KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Agnė Paulauskaitė

Z formaliųjų metodų panaudojimas informacinių sistemų projektavime

Informatikos magistro baigiamasis darbas

Vadovas
prof.habil.dr. H. Pranevičius

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. H. Pranevičius
2005-05-23

Z formaliųjų metodų panaudojimas informacinių sistemų projektavime

Informatikos magistro baigiamasis darbas

Kalbos konsultantė
Lietuviu k. katedros lekt.
dr. J. Mikelionienė
2005-05-16

Recenzentas
doc. E. Karčiauskas
2005-05-23

Vadovas
prof. H. Pranevičius
2005-05-23

Atliko
IFM 9/1 gr. stud.
A. Paulauskaitė
2005-05-16

KAUNAS, 2005

Summary

The use of Z formal methods for information systems design

Still today informal methods are the most common for informational systems design. They don‘t allows unambiguously understand formulating tasks, moreover availably specifications not always are complete. Because of this informational system does not correspond to the users needs. Using informal methods specification transformation to software code isn‘t always possible.

In real time informational systems problematic domain is varying in time. Thus are changing requirements for informational systems. Using informal methods, to solve this problem, usually we need to rewrite software. Using formal methods we don‘t have to rewrite software, it is enough organization business instructions specified in Z transform to software code.

In this paper we present research results about Z specification method use for formal requirements specification for informational systems design. Using Z/EVES - an interactive system for composing, checking, and analyzing Z specifications, was accomplished Z specification validation, theretofore reviewing the list of available Z specification validation tools. Z specification language was compared with object-oriented language Object–Z to find out differences of these two formal specification languages. Were discussed questions about Z specification transformation to Object-Z, which facilities an object-oriented specification extension to object-oriented programming languages. In this paper transformation methodology from object-Z to object-oriented programming languages C++ and SQL DDL statements is suggested. This methodology makes a reason to use formal methods for informational systems design.

Turinys

Lentelių sąrašas

Paveikslėlių sąrašas

1 ĮVADAS

Iki šiol nepakankamai plačiai yra naudojami formalieji metodai, kurių panaudojimo tikslingumas pagrįstas visoje eilėje mokslinių publikacijų [1, 2]. Programinės įrangos išvystymas yra sudėtingas procesas, prasidedantis nuo sistemos reikalavimų apibrėžimo, projektavimo ir baigiant sistemos realizacija. Proceso išvystymo problemos proporcingai auga su sistemos dydžiu. Pagrindinė problema iškyla siekiant išsiaiškinti probleminę sritį ankstyvuosiuose projektavimo etapuose. Tokiu atveju yra tikslinga naudoti formaliuosius metodus, kuriuos naudojant yra išvengiamos šios iškylančios problemos. Formalieji metodai padeda išsiaiškinti ir pašalinti dviprasmiškumus ir neapibrėžtumus dar analizės ir projektavimo stadijose [3]. Padeda sukurti įrankius, tokius kaip sintaksės, semantikos tikrintojai ar validavimo priemonės.

Šiame darbe yra pateikiami tyrimo rezultatai apie Z specifikavimo metodo panaudojimą formaliam reikalavimų specifikavimui, projektuojant informacines sistemas (IS). Taip pat išnagrinėta, kaip Z formalioje specifikavimo kalboje aprašytus informacinės sistemos reikalavimus galima transformuoti į Object-Z ir iš Object-Z į programinės įrangos kodą.

Visų pirma ir svarbiausia reikia pasirinkti metodą, kuriuo aprašytume reikalavimus kuriamai informacinei sistemai. Pasirinkus netinkamą metodą gali būti netiksliai nusakyti reikalavimai**,** dėl ko atsiranda daug problemų. Nevienareikšmiai reikalavimai gali būti skirtingai interpretuojami kūrėjo ir vartotojo. Kadangi klientai dažnai tiksliai nežino ko jie tikisi iš kuriamos sistemos, todėl sukurta sistema gali elgtis ne visiškai pagal jų poreikius. Reikalavimai turi būti pilni ir suderinti. Juose turi būti visi aprašymai apie visas reikalaujamas galimybes. Neturėtų būti konfliktų ir prieštaravimų sistemos galimybių aprašymuose. Tokių problemų galima išvengti naudojant formalius metodus, kurie reikalavimus aprašo matematiškai ir užtikrina sistemos savybių korektiškumą, pilnumą bei vienareikšmiškumą.

Pasirinkus formalų metodą reikia apžvelgti kokios yra kalbos, tinkamos informacinės sistemos specifikavimui. Parinkti konkrečiai informacinei sistemai specifikuoti geriausiai tinkančią specifikavimo kalbą nėra paprasta. Viena vertus, kalba turi turėti raiškos gebą, pakankamą visiems kuriamosios sistemos aspektams specifikuoti. Kita vertus, ji turi būti kuo paprastesne, patikimesne ir tenkinti daugeli kitų reikalavimų, išplaukiančiu iš konkretaus projekto pobūdžio.

Šiame darbe IS reikalavimų aprašymui pasirinkta Z specifikavimo kalba, kuri vienareikšmiškai aprašo analizuojamą sistemą, lengvai aprašo įvairius abstrakcijos lygius. Z lengvai susidoroja su didelėmis specifikacijomis, kūrimui panaudojant schemų notaciją. Prie kiekvienos schemos gali būti neformalus komentaras, kuris ją paaiškina ir padeda greičiau susiorientuoti visoje sistemos specifikacijoje.

Specifikacijos validavimas yra būtinas norint patikrinti ar specifikacija yra parašyta korektiškai, ar atitinka sistemai keliamus reikalavimus. Validuojant yra gerinama produkto kokybė, klaidos gali būti aptiktos ankstesnėje projektavimo pakopoje, tokiu būdu sutaupomos lėšos jų taisymui, taip pat tai padeda geriau suprasti ir realizuoti kuriamą produktą. Yra daugybė validavimo įrankių, kurie skirti specifikacijos analizavimui, tipų tikrinimui ar net teoremų įrodinėjimui. Z specifikacijos patikrinimui buvo pasirinktas Z/EVES validavimo įrankis, kuris leidžia patikrinti specifikacijos sintaksės bei tipų korektiškumą, nustatyti invariantų teisingumą.

Transformacija yra operacija, kuri vienos programavimo kalbos programą ar specifikaciją konvertuoja į kitos programavimo kalbos programą ar specifikaciją, išlaikant ar praplečiant jos semantiką. Validuotą formalią specifikaciją galima transformuoti į reikalavimus atitinkantį programinės įrangos kodą ir taip sumažinti žmogaus daromų klaidų kiekį bei sąnaudas rašant programas. Šiame darbe yra siūloma formalios specifikacijos transformavimo į programinės įrangos kodą metodika.

1.1 Tyrimo Objektas ir problema

Tyrimo objektas – Z formalios specifikavimo kalbos panaudojimas informacinės sistemos reikalavimų specifikavimui bei šių reikalavimų panaudojimas realizuojant informacines sistemas.
Pagrindinė sprendžiama problema – Z specifikacijų sudarymas, jų validavimas bei sudarytų specifikacijų praplėtimas su tikslu gauti informacinės sistemos programas.

1.2 Tikslas ir uždaviniai

Panaudojant Z formaliąją specifikaciją sukurti šių specifikacijų transformavimo į Object-Z ir Object-Z transformavimo į programinės įrangos kodą metodikas, kurios sudarytų pagrindą panaudoti formaliuosius metodus projektuojant informacines sistemas.

Sprendžiami uždaviniai:

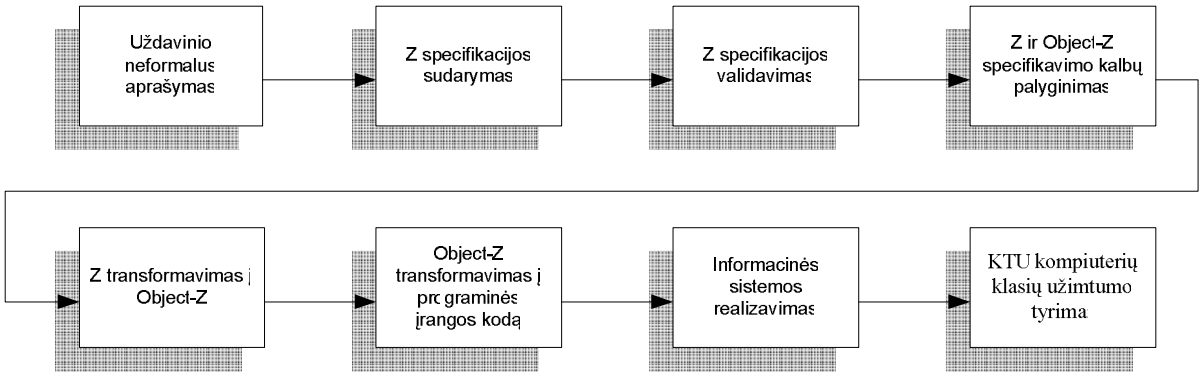
1. Z ir Object-Z specifikavimo metodų palyginimas;
2. formaliųjų specifikacijų validavimo įrankių parinkimas;
3. Object-Z formalios specifikacijos transformavimas į programinės įrangos kodą;
4. sukurti Z formalios specifikacijos transformavimo metodiką, skirtą projektuojant ir programiškai realizuojant kompiuterių klasių užimtumo IS;

5.1. kompiuterių klasių užimtumo uždavinio neformalus aprašymas;
5.2. kompiuterių klasių užimtumo formalios specifikacijos sudarymas Z specifikavimo kalba;
5.3. Z formalios specifikacijos validavimas panaudojant Z/EVES validavimo įrankį;
5.4. Z specifikacijų transformavimas į Object-Z specifikacijas;

5.5 Object-Z specifikacijos transformavimas į C++ programavimo kalbą ir SQL DDL aprašą;

5. sukurtos informacinės sistemos realizacija;
6. Kauno technologijos universiteto kompiuterių klasių užimtumo tyrimai.

Žemiau apteikta darbe panaudotos tiriamos metodikos schema (žr. 1.1 pav.).



1.1 pav. Darbe panaudotos tiriamos metodikos schema

2 Z FORMALIZAVIMO METODO GALIMYBIŲ PANAUDOJIMAS PROJEKTUOJANT INFORMACINES SISTEMAS

2.1 Informacinių sistemų specifikavimo metodai ir kalbos

Specifikacija yra sistemos funkcionalumo modelis. Specifikacija daugiau aprašo ką sistema turėtų daryti, nei kaip tai daryti. Specifikacija gali būti neformali (aprašoma žodžiai) ir formali (išreikšta matematine kalba). Yra skirtumas tarp specifikavimo metodo ir specifikavimo kalbos. Metodas nusako ką turi pasakyti specifikacija. Specifikavimo kalba detaliai apibrėžia kaip specifikacijoje gali būti išreikštos sąvokos. Kai kurios kalbos padeda geriau nei metodai. Kai kurie metodai lengvai naudojami su tam tikromis kalbomis.

Specifikavimo kalbos gali būti klasifikuojamos pagal semantinę sritį:

- abstrakčių duomenų tipų (ADT) specifikavimo kalba;
- proceso specifikavimo kalba;
- programavimo kalba.

Abstrakčių duomenų tipų specifikavimo kalbos gali būti naudojamos algebrai specifikuoti. ADT apibrėžia duomenų tipų formalias savybes. Z yra ADT specifikavimo kalbos pavyzdys. Proceso specifikavimo kalbos aprašo būsenos, įvykių sekas, srautus, dalinius nurodymus. Programavimo kalbos pateikia aiškius kalbos su modeliais pavyzdžius.

Specifikavimo metodai skirstomi į formaliuosius ir struktūrinius neformaliuosius metodus. Formaliuosiuose metoduose yra aiškiai apibrėžtas žodynėlis, sintaksė, semantika naudojama konstruojant specifikacijos dokumentą [4]. Be to formali specifikacija yra pagrįsta matematiniu pagrindu. Kai kurie metodai pagrįsti sistemos modeliavimu, įtraukiant duomenų dekompoziciją [5].

Skirtingi specifikavimo metodai kai kuriems tikslams yra naudingesni už kitus. Formalieji metodai yra daugiau skirti aiškesnėms specifikacijoms. Nesusipratimai ir klaidos tokiu būdu gali būti aptiktos žymiai anksčiau. Kuo anksčiau yra aptiktos klaidos, tuo pigiau jos gali būti ištaisytos. Formalieji metodai gali pagerinti tiek produktyvumą, tiek kokybę [6]. Kainos sutaupymas gali būti pasiektas tik tada, jei formalieji metodai naudojami atitinkamai ir protingai. Kai yra geriausia juos naudoti specifinėje aplinkoje gali būti nustatyta tik eksperimentuojant.

2.1.1 Neformalieji metodai

Neformalieji metodai [7] yra mažiau strukturizuoti tyrinėjimo metodai tokie, kaip pavienių atvejų tyrinėjimas ar sistemos vartotojų stebėjimas.

Neformaliųjų metodų savybės [8] yra išvardintos žemiau:

Elgsenos integravimas. Daugumai neformalių metodų nėra formalaus mechanizmo. kuris integruotu atskirai aprašytas elgsenas.

Kūrimas pažingsniui. Kūrimas pažingsniui buvo pripažintas, kaip naudingiausias būdas, specifikuojant dideles ir sudėtingas sistemas. Sistemos elgsenos aprašomos po truputį, po žingsniuką. Tik tada šios specifikacijos yra integruojamos į visą sistemą. Neformaliuosiuose metoduose elgsenos komponentų integravimo metu yra reikalingas specifikacijos peržiūrėjimas ir pataisymas. Kadangi nėra jokio specifikacijos taisymo mechanizmo, todėl į šį procesą gali būti lengvai įtraukiamos klaidos.

Palaikymas. Pradiniame specifikavimo lygyje gali pasikeisti reikalavimai, pvz., labai dažnai sistemos analitikas gali padėti galutiniams vartotojams geriau suprasti jų norus. Tada vartotojai gali pakeisti savo reikalavimus. Taigi reikėtų pakeisti specifikaciją. Tai yra dar viena silpnoji neformalių metodų pusė. Nėra jokio mechanizmo, kuris nustatytų pasikeitusius komponentus ir rezultatus po tokių pasikeitimų

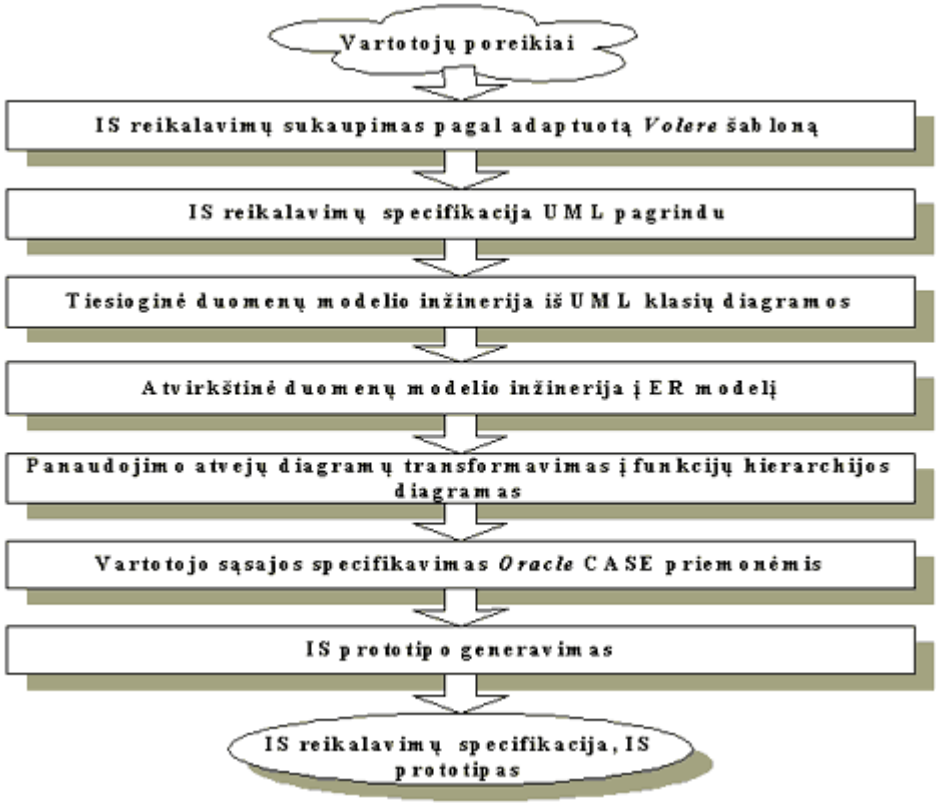
Analizė. Dirbant su neformaliaisiais metodais reikalavimus vartotojai gali analizuoti tik rankiniu būdu. Nėra jokių mechaninių taisyklių, kurios palaikytų analizę. Tai yra gerai žinomas neformalių metodų trūkumas. Iš kitos pusės tai formalių metodų stiprioji pusė.

Specifikacija per kelias diagramas. Pavyzdžiui, knygų pasiskolinimas ir knygų rezervavimas yra dviejų ortogonalių dimensijų elgsenos. Dauguma neformalių metodų suspaudžia šias dvi ortogonalias elgsenas į dvi dimensijas. Neformalieji specifikavimo metodai, tokie kaip unifikuota modeliavimo kalba (UML) šias ortogonalias elgsenas vaizduoja skirtingomis diagramomis. Taigi vartotojai turi integruoti ir analizuoti šias specifikacijas skirtingose diagramose rankiniu būdu. Atsiranda perėjimo problemos tokios kaip neišbaigtumas, kuris labai dažnai atsiranda perėjimo lygyje. Tačiau iš kitos pusės neformalieji metodai yra lankstesni už formaliuosius metodus, todėl yra daug lengviau susidoroti su perėjimo problemomis.

Suvaržymai. Vienas iš sunkumų specifikuojant sistemas yra suvaržymų apdorojimas. Jacksono išsivystymo sistema (JSD) siūlo susidoroti su suvaržymais pasitelkiant į pagalbą kitą JSD. UML suvaržymų pateikimui naudoja predikatus, kurie yra tam tikros diagramų anotacijos. Specifikuojant sudėtingesnes sistemas gali būti reikalingi aukštesnės pakopos suvaržymai. Ši problema nėra tinkamai išspręsta daugumoje neformaliųjų metodų.

Reikalavimai procesui gali būti specifikuojami panaudojant specialią neformalią CASE (angl. *Computer Aided Software Enginiering*) sistemą, programavimo kalbą ar projektavimo metodą. CASE įrankis [9] yra programinis paketas, palaikantis vieną ar kelis programinės įrangos kūrimo proceso veiksmus. CASE įrankių atstovų pavyzdžiais gali būti: Oracle CASE – sistemų projektavimo metodologija, apimanti strategijos, analizės, projektavimo bei generavimo etapus [10], UML CASE

įrankiai [11], Virtuali UML paradigma (VP-UML), OOAD ir kiti . Jie dažnai naudojami analizuojant ir projektuojant informacines sistemas. Kaip pavyzdys, informacinės sistemos analizės ir projektavimo etapai išplėstu *Oracle* metodu pateikiami 2.1 paveiksle.



2.1 pav. Informacijos sistemos analizės ir projektavimo etapai, išplėtus Oracle metodą

Publikacijoje [12] pateikiama koncepcija, kaip išplėsti informacijos sistemos reikalavimų specifikavimo galimybes, kai informacijos sistemai kurti naudojamas *Oracle* CASE metodas. Vartotojo reikalavimams aprašyti pirmine forma naudojamas adaptuotas *Volere* šablonas, o reikalavimų specifikacija pateikiama UML diagramų pagrindu. Pereinant nuo UML diagramų sudarymo etapo prie darbo su *Oracle* CASE, generuojamas fizinis duomenų modelis iš UML klasių diagramos. Fizinis duomenų modelis *Oracle* aplinkoje automatizuotai transformuojamas į ER modelį, o UML panaudojimo atvejų diagramos pagal pateiktas transformavimo taisykles keičiamos į *Oracle* CASE funkcijų hierarchijos diagramą. Sistemos galutinės specifikavimo fazės metu *Oracle* CASE priemonėmis specifikuojami vartotojo sąsajos reikalavimai, kadangi *Oracle* CASE turi tam tikslui išplėtotas priemones.

CASE įrankiai iš tiesų plačiai naudojami kompiuterizuotoms informacinėms sistemoms tobulinti arba pakartotinėje inžinerijoje (re-inžinerijoje). Nors tokie įrankiai buvo populiarūs ir perspektyvūs jų atsiradimo pradžioje, tačiau greitai buvo pastebėti jų trūkumai ir apribojimai [13]. CASE įrankiai užima labai svarbią vietą programinės įrangos (PI) kūrimo procese, tačiau dažnai jie nesuteikia priemonių naujos PI kokybiniams rodikliams užtikrinti. Egzistuoja daug neišnaudotų

galimybių, kuriomis būtų galima patobulinti CASE įrankiais vykdomus informacinių sistemų projektavimo metodus bei padidinti jų automatizavimo laipsnį.

2.1.2 Formalieji metodai

Formalieji metodai tampa vis labiau pripažinti tiek mokslo, tiek verslo aplinkoje, kaip vienas iš būdų, kuriuo galima padėti pagerinti tiek programinės įrangos, tiek techninės įrangos sistemų kokybę.

Turėtų būti nepamirštama, jog tai nėra universali priemonė, bet daugiau kaip dar vienas ginklas nuo projektavimo klaidų (*Prof. Tony Hoare*).

Žinoma, kad nėra visiškai įrodančios metodikos ar formulės, kuri užtikrintu gerą, efektyvų projektavimą. Tam reikia, kad projektuotojas turėtų patirties, įžvalgumo, gabumus, nuovoką, išradingumą. Formalieji metodai gali tik paaiškinti projektavimo alternatyvas, padėti tyrinėjant projektavimo padarinius, formalizuoti, perduoti projektuotojo sprendimus bei padėti įsitikinti, kad viskas atlikta teisingai (*C.A.R. Hoare*, 1988).

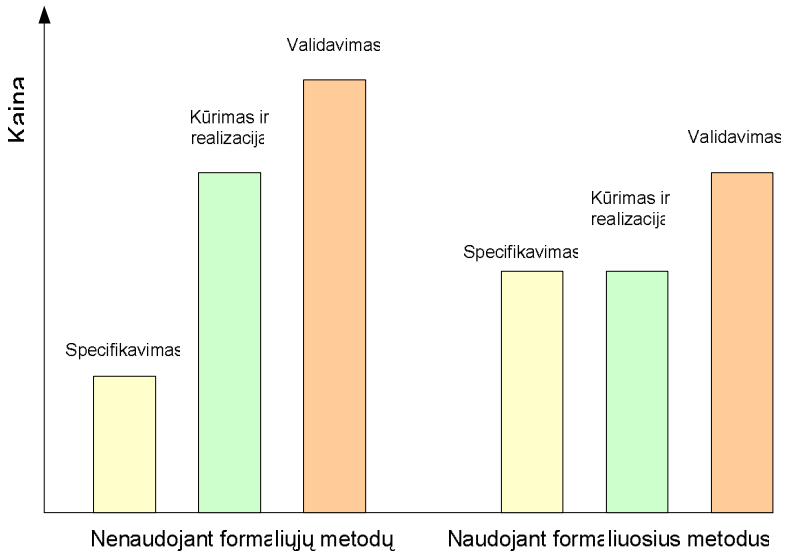
Formalieji metodai apima:

- formaliąją specifikaciją;
- specifikacijos analizę ir įrodymus;
- transformacinį išsivystymą;
- programos validavimą, verifikavimą.

Naudojant formalią notaciją didėja supratimas apie sistemos veikimą. Tai padeda projektuotojams ir projektavimas tampa kiek įmanoma aiškesnis ir paprastesnis. Yra įmanoma formaliai mąstyti apie sistemą, formuluojant ir pateikiant teoremas. Jos patikrina ar sistema elgiasi taip kaip tikėjosi projektuotojai. Formaliųjų metodų panaudojimas padeda išnagrinėti projektavimo alternatyvas. Tokie metodai padeda projektavimo komandai mąstyti apie sistemos darbą dar prieš jos realizavimą. Trūkstamos, nebaigtos specifikacijos dalys tampa vis akivaizdesnės. Jos gali būti greitai aptiktos ir tada svarstomos alternatyvios galimybės. Klaidų vietos gali būti tikrinamos skaičiuojant išankstines veikimo sąlygas. Naudojant formaliuosius metodus galima lengviau išsiaiškinti tokias detales dar iki realizavimo lygio.

Sistemos dokumentacija taip pat gali būti tobulinama. Naudojant formalųjį projektavimą kaip sistemos vadovo pagrindą, yra tikimybė, jog bus praleista mažiau reikalingos informacijos. Galutinis dokumentas turėtų apimti formalaus teksto vertimą į realius žodžius. Pagaliau svarbiausia pramonėje tai, kad galutinė kaina turi būti kiek įmanoma mažesnė (žr. 2.1 pav.). Kaina gali būti dvigubai mažesnė, kai klaidos yra aptiktos ir pataisytos projektavimo lygyje. Pagrindinis barjeras naudojant formaliuosius metodus yra tas, jog jie naudoja nepažįstamus simbolius, ir dėl to priverčia

projektuotojus lankyti mokymosi kursus. Kaip bebūtų, tai nėra blogiau nei įsisavinti naują programavimo kalbą.



2.2 pav. Vystymosi kaina naudojant formalius ir neformalius metodus

Formaliąją specifikaciją gali būti labai sunku skaityti, ypač jei ji parašyta didelėmis matematinėmis formulėmis. Z kalbos projektuotojai ypatingą dėmesį skyrė šiai problemai spręsti. Z specifikacijos neformalus tekstas yra papildytas formaliais aprašymais.

Apibendrinant galima išvardinti formaliųjų metodų stipriąsias savybes:

- aprašymai be dviprasmybių;
- automatizuoti įrankiai, kurie padeda:
  - validuoti ir verifikuoti;
  - kodui generuoti;
  - testiniams rinkiniam generuoti;
  - sistemingai dokumentacijai.

Labiausiai paplitę yra šie formalieji metodai:

- baigtiniai automatai,
- B-metodas,
- Petri tinklai,
- Z.

Kiekvieno iš jų panaudojimas priklauso nuo formalizuojamos sistemos tipo. Baigtiniai automatai ir Petri tinklai daugiau naudojami tokiose sistemose, kuriose sprendžiami valdymo uždaviniai. Baigtiniai automatai [14, 15] gali būti naudojami realaus laiko sistemų modeliavimui. Tačiau baigtinių automatų modeliai stokoja struktūrinių elementų, netgi paprastos sistemos gali turėti sudėtingą modelį. Petri tinklai naudojami modeliuojant sistemas, kuriose vienu metu gali įvykti keletas įvykių ir yra apribojimai procesu dažnumui, sekai [16, 17]. B-metodas [18] ir Z daugiausiai

naudojami sudėtingoms, kritinėms sistemoms aprašyti. Šiuo atveju informacinės sistemos reikalavimų aprašymui Z yra labiausiai tinkama formalizavimo kalba. Z specifikavimo pasirinkta dėl keleto priežasčių: Z yra gana plačiai žinoma formali specifikavimo kalba ir jau yra panaudota daugybėje įvairių sistemų [19]. Yra nemažai prieinamų vadovėlių apie Z formalų specifikavimo metodą [20]. Vis didesnį susidomėjimą kelia Z specifikavimo kalbos mokymasis [21]. Lyginant su kitais formaliaisiais metodais, Z turi daugybę įrankių specifikacijos validavimui ir verifikavimui [22].

2.1.2.1 Z specifikavimo kalba

Z yra tipizuota specifikavimo notacija [23, 24], paremta predikatų logika ir Zermelo-Fraenkelo (ZF) aibių teorija. Notacija sukūrė Jean-Raymond Abrial, kai lankėsi Oxfordo universiteto skaičiavimo laboratorijoje. Vėliau buvo vystoma Hayeso, Morgano, Spivey, Sufrino ir kitų. Z yra populiari verslo sistemose, universitetinėje aplinkoje, kai kuriose pramonės šakose, ypač ten, kur kuria kritines sistemas [25], kur yra labai svarbi programinės įrangos kokybė ir kuo mažesnis klaidų skaičius.

Z specifikacija gali būti parašyta daugybe stiliumi. Tačiau buvo išsiaiškinta, jog daugeliu atveju patogiausia yra naudoti būsenomis ir modeliais pagrįstą modelį. Sistema gali būti modeliuojama kaip abstrakti būsena, arba šios būsenos operacijų seka. Pirmiausia turi būti aprašytos pagrindinės aibės. Šiame lygyje, nėra būtina detalizuoti aprašymo. Yra naudinga pateikti pagrindinius operatorius, kad specifikaciją padaryti labiau skaitomą. Abstrakti būsena yra aprašoma aibėmis, ryšiais, funkcijomis ir sekomis. Tai neturėtų būti įtakojama realizavimo sprendimais, bet sukurtos taip, jog specifikacija būtų kuo lengviau suprantama ir skaitoma.

Turėtų būti apibrėžta ir vidinė būsena. Tai apibrėžiama abstrakčia būsena ir kai kuriais predikatais apibrėžtais sistemos pradinėse sąlygose. Sistemos operacijos iššaukia būsenos pasikeitimą. Čia atsiranda *prieš* būsena ir *po* būsena. Gali atsirasti invariantai, kurie susiję su sistemos visų operacijų *prieš* ir *po* būsenomis. Jie gali būti įtraukti į schemą kaip predikatai, kurie apibrėžia sistemos būsenos pasikeitimą. Kartais būsena *po* yra tokia pati kaip ir būsena *prieš*. Taip pat operacijų grupė gali neturėti jokio poveikio tam tikrai būsenos daliai. Kiekvienos operacijos predikatai aprašo tiksliai, ką reikia padaryti. *Įvestis* ir *išvestis* taip pat gali būti įtraukti. Kiti laikini būsenos komponentai gali būti pridėti, jei tai padeda specifikaciją padaryti aiškesnę.

Formalus Z pagrindas yra pirmos eilės predikatų logika praplėsta aibių teorija. Tačiau praktikoje daugelis Z specifikacijų naudoja tik kelis loginius simbolius.

Yra dvi loginės konstantos predikatų logikoje: *tiesa* (angl. *true*)ir *netiesa* (angl. *false*). Z kalboje predikatai turi viena iš šių reikšmių. Čia nėra trečios neapibrėžtos reikšmės, kuri padėtų supaprastinti šios kalbos interpretacijos sudėtingumą. Tai skiriasi nuo kai kurių kitų logikų, kurios

kartais prideda neapibrėžtą trečią reikšmę, kad susitvarkytų su tokiais atvejais. Ši ekstra reikšmė gali pridėti sudėtingumo, o Z tikslas viską pateikti kuo įmanoma lengviau ir suprantamiau.

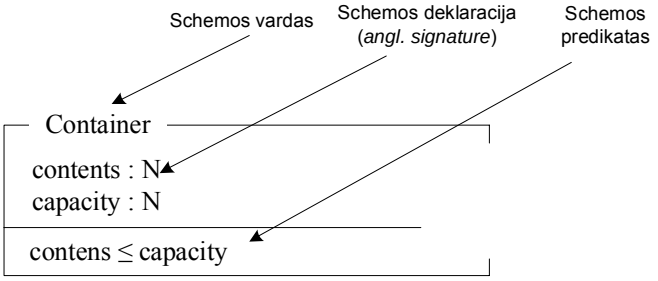
Z specifikacijose dažniausiai naudojami standartiniai dvinariai junginiai yra tokie:

- loginė konjunkcija $p \wedge q$ grąžina *tiesą*, jei p ir q yra tiesa, priešingu atveju gražina *netiesą*;
- loginė disjunkcija $p \vee q$ grąžina *tiesą,* jei p ar q yra tiesa, priešingu atveju *netiesa*;
- loginė implikacija $p \Rightarrow q$, yra tas pats kas $\neg p \vee q$ **.** Jei p yra tiesa tai ir q turetų būti, *tiesa***;** priešingu atveju q gali pasiimti bet kurią reikšmę;
- loginis ekvivalentiškumas $p \Leftrightarrow q$, yra tas pats kas $p \Rightarrow q \wedge q \Rightarrow p$**.** Ir p ir q turėtų būti vienodos reikšmės, kad gautume *tiesą* , kitu atveju yra gražinama *netiesa***.**

Pilna predikatų logika papildo teiginių logiką kvantavimu:

- universalus kvantavimas $\forall$ **X** • **q** yra tik tada *tiesa*, kai q yra tiesa visoms įmanomoms X reikšmėms;
- egzistencinis kvantavimas $\exists$ **X**• **q** yra *tiesa*, jei yra įmanoma nors viena reikšmių grupė X, kad q paverstu tiesa. Gali būti ir daugiau nei viena reikšmė;
- unikalus egzistencinis kvantavimas $\exists_1$ X• q tai vienas bendresnių egzistencinio kvantavimo atvejų, kai X leidžiama paimti tik vieną reikšmę, o ne pasirenkamą reikšmių skaičių.

Specifikacijos yra sudarytos iš komponentų, kurie vadinami schemomis (žr. 2.3 pav.). Schema naudojama, tam kad pateikti būsenos kintamuosius ir apibrėžti ryšius bei būsenos operacijas. Visos Z schemos turi invariantus, kurie aprašo sąlygas ir kurios visada yra *tiesa*.



2.3 pav. Z schemos pavyzdys

Operacijos gali būti specifikuojamos tam, kad aprašytų jų poveikį sistemos būsenai. Galima rašyti operacijas ir kombinuoti specifikacijos fragmentus tam, kad pilnai aprašyti specifikaciją.

Pagrindiniai operacijų simboliai yra šie:

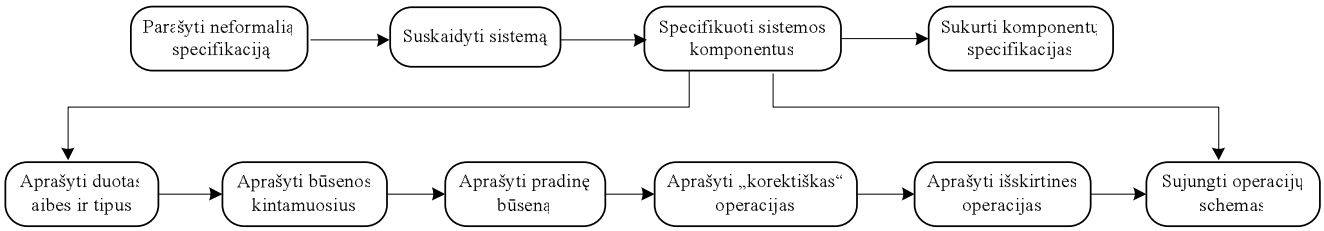
- „?“ operacijos įvestis (angl. *input*),
- „!“ operacijos išvestis (angl. *output*),
- „a“ būsenos komponentė prieš operaciją, „ a′ “ būsenos komponentė po operacijos,
- „ΔS“ būsenos pasikeitimas.

Operacijos nebūtinai turi būti determinuotos. Kartais gali būti daugiau nei vienas operacijos rezultatas. Pavyzdžiui, sistema gali suteikti failui vardą iš vardų aibės. Projektuotojas neturėtų rūpintis, kuris vardas buvo išrinktas. Paskutiniuosiuose specifikacijos lygiuose, sistema modeliuojama vidinėmis būsenomis, laikantis pasirinktos operacijų sekos. Jei nors viena operacija turi predikatus, vykdytojas turi užtikrinti, kad operacija gali būti vykdoma tik tada, kai yra patenkintos šios sąlygos. Pilna operacijų seka yra įmanoma tik tokiu atveju, jei visų operacijų predikatai yra *tiesa*.

Z gali būti naudojama įskaitomai ir aiškiai specifikacijai sukurti. Z buvo sukurta daugiau žmonėms, nei kompiuteriams skaityti. Be to, ji gali suformuoti dokumentacijos pagrindą. Z lengvai susidoroja su didelėmis specifikacijomis, kūrimui naudojant schemų notaciją. Naudojant Z yra įmanoma sukurti hierarchinę specifikaciją. Dalis sistemos gali būti specifikuojama atskirai, ir tada gali būti patalpinta į bendrą kontekstą.

Kai projektas yra parengtas, galima išdėstyti ir įrodyti sistemos teoremas. Tai padeda verifikuoti projektą ir patikrinti klaidas. Šis procesas labai reikalingas tam, kad sumažinti klaidas ir gauti supratimą apie sistemos veikimą dar prieš jos realizavimą.

Žemiau pateiktas Z specifikacijos sudarymo procesas (žr. 2.3 pav.)



2.4 pav. Z specifikacijos sudarymo procesas

2.1.2.2 Specifikacijos validavimas

Specifikacijos validavimas [26] yra panašus į tradicinę validavimo notaciją, nurodant specifikacijos tikrinimo procesą į neformalius vartotojo reikalavimus sistemai.

Yra nemažai Z palaikymo įrankių, kurie atlieka sintaksės analizę [27, 28], tipų ir klaidų tikrinimą [29]. Kai kurie iš jų iš dalies palaiko Z specifikacijos teoremų pateikimą. Z specifikacijose dažniausiai yra tikrinamos šios savybės:

- tikrina ar būsenos pasikeitimas visiems abstrakčios būsenos komponentas buvo įvertintas kiekvienoje operacijoje;
- tikrina ar operacijos yra sėkmingos. Taip pat yra tikrinama ar yra išskirtos klaidingos dalies išankstinės sąlygos. Priešingu atveju gali atsirasti nesuderinamumai arba galimybė klaidingai specifikacijai;

- tikrina ar visos operacijų išankstinės sąlygos yra *tiesa*. Jei taip nėra, atsiranda problemų realizacijos metu, arba vėlesniuose palaikymuose;
- tikrina specifikacijos tipus, naudojant automatizuotus tipų tikrintojus. Jei specifikacijos tipai nėra korektiški, tai atsiranda formalus beprasmiškumas. Yra nemokamų (ZTC [30]) ir komercinių (FuZZ [31], CADiZ [32]) tipų tikrinimo įrankių;
- pasitelkiant validavimo įrodymus patikrinti, ar specifikacija elgiasi taip, kaip buvo tikimąsi. Jei teoremos yra įrodomos, tai tik patvirtina, jog specifikacija yra teisingai suprasta ir specifikuota. Priešingu atveju gali atsirasti problemos specifikuojant arba tiesiog tai parodo specifikacijos nesupratimą. Z/EVES [33, 34] yra mechaninis įrankis skirtas Z specifikacijos teoremoms įrodyti. Tačiau reikia įgūdžių efektyviam šio įrankio panaudojimui;
- specifikacija gali būti vizualizuojama tokiu įrankiu kaip ZANS [35], kuris yra vizualizatorius (angl. *animator*), veikiantis kartu su ZTC tipų tikrintoju. Vizualizavimas gali būti naudingas patikrinant ar specifikacija elgiasi taip kaip tikėtasi. Paprastai specifikaciją reikia perdirbti tam, kad jinai galėtų būti vizualizuojama. Tai galėtu būti naudingas pratimas ištaisant klaidas originalioje specifikacijoje.

Žemiau pateikta Z specifikacijas palaikančių įrankių lentelė (2.1 lentelė)

2.1 lentelė

Z specifikacijos validavimo įrankiai

Z specifikacijos validavimo įrankis	**Aprašymas**
FuZZ	**FuZZ** yra įrankių rinkinys, kuris padeda spausdinti, patikrinti Z specifikaciją, atsižvelgiant į Z galimybes ir tipų taisykles. Viena paketo dalis yra stiliaus opcija (angl. *style optinion*), skirta LaTeX tipams nustatyti, kuri apibrėžia extra LaTeX komandas ir turi šriftą, kuris turi specialius Z simbolius. Kita paketo dalis yra programa skirta analizuoti ir tikrinti specifikacijas, kurios parašytos naudojant šias komandas. FuZZ parodo tipų klaidų žinutes kartu su klaidos paaiškinimu.
CaDiz	**CADiZ** – įrankių komplektas, skirtas Z specifikacijai tikrinti. CADiZ parašytas troff (troff-validavimo įrankių šeima) pagrįsta notacija. Jis

2.1 lentelės tęsinys kitame puslapyje

	tikrina Z specifikacijos sintaksę bei tipų korektiškumą. Juos renka ir palaiko interaktyvų jų savybių tyrinėjimą. CADiZ sukurtas UNIX aplinkai ir gali būti naudojamas su troff ar LaTeX formatavimo sistemomis.
Zeta	**Zeta** – aplinka pagrįsta Z specifikacijomis, kuri pateikia susijungusią įrankių sistemą skirtą, Z specifikacijoms taisyti ir analizuoti. Zeta turi kelis integruotus įrankius, pvz., ESZ – tipų tikrintojas, ZAP – Z kompiliatorius bei adapteriai kurie skirti įrodinėti (Isabelle) ir tikrinti modelius (SMV).
ZUS	**ZUS** – interaktyvus vartotojo įrankis, kuris pateikia galimybes Z specifikacijai kurti, taisyti ir tikrinti. ZUS turi tekstinį bei grafinį redaktorių, testinių atvejų generatorių bei pagalbos sistemą. ZUS konstruoja ir taiso Z specifikaciją, kuri parašyta ZSL tekstiniame faile. Tekstinis redaktorius automatiškai transformuoja ZSL tekstinę specifikaciją į Z specifikaciją.
Z/EVES	**Z/EVES** – įrankis, skirtas Z specifikacijos kūrimui, tipų ir atvejų tikrinimui, redagavimui bei teoremų įrodinėjimui. Z/EVES įrankis LATeX rėžime leidžia vartotojui komponuoti specifikacijas naudojant LeTeX mark-up kalbą.
ERGO	**ERGO** – interaktyvus teoremų įjodinėtojas, pagrįstas sekų skaičiavimu. ERGO yra įdiegtas į Qu-Prolog, kuris yra Prologo praplėtimas. Jis turi Gumtree įrodymų interfeisą, kuris turi savo Gumtree taktinę kalbą ir kompiliatorių.
ZTC	**ZTC**- Z specifikacijos tipų tikrintojas. ZTC priima dvi įvesties formas: LaTeX arba ZSL. Viena ZSL neigiama savybė yra ta, kad ZTC klaidų žinutės neretai būna sunkiai suprantamos. Tai nėra komercinis įrankis, jis yra prieinamas visiems vartotojams.

