas atėjo ir nori suplanuoti vizitą, kai nėra numatytas vizitas, tada jis planuojamas iš karto.



10 pav. Atvykimo registravimo sekų diagrama

Paciento atvykimo registravimo diagrama (10 pav.) parodo, kaip registruojamas paciento atvykimas pas gydytoją. Pirmiausia yra ieškoma užregistruoto paciento, vykdomas tikrinimas Sodroje ir pagal gautus rezultatus, atliekami tolesni veiksmai. Jei pacientas finansuojamas ir vizitas yra suplanuotas, yra randamas suplanuotas vizitas. Jei vizitas dar nėra suplanuotas, jis suplanuojamas vietoje ir patvirtinamas.



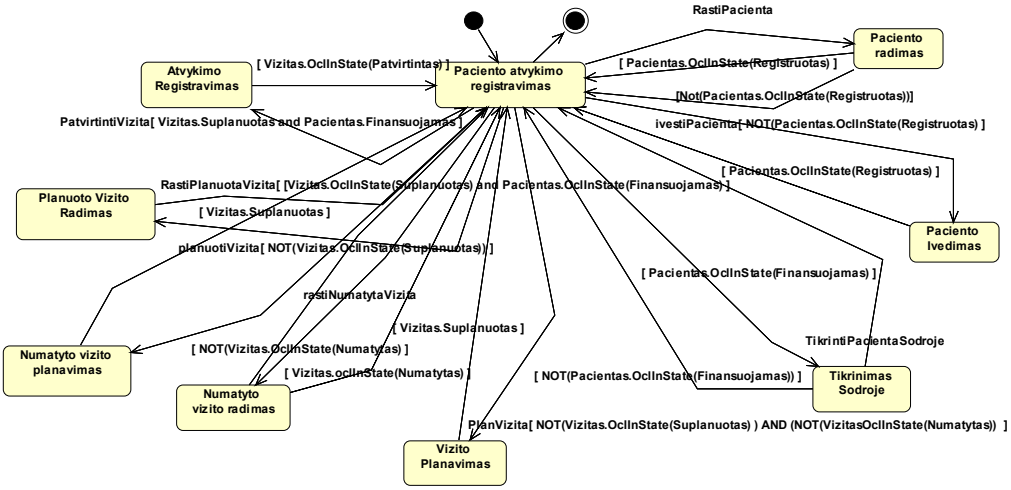
11 pav. Vizito registravimo sekų diagrama

Dar viena svarbi sekų diagrama yra vizito registravimo modelis (11 pav.). Čia yra parodoma, kaip vykdomas vizito registravimas, jau pacientui patekus pas gydytoją. Gydytojas randa vizitą, po to yra registruojama diagnozė ir paslauga. Jei pacientui daugiau konsultacijų nereikia, vizitas baigiamas. Tačiau gydytojas gali pacientą pasiųsti pas kitą specialistą, o taip pat numatyti paciento vizitą kitam kartui pas save.

4.6 Sąsajų būsenų diagramos

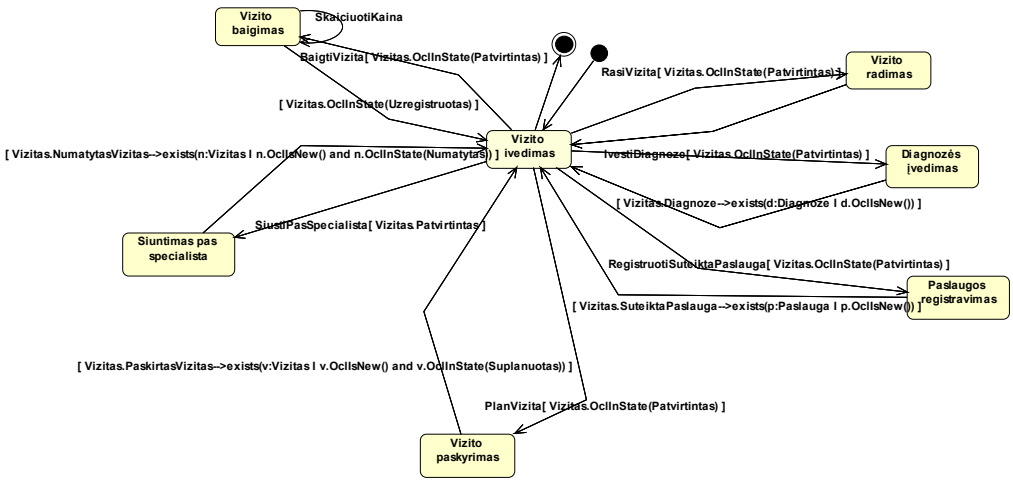
Toliau yra sudaromos būsenų diagramos, kurios nusako būsenas, į kurias sistema pereina vykdant vienokį ar kitokį veiksmą. Čia pavaizduotos būsenų diagramos neturi nuoseklios veiksmų sekos, kadangi sistema yra tokia, jog viskas priklauso nuo vartotojo, vartotojas pasirenka atitinkamus veiksmus.

Yra nurodomos būsenos, į kurias patenka sistema vykdant paciento atvykimo registravimą. Perėjimai, siekiant didesnio formalumo, aprašyti OCL kalba.



12 pav. Paciento atvykimo registravimo būsenų diagrama

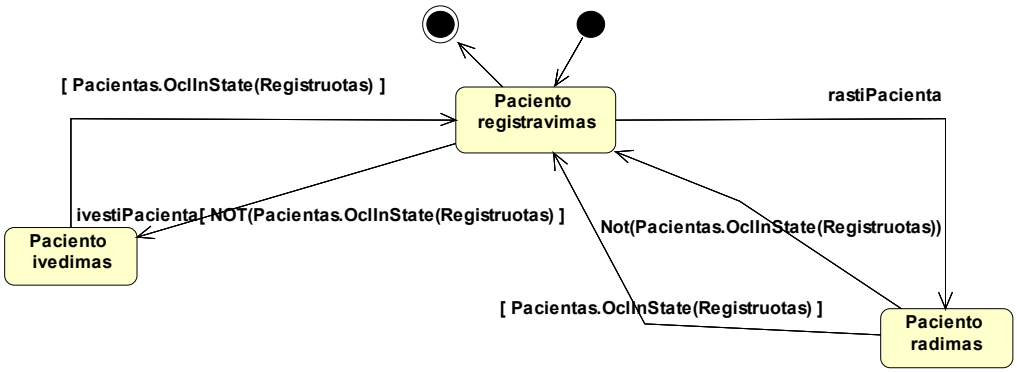
12 pav. pavaizduotoje diagramoje nurodytos būsenos, į kurias patenka sistema vykdant paciento atvykimo registravimą. Perėjimai, siekiant didesnio formalumo, aprašyti OCL kalba.



