KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

LINAS SALELIONIS

DUOMENŲ INŽINERIJOS UŽDAVINIAI

MAGISTRO TEZĖS

VADOVAS

DOC. B. PARADAUSKAS

RECENZENTAS

DOC. V. KIAULEIKIS

KAUNAS, 1998

Turinys

Sutrumpinimų sąrašas

ADT	-	Abstraktus duomenų tipas
DB	-	Duomenų bazė
DBVS	-	Duomenų bazių valdymo sistema
ES	-	Esybės-sąryšiai
FP	-	Funkcinė priklausomybė
IESD	-	Išplėstinė esybių-sąryšių diagrama
IP	-	Išskyrimo priklausomybė
IS	-	Informacijos sistema
MDB	-	Meta duomenų bazė
OO	-	Objektinė orientacija
OODB	-	Objektinės orientacijos duomenų bazė
OOTP	-	Objektinės orientacijos taikomoji programa
PIRS	-	Pilnoji išplėstinė ryšių schema
PP	-	Poaibio priklausomybė
RDB	-	Reliacinė duomenų bazė
RDBVS	-	Reliacinė duomenų bazių valdymo sistema
RS	-	Reliacinė schema
RTP	-	Reliacinė taikomoji programa
SA	-	Sudėtinis atributas
SFP	-	Sudėtinė funkcinė priklausomybė
SI	-	Sudėtinis identifikatorius
UA	-	Unarinis atributas
UI	-	Unarinis identifikatorius
VS	-	Vidinis sąryšis

Referatas

Išnagrinėjus tiesioginės ir atvirkštinės duomenų inžinerijos tikslus, pasiūlyta duomenų schemų transformacijas atlikti trimis etapais: atributų reagregavimas, *is_a* sąryšių transformavimas ir abstrakčių duomenų tipų elgsenos metodų prijungimas.

Pasiūlyta atvirkštinės inžinerijos procesuose panaudoti poaibių, funkcines, sudėtines funkcines ir *is_a* sąryšių priklausomybes; šios priklausomybės atvirkštinės inžinerijos procesuose kaupiamos ir atvaizduojamos semantiniame duomenų modelyje. *Is_a* sąryšių išvedimui iš sudėtinių funkcinių ir poaibių priklausomybių sudarytos išvedimo taisyklės.

Sudarytas grafinis interfeisas objektinei schemai specifikuoti. Schemos grafiniame redaktoriuje numatytas rėžimas objektinės duomenų bazės aprašo generavimui.

Abstract

Analyzed goals of forward and reverse data engineering and proposed to perform the transformations of data schemes through three stages: an attribute reaggregation, *is_a* relationships transformations and adding the methods to abstract data types scheme.

Proposed to use inclusion, functional, compound functional and *is_a* relationships dependences in the processes of the reverse engineering; these dependencies are collected in the processes of the reverse engineering and expressed in the semantic data model. There are created rules for generation of *is_a* relationships from compound functional and inclusion dependences.

There is made the graphic interface to specify an object-oriented scheme and there is provided possibility for generation of the object-oriented database description in the graphic editor.

1. Duomenų inžinerijos pagrindiniai uždaviniai ir darbo tikslas

Esybių-sąryšių (ES) požiūris, kurį pasiūlė Chen [18] 1976 metais, į duomenų modeliavimą yra paprasčiausias duomenų atvaizdavimo būdas, aprašant juos esybėmis ir sąryšiais tarp tų esybių. Esybės, jų atributai ir sąryšiai tarp jų sudaro esybių-sąryšių modelį. Toks modelis duomenų bazių projektavimo technologijose yra naudojamas jau daugiau negu 20 metų.

Objektinės orientacijos (OO) požiūris į duomenų modeliavimą yra santykinai naujas požiūris. Objektinės orientacijos požiūryje yra akcentuojamas panašių duomenų grupavimas ir klasifikavimas, sudarant klases ir poklases. Ši metodologija apima realaus pasaulio objektų atvaizdavimą duomenų objektais. Vidinė objektų struktūra yra sudaryta egzempliorių kintamųjų, kurių reikšmės apibūdina objektą. Metodai yra funkcijos, kurios, keisdamos egzempliorių kintamųjų reikšmes, keičia objektų būsenas. Viršklasės-poklasės sąryšis leidžia poklasėms paveldėti visas viršklasės savybes bei turėti savo specifinių savybių.

Normalizavimas, taikomas ES schemoms, sukelia panašių duomenų suskirstymą į skirtingus esybių tipus ir sukelia sunkumus tam tikrose duomenų struktūrose (pavyzdžiui, atributai, vienu metu galintys turėti keletą reikšmių, sudėtiniai atributai). Objektinės orientacijos požiūris nėra suvaržytas tokiais nepatogumais. Tai sukelia prielaidą ieškoti metodų, kaip ES schemas pervesti į objektinės orientacijos schemas [19].

Informacijos sistemų kūrimo objektinės orientacijos technologijos suteikia keletą privalumų [2]:

- padidintas produktyvumas,
- sumažinta sistemos palaikymo kaina.

Objektiškai orientuotos kalbos palaiko tokias savybes [3], kaip

- duomenų ir funkcijų inkapsuliacija,
- abstraktūs duomenų tipai,
- pranešimų perdavimas,
- paveldėjimas,
- polimorfizmas.

Svarbi objektiškai orientuotų kalbų savybė yra ta, kad jos iš esmės nepalaiko duomenų pastovumo. Dėl šios priežasties yra būtina apjungti OO kalbas su tam tikromis duomenų bazių technologijomis.

Pagrindas apjungiant objektinės orientacijos ir reliacines technologijas yra:

- Reliacinių duomenų bazių išsaugojimas, nes reliacinės duomenų bazių sistemos yra plačiai naudojamos pramoninėse ir komercinėse srityse.
- Reliacinių taikomųjų programų (RTP), t. y. programų, parašytų kokia nors procedūrine kalba, išsaugojimas ir duomenų apdorojimas tradiciniu būdu, tokiu kaip SQL. Tai yra būtina, nes į egzistuojančias reliacines taikomąsias programas, kompanijos yra investavusios nemažai pinigų. Todėl jos nenorėtų išmesti savo duomenų ir programų ir pradėti viską kurti iš pradžių. Dėl šios priežasties, apjungiant OO ir reliacines technologijas, yra būtinas reikalavimas, kad naujai sukurtos informacijos sistemos ir toliau palaikytų reliacines taikomąsias programas.
- Konceptai OO modelyje yra aiškesni už reliacinio modelio konceptus.

Pagal tai, kaip yra kuriamos objektiškai orientuotos informacijos sistemos (IS) iš reliacinių IS, jų kūrimo technologijas galima suskirstyti į dvi grupes:

- atvirkštinė duomenų inžinerija [4],
- objektiškai orientuotų ir reliacinių technologijų apjungimas.

1.1. Atvirkštinė duomenų inžinerija

Daugelis duomenų bazių taikomųjų programų šiais laikais susiduria su perėjimo iš reliacinių duomenų bazių sistemų į išplėstines reliacines, objektines-reliacines ir objektiškai orientuotas duomenų bazių sistemas. Šios problemos sprendimas reikalauja atitinkamų technologijų schemų ir egzempliorių transformacijai. Tokios technologijos bendrai yra vadinamos *atvirkštine duomenų inžinerija*.

Atvirkštinė duomenų inžinerija atlieka abstrakčios, semantiškai prasmingos informacijos išgavimą iš konkrečios duomenų bazės padidinti konceptualų supratimą apie saugomą informaciją [5]. Abstrakcijos proceso tikslo modelis, kuris yra aiškiau išreikštas negu taikomasis modelis sukurtas saugomų duomenų pagrindu.

Šio požiūrio technologijas galima suskirstyti į dvi grupes:

- atvirkštinė duomenų inžinerija, taikoma esybių-sąryšių modelių sukūrimui iš reliacinių duomenų bazių [6],
- atvirkštinė duomenų inžinerija, taikoma objektiškai orientuotų modelių sukūrimui iš ES modelių [7, 8].

Daugelis atvirkštinės inžinerijos technologijų yra skirtos transformacijoms, kada šaltiniu yra naudojama reliacinė schema, o tikslas yra ES modelis [5, 14, 21, 22, 24, 25, 26, 27, 28, 29] ir tik nedaugelis tikslu laiko objektiškai orientuotą modelį [23, 30]. Kai kuriuose požiūriuose, pavyzdžiui Chiang ir kt. [14], yra aprašyta metodologija, kaip išgauti ES schemas iš reliacinių DB, tačiau analizuojant duomenų bazės egzempliorius ir iš jų išgaunant poaibio priklausomybes (PP), yra kreipiamas dėmesys tiktai į PP, kurių pagrindas yra raktai. Painios sąryšių struktūros, tokios, kaip skirtingų paveldėjimo sąryšių struktūrų identifikavimas, arba painios sąryšių struktūros, kurios negali būti aprašytos tiesiogiai per raktais besiremiančios PP, nėra tyrimų objektas. Priešingai Chiang požiūriui, Castellanos ir kt. [30] požiūryje iš duotos duomenų bazės egzempliorių yra išgaunamos ir poaibio priklausomybės, kurios nesiremia raktais. Tokios PP yra interpretuojamos kaip "paslėptos" esybės ir atvaizduojamos per virtualius sąryšius. Tačiau, kaip ir Chiang [14], šis požiūris labai priklauso nuo duotos duomenų bazės egzempliorių kiekio ir kokybės.

Beveik visiems atvirkštinės inžinerijos požiūriams yra būdinga tai, kad jie yra apriboti tam tikrų būtinų sąlygų, kurias galima sugrupuoti sekančiai:

- visos konceptualios specifikacijos turi būti transliuojamos į duomenų struktūras ir apribojimus, pavyzdžiui, reliacinės duomenų bazės atveju viso poaibio priklausomybės ir pirminiai raktai turi būti aiškiai ir tiksliai apibrėžti;
- pagal tam tikras taisykles parinkti vardai (pavyzdžiui, išorinio rakto ir rakto, į kurį tas išorinis raktas turi nuorodą, vardai turi sutapti);
- šaltinio schema negali turėti „painių" objektų ir sąryšių struktūrų atvaizdavimų;
- laikoma, kad schema nebuvo optimizuota ar blogai suprojektuota.

Be to, aprašytuose požiūriuose šaltinio schemos semantinis papildymas yra išskirtinai atliekamas analizuojant turimas duomenų bazės struktūras. Kadangi šių struktūrų prasmė gali būti nevienareikšmė, tai požiūrius galima taikyti tik tada, kai schema yra gauta naudojant griežtą projektavimo metodą. Nei vienas požiūris nenagrinėja kokybės kriterijaus ar pageidaujamų savybių tokių, kaip, ar visi sąryšiai yra identifikuoti, ar kiekviena schemos struktūra atstovauja nedaugiau kaip vieną realaus pasaulio abstrakciją, nei iš šaltinio, nei iš tikslo schemos.

Tačiau, išnagrinėjus kai kuriuos požiūrius, pasidaro aišku, kad yra įmanoma atsisakyti reliacinių technologijų ir vietoje jų naudoti objektiškai orientuotas duomenų bazes [9], bei

pasinaudojant atvirkštine duomenų inžinerija egzistuojančius senos duomenų bazės duomenis pernešti į naująją sistemą. Vis dėlto čia iškyla mažiausiai dvi problemos:

- Egzistuojančios reliacinės taikomosios programos daugiau nebepalaikomos ir turi būti perrašytos kokiu nors objektiškai orientuotu būdu.
- Dauguma egzistuojančių komercinių objektiškai orientuotų duomenų bazių (OODB) nėra pakankamai stabilios. Naujai išleistose versijose būna esminių pakeitimų lyginant su ankstesnėmis versijomis ir todėl sunku numatyti, kurios sistemos išliks ateityje.

Dėl šių priežasčių kompanijoms, daug investavusioms į reliacines technologijas, šios grupės technologijos netinka dėl finansinių ir sunaudoto laiko priežasčių.

1.2. Objektiškai orientuotų ir reliacinių technologijų apjungimas

Daugumos šios grupės technologijų pagrindas yra sukurti iš anksto apibrėžtą objektinių klasių, kurios paslėptų prisijungimą prie reliacinių duomenų bazių (RDB), apvalkalą. Šios objektinės klasės suformuoja tokias reliacines savybes, kaip lentelės, įrašai ir laukai. Kadangi OO modeliai yra išraiškingesni negu reliaciniai, objektų dekompozicija į įrašus ir atvirkščiai turi būti sukurta rankiniu būdu taikomųjų programų programuotojų. To priežastis yra tai, kad normalizuotose RDB gali būti saugomos tik tokios atominės reikšmės, kaip skaičiai, simbolinės eilutės ir kt. Objektiškai orientuotose aplinkose gali būti saugomi ir sudėtiniai objektai. Tai objektai, kurių atributų reikšmėmis gali būti ne tik atominės reikšmės, bet ir nuorodos į kitus objektus. Tačiau, šalia šių šio požiūrio trūkumų, yra ir didelis privalumas: egzistuojančios reliacinės taikomosios programos gali būti panaudotos kartu su egzistuojančiais reliaciniais duomenimis.

Pagrindinis požiūriai apjungiant objektiškai orientuotas (ar išplėstines esybių-sąryšių) ir reliacines technologijas yra sekantys:

- tiesioginė duomenų inžinerija,
- atvirkštinė duomenų inžinerija.

Iš esmės, atvirkštinės inžinerijos proceso tikslas yra sukurti duotos duomenų bazės konceptualinį aprašą, jį išreiškiant semantiniu duomenų modeliu, kur įėjimo duomenis galėtų sudaryti šaltinio kodo aprašymas, duomenų žodynas, taikomosios programos, duomenų bazės egzemplioriai arba bet kokia išvardintų konceptų kombinacija. Žemiau yra pateikta keletas požiūrių, kaip atlikti tokią atvirkštinę inžineriją reliacinės duomenų bazės schemai, remiantis tuo, kad atvirkštinė inžinerija iš tikrųjų yra transformacijos problema. Priešingai negu klasikinės schemų transformacijos atliekamos tiesioginėje inžinerijoje, pavyzdžiui, Išplėstinių esybių-sąryšių diagramų (IESD) ar OO schemų transformacijos, atvirkštinė inžinerija yra sunkiai automatizuojama. Štai keletas to priežasčių:

- Informacija apie šaltinio schemos struktūrą ir semantiką dažnai nėra tiksliai pateikta schemos aprašyme, o tik numanoma iš taikomosios programos, iš kurios šią informaciją yra sunku išgauti automatiniu būdu.
- Šaltinio schemos struktūra gali būti optimizuota dėl efektyvesnio taikomųjų programų darbo. Taigi gali būti sunku išsiaiškinti objektų ir sąryšių struktūras bei jas teisingai interpretuoti.
- Šaltinio reliacinio modelio modeliavimo konstruktai gali būti semantiškai persidengę (pavyzdžiui, reliacinio modelio sąryšiai, kurie gali būti aprašyti arba kaip savybės, arba kaip jų sąryšiai). Todėl šaltinio schemos struktūros dažnai gali būti interpretuojamos ir transformuojamos į tikslo objektinį modelį daugiau negu vienu būdu.
- Šaltinio reliacinė schema gali būti suprojektuota ir realizuota naudojant keletą (kartais netgi nepanašių) projektavimo metodologijų. Taigi ekvivalenčios sąryšių struktūros,

atstovaujančias ekvivalenčius arba panašias realaus pasaulio abstrakcijas, gali būti realizuotos skirtingais būdais.

Kaip to pasekmė, atvirkštinė inžinerija dažnai reikalauja kai kurių duotos reliacinės schemos semantinių papildymų, kurių metu yra kaupiama informacija apie šaltinio reliacinės duomenų bazės struktūrą ir semantiką. Ypatingas dėmesys semantiškai papildant duotą duomenų bazę turi būti atkreipiamas į:

- Srities semantikos [20], nevisiškai aiškiai ir tiksliai duotos schemos aprašyme atstatymas. Tai apima raktus, funkcines priklausomybes (FP), poaibio priklausomybes (PP) ir išskyrimo priklausomybes (IP), esančias šaltinio reliacinėje schemoje.
- Toks schemos papildymas, kaip objektų identifikatoriai ir sąryšiai tarp objektų bei duotų struktūrų interpretavimas.
- Optimizavimo struktūrų ir painių sąryšių atvaizdavimų panaikinimas.
- Schemos transformavimas į koncepcinio modelio kalbą, tam, kad gauti konceptualų duomenų bazės aprašymą.

1.3. Complex Object Manager (COMan)

Prie pirmosios grupės technologijų (atvirkštinė duomenų inžinerija) galima priskirti *Complex Object Manager* (COMan) [1]. Pastarasis apjungia objektiškai orientuotas ir reliacines technologijas [10]. Pagrindinės COMan savybės yra:

- *Egzistuojančių reliacinių duomenų reinžinerija objektiškai orientuotoje programinėje aplinkoje*. COMan leidžia objektiškai orientuotoms taikomosioms programoms (OOTP) naudotis jau egzistuojančiais reliaciniais duomenimis. Labai svarbu yra tai, kad COMan panaikina aukščiau paminėtą trūkumą, t. y. rankiniu būdu sukuriamus dekompozicijos ir kompozicijos procesus, išsaugodamas informaciją apie objektiškai orientuotą schemą, reliacinę schemą ir perėjimus tarp šių schemų meta duomenų bazėje (MDB). Taigi programos vykdymo metu dekompozicijos ir kompozicijos procesai yra automatiškai atliekami pačios sistemos. Kaip būtina tam sąlyga, meta duomenų bazės informacija yra sukuriama interaktyviai bendraujant su vartotoju atvirkštinės inžinerijos proceso metu.
- *Jau egzistuojančių reliacinių taikomųjų programų panaudojimas.* Kadangi reliacinės schemos semantiniai papildymai yra saugomi meta duomenų bazėje, pati reliacinė schema išlieka nepakitusi. Todėl egzistuojančios reliacinės taikomosios programos vis dar gali būti naudojamos, netgi lygiagrečiai naujoms OO taikomosioms programoms.
- *Lengvas perėjimas nuo OO taikomųjų programų be objektų pastovumo prie OO taikomųjų programų su objektų pastovumu.* COMan leidžia generuoti reliacines schemas atskirai nuo OOTP. Reliacinės schemos yra naudojamos saugoti sudėtiniams objektams programos vykdymo metu. Kadangi programos vykdymo metu programuotojui nereikia dekomponuoti objektus į įrašus ir atvirkščiai, tai tik keletas funkcijų, tokių, kaip sudėtinių objektų saugojimas ir išgavimas, turi būti įdėta į taikomąją programą.
- *Lengvas perėjimas tarp įvairių reliacinių duomenų bazių valdymo sistemų (DBVS).* COMan, o taip pat ir OOTP, naudojančios COMan, gali lengvai pereiti nuo vienų reliacinių DBVS prie kitų, nes nuo DBVS specifinis programinis kodas yra paslėptas ir saugomas specialiose COMan objektinėse klasėse.