2.1.2.3 Object-Z

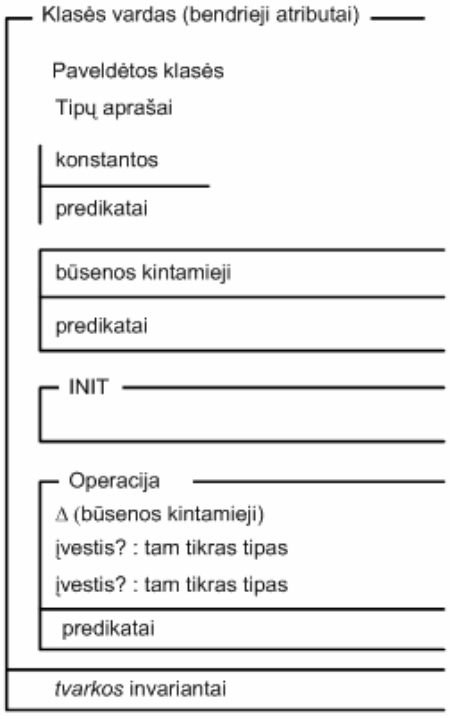
Object-Z [36] yra gerai žinoma kaip Z kalbos objektinis praplėtimas, tam kad sujungti objektines sąvokas (*Meyer 1988*). Ji prie Z prideda klasių ir objektų notacijas. Praplečiant Z semantikos pagrindą, Object-Z sistemos specifikaciją aprašo atskirais, nepriklausomais objektais. Object-Z buvo naudojama daugelyje programų, įskaitant realaus laiko sistemas telekomunikacijų srityje [37]. Tai objektais pagrįsta kalba, kurios pagrindas kaip ir Z kalbos yra aibių teorija.

Object-Z kalba:

- yra Z kalbos praplėtimas;
- sujungia operacijas į vieną būsenos schemą;
- įveda klasės sąvoką, susidedančią iš būsenos schemos kartu su sujungtom operacijomis, ir naudojama kaip objektų modelis;
- sudaro galimybę klasėms būti naudojamos kaip tipams;
- palaiko klasių paveldėjimą;

Klasės struktūra

Klasė (žr. 2.4 pav.) yra objektų modelis. Kiekvienas klasės objektas turi būseną, kuri atitinka klasės būsenos schemą, ir priklauso nuo būsenos perėjimų. Klasė apima tipus, konstantas ir kintamuosius, aprašant tam tikrų objektų būsenas sistemose su schemomis aprašančiomis jų vidines būsenas ir operacijas. Būsenos schema apibrėžia klasės invariantus bei būsenos atributus. Pradinė būsena aprašo klasės objektų pradinę būseną. Operacijų schemos apibrėžia klasės operacijų būsenas - „iki“ ir būsenas - „po“. Tvarkos invariantai apibrėžia operacijų tvarką.

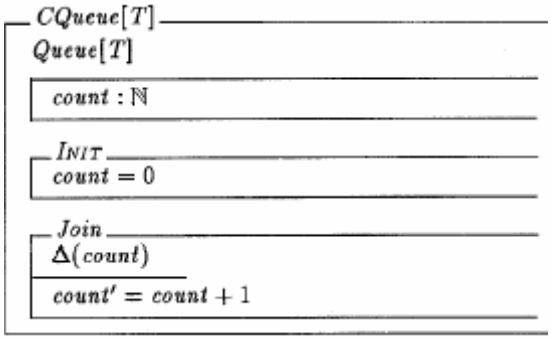


2.5 pav. Object-Z klasės struktūra

Paveldėjimas

Paveldėjimas yra mechanizmas, leidžiantis pažingsniui modifikuoti klasę tam, kad gauti naują klasę. Nauja klasė yra arba praplėtimas, arba originalios klasės specializacija. Nauja klasė gali būti gaunama iš vienos ar kelių egzistuojančių klasių. Tai reiškia, kad visos originalios klasės schemų savybės yra ir naujos klasės atitinkamų schemų savybėmis. Gautos klasės apibrėžimai yra sujungti su tomis, iš kurių atėjęs paveldėjimas. Paveldėjimas leidžia pervadinti atributus, taigi klasių sutapimas gali būti išspręstas pervardinant.

Kaip paveldėjimo pavyzdys (žr. 2.5 pav.) yra pateikiama *eilės* specifikacija, kuri turi skaitiklį, kuris skaičiuoja elementų skaičių, kurie buvo prie jo prijungti.



2.6. pav. Object-Z klasė „Cqueue“

Suprantama, jog paveldėtos klasės *Queue[T]* apibrėžimai yra tokie patys, kaip klasės *CQueue[T]*. Tipas *Length,* konstanta *max* ir *CQueue[T]* operacija yra ekvivalentiškos aprašytoms klasėje *Queue[T]*. Jei atsirado lokalių tipų ar konstantų tokiu pačiu vardu paveldėtose ir pavedimosiose klasėse, tai jie gali būti semantiškai identifikuoti. Būsenos schema, pradinė būsenos schema ir operacijos yra sujungiamos. Taip pat Object-Z palaiko sudėtinį paveldėjimą.

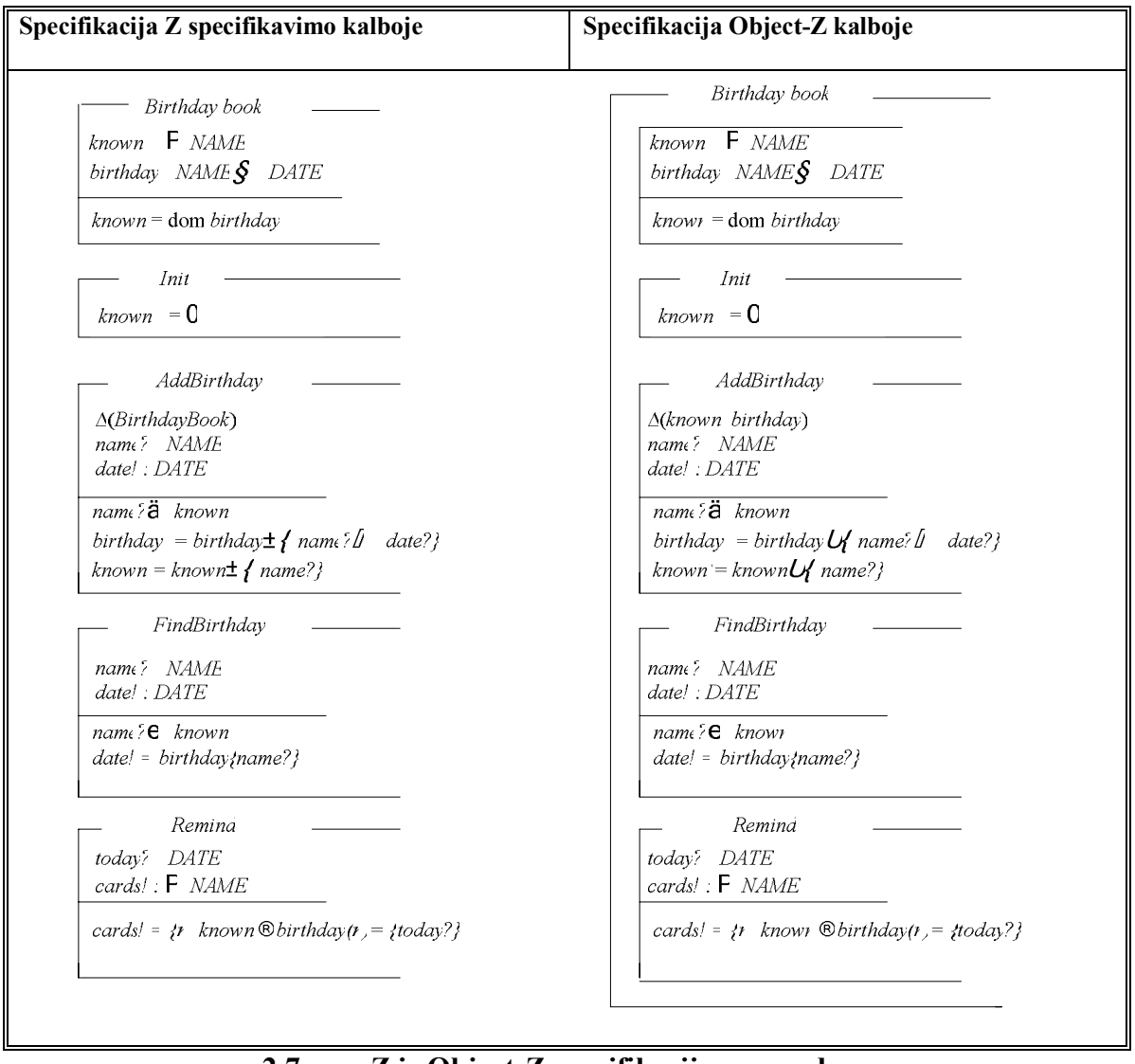
2.2 Z ir į Object-Z specifikavimo kalbų palyginimas

Apžvelgus Z ir Object-Z specifikacijos teoriją, galima pateikti šių formaliųjų metodų savybes, jas palyginti išskiriant panašumus ir skirtumus [38, 39] .

Panašumai. Z ir Object-Z yra formalios specifikavimo kalbos**.** Tiek Z, tiek Object-Z semantika pagrįsta matematikos aibių teorija. Objektai ir klasės yra laikomos kaip aibės. Kadangi Object-Z pagrindas yra Z specifikavimo kalba, todėl ji turi tokias pat kaip Z būsenos schemas, operacijas.

Skirtumai. Pirmiausia Object-Z specifikavimo kalba palaiko objektinį modelį, todėl ji yra objektinė programavimo kalba. Z specifikavimo kalba objektinio modelio nepalaiko, kadangi ji yra modeliais orientuota kalba. Z specifikacija yra aprašoma schemomis, tuo tarpu Object-Z specifikacija aprašoma klasėmis. Z specifikavimo kalba aprašo pilnai visą specifikaciją ir ją traktuoja kaip vieną modelį. Object-Z specifikavimo kalba specifikacija aprašoma atskirais paragrafais, t.y klasėmis.

Žemiau pateiktas pavyzdys, kuriame specifikacija aprašyta Z ir Object – Z kalbose (žr. 2.7 pav.). Galima pastebėti Object-Z kalboje atsirado klasė, kuri turi būsenos schemas ir operacijas. Tuo tarpu Z kalba parašytoje specifikacijoje yra atskiros schemos ir operacijos. Tipų aprašai Z kalboje aprašomi specifikacijos pradžioje, prieš schemų aprašymus. Object-Z kalboje tipai ir paveldėtos klasės apsirašomos pačioje klasėje prieš būsenos schemas ir operacijas. Object-Z klasės būsenos schema neturi schemos vardo. Object -Z inicializavimo schema skiriasi nuo Z. Δ notacijoje nurodytas kiekvienas būsenos kintamasis, kuris yra keičiamas, tuo tarpu Z yra nurodomas visos schemos vardas.

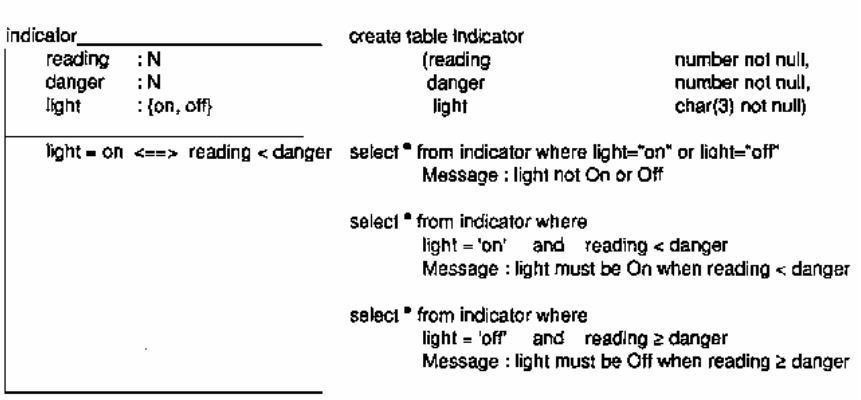


2.7 pav. Z ir Object-Z specifikacijos pavyzdys

2.3 Specifikacijos transformavimas į objektinę programavimo kalbą

Transformacijos reikalingos tam, kad iš specifikacijos gauti programos kodą atitinkantį reikalavimus, tokiu būdu sumažinant žmogaus daromų klaidų kiekį rašant programas.

Transformacijos apima pradinės specifikacijos parašytos Object-Z kalba pavertimą į programos kodą, tam tikroje programavimo kalboje. Yra nemažai publikacijų, kurios aprašo Z specifikacijų transformavimą į vieną ar kitą programavimo kalbą, pvz.: SQL, BON/Eiffel ir Kitas. Z specifikacijos transformavimo į SQL [40] metodika siūlo kiekvieną Z specifikacijos schemą atvaizduoti į SQL lentelę, schemos predikatus transformuoti į vieną ar kelis SQL predikatus (žr. 2.8 pav.).



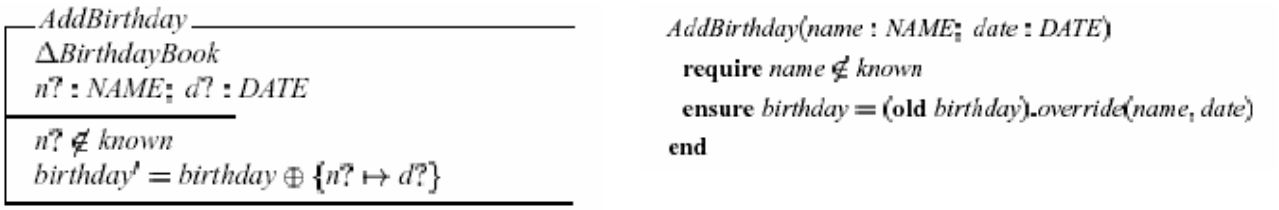
2.8 pav. Z specifikacijos transformacija į SQL

Z operacijos schemos predikatas transformavimo į SQL sakinį pavyzdys pateiktas žemiau:

Z: *contents' = contents + ...*

SQL *update container 1 set contents = (select contents + ... from container*

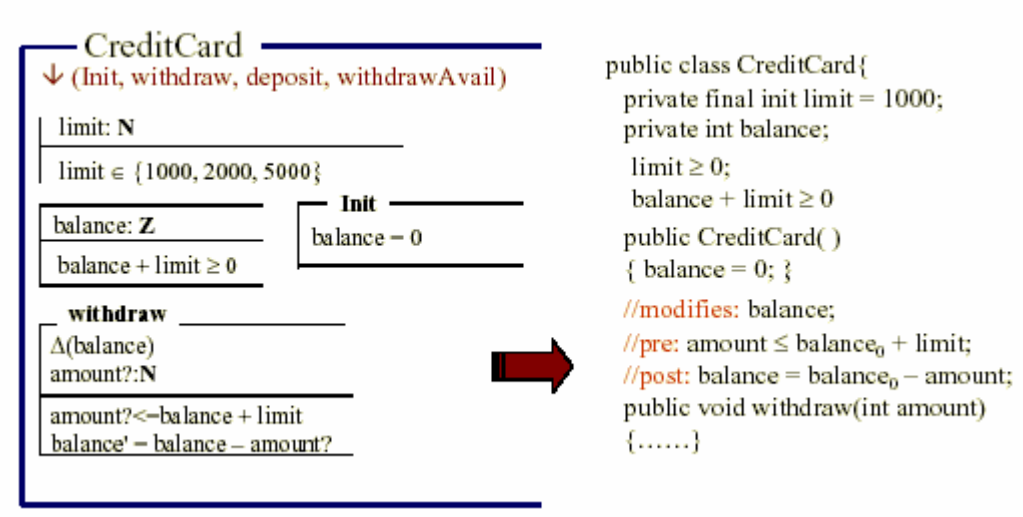
Yra ir kitų metodikų aprašančių Z specifikacijos transformavimą į vieną ar kitą nesudėtingą programavimo kalbą [41], tačiau susiduriame su sunkumais norint transformuoti Z specifikaciją į objektinę programavimo kalbą. Kaip pavyzdys galėtų būtų Z specifikacijos transformavimas į BON [42] (žr.2.9 pav.).



2.9 pav. Z specifikacijos transformavimas į BON

Esminė šio transformavimo problema yra tai, jog skirtingai nuo BON, Z nėra objektinė kalba. Taigi iš Z susijusių schemų rinkinio reikia parašyti į klases. Tokiu būdu, jei operacijos schemos dalinasi būsenos schemomis, tai transformuotos operacijos ir kintamieji turėtų priklausyti tai pačiai klasei. Ši transformacija būtų daug lengviau atliekama, jei Z būtų objektinė kalba. Tokiu atveju, jei vietoje Z naudotume objektinę specifikavimo kalbą Object-Z, kuri specifikacijas aprašo klasėmis, tai išvengtume papildomų pertvarkymų, kurie atliekami su Z specifikacija. Tačiau publikacijose yra aprašoma BON transformacija į Object-Z, ypač daug darbų kuriuose pateikiamos UML transformavimo į Object-Z metodikos [43, 44]. Rečiau yra aptinkamos publikacijos, kuriose būtų siūlomos Obejct-Z transformavimo metodikos į objektines programavimo kalbas. Transformuojant

Object-Z specifikaciją į JAVA yra siūloma transformavimo metodika [45], kur specifikacijos transformavimas į JAVA programavimo kalbą yra atliekamas rankiniu būdu. Object-Z konstantos atvaizduojamos į Java klasės atributus, būsenos schemos aksiomos į invariantus, pradinė schema į klasės konstruktą ir t.t (žr. 2.10 pav.)



2.10 pav. Object-Z transformacija į JAVA

Kadangi programinės įrangos kodas parašytas objektine programavimo kalba, todėl transformacijai pasirinkta Object-Z specifikavimo kalba. Išanalizavus Object-Z transformavimo į įvairias programavimo kalbas literatūrą galima pasiūlyti disciplinuotą metodinį Object-Z panaudojimo būdą, kuriuo būtų lengvai atliekamas transformavimas į programinės įrangos kodą.

3 Z IR OBJECT-Z PANAUDOJIMAS FORMALIZUOJANT INFORMACINES SISTEMAS

3.1 Z specifikacija

Šiame darbe reikalavimus informacinei sistemai siūloma aprašyti Z specifikavimo kalba. Z specifikacijos aprašymas susideda iš 4 dalių:

- aibės, duomenų tipai, konstantos;
- būsenų aprašymai;
- pradinė būsena;
- operacijos.

Z specifikacijos aprašymui naudojamos Cogito priemonės [47]. Pagrindinis Cogito projekto tikslas yra pateikti integruotą metodiką ir įrankių komplektą palaikant formalios programos išsivystymą. Cogito metodika nukreipta į specifikacijas, jų kūrimą ir vystymą bei įrankių konstrukcijas. Cogito metodikos pačiame centre yra *Sum* programa, kuri pagrįstas Z specifikavimo kalba. Šiame darbe formalių reikalavimų specifikavimui buvo panaudota *Sum* programa.

3.2 Specifikacijos validavimas

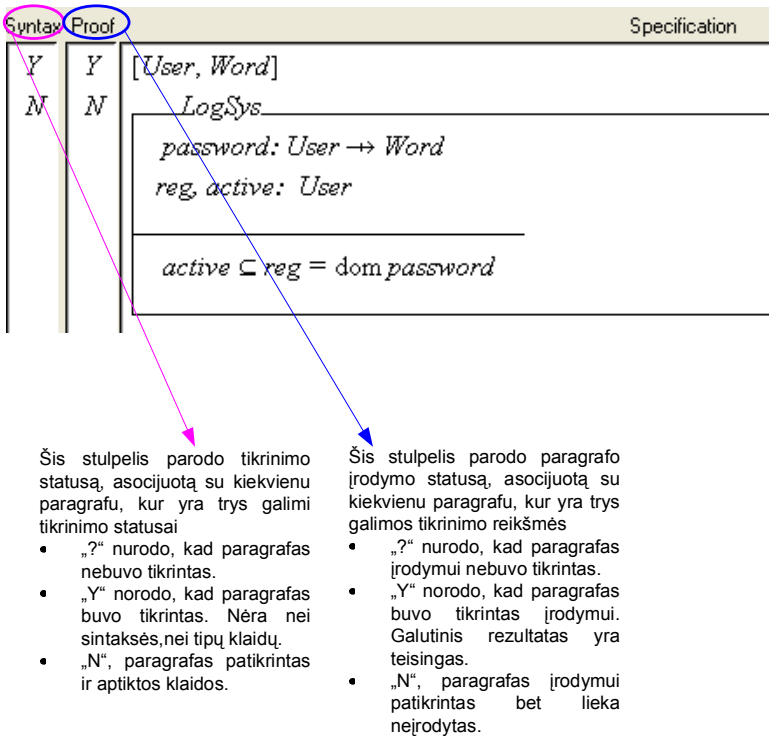
Šiame darbe specifikacijai validuoti yra siūloma panaudoti Z/EVES įrankį. Z/EVES yra interaktyvi sistema skirta Z specifikacijai sudaryti, tikrinti ir analizuoti. Vienu metu vienas specifikacijos paragrafas gali būti suvedinėjamas ir tikrinamas; patikrintas specifikacijos paragrafas gali būti peržiūrėtas ir ištaisytas; bet kuriuo metu gali būti formuojamos ir įrodinėjamos teoremos.
Z/EVES įrankio savybės:

- specifikacijų kūrimas;
- sintaksės tipų, atvejų tikrinimas;
- specifikacijos ir jos dalių redagavimas;
- specifikacija gali būti tikrinama visa iš karto arba pagal pasirinktus paragrafus;
- visada specifikacija gali būti patikslinama ir redaguojama;
- specifikacija gali būti iš naujo tikrinama;
- teoremų rašymas ir įrodinėjimas;
- specifikacija ar pavieniai paragrafai gali būti eksportuojami į Microsoft Word *rtf* formato bylą.

Jei specifikacija užrašyta formaliais ženklais, tai dar nereiškia kad ji yra teisinga, pilna ar net prasminga. Specifikacijoje gali būti keletas skirtingų klaidų, pradedant nuo trivialių rašybos klaidų iki subtilių nesuderinamumo klaidų. Z/EVES gali padėti specifikuotojui išvengti šių klaidų.

Sintaksės ir tipų klaidos

Z specifikavimo kalba turi pakankamai sudėtingą sintaksę. Todėl yra nesunku, ypač nepatyrusiam Z vartotojui padaryti nemažai klaidų. Z/EVES kaip ir daugelis kitų Z įrankių aptinka ir praneša apie tokias klaidas. Tačiau Z/EVES kiekvieną parašytą specifikacijos paragrafą gali iškarto patikrinti ir jei būtina pataisyti. Žemiau pateiktas specifikacijos pavyzdys (žr. 3.1 pav.). Kaip matyti iš paveikslėlio antrame paragrafe yra klaidų.

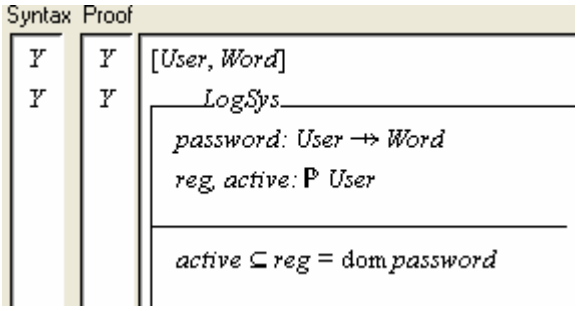


3.1 pav. Z/EVES Specifikacijos pavyzdys

Pasirinkus paragrafo komandą*“ Show Errors“* ekrane yra pateikiamos klaidų žinutės:

Error *RelationArgType* (line 2) [Type checker] : type of **local** *active* is not the same as the domain of the relation (**global** _⊆_) .
Error *RelationArgType* (line 2) [Type checker] : type of **local** *reg* is not the same as the range of the relation (**global** _⊆_) .
Error *TypesNotSame* (line 2) [Type checker] : types of **local** *reg* and **global** dom **local** *password* are not the same.

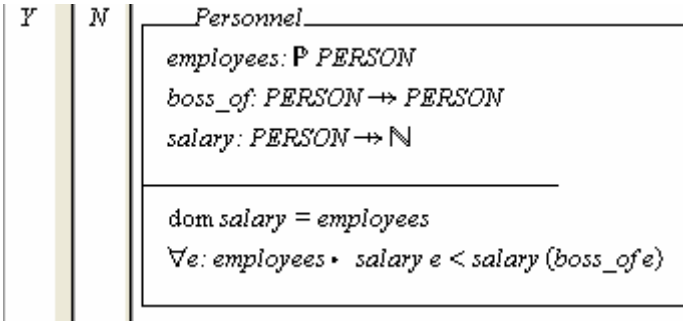
Neabejotinai *active* ir *reg* nėra tinkamo tipo poaibio ryšiui ir šios dvi lygybės pusės yra skirtingo tipo. Taip galėjo atsitikti, dėl to kad praleista Π operacija deklaruojant *active* ir *reg* (tai yra dėl to kad yra parašyta reg, *ative : User,* vietoje *reg, active : Π User*). Ištaisius šią klaidą gauname teisingą specifikacijos paragrafą (žr. 3.2 pav.).



3.2 pav. Ištaisyta „LogSys“specifikacijos schema .

Srities klaidos

Z/EVES nagrinėja kiekvieną parašytą paragrafą, tikrina funkcinį aprašymą.Žemiau pateiktas specifikacijos schemos paragrafo pavyzdys (žr. 3.3 pav.).



3.3 pav. Z/EVES specifikacijos schemos pavyzdys

Sintaksiškai ši schema yra aprašyta korektiškai, tačiau turi neįrodytą tikslą. Z/EVES pateikia teoremą *Personnel$DomainCheck*, su tikslo predikatu:

theorem axiom *Personnel$domainCheck*

$$employees \in \mathbb{P}\ PERSON$$
$$\wedge\ boss\_of \in PERSON \pfun PERSON$$
$$\wedge\ salary \in PERSON \pfun \mathbb{N}$$
$$\wedge\ \mathrm{dom}\ salary = employees$$
$$\wedge\ e \in employees$$
$$\Rightarrow e \in \mathrm{dom}\ salary \wedge e \in \mathrm{dom}\ boss\_of \wedge boss\_of\ e \in \mathrm{dom}\ salary$$

Šį predikatą galima supaprastinti naudojant komandą “*rewrite”*.

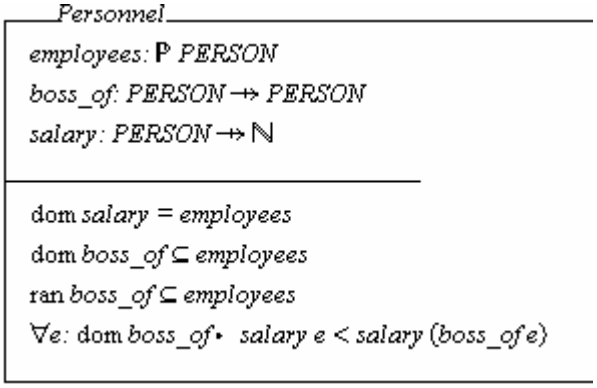
$$employees \in \mathbb{P}\ PERSON$$
$$\wedge\ boss\_of \in PERSON \pfun PERSON$$
$$\wedge\ salary \in PERSON \pfun \mathbb{N}$$
$$\wedge\ \mathrm{dom}\ salary = employees$$
$$\wedge\ e \in employees$$
$$\Rightarrow e \in \mathrm{dom}\ boss\_of \wedge boss\_of\ e \in \mathrm{dom}\ salary$$

Viena iš jungčių *e ε dom salary* buvo pašalinta, tačiau kitos dvi dar liko. Srities tikrinimo sąlygos parodo, kad schemoje yra praleisti kai kurie suvaržymai. Abi sąlygos turi ryšį su galutine schemos sąlyga, A *e: employees • salary(e) < salary(boss_of e)*. Pirma , *e ε dom boss_of* siejasi su išraiška *boss_of e*. Šiame kontekste *e* yra žinomas kaip darbuotojas. Tačiau ne kiekvienas darbuotojas

turi viršininką, pvz., prezidentas neturi. Taigi turėtu būti aprėpiami tik tie darbuotojai, kurie turi viršininkus.

Sekanti salyga, *boss_of e ε dom salary*, turi ryšį su išraiška salary (*boss_of e*). Tačiau juk ir viršininkas turi uždarbį. Specifikacijoje tai nėra aprašoma, todėl gaunasi, kad viršininkas nėra darbuotojas.

Klaidos yra ištaisomos ir specifikacijos paragrafe atsiranda dar dvi predikato eilutės (žr. 3.4 pav.)

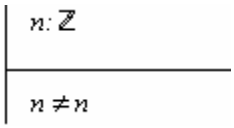


3.4 pav. Schema „Personnel“

Nesuderinamumas, prieštaringumas

Specifikacijos paskirtis - būti tam tikros įmanomos sistemos modeliu. Specifikacija yra prieštaringa jei ji neturi modelių.

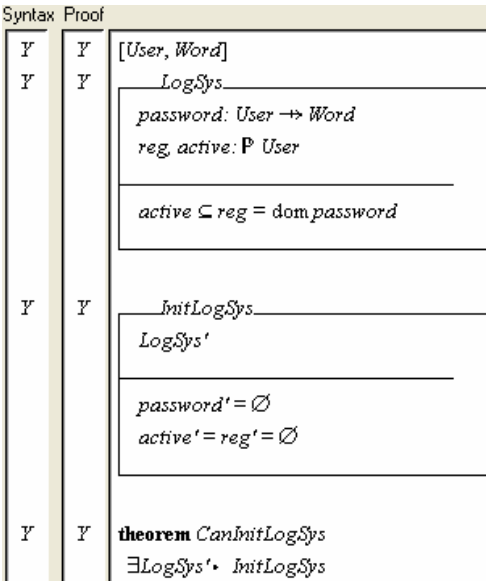
Specifikacija gali būti prieštaringa kai kuriose aksiominėse, bendrosiose dalyse, ar predikatuose. Pavyzdžiui, bet kuris specifikacijos paragrafas, kuris neturi modelių, neturi jokios *n* įmanomos reikšmės (žr. 3.5 pav.).



3.5 pav. Z/EVES paragrafo pavyzdys

Tokiu atveju prieštaringumas yra akivaizdus.

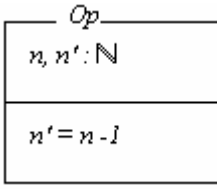
Pavyzdžiui, schema gali turėti predikatą, kuris nėra patenkinamas. Yra nesunku pateikti *S* schemos patenkinimą, naudojant *E S• true* predikatą. Daugelis specifikacijų pateikia inicializavimo formos *Init_S ⁄ [S | P]* schemą. Tokiu būdu **teorema** *E S•Init_S* parodo, kad yra įmanoma pradinė būsena (žr . 3.6 pav).



3.6 pav. Specifikacijos fragmentas su įrodyta teorema

Išankstinės sąlygos

Z stilius aprašant operacijas yra reliacinis. Operacija yra apibrėžiama kaip ryšys tarp pradinės ir galutinės būsenos. Toks stilius turi daugybę privalumų, bet yra vienas trūkumas, jog operacijos išankstinės sąlygos yra paliekamos kaip implicitinės. Pavyzdžiui, operacija *O*p (žr. 3.7 pav.) aprašo tik vieną situaciją, kur pradinė *n* reikšmė nėra nulis. Tokios situacijos paprasčiausia interpretacija galėtų būti, pvz., *Op* iškviečia *n=0*. Nėra aišku, kas gali nutikti. Pavyzdžiui, operacija gali katastrofiškai žlugti arba gali niekada nesibaigti.

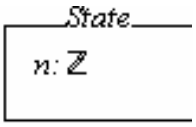


3.7 pav. Operacija „Op“

Duota Z schema *Op ⁄[/ΔS ; in? : IN; out ! : OUT],* kur schemos *pre Op* yra ekvivalentiška *E S‘ : out ! : OUT • Op*, ir aprašo pradines būsenas, kurioms yra galima galinė būsena ir išvesties duomenys. Jei operacija yra užbaigta, tai ji gali būti įvykdyta bet kurioje pradinėje būsenoje ir su bet kokiais įvesties duomenimis. Taigi *A S; in ? : IN• pre Op* turėtų būti teorema. Bandant įrodyti šią spėjamą teoremą galima atskleisti daugybę praleistų hipotezių.

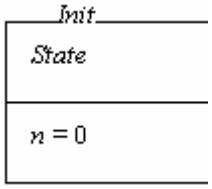
Invariantai

Invariantų sąlygos yra tiesiogiai išreikštos Z specifikacijoje, taigi Z/EVES nereikia jokių priemonių, kad galėtų išreikšti jų įrodymus. Žemiau nagrinėjamas invarianto įrodymo pavyzdys. Sistema turi paprastą būsena, kuri priklauso nuo skaičių (žr 3.8 pav.).



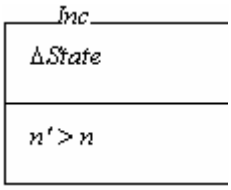
3.8 pav. Specifikacijos būsena „State“

Aprašoma *Init* schema, kurioje n priskiriamas 0 (žr 3.9 pav.),



3.9 pav. Pradinė schema „Init“

ir operacija *Inc* , kuri skirta skaičių padidinti (žr 3.10 pav.).



3.10 pav. Specifikacijos operacija „Inc“

Akivaizdu, *kad n 70* yra sistemos invariantas. Taigi galima parašyti teoremą, kuri tai patikrintų:

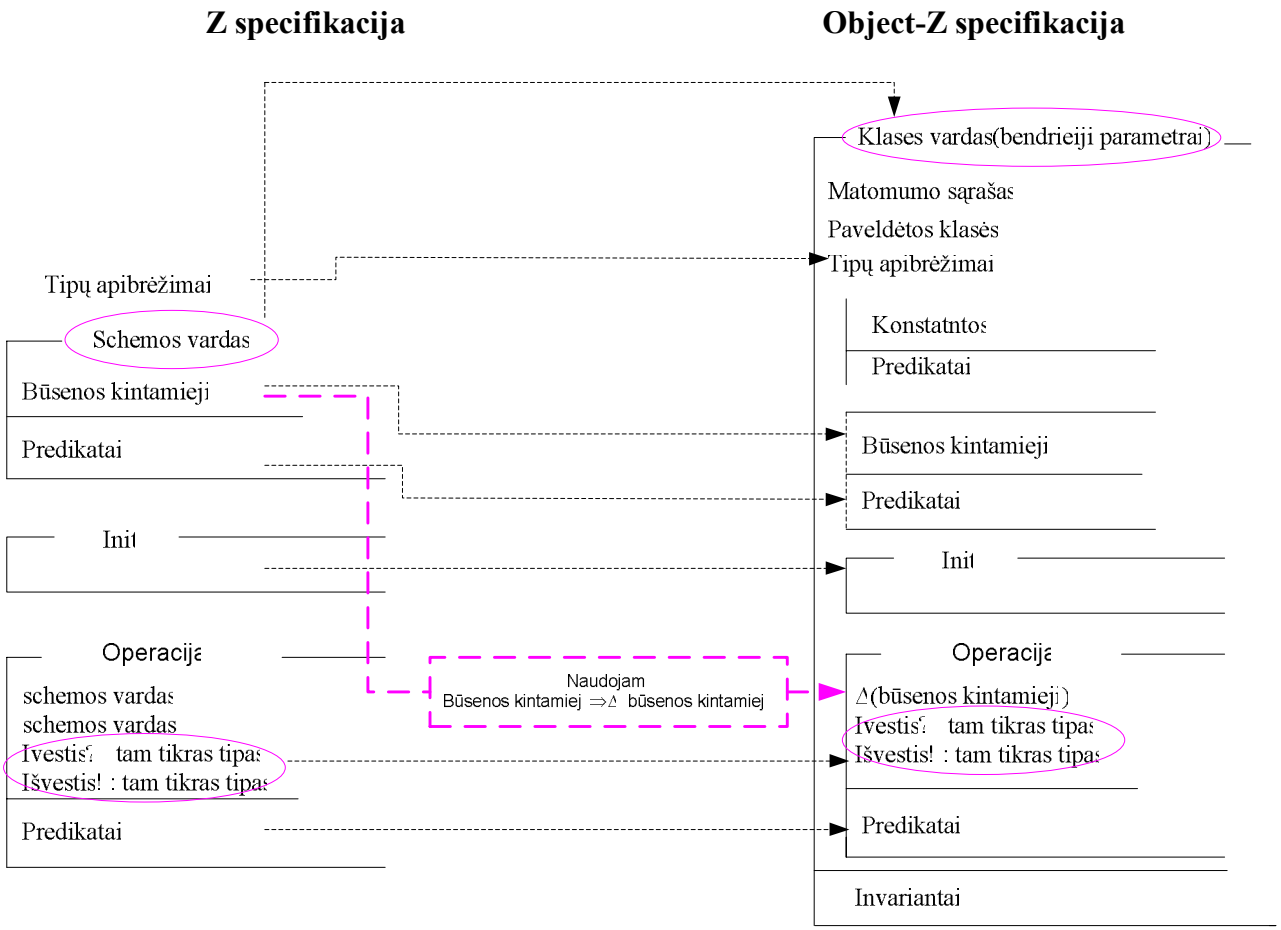
theorem *SystemInvariant*
$(Init \Rightarrow n \geqslant 0) \land (Inc \land n \geqslant 0 \Rightarrow n' \geqslant 0)$

Visos specifikacijos teoremos yra saugomos Z/EVES teoremų lange. Sugeneruotos teoremos gali būti aksimos, kurios yra faktai arba tikslai, kurie yra reikalingi įrodymams.

3.3 Z transformacija į Object-Z

Išanalizavimus ir palyginus Z ir Object-Z formalias specifikavimo kalbas galima pasiūlyti disciplinuotą būdą, kaip nuo Z specifikacijos pereiti prie objektinės Object-Z specifikacijos. Žemiau yra pateikta Z specifikacijos transformavimo į Object-Z metodika (žr. 3.11 pav.).

Visų pirma, Z kalba parašytoje specifikacijoje yra atskiros schemos ir operacijos. Z specifikacija yra suprantama kaip vienas bendras modelis. Object–Z specifikavimo kalboje atsiranda klasės sąvoka. Specifikacija parašyta Object–Z kalboje susideda iš kelių atskirų klasių. Klasė kuri turi tipus, paveldėtas klases, būsenos kintamuosius, schemas bei operacijas. Kaip matyti paveiksle Object-Z būsenos schema neturi vardo. Z specifikacijos schemos vardas virsta Object–Z klasės vardu. Tipų aprašai Z kalboje aprašomi specifikacijos pradžioje. Object-Z kalboje tipai ir paveldėtos klasės apsirašomos pačioje klasėje prieš būsenos schemas ir operacijas. Toliau surašomos atitinkamos operacijos kaip ir Z taip ir Object-Z kalboje. Object-Z Δ notacijoje nurodomi visi būsenos kintamieji, kurie yra keičiami, Z specifikacijoje yra nurodomas visos schemos vardas.