13 pav. Vizito registravimo būsenų diagrama

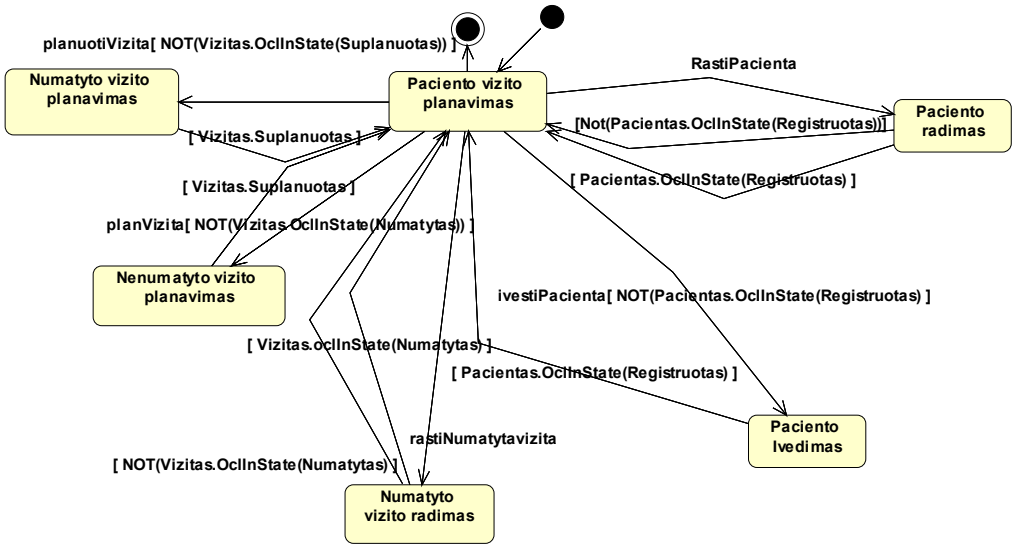
13 pav. pavaizduotoje diagramoje nurodytos būsenos, į kurias patenka sistema vykdant vizito registravimą. Perėjimai, siekiant didesnio formalumo, aprašyti OCL kalba.

.



14 pav. Paciento registravimo būsenų diagrama

14 pav. esančioje diagramoje nurodytos būsenos, į kurias patenka sistema vykdant paciento registravimą. Perėjimai, siekiant didesnio formalumo, aprašyti OCL kalba



15 pav. Vizito planavimo būsenų diagrama

Aukščiau esančioje diagramoje nurodytos būsenos, į kurias patenka sistema vykdant vizito planavimą. Perėjimai, siekiant didesnio formalumo, aprašyti OCL kalba.

5 IŠVADOS

- Išanalizuoti keli objektiniame projektavime naudojami reikalavimų apibrėžimo metodai ir pasiūlyta patobulinta reikalavimų modelio struktūra.
- Sudarytas vaizdinis reikalavimų modelis, aprašomas UML kalba su OCL apribojimais, išsamiai apibrėžiantis sistemos struktūrą ir elgseną nepriklausomai nuo būsimo projekto.
- Reikalavimų suderinimui sudaromos dalykinės srities ir sistemos sąsajų klasių, sekų ir būsenų diagramos, aprašomi apribojimai, naudojant dalykinės srities objektų būsenas.
- Vien iš reikalavimų metamodelio nėra aišku, kaip tokį rezultatą pasiekti ir kokia veiksmų eiga tam turi būti atlikta, todėl buvo pasiūlytas ir detaliai aprašytas reikalavimų apibrėžimo ir suderinimo procesas
- Atliekant darbą, buvo nagrinėjamos UML 2.0 ir OCL 2.0 versijos ir panaudotos kai kurios naujos šių kalbų savybės: (kombinuoti sekų diagramų fragmentai *(Combined Fragments),* protokolų būsenų mašinos, nauji OCL operatoriai).
- Metodika iliustruota apibrėžiant reikalavimus poliklinikos informacinei sistemai.
- Mokslinis naujumas atsispindi tame, jog pagal pateiktą metamodelį ir algoritmą galima UML diagramomis ir apribojimais išsamiai ir formaliai aprašyti kuriamos sistemos elgseną nepriklausomai nuo projekto:
    - Panaudojimo atvejai formalizuojami kaip sąsajos.
    - Aprašomos abstrakčios operacijos su signatūromis, „prieš” ir „po” sąlygomis.
    - Reikalavimų modelio elementai tarpusavyje suderinami.
- Konferencijoje „Informacinės Technologijos 2004” (2004 01 28–29) buvo pristatytas straipsnis „Informacinei sistemai keliamų reikalavimų metodika naudojant UML ir OCL”.
- Sudarytas metamodelis gali būti toliau tobulinamas keliomis kryptimis:
    - Formalizuoti perėjimą nuo sudaryto reikalavimų modelio prie projekto modelio (įvedant valdiklius ir kitus projekto modelio elementus).
    - Pritaikyti gautą metamodelį verslo procesams (valdymo srautams) įvedant veiklos diagramas.

6 LITERATŪRA

[1] OMG Model Driven Architecture. [interaktyvus] 1997–2003 [žiūrėta 2003 11 14]. Prieiga per internetą: http://www.omg.org/mda/ .

[2] **Nemuraitė L., Kavaliauskaitė L., Ambrazevičius E**. Towards Ensuring Consistency to UML Models // Informacijos mokslai. ISSN 1392–0561. Vilniaus universitetas, 2002, No. 24, p. 85–97.

[3] **Kosters G., Six H.W., Winter M.** Coupling Use Cases and Class Models as a Means for Validation and verification of requirements Specifications. – Requirements Engineering, *Springer–Verlag*, 2001, Volume 6, Issue 1, pp. 3–17.

[4] **Astesiano E., Reggio G.** Knowledge Structuring and Representation in Requirement Specification– Italy: DISI–Universita di Genova.

[5] **Čeponienė L., Nemuraitė L., Paradauskas B.** Design of schemas of state and behaviour for emerging information systems. Proceedings of the 1st Workshop on Emerging Database Research in Eastern Europe, co–located with VLDB 2003. Humboldt–Universität Berlin, Germany, September 8 2003, p. 27–31

[6] Unified Modeling Language: Superstructure. *2nd revised submission to OMG RFP*. 2003 [Žiūrėta 2003 11 15]. Prieiga per internetą: www.omg.org

[7] Response to the UML 2.0 OCL RfP. *OMG Document*. 2003 [Žiūrėta 2003 11 15]. Prieiga per internetą: www.omg.org

[8] **Nemuraitė L., Kavaliauskaitė L.** Informacinių sistemų projektavimo metodų tobulinimas ir automatizavimas, taikant UML // Informacinės Technologijos'2002. – Kaunas: Technologija, 2002, p. 408–414.

[9] **Nemuraitė L., Zaksaitė E.** Informacinei sistemai keliamų reikalavimų apibrėžimo metodika naudojant UML ir OCL // Informacinės Technologijos'2004. – Kaunas: Technologija, 2004, p. 512–520.

7 SUMMARY

In this work some progressive object–oriented requirement definition methods were analyzed (ICONIX, Reggio, Scores). Rules and principles of these methods were generalized and comprehensive content of requirements model was presented. Requirements definition must define state and behavior of target information system independently of future design. Requirements model was proposed where use cases are presented as interfaces together with associated classes of problem domain, operations are specified with signatures, pre and post conditions wich must be written in formal language OCL. Model was described using UML class, sequence diagrams, state charts and OCL constraints and is illustrated with examples from case study.

8 PRIEDAI

Konferencijoje „Informacinės Technologijos 2004” pristatytas straipsnis „Informacinei sistemai keliamų reikalavimų metodika naudojant UML ir OCL”.