COMan pagrindinis elementas yra meta duomenų bazė. Ji yra naudojama, iš vienos pusės, sudėtinių objektų dekompozicijai į atomines lenteles ir, iš kitos pusės, sudėtinių objektų generavimui iš atominių duomenų. O tai reiškia, kad MDB yra naudojama kiekvienoje duomenų bazės operacijoje, atliekamoje vartotojo.

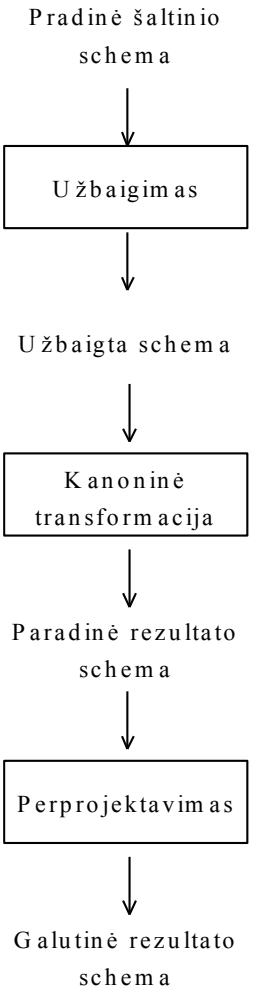
Remiantis šiomis savybėmis, galima tvirtinti, kad COMan pateikia reinžinerijos technologiją, kuri duoda didelį pagrindą išsaugoti egzistuojančias reliacines duomenų bazes.

Tačiau, žiūrint iš kitos pusės, jei jau yra sukurtų OO taikomųjų programų, kurios naudoja jau egzistuojančias OO duomenų bazes, tai COMan netinka dėl tos priežasties, kad duomenis saugo reliacinėse DB ir nesukuria OODB. Todėl reikėtų technologijų, kurios vis dėlto reliacines duomenų bazes leistų konvertuoti į objektiškai orientuotas.

1.4. Modulinis požiūris į reliacinę atvirkštinę inžineriją

Kitas požiūris į atvirkštinę reinžineriją yra *Modulinis požiūris į reliacinę atvirkštinę inžineriją* [4]. Iš esmės, jis beveik neturi reikalavimų šaltinio duomenų bazei ir gali garantuoti pageidaujamas rezultato savybes. Šiame požiūryje atvirkštinė inžinerija atliekama trimis etapais (pav. 1):

- užbaigimas,
- kanoninė transformacija,
- perprojektavimas.



1 pav. Trijų etapų požiūris į atvirkštinę inžineriją

Pagrindinė papildymo etapo idėja yra sukurti „idealius“ duomenis transformacijai iš pradinės reliacinės schemos, kurie vėliau galėtų būti kanoniškai transformuoti į tam tikrą tikslo

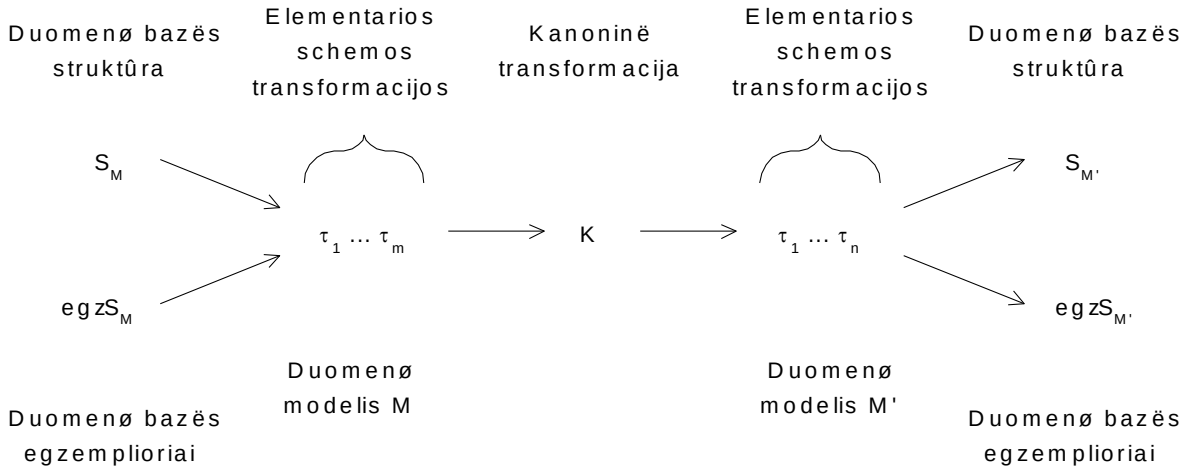
modelio schemą. Šiame etape įvedamas naujas reliacinės schemos kriterijus *pilnumas*, kuris formaliai apima reliacinės schemos savybes, kurias ji turi turėti norint gauti teisingą, minimalų ir primityvų aukštos semantinės abstrakcijos rezultatą. Pilnumas susideda iš dviejų dalių:

- struktūrinis pilnumas,
- semantinis pilnumas.

Struktūrinio pilnumo pagrindas yra *atributų ekvivalentiškumas* ir jis apima situaciją, kai ir visi sąryšiai, ir visos objektų aibės yra identifikuotos poaibio priklausomybėmis. *Semantinis pilnumas* reikalauja, kad schemos struktūros (sąryšiai, raktai ir poaibio priklausomybės) būtų semantiškai interpretuojamos modeliuojamos sistemos kontekste. Be to, yra reikalaujama, kad kiekviena reliacinė schema atstovautų nedaugiau kaip vieną realaus pasaulio abstrakciją, o taip pat daugiau nebeliktų perteklinių schemos struktūrų. Iš esmės, semantinis pilnumas apibrėžia tikslumo, minimalumo ir primityvumo kriterijus [5, 11] reliaciniam modeliui. Pilnumas yra nepriklausomas nuo specifinio tikslo modelio. Jis yra apibrėžiamas išreiškiant jį bendromis abstrakcijos sąvokomis, kaip agregavimas, apibendrinimas ir klasifikavimas, ir gali būti lengvai pritaikomas specifiniam duomenų modeliui.

Kai šaltinio schema yra užbaigta, transformacija į specifinį tikslo modelį gali būti atliekama kanoniškai, t. y. tokiu tiesioginiu būdu, kai struktūros iš šaltinio schemos yra pernešamos į tikslo modelį. Šiuo atveju iš reliacinio šaltinio yra sukuriamas rezultatas, kuris vis dar "atrodo" reliacinis. Tokiu būdu, specifinis tikslo perprojektavimo etapas yra atliekamas tam, kad pradinė rezultato schema būtų restruktūrizuota pagal tikslo modelio projektavimo principus.

Schemų transformacijos pirmame (užbaigimas) ir trečiame (perprojektavimas) etapuose yra atliekamas, naudojant taip vadinamas *elementarias schemų transformacijas*. Elementarios transformacijos yra schemų modifikavimo operacijos, kurios negali būti išreikštos suskaldžius kitas elementarias schemų transformacijas. Tai reiškia, kad pradinės šaltinio schemos transformacija į galutinę tikslo schemą gali būti išreikšta kaip elementarių schemos transformacijų seka ir kanoninė transformacija (pav. 2).



2 pav. Elementarių transformacijų vaidmuo

Kiekviena elementari schemos transformacija apibrėžia ir DB egzempliorių, ir DB struktūrų pervedimus.

Tačiau modulinis požiūris į reliacinę atvirkštinę inžineriją neleidžia sukurtų objektiškai orientuotų schemų vėliau panaudoti reliaciniuose modeliuose.

1.5. Atvirkštinės duomenų inžinerijos kokybės kriterijai

Geros kokybės duomenų bazės pateikimas turi būti laikomas pirminiu atvirkštinės inžinerijos tikslu. Šiame skyriuje yra atskiriamos schemų ar duomenų modeliavimo kokybės nuo transformacijų kokybių. Schemų ir duomenų modeliavimo kokybės kriterijai [5, 11, 12] dažniausiai remiasi intuityviomis sąvokomis, kaip skaitomumas, aiškumas ar išraiškingumas, ir dažniausiai yra svarbūs tik koncepciniame duomenų bazių projektavime. Kriterijai, būdingi atvirkštinei inžinerijai, gali būti sugrupuoti sekančiai:

- *Teisingumas* [5]: Schema yra *sintaksiškai teisinga*, jei visi schemoje esantys konceptai yra tinkamai apibrėžti. Schema yra *semantiškai teisinga*, jei visi jos konceptai yra naudojami pagal jų apibrėžimą, pavyzdžiui, asociacija nėra modeliuojama kaip apibendrinimas ar atvirkščiai.
- *Primityvumas* [11]: Schema yra *primityvi*, jei visos schemos struktūros atstovauja nedaugiau kaip vieną realaus pasaulio abstrakciją.
- *Minimalumas*: Schema yra *minimali*, jei nei vienas schemos elementas negali būti pašalintas tuo neiššaukiant informacijos praradimo. Schemas, kurios nėra minimalios dažnai yra daug sunkiau suprasti.
- *Relevantiškumas* [5, 11, 12]: Schema yra *relevantinė*, jei visi jos objektai ir sąryšiai yra aiškiai ir tiksliai atvaizduoti duomenų bazės schemoje, panaudojant atitinkamo modelio konceptus.
- *Normališkumas* [5]: Duomenų bazės schemos *normališkumas* yra apibrėžiamas analogiškai reliacinio modelio normališkumui, t. y. jis patenkina tam tikrą specifinę normalinę formą.

Semantinis teisingumas yra tarp svarbiausias kokybės kriterijus atvirkštinėje inžinerijoje, kadangi atvirkštinės inžinerijos tikslas yra pagaminti semantinę abstrakciją, atvaizduotą per šaltinio schemą. Minimalumo ir primityvumo kriterijai yra glaudžiai susiję su semantinės abstrakcijos lygiu, kadangi pertekliškumas ir paslėpti objektai sumažina abstrakcijos lygį. Tinkamumo kriterijus yra svarbus tuo, kad jo dėka visi sąryšiai ir objektai yra sumodeliuojami tiksliai modeliavimo konceptų dėka, o ne tiktai loginės informacijos ar paslėptų sąryšių dėka.

Be struktūros transformacijų, atvirkštinė inžinerija yra susijusi ir su egzempliorių bei taikomųjų programų pervedimu. Dėl to reikia atsižvelgti į formalias transformacijos savybes. Ypač svarbus yra duomenų bazės informacijos turinys, t. y. aibė galimų duomenų bazės egzempliorių, kurie gali būti susieti su duomenų bazės schema. Idealiu atveju informacijos turinys transformacijos eigoje nepakinta. Tačiau taip yra ne visada. Tai galima apibrėžti sekančiai:

Sakykim, kad M ir M' yra du (nebūtinai skirtingi) duomenų modeliai, o S_M ir $S_{M'}$ yra galimos duomenų bazių schemų aibės (pav. 2). Schemai $S \in S_M$ ($S \in S_{M'}$), DB_S ($DB_{S'}$) pažymėkime kaip visų galimų egzempliorių aibę iš S (S') (informacijos turinys). Ir tegul egz_S ($egz_{S'}$) būna tam tikros duomenų bazės egzempliorius. Tada *schemos transformacija* yra pora $\iota = (T, t)$ [13] susidedanti struktūros transformacijos T:

$$T: S_M \to S_{M'},\ T: S \to S' = T(S)$$

ir egzempliorių pervedimo t:

$$t: DB_S \to DB_{S'},\ t: egz_S \to egz_{S'} = t(egz_S).$$

Remiantis šiuo apibrėžimu, transformacijos gali būti suklasifikuotos pagal šaltinio ir tikslo schemų informacijos turinius [13]:

- *Informaciją išlaikančios* transformacijos yra tokios, kurios nepakeičia duotos duomenų bazės informacijos turinį. Formaliai, $\iota = (T, t)$ informaciją išlaikanti, jei t yra bendrasis arba bijektyvus pervedimas.
- *Informaciją keičiančios* transformacijos gali būti toliau klasifikuojamos sekančiai:
    1) *Turinį didinančios* transformacijos, po kurių rezultato schemos informacijos turinys yra didesnis už pradinės schemos informacijos turinį, pavyzdžiui, egzempliorių pervedimas yra pilnasis ir injekcinis, bet ne surjekcinis.
    2) *Turinį mažinančios* transformacijos, po kurių rezultato schemos informacijos turinys yra mažesnis už pradinės schemos informacijos turinį, pavyzdžiui, egzempliorių pervedimas yra pilnasis ir surjekcinis, bet ne injekcinis.
    3) *Turinį keičiančios* transformacijos (kurios nei didina, nei mažina informacijos turinį), po kurių pasikeičia informacijos turinys, bet tai įvyksta ne pagal (1) arba (2) kategorijas.

Informaciją išlaikančios ir informaciją didinančios transformacijos turi pageidaujamas savybes, kad atlikus transformaciją iš šaltinio modelio į tikslo modelį, abiejų modelių užklausos duotų tą patį rezultatą. Todėl atvirkštinės inžinerijos transformacijos turi tenkinti šias sąlygas. Kadangi atvirkštinė inžinerija yra elementarių transformacijų seka, tai reiškia, kad kiekviena elementari transformacija turi būti informaciją išlaikanti ir informaciją didinanti.

Be kokybės, susijusios su informacijos turiniu, reinžinerijos metodai turi būti palyginami ir pagal sekančius aspektus:

- *Transformacijos prielaidos*: Praktiniame panaudojime reinžinerijos požiūriai neturėtų užduoti prielaidas ar sąlygas duomenų bazei atsižvelgiant į ką nors. Iš kitos pusės, į visą aiškiai apibrėžtą informaciją, susijusią su duomenų baze, kaip pavyzdžiui, integruotumo apribojimai ar programos, turi būti atsižvelgta.
- *Rezultato kokybė*: Reinžinerijos proceso rezultato schema turi būti aukštos kokybės, kuri yra apibūdinta aukščiau.
- *Aiškumas ir suprantamumas*: Transformacija arba reinžinerijos procesas turi būti lengvai suprantamas vartotojui. Tai ypatingai svarbu vartotojo taikomųjų programų kūrimui ir keitimui.
- *Automatizacijos lygis*: Transformacijos metodas turi būti aukšto automatizacijos lygio. Tai priklauso nuo euristikos ir sprendimų taisyklių, kurios yra naudojamos atvirkštinėje inžinerijoje. Kuo daugiau jų naudojama, tuo automatizacijos lygis yra aukštesnis. Iš kitos pusės, euristikos panaudojimas gali iššaukti blogą rezultato schemos kokybę.
- *Išplėtimo galimybės*: Metodas turi būti išplečiamas tuo požiūriu, kad lengvai būtų galima įdėti, pakeisti ar pašalinti euristikas ir sprendimų taisykles. Be to, metodas turi būti pritaikytas vartotojų reikalavimų pasikeitimams.

Šie aspektai dažnai yra nesirūpinama atvirkštinėje inžinerijoje. Automatizacijos lygis yra tiriamas tik keliuose požiūriuose [14].

1.6. Mišrios duomenų inžinerijos technologija ir darbo tikslas

Kadangi nei vienas iš ištirtų metodų neleido sukurti tokio transformacijų požiūrio, kuris leistų reliacinę duomenų bazės schemą pilnai transformuoti į objektiškai orientuotą, išsaugant informaciją apie pradinę schemą, kad vėliau būtų galima atlikti atvirkštinę transformaciją, tai savo tezėse siūlau dar vieną požiūrį.

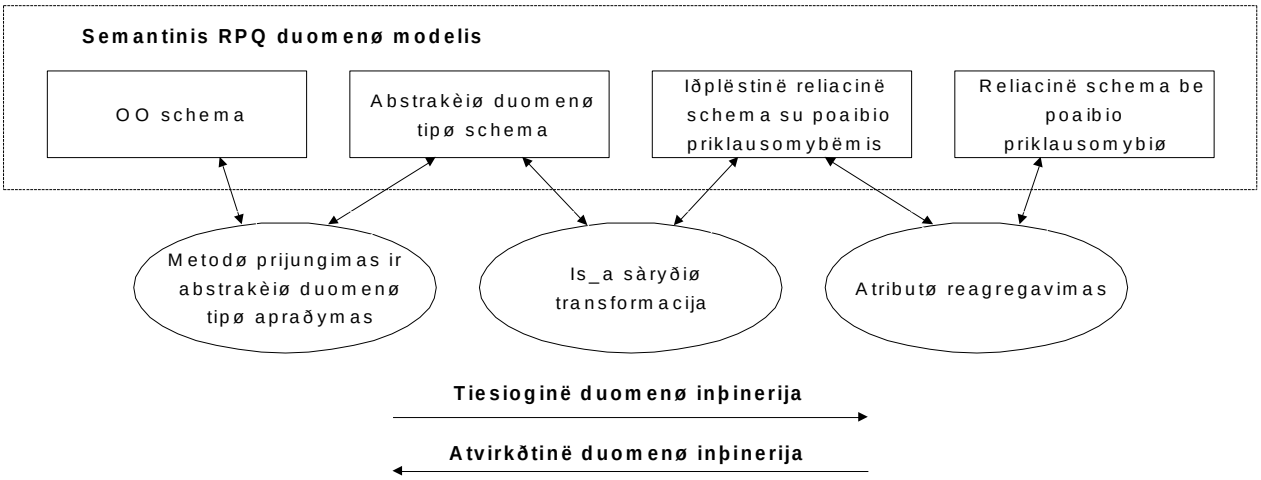
Kaip jau buvo anksčiau minėta, atvirkštinės duomenų inžinerijos proceso metu yra kaupiama informacija apie šaltinio reliacinės duomenų bazės struktūrą ir semantiką. Pateiktame požiūryje ši informacija yra kaupiama sąryšių-savybių-objektų (RPQ) modelyje. Šios schemos

aprašymo kalba semantikos prasme artima objektinės orientacijos kalbai, kurios prototipu galima nurodyti SQL3 kalbą [17]. SQL3 kalba atžvilgiu SQL turi abstrakčių duomenų tipų praplėtimą.