3.11 pav. Z specifikacijos transformavimo į Object-Z metodika

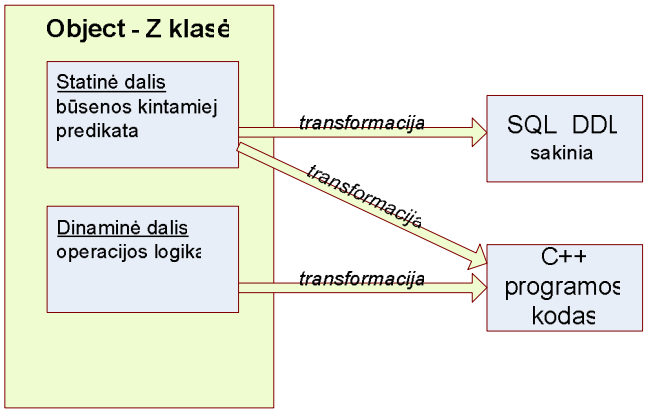
3.4 Object-Z transformacija į objektinę programavimo kalbą

Object-Z yra objektinė specifikavimo kalba. Šioje formalioje kalboje atsiranda klasės ir objektai, ko nėra Z specifikavimo kalboje. Objektinė Z specifikavimo kalba dažniausiai naudoja objektinėms sistemoms specifikuoti, todėl ir transformaciją į objektinę programavimo kalbą yra lengviau atlikti iš Object-Z kalboje parašytos specifikacijos.

Objektinė orientacija formaliose specifikacijose yra viena iš plačiausiai tyrinėjamų sferų programinėje technikoje. C++ programavimo kalba yra viena iš populiaresnių objektinių programavimo kalbų, kuri yra C programavimo kalbos praplėtimas. Kadangi programinės įrangos kodui rašyti buvo pasirinkta C++ programavimo kalba, todėl galima transformuoti Object-Z specifikaciją sudarytą iš klasių į C++ sąsajos klases, pvz., į duomenų elementus ir funkcijų prototipus ir t.t. Kadangi tiek Object-Z, tiek C++ yra objektinės kalbos ir abi sudarytos iš klasių, todėl specifikacijos transformavimas į programinės įrangos kodą yra gana nesudėtingas.

Šiame darbe yra siūloma Object-Z specifikacijos transformavimo į programinės įrangos kodą metodika. Kompiuterių klasių užimtumo informacinės sistemos duomenų struktūros aprašytos SQL DDL sakiniais, valdymo logika aprašyta C++ programavimo kalba. Dėl to specifikacijos transformavimas (žr. 3.12 pav.) atliekamas dviem etapais :

- Specifikacijos statinės dalies transformavimas į SQL DDL sakinius;
- Specifikacijos statinės ir dinaminės dalies transformavimas į C++ programavimo kalbą.



3.12 pav. Object-Z specifikacijos transformavimas į programinės įrangos kodą

Pagrindinės Object-Z specifikacijos transformavimo į C++ programavimo kalbą taisyklės pateiktos žemiau (žr. 3.1 lentelę.).

3.1.lentelė

Pagrindinės Object-Z specifikacijos transformavimo į C++ programavimo kalbą taisyklės

Object-Z	**C++**
Klasės schema „Klasės vardas“	⟼ *public klasė* „Klasės vardas“
Paveldėjimas	⟼ *public, protected* paveldėjimas
Konstantos	⟼ Konstantos
Atributai	⟼ Klasės atributai
Būsenos schemos aksiomos	⟼ *Invariantai*
Pradinė schema	⟼ Klasės konstruktorius
Operacijos schema	⟼ Metodo aprašas
Δ-sąrašas	*Modifikacijų sąrašas*
Būsenos – „iki“	*Sąlyga –„iki „*
aksiomos	*Sąlyga -„po“*
Matomumo (angl. *visibility*) **sąrašas**	⟼ *public/ protected* prefiksai

Kiekvienas Obejct-Z klasės pavadinimas tampa C++ klasės pavadinimu.

Object-Z paveldėjimas transformuojami į C++ paveldėjimą. Šis paveldėjimas yra *public* ar *protected*, priklausomai nuo matomumo sąrašo. Jei klasė yra paveldėta iš kitos klasės bei tos klasės klientams padaro matomą paveldėtų savybių sąrašą, tokiu atveju yra *public* paveldėjimas.Pavyzdžiui:

tempale <**class** X, **class**Y>

class Relation : **public** Set <Pair <X,Y>> ,

kur C++ klasė Pair pateikia dekartinę sandaugą X ξ Y

Atributai (konstantos ir kintamieji) aprašyti klasės schemoje yra transformuojami į C++ klasės atributus, išlaikant tą patį konstantų ir kintamųjų savitumą. Taigi visos Object-Z klasės

schemos konstantos yra C++ klasės konstantos. Object-Z tipų aprašymai tampa C++ tipų aprašymais, pateiktais *protected* dalyje.

Object-Z būsenos schemos aksiomos yra transformuojamos į C++ klasės invariantus.

C++ programavimo kalboje elementų inicializavimo funkcija vadinama klasės kontruktu, kuris yra iškviečiamas kiekvieną kartą kur klasės objektas sukuriamas operatorių *new*. Todėl pradinė Object-Z schema atitinka C++ klasės konstruktorių.

Operacijos schemos transformuojamos į C++ specifikacijas tokiu būdu: Δ- sąrašas transformuojamas į „modifikacijų“ konstruktą, būsenos – „iki“ aksiomos atitinka C++ būsenas – „iki“, ir likusios operacijos schemos aksiomos transformuojamos į metodo būsenas – „po“. Modifikacijų sąraše nurodomas kintamųjų, kurie gali pasikeisti sąrašas. Sąlygos – „iki“ yra tokios sąlygos, kurias turi patenkinti C++ įėjimo kintamieji. Sąlygos –„po“ yra tokios sąlygos, kurias atlikus operaciją turi tenkinti išėjimo kintamieji ir/arba objekto reikšmės. Matomos operacijos yra *public*, likusios yra *protected*.

Matotmumo (angl. *visibility*) sąrašas nusako kokie atributai ir operacijos yra matomi klasės vartotojams. C++ programavimo kalboje elementai aprašyti kaip *public* gali būti prienami visiems vartotojams, tuo tarpu elementai aprašyti kaip *protected* ar *private* nėra laisvai prieinami. Elementai, atitinkamai pagal matomumo sąrašo savybes yra sukuriami kaip *publi*c arba *protected*. Būna atvejų, kaip Object-Z nėra matomumo sąrašo, tokiu atveju atributai sukuriami kaip *protected* operacijos yra *public*.

3.2.lentelė

Pagrindinės Object-Z specifikacijos transformavimo į SQL DDL sakinius taisyklės

Object-Z		SQL
Klasės schema „Klasės vardas“	⟼	Lentelės vardas
Atributai	⟼	Lentelės stulpeliai
Invariantai	⟼	Suvaržymai (angl. *constraints*)

Visi Object-Z klasės būsenos kintamieji ir predikatai transformuojami į SQL DDL sakinius. SQL DDL sakiniu CREATE TABLE ne tik apibrėžiama duomenų struktūra, bet ir sukuriamas realus duomenų bazės objektas. Lentelės apibrėžimas yra vienkartinis veiksmas. DBVS neleidžia kurti lentelės su tokiu pat vardu kaip jau egzistuojančios. Kuriant lentelę būtina nurodyti lentelės vardą, lentelės stulpelių vardus ir jų tipus. Object – Z klasės schemos vardas atitinka SQL lentelės vardą, atributai transformuojami į SQL lentelės stulpelius, invariantai į suvaržymus (žr. 3.2 lentelė). Daugumą lentelės savybių galima nurodyti vėliau jau sukurtai lentelei. Tai atliekama sakiniu ALTER TABLE. Juo galima papildyti lentelę, pavyzdžiui nauju stulpeliu. Tačiau keičiant lentelės struktūrą, susiduriame su suvaržymais.

4 Z SPECIFIKAVIMO METODO PANAUDOJIMAS, KURIANT KOMPIUTERIŲ KLASIŲ UŽIMTUMO INFORMACINĘ SISTEMĄ

4.1 Kompiuterių klasių užimtumo informacinės sistemos neformalūs reikalavimai

Šiuo metu studijų procesas kompiuterių klasėse organizuojamas, remiantis daugiamete dirbančiojo personalo patirtimi. Užsiėmimų tvarkaraščiai sudaromi tik konkretaus fakulteto kompiuterių klasėms, nedisponuojama informacija apie galimybę pravesti užsiėmimus kitose kompiuterinėse klasėse. Tvarkaraščiai sudaromi neatsižvelgiant į programinę įrangą įdiegtą kompiuterių klasėse, neįvertinamas turimų licenzijų kiekis. Iškyla problemos aptarnaujančiam personalui, paruošiant kompiuterines klases užsiėmimų pravedimui. Universitete nėra sukaupta informacijos, kokie studijų moduliai mokymo procese naudoja informacines technologijas ir kokie taikomųjų programų paketai reikalingi. Planuojant naujų kompiuterių klasių steigimą fakultetuose, darosi aktualu išanalizuoti jau esamų kompiuterinių klasių užimtumo lygį. Tam tikslui yra sudarytas kompiuterių klasių užimtumo IS neformalus aprašymas.

Kompiuterių klases, kuriose vedami užsiėmimai, apibūdina tokie rodikliai:

- adresas;
- fakulteto pavadinimas;
- kompiuterių klasės numeris;
- klasės programinė įranga;
- kompiuterių skaičius.

Kompiuterių klasėse vedamų modulių sąrašas apibūdinamas tokiais rodikliais:

- modulio pavadinimas;
- modulį vedančio dėstytojo duomenys;
- modulį studijuojančių studentų kiekis;
- modulio pravedimui reikalinga programinė įranga;
- fakultetas;
- modulio pravedimo pradžios data;
- modulio pravedimo pabaigos data.

Universiteto programinę įrangą apibūdina tokie rodikliai:

- programinės įrangos pavadinimas;
- licenzijos.

Užsiėmimą apibūdina tokie rodikliai:

- modulis;
- kompiuterių klasė;

- užsiėmimo pravedimo data;
- užsiėmimo pravedimo laikas.

Su aprašytais objektais yra atliekamos tokios pagrindinės operacijos, kaip naujos klasės, modulio, programinės įrangos ar užsiėmimo įtraukimas, esančio pašalinimas, užsiėmimų rezervavimas, įvertinant galimybę pravesti užsiėmimus konkrečioje klasėje, konkrečiu laiku. Taip pat atliekamos tokios funkcijos, kaip pasirinkto laikotarpio kompiuterių klasių tvarkaraščio peržiūrėjimas, vartotojo prisijungimas.

Užsiėmimų pravedimui kompiuterių klasėse yra vykdomos tokios sąlygos:

- kiekvienas modulis gali būti vedamas tik tokioje kompiuterių klasėje, kurioje yra įdiegta tam moduliui reikalinga programinė įranga;
- kiekvienam moduliui rezervuojamų klasių užsiėmimų datos patenka į intervalą, nusakomą modulio pravedimo pradžios data ir modulio pravedimo pabaigos data;
- rezervuojant klasę studentų skaičius neturi viršyti darbo vietų skaičiaus kompiuterių klasėje;
- prieš rezervuojant klasę turi būti nurodoma, kokiu būdu bus rezervuojamos datos užsiėmimų pravedimui ar cikliniu, nurodant pirmą datą ir kitos bus rezervuojamos automatiškai kas 7(14) dienų, ar nurodant konkrečią datą.

4.2 Informacinės sistemos reikalavimų aprašymas Z specifikavimo kalba

Panaudojant Z specifikavimo metodiką sukuriamas klasių užimtumo formalus modelis, kuris nusako reikalavimus kompiuterių klasių užimtumo informacinei sistemai. Z schemų aprašymui panaudotos standartinės programinės priemonės Cogito .

Specifikacijos pradžioje surašomi tipai, po jų surašomos specifikacijos schemos: Pr_Iranga, Klase, Modulis, Rezervacija. Po būsenos schemų aprašomos visos operacijos: Pr_Irangos_itraukimas_ok, Klases_itraukimas_ok, Modulio_itraukimas_ok, Rezervacija_ok, Rezervacija_kasndienu_ok Pr_Irangos_itraukimas_klaida, Pr_Irangos_ismetimas_ok, Pr_Irangos_ismetimas_klaida, Klases_itraukimas_klaida ir kitos.

Tipų aprašai

Data == N

Data yra natūrinis skaičius

Laikas == N

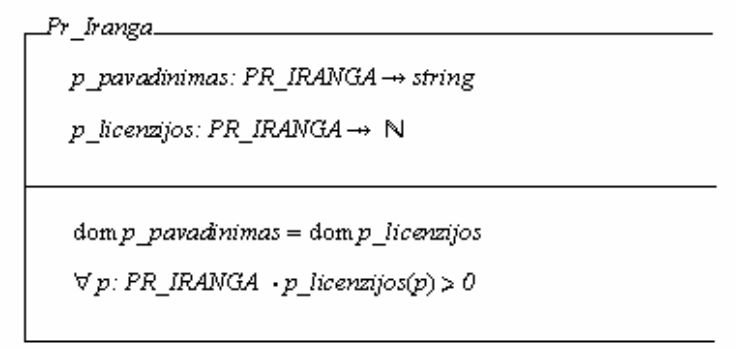
Laikas yra natūrinis skaičius.

Adresas == *string*

Adresas yra simbolių (angl. *string*) tipo.

Schemos

Schema **Pr_Iranga** aprašo universiteto programinę įrangą (žr. 4.1 pav.).



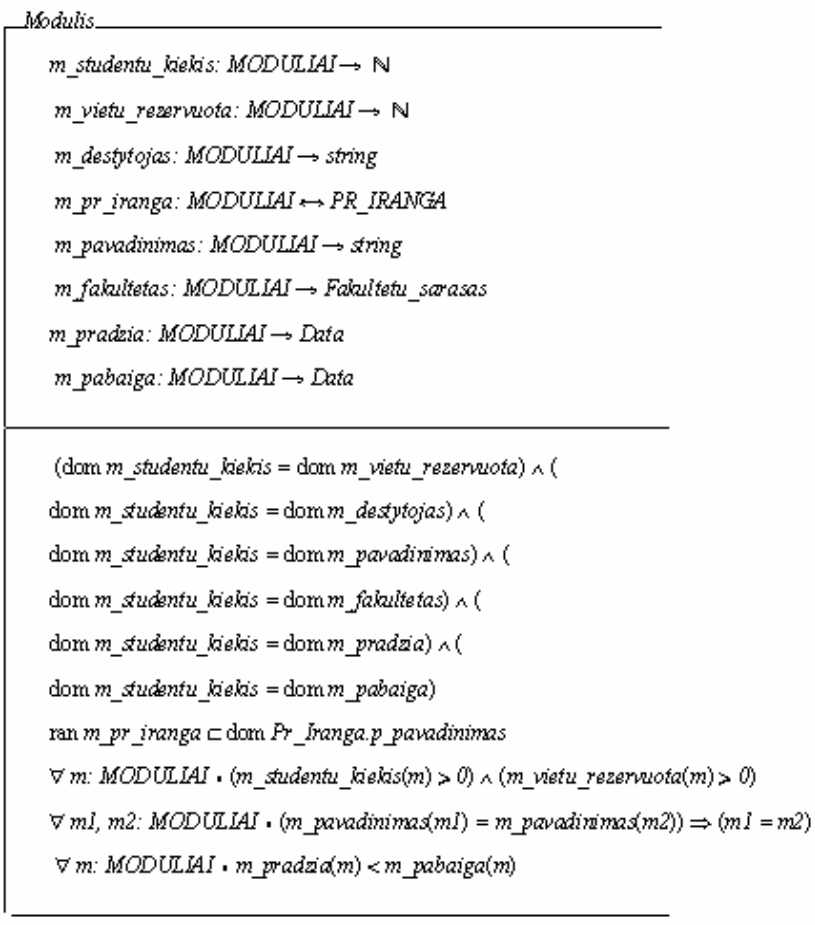
4.1 pav. Schema „Pr_Iranga“

p_pavadinimas- programinės įrangos pavadinimas, kuris yra simbolių (angl. *string*) tipo,

p_licenzijos- programinės įrangos licenzijos, kurios yra natūrinio skaičiaus tipo,

Predikato $A\, p: PR\_IRANGA \propto p\_licenzijos(p)\ 7 0$ sakiniu nurodoma, jog programinės įrangos licenzijų kiekis turi būti lygus arba daugiau už nulį.

Schema **Modulis** aprašo universitete dėstomus modulius (žr. 4.2 pav.).



4.2 pav. Schema „Modulis“

m_studentu_kiekis – modulį studijuojančių studentų skaičius, kuris yra natūrinio skaičiaus tipo,

m_dėstytojas – modulį vedančiojo dėstytojo vardas ir pavardė,

m_pr_iranga – modulio pravedimui reikalinga programinė įranga. Ji turi būti iš bendros programinės įrangos sąrašo,

m_pavadinimas- modulio pavadinimas, kuris yra simbolių tipo,

m_fakultetas – fakultetas, kuriam priklauso vedamas modulis. Fakultetas yra iš bendrojo fakultetų sąrašo,

m_pradzia – data nuo kurios pradedamas vesti, modulis,

m_pabaiga – data su kuria baigiasi modulio vedimas,

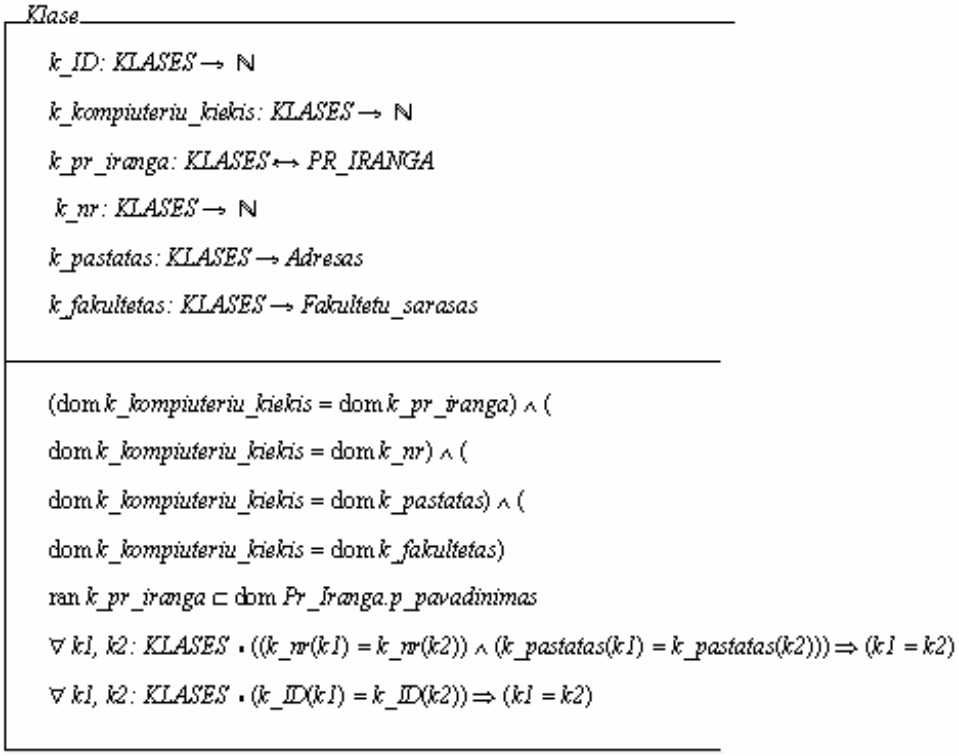
Predikato *ran m_pr_iranga* ζ dom *Pr_Iranga.p_pavadinimas* sakiniu, aprašoma sąlyga programinei įrangai. Modulio programinė įranga yra poaibis visos programinės įrangos.

Predikato $A\ m{:}\ MODULIAI \infty (m\_studentu\_kiekis(m) \rceil 0)\ f\ (m\_vietu\_rezervuota(m) \rceil 0)$ sakiniu nurodoma, jos modulį studijuojančių studentų skaičius yra lygus arba daugiau už nulį. Taip pat ir moduliui rezervuotų vietų skaičius kompiuterių klasėje (klasėse) negali būti mažesnis už nulį.

Sakinių $A\ m1,\ m2{:}\ MODULIAI \infty (m\_pavadinimas(m1) = m\_pavadinimas(m2)) \Rightarrow (m1 = m2)$ aprašoma sąlyga, jog jei bus rasti du vienodi mudulių pavadinimai ,tai jie bus laikomi kaip vienas ir tas pats modulis.

Paskutiniuoju schemos sakiniu $A\ m{:}\ MODULIAI \infty m\_pradzia(m) < m\_pabaiga(m)$ įvedama sąlyga, jog modulio vedimo pradžios data turi būti ankstesnė nei modulio vedimo pabaigos data.

Schema **Klasė** aprašo universiteto kompiuterių klases (žr. 4.3 pav.).



4.3 pav. Schema „ Klase“

k_id – kompiuterių klasės id numeris, kuris yra natūrinio skaičiaus tipo,

k_kompiuteriu_kiekis – klasėje esantis kompiuterių skaičius,

k_pr_iranga – kompiuterių klasėje įdiegta programinė įranga, kuri yra iš bendros programinės įrangos sąrašo,

k_nr – klasės numeris,

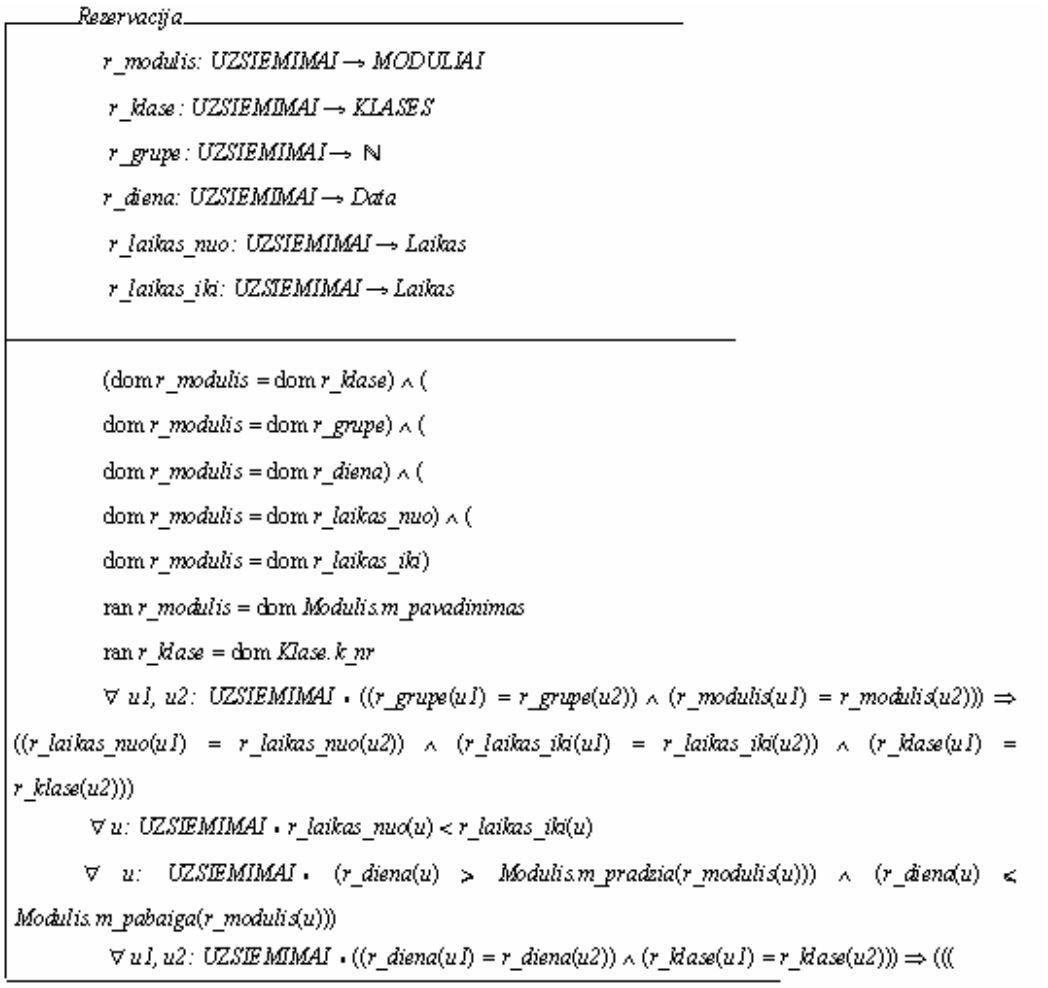
k_pastatas – pastato adresas, kuriame yra nurodyta kompiuterių klasė,

k_fakultetas – fakulteto pavadinimas, kuriame yra nurodyta kompiuterių klasė

Predikato *ran k_pr_iranga* ζ dom *Pr_Iranga.p_pavadinimas* sakiniu nurodoma jog, klasės programinė įranga yra poaibis bendros programinės įrangos

Paskutinieji du schemos sakiniai aprašo sąlygas klasės numeriui. Tiek klasės id numeris, tiek kompiuterių klasės kabineto numeris yra unikalūs. Dviejų tokių pačių numerių būti negali. Jei aptinkami du tokie patys (klasių kabinetų ar klasių id) numeriai, tai jie laikomi kaip viena ir ta pati klasė, skirtingos klasės tame pačiame pastate turi turėti skirtingus klasių numerius.

Schema **Rezervacija** aprašo kompiuterių klasės rezervavimą skirtą užsiėmimo pravedimui (žr. 4.4 pav.).



4.4 pav. Schema „ Rezervacija“

Norint rezervuoti kompiuterių klasę, reikia nurodyti užsiėmimo modulį, klasę, grupę, dienas, kuriomis pageidaujama vesti užsiėmimą bei paskaitos laiką.

r_modulis – modulis, kuris yra iš bendro modulių sąrašo,

r_klasė – klasė, kuri yra iš bendro kompiuterių klasių sąrašo,

r_grupė – užsiėmimų grupė,

r_diena – užsiėmimo pravedimui pageidaujama diena (datos),

r_laikas_nuo – užsiėmimo pravedimo laiko pradžia (paskaitos pradžia),

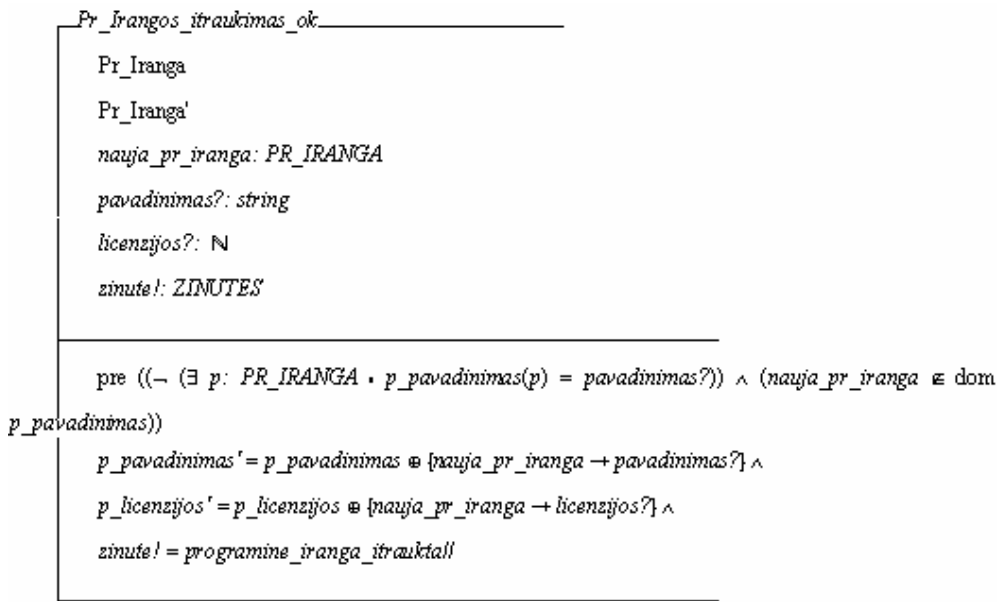
r_laikas_iki – užsiėmimo pravedimo laiko pabaiga (paskaitos pabaiga).

Predikato sakiniu *A u1, u2: UZSIEMIMAI* ∞ ((*r_grupe*(*u1*) = *r_grupe*(*u2*)) *ƒ* (*r_modulis*(*u1*) = *r_modulis*(*u2*))) įvedama sąlyga, jog 2 elementai yra tos pačios užsiėmimų grupės tik tada, kai tie elementai priklauso tam pačiam moduliui. Sakiniu ((*r_laikas_nuo*(*u1*) = *r_laikas_nuo*(*u2*)) *ƒ* (*r_laikas_iki*(*u1*) = *r_laikas_iki*(*u2*)) *ƒ* (*r_klase*(*u1*) = *r_klase*(*u2*))) nurodoma, jog tos pačios užsiėmimų grupės visų elementų modulio pravedimų laikai *nuo* ir *iki* yra tie patys. Taip pat yra ta pati klasė. Skiriasi tik užsiėmimų pravedimo datos.

Sakiniu *A u: UZSIEMIMAI* ∞ (*r_diena*(*u*) ⌉ *Modulis.m_pradzia*(*r_modulis*(*u*))) *f* (*r_diena*(*u*) ⌋ *Modulis.m_pabaiga*(*r_modulis*(*u*))) įvedama sąlyga, jog užsiėmimo rezervavimo datos turi tilpti į modulio pravedimo laiko intervalą ir negali išeiti už jo ribų.

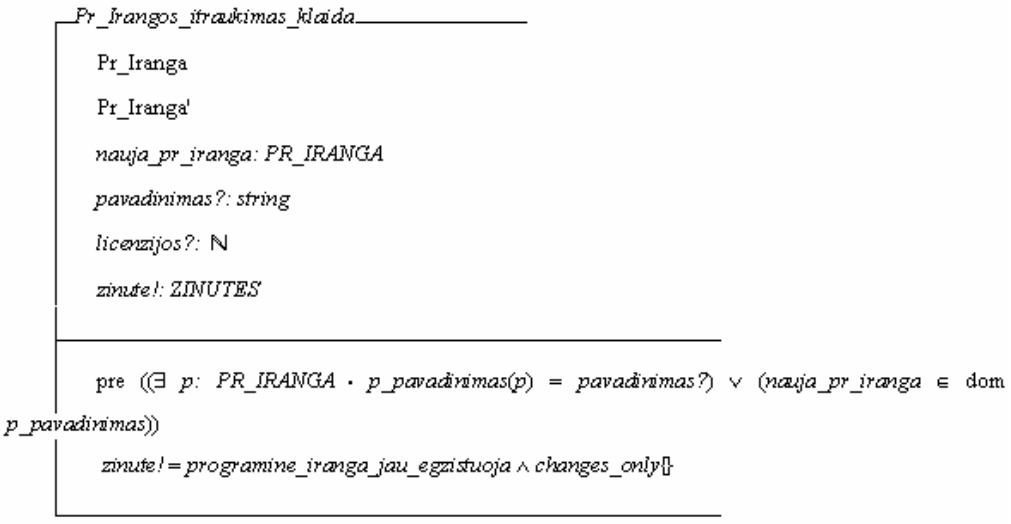
Operacijos

Operacija **Pr_Irangos_itraukimas_ok** aprašo naujos programinės įrangos įtraukimą (žr. 4.5 pav.)



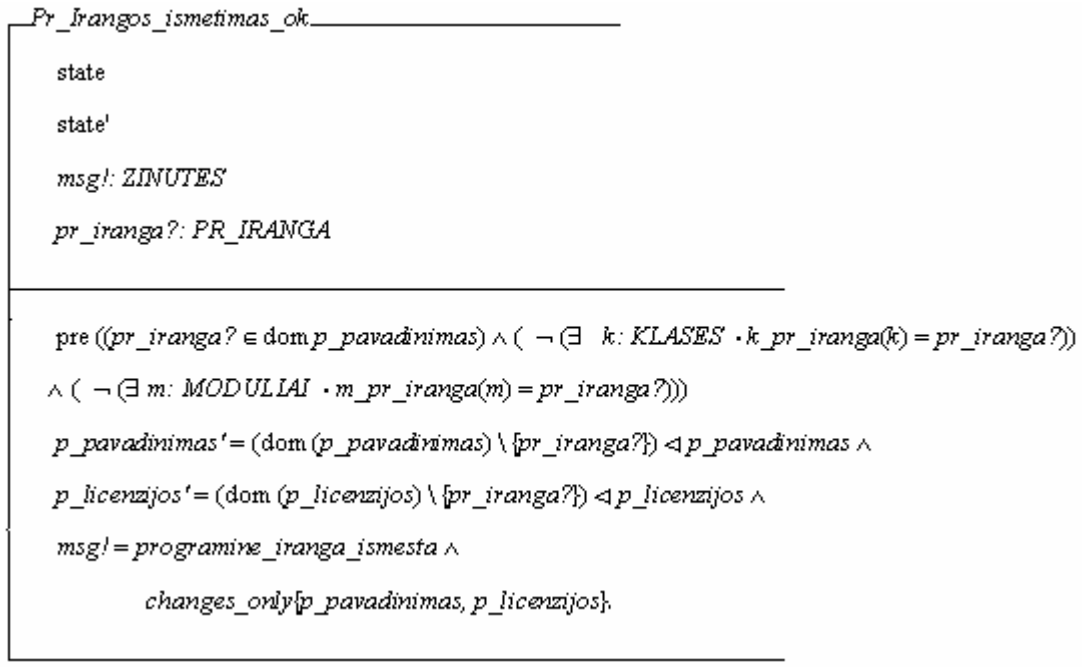
4.5 pav. Operacija „Pr_Irangos_itraukimas_Ok“

Predikate įvedama sąlyga, jog naujos programinės įrangos pavadinimas neturi egzistuoti bendrajame programinės įrangos sąraše *pre ((!* (E *p: PR_IRANGA* ∞ *p_pavadinimas*(*p*) = *pavadinimas?*)) *f* (*nauja_pr_iranga* ™ dom *p_pavadinimas*)). Jei toks pavadinimas jau egzistuoja, tai naujos programinės įrangos įtraukimas yra neįmanomas bei klaidingas, kaip yra aprašyta operacijoje **Pr_Irangos_itraukimas_klaida** (žr. 4.6 pav.).



4.6 pav. Operacija „Pr_Iranga_itraukimas_klaida“

Operacija **Pr_irangos _ismetimas_Ok** pašalina egzistuojančia programinę įrangą (žr. 4.7 pav.).



4.7 pav. Operacija „Pr_irangos _ismetimas_ok“

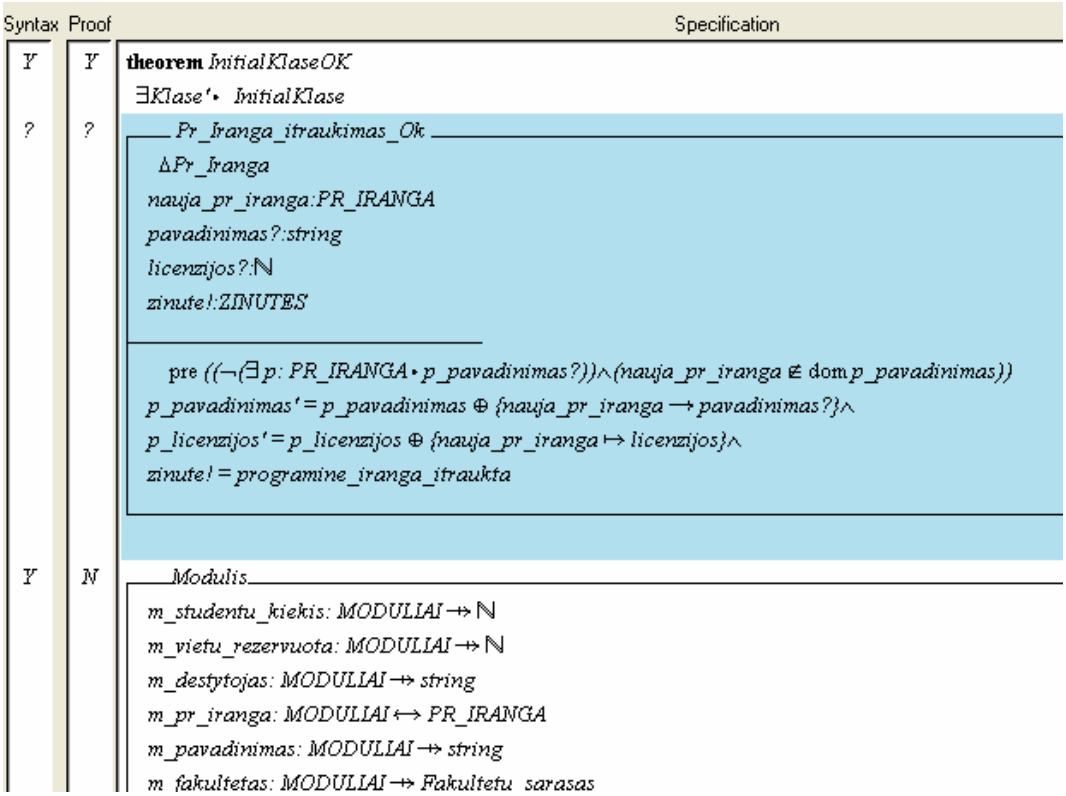
Predikato sakiniu *pre* ((*pr_iranga?* ε dom *p_pavadinimas*) patikrinama ar egzistuoja išmesti pageidaujama programinė įranga. Turi egzistuoti nurodytas programinės įrangos pavadinimas, priešingu atveju jos pašalinimas bus neįmanomas (Operacija **Pr_irangos_ismetimas_klaida**).

Šalinama programinė įranga negali būti įdiegta kompiuterių klasėje, taip pat negali būti naudojama kokio nors modulio užsiėmimuose (*k: KLASES* ∞ *k_pr_iranga*(*k*) = *pr_iranga?*)) *ƒ* (! (E *m: MODULIAI* ∞ *m_pr_iranga*(*m*) = *pr_iranga?*))). Šalinant programinę įranga taip pat yra šalinamos ir jos licenzijos.

Analogiškos operacijos yra atliekamos su moduliais, klasėmis ir užsiėmimais.

4.3 Z Specifikacijos validavimas

Z specifikacija buvo patikrinta panaudojant Z/EVES įrankį. Tipų apibrėžimai, schemos bei operacijos aprašytos kaip atskiri paragrafai, kurie patikrinti pagal tipus, sintaksiškai ir įrodant parašytas teoremas.



4.8 pav. Z/EVES specifikacijos fragmentas

Kaip matyti iš paveikslo (žr 4.8 pav.) **teorema** *InitialKlase_Ok* yra patikrinta sintaksiškai. Sintaksės tikrinimo statuso reikšme „Y“ reiškia, jog nei tipų, nei sintaksės klaidų rasta nebuvo. Teoremos paragrafas taip pat buvo patikrintas įrodymui. Kadangi gauta reikšmė taip pat yra „Y“, tai rodo jog įrodymas yra teisingas.

Sekantis paveikslėlio paragrafas yra **operacija** „Pr_Iranga_itraukimas_Ok“. Tiek sintaksės tikrinimo statuso, tiek įrodymo statuso reikšmė šiam paragrafui yra „?“. Tai reiškia, kad operacija nebuvo tikrinta nei sintaksiškai, nei įrodymui.

Schema „Modulis“ turi dvi skirtingas reikšmes: sintaksės statuso reikšmę „Y“ ir įrodymo statuso reikšmę „N“. Schema sintaksiškai yra teisinga. Paragrafas įrodymui buvo tikrintas, bet lieka neįrodytas.

Z/EVES teoremos yra formuluojamos tam, kad aprašyti specifikacijos savybes. Jos vaizduojamos aksiomos forma, kaip perrašytos prielaidinės arba kaip numanomos taisyklės. Dažniausiai teoremos rašomos patikrinti inicializavimo schemas, išankstines sąlygas, schemų ir operacijų invariantus, jų neprieštaringumą.

Specifikacijos neprieštaringumo patikrinimui rašomos inicializavimo schemų teoremos. Žemiau pateiktas „Klases“ inicializavimo schemos validavimo teorema, kur tikrinama ar egzistuoja pradinė būsena. Taip pat patikrina ar būsenos schemos invariantas yra neprieštaringas specifikacijai.