INFORMACINEI SISTEMAI KELIAMŲ REIKALAVIMŲ APIBRĖŽIMO METODIKA NAUDOJANT UML IR OCL

Lina Nemuraitė, Eglė Zaksaitė
Kauno technologijos universitetas, Informatikos fakultetas, Informacinių sistemų katedra
Studentų g. 50-308, LT-3031 Kaunas

Šiame straipsnyje analizuojami egzistuojantys reikalavimų apibrėžimo metodai, pateikiama patobulinta reikalavimų modelio sudėtis bei pagal ją sudarytas reikalavimų apibrėžimo pavyzdys. Reikalavimų aprašas turi apimti sistemos būsenos schemą (dalykinės srities klases, jų tarpusavio ryšius ir apribojimus, arba invariantus) bei elgsenos schemą, kuri vaizduojama sistemos interfeisais, jų ryšiais ir apribojimais. Formalizuotas reikalavimų modelis toliau bus naudojamas atvaizdavimui į projekto modelį.

1 Įžanga

Pastaraisiais metais programinės įrangos, ypač informacinių sistemų, strateginė vertė įmonėms labai išaugo, todėl vis labiau yra siekiama pakelti IS kokybę, sumažinti kainą ir laikotarpį, per kurį reikalingas programinis produktas pasiekia rinką. UML (angl. *Unified Modeling Language*) buvo sukurta patenkinti naujai iškilusius rinkos reikalavimus ir šiuo metu informacinių sistemų (IS) projektavimas neįsivaizduojamas be automatizuoto projektavimo (angl. *Computer Aided Software Engineering* – CASE) įrankių.

IS metodai nurodo sistemos projektavimo veiksmų seką – kaip, kokia tvarka ir kokias UML diagramas naudoti projektavimo procese bei kaip tą procesą vykdyti. Daugelis jų remiasi kelių tipų diagramomis, aprašančiomis skirtingų sistemos aspektų savybes. Kiekvienos diagramos prasmę galima nusakyti atskirai, bet svarbu suprasti, kad kiekviena diagrama yra vieno sistemos modelio projekcija, todėl visos diagramos turi būti suderintos tarpusavyje.

Vis dėlto toks sistemos aprašymas yra gana painus, nes daugumoje diagramų pateikiama informacija iš dalies sutampa ir aprašo tuos pačius dalykus, tik skirtingais aspektais. Be to, nepatyrusio specialisto netinkamai naudojamos UML diagramos gali būti nenuoseklios, nepilnos ir nevienareikšmiškai aprašyti sistemą. Dabartiniai UML CASE įrankiai tokiais atvejais labai nedaug kuo tegali padėti projektuotojui. Yra siekiama kiek įmanoma formalizuoti modelių sudarymą, tam sukurta įvairių metodikų, tačiau dar egzistuoja ir neišnaudotų galimybių, kurios leistų patobulinti CASE įrankiais valdomus informacinių sistemų projektavimo metodus.

Siekis formalizuoti UML, pasiekti maksimalų modelių suderinamumą yra susijęs su nauja programinės įrangos kūrimo idėja, pagal kurią siekiama atskirti verslo arba programos logiką nuo technologinės realizacijos. Šis metodas dar vadinamas modeliu paremta architektūra arba MDA [1] (angl. *Model Driven Architecture*) ir laikomas ateities programinės įrangos kūrimo pagrindu.

MDA apibrėžia būdą, kaip atskirti sistemos funkcionalumo specifikaciją nuo jos įgyvendinimo tam tikroje platformoje. Tai įgalina tokį informacinių sistemų kūrimą, kai nuo realizacijos nepriklausomi modeliai naudojami sistemos realizacijų generavimui įvairiose platformose. Kuriant MDA metodu pagrįstas sistemas, pirmas žingsnis – nuo platformos nepriklausomo modelio PIM (angl. *Platform Independent Model*) sukūrimas, UML kalba. Modelis privalo pilnai išreikšti loginį vaizdą, kuriame vienareikšmiškai specifikuoti komponentai ir jų elgsena. Kadangi PIM apibrėžia tik būsimos sistemos logiką, sekančiame etape PIM logika turi būti įgyvendinta viename ar keliuose specifiniuose platformų modeliuose PSM (angl. *Platform Specific Model*). PSM yra išeities tekstai arba jų grafinis vaizdas išreikštas UML. PSM vėliau gali būti naudojamas generuoti konkrečiam programiniam kodui, tačiau viskas priklauso nuo PIM ir PSM kokybės. Kuo aiškesnė, griežtesnė PIM logika, tuo kokybiškesnis gaunamas PSM ir programinis kodas.

Šiame darbe nagrinėjamas projektavimo etapas, kuris turėtų sekti prieš PIM sudarymą – reikalavimų apibrėžimas, kuris turi būti nepriklausomas nuo sistemos projekto, tai yra, jis turi apibrėžti tik dalykinę sritį ir vartotojo reikalavimus būsimai sistemai. Pagal [5], reikalavimų (arba nuo projekto nepriklausomas) modelis turi pilnai aprašyti siekiamos sistemos būsenos ir elgsenos schemą. Tam siūloma naudoti panaudojimo atvejų, klasių, veiklos, būsenų bei sekų diagramas ir formalizuoti schemą, aprašant panaudojimo atvejus kaip kuriamos sistemos interfeisus. Reikalavimų specifikavime naudojama objektų apribojimų kalba OCL [7]. Darbe buvo stengiamasi ištirti ir naudoti naujos UML versijos savybes [6].

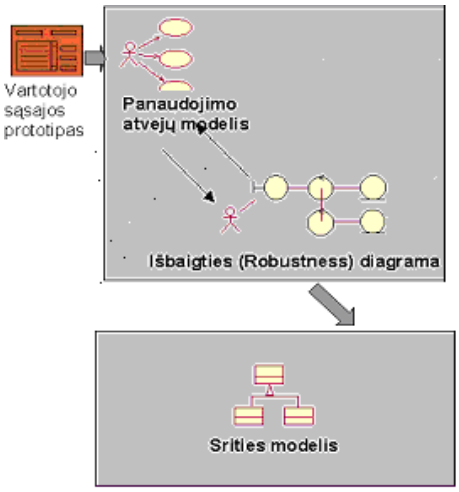
2 Reikalavimų apibrėžimo metodų analizė

2.1 Reikalavimų apibrėžimas *ICONIX* procese

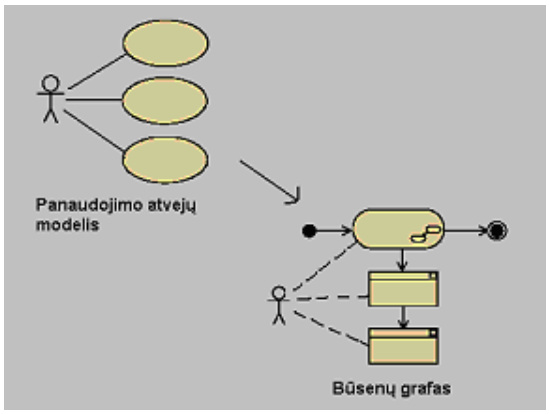
Pagal [8] pateiktą analizę *ICONIX* programinės įrangos kūrimo metodas pagrįstas minimaliu UML diagramų kiekiu ir efektyvia metodika, kurios dėka kūrimo procesas „nuo panaudojimo atvejų iki kodo" yra greitas ir efektyvus. *ICONIX* labai daug dėmesio skiria reikalavimų apibrėžimui. Reikalavimų specifikaciją

ICONIX procese sudaro panaudojimo atvejų modelis, išbaigties diagrama (ją sudarant yra patikrinamas ir pataisomas panaudojimo atvejų aprašymo glaustumas ir tikslumas bei sudaromos geresnės sąlygos detalesniam projektavimui) ir srities modelis. Taigi reikalavimų apibrėžimo etapo rezultatai yra išbaigtumo analizės metu gaunamas pilnai aprašytas ir teisingas panaudojimo atvejų modelis bei iš jo sudaromas srities modelis.