Šiame darbe pateiktas požiūris palaiko kai kurias objektinės orientacijos ir reliacinių technologijų kombinacijas, tokias kaip atvirkštinė inžinerija egzistuojančių reliacinių duomenų pervedimui į objektiškai orientuos duomenis ir pastovumo išgavimas egzistuojančioms objektiškai orientuotoms taikomosioms programoms.

Tam, kad išlaikyti pasiūlymus kaip galima bendresnius, mes sukursime trijų žingsnių schemą apdorojant duomenų schemas (pav. 3):

- Atributų reagregavimas,
- *Is_a* sąryšių transformavimas,
- Metodų prijungimas ir abstrakčių duomenų tipų aprašymas.



3 pav. Bendra OO schemos transformacijos į reliacinę schemą ir atvirkščiai analizavimo schema

Darbo tikslas:

1. Parinkti duomenų tiesioginės ir atvirkštinės inžinerijos mišrios technologijos principus, numatančius objektines duomenų schemas transformuoti į tradicines reliacines ir, atvirkščiai, reliacines schemas transformuoti į objektines.
2. Parinkti duomenų priklausomybes, kurių pagrindu būtų galima atlikti tiesiogines ir atvirkštines transformacijas.
3. Sudaryti *is_a* sąryšių išvedimo taisykles iš priklausomybių, specifikuojamų išplėstinėse reliacinėse schemose.
4. Sudaryti grafinį redaktorių duomenų priklausomybių specifikavimui.

2. Atributų reagregavimo procesas

Pagrindinė atributų reagregavimo idėja yra sukurti *Pilnąją išplėstinę ryšių schemą* (PIRS).

Pagrindas analizuoti duotą reliacinę schemą yra tai, kad du sąryšiai yra susiję tiktai tada, kai jie turi bendrų atributų su identiškomis prasmėmis. Todėl yra kritiškai svarbu teisingai identifikuoti sinonimus ir homonimus. Tai formaliai atliekama per *sudėtines funkcines priklausomybes* (SFP) ir *poaibio priklausomybes*. Galima išskirti dvi atominių atributų ir agreguotų atributų ekvivalentiškumo notacijas:

- reikšmių srities ekvivalentiškumas,
- funkcinis ekvivalentiškumas.

Sakykim, kad $D = (\overline{R}, \overline{I}, \overline{F})$ yra reliacinė duomenų bazės schema, kur *R* yra (baigtinė) reliacinių schemų R_i aibė, *I* yra poaibio priklausomybių aibė, *F* yra funkcinių priklausomybių aibė, o *U* yra atributų visuma.

Apibrėžimas 1. Atributas $A \in U$ yra ekvivalenti pagal sritį atributui $B \in U$ (žymima $A \approx B$), jei *A* ir *B* turi tą pačią reikšmių sritį:

$$Dom(A) = Dom(B).$$

Apibrėžimas 2. Dvi atributų aibės $V, Z \subseteq U$ yra ekvivalenčios pagal sritį (žymima $V \approx Z$), jei egzistuoja tokia bijektyvi funkcija, kad

$$t: V \rightarrow Z; (\forall A \in V)\ t(A) \approx A.$$

Apibrėžimas 3. Dvi atributų aibės $V, Z \subseteq U$ yra funkcionaliai ekvivalenčios funkcinių priklausomybių aibėje $\overline{F}$ (žymima $V \leftrightarrow Y$), *jei* $\overline{F} \models V \rightarrow Z$ *ir* $\overline{F} \models Z \rightarrow V$.

SFP pilnumas susideda iš dviejų dalių:

- struktūrinis pilnumas,
- semantinis pilnumas.

Šios pilnumo dalys bus apibūdintos atskirai poaibio priklausomybėms ir funkcinėms priklausomybėms.

2.1. Srities ekvivalentiškumo klasės ir struktūrinis pilnumas

Reliacinis modelis nepajėgia tiksliai ir aiškiai išreikšti sąryšių. Vietoje to, sąryšiai turi būti išreiškiami per poaibio priklausomybes. Tačiau schemos projektuotojas gali neatkreipti pakankamai dėmesio į PP specifikavimą, todėl sąryšiai, esantys schemoje, gali būti nevisiškai aiškūs. Viena iš pagrindinių užduočių, taikomų atvirkštinei inžinerijai, yra palengvinti šią problemą, t. y. identifikuoti sąryšius ir išreikšti juos poaibio priklausomybių forma. Dėl to yra svarbu žinoti, kaip identifikuoti sąryšius ir kada nutraukti jų paiešką.

Struktūrinis pilnumas koncentruojasi ties poaibio priklausomybėmis, kadangi jos gali būti tiesiogiai interpretuojamos, t. y. pagal jų semantinę prasmę.

Sakykime, turim reliacinės schemos *R* pozicijonuotų atributų $A_i \in V\ (i = 1, 2, \ldots, n)$ sąrašą *V*:

$$R.V.A_1, \ldots, A_n.$$

R.V bus vadinama reliacinės schemos *R* vidiniu sąryšiu arba atributų agregatu. Atributų agregatams $R.V = R.V.A_1, A_2, \ldots, A_n$ ir $Z.S = Z.S.B_1, B_2, \ldots, B_n$ iš dviejų reliacinių schemų *R* ir *Z* išorinis poaibio sąryšis turės sekančią formą:

$$R[V] \subseteq Z[S],$$

kai $A_i \approx B_i$ *ir* $A_i \subseteq B_i$, $i = 1, 2, \ldots, n$.

k_x bus vadinama srities ekvivalentiškumo klase su kardinalumu $|k_x| = n$, jei reliacinėje schemoje egzistuoja mažiausiai du atributų agregatai $Y_i.X_i$, tokie, kad

$$X \leftrightarrow Y_i.X_i. \text{ t.y.}$$

$$k_x = \{X \mid X \leftrightarrow Y_i.X_i,\ 2 \le i \le n\}$$

Apibrėžimas 4. Reliacinės duomenų bazės schema D yra k_x-struktūriškai pilna, jei kiekvienai susijusiai reliacinės schemos porai R_i, R_j, kur $R_i.X_i \leftrightarrow X$ ir $R_j.X_j \leftrightarrow X$, egzistuoja poaibio priklausomybė tarp $R_i.X_i$ ir $Rj.Xj$, t.y. $R_i.X_i \subseteq R_j.X_j$ arba $R_i.X_i \supseteq R_j.X_j$, arba išskyrimo apribojimas $R_i.X_i \cap R_j.Z_j = \varnothing$, arba persidengimo apribojimas $R_i.X_i \cap R_j.X_j \neq \varnothing$.

Apibrėžimas 5. Reliacinės duomenų bazės schema struktūriškai pilna poaibio priklausomybių atžvilgiu, jei D yra struktūriškai pilna visoms apibrėžimo ekvivalentiškumo klasėms k_x.

2.2. Srities ekvivalentiškumo klasės ir semantinis pilnumas

Bendru atveju, reliacinės schemos struktūrinis pilnumas nepakankama būtina sąlyga transformacijai, nes poaibio priklausomybės ir ryšiai gali turėti skirtingą interpretaciją atsižvelgiant į tikslo modelio projektavimo konstruktus. Poaibio priklausomybė, pavyzdžiui, gali atstovauti *is_a* sąryšį ar *part_of* sąryšį agregate arba paprasčiausią egzistavimo apribojimą. Be to, optimizavimo technologijos, tokios kaip išvestų ar pakartotų atributų įvedimas arba horizontalios ar vertikalios suskaidymo ar apjungimo operacijos gali būti panaudotos reliacinei schemai dėl programos vykdymo efektyvumo padidinimo. Taigi, prieš atliekant transformaciją, yra būtina išmesti optimizavimo struktūras ir schemą padaryti tokią, kad ji tenkintų primityvumo ir minimalumo kriterijus. Tai atlikus, šaltinio schemos struktūros turėtų būti tinkamai interpretuojamos, tam, kad gauti teisingą jų atvaizdavimą tikslo modelyje. Semantinis pilnumas poaibio priklausomybių atžvilgiu yra esminis dalykas aiškiam ir tiksliam paslėptų objektų aibių atvaizdavimui.

Sakykim, kad D yra struktūriškai poaibio priklausomybių atžvilgiu pilna reliacinės duomenų bazės schema, o $Y_i.X_i$ ir $Y_j.X_j$ yra du srities ekvivalentiškumo klasės k_x atributų agregatai.

Apibrėžimas 6. Schema D yra X-semantiškai pilna, jei kiekvienai reliacinių schemų Y_i, Y_j porai $Y_i.X_i$, $Y_j.X_j$ egzistuoja schema Y su atributų agregatu $Y.X$, tokia, kad

$$Y_i[X_i] \subseteq Y[X] \text{ ir}$$

$$Y_j[X_j] \subseteq Y[X].$$

Apibrėžimas 7. Schema D yra semantiškai pilna, jei ji yra X-semantiškai pilna visoms ekvivalentiškumo klasėms $k_x \in K$.

2.3. Funkcinės priklausomybės ir struktūrinis pilnumas

Sudėtinė funkcinė priklausomybė turi sekančią forma:

$$(X_1, X_2, \ldots, X_n) \rightarrow Y,$$

kur $X_1, X_2, \ldots, X_n$ yra visos reliacinės schemos R skirtingi poaibiai, o Y taip pat yra schemos R poaibis. Sąryšis $r(R)$ tenkina sudėtinę funkcinę priklausomybę $(X_1, X_2, \ldots, X_n) \rightarrow Y$, jei jis tenkina funkcinę priklausomybes $X_i \rightarrow X_j$ ir $X_i \rightarrow Y$, kur $1 \leq i, j \leq k$. Šioje sudėtinėje funkcinėje priklausomybėje $(X_1, X_2, \ldots, X_n)$ yra kairioji pusė, $X_1, X_2, \ldots, X_n$ yra kairiosios pusės aibės, o yra kairiosios pusės aibės, o Y yra dešinioji pusė.

SFP yra nieko daugiau negu sutrumpintas užrašymas FP su ekvivalentiškomis kairiosiomis pusėmis. Čia dėl patogumo yra daromas nedidelis nukrypimas, priimant prielaidą, kad $Y = \varnothing$. Tokiu atveju SFP užrašoma kaip $(X_1, X_2, \ldots, X_n)$.

Sakykime, kad $\overline{Q}$ yra sudėtinių funkcinių priklausomybių aibė. $\overline{Q}$ yra ekvivalenti $\overline{F}$ (žymima $\overline{Q} \equiv \overline{F}$), jei kiekvienas sąryšis $r(R)$, kuris tenkina $\overline{Q}$, tenkina ir $\overline{F}$, ir atvirkščiai.

Tegul reliacinės schemos R funkcinių priklausomybių aibei $\overline{F}$ ir aibei $X \subseteq R$, $\overline{E}_{\overline{F}}(X)$ būna FP aibės $\overline{F}$ poaibis, kurio funkcinių priklausomybių kairiosios pusės būna ekvivalenčios X.

Tegul SFP aibės $\overline{Q}$ $\overline{E}_{\overline{F}}$ būna aibė

$$\{\overline{E}_{\overline{F}}(X) \mid X \subseteq R \text{ ir } \overline{E}_{\overline{F}}(X) \neq \varnothing\}.$$

$\overline{E}_{\overline{F}}(X)$ yra tuščia, kai nei viena funkcinės priklausomybės iš $\overline{F}$ kairioji pusė nėra lygi X. $\overline{E}_{\overline{F}}$ visada yra $\overline{F}$ poaibis.

Sudėtinės funkcinės priklausomybės $(X_1, X_2, \ldots, X_n) \rightarrow Y$ iš $\overline{Q}$ gali būti redukuojamos, minimizuojamos ir optimizuojamos.

Apibrėžimas 8. Aibė $\overline{Q}$ yra objektinė dengtis aibei $\overline{F}$, jei $\overline{Q} \equiv \overline{F}$ ir jei neegzistuoja aibės X ir Z skirtingose kairiosiose pusėse su $X \leftrightarrow Y$ aibėje $\overline{Q}$.

Sakykime, kad $\overline{F}$ yra duota funkcinių priklausomybių aibė, o $\overline{E}_{\overline{F}}$ - sudėtinių funkcinių priklausomybių aibė.

Apibrėžimas 9. Reliacinė schema yra struktūriškai pilna FP atžvilgiu, jei

1) $\overline{E}_{\overline{F}} \models \overline{F}$,
2) $\overline{F}$ yra objektinė dengtis.

Apibrėžimas 10. Reliacinės duomenų bazės schema D yra struktūriškai pilna (FP ir PP atžvilgiu), jei:

1) egzistuoja objektinė dengtis $\overline{Q}$, minimali ir redukuota kairiosiomis pusėmis, tokia, kad $\overline{Q} \models \overline{F}$;
2) kiekvienai poaibio priklausomybei $R[V] \subseteq Z[S]$ egzistuoja sąryšio schemos Q ir O bei vidiniai sąryšiai $Q.V$ ir $O.S$, kuriems $Q[V] \subseteq O[S]$;
3) objektinė dengtis yra redukuota pagal dešiniąsias puses tokiu laipsniu, kad jis būtų pakankamas atvaizduoti visas poaibio priklausomybes.

2.4. Semantinis pilnumas

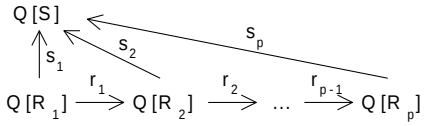
Sakykime, kad turime išplėstąją reliacinę schemą D:

$$D = (\overline{Q}, \overline{I}, \overline{F}), \; Q \in \overline{Q};$$

$$Q: (R_1, R_2, \ldots, R_p) \rightarrow S;$$

$$Q = \bigcup_{i=1}^{p} R_i \cup S .$$

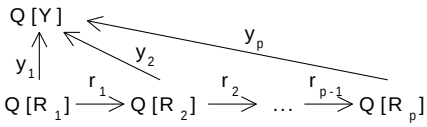
Apibrėžimas 11. Schema *D* yra *Q*-semantiškai pilna atžvilgiu $\overline{F}$, jei *Q* gali būti atvaizduota *Q*-komutacine diagrama (pav. 4).



4 pav. *Q*-komutacinė diagrama

Apibrėžimas 12. Schema *D* yra semantiškai pilna atžvilgiu $\overline{F}$, jei visos $Q \in \overline{Q}$ gali būti atvaizduojamos komutacinėmis diagramomis.

Tegul schema *Q* būna semantiškai pilna ir *Q* yra suprojektuota į schemą *Y*, pavyzdžiui, turime *Q.R*, $R \subseteq Q$, ir *Q[Y]* yra kairioji arba dešinioji bet kokios poaibio priklausomybės iš $\bar{I}$ pusė. Tada akivaizdu, kad *Q.Y*-komutacinė diagrama (pav. 5) taip pat yra semantiškai pilna atžvilgiu $\overline{F}$.



5 pav. *Q.Y*-komutacinė diagrama

Apibrėžimas 13. Schema *D* yra semantiškai pilna (atžvilgiu $\overline{F}$ ir $\bar{I}$), jei ji semantiškai pilna atžvilgiu $\overline{F}$ ir semantiškai pilna kiekvienai $Q.Y \in Q.\overline{Y}$ ir visiems $Q \in \overline{Q}$.

Čia $Q.\overline{Y}$ yra schemos *Q* vidinių sąryšių, kurie priklauso bent vienos poaibio priklausomybės kairiajai ar dešiniajai pusei, aibė.

2.5. Išplėstinės reliacinės schemos užbaigimas

Pateiktame požiūryje pirmasis duotos reliacinės schemos semantinio papildymo etapas susideda iš:

- visų galimų atominių, vienos reikšmės atributų agregatų surinkimo iš visų galimų šaltinių (duomenų bazių reliacinių schemų ir taikomųjų programų, vartotojo uždavinių scenarijų [33], vartotojo dokumentų) tam, kad būtų galima juos panaudoti srities semantikos atkūrimui funkcinių priklausomybių aibės formoje; čia turi būti identifikuojami atributų ir atributų agregatų vardų homonimai.
- visų galimų atominių, vienos reikšmės atributų agregatų surinkimo iš visų galimų šaltinių tam, kad būtų galima juos panaudoti srities semantikos atkūrimui poaibio priklausomybių aibės formoje; čia turi būti identifikuojami atributų vardų homonimai.
- paslėptų objektų aibių atradimo ir trūkstamų esybių įvedimo į išplėstinę reliacinę schemą.

Šiame etape nei vienas iš surinktų schemų negali turėti PJ priklausomybių [34].

Antrasis šaltinio schemos semantinio papildymo etapas susideda iš:

- duomenų pastovumo apskaičiavimas sudėtinių funkcinių priklausomybių formoje minimizuojant ir redukuojant duotos funkcinių priklausomybių aibės duomenų objektinę dengtį; čia turi būti identifikuojami schemų ir atributų vardų sinonimai.
- paslėptų objektų aibių atradimo ir trūkstamų esybių įvedimo į išplėstinę reliacinę schemą.