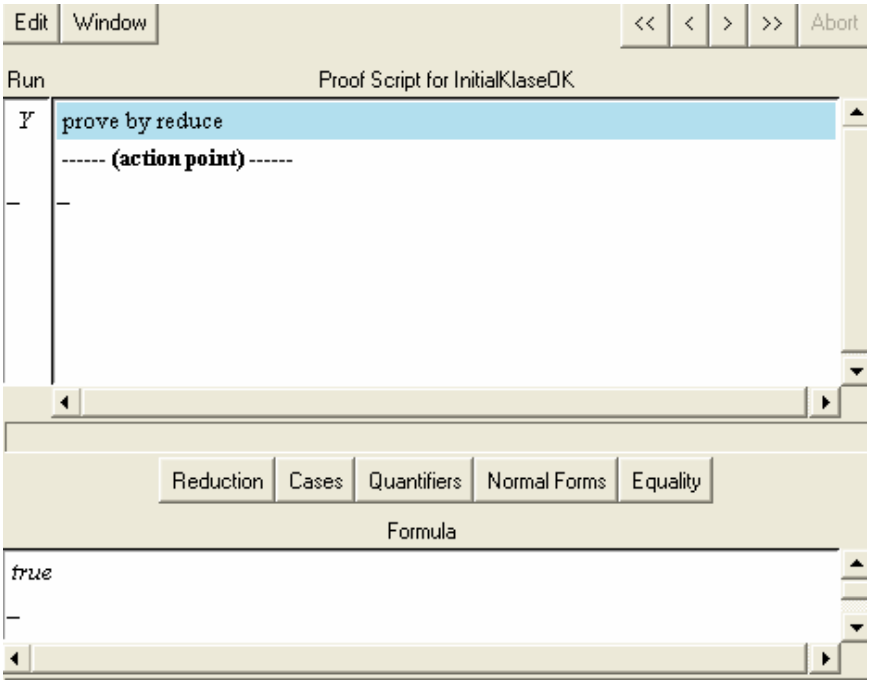
theorem InitialKlaseOK

E Klase' ∞ InitialKlase

Teoremos *InitialKlaseOK* įrodymas:

$$Klase[k\_ID := \{\}, k\_fakultetas := \{\}, k\_kompiuteriu\_kiekis := \{\}, k\_nr := \{\},$$
$$k\_pastatas := \{\}, k\_pr\_iranga := \{\}]$$
$$\wedge\ true$$

Teoremų lange yra iškviečiama komandą "prove by reduce". Jei įrodymas teisingas yra gražinama „true" reikšmė. Kaip matyti paveiksle teorema yra teisinga ir įrodyta (žr. 4.9 pav.).



4.9 pav. Z/EVES teoremų langas

Inicializavimo teorema analogiškai įrodoma ir kitoms schemoms. Žemiau pateikta „Pr_Iranga" inicializavimo schemos validavimo teorema:

theorem *InitialPr_Iranga*

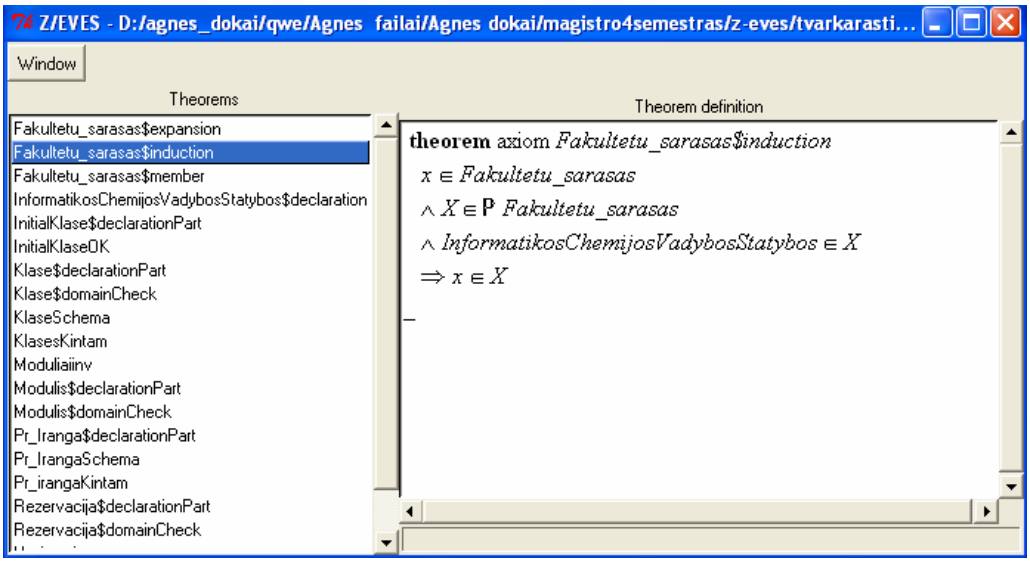
$$\exists Pr\_Iranga \bullet IntiPr\_iranga$$

Specifikacijos operacijoms patikrinti yra formuluojamos teoremos, kurios patikrina išankstines sąlygas bei invariantus. Pvz., *Rezervacija_ok* operacijos schemos invarianto teorema, kuri patikrina invariantą *laikas_nuo(u) < r_laikas_iki(u).*

theorem *Uzsiem_inv*

$$Rezervacija\_ok \wedge r\_laikas\_iki\ u = n \Rightarrow r\_laikas\_nuo\ u < n$$

Visos specifikacijos teoremos yra saugomos Z/EVES teoremų lange (žr. 4.10 pav.). Kairiajame lange matomi visi specifikacijos teoremų pavadinimai, dešiniajame pasirinktos teoremos apibrėžimas.



4.10 pav. Z/EVES teoremų langas

4.4 Z specifikacijos transformavimas į Object-Z specifikacija

Žemiau pateiktas (žr 4.11 pav.) Z kalba parašytos specifikacijos fragmentas , kuriame aprašyta schema „Klase“ ir operacijos „Klases_itraukimas_ok“, „Klases_itraukimas_klaida“, „Klases_ismetimas_ok“, „Klases_ismetimas_klaida“. Dešinėje pusėje tas pats specifikacijos fragmentas parašytas Object-Z kalba. Z schema ir operacija transformuojama į Object-Z klasę „Klase“. Z specifikacijos schemos vardas „Klasė“ atitinka Object-Z klasės vardą. Po pradinės būsenos tiek Z, tiek Object-Z specifikacijoje aprašomos operacijos. Object-Z Δ notacijoje nurodomi visi būsenos kintamieji, kurie yra keičiami, tuo tarpu Z specifikacijoje yra nurodomas visos schemos vardas „Klasė“.

Z specifikacija

$$Adresas == string$$

$$Fakultetu\_sarasas ::= Informatikos \mid Chemijos \mid Dizaino\_ir\_technologiju \mid Ekonomikos\_ir\_vadybos \mid Elektronikos\_ir\_automatikos \mid ir\_t\_t$$

Klase

$$k\_ID: KLASES \rightarrow \mathbb{N}$$
$$k\_kompiuteriu\_kiekis: KLASES \rightarrow \mathbb{N}$$
$$k\_pr\_iranga: KLASES \leftrightarrow PR\_IRANGA$$
$$k\_nr: KLASES \rightarrow \mathbb{N}$$
$$k\_pastatas: KLASES \rightarrow Adresas$$
$$k\_fakultetas: KLASES \rightarrow Fakultetu\_sarasas$$

$$(\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_pr\_iranga) \land (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_nr) \land (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_pastatas) \land (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_fakultetas)$$
$$\mathrm{ran}\ k\_pr\_iranga \subseteq \mathrm{dom}\ Pr\_Iranga.p\_pavadinimas$$
$$\forall k1, k2: KLASES \bullet ((k\_nr(k1) = k\_nr(k2)) \land (k\_pastatas(k1) = k\_pastatas(k2))) \Rightarrow (k1 = k2)$$
$$\forall k1, k2: KLASES \bullet (k\_ID(k1) = k\_ID(k2)) \Rightarrow (k1 = k2)$$

init

$$\mathrm{dom}\ p\_pavadinimas' = \{\}$$
$$\mathrm{dom}\ r\_modulis' = \{\}$$
$$\mathrm{dom}\ m\_pavadinimas' = \{\}$$
$$\mathrm{dom}\ k\_nr' = \{\}$$

Object-Z specifikacija

$$Fakultetu\_sarasas ::= Informatikos \mid Chemijos \mid Dizaino\_ir\_technologiju \mid Ekonomikos\_ir\_vadybos \mid Elektronikos\_ir\_automatikos \mid ir\_t\_t$$

Klase

(*init*, *Klases_itraukimas_ok*, *Klases_itraukimas_klaida*, *Klases_ismetimas_ok*, *Klases_ismetimas_klaida*)

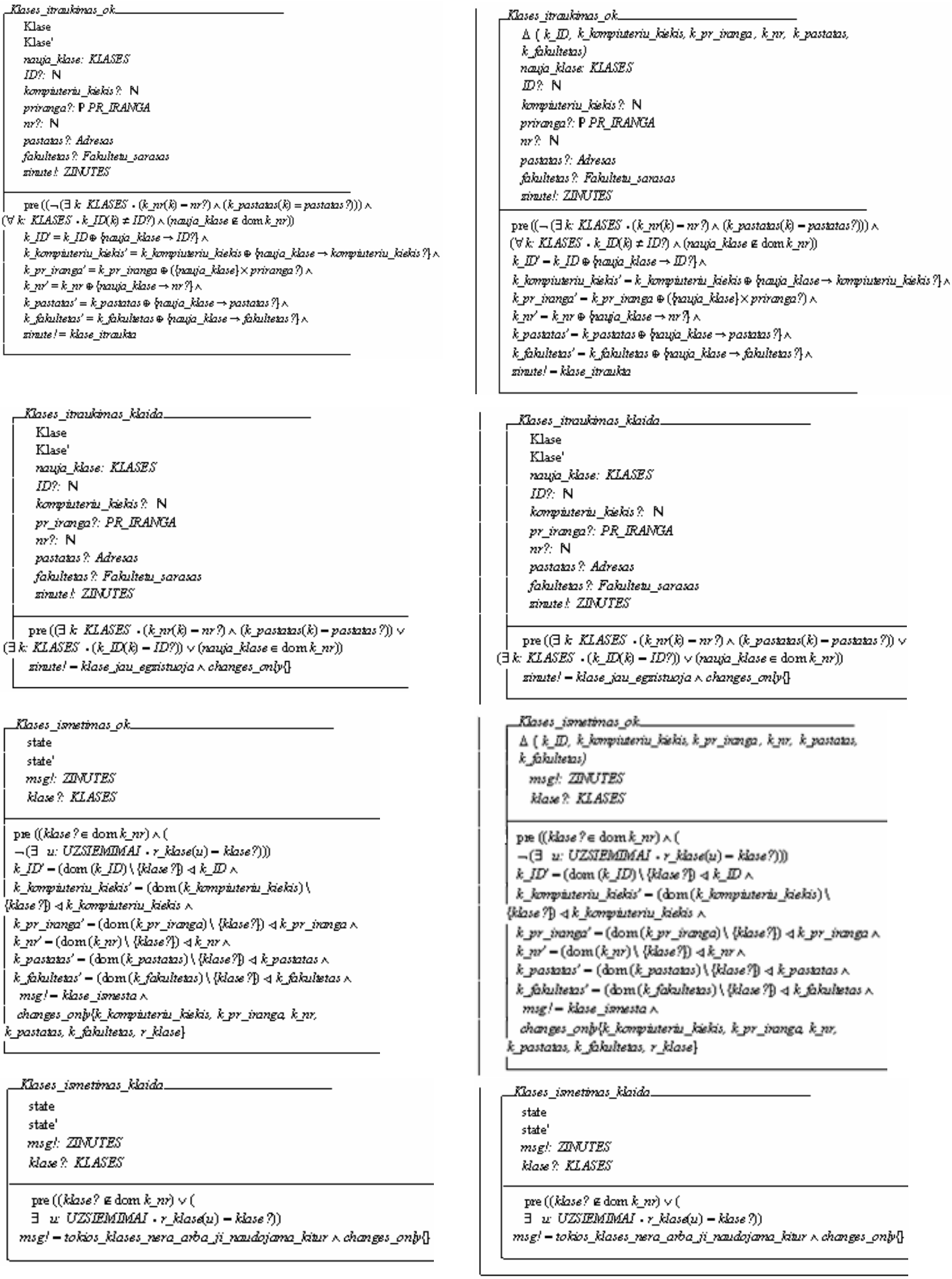
$$Adresas: string$$

$$k\_ID: KLASES \rightarrow \mathbb{N}$$
$$k\_kompiuteriu\_kiekis: KLASES \rightarrow \mathbb{N}$$
$$k\_pr\_iranga: KLASES \leftrightarrow PR\_IRANGA$$
$$k\_nr: KLASES \rightarrow \mathbb{N}$$
$$k\_pastatas: KLASES \rightarrow Adresas$$
$$k\_fakultetas: KLASES \rightarrow Fakultetu\_sarasas$$

$$(\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_pr\_iranga) \land (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_nr) \land (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_pastatas) \land (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_fakultetas)$$
$$\mathrm{ran}\ k\_pr\_iranga \subseteq \mathrm{dom}\ Pr\_Iranga.p\_pavadinimas$$
$$\forall k1, k2: KLASES \bullet ((k\_nr(k1) = k\_nr(k2)) \land (k\_pastatas(k1) = k\_pastatas(k2))) \Rightarrow (k1 = k2)$$
$$\forall k1, k2: KLASES \bullet (k\_ID(k1) = k\_ID(k2)) \Rightarrow (k1 = k2)$$

init

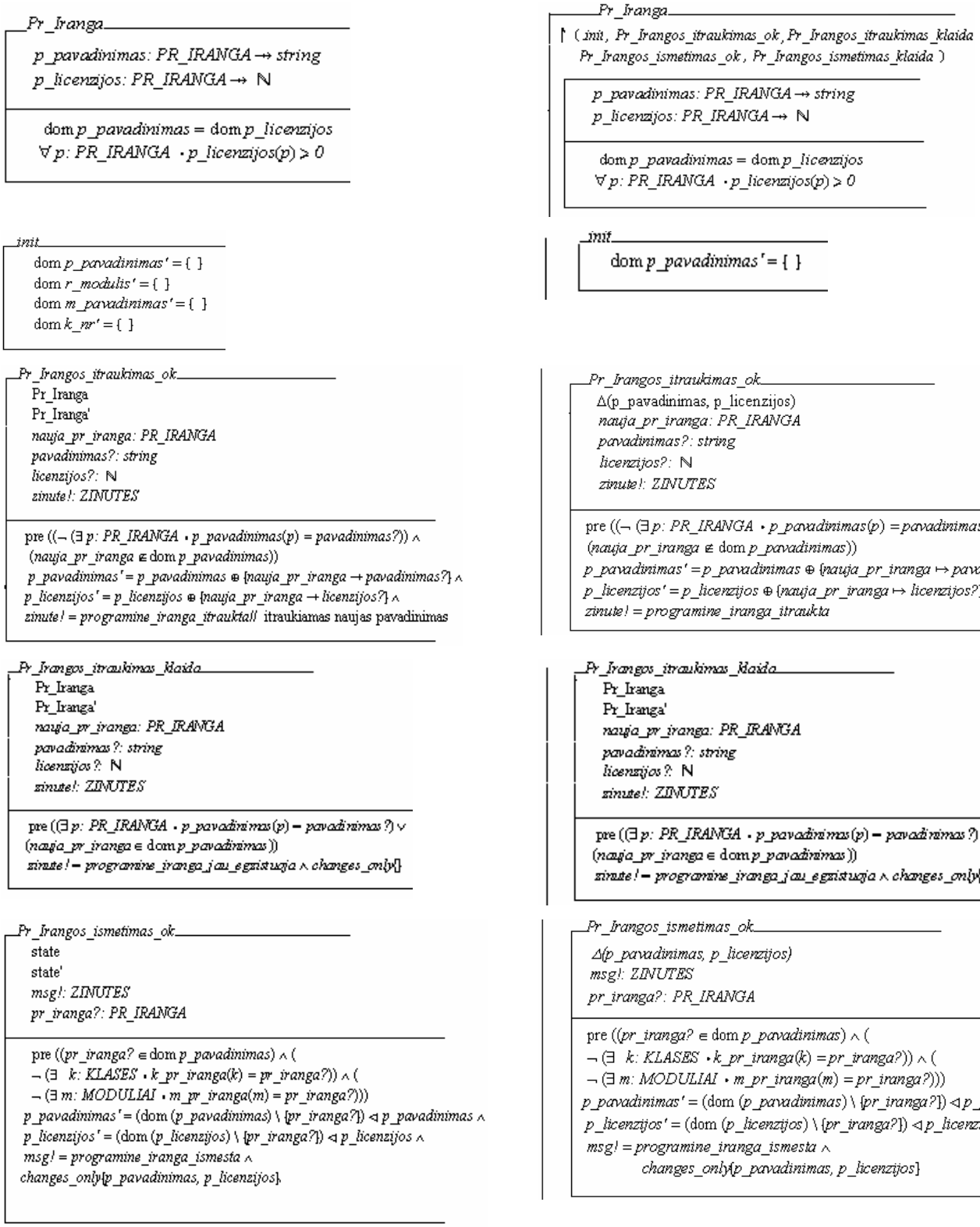
$$\mathrm{dom}\ k\_nr' = \{\}$$

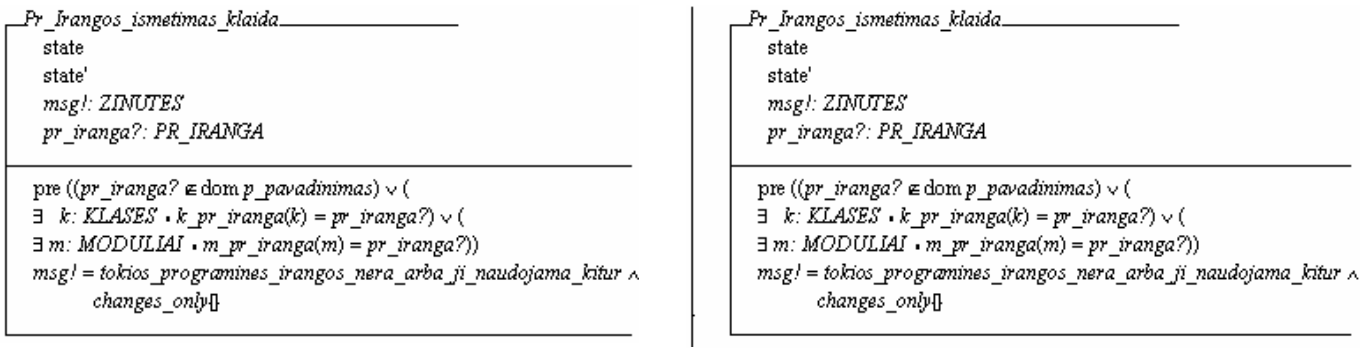


4.11 pav. Z specifikacijos transformavimo į Object-Z fragmentas

Žemiau pateiktame paveiksle (žr. 4.12 pav.) kairioje pusėje aprašoma Z schema „Pr_Iranga“ ir operacijos (programinės įrangos įtraukimas, klaidingas įtraukimas, pašalinimas ir klaidingas pašalinimas) susijusios su programine įranga. Dešinėje pusėje Z specifikacija transformuota į Object-Z kalbą.

Z specifikacija **Object-Z specifikacija**





4.12 pav. Z specifikacijos transformavimo į Object-Z fragmentas

Analogiškai transformuojamos visos kitos Z specifikacijos schemos ir operacijos į Object-Z klases.

4.5 Specifikacijos transformavimas į programinės įrangos kodą.

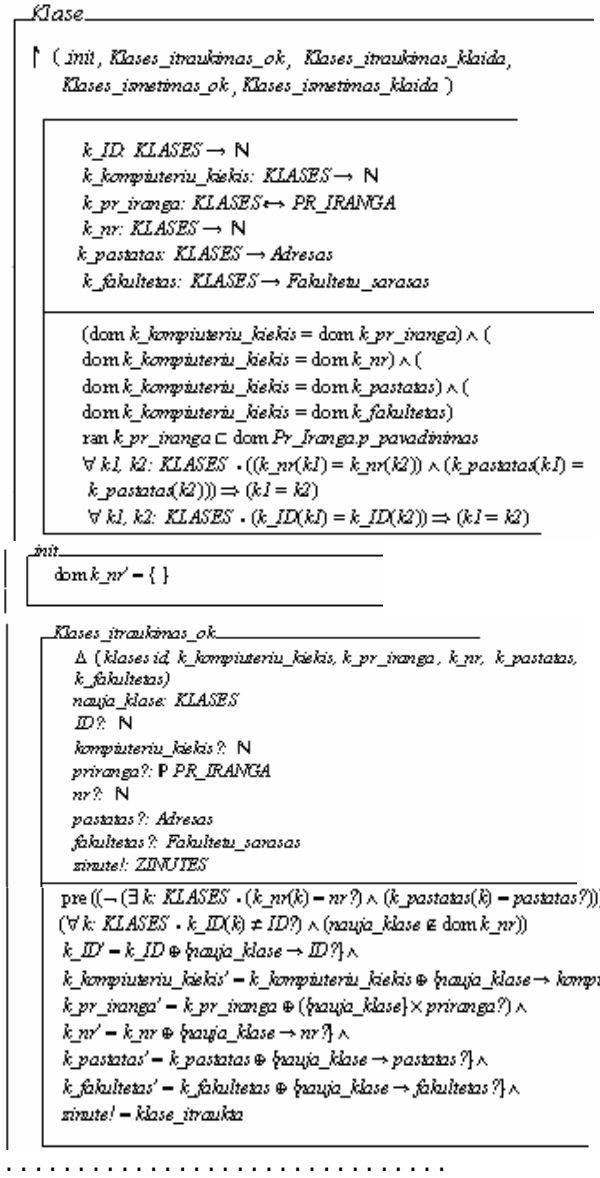
Šiame darbe yra siūloma Object-Z specifikacijos transformavimo į programinės įrangos kodą metodika. Kompiuterių klasių užimtumo informacinės sistemos duomenų struktūros aprašytos SQL DDL sakiniais, valdymo logika aprašyta C++ programavimo kalba. Todėl Object-Z specifikacijos transformavimas į informacinės sistemos programinės įrangos kodą atliekamas dviem etapais:

- Specifikacijos statinės dalies transformavimas į SQL DDL sakinius;
- Specifikacijos statinės ir dinaminės dalies transformavimas į C++.

Object-Z specifikacija transformuojama į C++ programavimo kalbą remiantis pagrindinėmis Object-Z specifikacijos transformavimo į C++ taisyklėmis (žr. 3.1 lentelę). Žemiau (žr. 4.13 pav.) pateiktas Object-Z schemos „Klase“, pradinės būsenos „init“ bei operacijos „Klases _itraukimas_ok“ transformavimas į C++ programavimo kalbą. Atributai (konstantos ir kintamieji) aprašyti klasės schemoje yra transformuojami į C++ klasės atributus, išlaikant tą patį konstantų ir kintamųjų savitumą, pvz., *k_ID: N* ir private *int * k_ID*. Kintamasis k_ID tiek object-Z, tiek C++ yra natūrinio skaičiaus tipo, tik natūrinis skaičius Object-Z žymimas kaip *N*, o C++ kaip *int*. Object-Z būsenos schemos aksiomos yra transformuojamos į C++ klasės invariantus. Paskutiniai Object-Z klasės schemos predikato sakiniai $\forall k1, k2: KLASES \bullet ((k\_nr(k1) = k\_nr(k2)) \; f \; (k\_pastatas(k1) = k\_pastatas(k2))) \Rightarrow (k1 = k2)$ ir $\forall k1, k2: KLASES \bullet (k\_ID(k1) = k\_ID(k2)) \Rightarrow (k1 = k2)$ transformuojami į C++ invariantą *unique (k_nr, k_pastatas)*. Object-Z inicializavimo schema *init* transformuojama į C++ *public Klase*, kurioje visiems atributams priskiriama nulinė reikšmė. Object-Z „init“ schemoje pakanka nurodyti tik vieną kintamąjį. Tuo tarpu C++ reikia išvardinti visus kintamuosius. Operacijos schema Klases_itraukimas_ok transformuojamos į C++ tokiu būdu: Δ-sąrašas transformuojamas į „modifikacijų“ konstruktą, būsenos – „iki“ aksiomos atitinka būsenas – „iki“, ir likusios operacijos schemos aksiomos transformuojamos į C++ būsenas – „po“. Object-Z Δ sąraše nurodomi atributai, kurie yra keičiami. Šie atributai C++ nurodomi kaip modifikacijos. Object-

Z būsena – „iki“ pre ((! (E *k: KLASES* ∞ (*k_nr*(*k*) = *nr?*) *ƒ* (*k_pastatas*(*k*) = *pastatas?*))) *ƒ* (A *k: KLASES* ∞ *k_ID*(*k*) ⌊ *ID?*) *ƒ* (*nauja_klase* ™ dom *k_nr*)) transformuojama į C++ būseną – „iki“ *unique (k_nr, k_pastatas); unique (k_ID).* Operacijos būsenos – „po“ transformuojamos į C++ būsenas – „po“, kuriose įtraukiama nauja klasė, įtraukiant visus tos klasės atributus: *k_ID[×××××] =ID; k_kompiuteriu_kiekis[×××××] = k_kompiuteriu_kiekis; k_pr_iranga[×××××] =k_pr_iranga; k_nr [×××××] = nr; k_pastatas[×××××] =k_pastatas; k_fakultetas[×××××] =k_fakultetas*

Object-Z



. . . . . .

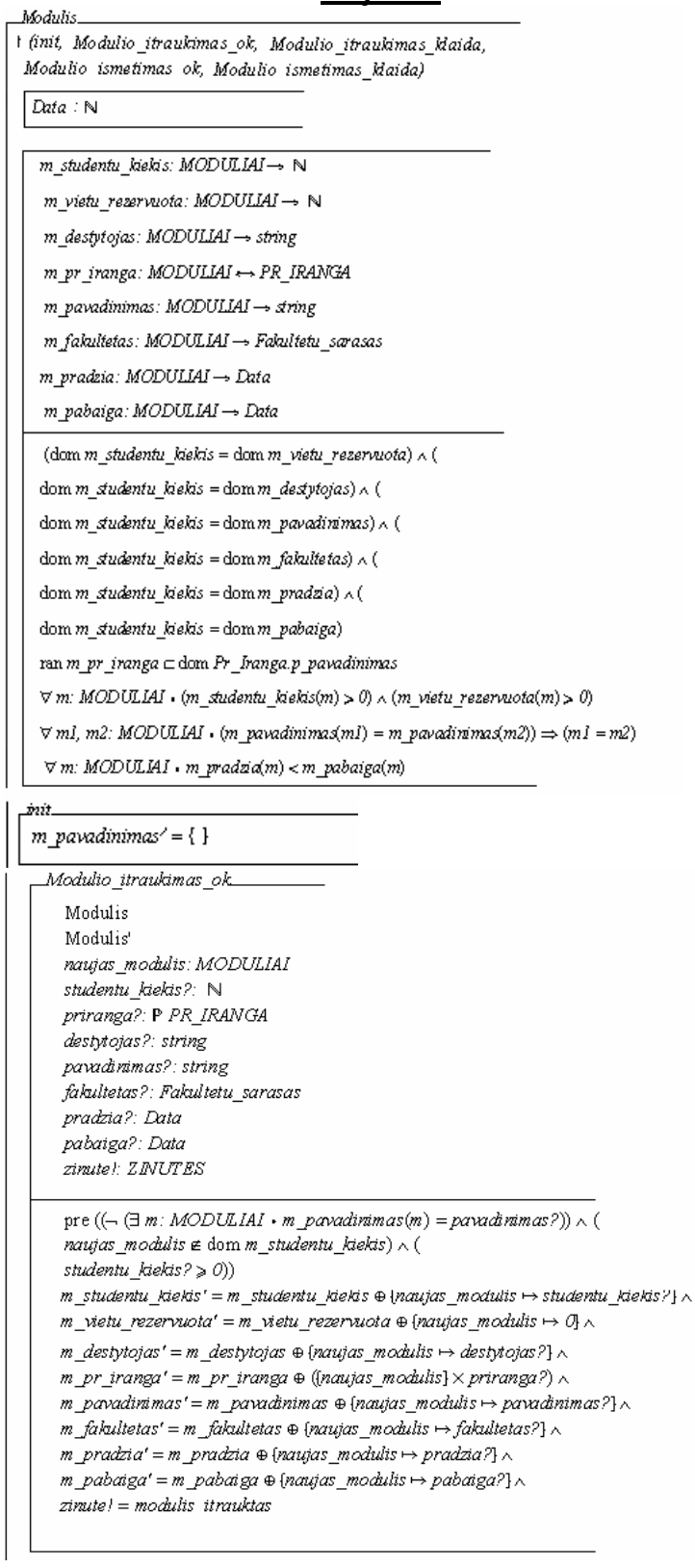
C++ po transformacijos

```
public class Klase{
  private int    k_ID;
  private int
k_kompiuteriu_kiekis;
  private Pr_iranga *
k_pr_iranga[40];
  private int    k_nr;
  private char   k_pastatas[50];
  private char   k_fakultetas[50];

//invariant:        unique    (k_nr,
k_pastatas)

public Klase( )
{ k_nr =null;   k_ID=null;
  k_kompiuteriu_kiekis=null;
  k_pr_iranga=null;
  k_pastatas=null;
  k_fakultetas=null}

void Klases_itraukimas_ok

// modifies:
   (k_ID, k_kompiuteriu_kiekis,
    k_pr_iranga, k_nr, k_pastatas,
    k_fakultetas);

  //pre:
 unique (k_nr, k_pastatas);
 unique (k_ID);

  //post:
k_ID[×××××] =k_ID;
k_kompiuteriu_kiekis[×××××]=
            k_kompiuteriu_kiekis;
k_pr_iranga[×××××] =k_pr_iranga;
k_nr[×××××] = k_nr;
k_pastatas[×××××] =k_pastatas;
k_fakultetas[×××××] =k_fakultetas;
  public void klases_itraukimas_ok()
  {.....}
}
. . . . . .
```

4.13 pav. Object-Z specifikacijos transformavimo į C++ programavimo kalbą fragmentas (1)

Žemiau (žr. 4.14 pav.) pateiktas Object-Z schemos „Modulis“, pradinės būsenos „init“ bei operacijos „Modulio _itraukimas_ok“ transformavimas į C++ programavimo kalbą.

Object-Z



. . . . . .

C++ po transformacijos

```
public class Modulis{
  private int    m_studentu_kiekis;
  private int    m_vietu_rezervuota;
  private char   m_destytojas [40];
  private Pr_iranga * m_pr_iranga [40];
  private char   m_pavadinimas;
  private char   m_fakultetas [50];
  private int    m_pradzia;
  private int    m_pabaiga ;

  //invariant:   m_studentu_kiekis>=0;
           m_vietu_rezervuota>=0;
           unique (m_pavadiniams);
           m_pradzia<m_pabaiga

public Modulis( )
{m_studentu_kiekis=null;
 m_vietu_rezervuota=null;
 m_destytojas=null; m_pr_iranga=null;
 m_pavadinimas=null; m_fakultetas=null;
 m_pradzia =null; m_pabaiga=null }

void Modulio_itraukimas_ok

 //modifies:(m_studentu_kiekis,
m_vietu_rezervuota,        m_destytojas,
m_pr_iranga,              m_pavadinimas,
m_fakultetas, m_pradzia, m_pabaiga);



//pre:   unique (m_pavadinimas);
           M_udentu_kiekis>=0

  //post:    m_studentu_kiekis[×××××]=
             m_studentu_kiekis;

m_vietu_rezevuota[×××××]=
            m_vietu_rezervuota
m_destytojas [×××××] = m_destytojas;
m_pr_iranga  [×××××] = m_pr_iranga;
m_pavadinimas[×××××] = m_pavadinimas;
m_fakultetas [×××××] = m_fakultetas;
m_pradzia    [×××××] = m_pradzia;
m_pabaiga    [×××××] = m_pabaiga.

  public void Modulio_itraukimas_ok()
  {.....}
}
.............................
```

4.14 pav. Object-Z specifikacijos transformavimo į C++ programavimo kalbą fragmentas (2)

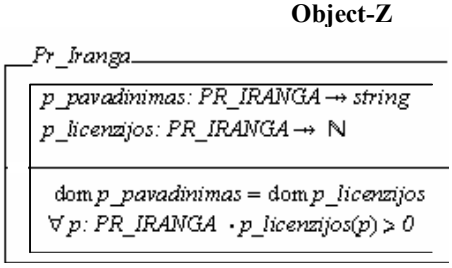
Transformuotas C++ kodas yra naudingas realizacijos startinis taškas, padedantis programuotojui daug lengviau, greičiau ir tiksliau parašyti programinės įrangos kodą (žr. 4.15 pav).

4.15 pav. C ++ po transformacijos ir po realizacijos

Visi Object-Z klasės būsenos kintamieji ir predikatai transformuojami į SQL DDL sakinius remiantis, pagrindinėmis Object-Z specifikacijos transformavimo į SQL DDL sakinius taisyklėmis (žr. 3.2 lentelę).

Žemiau iliustruojamas Object-Z klasės "Pr_iranga" schemos transformavimas į SQL DDL (žr. 4.16 pav.). Object-Z klasės pavadinimas „Pr_Iranga" atitinka SQL lentelės pavadinimą. Object-Z kintamieji *p_pavadinimas, p_licenzijos* yra SQL lentelės stulpeliai. Object-Z atributas *p_pavadinimas* yra *string* tipo, SQL p_pavadinimas aprašomas kaip *varchar* tipo. Object-Z atributas p_licenzijos yra natūrinio skaičiaus tipo, SQL jis aprašomas kaip *integer* tipo kintamasis. Object – Z atributas *p_pavadinimas* yra *string* tipo ir jis atitinka lentelės stulpelį p_pavadinimas, kuris turi VARCHAR[45] tipą. SQL sakiniuose frazė NOT NULL pažymima, kad stulpelyje negali būti NULL reikšmės. Apibrėžiant lentelę, papildomai galima nurodyti pirminį raktą bei išorinius raktus.

Object-Z

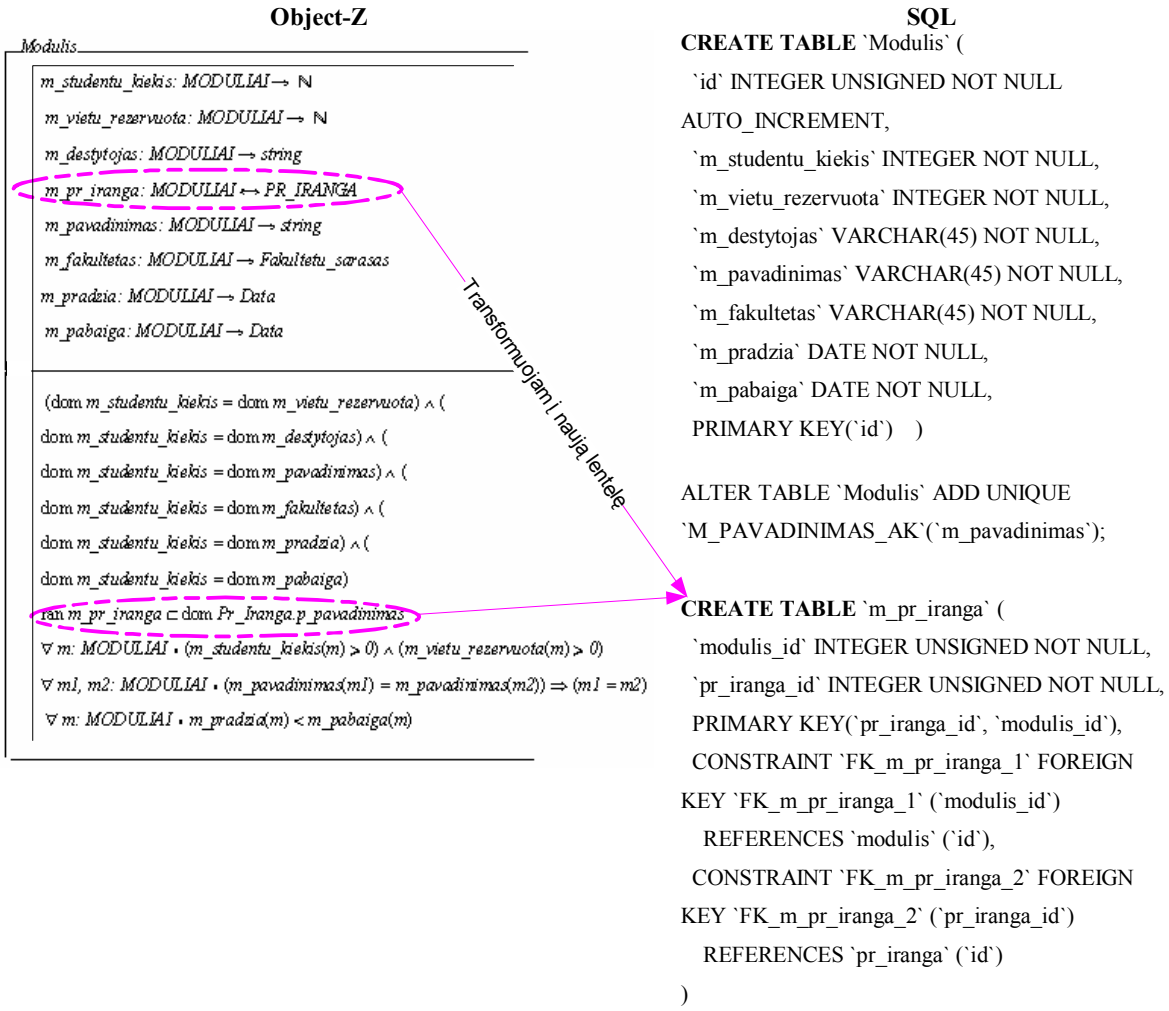


SQL

```
CREATE TABLE `Pr_Iranga` (
 `id` INTEGER UNSIGNED NOT NULL
AUTO_INCREMENT,
 `p_pavadinimas` VARCHAR(45) NOT NULL,
 `p_licencijos` INTEGER UNSIGNED NOT NULL,
 PRIMARY KEY(`id`)  )
```

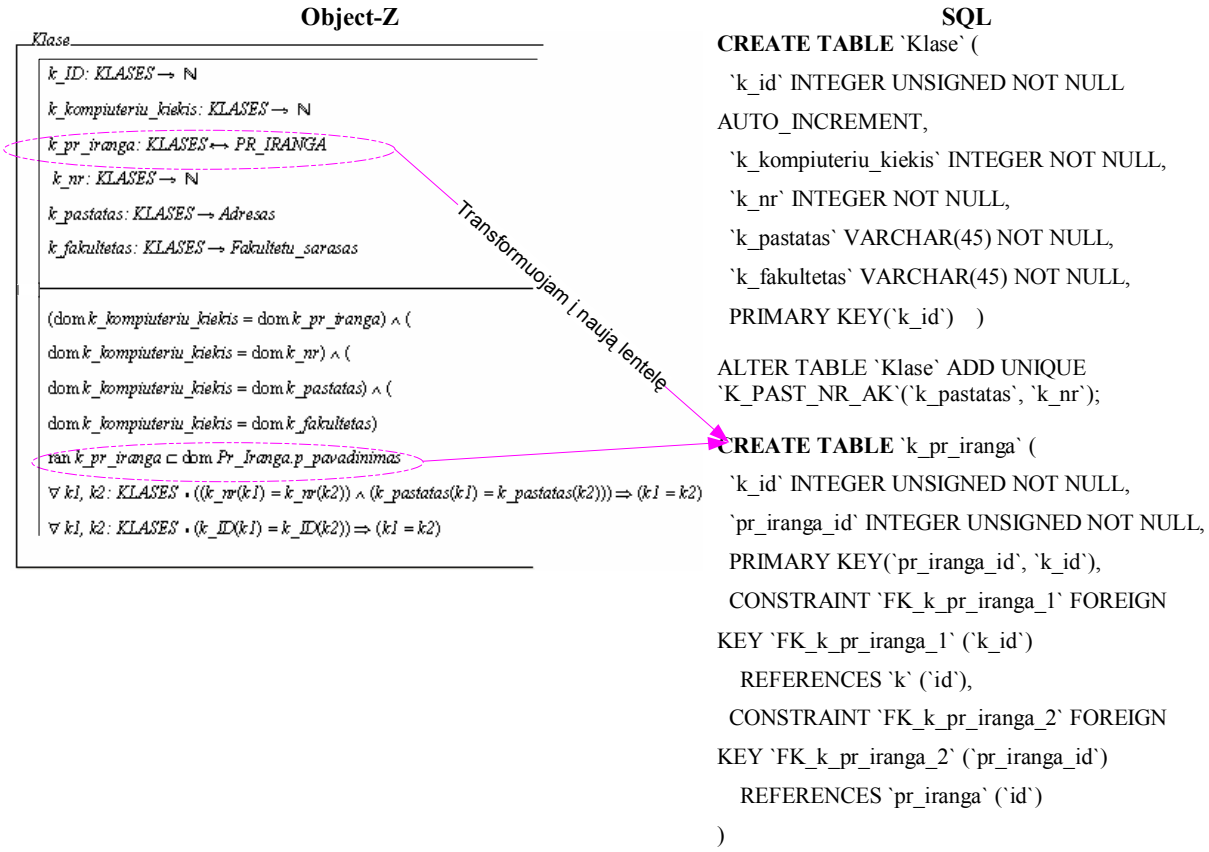
4.16. pav. "Pr_iranga" schemos transformavimas į SQL DDL

Žemiau (žr. 4.17 pav.) pateiktas Object-Z klasės "Modulis" fragmentas. Schemos būsenos kintamieji transformuojami į lentelės „Modulis" stulpelius, išskyrus kintamąjį *m_pr_iranga*. Jis kartu su predikato sakiniu *ran m_pr_iranga⊆Pr-Iranga.p_pavadinimas* transformuojamas į naują SQL lentelę **m_pr_iranga**. Su sakiniu ALTER TABLE apibrėžiamas schemos invariantas, kad modulio pavadinimas yra unikalus.