Šio proceso etapai: probleminės srities aprašymas, panaudojimo atvejų aprašymas, išbaigtumo (*Robustness*) analizė ir sekos diagramų sudarymas.



1 pav. ICONIX procese naudojama reikalavimų specifikacijos struktūra



2 pav. SCORES reikalavimų specifikacijos struktūra

ICONIX labai daug dėmesio skiria reikalavimų apibrėžimui, tačiau suderinamumas tarp panaudojimo atvejų, kuriuose užregistruoti reikalavimai, ir kitų modelių, ypač klasių diagramų, nėra užtikrintas.

2.2 *Scores* reikalavimų inžinerijos metodas

Šiame metode [3] akcentuojama reikalavimų specifikavimo problema, kai reikalavimus atspindi panaudojimo atvejų ir klasių diagramos, tačiau jos sudaromos naudojant skirtingus metodus bei pasižymi skirtingu abstrakcijos lygiu. Tokiam reikalavimų modeliui trūksta suderinamumo ir pilnumo. Metodo autoriai šią problemą sprendžia įvesdami būsenų grafus, kurie lemia vientisą, atsekamą perėjimą nuo panaudojimo atvejų prie klasių diagramų. Be to, naudojant tokį panaudojimo atvejų ir klasių diagramų susiejimą, galima užtikrinti modelių teisingumą.

Scores reikalavimų specifikavimo metodo pirmame etape yra patobulinami panaudojimo atvejai, kad būtų galima pasiekti kuo aiškesnę specifikaciją. Įvedami būsenų grafai, atspindintys panaudojimo atvejų elgseną.

Būsenų grafas *Scores* reikalavimų specifikavimo metode apibūdinamas:

- netuščia viršūnių (t. y. įvykių) aibe S;
- kryptingų lankų aibe $E \subseteq S \times S$ (perėjimai);
- pradiniu įvykiu, iš kurio eina lankai į kitas gretimas viršūnes (sekančius įvykius);
- netuščia galutinių veiksmų aibė $SF \subseteq S$.

UML panaudojimo atvejų terminologijoje panaudojimo atvejį aprašo aibė UML scenarijų, kurie vaizduoja konkretų atitinkamo veiksmo vykdymą. Scenarijus čia apibrėžiamas kaip įvykių seka, t.y. kelias per būsenų grafą, kuris prasideda pradiniu veiksmu ir tęsiamas atsižvelgiant į sąlygas, kurios tikrinamos įvykdžius būsenų grafo įvykį, kol pasiekiamas paskutinis veiksmas.

Vėliau būsenų grafais paremta panaudojimo atvejų specifikacija gali būti papildyta. Tam, kad būtų galima naudoti būsenų grafus panaudojimo atvejų modeliavimui, jie yra patobulinami įvedant 3 informacinių srautų, susiejančių skirtingas būsenas, tipus: konteksto ir sąveikos informaciją atspindi stereotipai «*contextual action*» ir «*interaction*», aktorių susiejimui su įvykiais naudojamas «*actor in action*» stereotipas, o «*macro action*» atspindi «*include*» ir «*extend*» sąryšius būsenų grafuose.

Sekančiame etape yra sudaromas klasių modelis. Kiekvienam panaudojimo atvejui yra nustatomos klasės, kurios susijusios su jo vykdymu. Tokia klasių aibė yra vadinama konkretaus panaudojimo atvejo klasių vaizdu, kuris gali būti pavaizduotas UML bendradarbiavimo diagrama. Analogiškai klasių vaizdas gaunamas ir būsenų grafo veiksmams, tik čia kiekvieną veiksmą būsenų grafe turi atitikti operacija klasių modelyje. Tokia klasė, kurioje yra operacija atitinkanti veiksmą iš būsenų grafo, vadinama šaknine klase, o jos operacija – šaknine operacija.

Sekančiame etape sudaryti modeliai yra tikrinami – atliekamas validavimas ir verifikavimas. Validavimo metu tikrinama, ar visi reikalavimai yra užfiksuoti, ar jie atitinka vartotojų norus ir tikslus. Verifikavimo metu atliekamas panaudojimo atvejų ir klasių modelio suderinamumo tikrinimas. Toks tikrinimas yra dviejų lygių procesas.

2.3 Reggio reikalavimų inžinerijos metodas

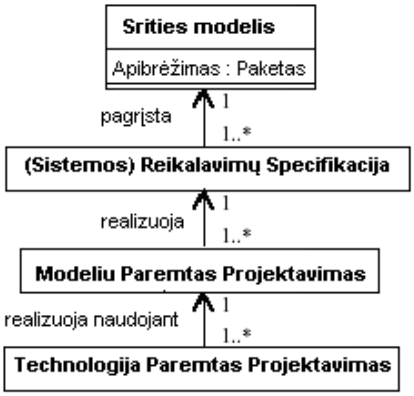
Šio metodo [4] tikslas – sukurti patobulintą ir griežtesnę reikalavimų specifikaciją, kuri leistų lengviau patikrinti įvairių modelių suderinamumą. Vienas esminių bruožų yra tai, jog sudaroma keletas su kuriama sistema susijusių vaizdų, be to, metodas remiasi panaudojimo atvejų diagramomis. Kartais yra nukrypstama ir nuo tradicinių objektinio projektavimo idėjų, bet tuo pačiu apjungiamos kitų metodų mintys, pvz. struktūrinės analizės. Išskiriamos 3 pagrindinės idėjos:

Visiškas srities modelio (angl. *Domain Model*) atskyrimas nuo sistemos aprašo.

Sistemos atskyrimas nuo jos aplinkos, kuris formalizuojamas konteksto vaizdu (angl. *Context View*). Konteksto vaizdas kartu su panaudojimo atvejais yra pagrindas reikalavimų apibrėžimui.

Abstrakčios būsenos (angl. *Abstract State*) naudojimas. Vietoje idėjos vaizduoti daug neprivalomų panaudojimo atvejų būsenų, yra apibrėžiama abstrakti būsena, kuri leidžia išreikšti abstrakčius sistemos ir jos konteksto sąveikos reikalavimus. Šioje stadijoje nenaudojama objektinės sistemos struktūra, kadangi jos dar nereikia. Tolesniuose projektavimo etapuose toks sistemos vaizdavimas išnyks, nes bus pereinama prie modeliais paremtos architektūros bei technologija paremtos architektūros.

Akivaizdu, kad Reggio reikalavimų inžinerijos metodas yra suderinamas su šiuolaikiniu programinės įrangos kūrimo principu atskirti verslo arba programos logiką nuo technologinės realizacijos.



3 pav. Šiuolaikinės programinės įrangos kūrimo procesas Reggio požiūriu

Kuriant reikalavimų specifikaciją pagal šį metodą, priimama, kad srities modelis jau yra sukurtas ir pateiktas UML diagramų pavidalu.

Iš srities modelio sudaroma specifikacija susideda iš skirtingų sistemos vaizdų ir žodyno (9 pav.). Konteksto vaizdas parodo sistemos kontekstą, t.y. esybes, kurios gali sąveikauti su sistema ir būdus, kaip yra sąveikaujama. Toks detalus sistemos ir jos aplinkos atskyrimas padeda išvengti painiavos tarp to, kas jau egzistuoja ir turi būti kuo tiksliau aprašyta (aplinka), ir to, ką reikia sukurti (sistema) bei užregistruoti reikalavimus.