2.5.1. Paslėptų objektų aibių atradimas ir trūkstamų esybių įvedimas

Sakykime, turime dvi reliacines schemas Studentas ir Dėstytojas (pav. 6), kur

$$Dom(Stud\_ID) = Dom(Dėst\_ID).$$

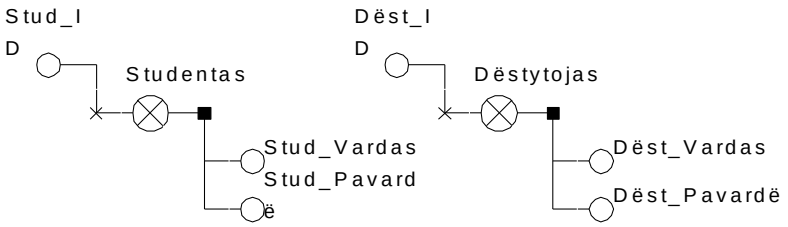
Tačiau

$$Studentas[Stud\_ID] \cap Dėstytojas[Dėst\_ID] = \varnothing.$$

Todėl būtų netiesa, kad

$$Studentas[Stud\_ID] \subseteq Dėstytojas[Dėst\_ID]$$ arba

$$Dėstytojas[Dėst\_ID] \subseteq Studentas[Stud\_ID].$$



6 pav. Reliacinės schemos *Studentas* ir *Dėstytojas*

Pastebėsime, kad studentų identifikatorių ir dėstytojų identifikatorių reikšmių aibės priklauso tai pačiai identifikatorių reikšmių aibei (pažymėkime ją *Asm_ID*), t. y.

$$Dom(Stud\_ID) = Dom(Asm\_ID)$$ ir

$$Dom(Dėst\_ID) = Dom(Asm\_ID)$$

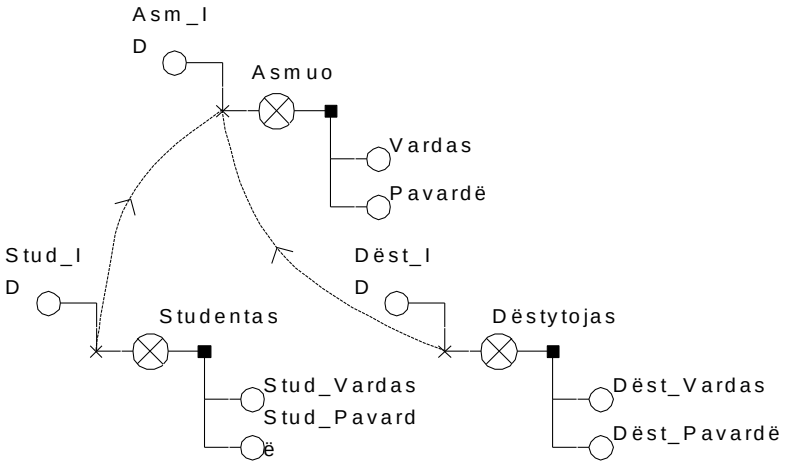
ir, be to,

$$Studentas[Stud\_ID] \subseteq Asmuo[Asm\_ID]$$ ir

$$Dėstytojas[Dėst\_ID] \subseteq Asmuo[Asm\_ID],$$

Čia klasių *Studentas* ir *Dėstytojas* egzempliorių sąjunga sudaro paslėptą objektų aibę, ir, jai identifikuoti, įvedama nauja trūkstama sąvoka *Asmuo* (pav. 7)

Tokiu būdu schema yra papildoma dar viena esybe ir yra kuriama objektų klasifikacija, kuri vėliau pavirs į klasių-poklasių sąryšius.



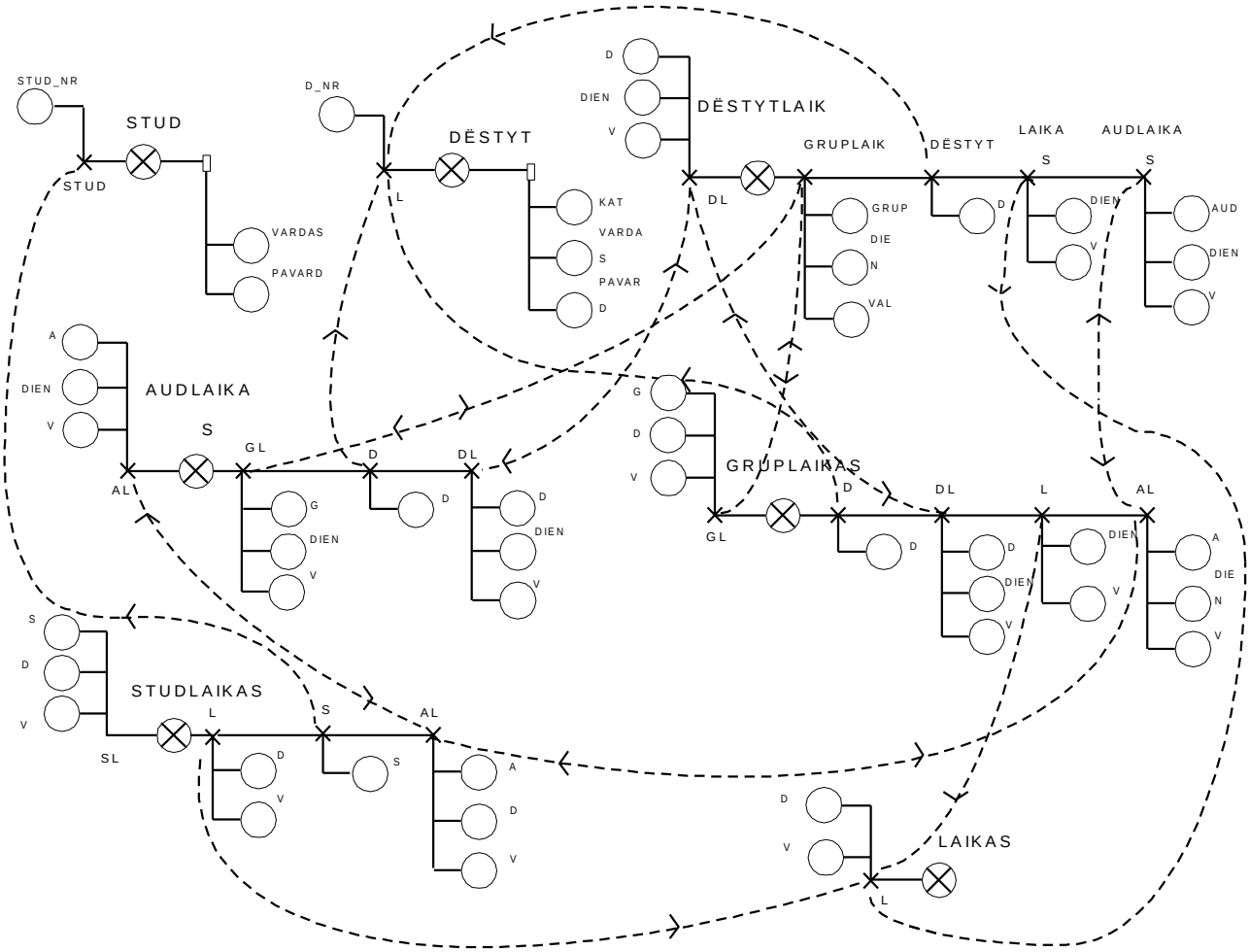
7 pav. Sukurta nauja reliacinė schema *Asmuo*

2.5.2. Objektinės dengtys

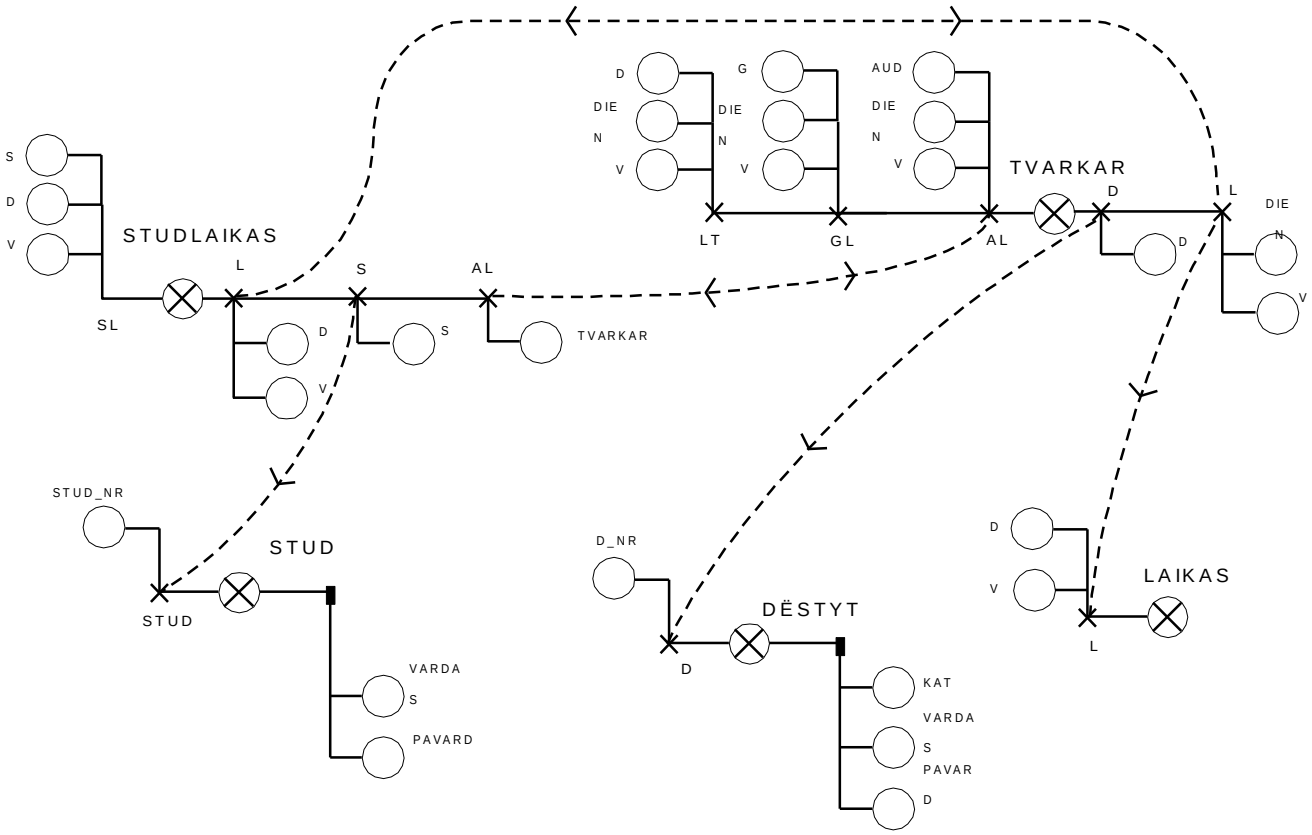
Objektinės dengtys yra naudojamos schemų kiekio sumažinimui. Formalus objektinės dengties apibrėžimas yra pateiktas 2.3 skyrelyje (Funkcinės priklausomybės ir struktūrinis pilnumas) apibrėžime 8. Detaliai objektinių dengčių skaičiavimas ir sinonimų identifikavimas yra aprašytas [35].

Nagrinėjant objektines dengtis, yra laikoma, kad atributai, esantys agregate, yra sunumeruoti, tai yra visada išlaiko tą pačią eilės tvarką, visose reliacinėse schemose (RS) vienodai. Sudarant objektines dengtis yra apjungiamos reliacinės schemos su vienodomis kairiosiomis funkcinių priklausomybių pusėmis.

Panagrinėkime atvejį, pavaizduotą 8 paveiksle. Čia egzistuoja trys reliacinės schemos (*DĖSTYTLAIK*, *AUDLAIK*, *GRUPLAIK*) su ekvivalentiškomis kairiosiomis pusėmis. Todėl šias schemas reikia apjungti. Apjungus šias tris reliacines schemas, yra sukuriama nauja RS, kuri yra pavadinama *TVARKAR* (pav. 9).



8 pav. Reliacinė schema su FP ir PP



9 pav. Reliacinė schema su SFP ir PP

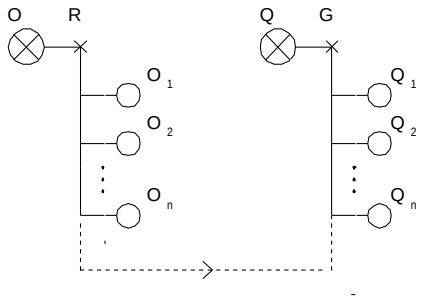
3. *Is_a* sąryšių transformacija

Atvirkštinės inžinerijos *is_a* sąryšių transformacijos etape iš poaibio priklausomybių tarp sudėtinių atributų reikšmių yra nustatomi (išvedami) *is_a* sąryšiai tarp koncepcinės schemos esybių [36]. Bendrasis poaibio priklausomybės atvejis yra pateiktas 10 paveiksle. Išorinė poaibio priklausomybė $O[R] \subseteq Q[G]$ koncepcinėje RPQ schemoje specifikuojama tada, kai kiekviena klasės *O* sudėtinio atributo *R* reikšmė $<o_1, o_2, ..., o_n>$ yra klasės *Q* sudėtinio atributo *G* reikšmė, t. y. iš

$$<o_1, o_2, ..., o_n> \in O[R]$$

seka

$$<o_1, o_2, ..., o_n> \in Q[G]$$



10 pav. Bendrasis poaibio priklausomybės atvejis

Egzistuoja šešios pagrindinės *is_a* sąryšių transformacijos taisyklės. Pateiktose taisyklėse tipų struktūrai specifikuoti panaudota trijų pozicijų sintaksė [15]:

X - klasė,
X.Y - klasės *X* sudėtinis atributas *Y*,
X.^Y - klasės *X* sudėtinis identifikatorius ^*Y*,
X.Y.Z - sudėtinio atributo (vidinio sąryšio) *X.Y* komponentinis atributas *Z*.

Jei klasės sudėtinis atributas identifikuoja klasės egzempliorius, tai toks sudėtinis atributas yra vadinamas sudėtiniu identifikatoriumi.

Taisyklė 1. Jei klasės *O* sudėtinio identifikatoriaus (SI) ^*R* reikšmių aibė yra klasės *Q* vidinio sąryšio *G* reikšmių aibės poaibis (pav. 11a), tai klasė *O* yra išoriniame *is_a* sąryšyje su klasės *Q* vidiniu sąryšiu *G* (pav. 11b):

$$\frac{O[\hat{}R] \subseteq Q[G]}{O \Rightarrow Q.G}$$

a **b**

11 pav. *Is_a* sąryšių transformacijos 1 taisyklė

Jei vidinis sąryšis susideda tik iš vieno atributo, tai toks sąryšis yra vadinamas unariniu ir vietoje jo galima naudoti tiesiog jo atributą.

Taisyklė 2. Jei klasės *O* unarinio identifikatoriaus (UI) ^*R* reikšmių aibė yra klasės *Q* unarinio atributo (UA) *G* reikšmių aibės poaibis (pav. 12a), tai sakome, kad klasė *O* yra išoriniame *is_a* sąryšyje su atributu *Q..P* (pav. 12b):

$$\frac{O \subseteq Q[G]}{O \Rightarrow Q..P}$$

Tai taisyklės (1) dalinis atvejis.

a **b**

12 pav. *Is_a* sąryšių transformacijos 2 taisyklė

Taisyklė 3. Jei klasės *O* vidinio sąryšio *R* reikšmių aibė yra klasės *Q* sudėtinio identifikatoriaus ^*G* reikšmių aibės poaibis (pav. 13a), tai klasės *O* vidinis sąryšis *R* yra išoriniame *is_a* sąryšyje su klase *Q* (pav. 13b):

$$\frac{O[R] \subseteq Q[\hat{}\,G]}{O.R \Rightarrow Q}$$

a **b**

13 pav. *Is_a* sąryšių transformacijos 3 taisyklė

Taisyklė 4. Jei klasės *O* unarinio atributo *R* reikšmių aibė yra klasės *Q* unarinio atributo ^*G* reikšmių aibės poaibis (pav. 14a), tai klasės *O* atributas *P* yra išoriniame *is_a* sąryšyje su klase *Q* (pav. 14b):

$$\frac{O[R] \subseteq Q}{O..P \Rightarrow Q}$$

Tai taisyklės (3) dalinis atvejis.

a **b**

14 pav. *Is_a* sąryšių transformacijos 4 taisyklė

Taisyklė 5. Jei klasės *O* sudėtinio identifikatoriaus *^R* reikšmių aibė yra klasės *Q* sudėtinio identifikatoriaus *^G* reikšmių aibės poaibis (pav. 15a), tai klasė *O* yra išoriniame *is_a* sąryšyje su klase *Q* (pav. 15b):

$$\frac{O\,[\wedge R] \subseteq Q\,[\wedge G]}{O \Rightarrow Q}$$

a **b**

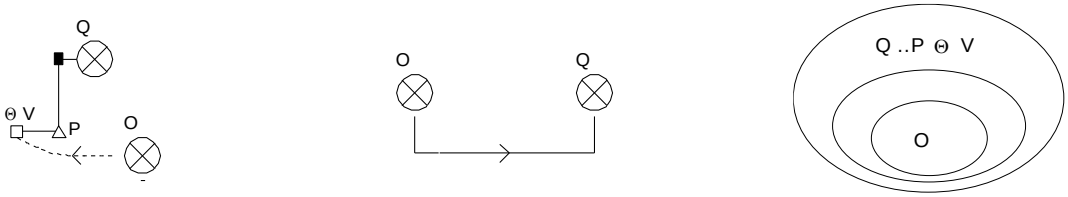
15 pav. *Is_a* sąryšių transformacijos 5 taisyklė

Jei klasės *Q* konstruktas trečioje (atributo) pozicijoje turi aritmetinį palyginimo operatorių Θ (=, $\neq$, <, >, $\leq$, $\geq$), tai jis yra žymimas *Q..P* Θ *V*, kur *V* - atributo reikšmė. Šiuo konstruktu deklaruojamas klasifikacijos sąryšis: klasėje *Q* išskiriamas objektų poaibis, kurio egzempliorių atributas *P* įgyja reikšmes, užduotas aritmetiniu palyginimo operatoriumi Θ atžvilgiu reikšmės *V*, pavyzdžiui, jeigu *P* = *V*, tai poaibio kiekvieno egzemplioriaus atributo *P* reikšmė yra lygi *V*.