4.17 pav. Object-Z klasės "Modulis" fragmento transformacija į SQL DDL sakinius

Analogiškai yra atliekama klasės "Klase" schemos transformacija į SQL DDL aprašą (žr 4.18 pav.). Būsenos kintamieji transformuojami į lentelės „Klase" stulpelius. Object-Z schemos būsenos kintamasis *k_pr_iranga* kartu su predikato sakiniu *ran k_pr_iranga⨯Pr_Iranga.p_pavadinimas* transformuojami į naują SQL lentelę **k_pr_iranga**.



4.18 pav. Object-Z klasės "Klase" fragmento transformacija į SQL DDL sakinius

4.6 Informacinės sistemos realizacija

Aprašant kompiuterinių klasių užimtumo modelį buvo panaudoti Z formalūs metodai. Formalieji metodai leido aprašyti klases, modulius, programinę įrangą, operacijas, įvertinti galimybę pravesti užsiėmimus konkrečioje klasėje, konkrečiu laiku, suformuoti užklausas apie kompiuterių klasių užimtumą; modulius dėstomus konkrečiose kompiuterių klasėse ir programinę įrangą naudojamą mokymo procese.

Kompiuterių klasių užimtumo informacinė sistema realizuoja funkcijas, atitinkančias Z specifikacijos operacijas:

- įtraukia naują programinę įrangą;
- pašalina egzistuojančią programinę įrangą;
- įtraukia naują kompiuterių klasę;
- pašalina egzistuojančią kompiuterių klasę;
- įtraukia naują modulį;
- pašalina egzistuojantį modulį
- rezervuoja pageidaujamą kompiuterių klasę (klases) pageidautam laikotarpiui, pasirinktu laiku;

- pašalina rezervaciją

Formalūs specifikavimo metodai leido apibrėžti reikalavimus kompiuterių klasių užimtumo IS programinei realizacijai. Šie reikalavimai aprašyti Z schemomis, kurios aprašo klases, modulius, programinę įrangą, užsiėmimų rezervavimą.

Sukurtoje informacinėje sistemoje naudojama 3 lygių architektūra: duomenų bazė,. serveris, klientas. Duomenų bazė sukurta naudojant MySQL 4.1. Kliento lygmuo buvo aprašytas panaudojant C# programavimo kalbą, serveris aprašytas panaudojant C++ programavimo kalbą.

Sukurta kompiuterinių klasių užimtumo informacinė sistema leidžia spręsti šiuos uždavinius: interaktyviai sudaryti užsiėmimų kompiuterių klasėse tvarkaraštį, t.y. centralizuoti mokymo proceso organizavimą kompiuterių klasėse, sukaupti informaciją apie dėstomus kompiuterių klasėse modulius; naudojamą programinę įrangą studijų procese; analizuoti kompiuterių klasių užimtumą. Informacinė sistema leido atlikti kompiuterinių klasių užimtumo eksperimentinius tyrimus (žr. 2, 3 prieduose).

Informacinės sistemos programinę realizaciją pagal pateiktus reikalavimus atliko KTU informatikos fakulteto III kurso studentai: V.Šeinauskas ir N.Ambrazas

5 MAGISTRINIO DARBO BAIGIAMOSIOS IŠVADOS

1. Darbe parodyta, kaip nuosekliai detalizuojant pradinius formaliuosius reikalavimus aprašytus Z specifikavimo metodu yra pereinama prie informacinės sistemos realizacijos.
2. Sudarant projektuojamos sistemos reikalavimų formaliąją specifikaciją rekomenduotina atlikti validavimą naudojant interaktyviąją Z/EVES validavimo sistemą, kuri leidžia suformuoti teoremas, tikrinančias specifikacijos korektiškumą bei neprieštaringumą.
3. Nustatyta, kad norint atlikti formaliosios specifikacijos transformavimą į objektines programavimo kalbas yra tikslinga naudoti objektinę Object – Z specifikavimo kalbą, kuri praplečia Z kalbą, įvesdama klasės ir objekto sąvokas.
4. Darbe pasiūlyta Z specifikacijos transformavimo į Object-Z metodika. Parodyta, kad informacijos, kuri yra įprastoje Z specifikacijoje pakanka Object-Z specifikacijos sudarymui, interpretuojant Z schemas kaip objektus.
5. Pagal sukurtą metodiką Z specifikaciją transformavus į Object-Z, buvo nustatyta, kad susidaro galimybės sistemos objektinį aprašą transformuoti į C++ objektinę programavimo kalbą ir SQL DDL aprašą. Tam tikslui sukurtos Object-Z specifikacijos transformavimo į C++ programos kodą ir SQL DDL aprašą metodikos.
6. Sukurtos Kauno technologijos universiteto klasių užimtumo informacinės sistemos duomenų bazė užpildyta realiais duomenimis, kas leido gauti statistinių duomenų apie: kompiuterių klasių užimtumą, programinės įrangos panaudojimą mokymo procese, dėstomus modulius kompiuterių klasėse.

Šio darbo tema parašyti trys straipsniai bei sudalyvauta trijose mokslinėse konferencijose. Straipsnių kopijos pridedamos prieduose (žr. 4 priedą).

6 LITERATŪRA

[1]. ALMENDROS, J.M., GONZALEZ, L. *The LAST project: development of a formal method for IS-specification and of a CASE-tool for IS-design* [interaktyvus]. Seventha-pacific Software Engineering Conference, Singapore, December 2000 [žiūrėta 2003 m. lapkričio 5 d.]. Prieiga per internetą : <http://csdl.computer.org/comp/proceedings/apsec/2000/0915/00/09150054abs.pdf>

[2]. KOWSARI, M., GRAU, A. *An Evaluation of an Object Oriented Formal Method for Specifying Information Systems* [interaktyvus]. In Proceedings of the third CaiSE/IFIP 8.1 International Workshop on Evalution of Modeling Methods in Systems Analys and Design (EMMSA'98). Pisa, Italy. June 8-9, 1998, University of Nebraska - Lincoln, USA, 1998 [žiūrėta 2003 m. lapkričio 5 d.]. Prieiga per internetą: <http://homepages.inf.ed.ac.uk/agrau/Publications/emmsad98.pdf>

[3]. BLOW, J.R., GALLOWAY, A.J., MCDERMID, J.A. *The Industrial Use of Formal method in a Gas Turbine Engine Electronic Control System* [interaktyvus]. High Integrity Systems Engineering, Department of Computer Science , UK, 2000 [žiūrėta 2003 m. lapkričio 5 d.]. Prieiga per internetą: <http://www.branchlines.org.uk/Research/PFS.PDF>

[4]. FRAPPIER, M., HABRIAS, H. *A Comparison of the Specification Methods* [interaktyvus]. Universit´e de Sherbrooke, D´epartement de math´ematiques et d'informatique, Sherbrooke, Qu´ebec, Canada, Institut de Recherches en Informatique de Nantes, 2001 [žiūrėta 2003 m. spalio 25 d.]. Prieiga per internetą :< www.dmi.usherb.ca/~spec/A3-**comparison**-v4.pdf >.

[5]. FINKELSTEIN, L., HUANG, J., FINKELSTEIN, A.C.W., NUSEIBEH. B.**.** *Using Software Specification Methods for Measurement Instrument System* [interaktyvus]. Measurement Journal, Vol 10, No 2, Apr-Jun 1992, p. 87-92 [žiūrėta 2003m. spalio 25d.]. Prieiga per internetą <http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/measurement1.pdf >

[6]. BOURAHLA, M., BENMOHAMED, M**.** *Formal Specification and Verification of Multi-Agent Systems* [interaktyvus]. Electronic Notes in Theoretical Computer Science, Volume 123, 1 March 2005, p. 5-17 [žiūrėta 2003m. spalio 25d.]. Prieiga per internetą: <http://search2.computer.org/advanced/Advanced_Result.jsp >

[7]. CHARLIER, B. L., FLENER, P**.** *Specifications are Necessarily Informal or: Some more myths of formal methods* [interaktyvus]. Journal of Systems and Software, Volume 40, Issue 3, March 1998, 275-296p. [žiūrėta 2003m. spalio 25d.]. Prieiga per internetą: <www.cs.york.ac.uk/ftpdir/reports/YCS-2000-327.ps. >

[8]. FUHRMAN, C.P. *Lightweight Models for Interpreting Informal Specifications* [interaktyvus]. Springer-Verlag London, Volume 8, Number 4, November 2003, 206–221p. [žiūrėta 2003m. lapkričio 17 d.]. Prieiga per internetą: <www.springerlink.com/index/9WV21AKG33J9JWJ6.pdf>

[9]. Case tool information. Iš Queen‘s University, school of computing [interaktyvus], 2004. [žiūrėta 2003m. lapkričio 18 d.]. Prieiga per internetą: < http://www.qucis.queensu.ca/Software-Engineering/case.html>

[10]. Introduction to Oracle Case. Iš Center for Infomation Managment, Integration and Connectivity [interaktyvus], 2004. [žiūrėta 2005m. vasario 4 d.]. Prieiga per internetą: <http://cimic.rutgers.edu/~holowcza/present/oracase/sld004.htm>

[11]. Running UML case tools. Iš Megaspace [interaktyvus]. 2001 [žiūrėta 2005m. vasario 4 d.]. Prieiga per internetą: < http://www.megspace.com/computers/chansinsha/ >

[12]. BUTLERIS, R., DANIKAUSKAS. T. *Reikalavimo specifikavimo Oracle Case trepėje plėtra* [interaktyvus]. Kaunas University of Technology Information Systems Department, 2001. [žiūrėta 2004 m. gruodžio 16 d.]. Prieiga per internetą: http://www.leidykla.vu.lt/inetleid/inf-mok/19/str6.html

[13]. DYKMAN, N. Sage*: Generating applications with UML and components* [interaktyvus]. Department of Cumputer Science, The University of Utah , August 1999. [žiūrėta 2005m. sausio 10 d.]. Prieiga per internetą: < http://www.cs.utah.edu/~ndykman/Draft.pdf >

[14]. BRÄNDÉN, P., MANSOUR. T. *Finite Automata and Pattern Avoidance in Words* [interaktyvus]. Journal of Combinatorial Theory, Series A, Volume 110, Issue 1, April 2005, 127-145 p. [žiūrėta 2005m. kovo 7 d.]. Prieiga per internetą: <http://www.cs.utah.edu/~ndykman/Draft.pdf >

[15]. Finite Automata. Iš University of Kentucky, College of Engineering [interaktyvus]. 2004 [žiūrėta 2005m. kovo 7 d.]. Prieiga per internetą: <http://cs.engr.uky.edu/~lewis/texts/theory/automata/fin-auto.pdf>

[16]. Petri Nets. Iš Prozedatenverarbeitung (PDV) [interaktyvus]. 2005 [žiūrėta 2005 m. kovo 5 d.]. Prieiga per internetą: <http://pdv.cs.tu-berlin.de/~azi/petri.html >

[17]. An Introduction to petri nets. Iš Systems architecture laboratory [interaktyvus]. 2001 [žiūrėta 2003 m. balandžio 10 d.]. Prieiga per internetą: <http://viking.gmu.edu/http/syst511/vg511/AppC.html>

[18]. The B-Method. Iš B-Core [interaktyvus]. 2002 [žiūrėta 2003 m. balandžio 10 d.]. Prieiga per internetą: <www.**b**-core.com/ONLINEDOC/BMethod.html>

[19]. BOWEN, J.P. *Formal Specification and Documentation Using Z : A Case Study Approach* [interaktyvus]. International Thomson Computer press, 1996 [žiūrėta 2003 m. lapkričio 4 d.]. Prieiga per internetą: <http://www.zuser.org/zbook/pub/indfoils.pdf >

[20]. JACKY, J. The *Way of Z: Practical Programming with Formal Mehods* [interaktyvus]. Cambridge university Press, 1997 [žiūrėta 2003 m. lapkričio 4 d.]. Prieiga per internetą: <http://www.jpbowen.com/pub/ssm-z.pdf>

[21]. BOWEN, J.P. *Experience Teaching Z with Tool and Web Support* [interaktyvus]. ACM SIGSOFT Software Engneering Notes, Vol.26, No.2, Marc 2001, 69-75p. [žiūrėta 2003 m. lapkričio 4 d.]. Prieiga per internetą <www.cs.utexas.edu/users/csed/FM/docs/**bowen**.pdf>

[22]. Formal methods-Z Notation. Iš prof. Jonathan Bowen website [interaktyvus]. 2004 [žiūrėta 2004 m. vasario 20 d.]. Prieiga per internetą:<http://www.jpbowen.com/teaching/z.html>

[23]. SPIVEY, J. *The Z Notation: A. Reference Manual* [interaktyvus]. Prectice Hall, 1992 [žiūrėta 2003 m. lapkričio 4 d.]. Prieiga per internetą <http://spivey.oriel.ox.ac.uk/~mike/zrm/zrm.pdf>

[24]. POTTER, B., SINCLAIR, J., TILL, D. *.An introductio to Formal Specification and Z.* London, Prentice Hall, 1996.- 434p. ISBN 0-13-242207-7

[25]. Z notation. Iš Z notation [interaktyvus]. 2004 [žiūrėta 2005 m. sausio 25 d.]. Prieiga per internetą: < http://vl.zuser.org/>

[26]. HAYEK, G., ROBACH C. *From Specification Validation to Hardware Testing: A unified Method* [interaktyvus]. Proceedings International Test Conference, September 1996 [žiūrėta 2003 m. kovo 5 d.]. Prieiga per internetą: <http://csdl.computer.org/comp/proceedings/itc/1996/2121/00/2121088.pdf>

[27]. BARJAKTAROVICZ, M., CHIN, S., JABBOUR, K**.** *Formal Specification and Verification of Communication Protocols Using Automated Tools* [interaktyvus]. 1st International Conference on Engineering of Complex Computer Systems, November 1995 [žiūrėta 2003 m. kovo 5 d.]. Prieiga per internetą: <http://csdl2.computer.org/dl/proceedings/iceccs/1995/7123/00/71230246.pdf>

[28]. NELLORE, R**.** *Validating Specifications: A Contract-Based Approach* [interaktyvus]. IEEE transactions on engineering management, Vol.48, No 4, November 2001 [žiūrėta 2003 m. kovo 23 d.]. Prieiga per internetą: <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/17/20910/00969429.pdf>

[29]. GNESI, S., LENZINI, G., MARTINELLI, F. *Logical Specification and Analysis of Fault Tolerant Systems Through Partial Model Checking* [interaktyvus]. Electronic Notes in Theoretical Computer Science, 2005. -57–70p. [žiūrėta 2003 m. kovo 23 d.]. Prieiga per internetą: <www.dcs.gla.ac.uk/wirul96/papers/pdf/mueller.pdf>

[30]. ZTC. Iš Z notation [interaktyvus]. 2004 [žiūrėta 2004 m. balandžio 4 d.]. Prieiga per internetą: <http://www.jpbowen.com/cs/people/jpb/teaching/z/practical.html#ZTC>

[31]. SPIVEY, M**.** *The fuzz Manual* [interaktyvus]. The Spivey Partnership, 2nd edition, England, 1992 [žiūrėta 2004 m. balandžio 5 d.]. Prieiga per internetą: < ftp://ftp.comlab.ox.ac.uk/pub/Zforum/fuzz.>

[32]. TOYN, I ., MCDERMID, J.A. *CADiZ: An Architecture for Z Tools and its Implementation* [interaktyvus]. Software – Practice and efperience, Vol 25(3), 305–330, March 1995. [žiūrėta 2005 m. gegužės 10 d.]. Prieiga per internetą: <http://www.cs.ubc.ca/local/reading/proceedings/spe91-95/spe/vol25/issue3/spe954.pdf>

[33]. RIMKUS, M**.** Z/EVES vartotojo vadovas, *KTU, Lietuva, 2004.*

[34]. SAALTINK, M**.** *The Z/EVES 2.0 User's Guide* [interaktyvus]. TR-99-5493-06a, October 1999. [žiūrėta 2003 m. kovo 25 d.]. Prieiga per internetą: <http://www.cs.kent.ac.uk/people/staff/gsn2/zeves/usersguide.pdf>

[35]. Zans. Iš Z animations [interaktyvus]. 2002 [žiūrėta 2005 m. vasario 25 d.]. Prieiga per internetą: < http://se.cs.depaul.edu/fm/zans.html>

[36]. DUKE, R., ROSE,G., SMITH, G. *Object-Z: A specification language advocated for the description of standards* [interaktyvus]. Computer Standards & Interfaces, Volume 17, Issues 5-6, 30 September 1995, 511-533p. [žiūrėta 2005 m. sausio 5 d.]. Prieiga per internetą: <http://www.itee.uq.edu.au/~smith/svrc/tr94-45.ps.gz>

[37]. DUKE, R., ROSE,G. *Formal Object Oriented Specification using Obect Z* [interaktyvus]. Cornerstones of Computing, Macmillan 2000 [žiūrėta 2005 m. sausio 5 d.]. Prieiga per internetą: http://www.palgrave.com/pdfs/0333801237.pdf

[38]. Z and Object-Z notations in FrameMaker Documents. Iš Official Records of the ANSA Project [interaktyvus]. 2002 [žiūrėta 2005 m. sausio 10 d.]. Prieiga per internetą: http://www.ansa.co.uk/ANSATech/94/Bground/12670001.pdf

[39]. SMITH, G**.** *Reasoning about Object-Z specification* [interaktyvus]. Asia Pasific Software Engineering Conference, 1995 [žiūrėta 2004 m. sausio 20 d.]. Prieiga per internetą: <http://csdl.computer.org/comp/proceedings/apsec/1995/7171/00/71710489abs.pdf>

[40]. LOVE, M. Using Oracle/SQL to animate Z specifications [interaktyvus]. School of Computing and Managament Sciences, 1998 [žiūrėta 2004 m. sausio 20 d.]. Prieiga per internetą: <www.prakinf.tu-ilmenau.de/~czarn/anim-full.ps>

[41]. MARTIN, N., OSTROLENK, G., TOBIN, M., SOUTHWORTH, M. *Lessons from Using Z to Specify a Software Tool* [interaktyvus]. IEEE Transactions on Software Engeneering, Vol. 24, No.1, January 1998 [žiūrėta 2004 m. sausio 20 d.]. Prieiga per internetą: <http://csdl.computer.org/comp/trans/ts/1998/01/e0015abs.pdf>

[42]. PAIGE, R.F., OSTROFF, J.S. *From Z to BON/Eiffel* [interaktyvus]. Technical Report TR-98-05, York University, July 1998. [žiūrėta 2004 m. sausio 20 d.]. Prieiga per internetą: < http://www-users.cs.york.ac.uk/~paige/Writing/ztobon.pdf>

[43]. KIM, S.K., CARRINGTON, D., DUKE, R. *A Metamodel-based transformation between UML and Object-Z* [interaktyvus]. Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments, London, England, 2001. [žiūrėta 2004 m. vasario 10 d.]. Prieiga per internetą: <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/7811/21467/00995246.pdf>

[44]. Z family on the Web with their UML Photos Z [interaktyvus]. Iš Formal Specification Research. 2002 [žiūrėta 2004 m. spalio 10 d.]. Prieiga per internetą: <http://nt-appn.comp.nus.edu.sg/fm>

[45]. Specifying Java programs. Iš Imperial College [interaktyvus]. 2003 [žiūrėta 2004 m. spalio 10 d.]. Prieiga per internetą <http://www.doc.ic.ac.uk/~ar3/lectures/IndustrialMaster/SoftwareEngineering/Tutorial3/Tutorial3.pdf>

[46]. KREUZ, D**.** *Formal specification of CORBA Services using Object-Z* [interaktyvus]. Second IEEE Conference on Formal Engineering Methods, 12 09 -12, 1998, Australia **[**žiūrėta 2005 m. balandžio 20 d.]. Prieiga per internetą: <http://csdl.computer.org/comp/proceedings/icfem/1998/9198/00/91980180abs.pdf>

[47]. Cogito [interaktyvus].. Iš Cogitoergosum. 1997 [žiūrėta 2003 m. gruodžio 3 d.]. Prieiga per internetą: <http://www.cogitoergosum.co.uk>

7 TERMINŲ IR SANTRUMPŲ ŽODYNAS

ADT - abstraktūs duomenų tipai;

CASE – automatizuoto programų kūrimo sistema;

UML – unifikuota modeliavimo kalba;

SQL– struktūrizuota užklausų kalba duomenims apdoroti;

DDL – duomenų aprašymo kalba;

LaTeX – aukštos kokybės rinkimo sistema. Tai *de facto* standartas mokslinių dokumentų publikacijoms;

BON – verslo objektų notacija. Tai objektinė programavimo kalba;

OOAD – objektinė analizė ir modeliavimas;

DBVS – duomenų bazės valdymo sistema.

8 1 PRIEDAS. Z specifikacija

∪*Tvarkarastis* _____
→ [*MODULIAI, PR_IRANGA, KLASES, UZSIEMIMAI, ZINUTES*]
→ *Data* == N
→ *Laikas* == N
→ *Adresas* == *string*
→
→ → *programine_iranga_itraukta: ZINUTES*
→ → *programine_iranga_jau_egzistuoja: ZINUTES*
→ → *programine_iranga_ismesta: ZINUTES*
→ → *tokios_programines_irangos_nera_arba_ji_naudojama_kitur: ZINUTES*
→ → *modulis_itrauktas: ZINUTES*
→ → *modulis_jau_egzistuoja: ZINUTES*
→ → *modulis_ismestas: ZINUTES*
→ → *tokio_modulio_nera_arba_jis_naudojamas_kitur: ZINUTES*
→ → *klase_itraukta: ZINUTES*
→ → *klase_jau_egzistuoja: ZINUTES*
→ → *klase_ismesta: ZINUTES*
→ → *tokios_klases_nera_arba_ji_naudojama_kitur: ZINUTES*
→ → *klase_rezervuota: ZINUTES*
→ → *rezervacijai_klasiu_nera: ZINUTES*
→ → *rezervacija_panaikinta: ZINUTES*
→ → *tokios_rezervacijos_nera: ZINUTES*
→ → *taip: ZINUTES*
→ → *ne: ZINUTES*
→
→ *Fakultetu_sarasas ::= Informatikos→ Chemijos→ Dizaino_ir_technologiju→*

Ekonomikos_ir_vadybos→ Elektronikos_ir_automatikos→ ir_t_t

→ ∪*Pr_Iranga*_____

→ → *p_pavadinimas: PR_IRANGA ♣ string*
→ → // programinės įrangos pavadinimas
→ → *p_licenzijos: PR_IRANGA ♣* N
→ ∩_____
→ → // Licenzijų kiekis
→ → dom *p_pavadinimas* = dom *p_licenzijos*
→ → A *p: PR_IRANGA ∞ p_licenzijos*(*p*) ⎪ *0*
→ ∠_____
→
→ // licenzijų kiekis turi būti daugiau arba lygu 0
→ ∪*Modulis*_____
→ → *m_studentu_kiekis: MODULIAI ♣* N
→ → // studentų kiekis
→ → *m_vietu_rezervuota: MODULIAI ♣* N
→ → // kiek yra rezervuota moduliui vietų
→ →
→ → *m_destytojas: MODULIAI ♣ string*
→ → // modulį vedantis dėstytojas
→ → *m_pr_iranga: MODULIAI* φ *PR_IRANGA*

→ → // moduliui reikalinga programinė įranga
→ → *m_pavadinimas: MODULIAI ♣ string*
→ → // modulio pavadinimas
→ → *m_fakultetas: MODULIAI ♣ Fakultetu_sarasas*
→ → *m_pradzia: MODULIAI ♣ Data*
→ → // modulio pradžios data
→ → *m_pabaiga: MODULIAI ♣ Data*
→ ∩_____
→ → // modulio pabaigos data
→ → (dom *m_studentu_kiekis* = dom *m_vietu_rezervuota*) *f* (
→ → dom *m_studentu_kiekis* = dom *m_destytojas*) *f* (
→ → dom *m_studentu_kiekis* = dom *m_pavadinimas*) *f* (
→ → dom *m_studentu_kiekis* = dom *m_fakultetas*) *f* (
→ → dom *m_studentu_kiekis* = dom *m_pradzia*) *f* (
→ → dom *m_studentu_kiekis* = dom *m_pabaiga*)
→ → ran *m_pr_iranga* ζ dom *Pr_Iranga.p_pavadinimas*
→ → // modulio pr.iranga yra dalis(poaibis) bendros programinės įrangos
→ → A *m: MODULIAI* ∞ (*m_studentu_kiekis*(*m*) ⎡ *0*) *f* (*m_vietu_rezervuota*(*m*) ⎡ *0*)
→ → A *m1, m2: MODULIAI* ∞ (*m_pavadinimas*(*m1*) = *m_pavadinimas*(*m2*)) ⇒ (*m1* = *m2*)
→ → // negali būti 2 modulių su tuo pačiu pavadinimu
→ → A *m: MODULIAI* ∞ *m_pradzia*(*m*) < *m_pabaiga*(*m*)
→ ∠_____
→
→ // modulio pradzios data yra ankstenesne už modulio pabaigos datą
→ ∪*Klase*_____
→ → *k_ID: KLASES* ♣ N
→ → // klasės id
→ → *k_kompiuteriu_kiekis: KLASES* ♣ N
→ → // kompiuterių kiekis klasėje
→ → *k_pr_iranga: KLASES* φ *PR_IRANGA*
→ → // klasės pr.įranga
→ → *k_nr: KLASES* ♣ N
→ → // klasės numeris
→ → *k_pastatas: KLASES* ♣ *Adresas*
→ → // adresas
→ → *k_fakultetas: KLASES* ♣ *Fakultetu_sarasas*
→ ∩_____
→ → // fakultetas
→ → (dom *k_kompiuteriu_kiekis* = dom *k_pr_iranga*) *f* (
→ → dom *k_kompiuteriu_kiekis* = dom *k_nr*) *f* (
→ → dom *k_kompiuteriu_kiekis* = dom *k_pastatas*) *f* (
→ → dom *k_kompiuteriu_kiekis* = dom *k_fakultetas*)
→ → ran *k_pr_iranga* ζ dom *Pr_Iranga.p_pavadinimas*
→ → A *k1, k2: KLASES* ∞ ((*k_nr*(*k1*) = *k_nr*(*k2*)) *f* (*k_pastatas*(*k1*) = *k_pastatas*(*k2*))) ⇒ (*k1* = *k2*)
→ → // jei 2 klasių yra tas pats numeris ir tas pats adresas, tai vadinasi 1 ir 2 klasės
→ → // yra viena ir ta pati klasė
→ → A *k1, k2: KLASES* ∞ (*k_ID*(*k1*) = *k_ID*(*k2*)) ⇒ (*k1* = *k2*)
→ ∠_____
→
→ // dvi skirtingos klasės negali tureti tu pačiu ID numeriu
→ ∪*Rezervacija*_____
→ → *r_modulis: UZSIEMIMAI ♣ MODULIAI*
→ → // užsiemimą sudaro modulis
→ → *r_klase: UZSIEMIMAI ♣ KLASES*

→ → // užsiemimą sudaro klasė
→ → *r_grupe: UZSIEMIMAI* ♣ N
→ → // užsiemimą sudro užsiemimu grupės
→ → *r_diena: UZSIEMIMAI* ♣ *Data*
→ → // užsiemimą sudaro data
→ → *r_laikas_nuo: UZSIEMIMAI* ♣ *Laikas*
→ → // užsiemimo pradžios laikas
→ → *r_laikas_iki: UZSIEMIMAI* ♣ *Laikas*
→ ∩_____
→ → // užsiemimo pabaigos laikas
→ → (dom *r_modulis* = dom *r_klase*) *ƒ* (
→ → dom *r_modulis* = dom *r_grupe*) *ƒ* (
→ → dom *r_modulis* = dom *r_diena*) *ƒ* (
→ → dom *r_modulis* = dom *r_laikas_nuo*) *ƒ* (
→ → dom *r_modulis* = dom *r_laikas_iki*)
→ → ran *r_modulis* = dom *Modulis.m_pavadinimas*
→ → ran *r_klase* = dom *Klase.k_nr*
→ → A *u1, u2: UZSIEMIMAI* ∞ ((*r_grupe*(*u1*) = *r_grupe*(*u2*)) *ƒ* (*r_modulis*(*u1*) = *r_modulis*(*u2*))) ⇒ ((*r_laikas_nuo*(*u1*) = *r_laikas_nuo*(*u2*)) *ƒ* (*r_laikas_iki*(*u1*) = *r_laikas_iki*(*u2*)) *ƒ* (*r_klase*(*u1*) = *r_klase*(*u2*)))
→ → // 2 elementai yra tos pačios užsiemimų grupės, tik tada, kai yra tas pats modulis,
→ → // tie patys laikai nuo ir iki bei ta pati klase. Skiriasi tik datos
→ → A *u: UZSIEMIMAI* ∞ *r_laikas_nuo*(*u*) < *r_laikas_iki*(*u*)
→ → // užsiemimo pradžios laikas turi buti mažesnis nei užsiemimo pabaigos laikas
→ → A *u: UZSIEMIMAI* ∞ (*r_diena*(*u*) ⌉ *Modulis.m_pradzia*(*r_modulis*(*u*))) *ƒ* (*r_diena*(*u*) ⌋ *Modulis.m_pabaiga*(*r_modulis*(*u*)))
→ → // dienų turi buti daugiau arba lygu nei tesiasi modulis
→ → A *u1, u2: UZSIEMIMAI* ∞ ((*r_diena*(*u1*) = *r_diena*(*u2*)) *ƒ* (*r_klase*(*u1*) = *r_klase*(*u2*))) ⇒ (((
→ → *r_laikas_nuo*(*u1*) ⌉ *r_laikas_iki*(*u2*)) ϖ (*r_laikas_iki*(*u1*) ⌋ *r_laikas_nuo*(*u2*))))
→ ∠_____
→
→ ∪*state*_____
→ → Pr_Iranga
→ → Modulis
→ → Klase
→ → Rezervacija
→ ∠_____
→

→ ∪*init*_____
→ → dom *p_pavadinimas'* = { }
→ → dom *r_modulis'* = { }
→ → dom *m_pavadinimas'* = { }
→ → dom *k_nr'* = { }
→ ∠_____
→
→ ∪*Pr_Irangos_itraukimas_ok*_____
→ → Pr_Iranga
→ → Pr_Iranga'
→ → *nauja_pr_iranga: PR_IRANGA*
→ → *pavadinimas?: string*
→ → *licenzijos?:* N
→ → *zinute!: ZINUTES*

→ ∩_____
→ → pre ((! (E *p: PR_IRANGA* ∞ *p_pavadinimas*(*p*) = *pavadinimas?*)) *f*
(*nauja_pr_iranga* ™ dom *p_pavadinimas*))
→ → // naujos pr.įrangos pavadinimas negali egzistuoti jau tarp įtrauktųjų
→ → *p_pavadinimas'* = *p_pavadinimas* ± {*nauja_pr_iranga* □ *pavadinimas?*} *f*
→ → *p_licenzijos'* = *p_licenzijos* ± {*nauja_pr_iranga* □ *licenzijos?*} *f*
→ → *zinute!* = *programine_iranga_itraukta*// įtraukiamas naujas pavadinimas
→ → // įtraukiamos naujos licenzijos
→ →
→ ∠_____
→
→ ∪*Pr_Irangos_itraukimas_klaida*_____
→ → Pr_Iranga
→ → Pr_Iranga'
→ → *nauja_pr_iranga: PR_IRANGA*
→ → *pavadinimas?: string*
→ → *licenzijos?:* N
→ → *zinute!: ZINUTES*
→ ∩_____
→ → pre ((E *p: PR_IRANGA* ∞ *p_pavadinimas*(*p*) = *pavadinimas?*) ϖ
(*nauja_pr_iranga* ε dom *p_pavadinimas*))
→ → // negali įtraukti naujos pr.įrangos, nes tokia jau egzistuoja
→ → *zinute!* = *programine_iranga_jau_egzistuoja f changes_only*{}
→ ∠_____
→
→ *Pr_Irangos_itraukimas* == (*Pr_Irangos_itraukimas_ok* ϖ *Pr_Irangos_itraukimas_klaida*)
→ ∪*op Itraukti_pr_iranga*_____
→ → Pr_Irangos_itraukimas
→ ∠_____
→
→ ∪*Pr_Irangos_ismetimas_ok*_____
→ → state
→ → state'
→ → *msg!: ZINUTES*
→ → *pr_iranga?: PR_IRANGA*
→ ∩_____
→ → pre ((*pr_iranga?* ε dom *p_pavadinimas*) *f* (
→ → ! (E // turi egzistuoti nurodyta pr.įranga
→ → *k: KLASES* ∞ *k_pr_iranga*(*k*) = *pr_iranga?*)) *f* (
→ → ! (E *m: MODULIAI* ∞ *m_pr_iranga*(*m*) = *pr_iranga?*)))
→ → // pr.įranga negali buti naudojama klaseje ar moduliui
→ → *p_pavadinimas'* = (dom (*p_pavadinimas*) ∴ {*pr_iranga?*}) ρ *p_pavadinimas f*
→ → *p_licenzijos'* = (dom (*p_licenzijos*) ∴ {*pr_iranga?*}) ρ *p_licenzijos f*
→ → *msg!* = *programine_iranga_ismesta f*
→ → *changes_only*{*p_pavadinimas, p_licenzijos*}// išmetamas pr.Irangos pavadinimas
→ → // išmetamos Pr.Irangos licenzijos
→ →
→ ∠_____
→
→ ∪*Pr_Irangos_ismetimas_klaida*_____
→ → state
→ → state'
→ → *msg!: ZINUTES*
→ → *pr_iranga?: PR_IRANGA*

→ ∩_____
→ → pre ((*pr_iranga?* ™ dom *p_pavadinimas*) ϖ (
→ → E // pr.įrangos pavadinimas neegzistuoja
→ → *k: KLASES* ∞ *k_pr_iranga*(*k*) = *pr_iranga?*) ϖ (
→ → E *m: MODULIAI* ∞ *m_pr_iranga*(*m*) = *pr_iranga?*))
→ → *msg! = tokios_programines_irangos_nera_arba_ji_naudojama_kitur f changes_only*{}
→ ∠_____
→
→ *Pr_Irangos_ismetimas* == (*Pr_Irangos_ismetimas_ok* ϖ *Pr_Irangos_ismetimas_klaida*)
→ ∪*op Ismesti_pr_iranga*_____
→ → Pr_Irangos_ismetimas
→ ∠_____
→
→ ∪*Modulio_itraukimas_ok*_____
→ → Modulis
→ → Modulis'
→ → *naujas_modulis: MODULIAI*
→ → *studentu_kiekis?:* N
→ → *priranga?:* Π *PR_IRANGA*
→ → *destytojas?: string*
→ → *pavadinimas?: string*
→ → *fakultetas?: Fakultetu_sarasas*
→ → *pradzia?: Data*
→ → *pabaiga?: Data*
→ → *zinute!: ZINUTES*
→ ∩_____
→ → pre ((! (E *m: MODULIAI* ∞ *m_pavadinimas*(*m*) = *pavadinimas?*)) *f* (
→ → *naujas_modulis* ™ dom *m_studentu_kiekis*) *f* (
→ → *studentu_kiekis?* ⌉ *0*))// neturi egzistuoti nurodyto modulio pavadinimas
→ → // studentų kiekis turi buti daugiau arba lygu 0
→ → *m_studentu_kiekis'* = *m_studentu_kiekis* ± {*naujas_modulis* □ *studentu_kiekis?*} *f*
→ → *m_vietu_rezervuota'* = *m_vietu_rezervuota* ± {*naujas_modulis* □ *0*} *f*
→ → *m_destytojas'* = *m_destytojas* ± {*naujas_modulis* □ *destytojas?*} *f*
→ → *m_pr_iranga'* = *m_pr_iranga* ± ({*naujas_modulis*} ξ *priranga?*) *f*
→ → *m_pavadinimas'* = *m_pavadinimas* ± {*naujas_modulis* □ *pavadinimas?*} *f*
→ → *m_fakultetas'* = *m_fakultetas* ± {*naujas_modulis* □ *fakultetas?*} *f*
→ → *m_pradzia'* = *m_pradzia* ± {*naujas_modulis* □ *pradzia?*} *f*
→ → *m_pabaiga'* = *m_pabaiga* ± {*naujas_modulis* □ *pabaiga?*} *f*
→ → *zinute!* = *modulis_itrauktas*
→ ∠_____
→
→ ∪*Modulio_itraukimas_klaida*_____
→ → Modulis
→ → Modulis'
→ → *naujas_modulis: MODULIAI*
→ → *studentu_kiekis?:* N
→ → *destytojas?: string*
→ → *pr_iranga?: PR_IRANGA*
→ → *pavadinimas?: string*
→ → *fakultetas?: Fakultetu_sarasas*
→ → *pradzia?: Data*
→ → *pabaiga?: Data*
→ → *zinute!: ZINUTES*
→ ∩_____

→ → pre ((E *m: MODULIAI* ∞ *m_pavadinimas*(*m*) = *pavadinimas?*) ϖ (
→ → *naujas_modulis* ε dom *m_studentu_kiekis*) ϖ (
→ → *studentu_kiekis? < 0*))// negali įtraukti modulio, jei nurodyto modulio pavadinimas jau egzistuoja
→ →
→ → *zinute! = modulis_jau_egzistuoja f changes_only*{}
→ ∠_____
→
→ *Modulio_itraukimas* == (*Modulio_itraukimas_ok* ϖ *Modulio_itraukimas_klaida*)
→ ∪*op Itraukti_moduli*_____
→ → Modulio_itraukimas
→ ∠_____
→
→ ∪*Modulio_ismetimas_ok*_____
→ → state
→ → state'
→ → *msg!: ZINUTES*
→ → *modulis?: MODULIAI*
→ ∩_____
→ → pre ((*modulis?* ε dom *m_pavadinimas*) *f* (
→ → ! (E // turi egzistuoti nurodyto modulio pavadinimas
→ → *u: UZSIEMIMAI* ∞ *r_modulis*(*u*) = *modulis?*)))
→ → // negali vykti uzsiemimai nurodytam moduliui
→ → *m_studentu_kiekis'* = (dom (*m_studentu_kiekis*) ∴ {*modulis?*}) ρ *m_studentu_kiekis f*
→ → *m_vietu_rezervuota'* = (dom (*m_vietu_rezervuota*) ∴ {*modulis?*}) ρ *m_vietu_rezervuota f*
→ → *m_destytojas'* = (dom (*m_destytojas*) ∴ {*modulis?*}) ρ *m_destytojas f*
→ → *m_pr_iranga'* = (dom (*m_pr_iranga*) ∴ {*modulis?*}) ρ *m_pr_iranga f*
→ → *m_pavadinimas'* = (dom (*m_pavadinimas*) ∴ {*modulis?*}) ρ *m_pavadinimas f*
→ → *m_fakultetas'* = (dom (*m_fakultetas*) ∴ {*modulis?*}) ρ *m_fakultetas f*
→ → *m_pradzia'* = (dom (*m_pradzia*) ∴ {*modulis?*}) ρ *m_pradzia f*
→ → *m_pabaiga'* = (dom (*m_pabaiga*) ∴ {*modulis?*}) ρ *m_pabaiga f*
→ → *msg! = modulis_ismestas f*
→ → *changes_only*{*m_studentu_kiekis, m_vietu_rezervuota, m_destytojas, m_pr_iranga, m_pavadinimas, m_fakultetas*}
→ ∠_____
→
→ ∪*Modulio_ismetimas_klaida*_____
→ → state
→ → state'
→ → *msg!: ZINUTES*
→ → *modulis?: MODULIAI*
→ ∩_____
→ → pre ((*modulis?* ™ dom *m_pavadinimas*) ϖ (
→ → E // nurodyto modulio pavadinimas neegzsituoja
→ → *u: UZSIEMIMAI* ∞ *r_modulis*(*u*) = *modulis?*))
→ → // egzistuoja užsiėmimai nurodytam moduliui
→ → *msg! = tokio_modulio_nera_arba_jis_naudojamas_kitur f changes_only*{}
→ ∠_____
→
→ *Modulio_ismetimas* == (*Modulio_ismetimas_ok* ϖ *Modulio_ismetimas_klaida*)
→ ∪*op Ismesti_moduli*_____
→ → Modulio_ismetimas
→ ∠_____
→
→ ∪*Klases_itraukimas_ok*_____