Panaudojimo atvejų vaizdas (angl. *Use Case View*) standartiškai parodo visus sistemos naudojimo būdus ir tikslus.

Vidinis vaizdas (angl. *Internal View*) abstrakčiai aprašo vidinę sistemos struktūrą, kas dar vadinama abstrakčia sistemos būsena. Ji leidžia labai tiksliai apibrėžti panaudojimo atvejų elgseną pagal tai, kaip kiekvienas iš jų skaito ir atnaujina abstrakčią būseną.

Žodyne yra tiksliai aprašomos visos įvairiuose sistemos vaizduose aptinkamos esybės.

Esminiai ir svarbiausi modeliai yra sistemos vidinis vaizdas ir konteksto vaizdas. Jie padeda užtikrinti panaudojimo atvejų ir visos reikalavimų specifikacijos suderinamumą.

Reikalavimų specifikacijos sudarymo procesą galima suskirstyti į uždavinius:

- Panaudojimo atvejų diagramos sudarymas
- Žodyno pradinės versijos sukūrimas
- Pradinio konteksto vaizdo sudarymas
- Konteksto vaizdo išplėtimas

- Pradinio sistemos vaizdo sudarymas
- Atskirų vaizdų kiekvienam panaudojimo atvejui sukūrimas ir žodyno, vidinio bei konteksto vaizdų atnaujinimas

2.4 Išnagrinėtų modelių tinkamumas pasirinktam uždaviniui

Apžvelgtuose programinės įrangos kūrimo metoduose įvairiai sprendžiamas UML diagramų suderinamumas. Nors ne visas išnagrinėtų metodų idėjas eina panaudoti mano sprendžiamam uždaviniui, tačiau atskirus jų elementus galima pritaikyti sudarant reikalavimų suderinimo su UML klasių modeliu metodiką.

Scores reikalavimų inžinerijos metodas pasižymi aukštu pradinių vartotojo reikalavimų išpildymo laipsniu, tačiau yra gana sudėtingas ir reikalauja nemažai darbo, be to, neatsižvelgiama į gauto projekto klasių modelio kokybę (pvz. klasių tarpusavio susietumo pobūdį). Tačiau tinkama pasirodė panaudojimo atvejų ir klasių modelio susiejimo idėja pagal panaudojimo atvejų žingsnius.

Reggio reikalavimų inžinerijos metodas neapibrėžia aiškaus būdo, kaip pereinama nuo panaudojimo atvejų prie projekto klasių diagramų, taip pat yra gana sudėtingas ir reikalaujantis nemažai darbo. Tačiau projektavimas, paremtas panaudojimo atvejais, duomenų žodynu, dalykinės srities modeliu, yra perspektyvus, siekiant geresnio modelių suderinamumo.

3 Reikalavimų apibrėžimo struktūra

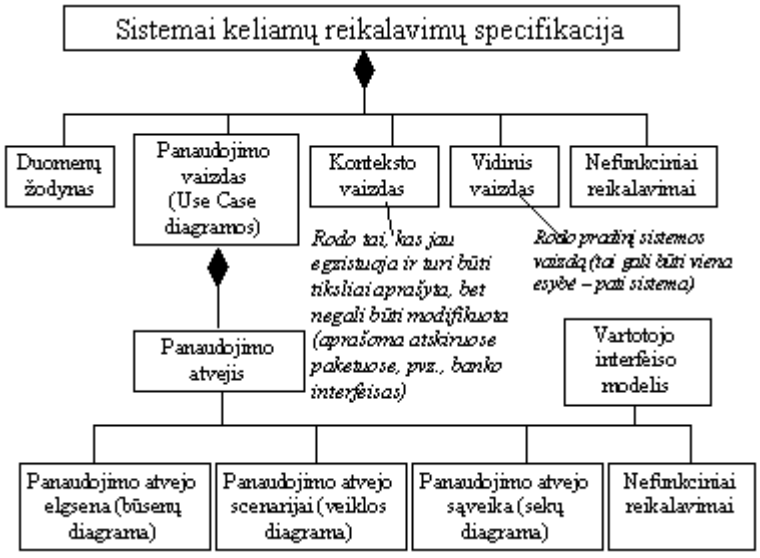
Išnagrinėjus egzistuojančius reikalavimų apibrėžimo sprendimus ir siekiant sudaryti kuo pilnesnį, tarpusavyje suderintą reikalavimų apibrėžimo modelį, pagrindu buvo pasirinkta Reggio reikalavimų specifikacija, kuri buvo šiek tiek papildyta.

Reikalavimų modelį sudaro duomenų žodynas, kurį galima apibrėžti kaip dalykinės srities klasių diagramą, nusakančią pagrindines, su projektuojama sistema susijusias esybes.

Labai svarbus yra panaudojimo vaizdas, susidedantis iš panaudojimo atvejų. Panaudojimo atvejį šioje reikalavimų specifikacijoje pilnai nusako panaudojimo atvejo elgsena (jo būsenų diagrama), panaudojimo atvejo scenarijai (veiklos diagrama), panaudojimo atvejo sąveika (sekų diagrama) ir papildomai įvesti atskiro panaudojimo atvejo nefunkciniai reikalavimai bei vartotojo interfeiso modelis.

Reikalavimų specifikacijos konteksto vaizdas aprašo kuriamos sistemos kontekstą, aplinką, t.y. tai, kas jau egzistuoja ir kas padeda geriau suvokti sistemai keliamus reikalavimus. Šis vaizdas aprašomas atskiruose paketuose.

Vidinis vaizdas nusako pradinį sistemos vaizdą, kurį pradžioje dažniausiai sudaro tik viena esybė – kuriama sistema.



4 pav. Reikalavimų specifikacijos struktūra

4 Reikalavimų apibrėžimo metodikos paaiškinimas pavyzdžiu

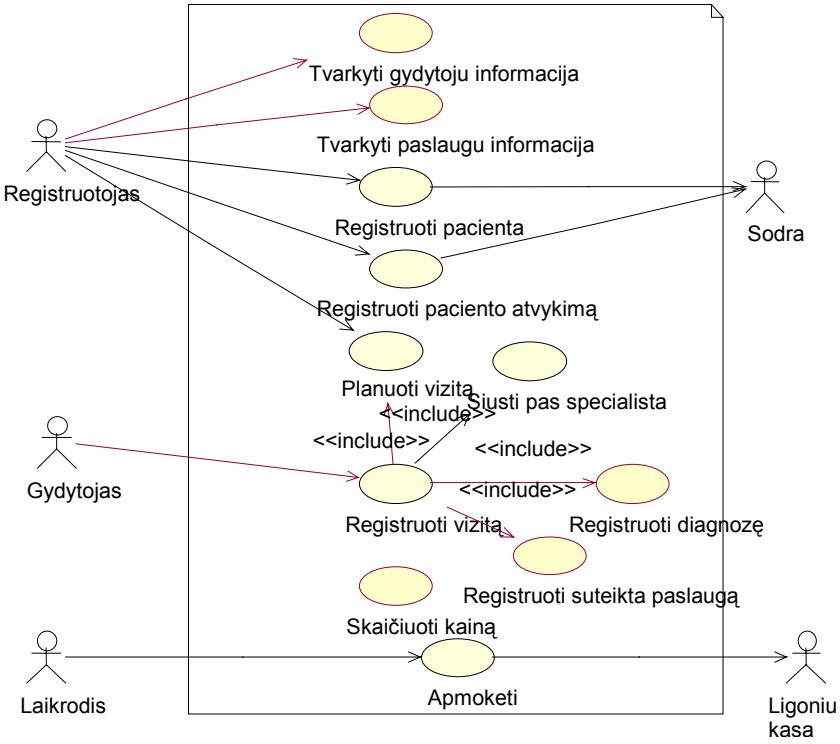
Dalykine sritimi, kuriai bus taikomas nagrinėjamas metodas, pasirinkta medicinos paslaugas pacientams teikianti poliklinika.