Taisyklė 6. Jei klasės *O* reikšmių aibė yra klasės *Q* reikšmių aibės, apribotos operatoriumi Θ, reikšmių aibės poaibis (pav. 16a), tai kiekvienas *o*, $o \in O$, bus poaibio *Q..P* Θ *V* egzempliorius, o tuo pačiu ir klasės *Q* egzempliorius (pav. 16b):

$$\frac{O \subseteq Q..P\ \Theta\ V}{O \Rightarrow Q}$$

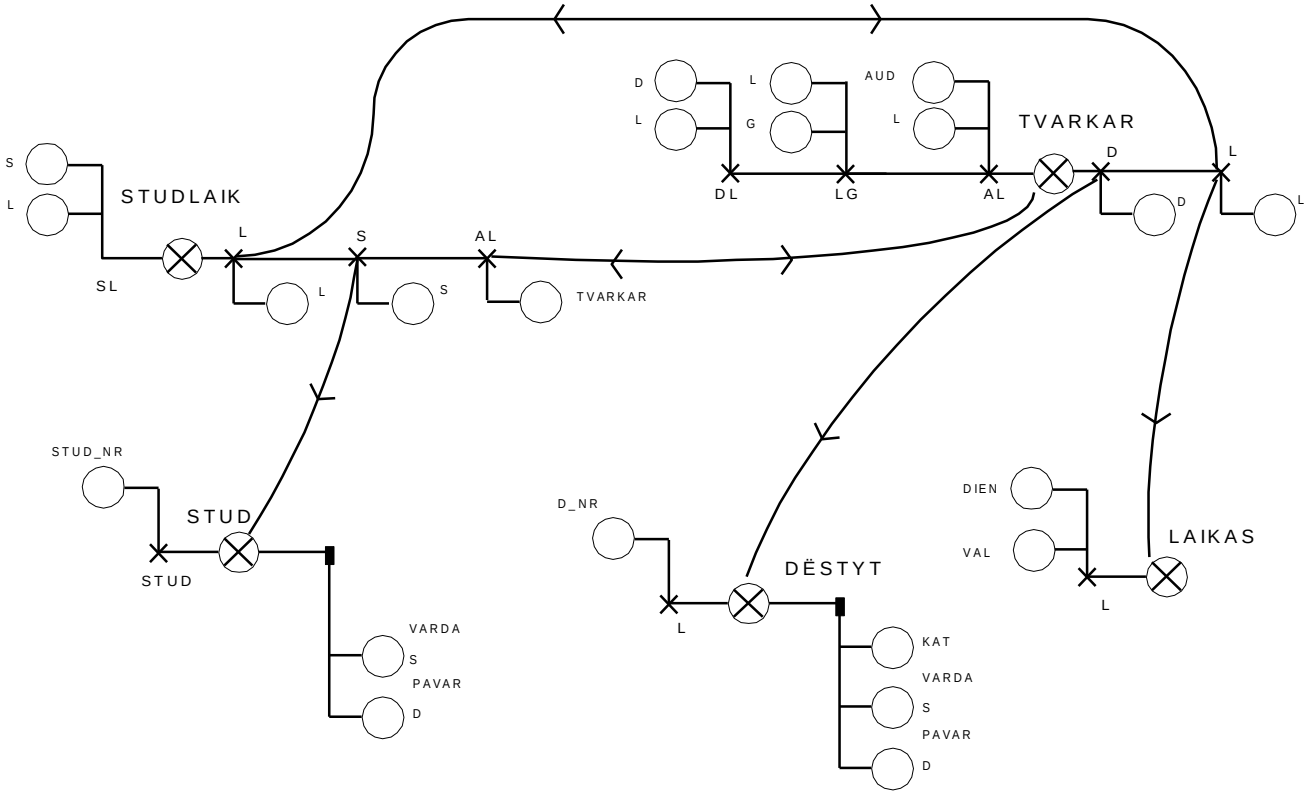
Šios taisyklės grafinis atvaizdavimas yra paveiksle 16c.



a **b** **c**

16 pav. *Is_a* sąryšių transformacijos 6 taisyklė

Tam, kad pailiustruoti *is_a* sąryšių sudarymą iš poaibio priklausomybių, grįžkime prie reliacinės duomenų bazės schemos su sudėtinėmis funkcinėmis ir poaibio priklausomybėmis, pavaizduotos 9 paveiksle. Šias priklausomybes transformavus į *is_a* sąryšius, gausime schemą su *is_a* sąryšiais (pav. 17).



17 pav. Schema su *is_a* sąryšiais

4. Metodų prijungimas prie abstrakčių duomenų tipų schemos

Šiame etape atvirkštinės inžinerijos tikslas yra abstrakčių duomenų tipų schemą papildyti metodais, taikomais abstraktiems duomenų tipams. Metodai yra dviejų rūšių:

- standartiniai,
- vartotojų definuoti.

Savo ruožtu vartotojų definuoti metodai yra skirstomi į:

- metodus, kurie paskaičiuoja klasės egzempliorių savybes ir išvestines savybes t. y. metodus, nusakančius klasės egzempliorių specifinę elgseną;
- metodai, kuriais definuojamos specifinės poklases.

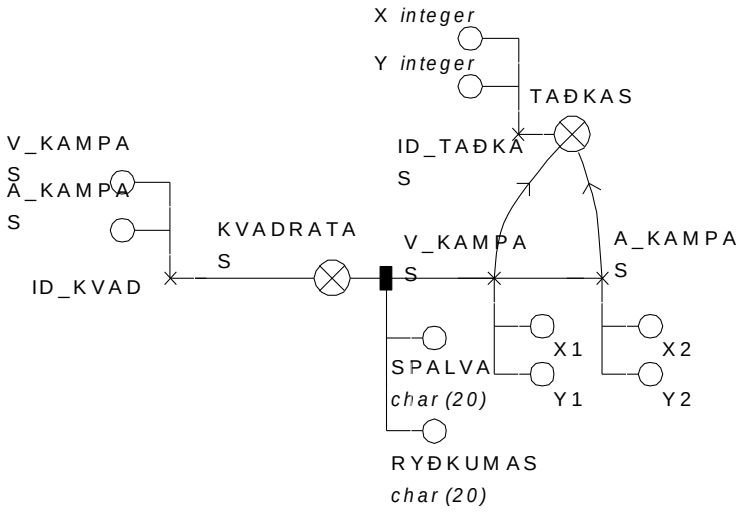
Standartiniai metodai yra charakterizuojami skyrelyje 4.2 (Abstrakčių duomenų tipų elgsenos standartinės funkcijos), o skyrelyje 4.1 (Vartotojų metodų specifikavimas) iliustruojamas vartotojų metodų panaudojimas.

4.1. Vartotojų metodų specifikavimas

Šis atvirkštinės inžinerijos etapas bus apibūdintas, panaudojant pavyzdį, kurio pagrindu yra paimtas *stačiakampio-kvadrato* pavyzdys [16]. Sakykim, turime klasę *KVADRATAS* (pav. 18). Jos egzemplioriai yra identifikuojami vidiniu sąryšiu *ID_KVAD*, kurį sudaro du abstrakčių duomenų tipo *TAŠKAS* atributai. Klasės *TAŠKAS* identifikatorius yra *ID_TAŠKAS*, kuris sudarytas agreguojant dvi koordinates *X* ir *Y*. Klasės *KVADRATAS* vidiniai sąryšiai su klase *TAŠKAS* yra sujungti *is_a* sąryšiais:

$$KVADRATAS.V\_KAMPAS \Rightarrow TAŠKAS;$$
$$KVADRATAS.A\_KAMPAS \Rightarrow TAŠKAS.$$

Taip pat klasės *KVADRATAS* egzemplioriai turi savybes *SPALVA* ir *RYŠKUMAS*.



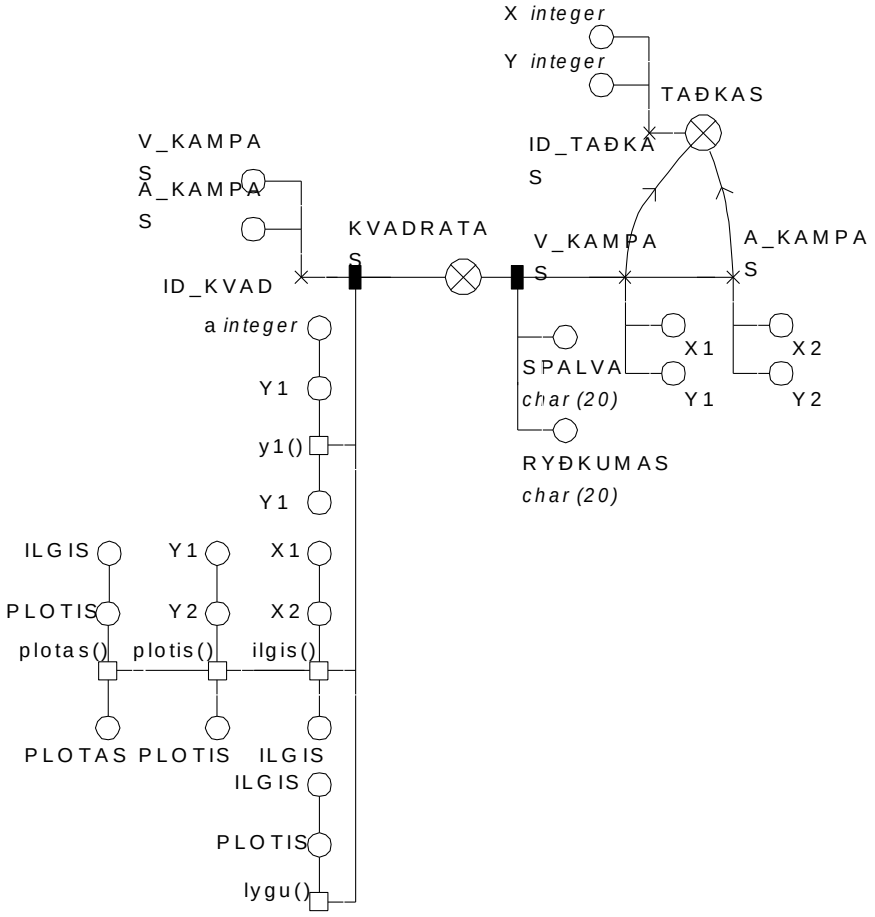
18 pav. Abstrakčių duomenų tipų schema

Šią abstrakčių duomenų tipų schemą papildžius metodais, nusakančiais klasės egzempliorių elgseną, gausime objektinės orientacijos schemą (pav. 19). Joje klasės *KVADRATAS* egzempliorių elgseną nusako funkcijos:

- *y1()*, kuri keičia koordinatę *Y1*; tai standartinis metodas;

- *plotas() ir ilgis()*, kurios nustato klasės *KVADRATAS* egzempliorių savybes *PLOTAS* ir *ILGIS*; šie metodai yra vartotojo definuoti metodai.

Be vartotojų sukurtų funkcijų gali būti panaudotos ir agreguotos funkcijos, tokios kaip maksimumo, minimumo, sumos ir kt.



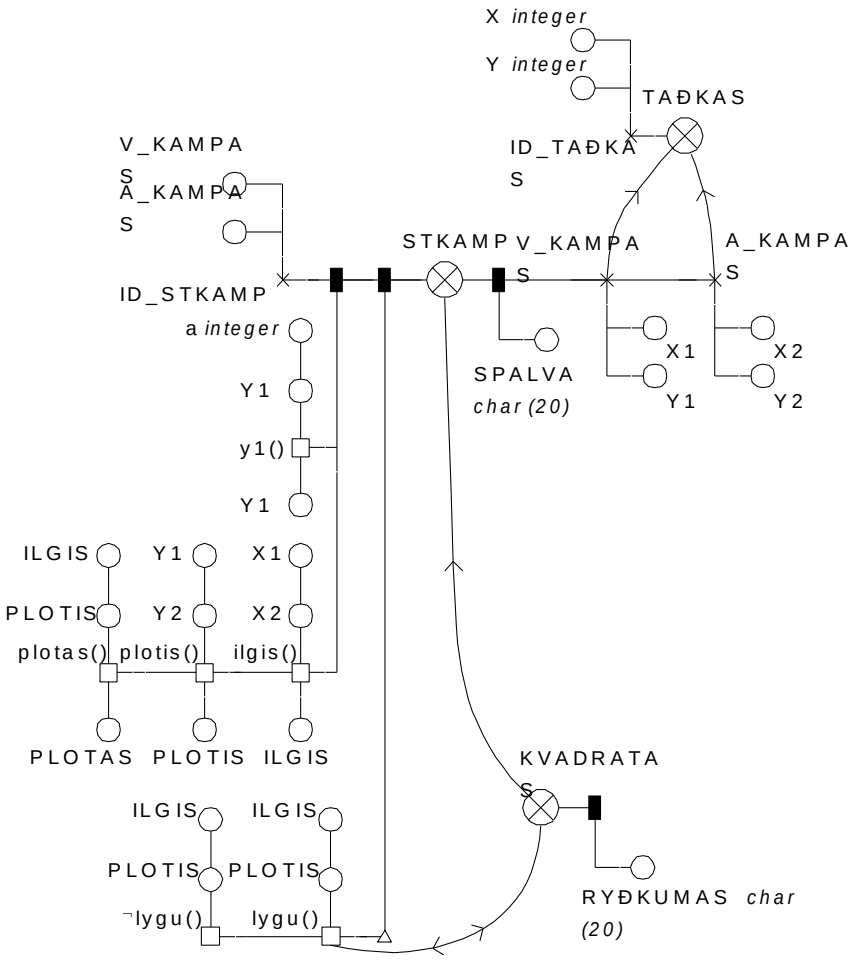
19 pav. Objektinės orientacijos schema su metodais, nusakančiais klasių egzempliorių elgseną

Gautą OO schemą galima dar papildyti metodais, skirtais išskirti poklases. Šie metodai inicijuoja išorinius paveldo *is_a* sąryšius tarp virš- ir poklasių.

Kas bus, jei funkcijos *lygu()* išėjimo reikšmė *TRUE* (ji specifikuota pagal nutylėjimą) dėl standartinės funkcijos *y1()* panaudojimo pavirs reikšme *FALSE*?

Šis funkcijos *y1()* panaudojimas inicijuoja schemos papildymą nauja sąvoka *STKAMP* su nauja objektų aibe (pav. 20). Kadangi kvadratas yra stačiakampis, kurio visos kraštinės lygios, tai klasei *KVADRATAS* galima sukurti viršklasę *STKAMP*, kurios egzemplioriai būtų stačiakampiai. Klasės *KVADRATAS* egzempliorių aibė bus klasės *STKAMP* egzempliorių aibės poaibis, kai savybės *ILGIS* ir *PLOTIS* bus lygios. Viršklasė nebūtinai turi turėti visus poklasės atributus ir metodus, dalis jų gali likti prie poklasės ir būti specifiniais tai poklasei (pvz., *RYŠKUMAS*). Be to gali tekti pakeisti kai kuriuos vidinių sąryšių ir atributų pavadinimus (pvz., *ID_KVAD* ir *ID_STKAMP*).

Viršklasės egzempliorius bus priskiriamas poklasei tada, kai ryšio metodas grąžins *TRUE* reikšmę.



20 pav. Objektinės orientacijos schema su metodais, leidžiančiais išskirti poklases, panaudojant paveldėjimo sąryšius

Panaudojant paveldėjimo sąryšius, galima sukurti ne tik viršklases, kurios išreikštų apibendrinimą, bet ir poklases, kurios išreikštų klasifikavimą.

Tam, kad užtikrinti, kad nebūtų pažeisti tam tikri objektų apribojimai programų vykdymo metu, pakeitus objekto savybes, reikia patikrinti, ar tas objektas vis dar priklauso tai klasei, t. y. galbūt jis jau gali priklausyti tik viršklasės objektų aibei, nes nebetenkina tam tikrų apribojimų, o galbūt jį galima priskirti poklasei, nes jis patenkino tam tikrus apribojimus. Pateiktame pavyzdyje metodas *y1()* keičia objekto (stačiakampio, ar jo specialaus atvejo - kvadrato) vertikalią kraštinę. Tačiau jei tas objektas priklauso klasei *KVADRATAS* ir pakeisime savybės *ILGIS* reikšmę, tai jis nebetenkins reikalavimo, kad funkcija *ilgis(ILGIS, PLOTIS)* grąžins *FALSE* reikšmę ir tas objektas nebegalės priklausyti klasei *KVADRATAS*. Analogiškai, gali atsitikti ir taip, kad stačiakampis taps kvadratu.

4.2. Abstrakčių duomenų tipų elgsenos standartinės funkcijos

Kai abstraktus duomenų tipas yra deklaruojamas, automatiškai yra užduodamos trijų tipų funkcijos:

- stebėjimo funkcijos,
- mutavimo funkcijos,
- ADT egzempliorių konstravimo funkcijos.

Deklaruosime 20 paveiksle pateikto pavyzdžio klasės *STKAMP* struktūrą.

```
CREATE TYPE stkamp
        (spalva varchar(20),
        id_stkamp id_stkamp)
CREATE TYPE id_stkamp
        (v_kampas v_kampas,
        a_kampas a_kampas)
CREATE TYPE v_kampas
        (x1 integer,
        y1 integer)
CREATE TYPE a_kampas
        (x2 integer,
        y2 integer)
```

Šios tipų definicijos automatiškai užduoda:

- konstruojančias funkcijas

  stkamp() → stkamp
  id_stkamp()→ id_stkamp
  v_kampas()→ v_kampas
  a_kampas()→ a_kampas

- stebėjimo funkcijas

  spalva(stkamp) → varchar(20)
  id_stkamp(stkamp) → id_stkamp
  v_kampas(id_stkamp) → v_kampas
  a_kampas(id_stkamp) → a_kampas
  x1(v_kampas) → integer
  y1(v_kampas) → integer
  x2(a_kampas) → integer
  y2(a_kampas) → integer

- mutatorius

  spalva(stkamp, varchar(20)) → stkamp
  id_stkamp(stkamp, id_stkamp) → stkamp
  v_kampas(id_stkamp, v_kampas) → id_stkamp
  a_kampas(id_stkamp, a_kampas) → id_stkamp
  x1(v_kampas, integer) → v_kampas
  y1(v_kampas, integer) → v_kampas
  x2(a_kampas, integer) → a_kampas
  y2(a_kampas, integer) → a_kampas

Deklaravus ADT paskelbiami tipų ir atributų kintamieji, kad būtų galima užduoti įvairias funkcijas tipų atžvilgiu. Toliau pateiktas programos fragmentas, kuriame du priskyrimo operatoriai yra tarpusavyje ekvivalentiški:

```
BEGIN
DECLARE st stkamp;
DECLARE sp varchar(20);
SET sp=st..spalva;
SET sp=spalva(st);
```

END;

ADT egzempliorių konstravimas išbaigiamas panaudojus konstravimo funkcijas, kurios yra palaikomos kaip ADT definicijos dalis. Kai konstravimo funkcija yra aktyvizuota, tada:

- generuojamas naujas ADT egzempliorius,
- priskiriama egzemplioriui tipo žymė,
- kiekvienam naujo egzemplioriaus atributui priskiriama nulinė (null) reikšmė.

Pavyzdžiui:

BEGIN
DECLARE st stkamp;
DECLARE sp varchar(20);
SET st=stkamp();
SET st..spalva='RAUDONA';
SET st..id_stkamp=id_stkamp();
SET st..id_stkamp..v_kampas=v_kampas();
SET st..id_stkamp..v_kampas..Y1=10;
END;

Pirmaisiais dviem deklaravimo sakiniais tipų *stkamp* ir *varchar* kintamiesiems *st* ir *sp* priskiriamos nulinės reikšmės. Čia reikia pastebėti, kad tipo kintamojo nulinė reikšmė nėra tas pats, kas tipo egzemplioriaus visų atributų nulinės reikšmės. Priskyrimo operatoriumi *SET st = stkamp()* sudaromas naujas stačiakampio egzempliorius ir priskiriamas kintamajam *st*. Šiuo atveju kintamasis *st* turi nenulinį stačiakampio egzempliorių, kurio visi atributai yra nuliniai. Priskyrimo operatorius *SET st..spalva = 'RAUDONA'* atributui *SPALVA* priskiria reikšmę *RAUDONA*. Operatorius *SET st..id_stkamp = id_stkamp()* sukuria tipo *id_stkamp* naują egzempliorių ir priskiria jį tipo *st* atributui *id_stkamp*. Naujas *id_stkamp* egzempliorius nėra nulinis egzempliorius. Šis naujas nenulinis identifikatorius turi nulinius atributus. Analogiškai sukuriamas tipo *v_kampas* egzempliorius. Ir pagaliau tipo *v_kampas* atributas *Y1* po paskutinio priskyrimo įgauna reikšmę *10*.