→ → Klase
→ → Klase'
→ → *nauja_klase: KLASES*
→ → *ID?:* N
→ → *kompiuteriu_kiekis?:* N
→ → *priranga?:* Π *PR_IRANGA*
→ → *nr?:* N
→ → *pastatas?: Adresas*
→ → *fakultetas?: Fakultetu_sarasas*
→ → *zinute!: ZINUTES*
→ ∩_____
→ → pre ((! (E *k: KLASES* ∞ (*k_nr*(*k*) = *nr?*) *ƒ* (*k_pastatas*(*k*) = *pastatas?*))) *ƒ*
(A *k: KLASES* ∞ *k_ID*(*k*) ⎣ *ID?*) *ƒ* (*nauja_klase* ™ dom *k_nr*))
→ → // negali egzsituoti nurodyta klasė
→ → *k_ID'* = *k_ID* ± {*nauja_klase* □ *ID?*} *ƒ*
→ → *k_kompiuteriu_kiekis'* = *k_kompiuteriu_kiekis* ± {*nauja_klase* □ *kompiuteriu_kiekis?*} *ƒ*
→ → *k_pr_iranga'* = *k_pr_iranga* ± ({*nauja_klase*} ξ *priranga?*) *ƒ*
→ → *k_nr'* = *k_nr* ± {*nauja_klase* □ *nr?*} *ƒ*
→ → *k_pastatas'* = *k_pastatas* ± {*nauja_klase* □ *pastatas?*} *ƒ*
→ → *k_fakultetas'* = *k_fakultetas* ± {*nauja_klase* □ *fakultetas?*} *ƒ*
→ → *zinute!* = *klase_itraukta*
→ ∠_____
→
→ ∪*Klases_itraukimas_klaida*_____
→ → Klase
→ → Klase'
→ → *nauja_klase: KLASES*
→ → *ID?:* N
→ → *kompiuteriu_kiekis?:* N
→ → *pr_iranga?: PR_IRANGA*
→ → *nr?:* N
→ → *pastatas?: Adresas*
→ → *fakultetas?: Fakultetu_sarasas*
→ → *zinute!: ZINUTES*
→ ∩_____
→ → pre ((E *k: KLASES* ∞ (*k_nr*(*k*) = *nr?*) *ƒ* (*k_pastatas*(*k*) = *pastatas?*)) ϖ
(E *k: KLASES* ∞ (*k_ID*(*k*) = *ID?*)) ϖ (*nauja_klase* ε dom *k_nr*))
→ → *zinute!* = *klase_jau_egzistuoja ƒ changes_only*{}
→ ∠_____
→
→ *Klases_itraukimas* == (*Klases_itraukimas_ok* ϖ *Klases_itraukimas_klaida*)
→ ∪*op Itraukti_klase*_____
→ → Klases_itraukimas
→ ∠_____
→
→ ∪*Klases_ismetimas_ok*_____
→ → state
→ → state'
→ → *msg!: ZINUTES*
→ → *klase?: KLASES*
→ ∩_____
→ → pre ((*klase?* ε dom *k_nr*) *ƒ* (
→ → ! (E // turi egzistuoti nurodytos klases numeris
→ → *u: UZSIEMIMAI* ∞ *r_klase*(*u*) = *klase?*)))

→ → // nurodytoje klasėje negali vykti užsiemimai
→ → *k_ID′* = (dom (*k_ID*) ∴ {*klase?*}) ρ *k_ID f*
→ → *k_kompiuteriu_kiekis′* = (dom (*k_kompiuteriu_kiekis*) ∴ {*klase?*}) ρ *k_kompiuteriu_kiekis f*
→ → *k_pr_iranga′* = (dom (*k_pr_iranga*) ∴ {*klase?*}) ρ *k_pr_iranga f*
→ → *k_nr′* = (dom (*k_nr*) ∴ {*klase?*}) ρ *k_nr f*
→ → *k_pastatas′* = (dom (*k_pastatas*) ∴ {*klase?*}) ρ *k_pastatas f*
→ → *k_fakultetas′* = (dom (*k_fakultetas*) ∴ {*klase?*}) ρ *k_fakultetas f*
→ → *msg!* = *klase_ismesta f*
→ → *changes_only*{*k_kompiuteriu_kiekis, k_pr_iranga, k_nr, k_pastatas, k_fakultetas, r_klase*}
→ ∠_____
→
→ ∪*Klases_ismetimas_klaida*_____
→ → state
→ → state'
→ → *msg!: ZINUTES*
→ → *klase?: KLASES*
→ ∩_____
→ → pre ((*klase?* ™ dom *k_nr*) ϖ (
→ → E // tokiu numeriu klasės nėra
→ → *u: UZSIEMIMAI* ∞ *r_klase*(*u*) = *klase?*))
→ → // nurodytoje klasėje vyksta užsiėmimai
→ → *msg!* = *tokios_klases_nera_arba_ji_naudojama_kitur f changes_only*{}
→ ∠_____
→
→ *Klases_ismetimas* == (*Klases_ismetimas_ok* ϖ *Klases_ismetimas_klaida*)
→ ∪*op Ismesti_klase*_____
→ → Klases_ismetimas
→ ∠_____
→
→ ∪*Rezervacija_ok*_____
→ → state
→ → state'
→ → *uzsiemimas: UZSIEMIMAI*
→ → *modulis?: MODULIAI*
→ → *klase?: KLASES*
→ → *grupe?:* N
→ → *data?: Data*
→ → *laikas_nuo?: Laikas*
→ → *laikas_iki?: Laikas*
→ → *k_pr_iranga?: PR_IRANGA*
→ → *m_pr_iranga?: PR_IRANGA*
→ → *msg!: ZINUTES*
→ ∩_____
→ → pre ((*modulis?* ε dom *m_pavadinimas*) *f* (
→ → *klase?* ε dom *k_nr*) *f* (
→ → A // turi egzistuoti nurodytas modulis
→ → // (k_pr_iranga(|klase?|) subset m_pr_iranga (|modulis?|)) and
→ → *u: UZSIEMIMAI* ∞ ((*r_grupe*(*u*) = *grupe?*) *f* (*r_modulis*(*u*) = *modulis?*)) ⇒
((*r_laikas_nuo*(*u*) = *laikas_nuo?*) *f* (*r_laikas_iki*(*u*) = *laikas_iki?*) *f* (*r_klase*(*u*) = *klase?*))) *f* (
→ → *m_studentu_kiekis*(*modulis?*) ⌋ *m_vietu_rezervuota*(*modulis?*) *f* ((
→ → *data?* ⌉ *m_pradzia*(*modulis?*)) *f* (*data?* ⌋ *m_pabaiga*(*modulis?*))) *f* (
→ → A // m_studentu_kiekis(modulis?) >= m_vietu_rezervuota(modulis?)
→ → *u: UZSIEMIMAI* ∞ ((*r_diena*(*u*) = *data?*) *f* (*r_klase*(*u*) = *klase?*)) ⇒ ((
→ → *r_laikas_nuo*(*u*) ⌋ *laikas_iki?*) ϖ (*r_laikas_iki*(*u*) ⌉*laikas_nuo?*))) *f* (

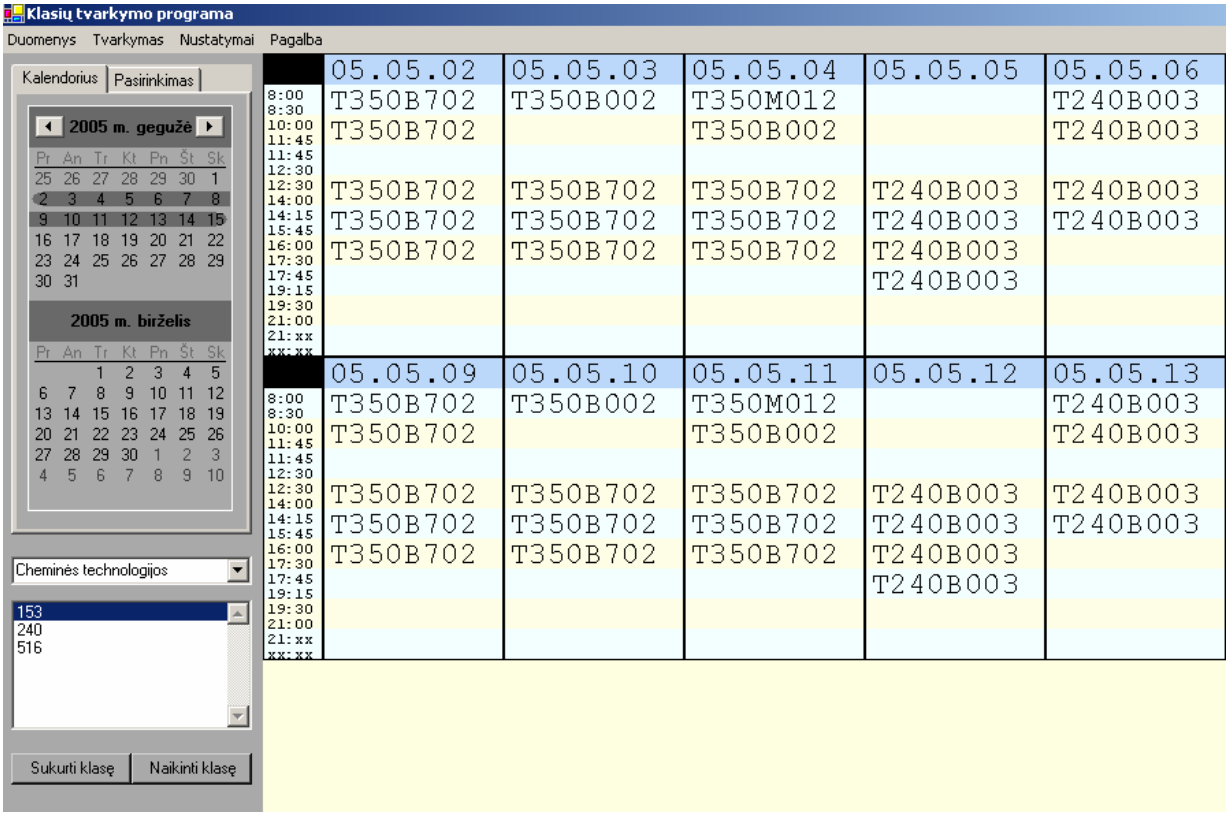
→ → *uzsiemimas* ™ dom *r_modulis*)) *f* (
→ → *r_klase'* = *r_klase* ± {*uzsiemimas* □ *klase?*}) *f* (
→ → *r_modulis'* = *r_modulis* ± {*uzsiemimas* □ *modulis?*}) *f* (
→ → *r_grupe'* = *r_grupe* ± {*uzsiemimas* □ *grupe?*}) *f* (
→ → *r_diena'* = *r_diena* ± {*uzsiemimas* □ *data?*}) *f* (
→ → *r_laikas_nuo'* = *r_laikas_nuo* ± {*uzsiemimas* □ *laikas_nuo?*}) *f* (
→ → *r_laikas_iki'* = *r_laikas_iki* ± {*uzsiemimas* □ *laikas_iki?*}) *f* (
→ → *m_vietu_rezervuota'* = *m_vietu_rezervuota* ± {*modulis?* □
(*m_vietu_rezervuota*(*modulis?*) + *k_kompiuteriu_kiekis*(*klase?*))}) *f*
→ → *msg!* = *klase_rezervuota f*
→ → *changes_only*{*r_modulis, r_klase, r_diena, r_laikas_nuo, r_laikas_iki, m_vietu_rezervuota*}
→ ∠_____
→
→ ∪*Rezervacija_kasndienu_ok*_____
→ → state
→ → state'
→ → *uzsiemimai:* seq *UZSIEMIMAI*
→ → *modulis?: MODULIAI*
→ → *klase?: KLASES*
→ → *grupe?:* N
→ → *data_nuo?: Data*
→ → *ndienu?:* N
→ → *laikas_nuo?: Laikas*
→ → *laikas_iki?: Laikas*
→ → *k_pr_iranga?: PR_IRANGA*
→ → *m_pr_iranga?: PR_IRANGA*
→ → *msg!: ZINUTES*
→ ∩_____
→ → (# *uzsiemimai* ⎡ ((*m_pabaiga*(*modulis?*) – *data_nuo?*) div *ndienu?*)) *f* (
→ → A *data: Data* ∞ (A *k:* N ∞ ((*data* = *data_nuo?* + *k* ∗ *ndienu?*) *f* (
→ → *data* ⎦ *m_pabaiga*(*modulis?*))) ⇒ (
→ → pre ((*modulis?* ε dom *m_pavadinimas*) *f* (
→ → *klase?* ε dom *k_nr*) *f* (
→ → A // klase gali buti rezervuojama nurodant pirma rezervavimo data ir kas kiek dienu vyksta
→ → // visi kiti uzsiemimai iki modulio pabaigos
→ → // uz kompiuteriu kieki klaseje
→ → // (k_pr_iranga(|klase?|) subset m_pr_iranga (|modulis?|)) and
→ → *u: UZSIEMIMAI* ∞ ((*r_grupe*(*u*) = *grupe?*) *f* (*r_modulis*(*u*) = *modulis?*)) ⇒
((*r_laikas_nuo*(*u*) = *laikas_nuo?*) *f* (*r_laikas_iki*(*u*) = *laikas_iki?*) *f* (*r_klase*(*u*) = *klase?*))) *f* (
→ → *m_studentu_kiekis*(*modulis?*) ⎦ *m_vietu_rezervuota*(*modulis?*) *f* ((
→ → *data* ⎡ *m_pradzia*(*modulis?*)) *f* (*data* ⎦ *m_pabaiga*(*modulis?*))) *f* (
→ → A // m_studentu_kiekis(modulis?) >= m_vietu_rezervuota(modulis?)
→ → *u: UZSIEMIMAI* ∞ ((*r_diena*(*u*) = *data*) *f* (*r_klase*(*u*) = *klase?*)) ⇒ ((
→ → *r_laikas_nuo*(*u*) ⎦ *laikas_iki?*) ϖ (*r_laikas_iki*(*u*) ⎡ *laikas_nuo?*))) *f* (
→ → *uzsiemimai*(*k*) ™ dom *r_modulis*)) *f* (
→ → *r_klase'* = *r_klase* ± {*uzsiemimai*(*k*) □ *klase?*}) *f* (
→ → *r_modulis'* = *r_modulis* ± {*uzsiemimai*(*k*) □ *modulis?*}) *f* (
→ → *r_grupe'* = *r_grupe* ± {*uzsiemimai*(*k*) □ *grupe?*}) *f* (
→ → *r_diena'* = *r_diena* ± {*uzsiemimai*(*k*) □ *data*}) *f* (
→ → *r_laikas_nuo'* = *r_laikas_nuo* ± {*uzsiemimai*(*k*) □ *laikas_nuo?*}) *f* (
→ → *r_laikas_iki'* = *r_laikas_iki* ± {*uzsiemimai*(*k*) □ *laikas_iki?*}) *f* (
→ → *m_vietu_rezervuota'* = *m_vietu_rezervuota* ± {*modulis?* □
(*m_vietu_rezervuota*(*modulis?*) + *k_kompiuteriu_kiekis*(*klase?*))}) *f*
→ → *msg!* = *klase_rezervuota f*

→ → *changes_only{r_modulis, r_klase, r_diena, r_laikas_nuo, r_laikas_iki, m_vietu_rezervuota}*// uzsiemimui priskiriama klase
→ → // uzsiemimui priskiriamas modulis
→ → // uzsiemimui priskiriama uzsiem.grupe
→ → // uzsiemimui priskiriama pirma diena
→ → // paskaitos pradzia
→ → // paskaitos pabaiga
→ → // rezervuota tiek vietu kiek rezervuota kompiuteriu klasese
→ → // modulio nerezervuotu vietu likutis
→ ∠_____
→
→ ∪*Rezervacija_klaida*_____
→ → state
→ → state'
→ → *uzsiemimas: UZSIEMIMAI*
→ → *modulis?: MODULIAI*
→ → *klase?: KLASES*
→ → *grupe?:* N
→ → *data?: Data*
→ → *laikas_nuo?: Laikas*
→ → *laikas_iki?: Laikas*
→ → *msg!: ZINUTES*
→ ∩_____
→ → pre ((*modulis?* ⁙ dom *m_pavadinimas*) ϖ (
→ → *klase?* ⁙ dom *k_nr*) ϖ
→ → ! (A // neegzistuoja nurodytas modulis
→ → // neegsituoja nurodytos klases numeris
→ → *u: UZSIEMIMAI* ∞ ((*r_grupe*(*u*) = *grupe?*) *f* (*r_modulis*(*u*) = *modulis?*)) ⇒
((*r_laikas_nuo*(*u*) = *laikas_nuo?*) *f* (*r_laikas_iki*(*u*) = *laikas_iki?*) *f* (*r_klase*(*u*) = *klase?*))) ϖ (
→ → *m_studentu_kiekis*(*modulis?*) ⌋ *m_vietu_rezervuota*(*modulis?*) ϖ
→ → ! ((*data?* ⌉ *m_pradzia*(*modulis?*)) *f* (*data?* ⌋ *m_pabaiga*(*modulis?*))) ϖ
→ → ! (A // (m_studentu_kiekis(modulis?) m_vietu_rezervuota(modulis?)
→ → // studentu kiekis
→ → // netinkamos datos nurodytam modulio
→ → *u: UZSIEMIMAI* ∞ ((*r_diena*(*u*) = *data?*) *f* (*r_klase*(*u*) = *klase?*)) ⇒ ((
→ → *r_laikas_nuo*(*u*) ⌉ *laikas_iki?*) ϖ (*r_laikas_iki*(*u*) ⌋ *laikas_nuo?*))) ϖ (
→ → *uzsiemimas* ε dom *r_modulis*)) *f*
→ → *msg! = rezervacijai_klasiu_nera f changes_only{}*
→ ∠_____
→
→ *Rezervacija_visa* == (*Rezervacija_ok* ϖ *Rezervacija_klaida*)
→ ∪*op Rezervuoti*_____
→ → Rezervacija_visa
→ ∠_____
→
→ ∪*Rezervacija_naikinti_ok*_____
→ → state
→ → state'
→ → *modulis?: MODULIAI*
→ → *grupe?:* N
→ → *msg!: ZINUTES*
→ ∩_____
→ → pre (E *u: UZSIEMIMAI* ∞ ((*r_modulis*(*u*) = *modulis?*) *f* (*r_grupe*(*u*) = *grupe?*))) *f* (
→ → A // egzistuoja uzsiemimas nurodytam moduliui

→ → *k: KLASES* ∞ A *u: UZSIEMIMAI* ∞ ((*r_modulis*(*u*) = *modulis?*) *ƒ* (*r_grupe*(*u*) = *grupe?*) *ƒ* (*r_klase*(*u*) = *k*)) ⇒
→ → *m_vietu_rezervuota′* = *m_vietu_rezervuota* ± {*modulis?* □ (*m_vietu_rezervuota*(*modulis?*) − *k_kompiuteriu_kiekis*(*k*))}) *ƒ* (
→ → A *u: UZSIEMIMAI* ∞ ((*r_modulis*(*u*) = *modulis?*) *ƒ* (*r_grupe*(*u*) = *grupe?*)) ⇒
→ → *r_modulis′* = (dom (*r_modulis*) ∴ {*u*}) ρ *r_modulis ƒ*
→ → *r_klase′* = (dom (*r_klase*) ∴ {*u*}) ρ *r_klase ƒ*
→ → *r_diena′* = (dom (*r_diena*) ∴ {*u*}) ρ *r_diena ƒ*
→ → *r_laikas_nuo′* = (dom (*r_laikas_nuo*) ∴ {*u*}) ρ *r_laikas_nuo ƒ*
→ → *r_laikas_iki′* = (dom (*r_laikas_iki*) ∴ {*u*}) ρ *r_laikas_iki*) *ƒ*
→ → *msg!* = *rezervacija_panaikinta ƒ*
→ → *changes_only*{*r_modulis, r_klase, r_diena, r_laikas_nuo, r_laikas_iki, m_vietu_rezervuota*}
→ ∠_____
→
→ ∪*Rezervacija_naikinti_klaida*_____
→ → state
→ → state'
→ → *modulis?: MODULIAI*
→ → *grupe?:* N
→ → *msg!: ZINUTES*
→ ∩_____
→ → pre (! (E *u: UZSIEMIMAI* ∞ ((*r_modulis*(*u*) = *modulis?*) *ƒ* (*r_grupe*(*u*) = *grupe?*)))) *ƒ*
→ → *msg!* = *tokios_rezervacijos_nera ƒ changes_only*{}// neegzsituoja uzsiemimas nurodytam moduliui
→ →
→ ∠_____
→
→ *Rezervacija_naikinti* == (*Rezervacija_naikinti_ok* ϖ *Rezervacija_naikinti_klaida*)
→ ∪*op Panaikinti_rezervacija*_____
→ → Rezervacija_naikinti
→ ∠_____
→
∠_____

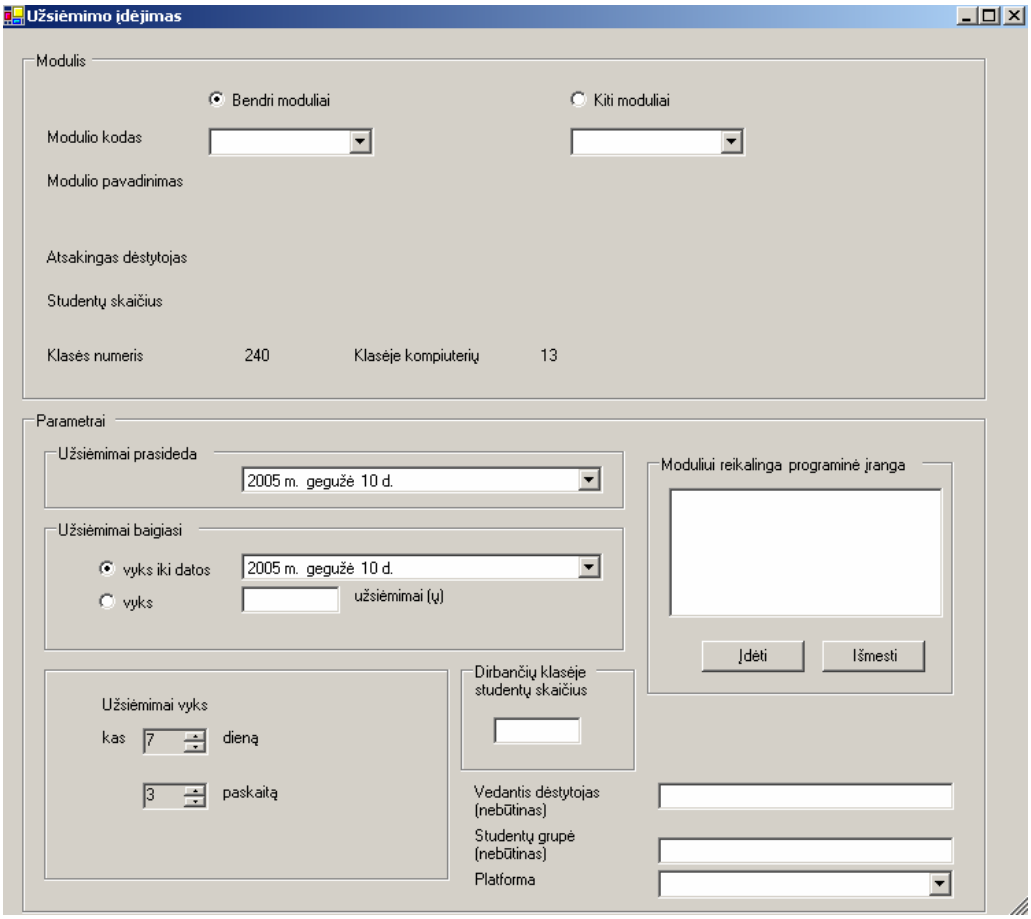
9 2 PRIEDAS. Kompiuterių klasių užimtumo informacinės sistemos ekrano kopijos

Vartotojas prisijungęs prie informacinės sistemos gali peržiūrėti jį dominančios klasės tvarkaraštį. Jame matosi, koks modulis ir kokiu laiku yra vedamas pasirinktoje klasėje bei kokiu laiku ir kokiomis dienomis kompiuterių klasė yra laisva. Paveikslėlyje pateiktas Cheminių technologijų fakulteto 153 kompiuterių klasės dviejų savaičių užsiėmimų tvarkaraštis (žr. 9.1 pav.).



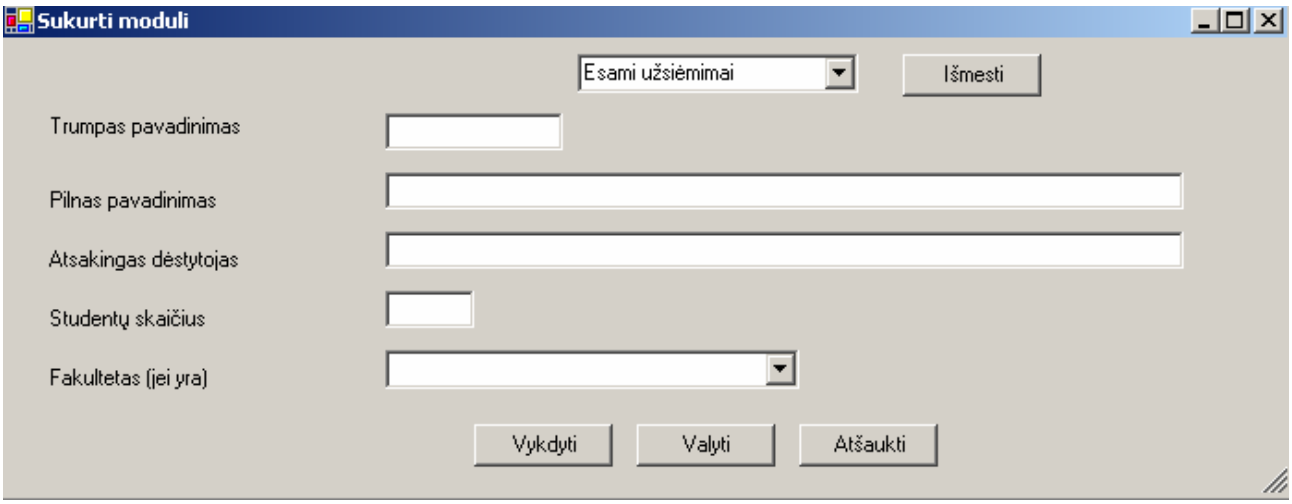
9.1 pav. Chemijos fakulteto 153 kompiuterių klasės dviejų savaičių užsiėmimų tvarkaraštis

Pasirinktu laiko periodu, nurodžius modulį galima rezervuoti vieną ar kelias neužimtas kompiuterių klases užsiėmimų pravedimui(žr. 9.2 pav.).

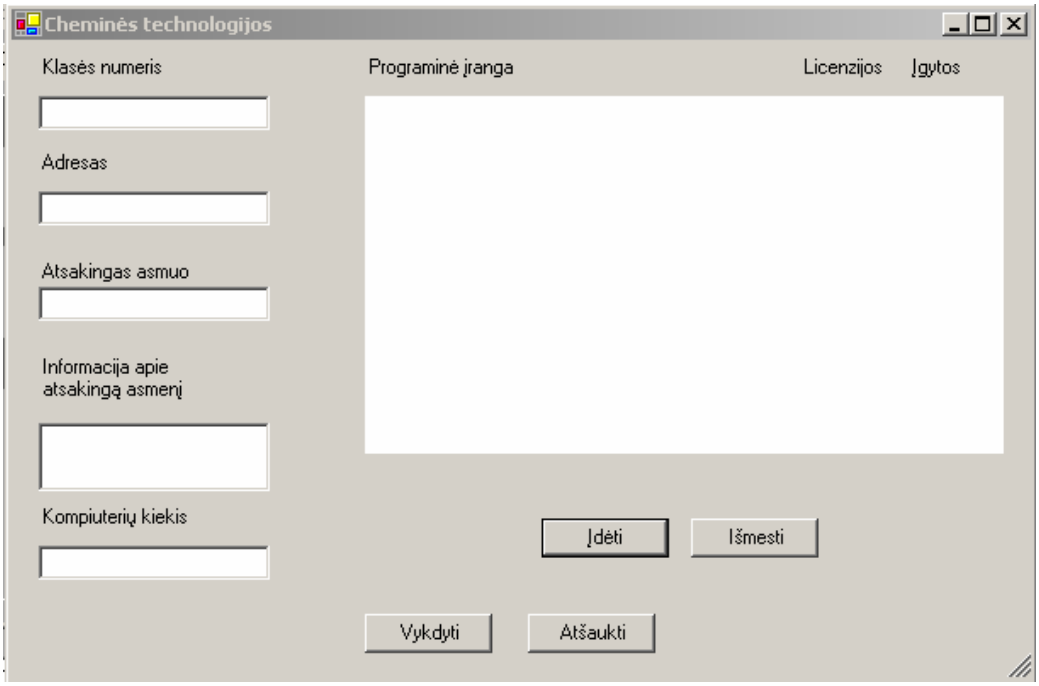


9.2 pav. Užsiėmimo rezervavimas

Informacinėje sistemoje galimos tokios operacijos: sukurti ir panaikinti užsiėmimo rezervavimą kompiuterių klasėje, įtraukti naujus ir panaikinti jau esamus modulius (žr. 9.3 pav.), įtraukti naujas ir pašalinti jau įtrauktas klases (žr. 9.4 pav.), įtraukti naują programinę įranga ir panaikinti esamą (žr. 9.5 pav.), rezervuoti klasę bet kokiam pageidaujamam laikotarpiui.

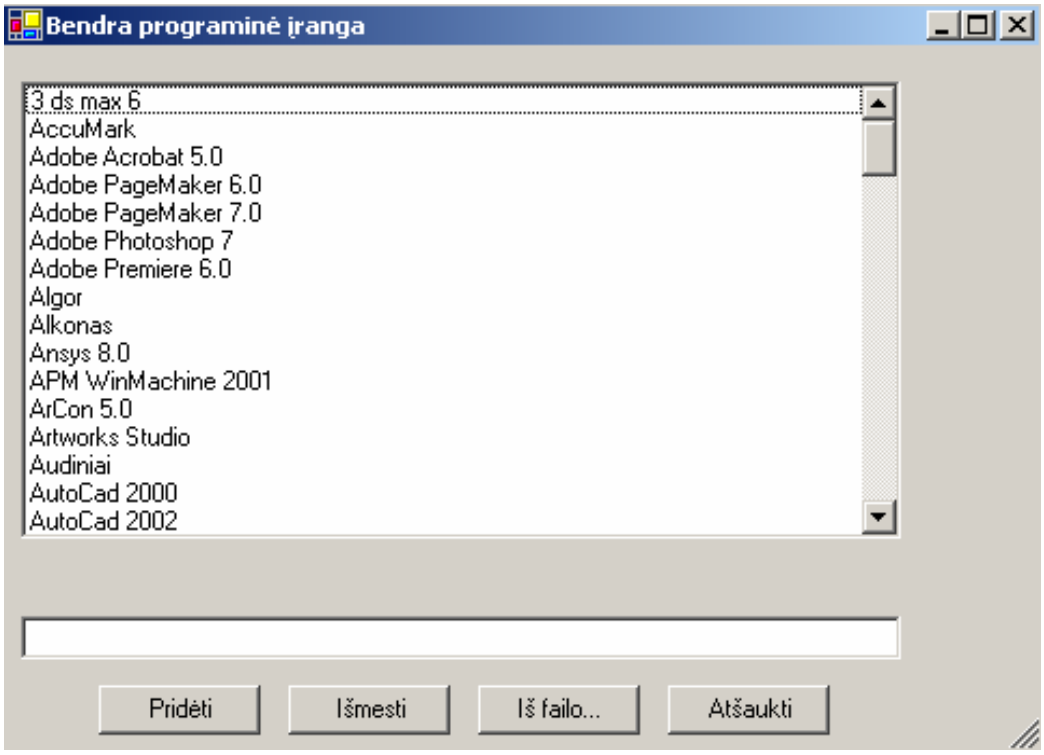


9.3 pav. Modulio įtraukimas/išmetimas



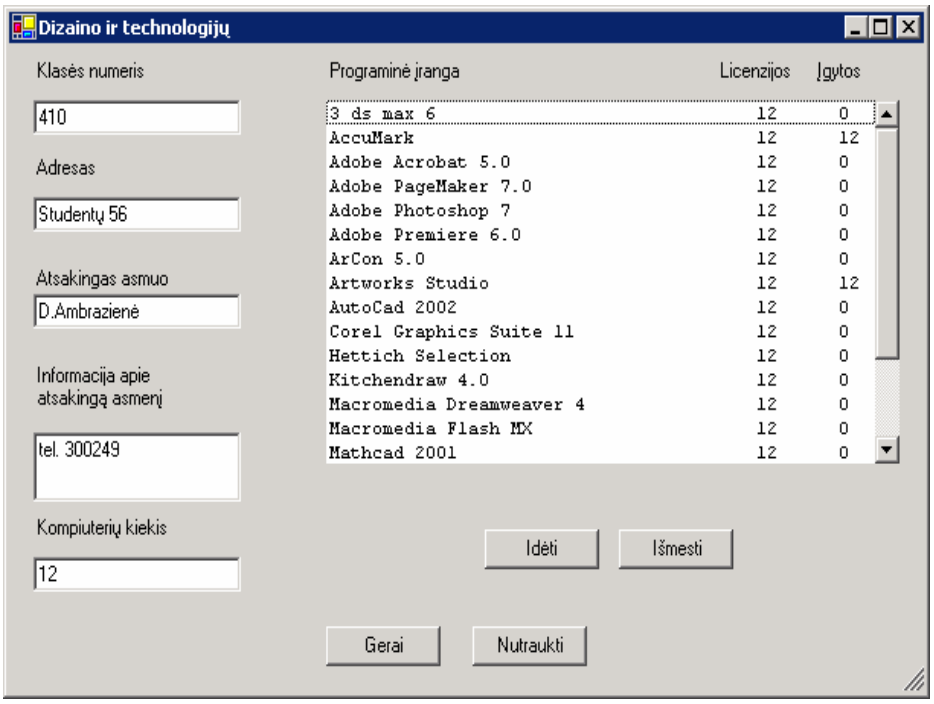
9.4 pav. Klasės įtraukimas/pašalinimas

Žemiau pateiktas programinės įrangos langas, kuriame yra matoma visa mokyme procese naudojama programinė įranga.



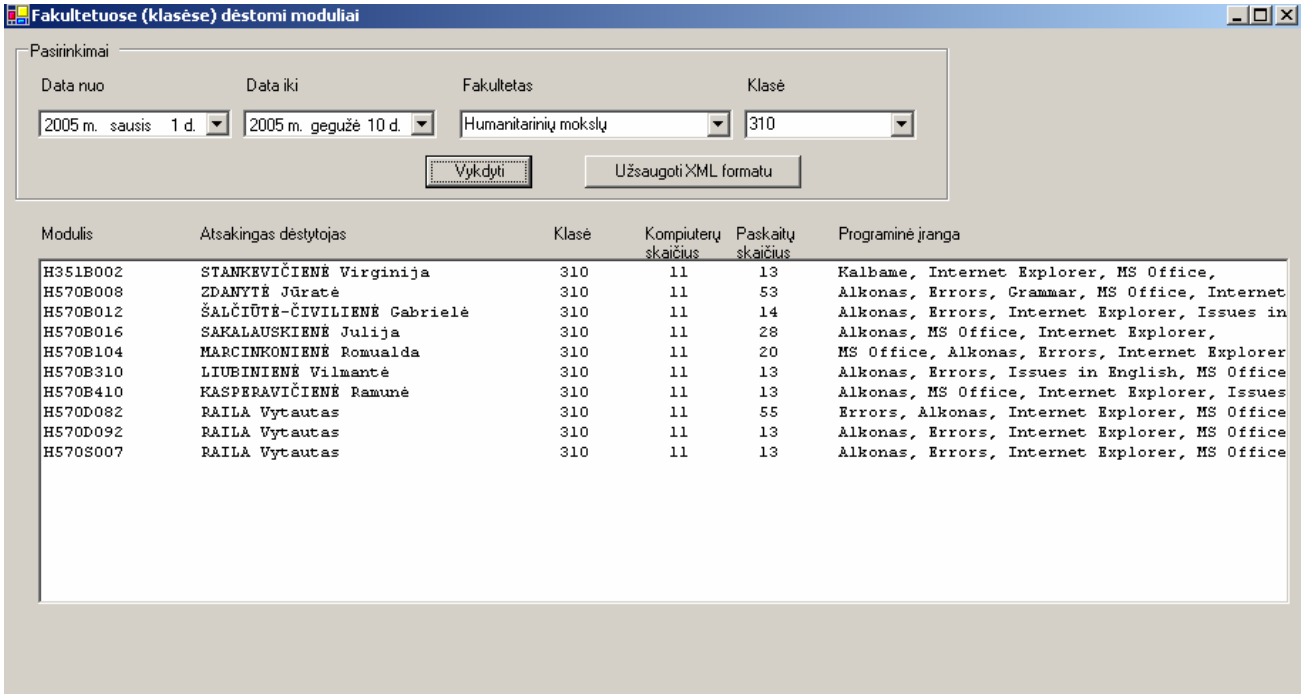
9.5 pav. Programinės įrangos įtraukimas/pašalinimas

IS kaupiama informacija apie bendrą programinę įrangą, KTU dėstomus modulius, kompiuterių klases (žr. 9.6 pav.). Paveikslėlyje pateikti duomenys apie Dizaino ir technologijų fakulteto 410 kompiuterių klasę, joje instaliuotą programinę įrangą, turimas licenzijas, kompiuterių skaičių bei informaciją apie atsakingą asmenį.



9.6 pav. Duomenys apie 410 kompiuterių klasę

Žemiau (žr. 9.7 pav.) pateikta informacija apie 310 klasėje, kuri yra humanitarinių mokslų fakultete, dėstomus modulius pasirinktu laikotarpiu. Taip pat šalia yra pateikiama informacija apie šiems moduliams pravesti reikalingą programinę įrangą, paskaitų skaičių, atsakingą asmenį.



9.7 pav. Humanitarinių mokslų fakultete, 310 kompiuterių klasėje dėstomi moduliai.

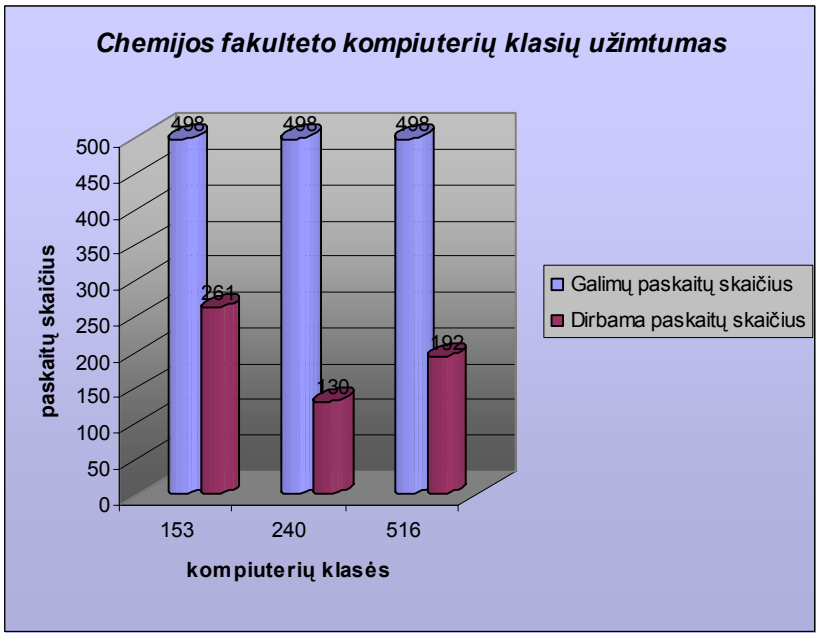
10 3 PRIEDAS. Kauno technologijos universiteto kompiuterių klasių užimtumo tyrimai

Naudojant užklausas, gautus duomenis galima užsaugoti „xml“ formatu, kuris suteikia galimybę atlikti statistinę duomenų analizę. Kompiuterių klasių analizės užklausa leidžia pateikti statistiką apie fakulteto kompiuterių klases (žr. 10.1 lentelė, 10.1 pav.).

10.. lentelė.

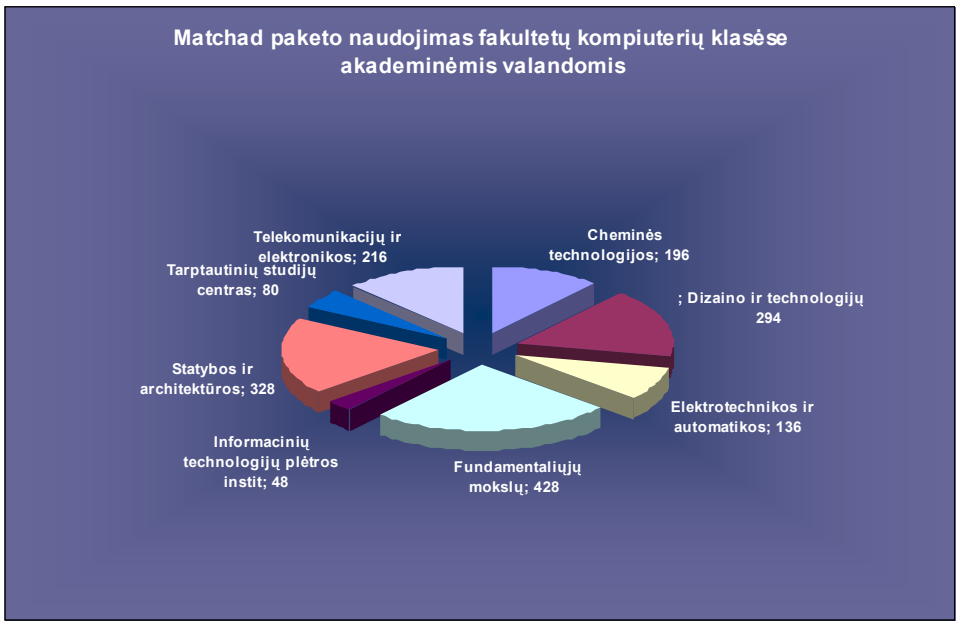
Informacija apie chemijos fakulteto kompiuterių klases.