Poliklinikoje saugoma informacija apie gydytojus, jų specialybes, poliklinikos teikiamas paslaugas pacientams ir jų kainas bei pačius pacientus. Pirmiausia pacientai turi užsiregistruoti poliklinikoje, t.y. pateikti savo duomenis. Vėliau pacientas, norėdamas apsilankyti pas gydytoją, kreipiasi į registruotoją, kuris nurodo laiką ir datą, kada atvykti. Sekančiam vizitui pas gydytoją arba siuntimui pas naują specialistą gali užregistruoti ir pats gydytojas paciento vizito metu. Jei pacientas ateina neužsiregistravęs vizitui, registracija vykdoma vietoje ir gydytojas gali jį priimti, jei nėra kitų pacientų. Atvykus užsiregistravusiam pacientui, registruojamas paciento atvykimas, o vizito metu gydytojas užrašo diagnozę, suteiktą paslaugą, apskaičiuojama vizito kaina. Nustatytais laiko momentais (pvz., paskutinę mėnesio dieną) Sodros finansuojamiems pacientams suteiktas paslaugas apmoka ligonių kasos.

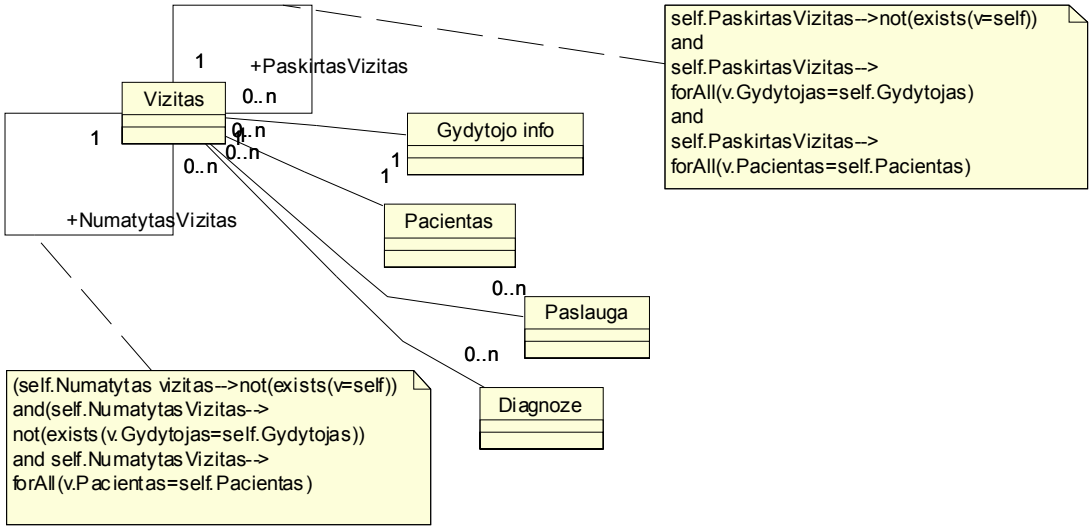
Šiame darbe naudojama 4 paveiksle aprašytų reikalavimų struktūros dalis, kuri reikalinga, aprašant statines elgsenos savybes, tai yra, nenagrinėjami galimi dinaminiai interfeisų ir operacijų tarpusavio apribojimai ir nenaudojamos veiklos diagramos. Esamos OCL kalbos galimybės šiuo metu neleidžia aprašyti tokio tipo apribojimų. Taigi laikomasi tradicinio interfeisų specifikavimo būdo, pagal kurį interfeisai atvaizduojami operacijų aibe, kiekvienai operacijai aprašoma signatūra, „prieš” ir „po” sąlygos.

Atvaizduojant panaudojimo atvejus interfeisais, gali kilti klausimų, kaip elgtis su panaudojimo atvejų ryšių <<*include*>> ir <<*extend*>> ryšių stereotipais. UML 2.0 versijoje, kaip ir ankstesnėse, nėra apibrėžta, kaip įtraukti priklausomus panaudojimo atvejus į viršesniųjų panaudojimo atvejų elgsenos modelius. [5] buvo priimta, kad specifikuojat panaudojimo atvejus, susietus <<*include*>> ir <<*extend*>> ryšiais, bus naudojami interfeisų apibendrinimo ryšiai, griežtai laikantis *Liskov* substitucijos principo.

Šis darbas taip pat neapima nefunkcinių reikalavimų ir vartotojo interfeiso modelio, kurie neįeina į informacinės sistemos schemą. Toliau pateikiami reikalavimų specifikacijos fragmentai, iliustruojantys siūlomą metodiką. Panaudojimo atvejų modelis nusako pagrindines kuriamos sistemos galimybes (5 pav.). Sistemą naudoja registruotojas ir gydytojas. Registruotojas registruoja pacientus ir tvarko įvairią sistemoje saugomą informaciją, gydytojas registruoja konkretaus vizito duomenis. Ligonių kasos apmoka suteiktas paslaugas Sodros finansuojamiems pacientams, kurių finansavimas tikrinamas, kreipiantis į Sodros informacinę sistemą. 6 paveiksle pateikiama dalykinės srities klasių diagrama rodo pagrindines klases, susijusias su tiriama sistema. Klasių apribojimai (invariantai) užrašyti OCL kalba.