ADT gali būti apibrėžtas kaip kito ADT potipis, priskiriant jam reikiamą vardą:

CREATE TYPE kvadratas UNDER stkamp(ryškumas varchar(20));

Naujas potipis *KVADRATAS* paveldi tipo *STKAMP* struktūrą ir elgseną. Struktūros paveldas reiškia, kad potipis *KVADRATAS* turi visus *STKAMP* atributus ir dar jam deklaruotas naujas papildomas atributas *RYŠKUMAS*. Be to, *KVADRATAS* paveldi visas supertipo *STKAMP* operacijas, kurios charakterizuoja stačiakampio elgseną. Tokiu būdu visos tipo *STKAMP* operacijos gali būti taikomos tipo *KVADRATAS* visiems egzemplioriams. Tipo *KVADRATAS* deklaravimas automatiškai sukuria konstruojančią funkciją

$$kvadratas() \rightarrow kvadratas.$$

Be to, visos stebėjimo ir mutavimo funkcijas, apibrėžtas tipui *STKAMP* paveldi subtipas *KVADRATAS*.

OOK palaiko tiek funkcijų perkrovimo, tiek funkcijų dinaminio persiuntimo procesus. Jeigu funkcija yra iškviečiama, tai vykdymui pasirinktas funkcijos egzempliorius programos vykdymo metu priklauso nuo aktualizuotų argumentų tipo. Funkcijos egzempliorius parenkamas iš taikytinų tarpo toks, kuris geriausiai vykdymo metu tinka duotiems argumentų tipams.

5. Semantinio modelio įvedimo grafinis redaktorius ir jo taikymas objektinėms schemoms generuoti

Sukurta programa, kuri leidžia redaguoti objektiškai orientuotą schemą, o vėliau iš deklaruotos schemos sugeneruoja duomenų bazės aprašą, skirtą duomenų bazei sukurti.

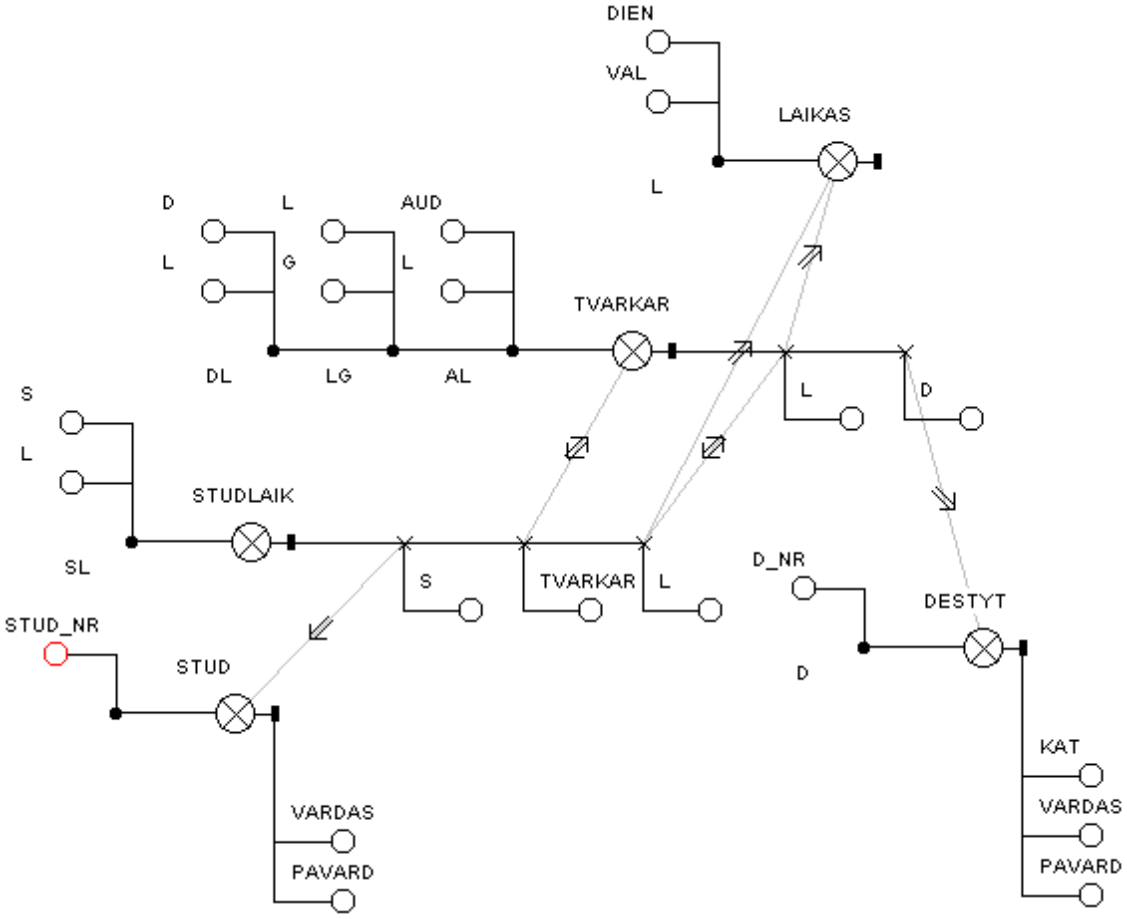
Programa leidžia schemoje naudoti sekančius konceptus:

- klasės (pastovios ir virtualios),
- vidiniai sąryšiai (identifikatoriai ir projekcijos),
- atributai (atominiai ir sudėtiniai),
- metodai (klasių egzempliorių elgseną apibūdinantys ir skirti išskirti klasių poklases),
- išoriniai sąryšiai.

Schemos aprašas yra sugeneruojamas POET 5.0 [31, 32] formate. Vėliau POET pagalba iš to sugeneruoto duomenų bazės aprašo yra sukuriamas vidinis POET duomenų bazės aprašas. Taigi galutinis mano sukurtos technologijos rezultatas yra POET OODB. Naudoti šios duomenų bazės duomenis galima iš taikomųjų programų, parašytų C++ kalba.

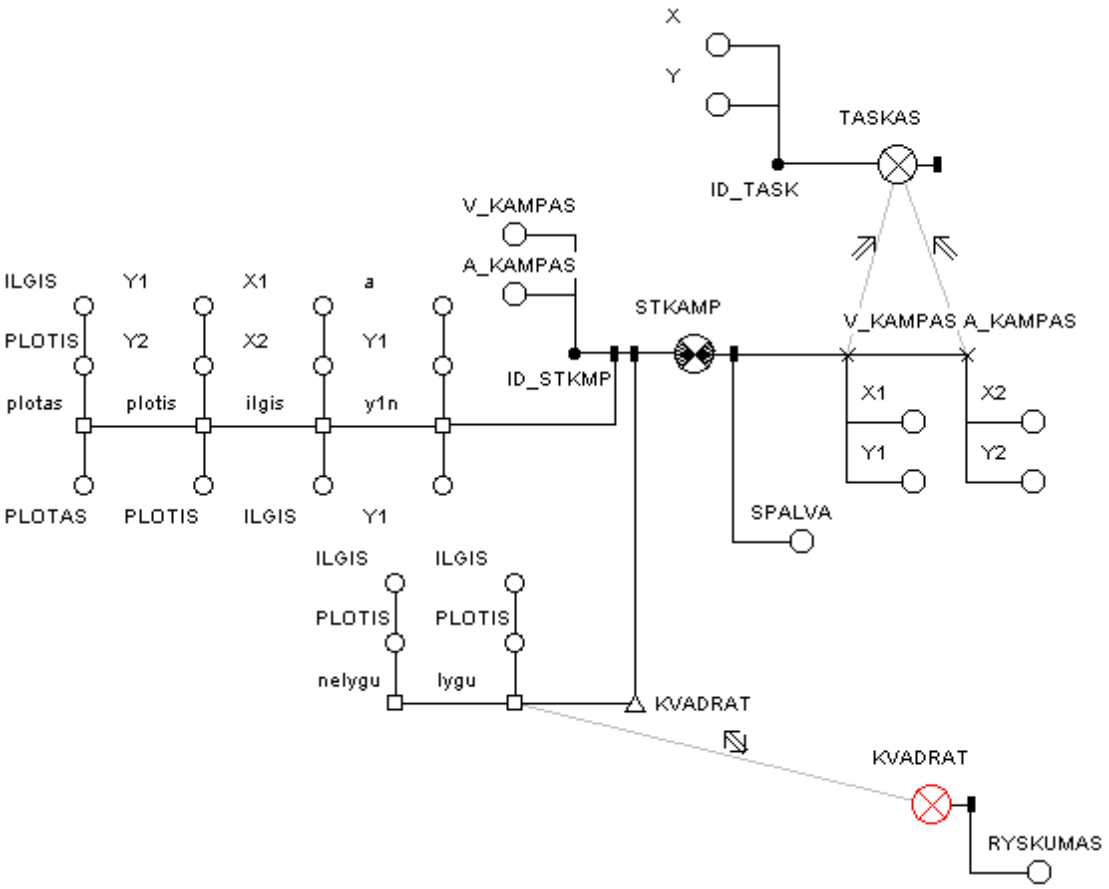
Sukurta mano duomenų bazės schema, neturi apribojimo, kad iš jos galima sukurti tik POET formato OODB. Esant reikalui, programą galima nesunkiai papildyti (t.y. sukurti naujus generatorius, kurie gali būti sukurti ir kaip atskiros programos), kad ji galėtų generuoti ir kitų OODB aprašus. Be to, sukurti nauji generatoriai galėtų generuoti ne tik OODB aprašus, bet ir reliacinių DB aprašus.

21 paveiksle yra pateiktas anksčiau aprašytas *Tvarkaraščio* pavyzdys. Skirtumas nuo ankstesnės *is_a* sąryšių schemos (pav. 17) yra toks, kad klasės *STUDLAIK* sudėtinio atributo *L* tipą būtinai reikia nurodyti sudarant tiesioginį sąryšį *STUDLAIK.L* $\Rightarrow$ *LAIKAS*. Be to šioje realizacijoje netinka vidinis sąryšis (VS) *AL*. Vietoje jo turi būti VS *TVARKAR*, nes sudėtinio atributo tipas yra nustatomas per to paties pavadinimo vidinį sąryšį.



21 pav. Tvarkaraščio schema

Be to, kaip jau buvo minėta, programa leidžia įvesti klasių egzempliorių elgseną apibūdinančius ir skirtus išskirti klasių poklases metodus. 22 paveiksle yra parodyta *Stačiakampio-kvadrato* OO schema (pav. 20), nubraižyta šia programa. Skirtumas nuo ankstesnės schemos (pav. 20) yra toks, kad visi klasių egzempliorių elgseną apibūdinantys metodai yra nubrėžti ant vienos linijos.



22 pav. Stačiakampio-kvadrato schema

Sugeneruotame POET duomenų bazės apraše yra ne tik klasių aprašai, bet ir naudojamų indeksų aprašai.

Sugeneruotų schemų aprašų tekstai yra pateikti priede.

Be to schemoje galima pavaizduoti ir kai kuriuos konceptus, kuriuos galima vėliau panaudoti tik reliacinės DB aprašyme:

- Klasės egzempliorių elgseną ar klasių egzempliorių apribojimus, skirtus suskirstyti į poklases, galima nurodyti ne tik metodų pagalba, bet ir specifinių, kokiai nors reliacinės duomenų bazių valdymo sistemos (RDBVS) būdingų išraiškų pagalba.
- Klasės egzempliorių elgseną ar klasių egzempliorių apribojimus, skirtus suskirstyti į poklases, galima nurodyti per Θ operatorių, kur $P\ \Theta V$ reiškia, kad operacija Θ yra taikoma atributui P, panaudojant reikšmę V.

6. Darbo išvados

1. Parinkti duomenų inžinerijos kriterijai ir principai, leidžiantys realizuoti tiesiogines ir atvirkštines objektinių ir reliacinių schemų transformacijas.
2. Mišrios duomenų inžinerijos principams realizuoti išskirti trys apibendrinti inžinerijos procesai – atributų reagregavimas, *is_a* sąryšių transformavimas, metodų prijungimas abstraktiems duomenų tipams.
3. Klasikinių bei išplėstinių reliacinių, abstrakčių duomenų tipų ir objektinių schemų taikymas, realizuojant mišrios duomenų inžinerijos technologinį principą, sudaro galimybę panaudoti fundamentalias, matematizuotas duomenų priklausomybes – poaibių, paprastąsias bei sudėtines funkcines ir *is_a* sąryšių priklausomybes.
4. Atvirkštinės inžinerijos procese atributų reagregavimo rezultate išvedamos sudėtinės funkcinės priklausomybės bei šių priklausomybių objektinės dengtys, kurios leidžia identifikuoti schemų ir jų atributų vardų sinonimus ir, tokiu būdu, padidinti reliacinių schemų semantinės išraiškos galimybes.
5. Sudarytos *is_a* sąryšių išvedimo iš išplėstinėmis reliacinėmis schemomis užduotų objektinių dengčių taisyklės; išvesti *is_a* sąryšiai savo kairiosiose ir dešiniosiose pusėse gali turėti struktūriniu požiūriu skirtingus (abstrakčius) duomenų tipus.
6. Abstraktiems duomenų tipams prijungiant elgsenos metodus, inicijuojami nauji *is_a* sąryšiai; šių sąryšių kairiosioms ir dešiniosioms pusėms įvedus naujas sąvokas, padidinamas abstrakčių duomenų tipų schemų semantinės išraiškos galimybės.
7. Sudarytas semantinio duomenų modelio grafinis redaktorius, kuriame numatyta galimybė generuoti objektinių schemų aprašus.

Literatūros sąrašas

[1] G. Kappel, S. Preishuber, E. Pröll, S. Rausch-Schott, W. Retschitzegger, R. Wagner, C. Gierlinger. *COMan - Coexistence of Object-Oriented and Relational Technology*. 13th Int. Conf. on the Entity-Relationship Approach - ER '94. Springer-Verlag LNCS 881, Manchester, UK, 1994. 259-277 psl.

[2] G. Engels, G. Kappel. *Object-Oriented Systems Development: Will the New Approach Solve Old Problems?* IFIP94 World Congress, Vol. III, K. Duncan and K. Krueger (eds.), Norht Holland, 1994.

[3] O.M. Nierstrasz. *A Survey of Object-Oriented Concepts*. Object-Oriented Concepts, Databases and Applications, W. Kim and F. Lochovsky (eds.), ACM Press and Addison-Wesley, 1989.

[4] Ch. Fahrner, G. Vossen. *A Modular Approach to Relational Reverse Engineering*. University of Muenster, Germany, Berich Nr. 22/96-I, 1996.

[5] C. Batini, S. Ceri, S.B. Navathe. *Conceptual Database Design - An Entity-Relationship Approach*. Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1992.

[6] R.H.L. Chiang, T.M. Barron, V.C. Storey. *Reverse Engineering of Relational Databases: Extraction of a EER Model from a Relational Database*. Data & Knowledge Engineering 12, 1994.

[7] M. Gogolla, R. Herzig, S. Conrad, G. Denker, N. Vlachantonis. *Integrating the ER Approach in an OO Environment*. 12th Int. Conf. on the Entity-Relationship Approach - ER '93. Springer-Verlag LNCS 823, Arlington, Texas, USA, 1993.

[8] B. Narasimhen, S.B. Navathe, S. Jayaraman. *On Mapping ER and Relational Models into OO Schemas*. 12th Int. Conf. on the Entity-Relationship Approach - ER '93. Springer-Verlag LNCS 823, Arlington, Texas, USA, 1993.

[9] R.G.G. Cattell. *Object data Management - Object-Oriented and Extended Relational Database Systems*. Addison-Wesley Publishing Company, Massachusetts, 1991.

[10] R. Ananthanarayanan, V. Gottemukkala, W. Käfer, T.J. Lehman, H. Pirahesh. *Using the Co-existence Approach to Achieve Combined Functionality of Object-Oriented and Relational Systems*. 1993 ACM SIGMOD Int. Conf. on Management of Data, P. Buneman and S. Jajodia (eds.), SIGMOD Record, Vo. 22, No. 2, Washington, DC, 1993.

[11] D. Moody. *What makes a Good Data Model? Evaluating the Quality of Entity-Relationship Models*. 13th Int. Conf. of the Entity-Relationship Approach 1994. Springer LNCS 881, Berlin, 1994. 94-111 psl.

[12] O. Lindland, G. Sindre, A. Solvberg. *Understanding Quality in Conceptual Modelling*. IEEE Software 11, 1994. 42-49 psl.

[13] R.J. Miller, Y.E. Ioannidis, R. Ramakrishnan. *The Use of Information Capacity in Schema Integration and Translation*. 19th Int. Conf. on Very Large Databases, 1993. 120-133 psl.

[14] R.H.L. Chiang, T.M. Barron, V.C. Storey. *Performance Evaluation of Reverse Engineering Relational Databases into Extended Entity-Relationship Models*. 12th Int. Conf. on the Entity-Relationship Approach - ER '93. Springer LNCS 823, Berlin, 1994. 352-363 psl.

[15] B. Paradauskas. *Conceptual Object-Relationship-Property Approach: Three Different Interpretations of the Same Entity*. Informatica, Vol. 6, No. 4, Institute of Mathematics and Informatics, Vilnius, 1995. 497-522 psl.

[16] B. Paradauskas, A. Vainauskas. *Koncepcinių schemų procedūrinė realizacija*. Konf. Informacinės technologijos. Kaunas, Technologija, 1997. 35-45 psl.

[17] American National Standards Institute (ANSI) Database Committee (X342). *Database Language SQL3*. J. Melton ed., August 1994.

[18] P.P. Chen. *The Entity-Relationship Model - Towards a Unified View of Data*. ACM Transactions on Databases, Vol. 1, No. 1, 1976. 9-36 psl.

[19] R.A. Elmasri, V. Kouramajian, B. Thalheim. *On Mapping ER and Relational Models into OO Schemas*. 12th Int. Conf. on the Entity-Relationship Approach - ER '93. Springer-Verlag LNCS 823, Arlington, Texas, USA, 1993. 402-413 psl.