Fakultetas	Klase	Savaites_dienos	Paskaito	Darbo_vietu_skaicius	Galimu_paskaitu_skaicius	Dirbama_paskaitu	Uzimta_pr	Vidutiniskai_dirba studentu	Laikotarpis
Cheminės technologijos	153	12345	124567	13	498	261	52,41	11,53	2004.09.01-2004.12.24
Cheminės technologijos	240	12345	124567	13	498	130	26,10	12,26	2004.09.01-2004.12.24
Cheminės technologijos	516	12345	124567	28	498	192	38,55	23,50	2004.09.01-2004.12.24



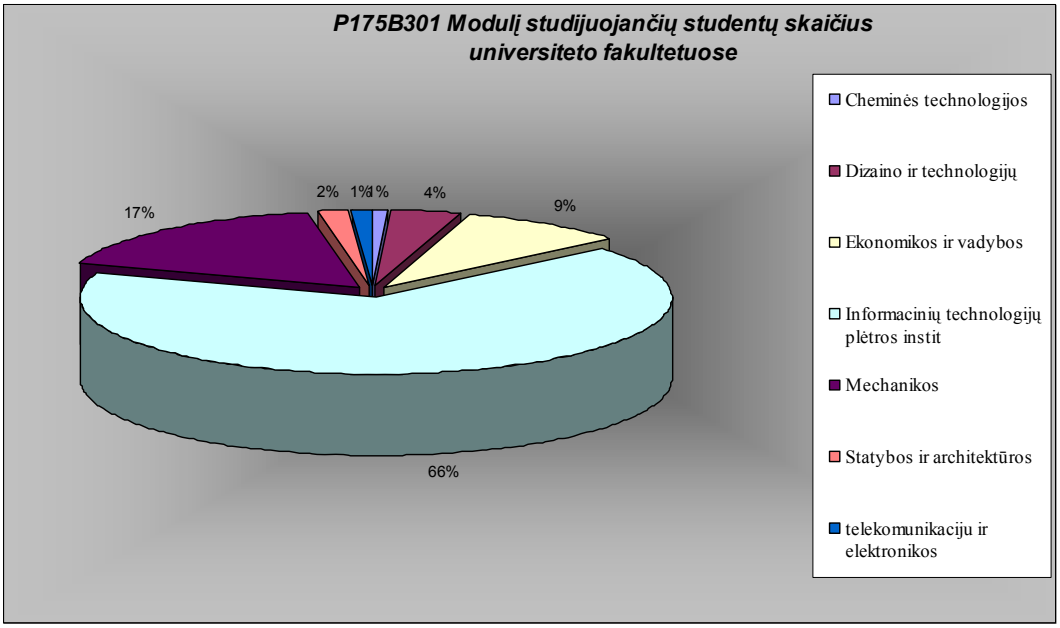
10.1 pav. Chemijos fakulteto kompiuterių klasių užimtumas.

Programinės įrangos panaudojimo užklausa leidžia analizuoti programinių paketų panaudojimo mokymo procese statistiką (žr. 10.2 pav.). Paveikslėlyje pateikiamas kompiuterių klasėse įdiegto programinio paketo Matchad panaudojimas mokymo proceso metu, vieno semestro laikotarpyje. Grafike duomenys pateikti akademinėmis valandomis.



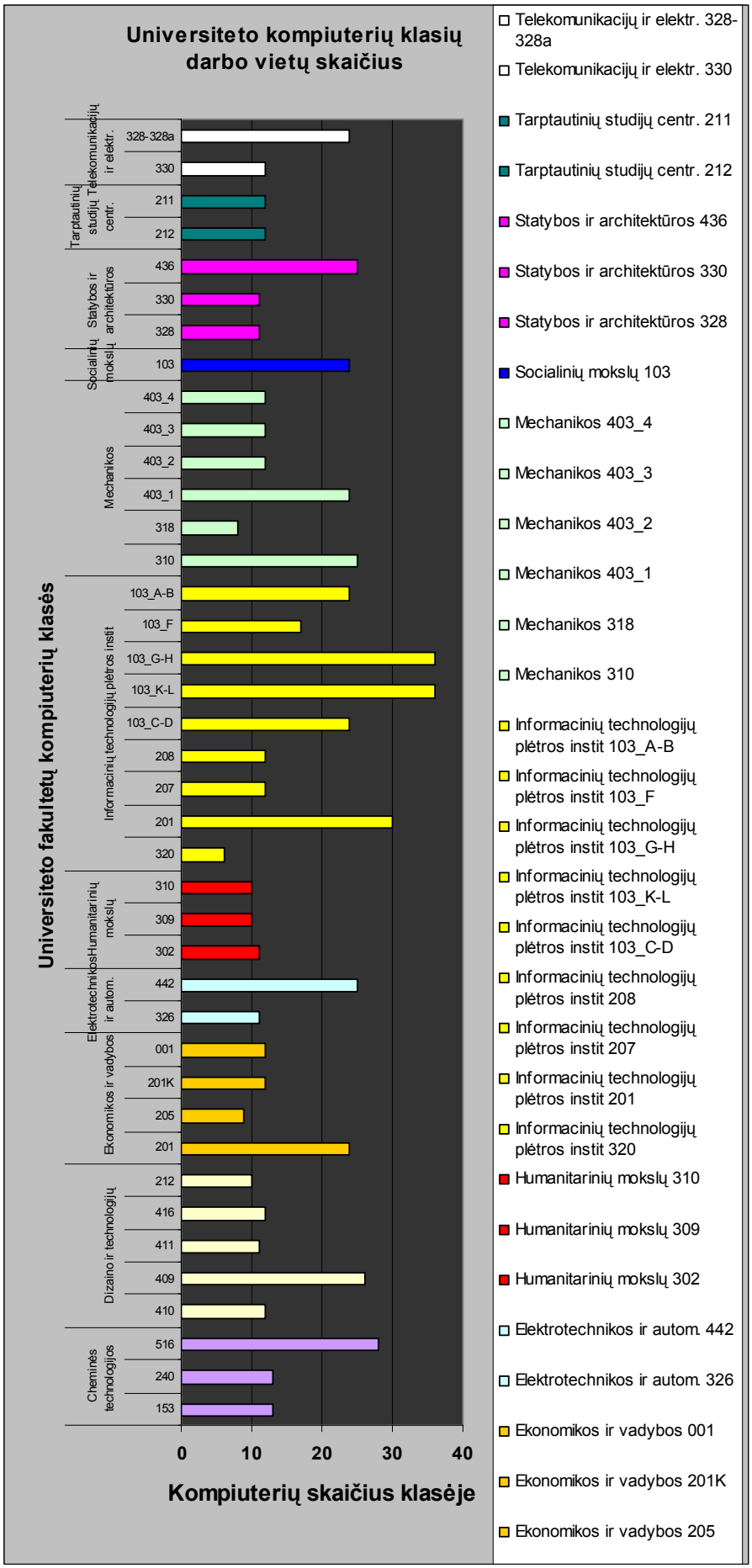
10.2 pav. Matchad 2001 naudojimas fakultetų kompiuterių klasėse

Modulio dėstymo užklausa leidžia pamatyti, kokiuose fakultetuose yra vedamas pasirinktas modulis, kiek studentų studijuoja šį modulį kiekviename fakultete atskirai (žr. 10.3 pav.). Paveikslėlyje matosi, jog modulį P175B301 (Informatika 1) daugiausiai studentų studijuoja Informacinių technologijų plėtros institute.



10.3 pav. Modulio P175B301 statistinė analizė

Kompiuterių klasių užklausa leidžia pamatyti, koks yra darbo vietų skaičius atskirų fakultetų kompiuterių klasėse (žr. 10.4 pav.). Kaip matyti iš paveikslėlio didžiausia kompiuterių klasė turi trisdešimt šešias darbo vietas, mažiausia yra kompiuterių klasė turinti šešias darbo vietas.



10.4 pav. Universiteto kompiuterių klasių darbo vietų skaičius.

11 4 PRIEDAS. Straipsniai

KOMPIUTERIŲ KLASIŲ UŽIMTUMO FORMALI ANALIZĖ

H. Pranevičius*, D. Ambraziene**, V. Kuosa*, A. Paulauskaite*
*Kauno Technologijos Universitetas *Verslo Informatikos katedra **Informaciniu technologijų plėtros institutas*
Studentų g. 56-301, LT-3031 Kaunas

Straipsnyje yra pateikiama kompiuterių klasių užimtumo formali specifikacija. Formali specifikacija aprašo klasių resursus, dėstomus modulius, galimus užsiėmimų pravedimo laikus bei užsiėmimu pravedimo rezervavimo tinklelį. Sudaryta formali specifikacija leidžia analizuoti, ar visi moduliai, numatyti dėstyti kompiuterių klasėse gali būti pravesti pageidaujamose klasėse, ar modulio pravedimas išsiplečia per kelias klases, esančias net skirtinguose fakultetuose.

1. Įvadas

Šiuolaikinės informacinės technologijos įtakoja Kauno Technologijos universiteto studijų procesą. Mokymo proceso kokybės užtikrinimas numato taikomųjų paketų ir projektavimo priemonių, procesų modeliavimo priemonių, dalykinių žaidimų, žinių globaliam kompiuterių tinkle, mokymosi medžiagos elektroninėje formoje naudojimą. Net senųjų tradicinių modulių (matematika, braižyba) dėstymas šiandien persipina su informacinių technologijų panaudojimu, kadangi tradiciniai moduliai dėstomi dideliam studentu skaičiui, tai ženkliai padidina užsiėmimų poreikį kompiuterių klasėse. Studijų procesas vyksta fakultetų ir Informacinių technologijų plėtros instituto kompiuterinėse klasėse. Universitete įrengta daugiau kaip 30 kompiuterių klasių, kuriose daugiau kaip 500 darbo vietų. Klasių kompiuteriuose įdiegtos 5 operacinės sistemos, kelios dešimtys klasikinių taikomųjų programų paketų, tokių kaip: MS Office, Adobe Photoshop, AutoCad, CorelDRAW, MatLab, eilė servisinių programų bei specializuota programinė įranga, būdinga atskiriems fakultetams bei juose dėstomiems moduliams.

2. Kompiuterių klasių užimtumo modelis

Universitete kompiuterių klasėse dėstomi moduliai pravedami labai įvairiam studentų skaičiui. Modulis dėstomas tam tikrą fiksuotą valandų skaičių per savaitę. Moduliai tiesiogiai susiję su konkrečiu programiniu paketu panaudojimu ir kompiuterinėmis klasėmis, kur šie moduliai bus pravedami. Kompiuterių klasės randasi įvairiuose fakultetuose, institutuose. Kompiuterių skaičius universiteto kompiuterių klasėse yra labai skirtingas (nuo 10 iki 36), kompiuterių specifikacija plataus diapazono.

Straipsnyje pateikiamas kompiuterių klasių užimtumo formalus aprašymas, naudojant Z kalbos notaciją [1,2]. Z schemų aprašymui panaudotos standartinės programinės priemonės Cogito [3].

Z specifikavimo kalba plačiai naudojama aprašant kompiuterines sistemas. Ji leidžia vienareikšmiškai, matematiškai aprašyti nagrinėjimą sistemą ir jos invariantines savybes, lengvai aprašo įvairius abstrakcijos lygius. Z specifikavimo kalba turi priemones, kuriomis galima aprašyti sudėtingas duomenų struktūras.

Sukurtas kompiuterių klasių užimtumo formalus aprašymas leidžia analizuoti:

- Ar visi moduliai, numatyti dėstyti kompiuterių klasėse gali būti pravesti pageidaujamose klasėse?
- Ar modulio pravedimas išsiplės per kelias mažesnes klases?
- Ar net bus ieškoma galimybės pravesti modulius kitų fakultetų klasėse?

Formalus metodai leidžia aprašyti klases, modulius, operacijas, kad įvertinti galimybę pravesti užsiėmimus konkrečioje klasėje, konkrečiu laiku.

Formalizavimo metu, kaip pradiniai duomenys imami: modulis, kuris bus pravedamas kompiuterių klasėje ar klasėse, studentų kiekis dėstomam moduliui, programinį įranga, pageidaujamas užsiėmimų pravedimo laikas, kompiuterių klasės identifikatorius.

3. Formalus aprašymas

Specifikacijoje naudojamos keturios bazinės aibės: studijų modulių, kompiuterių klasių, užsiėmimų pravedimo laikų ir užsiėmimų pravedimo rezervavimo.

[Modulis, Klase, Pask_Laikas, Rezervacija]

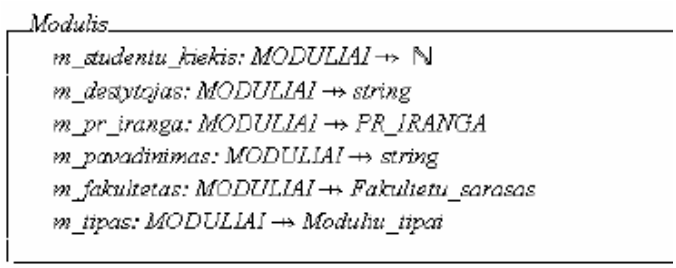
Klasių užimtumo analizės formalizavimui naudojamų aibių atributai

Modulis:

- Studentų kiekis,
- Modulį vedantis dėstytojas,
- Moduliui reikalinga programinė įranga,
- Modulio pavadinimas,
- Fakulteto pavadinimas
- Modulio tipas

Modulio tipas gali būti dvejopas: bendrasis, kuris dėstomas I ir II kurso studentams bei specialybinis , kuris dėstomas III, IV ir vyresniųjų kursų studentams.

Moduliu_tipai ::= bendrasis| specialybinis



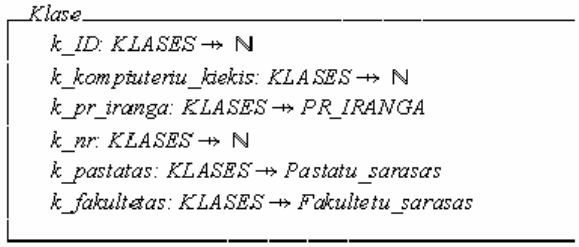
Aibėje negali būti dviejų modulių tuo pačiu pavadinimu

A *m1, m2: MODULIAI* • (*m_pavadinimas*(*m1*) = *m_pavadinimas*(*m2*)) . (*m1* = *m2*)

Klase:

• Klasės identifikavimo numeris,
• Kompiuterių kiekis,
• Kompiuterių klasėje esanti programinė įranga,
• Klasės numeris,
• Rūmai,
• Fakulteto pavadinimas.

Fakultetu_sarasas::= Informatikos| *Chemijos*| *Dizaino_ir_technologiju*| *Ekonomikos_ir_vadybos*| *Elektronikos_ir_automatikos*| *Mechanikos* | *Statybos_ir_Architekturos*| *Humanitariniu*_moksl*u**ir_t_t*



Aibėje negali būti dviejų klasių tam pačiame pastate, tokiais pačiais numeriais

A *k1, k2: KLASES* • *((k_nr(k1) = k_nr(k2))* ¶ *(k_pastatas(k1) = k_pastatas(k2))) . (k1 = k2)*

A *k1, k2: KLASES* • *(k_ID(k1) = k_ID(k2)) . (k1 = k2*)

Pask_Laikas:

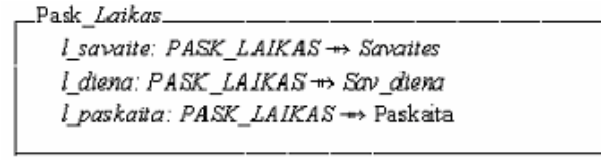
• Savaitė
• Paskaita

Universitetuose savaitė gali buti pirma arba antra. Paskaitos yra nuo pirmos iki šeštos.

Savaites ::= pirma | *antra;*

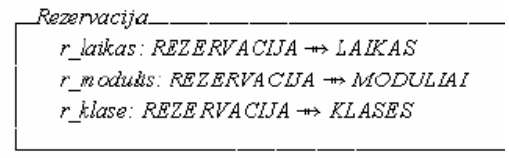
Paskaita :: = pirma | *antra* | *trečia* | *ketvirta* | *penkta* | *sesta*

Sav_diena ::= pirmadienis | *antradienis* | *treciadienis* | *ketvirtadienis* | *penktadienis* | *siastadienis* | *sekmadienis;*

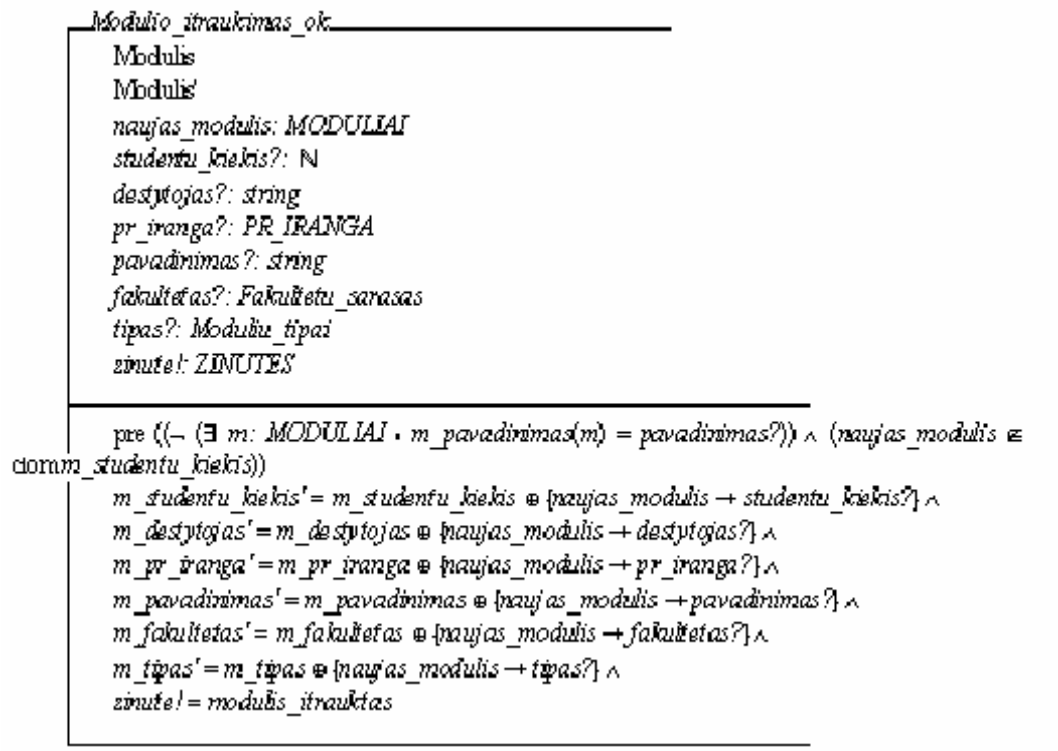


Rezervacija:

• Galimų užsiemimų pravedimo laikai,
• Modulis,
• Kompiuterių klasė.



Operacijos



Sistemoje naudojamos tokios operacijos:

Operacijos, aprašančios nauju moduliu atsiradim[1] ar pasikeitim[1]:

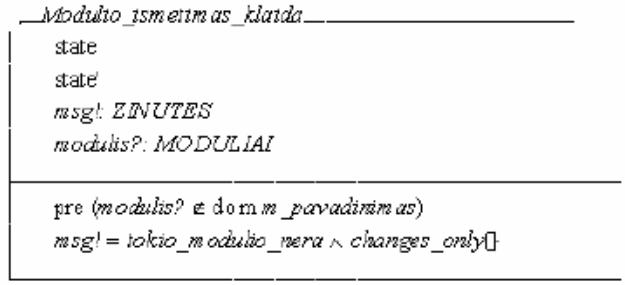
- Naujo modulio įtraukimas, (*Modulio_itraukimas_ok*)
- Klaidingas modulio įtraukimas, *(Modulio_itraukimas_klaida*)
- Įtraukto modulio panaikinimas, (*Modulio_ismetimas_ok*)
- Klaidingas modulio panaikinimas (*Modulio_ismetimas_klaida*)

Schema Modulio_itraukimas_ok aprašo naujo modulio įvedimą. Tam nurodomas studentų kiekis, dėstytojas, programinę įrangą, pavadinimas, fakultetas, modulio tipas, naujas modulis.

Predikate patikrinama ar neegzistuoja jau toks modulio pavadinimas.(1 pred. eilute). Jei tokio modulio nėra, tada naujo modulio įtraukimas yra sėkmingas.

zinute! = modulis_itrauktas

Schema Modulio._ismetimas_klaida išmeta pranešimą, kai modulis kurį pageidaujama išmesti neegzistuoja. T.y toks modulis nebuvo prieš tai įvestas.



. Predikate nustatoma, kad neegzistuoja toks modulio pavadinimas.(1 pred. eilute)

Operacijos, aprašančios naujų klasių steigimą ar pašalinimą:

- Naujos klasės įvedimas, (Klases_itraukimas_ok)
- Klaidingas klasės įvedimas, (Klases_itraukimas_klaida)
- Įtrauktos klasės panaikinimas, (Klases_ismetimas_ok)
- Klaidingas klasės panaikinimas, (klases_ismetimas_klaida)

Schema Klases_itraukimas_ok įtraukia naują klasę

Klases_itraukimas_ok

Klase
Klase'
nauja_klase: KLASES
ID?: ℕ
kompiuteriu_kiekis?: ℕ
pr_iranga?: *PR_IRANGA*
nr?: ℕ
pastatas?: *Pastatu_sarasas*
fakultetas?: *Fakultetu_sarasas*
zinute!: *ZINUTES*

---

pre $((\neg (\exists k: KLASES \cdot (k\_nr(k) = nr?) \wedge (k\_pastatas(k) = pastatas?))) \wedge (\forall k: KLASES \cdot k\_ID(k) \neq ID?) \wedge (nauja\_klase \notin \mathrm{dom} k\_nr))$
$k\_ID' = k\_ID \oplus \{nauja\_klase \mapsto ID?\} \wedge$
$k\_kompiuteriu\_kiekis' = k\_kompiuteriu\_kiekis \oplus \{nauja\_klase \mapsto kompiuteriu\_kiekis?\} \wedge$
$k\_pr\_iranga' = k\_pr\_iranga \oplus \{nauja\_klase \mapsto pr\_iranga?\} \wedge$
$k\_nr' = k\_nr \oplus \{nauja\_klase \mapsto nr?\} \wedge$
$k\_pastatas' = k\_pastatas \oplus \{nauja\_klase \mapsto pastatas?\} \wedge$
$k\_fakultetas' = k\_fakultetas \oplus \{nauja\_klase \mapsto fakultetas?\} \wedge$
$zinute! = klase\_itraukta$

Predikate patikrinama ar neegzistuoja tokia pati klasė (1 pred. eilute). Jei šios klasės dar nėra, tai įtraukiami naujos klasės parametrai: klasės identifikacijos numeris, kompiuterių kiekis, programinė įranga, klasės numeris, pastatas, bei fakultetas.

Schema Klases_itraukimas_klaida parodo, kad negalima įtraukti tokios klasės kadangi jau tokia klasė egzistuoja

Klases_itraukimas_klaida

Klase
Klase'
nauja_klase: KLASES
ID?: ℕ
kompiuteriu_kiekis?: ℕ
pr_iranga?: *PR_IRANGA*
nr?: ℕ
pastatas?: *Pastatu_sarasas*
fakultetas?: *Fakultetu_sarasas*
zinute!: *ZINUTES*

---

pre $((\exists k: KLASES \cdot (k\_nr(k) = nr?) \wedge (k\_pastatas(k) = pastatas?)) \vee (\exists k: KLASES \cdot (k\_ID(k) = ID?)) \vee (nauja\_klase \in \mathrm{dom}\, k\_nr))$
$zinute! = klase\_jau\_egzistuoja \wedge changes\_only[]$

Predikate nurodoma, jog egzistuoja klasė tame pačiame pastate su tokiu pat numeriu , todėl jos įtraukti nebegalima.

Schema Klases_Ismetimas_klaida praneša, jog klasės kurią norimą panaikinti nėra.

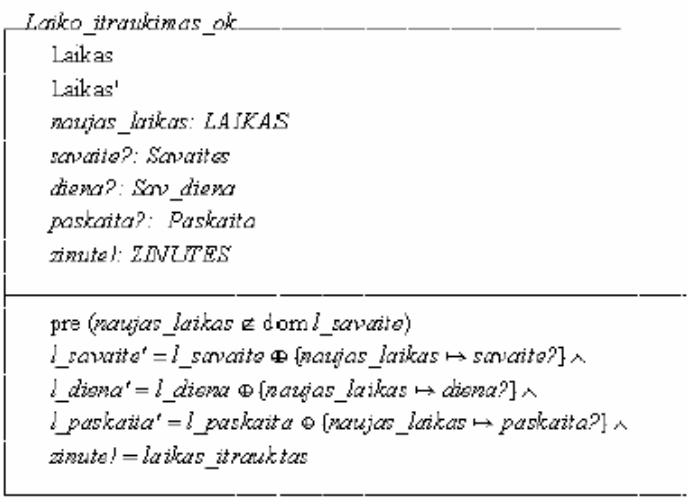
Klases_ismetimas_klaida

state
state'
msg!: *ZINUTES*
klase?: *KLASES*

---

pre $(klase? \notin \mathrm{dom}\, k\_nr)$
$msg! = tokios\_klases\_nera \wedge changes\_only[]$

Predikate nustatoma, jog klasė kurią norima pašalinti neegzistuoja. Pasirodo pranešimas, jog tokios klasės nėra.
msg! = tokios_klases_nera



Užsiėmimų pravedimo laiko koregavimo operacijos:

- Naujo užsiėmimų pravedimo laiko įvedimas, *(Laiko_itraukimas_ok)*
- Klaidingas laiko įvedimas, *(Laiko_itraukimas_klaida)*
- Itraukto užsiėmimų pravedimo laiko panaikinimas, *(Laiko_ismetimas_ok)*
- Klaidingas laiko panaikinimas, *(Laiko_ismetimas_klaida)*

SchemaLaiko_itraukimas_ok įtraukia naują laiką: tam tikrą savaitę, tam tikrą savaites dieną, tam tikrą paskaitą.

Užsiėmimų pravedimams skirtų klasių rezervavimo operacijos:

- Vienos klasės rezervavimas, *(Rezervacija_vieno_ok)*
- Dviejų klasių rezervavimas, *(Rezervacija_dvieju_ok)*
- Trijų klasių rezervavimas, *(Rezervacija_triju_ok)*
- Negalimas vienos klasės rezervavimas, *(Rezervacija_vieno_klaida)*
- Negalimas dviejų klasių rezervavimas, *(Rezervacija_dvieju_klaida)*
- Negalimas trijų klasių rezervavimas, *(Rezervacija_triju_klaida)*
- Vienos klases rezervavimas kitame fakultete, *(Rezervacija_vieno_ok2)*
- Dviejų klasių rezervavimas kitame fakultete, *(Rezervacija_dvieju_ok2)*
- Trijų klasių rezervavimas kitame fakultete. *(Rezervacija_triju_ok2)*
- Rezervuotos klasės atšaukimas *(Rezervacija_naikinti_ok)*
- Klaidingas rezervuotos klasės atšaukimas, *(Rezervacija_naikinti_klaida)*

Užsiėmimų rezervavimo taisyklės

Modulį aptarnaujančioje klasėje turi būti įdiegtos visos tam moduliui reikalingos programos.

I ir II kurso moduliai turi pirmenybę rezervuoti klases Informacinių technologijų plėtros institute.

Specialybiniai (III, IV ir vyresniųjų kursų) moduliai turi pirmenybę rezervuoti klases fakulteto kompiuterių klasėse.

Pageidaujama kompiuterių klasė yra rezervuojama, jei modulį studijuojančių studentų kiekis yra didesnis ne daugiau nei dviem vienetais už laisvų darbo vietų skaičių kompiuterių klasėse.

Schema Rezervacija_vieno_ok rezervuoja vieną klasę tam tikram moduliui tam tikru laiku.

```
Rezervacija_vieno_ak1
state
state'
rezervacija: REZERVACIJA
modulis?: MODULIAI
laikas?: PASK_LAIKAS
klase?: KLASES
msg!: ZINUTES
---
pre ((modulis? ∈ dom m_pavadinimas) ∧ (
laikas? ∈ dom l_savaite) ∧ (
klase? ∈ dom k_nr) ∧ (
m_studentu_kiekis(modulis?) < k_kompiuteriu_kiekis(klase?) + 2) ∧ (((
 m_tipas(modulis?) = bendrasis) ∧ (
        k_pastatas(klase?) = Skaiciavimo_centras)) ∨ ((
    m_tipas(modulis?) ≠ bendrasis) ∧ (
        k_fakultetas(klase?) = m_fakultetas(modulis?)))) ∧ (
rezervacija ∉ dom r_laikas) ∧
      ¬ (∃ r: REZERVACIJA • (r_klase(r) = klase?) ∧ (r_laikas(r) = laikas?))) ∧ (
r_klase' = r_klase ⊕ {rezervacija ↦ klase?}) ∧ (
r_modulis' = r_modulis ⊕ {rezervacija ↦ modulis?}) ∧ (
r_laikas' = r_laikas ⊕ {rezervacija ↦ laikas?}) ∧
msg! = klase_rezervuota ∧
   changes_only[r_laikas, r_modulis, r_klase]
```

Predikate nurodoma:

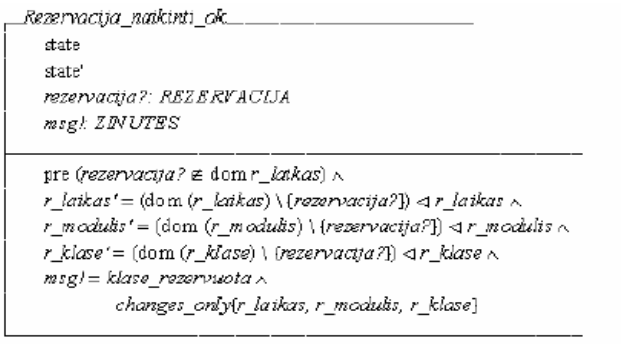
- Turi egzistuoti modulis, kuriam norime užrezervuoti klasę,(1 pred. eilutė)
- Turi egzistuoti klasė, kurioje norima pravesti užsiėmimą,(3 pred. eilutė)
- Turi egzistuoti toks laikas, kuriuo norima pravesti užsiėmimą,(2 pred. eilutė)
- Klasėje kompiuterių kiekis turi būti ne mažesnis, nei modulį studijuojančių studentų kiekis + 2. (4 pred. eilutė)
- Jei modulis yra bendrasis, tai klasę bandoma rezervuoti Skaičiavimo Centro kompiuteriu klasese (5, 6 pred. eilutė)
- Jei modulis specialybinis, tai klasę bandoma rezervuoti fakulteto kompiuteriu klasėse.(7,8 pred. eilutė).
- Schema rezervacija_vieno_klaida1 praneša, kad užsiėmimui pravesti pageidaujama klasė (klasių) yra užimta.

```
Rezervacija_vieno_klaida1
state
state'
rezervacija: REZERVACIJA
modulis?: MODULIAI
laikas?: PASK_LAIKAS
klase?: KLASES
msg!: ZINUTES
---
pre ((modulis? ∉ dom m_pavadinimas) ∨ (
laikas? ∉ dom l_savaite) ∨ (
klase? ∉ dom k_nr) ∨ (
m_studentu_kiekis(modulis?) > k_kompiuteriu_kiekis(klase?) + 2) ∨ ((
 m_tipas(modulis?) = bendrasis) ∧ (
        k_pastatas(klase?) ≠ Skaiciavimo_centras) ∨ ((
    m_tipas(modulis?) ≠ bendrasis) ∧ (
        k_fakultetas(klase?) ≠ m_fakultetas(modulis?)))) ∨ (
rezervacija ∈ dom r_laikas) ∨ (
∃ r: REZERVACIJA • (r_klase(r) = klase?) ∧ (r_laikas(r) = laikas?))) ∧
msg! = rezervacijai_klasiu_nera ∧ changes_only[]
```

Predikate (10 pred. eil) nustatoma, kad pageidaujama klasė nurodytu laiku yra užimta. Taip pat klasė nebus rezervuota, jei bus neteisingai nurodytas modulis, klasė ar laikas ar studentų kiekis viršys kompiuterių kiekį pridejus du:

m_studentu_kiekis(*modulis?*) > *k_kompiuteriu_kiekis*(*klase?*) + *2*)

SchemaRezervacija_naikinti_ok atlaisvina rezervuotą klasę tam tikru metu.



4. Išvados

Z specifikavimo metodas leido formalizuoti kompiuterių klasių užimtumą, aprašant studijų modulius, klases, užsiėmimų pravedimo laikus ir pan.

Formalus aprašymas sudaro prielaidas mokymo proceso organizavimo kompiuteriu klasėse analizei, tam sukuriant informacinę sistemą.

Literaturos sąrašas

[1] **Spivey J. M**. (1998) The Z Notation: A Reference Manual. *Oriel College, Oxford. OX1 4 EW, England.*

[2] **Woodcock J., Davies J.** (1996) Using Z: Specification, *Refinement and Proof. Prentice Fall*.

[3] **The Cogito Group.** (1996) The Sum Reference Manual v1.3, Software Verification Research Centre, School of Information Technology, The University of Queensland, QLD 4072.

Summary

Formal Specification of Computer Class

A formal specification of computer class load is presented in the paper. The specification describes class resources, teaching courses, and possible times for course delivery and reservation grid for courses. The created formal specification permits to analyse whether all courses, which are foreseen for teaching in computer classes, can be delivered in desirable classes; or course delivery is spreading through several classes that are located in different facilites.

KOMPIUTERIŲ KLASIŲ UŽIMTUMO MODELIS

H. Pranevičius[*], D. Ambrazienė[**], A. Paulauskaitė[*]
Verslo informatikos katedra[*], Informacinių technologijų plėtros institutas[**]
Kauno technologijos universitetas

Straipsnyje pateikiamas kompiuterinių klasių užimtumo modelis, kuris leidžia spręsti šiuos uždavinius: interaktyviai sudaryti užsiėmimų kompiuterių klasėse tvarkaraštį, t.y. centralizuoti mokymo proceso organizavimą kompiuterių klasėse, sukaupti informaciją apie dėstomus kompiuterių klasėse modulius; naudojamą programinę įrangą studijų procese; analizuoti kompiuterių klasių užimtumą. Šio modelio pagrindu sukurta informacinė sistema, kuri leido atlikti kompiuterinių klasių užimtumo eksperimentinius tyrimus.

1. Įvadas

Šiuolaikinės informacinės technologijos įtakoja Kauno Technologijos universiteto studijų procesą. Mokymo proceso kokybės užtikrinimas numato taikomųjų paketų ir projektavimo priemonių, procesų modeliavimo priemonių, dalykinių žaidimų, žinių globaliam kompiuterių tinkle, mokymosi medžiagos elektroninėje formoje naudojimą. Studijų procesas vyksta fakultetų ir Informacinių technologijų plėtros instituto kompiuterinėse klasėse. Universitete įrengta apie 50 kompiuterių klasių, netoli 800 darbo vietų, kurias prižiūri Informacinių technologijų plėtros instituto darbuotojai. Klasių kompiuteriuose įdiegtos 5 operacinės sistemos, kelios dešimtys klasikinių taikomųjų programų paketų, tokių kaip: MS Office, Adobe Photoshop, AutoCad, CorelDRAW, MatLab, eilė servisinių programų bei specializuota programinė įranga, būdinga atskiriems fakultetams bei juose dėstomiems moduliams. Informacinių technologijų plėtros institutas skatina, teikia paramą ir sudaro sąlygas universiteto darbuotojams užtikrinti mokymo proceso kokybę, naudojant anksčiau minėtas priemones. Šiandien studijų proceso aptarnavimas kompiuterių klasėse plečiasi ir darosi vis sudėtingesnis. Kompiuterių skaičius klasėse labai įvairus, kompiuterių specifikacija plataus diapazono. Kai kurie nauji studijų moduliai susiję su programinės įrangos diegimu, kuri kelia papildomus reikalavimus techninei įrangai. Specializuotų programinių paketų licenzijų trūkumas neleidžia užsiėmimų metu pilnai panaudoti visų darbo vietų kompiuterių klasėse. Su informacinių technologijų panaudojimu persipina net senųjų tradicinių modulių (matematika, braižyba, anglų k) dėstymas. Minėti moduliai dėstomi dideliam studentų skaičiui, tai ženkliai padidina užsiėmimų poreikį kompiuterių klasėse. Iškyla problemos ne tik derinant tvarkaraštį, bet užtikrinant studijų modulių dėstymo skirtingose kompiuterinėse klasėse vienodas galimybes. Didėjantis informacinių technologijų naudojimas universiteto studijų procese reikalauja užtikrinti reikalingus techninius ir programinius resursus užsiėmimų pravedimui, efektyviai naudoti turimus resursus, tolygiai ir pagrįstai vykdyti plėtrą.

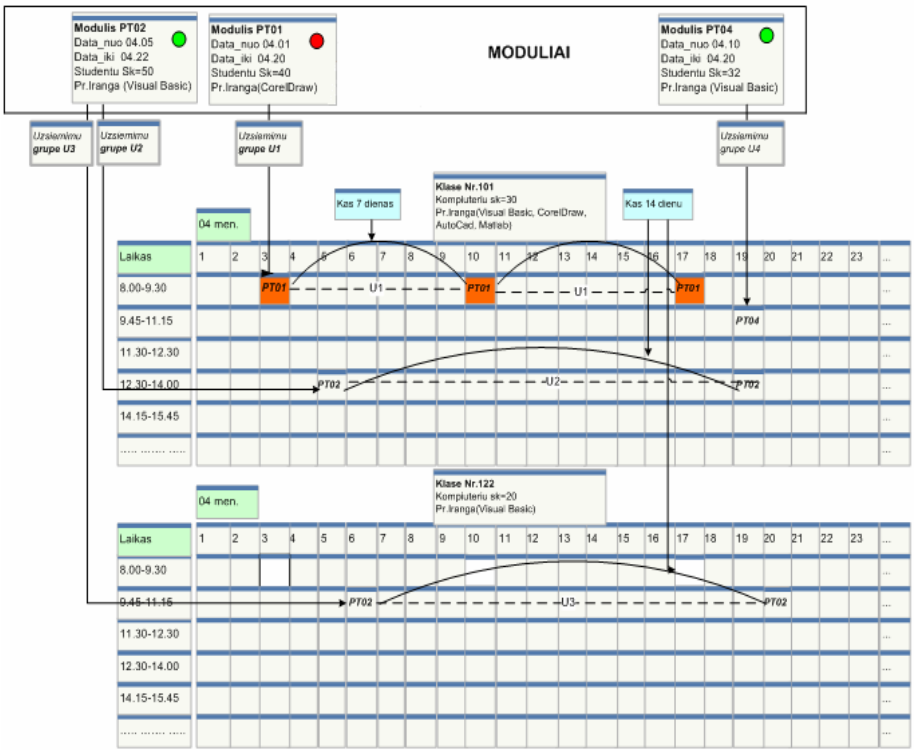
Šiuo metu studijų procesas kompiuterių klasėse organizuojamas, remiantis daugiamete dirbančiojo personalo patirtimi. Užsiėmimų tvarkaraščiai sudaromi tik konkretaus fakulteto kompiuterių klasėms, nedisponuojama informacija apie galimybę pravesti užsiėmimus kitose kompiuterinėse klasėse. Tvarkaraščiai sudaromi neatsižvelgiant į programinę įrangą suinstaliuotą kompiuterių klasėse, neįvertinamas turimų licenzijų kiekis. Iškyla problemos aptarnaujančiam personalui, paruošiant kompiuterines klases užsiėmimų pravedimui. Universitete nėra sukaupta informacijos, kokie studijų moduliai mokymo procese naudoja informacines technologijas ir kokie taikomųjų programų paketai reikalingi. Planuojant naujų kompiuterių klasių steigimą fakultetuose, darosi aktualu išanalizuoti jau esamų kompiuterinių klasių užimtumo lygį.