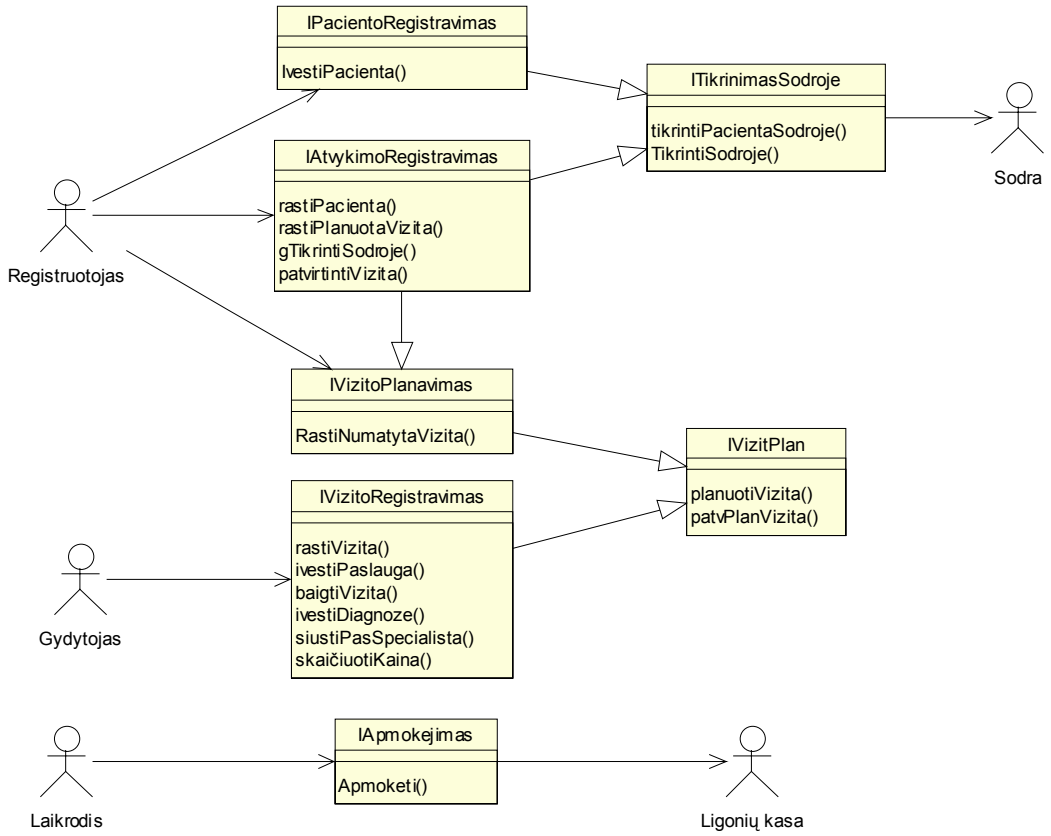


5pav. Panaudojimo atvejų diagrama



6 pav. Dalykinės srities klasių diagrama

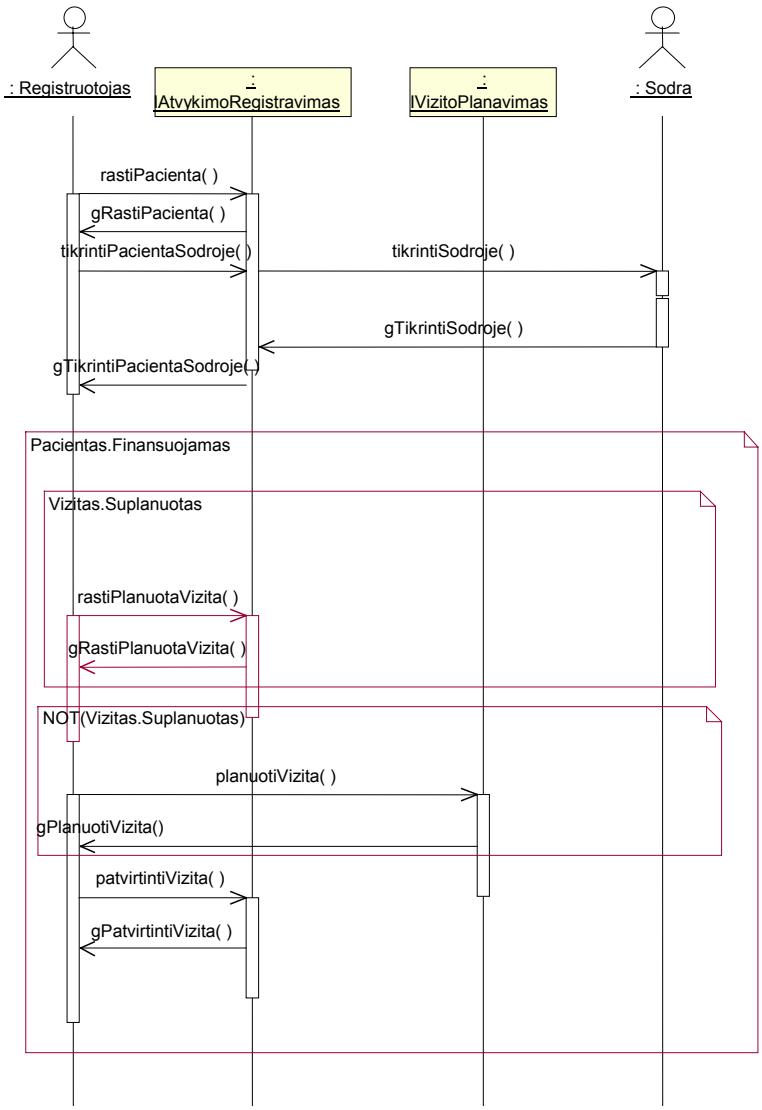
Interfeisų klasių diagramoje parodyti interfeisai, atitinkantys panaudojimo atvejus iš panaudojimo atvejų diagramos, jų operacijos tarpusavio ryšiai.



7 pav. Interfeisų klasių diagrama

Kiekvienas interfeisas atskirai aprašomas sekų ir būsenų diagramomis ir formalizuojamas operacijų sąrašu, kuriame kiekviena operacija aprašoma jos signatūra bei „prieš” ir „po”sąlygomis. 8 paveiksle pateikta

interfeiso *IAtvykimoregistravimas* sekų diagrama rodo, kokios sąveikos vyksta tarp registruotojo ir jo naudojamo sistemos interfeiso.