[20] R.H.L. Chiang, T.M. Barron, V.C. Storey. *Reverse Engineering of Relational Databases: Extraction of an EER Model from Relational Database*. Data & Knowledge Engrg. 12, 1994. 107-142 psl.

[21] M. Anderson. *Extracting an Entity-Relationship Schema from a Relational Database through Reverse Engineering*. 13th Int. Conf. on the Entity-Relationship Approach - ER '94. Springer LNCS, Berlin, 1994. 403-419 psl.

[22] M.A. Casanova, J.E.A. de Sa. *Designing Entity-Relationship Schemas for Conventional Information Systems*. 3rd Int. Conf. on the Entity-Relationship Approach. North Holland, Amsterdam, 1983. 265-278 psl.

[23] W. Premerlani, M.R. Blaha. *An Approach for Reverse Engineering of Relational Databases*. Working Conf. on Reverse-Engineering. Baltimore, 1993. 151-160 psl.

[24] K.H. Davis, A.K. Arora. *Converting a Relational Database Model into an Entity-Relationship Model*. 6th Int. Conf. on the Entity-Relationship Approach. North Holland, Amsterdam, 1987. 271-286 psl.

[25] S.R. Dumpala, S.K. Arora. *Schema Translation Using a Entity-Relationship Approach*. 2nd Int. Conf. on the Entity-Relationship Approach. North Holland, Amsterdam, 1983. 337-356 psl.

[26] P. Johannesson, K. Kalman. *A Method for Translating Relational Schemas into Conceptual Schemas*. 8th Int. Conf. on the Entity-Relationship Approach 1989. North Holland, Amsterdam, 1990. 271-286 psl.

[27] P. Johannesson. *A Method for Transforming Relational Schemas into Conceptual Schemas*. 10th IEEE Int. Conf. on Data Engineering. Los Alamitos, 1994. 190-201 psl.

[28] V.M. Markowitz, J.A. Makovsky. *Identifying Extended Entity-Relationship Object Structures in Relational Schemes*. IEEE Trans. Software Engrg. 16, 1990. 777-790 psl.

[29] O. Signore et al. *Reconstruction of ER-Schema from Database Applications: A Cognitive Approach*. 13th Int. Conf. on the Entity-Relationship Approach 1994. Springer LNCS 881, Berlin, 1994. 387-402 psl.

[30] M. Castellanos, F. Saltor, M. Garcia-Solaco. *Semantically Enrichment Relational Databases into an Object-Oriented Semantic Model*. 5th Int. Conf. on Database and Expert Systems Applications - DEXA '94, SpringerLNCS 856, Berlin, 1994. 125-134 psl.

[31] *POET 5.0 ODL Compiler. Reference Guide*. http://www.poet.com/f_ftp.htm

[32] *POET 5.0 C++ SDK. Programmer's Guide*. http://www.poet.com/f_ftp.htm

[33] B. Paradauskas, L. Salelionis. *Struktūrinių tipų panaudojimas vartotojų scenarijų sudarymu*. Konf. Informacinės technologijos '98. Kaunas, Technologija, 1998. 39-43 psl.

[34] D. Maier. *The Theory of Relational Databases*. Pitman, 1983.

[35] M. Minkevičius. *Struktūrinių tipų funkcinė analizė*. Magistro tezės. KTU, 1998.

[36] L. Salelionis. *Apibendrinimo sąryšio išvedimas duomenų struktūrose*. Konf. Informacinės technologijos ir universitetinės studijos. Vytauto Didžiojo universitetas, Kaunas, 1998. 139-141 psl.

Priedas

Sugeneruotų schemų aprašų tekstai

Tavrkar.hcd

```
/*****************************************************************/
/*   This file is generated by <<< Extended ER Diagrams >>>   */
/*                                                               */
/*    If superclass(-es) is (are) below subclass(-es), you   */
/*   must move it (them) up.                                     */
/*    If two classes have the same index, you must change    */
/*   index name.                                                 */
/*****************************************************************/


class STUDLAIK
 {
 public:
        STUDLAIK ();
        ~STUDLAIK ();
        LAIKAS L;
        STUD S;
        TVARKAR TVARKAR;
 useindex S_Index;
 useindex TVARKAR_Index;
 useindex L_Index;
 };

class TVARKAR
 {
 public:
        TVARKAR ();
        ~TVARKAR ();
        LAIKAS L;
        int AUD;
        char G[5];
        DESTYT D;
 useindex AL_Index;
 useindex L_Index;
 useindex D_Index;
 };

class LAIKAS
 {
 public:
        LAIKAS ();
        ~LAIKAS ();
        char VAL[8];
        char DIEN[2];
 useindex L_Index;
 };

class DESTYT
 {
```

```
 public:
        DESTYT ();
        ~DESTYT ();
        int D_NR;
        char KAT[10];
        char VARDAS[10];
        char PAVARD[10];
 useindex D_Index;
};

class STUD
{
 public:
        STUD ();
        ~STUD ();
        int STUD_NR;
        char VARDAS[10];
        char PAVARD[10];
 useindex STUD_Index;
};


indexdef S_Index: STUDLAIK
{
        S.STUD_NR;
};

indexdef TVARKAR_Index: STUDLAIK
{
        TVARKAR.L;
        TVARKAR.AUD;
};

unique indexdef AL_Index: TVARKAR
{
        L.VAL;
        L.DIEN;
        AUD;
};

indexdef L_Index: TVARKAR
{
        L.VAL;
        L.DIEN;
};

unique indexdef L_Index: LAIKAS
{
        VAL;
        DIEN;
};

unique indexdef D_Index: DESTYT
{
        D_NR;
};
```

```
unique indexdef STUD_Index: STUD
 {
        STUD_NR;
 };

indexdef D_Index: TVARKAR
 {
        D.D_NR;
 };

indexdef L_Index: STUDLAIK
 {
        L.VAL;
        L.DIEN;
 };
```

Stkamp.hcd

```
/*****************************************************************/
/*   This file is generated by <<< Extended ER Diagrams >>>   */
/*                                                               */
/*    If superclass(-es) is (are) below subclass(-es), you   */
/*   must move it (them) up.                                     */
/*    If two classes have the same index, you must change    */
/*   index name.                                                 */
/*****************************************************************/


persistent class STKAMP
 {
  public:
        STKAMP ();
        ~STKAMP ();
        TASKAS A_KAMPAS;
        TASKAS V_KAMPAS;
        char SPALVA[20];
        int X1;
        int Y1;
        int X2;
        int Y2;
        int y1n (int *Y1, int n1, int *a, int n2);
        int ilgis (int *X2, int n1, int *X1, int n2);
        int plotis (int *Y2, int n1, int *Y1, int n2);
        int plotas (int *PLOTIS, int n1, int *ILGIS, int n2);
  useindex V_KAMPAS_Index;
  useindex A_KAMPAS_Index;
 };

class TASKAS
 {
  public:
        TASKAS ();
        ~TASKAS ();
        int Y;
        int X;
 };
```

```
class KVADRAT: public STKAMP
 {
  public:
          KVADRAT ();
          ~KVADRAT ();
          char RYSKUMAS[20];
          int lygu (int *PLOTIS, int n1, int *ILGIS, int n2);
 };


indexdef V_KAMPAS_Index: STKAMP
 {
          X1;
          Y1;
 };

indexdef A_KAMPAS_Index: STKAMP
 {
          X2;
          Y2;
 };
```

STRUKTŪRINIŲ TIPŲ PANAUDOJIMAS VARTOTOJŲ SCENARIJŲ SUDARYMUI

B. Paradauskas, L. Salelionis

Kauno technologijos universitetas

Išanalizavus klasių, asociacijų ir funkcinio modelių semantinės išraiškos galimybes aprašant objektų sistemos statines ir dinamines savybes, pasiūlyta informacijos poreikiams integruoti panaudoti probleminės srities konceptų struktūrinius tipus. Šią tipų specifikavimo sistemą sujungus su objektų sistemos dinaminių savybių specifikacijomis sudaryta detalizuota vartotojų scenarijų diagrama, leidžianti tiksliai apibrėžti vartotojų informacijos poreikių turinį.

1. Įvadas

Informacijos sistemų projektavimo objektinėje metodologijoje [1, 2] naudojami klasių asocijavimo, dinaminis, funkcinis ir klasių komunikavimo modeliai. Statiniai ir dinaminiai objektų sistemos apribojimai specifikuojami įvairiomis diagramomis, kurios sudaro pramoninių CASE priemonių pagrindą taikomųjų uždavinių kodo generavimui. IS projektavimui sukurtame ObjectTeam pakete [3] objektų sistemos modeliavimui naudojamos 7 diagramos:

Objektų modelis:	1) Klasių asocijavimo diagrama (CAD);
Dinaminis modelis:	2) Būsenų transformavimo diagrama (STD),
	3) Vartotojų uždavinių diagrama (UCD),
	4) Įvykių sekos diagrama (ETD);
Funkcinis modelis:	5) Duomenų srautų diagrama (DFD);
Klasių komunikavimo modelis:	6) Klasių komunikavimo diagrama (CCD),
	7) Pranešimų apibendrinimo diagrama (MGD).

Įvairių autorių požiūriams apibendrinti sukurta unifikuota modeliavimo kalba (UML) [4]. Pagrindinius struktūrinius ir dinaminius aspektus, kurie reikalingi vartotojų uždavinių kodo generavimui, modeliuoja CAD-, STD-, DFD-diagramos. Visos kitos diagramos patikslina šių diagramų informaciją, jas tarpusavyje „subalansuojant“ pakankamo detalumo lygmenyje: „How the Modelling Techniques Fit Together“ [4]. Šiuose projektavimo procesuose dalyvauja vartotojai, analitikai, projektuotojai ir programuotojai. Sistemos išvystymo ir reinžinerijos procesuose svarbu yra tiksliai apibrėžti vartotojų veiklą ir jų informacijos poreikius visoje informacijos sistemoje. Šie aspektai specifikuojami sąsajų srautų diagramomis (CD) ir anksčiau minėtomis UCD-, ETD-, CCD-, MGD-diagramomis.

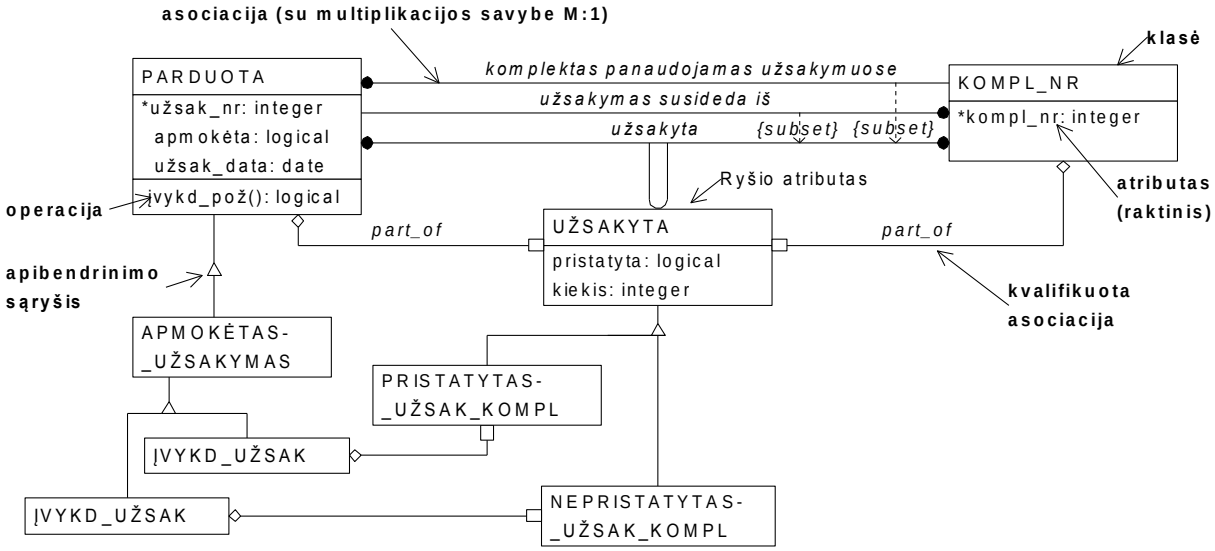
Šiame straipsnyje parodytas požiūris, kaip pagrindinių diagramų informaciją ir vartotojų vietą sistemoje galima apibendrinti, naudojant duomenų struktūrinius tipus [5] ir detalizuotas vartotojų scenarijų diagramas.

2. Iliustracinis užsakymų vykdymo kontrolės uždavinys

Užsakymų vykdymo kontrolės uždavinį [6] iliustruosime tokiomis diagramomis: klasių asocijavimo diagrama (1 pav.), duomenų srautų diagrama (2 pav.) ir būsenų transformavimo diagrama (3 pav.).

Čia parodytos klasės, klasių atributai ir kvalifikuotos klasių asociacijos. Pagrindiniai objektų sąryšiai yra apibendrinimo (*isa*) arba agregavimo (*part_of*) sąryšiai su nurodytomis kardinalumo (arba multiplikacijos) savybėmis. Pirkėjo pateiktas užsakymas (*PARDUOTA*) prekybinės firmos makleriui identifikuojamas užsakymo numeriu (*užsak_nr*). Užsakymas (*UŽSAKYTA*) susideda iš kelių skirtingų prekių komplektų (*KOMPL_NR*) su nurodytu komplektų kiekiu (kiekis). Apmokėtas užsakymas (*APMOKĖTAS_UŽSAKYMAS*) yra toks (*apmokėta=.T.*), kai už viso užsakymo prekes pinigai pervesti iš pirkėjo sąskaitos į prekybinės firmos sąskaitą. Pristatytas užsakymo komplektas (*PRISTATYTAS_UŽSAK_KOMPL*) yra pirkėjui atiduotas užsakymo komplektas. Užsakymas yra įvykdytas (*ĮVYKD_UŽSAK*), kai jis

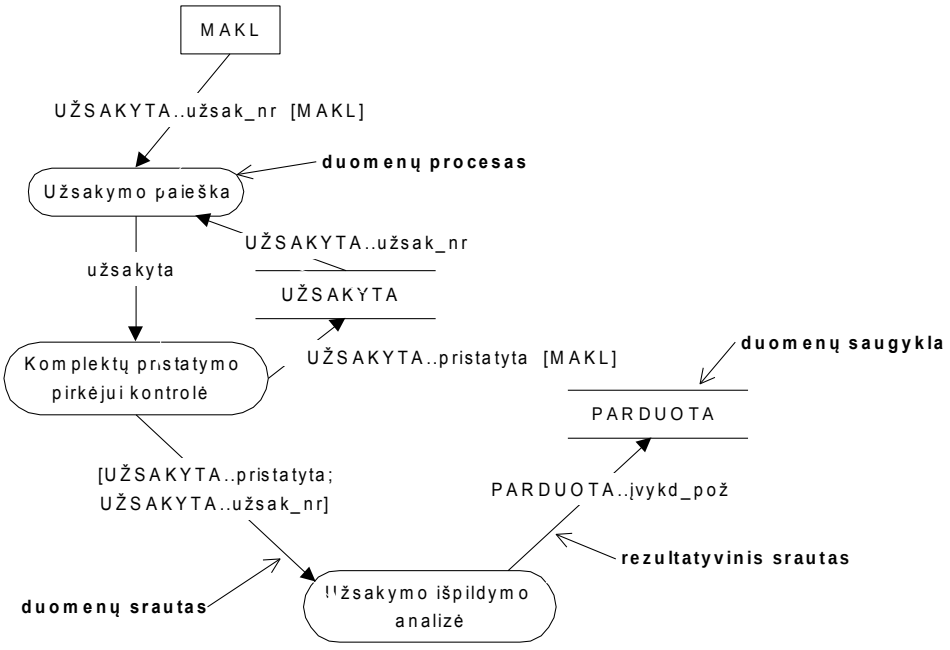
yra pilnai apmokėtas (*ĮVYKD_UŽSAK isa APMOKĖTAS_UŽSAKYMAS*) ir kai visi užsakymo komplektai yra pirkėjui pristatyti.



1 pav. Klasių asocijavimo diagrama.

Duomenų srautų diagramoje parodytos duomenų saugyklos (*UŽSAKYTA*, *PARDUOTA*), trys procesai (*Užsakymo paieška*, *Komplektų pristatymo pirkėjui kontrolė*, *Užsakymo išpildymo analizė*), aktorius *MAKLERIS* ir duomenų srautai. Srautų identifikavimui naudojami SQL3 kalbos [7] duomenų tipai. Ši kalba atžvilgiu SQL turi abstrakčių duomenų tipų praplėtimą. Tipų specifikavimo sintaksę galima charakterizuoti trijų pozicijų struktūra:

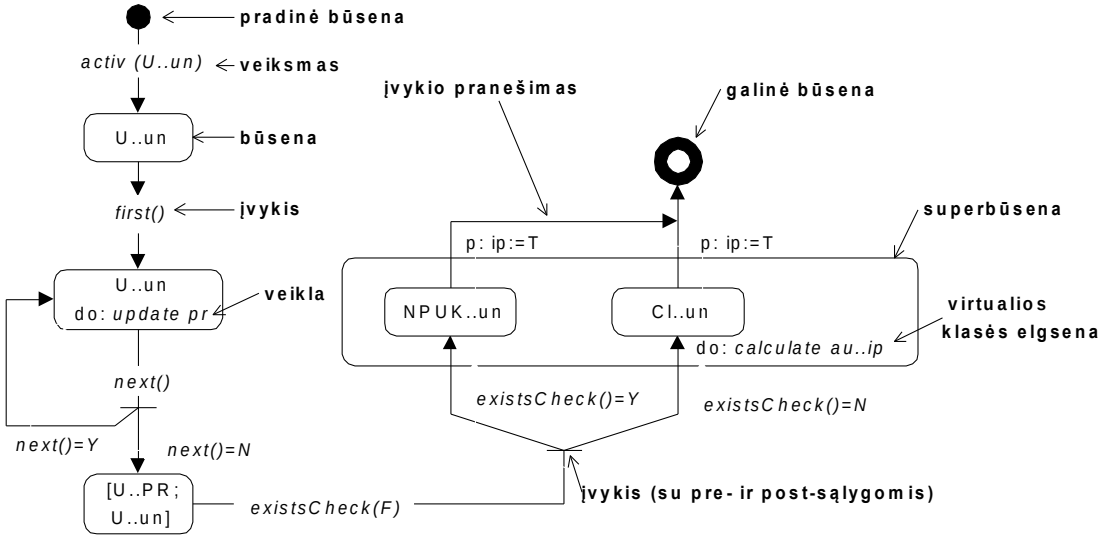
X - klasė,
X.Y - klasės X sudėtinis atributas Y,
X.Y.Z - sudėtinio atributo X.Y atributas Z,
x; X.y; X.Y.z; X..z - objektiniai kintamieji.