2. Kompiuterinių klasių formalizavimas

Straipsnyje pateikiamas kompiuterinių klasių užimtumo modelis, kuris leidžia spręsti šiuos uždavinius:

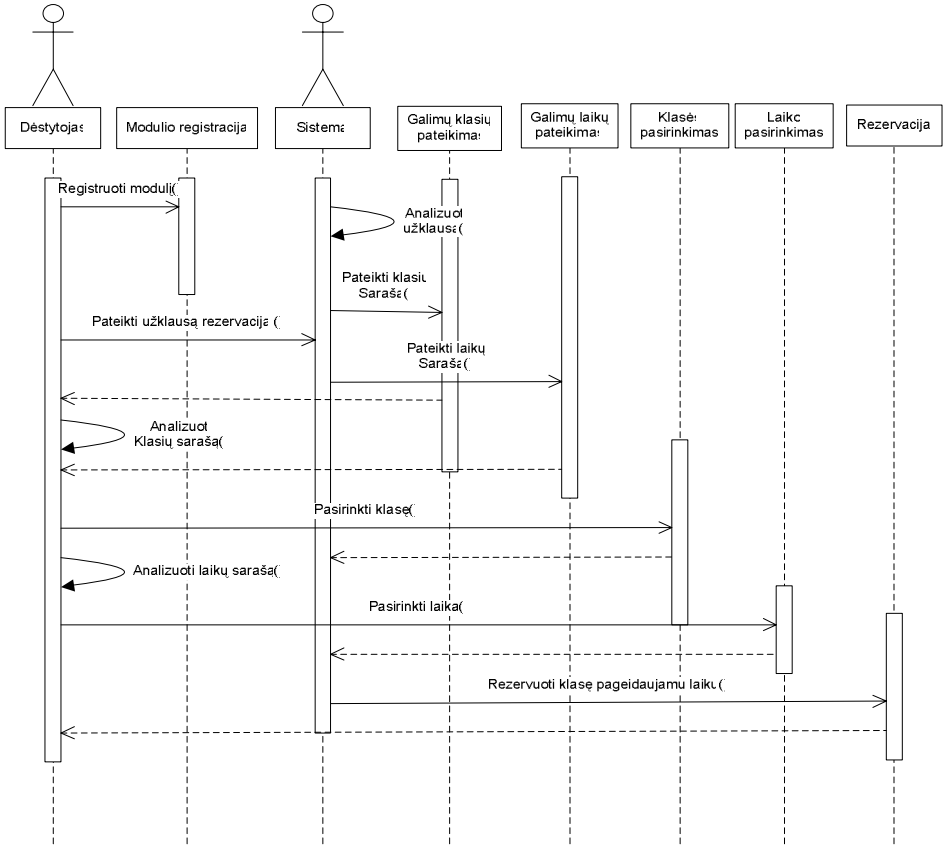
- interaktyviai sudaryti užsiėmimų kompiuterių klasėse tvarkaraštį, t.y. centralizuoti mokymo proceso organizavimą kompiuterių klasėse,
- sukaupti informaciją apie dėstomus kompiuterių klasėse modulius; naudojamą programinę įrangą studijų procese;
- analizuoti kompiuterių klasių užimtumą.

Kuriant modelį buvo nagrinėtos rezervavimo sistemos [10, 11, 12]. Ypač daug rezervavimo sistemų skirtų skrydžių lėktuvu bilietų rezervavimui, pardavimui bei viešbučio kambarių rezervavimui. Pagrindinis dalykas į ką buvo atkreiptas dėmesys, kad visos rezervavimo sistemos komunikuoja su vartotojais, pateikdamos įvairias pasirinkimo galimybes. Kurdami kompiuterinių klasių užimtumo modelį panaudojome rezervavimo sistemų principus, konkrečiam studijų moduliui, kurio dėstyme naudojamos informacinės technologijos, rezervuojamas užsiėmimų pravedimui laikas kompiuterių klasėje (1 pav.), išanalizavus situaciją toje klasėje (ji laisva ar užimta, ar yra reikalinga programinė įranga, ar tenkina darbo vietų skaičius). Tai interaktyvi rezervavimo sistema, suteikianti galimybę užsiėmimų rezervaciją panaikinti, pakeisti ar papildyti. Pasirinktas modelis leidžia kaupti ir analizuoti informaciją dinaminiame mokymo procese, sparčiai keičiantis dėstomiems dalykams kompiuterių klasėse, jų turiniui bei studentų skaičiui. Be to, sukaupta informacija apie mokymo procesą vykstantį kompiuterių klasėse, sudaro galimybes analizuoti ar pakanka programinių ir techninių resursų užsiėmimų pravedimui, ar tikslinga vykdyti kompiuterinių klasių plėtrą.



1 pav. Užsiėmimų rezervavimas.

Daugelio duomenų apdorojimo bei informacijos sistemų, kurių pagrindas darbas su duomenimis, projektavimo metodikos susideda iš eilės žingsnių, pradedant uždavinių koncepcijos išsiaiškinimu, baigiant formaliu procesų aprašymu ir duomenų bazės schemos aprašu. Apibrėžti ir vizualizuoti kompiuterių klasių užimtumo modelį buvo panaudota unifikuota modeliavimo kalba UML [7, 8, 9] (2 pav.). Modelis leido ne tik suprastinti sistemos struktūrą, bet ir jos funkcionalumą.



2 pav. Sekų diagrama.

Aprašant kompiuterinių klasių užimtumo modelį buvo panaudoti Z formalūs metodai. Z specifikavimo kalba yra naudojama kompiuterių sistemų aprašymui ir modeliavimui. Specifikacija parašyta Z kalboje yra mišinys formalių

matematinių sakinių ir neformalaus paaiškinamojo teksto. Abu yra svarbūs. Formalioji dalis duoda aiškų specifikuojamas sistemos apibrėžimą, savo ruožtu neformalus tekstas dokumentą daro lengviau skaitomą, geriau suprantamą. Formalūs metodai leido aprašyti klases, modulius, programinę įrangą, operacijas, įvertinti galimybę pravesti užsiėmimus konkrečioje klasėje, konkrečiu laiku, suformuoti užklausas apie kompiuterių klasių užimtumą; modulius dėstomus konkrečiose
kompiuterių klasėse ir programinę įrangą naudojamą mokymo procese. Z formalizavimo metodas pasirinktas dėl jo orientacijos į abstrakčių duomenų struktūrų aprašymą, kas leido adekvačiai aprašyti mokymo procesą kompiuterių klasėse.

Straipsnyje pateikiamas formalus mokymo proceso kompiuterių klasėse aprašymas, naudojant Z kalbos notaciją [1, 2, 4, 5, 6]. Z schemų aprašymui panaudotos standartinės programinės priemonės Cogito [3].

Formalūs metodai leido apibrėžti šiuos reikalavimus kuriamai sistemai:

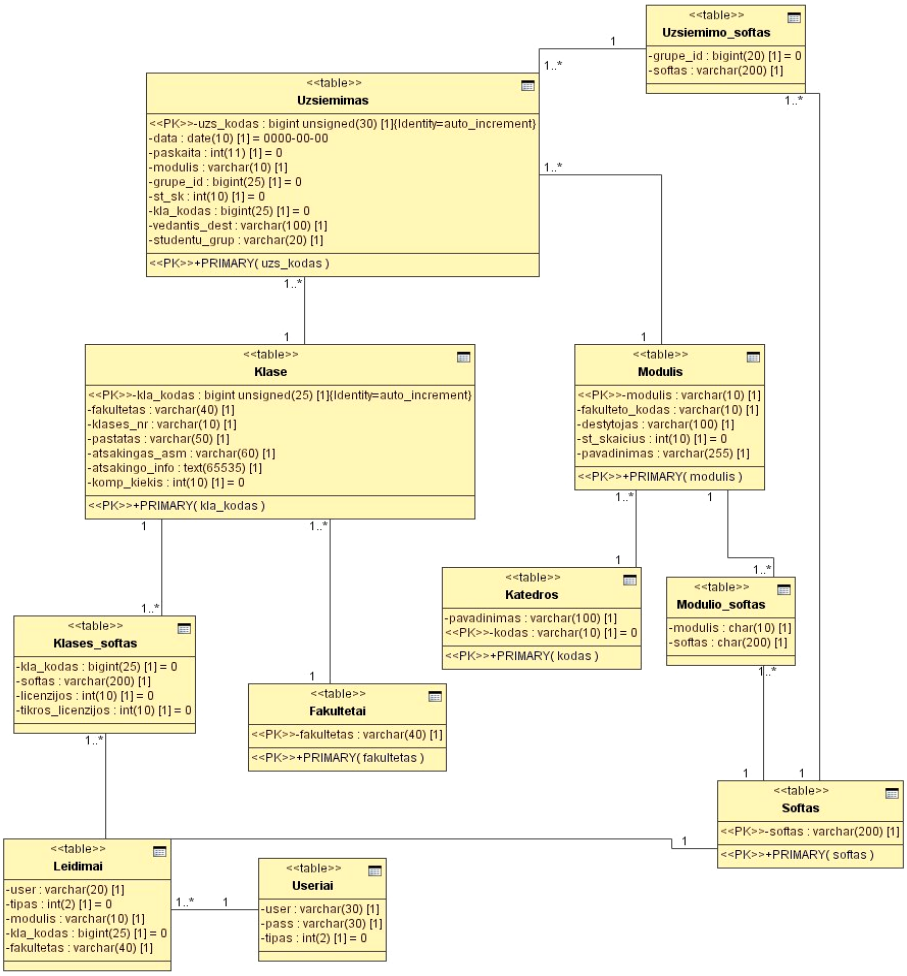
- Kiekvienas modulis gali būti vedamas tik tokioje kompiuterių klasėje, kurioje yra įdiegta tam moduliui reikalinga programinė įranga;
- Kiekvienam moduliui rezervuojamų klasių užsiėmimų datos patenka į intervalą, nusakomą modulio pravedimo pradžios data ir modulio pravedimo pabaigos data;
- Rezervuojant klasę studentų skaičius neturi viršyti darbo vietų skaičiaus kompiuterių klasėje;
- Prieš rezervuojant klasę turi būti nurodoma, kokiu būdu bus rezervuojamos datos užsiėmimų pravedimui ar cikliniu, nurodant pirmą datą ir kitos bus rezervuojamos automatiškai kas 7(14) dienų, ar nurodant konkrečią datą.

Naudojant formalius metodus sistemoje apibrėžtos tokios operacijos: sukurti ir panaikinti užsiėmimo rezervavimą kompiuterių klasėje, įtraukti naujus ir panaikinti jau esamus modulius, įtraukti naujas ir pašalinti jau įtrauktas klases, įtraukti naują programinę įranga ir panaikinti esamą, rezervuoti klasę bet kokiam pageidaujamam laikotarpiui (3 pav.). Rezervuojant užsiėmimą pirmiausia reikia pasirinkti klasę ir pageidaujamą užsiėmimo pravedimo datą bei laiką, nurodyti modulį. Patikrinama ar egzistuoja toks modulis, ar toje klasėje yra programinė įranga reikalinga tam moduliui. Klasės negalima rezervuoti, jei modulį studijuojančių studentų kiekis kompiuterių klasėje viršyja darbo vietų kiekį. Jei nurodytu laiku klasė yra laisva, rezervacija laikoma sėkminga.

3 pav. Užsiėmimo rezervavimo operacija

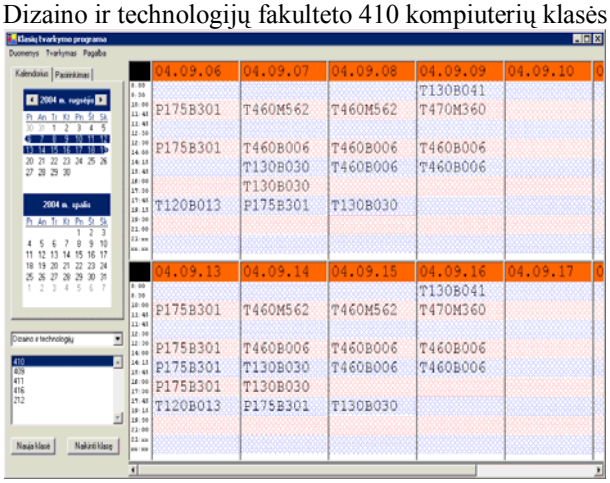
3. Kompiuterinių klasių užimtumo informacinės sistemos realizacija

Pagal Z specifikavimo kalba aprašytą modelį buvo sukurta informacinė sistema. Naudojama 3 lygių architektūra: duomenų bazė,. serveris, klientas. Duomenų bazė sukurta naudojant MySQL 4.1. Kliento lygmuo buvo aprašytas panaudojant C# programavimo kalbą, serveris aprašytas panaudojant C++ programavimo kalbą. Kaip formaliais metodais aprašytoje informacinėje sistemoje, taip ir duomenų bazėje yra aprašomos klasės, moduliai, programinė įranga, užsiėmimai. Žemiau pateikiama duomenų bazės klasių diagrama (4 pav .)

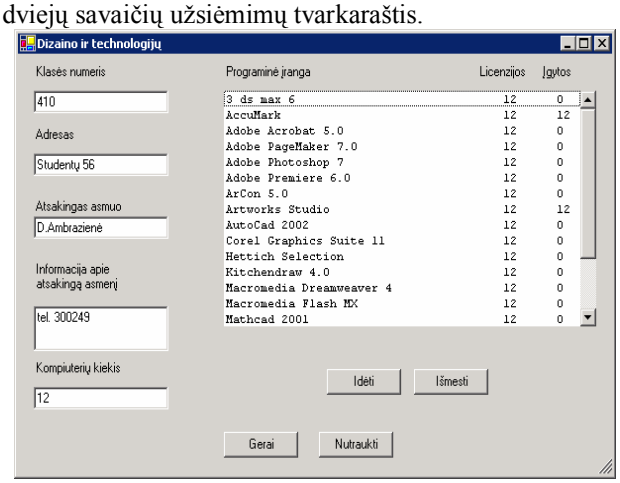


4 pav. Duomenų bazės klasių diagrama.

Vartotojas prisijungęs prie duomenų bazės gali tiek peržiūrėti jį dominančios klasės tvarkaraštį (5 pav.) pasirinktu laiko periodu, tiek pasirinkus modulį rezervuoti vieną ar kelias neužimtas kompiuterių klases užsiėmimų pravedimui, gauti informaciją apie modulį, kiek paskaitų pravesta per pasirinktą laikotarpį konkrečiose klasėse. Paveikslėlyje pateiktas Dizaino ir technologijų fakulteto 410 kompiuterių klasės dviejų savaičių užsiėmimų tvarkaraštis.



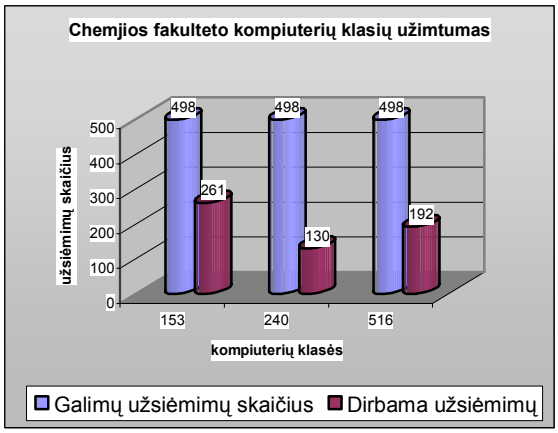
5 pav. 410 kompiuterių klasės užsiėmimų tvarkaraštis.



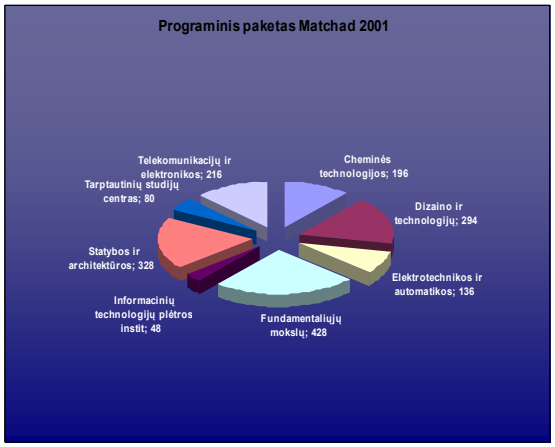
6 pav. Duomenys apie 410 kompiuterių klasę .

Duomenų bazėje kaupiama informacija apie kompiuterių klases (6 pav.). Paveikslėlyje pateikti duomenys apie Dizaino ir technologijų fakulteto 409 kompiuterių klasę, joje instaliuotą programinę įrangą, turimas licenzijas, kompiuterių skaičių bei informaciją apie atsakingą asmenį.

Naudojant užklausas, gautus duomenis galima užsaugoti „xml“ formatu, kuris suteikia galimybę atlikti statistinę duomenų analizę. Kompiuterių klasių analizės užklausa leidžia pateikti statistiką apie fakulteto kompiuterių klases (7 pav.).



7 pav. Chemijos fakulteto kompiuterių klasių užimtumas.



8 pav. Matchad 2001 naudojimas fakultetų kompiuterių klasėse.

Programinės įrangos panaudojimo užklausa leidžia analizuoti programinių paketų panaudojimo mokymo procese statistiką (8 pav.). Paveikslėlyje pateikiamas kompiuterių klasėse įdiegto programinio paketo Matchad panaudojimas mokymo proceso metu vieno semestro laikotarpyje. Grafike duomenys pateikti akademinėmis valandomis.

Informacinės sistemos programinę realizaciją atliko informatikos fakulteto III kurso studentai V.Šeinauskas ir N.Ambrazas. Straipsnio autoriai yra nuoširdžiai jiems dėkingi.

4. Išvados

- Z specifikavimo metodas leido formalizuoti kompiuterių klasių užimtumą, aprašant studijų modulius, klases, užsiėmimų pravedimo laiką ir pan.
- Formalus aprašymas leido suformuluoti reikalavimus kompiuterių klasių užimtumo informacinei sistemai.

Literatūros sąrašas

[1] **B.Potter, J. Sinclair and D.Till**. An introduction to formal Specification and Z, Second Edition. *Prentice Hall Europe* .1996.
[2] **J.C.P Woodcock, J.Davies**. Using Z: Specification, Proof and Refinement. Prentice Hall International Series in Computer Science, London. 1996.
[3] **The Cogito Group.** The Sum Reference Manual v1.3. Software Verification Research Centre, School of Information Technology, The University of Queensland, QLD 4072.
[4] **J.P. BOWEN.** Formal Specification and Documentation Using Z : A Case Study Approach. International Thomson Computer press, 1996. Prieiga per internetą: <http://www.zuser.org/zbook/pub/indfoils.pdf >
[5] http://www.jpbowen.com/pub/ssm-z.pdf
[6] http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/ElectronicSupplements/ModelSpec.pdf
[7] **M. Fowler, K. Scott** . UML Distilled: A Brief Guide to the Standard Object Modeling Language. 1999
[8] http://www.uml.org/
[9] http://www.smartdraw.com/tutorials/software-uml/uml.htm
[10] http://www.hotelslithuania.net/lhrs/cgi/myres.
[11] https://www.amadeus.net/pl/balticclipper/en/avail_only.htm
[12] http://www.cs.pdx.edu/~walpole/papers/rtss2002.pdf

Summary
Occupation model of computer classes
An occupation model of computer classes is presented in the paper. It allows to solve the following tasks: to create interactively a schedule of practical works at computer classes, i.e., to centralize an organization of a study process at the computer classes; to accumulate information about delivered courses at the computer classes and software used in the study process; to analyze an occupation of the computer classes. An information system has been developed based on this model. The system has allowed performing experimental investigations on the computer class occupation.

Z FORMALIŲJŲ SPECIFIKACIJŲ PANAUDOJIMAS KURIANT INFORMACINES SISTEMAS

H. Pranevičius, A. Paulauskaitė

Verslo informatikos katedra, Kauno technologijos universitetas

Šiame straipsnyje yra siūloma, panaudojant Z specifikavimo metodiką sukurti formalų modelį, kuris nusakytų reikalavimus kuriamai informacinei sistemai. Aptarti Z specifikacijos transformavimo į Object-Z specifikaciją klausimai, kurie palengvina objektiškai orientuotų specifikacijų praplėtimą į objektiškai orientuotas programavimo kalbas. Todėl transformaciją į objektiškai orientuotą programavimo kalbą yra lengviau atlikti iš Object-Z kalboje parašytos specifikacijos. Darbe yra siūloma Object-Z specifikacijos transformavimo metodika į objektiškai orientuotą programavimo kalbą C++ ir SQL DDL sakinius.

1.Įvadas

Iki šiol labiau paplitę neformalieji informacinių sistemų projektavimo metodai, kurie neleidžia vienareikšmiškai suprasti formuluojamus uždavinius. Be to gaunamos specifikacijos ne visada būna išbaigtos. Tokiu būdu sukurtos sistemos neatitinka vartotojo poreikių. Naudojant neformalius metodus, ne visada yra įmanomas specifikacijos transformavimas į programinės įrangos kodą.

Realiame pasaulyje informacinės sistemos probleminė sritis kinta laike. Tokiu būdu keičiasi reikalavimai informacinei sistemai. Naudojant neformaliuosius metodus, pasikeitus reikalavimams, reikia perrašyti programinę įrangą. Tuo tarpu naudojant formaliuosius metodus programinės įrangos perrašyti nereikia , pakanka organizacijos verslo taisykles specifikuotas Z kalboje transformuoti į programinės įrangos kodą.

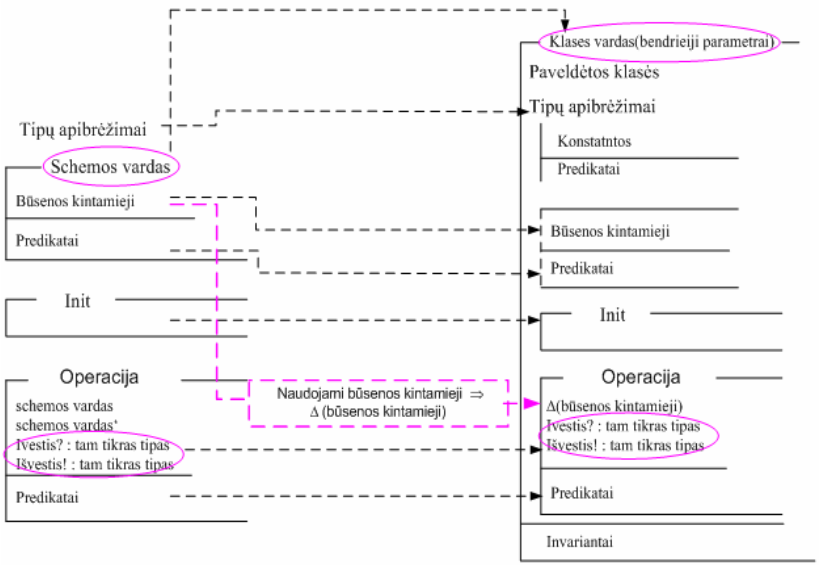
2.Informacinės sistemos formalus reikalavimų aprašymas

Formalieji metodai tampa vis labiau pripažinti tiek mokslo, tiek verslo aplinkoje, kaip vienas iš būdų, kuriuo galima padėti pagerinti tiek programinės įrangos, tiek techninės įrangos sistemų kokybę.

Naudojant formaliąją notaciją didėja supratimas apie sistemos veikimą. Tai padeda projektuotojams ir projektavimas tampa kiek įmanoma aiškesnis ir paprastesnis. Yra įmanoma formaliai mąstyti apie sistemą, formuluojant ir pateikiant teoremas. Jos patikrina ar sistema elgiasi taip kaip tikėjosi projektuotojai. Formalieji metodai padeda projektavimo komandai mąstyti apie sistemos darbą dar prieš jos realizavimą. Trūkstamos, neišbaigtos specifikacijos dalys tampa vis akivaizdesnės. Pagaliau, kas svarbiausia pramonėje, galutinė kaina turi būti kiek įmanoma mažesnė. Kuo anksčiau bus aptiktos klaidos tuo kaina bus mažesnė.

Labiausiai paplitę yra šie formalūs metodai: Petri tinklai, Baigtiniai automatai, B-metodas ir Z. Kiekvieno iš jų panaudojimas priklauso nuo formalizuojamos sistemos tipo. Baigtiniai automatai, Petri tinklai daugiau naudojami tokiose sistemose, kuriose reikia vertinti įvykių sekas vykstančias sistemoje. B-metodas ir Z daugumoje naudojami informacinėms, sudėtingoms ir kritinėms sistemoms aprašyti.

Šiame darbe informacinės sistemos „kompiuterinių klasių užimtumo modelis“ aprašymui buvo panaudota Z specifikavimo kalba[1,2,3,4], kuri yra naudojama kompiuterių sistemų aprašymui ir modeliavimui. Specifikacija parašyta Z kalboje yra mišinys formalių matematinių sakinių ir neformalaus paaiškinamojo teksto. Formalioji dalis duoda aiškų specifikuojamas sistemos apibrėžimą, savo ruožtu neformalus tekstas dokumentą daro lengviau skaitomą, geriau suprantamą. Z formalizavimo metodas pasirinktas dėl jo orientacijos į abstrakčių duomenų struktūrų aprašymą. Z gali būti naudojama įskaitomai bei aiškiai specifikacijai sukurti. Z lengvai susidoroja su didelėmis specifikacijomis. Kai projektas yra parengtas, galima išdėstyti ir įrodyti sistemos teoremas. Tai padeda verifikuoti projektą patikrinti klaidas.



1 pav. Z specifikacijos transformavimas į Object-Z

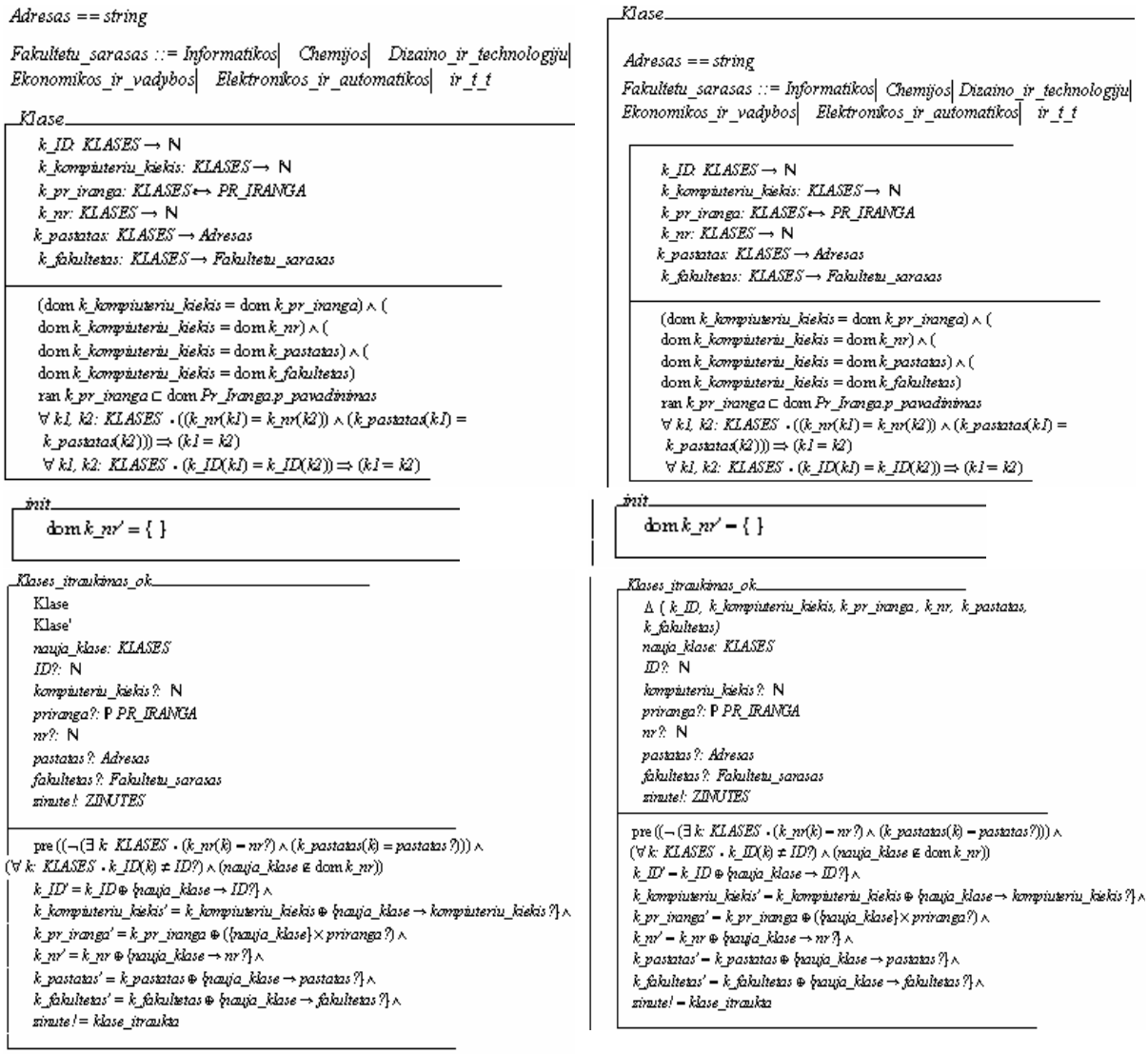
3.Z ir Object-Z palyginimas

Z kalba parašyta specifikacija leido apibrėžti reikalavimus kuriamai informacinei sistemai.

Transformuojant Z specifikaciją į objektiškai orientuotą (C++, C#, JAVA) kalbą yra naudojama tarpinė transformacija į Object-Z kalbą. Todėl transformacijai buvo pasirinkta Object-Z kalba [5,6]. Object-Z kalba visų pirma yra objektiškai orientuota kalba. Ji turi klases, paveldėjimus, tipus, todėl yra lengviau ir patogiau ją transformuoti į objektiškai orientuotą kalbą, t.y C++.

Pereinant nuo Z specifikacijos į Object-Z, reikia įvertinti šių formaliųjų metodų pagrindinius skirtumus. Pirmiausia, Object-Z klasės būsenos schema neturi schemos vardo. Antra, inicializavimo schema skiriasi nuo Z inicializavimo schemos. Trečia Object-Z Δ notacijoje turi būti nurodytas kiekvienas būsenos kintamasis, kuris yra keičiamas, tuo tarpu Z yra nurodomas visos schemos vardas. Z specifikacijos transformavimas į Object-Z kalbą iliustruojamas paveiksle(1 pav.).

Z specifikacija **Object-Z specifikacija**



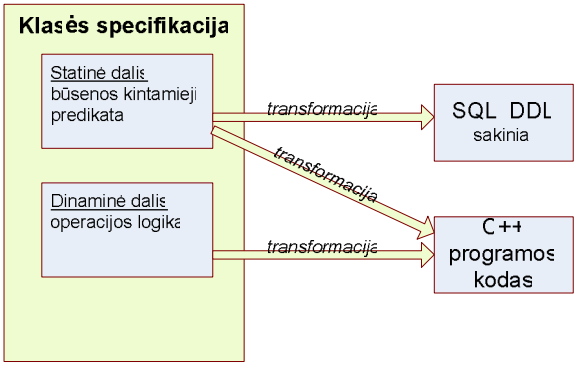
2 pav. Specifikacijos fragmentas parašytas Z ir Object-Z kalbose.

Aukščiau pateiktas (2 pav.) Z kalba parašytos specifikacijos fragmentas , kuriame aprašyta schema „Klase“ ir operacija „Klases_itraukimas_ok“. Dešinėje pusėje specifikacija parašyta Object-Z kalba. Z schema ir operacija transformuojama į Object-Z klasę „Klasė“. Z tipų aprašai transformuojami į Object-Z klasės tipų aprašus. Z specifikacijos schemos vardas „Klasė“ atitinka Object-Z klasės vardą. Po pradinės būsenos tiek Z, tiek Object-Z specifikacijoje aprašoma operacija. Object-Z Δ notacijoje nurodomi visi būsenos kintamieji, kurie yra keičiami, Z specifikacijoje yra nurodomas visos schemos vardas „Klasė“.

4.Object-Z specifikacijos transformavimas į programinės įrangos kodą

„Kompiuterių klasių užimtumo modelio“ informacinės sistemos duomenų struktūros aprašytos SQL DDL sakiniais, valdymo logika aprašyta C++ programavimo kalba. Todėl Object-Z specifikacijos transformavimas į programinės įrangos kodą (3 pav.) atliekamas 2 etapais :

- Specifikacijos statinės ir dinaminės dalies transformavimas į C++.
- Specifikacijos statinės dalies transformavimas į SQL;



3. pav. Object-Z specifikacijos transformavimas į programinės įrangos kodą

Šiame darbe yra siūloma Object-Z transformavimo į C++ programavimo kalbą metodika. Žemiau (1 lentelė) pateiktos pagrindinės taisyklės Object-Z specifikacijos transformavimui į C++ programavimo kalbą.

Object-Z		C++
Klasės schema „Klasės vardas“	⟼	*Vieša (public) klasė „Klasės vardas*
Paveldėjimai	⟼	*Vieši (public)* paveldėjimai
Konstantos	⟼	Konstantos
Atributai	⟼	Klasės atributai
Būsenos schemos aksiomos	⟼	*Invariantai*
Pradinė schema	⟼	Klasės konstruktorius
Operacijos schema	⟼	Metodo aprašas
Δ-sąrašas		*Modifikacijų sąrašas*
Būsenos – „iki “		*Būsena –„iki „*
aksiomos		*Būsena - „po “*
Matomumo(visibility) sąrašas	⟼	„vieši“prefiksai

Atributai (konstantos ir kintamieji) aprašyti klasės schemoje yra transformuojami į C++ klasės atributus. Visos Object-Z klasės schemos konstantos transformuojamos į C++ klasės konstantas. Object-Z būsenos schemos aksiomos yra transformuojamos į C++ klasės invariantus. Pradinė Object-Z schema atitinka C++ klasės konstruktorių.

1 lentelė. Pagrindinės transformavimo taisyklės iš Object-Z į C++ .

Operacijos schemos transformuojamos į C++ keliais būdais. *Δ*- sąrašas transformuojamas į *„modifikacijų“* konstruktą, būsenos – *„iki“* aksiomos atitinka būsenas – *„iki“*, ir likusios operacijos schemos aksiomos transformuojamos į metodo būsenas – *„po“*(4 pav .).

Object-Z

Klase

$k\_ID: KLASES \rightarrow \mathbb{N}$
$k\_kompiuteriu\_kiekis: KLASES \rightarrow \mathbb{N}$
$k\_pr\_iranga: KLASES \leftrightarrow PR\_IRANGA$
$k\_nr: KLASES \rightarrow \mathbb{N}$
$k\_pastatas: KLASES \rightarrow Adresas$
$k\_fakultetas: KLASES \rightarrow Fakultetu\_sarasas$

$(\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_pr\_iranga) \wedge (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_nr) \wedge (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_pastatas) \wedge (\mathrm{dom}\ k\_kompiuteriu\_kiekis = \mathrm{dom}\ k\_fakultetas)$
$\mathrm{ran}\ k\_pr\_iranga \subset \mathrm{dom}\ Pr\_Iranga.p\_pavadinimas$
$\forall k1, k2: KLASES \bullet ((k\_nr(k1) = k\_nr(k2)) \wedge (k\_pastatas(k1) = k\_pastatas(k2))) \Rightarrow (k1 = k2)$
$\forall k1, k2: KLASES \bullet (k\_ID(k1) = k\_ID(k2)) \Rightarrow (k1 = k2)$

init

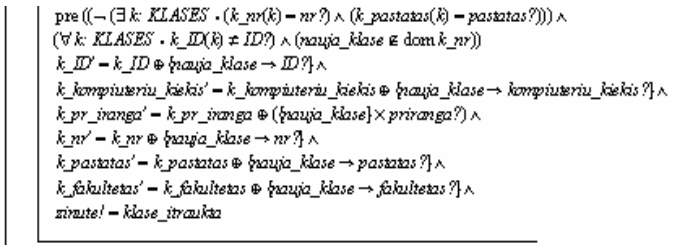
$\mathrm{dom}\ k\_nr' = \{\ \}$

Klases_itraukimas_ok

Δ (*klases id, k_kompiuteriu_kiekis, k_pr_iranga, k_nr, k_pastatas, k_fakultetas*)
naujo_klase: KLASES
ID?: N
kompiuteriu_kiekis?: N
priranga?: P PR_IRANGA
nr?: N
pastatas?: Adresas
fakultetas?: Fakultetu_sarasas
zinute!: ZINUTES

C++

```
public class Klase{
  private int    *  k_ID;
  private int    *  k_kompiuteriu_kiekis;
  private Pr_iranga   *  k_pr_iranga[40];
  private int    *  k_nr;
  private char *  k_pastatas[50];
  private char *  k_fakultetas[50];
  //invariant:   unique (k_nr, k_pastatas)

public Klase( )
  { k_nr =0; )

 // modifies:   (k_ID, k_kompiuteriu_kiekis,
                 k_pr_iranga, k_nr, k_pastatas,
                 k_fakultetas);

  //pre:   unique (k_nr, k_pastatas);
           unique (k_ID);

  //post:  k_ID[×××××] =ID;
           k_kompiuteriu_kiekis[×××××]=
           k_kompiuteriu_kiekis;
           k_pr_iranga[×××××] =k_pr_iranga;
           k_nr[×××××] = nr;
           k_pastatas[×××××] =k_pastatas;
           k_fakultetas[×××××] =k_fakultetas;
  public void klases_itraukimas_ok()
  {.....}
}
```



4 pav. Object-Z specifikacijos transformavimo į C++ programavimo kalbą pavyzdys.

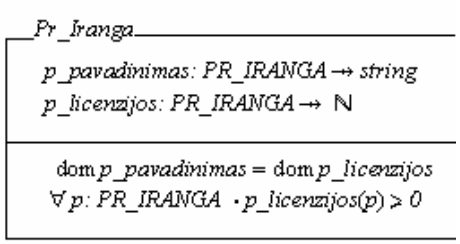
Visi Object-Z klasės būsenos kintamieji ir predikatai transformuojami į SQL DDL sakinius (pav 5.). Klasės schemos vardas atitinka SQL lentelės vardą, atributai transformuojami į lentelės stulpelius, invariantai į suvaržymus (2 lentelė.)

2 lentelė. Pagrindinės transformavimo taisyklės iš Object-Z į SQL DDL

Object-Z		SQL
Klasės schema „Klasės vardas“	⟼	Lentelės vardas
Atributai	⟼	Lentelės stulpeliai
Invariantai	⟼	Suvaržymai (CONSTRAINTS)

Žemiau pateiktas Object-Z specifikacijos framentas ir jo transformavimas į SQL DDL sakinius(pav 5.).

Object-Z



SQL

```
CREATE TABLE `Pr_Iranga` (
 `id` INTEGER UNSIGNED NOT NULL
AUTO_INCREMENT,
 `p_pavadinimas` VARCHAR(45) NOT NULL,
 `p_licencijos` INTEGER UNSIGNED NOT NULL,
 PRIMARY KEY(`id`)  )
```

5 pav. Object-Z transformavimas į SQL DDL sakinius.

5. Išvados

Atlikti tyrimai parodė, kad Z specifikavimo metodu formalizavus sistemą bei transformavus šią specifikaciją į Object-Z, susidaro galimybės sistemos objektinį aprašą transformuoti į C++ programavimo kalbos kodą ir SQL DDL sakinius. Tokia realizacija įmanoma dėl to, kad tiek Object-Z, tiek C++programavimo kalba yra sukurtos objektinio modelio pagrindu.

Literatūros sąrašas

[1] **B.Potter, J. Sinclair and D.Till**. An introduction to formal Specification and Z, Second Edition. *Prentice Hall Europe* .1996.

[2] **J.C.P Woodcock, J.Davies**. Using Z: Specification, Proof and Refinement. *Prentice Hall International Series in Computer Science*, *London*. 1996.

[3] **The Cogito Group.** The Sum Reference Manual v1.3. *Software Verification Research Centre, School of Information Technology, The University of Queensland,* QLD 4072.

[4] http://vl.zuser.org/

[5] **Greame Smith**. Reasoning about Object-Z specifications. Software *Verification Research Centre, Department of Computer Science, University of Queensland, Australia*. 1995

[6] **Greame Smith**. A logic fort Object-Z (additional rules). Software *Verification Research Centre, Department of ience, University of Queensland, Australia*. 1995

Summary

Development of Information Systems using Z formal specifications

In this paper we suggest create formal method by using Z specification methodology, wich would specify requirements for information system. This paper discusses questions about specification transformation from Z to Object-Z. Object-Z is extention to the formal specification language Z, wich facilities specification in an object-oriented style. Therefore it is easier transform specification written in Object –Z to object-oriented programming language. So primarily we need to transform Z specification to Object-Z. In this paper transformation methodology from object-Z to object-oriented programing languages C++ and SQL DDL statements is suggested.