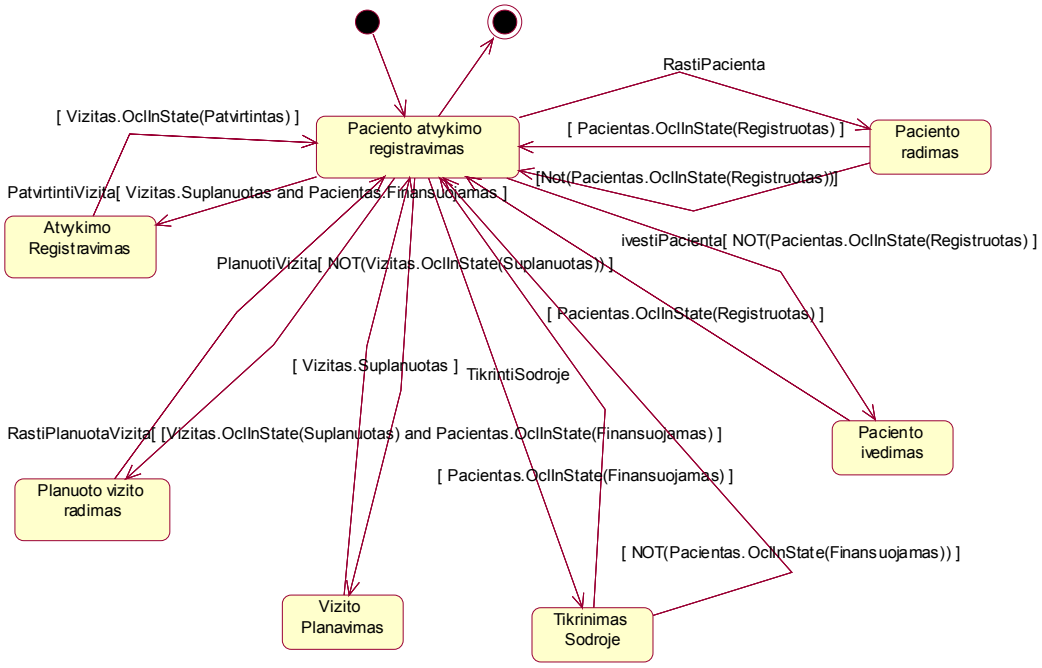


8 pav. Interfeiso "IAtvykimoRegistravimas" sekų diagrama

9 paveiksle parodyta interfeiso būsenų diagrama. Būsenų perėjimų apribojimai aprašyti klasės *IatvykimoRegistravimas* kontekste, kur *Pacientas, Gydytojas, Vizitas, Date* rodo operacijų argumentų reikšmes (*Date* šiame pavyzdyje reiškia ne tik datą, bet datą ir laiko intervalą, kuriame gali būti paskirtas vizitas). Operacijų argumentai yra dalykinės srities objektų tipai, jų reikšmės – konkretūs objektai, su kuriais atliekamos operacijos. Apribojimuose naudojamos dalykinės srities objektų būsenų reikšmės, pavyzdžiui, *v.OclInState(Patvirtintas)* reiškia, kad vizitas turi būti patvirtintas. Būsenų diagramoje perėjimų apribojimai atitinka sekų diagramose nurodytus apribojimus, skirtumas tik tame, kad perėjimas iš tam tikros būsenos į kitą tam tikrą būseną gali būti atvaizduotas keliose sekų diagramose, t.y., gali vykti pagal skirtingus scenarijus. Tokiais atvejais sekų diagramos apribojimai jungiami į būsenų perėjimų apribojimus logine jungtimi „or". Analogiškai tas pats perėjimo veiksmas (operacija) gali būti panaudotas keliuose būsenų perėjimuose. Todėl aprašant operacijas atitinkamos „prieš" ir „po" sąlygos gali jungti kelias atitinkamas perėjimų sąlygas. Visą panaudojimo atvejį galima aprašyti kaip operaciją, kurios argumentai – atitinkami dalykinės srities objektų tipai:

Context: IAtvykimo registravimas (p:Pacientas, g:Gydytojas, v:Vizitas,d:Date):Vizitas

Post: v.OclInState(Patvirtintas) or Vizitas → not(exists (p:Vizitas) = v)



9 pav. Interfeiso "IAtvykimoRegistravimas" būsenų diagrama

Toliau pateiktas interfeiso *IatvykimoRegistravimas* operacijų aprašymas OCL:

Operations

rastiPacienta(p:Pacientas):Pacientas

planuotiVizita(v:Vizitas, p:Pacientas, g:Gydytojas, d:Date):Vizitas

Context planuotiVizita(p:Pacientas,g:Gydytojas,v:Vizitas,d:Date):Vizitas

Pre

v.OclInState(Numatytas) or Vizitas → not(exists(p:Vizitas|p.Gydytojas=g and p.Date=d)

Post

v.oclInState(Suplanuotas)

rastiPlanuotaVizita(p:Pacientas,g:gydytojas,v:Vizitas, d:Date):Vizitas

gTikrintiSodroje(p:Pacientas):Boolean

patvirtintiVizita(p:Pacientas,g:Gydytojas,v:Vizitas, d:Date):Vizitas

Context patvirtintiVizita(p:Pacientas,g:Gydytojas,v:Vizitas, d:Date):Vizitas

Pre

v.oclInState(Suplanuotas) and v.Pacientas.oclInState(Finansuojamas)

Post

v.oclInState(Patvirtintas)

5 Išvados

Straipsnyje išanalizuoti keli objektiniame projektavime naudojami reikalavimų apibrėžimo metodai ir pasiūlyta reikalavimų modelio struktūra, kuria, autorių nuomone, galima išsamiai aprašyti reikalavimus kuriamai informacinei sistemai.

Reikalavimų aprašas turi apimti sistemos būsenos schemą (dalykinės srities klases, jų tarpusavio ryšius ir apribojimus, arba invariantus) bei elgsenos schemą, kuri vaizduojama sistemos interfeisais, jų ryšiais ir apribojimais.

Interfeisai išvedami iš panaudojimo atvejų. Interfeisų operacijos aprašomos signatūromis bei „prieš" ir „po" sąlygomis. Reikalavimų suderinimui sudaromos interfeisų sekų ir būsenų diagramos, apribojimai aprašomi naudojant dalykinės srities objektų būsenas. Tokio formalizuoto aprašymo privalumas – tikslus reikalavimų specifikavimas, leidžiantis vienareikšmiškai atvaizduoti reikalavimų aprašą į projekto modelį.

Išnagrinėtas reikalavimų apibrėžimo atvejis apsiriboja statiniais interfeiso operacijų apribojimais, kuriuos leidžia aprašyti OCL 2.0 versija. Norint aprašyti dinaminius operacijų apribojimus, t.y., interfeiso proceso valdymą, reikėtų išplėsti esamos OCL galimybes.

Atliekant šį darbą, buvo nagrinėjama UML 2.0 ir OCL 2.0 versijos ir panaudotos kai kurios naujos šių kalbų savybės, pavyzdžiui, kombinuoti sekų diagramų fragmentai (*Combined Fragments*), protokolų būsenų mašinos, nauji OCL operatoriai.

Straipsnyje pateikti pavyzdžiai iš poliklinikos informacinės sistemos reikalavimų apibrėžimo, kuris buvo sudarytas grafinio modelio pavidale ir OCL kalba. Tokį aprašą tikimasi panaudoti automatizuotam sistemos projekto modelio sudarymui.

6 Literatūra

[1] OMG Model Driven Architecture. [interaktyvus] 1997-2003 [žiūrėta 2003 11 14]. Prieiga per internetą: http://www.omg.org/mda/ .

[2] **Nemuraitė L., Kavaliauskaitė L., Ambrazevičius E**. Towards Ensuring Consistency to UML Models // Informacijos mokslai. ISSN 1392-0561. Vilniaus universitetas, 2002, No. 24, p. 85-97.

[3] **Kosters G., Six H.W., Winter M.** Coupling Use Cases and Class Models as a Means for Validation and verification of requirements Specifications. – Requirements Engineering, *Springer-Verlag*, 2001, Volume 6, Issue 1, pp. 3-17.

[4] **Astesiano E., Reggio G.** Knowledge Structuring and Representation in Requirement Specification- Italy: DISI-Universita di Genova.

[5] **Čeponienė L., Nemuraitė L., Paradauskas B.** Design of schemas of state and behaviour for emerging information systems. Proceedings of the 1st Workshop on Emerging Database Research in Eastern Europe, co-located with VLDB 2003. Humboldt-Universität Berlin, Germany, September 8 2003, p. 27-31

[6] Unified Modeling Language: Superstructure. *2nd revised submission to OMG RFP*. 2003 [Žiūrėta 2003 11 15]. Prieiga per internetą: www.omg.org

[7] Response to the UML 2.0 OCL RfP. *OMG Document*. 2003 [Žiūrėta 2003 11 15]. Prieiga per internetą: www.omg.org

[8] Nemuraitė L., Kavaliauskaitė L. Informacinių sistemų projektavimo metodų tobulinimas ir automatizavimas, taikant UML // Informacinės Technologijos'2002. - Kaunas: Technologija, 2002, p. 408-414.

METHOD FOR SPECIFICATION OF REQUIREMENTS FOR DEVELOPMENT OF INFORMATION SYSTEMS USING UML AND OCL

Lina Nemuraitė, Eglė Zaksaitė

In current paper some progressive object-oriented requirement definition methods were analyzed and comprehensive content of requirements model was presented. Requirements definition must define state and behavior of target information system independently of future design. Requirements model was proposed where use cases are presented as interfaces together with associated classes of problem domain. Model was described using UML class, sequence diagrams, state charts and OCL constraints. Model is illustrated with examples from case study.