2 pav. Duomenų srautų diagrama

Aktorius *MAKL* siunčia klasės *UŽSAKYTA* atributo reikšmę *užsak_nr* procesui *Užsakymo paieška*. Šis procesas objektą-egzempliorių *užsakyta* perduoda sekančiam procesui

Komplektų pristatymo pirkėjui kontrolė, kuris saugyklai *UŽSAKYTA* persiunčia objekto-egzemplioriaus *užsakyta* reikšmę *pristatyta*. Šią reikšmę nustato aktorius *MAKL*. Iš saugyklos *UŽSAKYTA* paimami įrašai su tuo pačiu užsakymo numeriu *užsak_nr*. Procesas *Komplektų pristatymo pirkėjui kontrolė* perduoda kitam procesui *Užsakymo išpildymo analizė* agreguotą duomenų *srautą [UŽSAKYTA..pristatyta; UŽSAKYTA..užsak_nr]*, kuriame fiksuojama aibė binarinių reikšmių (*.T.* arba *.F.*), asocijuota su tuo pačiu klasės *UŽSAKYTA* kintamuoju *užsak_nr*. Pastarasis procesas į saugyklą *PARDUOTA* persiunčia rezultatyvinį srautą, kuriame yra nustatyta objekto-egzemplioriaus *parduota* atributo reikšmė *įvykd_pož*.



3 pav. Būsenų transformavimo diagrama

Būsenų transformavimo diagramoje būsenos identifikuojamos objektiniais parametrais - klasių tipais (potipiais), atributų tipais, atributų reikšmėmis ir reikšmių poaibiais. Būsenos transformuojamos aktoriaus veiksmais ir įvykiais-funkcijomis. Įvykiai gali turėti pre- ir post-sąlygas, apibendrinančias lygiagrečius procesus, ir superbūsenas, t. y. būsenas, susidedančias iš kelių dekomponuotų būsenų. Pavyzdžiui, įvykis-funkcija *next()* gali objektų būseną pervesti į būseną *U..un* arba *[U..PR; U..un*], priklausomai nuo post-sąlygų *next()=Y* arba *next()=N*.

3. Objektų sistemos atvaizdavimas struktūriniais tipais

Objektų sistemos koncepcinėms priklausomybėms atvaizduoti paimsime koncepcinį **QRP**-modelį [5], kuriame struktūriniams tipams apibrėžti naudojamos tokios pagrindinės meta-prielaidos:

Koncepto vardas yra arba / ir:
- klasės (objekto),
- sudėtinio atributo vidinio sąryšio,
- atributo (savybės)

vardas.

Pav. 4 pateikta pastovių ir virtualių klasių struktūrinių tipų asocijavimo diagrama. Pavyzdžiui, objekto *NEPRISTATYTAS_UŽSAKymo_KOMPLektas* unarinio vidinio sąryšio vardas *UŽSAK_NR* yra objekto *NEĮVYKDytas_UŽSAKymas (NIU)* vardas. Čia poaibio priklausomybė *NPUK.UŽSAK_NR* $\subseteq$ *NIU* traktuojama kaip išorinis isa-sąryšis *NPUK.UŽSAK_NR isa NIU*. Kiekvienas klasės *APMOKĖTAS_UŽSAKymas* egzempliorius apmokėtas_užsak su atributo *ĮVYKDymo_POŽymis* reikšme *įvykd_pož=.F.* yra klasės *NIU* egzempliorius: *APMOKĖTAS_UŽSAK..ĮVYKD_POŽ=.F. isa NIU*. Kiekvienas *įvykd_užsak* egzempliorius yra aibės *APMOKĖTAS_UŽSAK* egzempliorius su atributo *ĮVYKD_POŽ reikšme įvykd_pož=.T.* arba formaliai:

ĮVYKD_UŽSAK isa APMOKĖTAS_UŽSAK..ĮVYKD_POŽ=.T.,

o taip pat

ĮVYKD_UŽSAK isa APMOKĖTAS_UŽSAK.

Čia gali būti pritaikyta meta-išvedimo taisyklė IU isa AU. Analogiškai:

AU isa PARDUOTA..APMOKĖTA=.T.

ir

PARDUOTA..APMOKĖTA=.T. isa AU.

Iš šių dviejų koncepcinių priklausomybių seka simetrinė konceptų vardų sinonimo priklausomybė

AU isa PARDUOTA..APMOKĖTA=.T.,

o taip pat

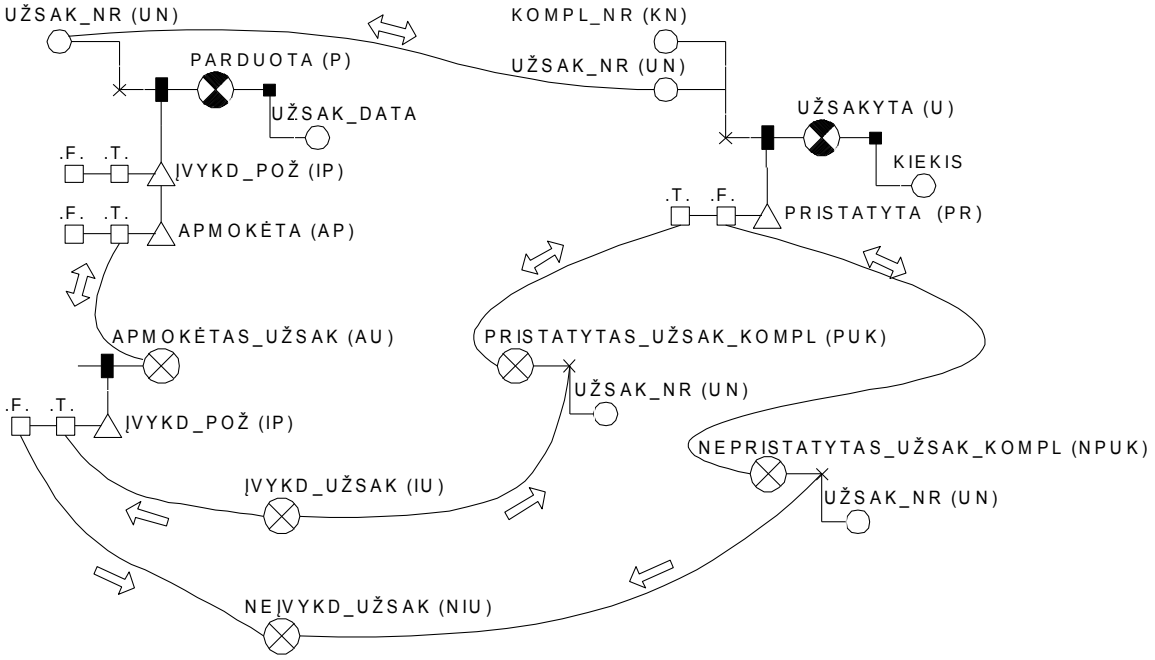
AU isa PARDUOTA.

Be to, isa-sąryšis yra tranzityvusis sąryšis, t. y. iš *IU isa AU* ir *AU isa PARDUOTA* seka *IU isa PARDUOTA.*

Šiame pavyzdyje parodyta dar koncepcinė priklausomybė

PARDUOTA..UŽSAK_NR isa UŽSAKYTA..UŽSAK_NR.

Be **sujungtųjų** atributų, kurie yra naudojami išorinėse priklausomybėse, šioje schemoje panaudoti **nesujungtieji** (arba **laisvieji**) atributai *PARDUOTA..UŽSAKymo_DATA* ir *UŽSAKYTA..KIEKIS.*



4 pav. Pastoviųjų (žym. ⋈) ir virtualiųjų (žym. ⊗) klasių struktūrinių tipų asocijavimo diagrama

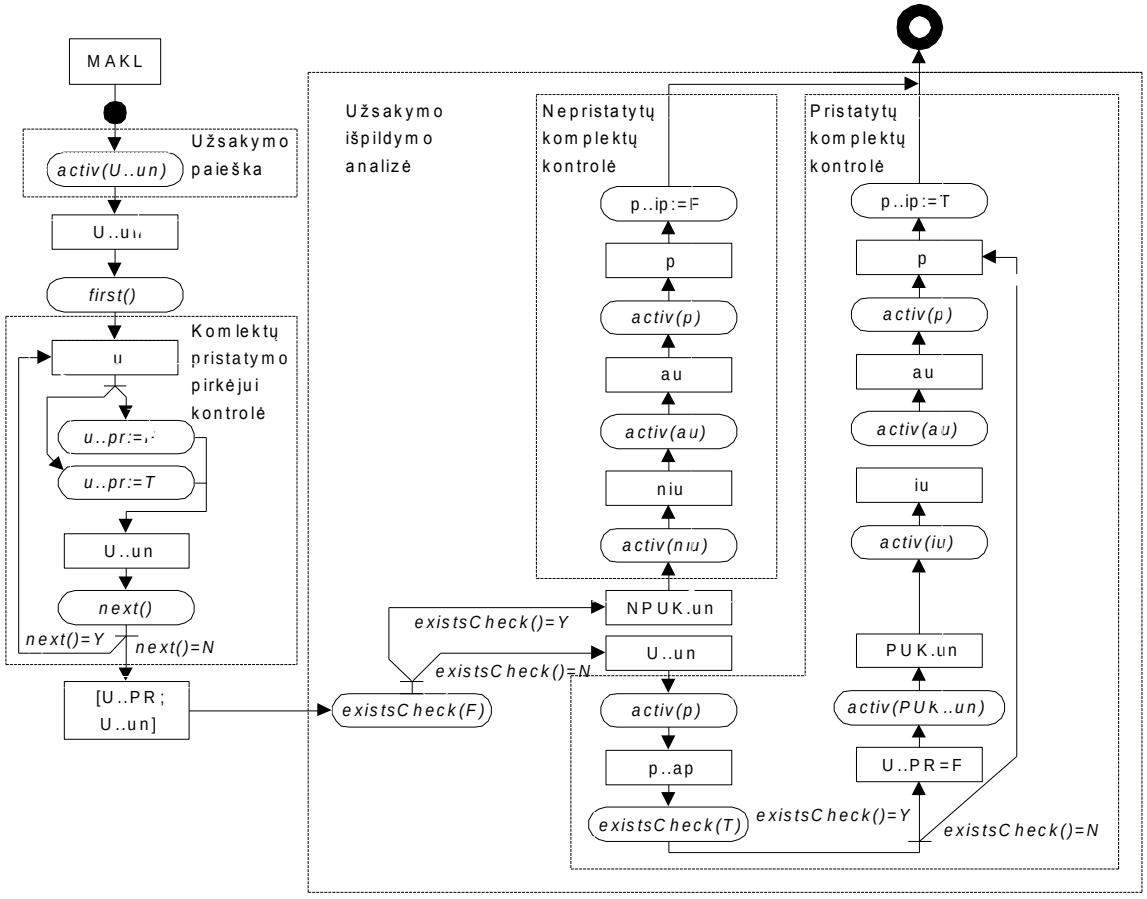
4. Detalizuoti vartotojų scenarijai

Detalizuojant duomenų srautų diagramos procesus (2 pav.) su duomenų srautais, identifikuojamais struktūrinių tipų asociacijomis (1 pav.), gali būti suprojektuota (sugeneruota) CASE priemonėmis vartotojo uždavinio scenarijaus diagrama (5 pav.). Šioje diagramoje sistemos dinaminės savybės perkeliamos iš būsenų transformavimo diagramos, panaudojant tokius pat grafinius žymėjimus. Scenarijaus diagrama bus detalizuota, jeigu būsenos identifikuojamos klasių (pastoviųjų ir virtualiųjų) asocijavimo struktūriniais tipais ir kiekvienas duomenų apdorojimo procesas yra **elementari** funkcija. Elementari funkcija yra standartinė arba tipinė klasės elgsenos funkcija [8]:

- konstruktorius arba destruktorius,
- stebėjimo funkcija,
- mutuojanti funkcija,
- statinės tipų kontrolės funkcija,
- objektų pakeičiamumo funkcija.

5. Išvados

1. Informacijos sistemų projektavimo CASE priemonėmis panaudojamuose objektų, dinaminiame ir funkciniuose modeliuose pasireiškia kriterijų stoka, integruojant procesų, klasių asocijavimo, būsenų transformavimo diagramas bei tiksliai apibrėžiant vartotojų informacijos poreikius.
2. Pateiktas iliustracinis šių diagramų panaudojimo pavyzdys, kuriuo iliustruojama ObjectTeam projektavimo priemonių semantinės galimybės specifikuoti statinius ir dinaminius objektų sistemos apribojimus.



5 pav. Užsakymo vykdymo kontrolės scenarijaus diagrama

3. Pasiūlyta statinėms semantinėms priklausomybėms specifikuoti panaudoti koncepcinę Objektų-Sąryšių-Savybių schemą, kurios pagrindą sudaro SQL3 kalbos abstrakčių duomenų tipų prototipas. Šiuo pagrindu probleminės srities struktūrinių tipų konceptai įgauna kontekstais motyvuotas klasių, asociacijų ir atributų kategorijas. Tai sudaro prielaidas paskirstytų vartotojų poreikiams integruoti į globalinę struktūrinių tipų specifikavimo schemą.
4. Procesų ir būsenų transformavimo diagramomis aprašytas objektų sistemos dinamines savybes apjungus su struktūrinių tipų specifikavimo schema, sudaryta vartotojų scenarijų detalizuota diagrama, kuri tiksliai atspindi vartotojų informacijos poreikių turinį ir sudaro prielaidas vartotojų uždavinių programinio kodo generavimui.

Literatūra

[1] **J. Rumbough, et al.** Object-Oriented Modelling and Design. *Englenood Cliffs, NJ: Prentice-Hall International, Inc.*, 1991.

[2] **G. Booch.** Object-Oriented Design. *Rednood City, Calif.: Benjamin/Cammings.*, 1991.

[3] Cayenne: ObjectTeam. Modeling Guide. Version 5.1.1. *Cayenne Software, Inc.*, January 1997.

[4] **Scott W. Amber.** The Unified Modelling Language v1.1 and Beyond: The Techniques of Object-Oriented Modeling. *http://www.ambysoft.com/amuAndBeyond.pdf*, September 20, 1997.

[5] **B. Paradauskas.** Conceptual Object-Relationship-Property Approach: Three Different Interpretations of the Same Entity. *Informatica*, 1995, Vol. 6, N° 4, pg. 497-522.

[6] **L. Nemuraitė, B. Paradauskas.** Duomenų bazių laboratoriniai darbai. *Technologija, Kaunas*, 1997, 175 psl.

[7] **American National Standards Institute (ANSI) Database Committee (X342).** Database Language SQL3. *J. Melton, ed.,* August 1994.
[8] **B. Paradauskas, A. Vainauskas.** Koncepcinių schemų procedūrinė interpretacija. *Konferencijos "Informacinės technologijos '97" pranešimų medžiaga, Technologija, Kaunas,* 1997, 35-45 psl.

STRUCTURAL TYPES FOR USER SCENARIO DEVELOPMENT

B. Paradauskas, L. Salelionis

Summary

Possibilities of semantic expressiveness of classes, associations and functional models for description of static and dynamic properties of object system were analysed. For information requirements integration the use of problem domain concept structural types was proposed. The integration of specification schema of these types with specifications of dynamic properties of object system results in detail diagram of user scenarios, which enables a clear definition of contents of user requirements.

Keywords: object system, class associations, attributes, static properties of object system, dynamic properties of object system, concepts, structural type, process diagram, state transition diagram, user scenario diagram.

APIBENDRINIMO SĄRYŠIO IŠVEDIMAS DUOMENŲ STRUKTŪROSE

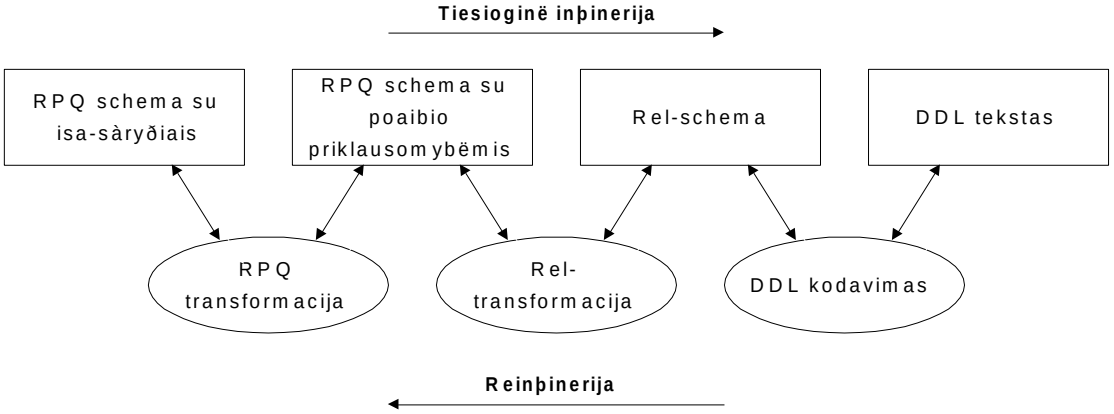
Linas Salelionis (Kauno technologijos universitetas)

Vadovas Bronius Paradauskas

Apibendrinimo - specializavimo ir agregavimo - detalizavimo sąryšiai yra baziniai semantiniai konstruktai, naudojami duomenų struktūrų projektavimo ir integravimo metodologijoje. Šie sąryšiai specifikuojami išplėstinėmis esybių - sąryšių diagramomis (EERD) arba klasių asocijavimo diagramomis (CAD). Šios diagramos sudaromos informacijos sistemų koncepcinio modeliavimo lygmenyje ir vėliau, duomenų bazių (DB) schemų ir informacijos sistemų (IS) vartotojų uždavinių projektavimo stadijoje, transformuojamos į duomenų struktūrinius tipus - reliacinius arba objektinius.

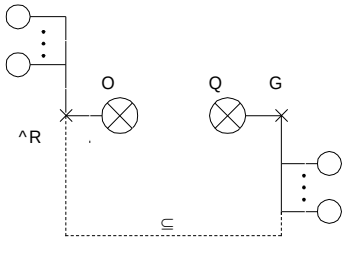
Objektiniams tipams specifikuoti panaudosime Sąryšių - Savybių - Objektų (RPQ) schemą [1]. Šios schemos aprašymo kalba semantikos prasme artima objektinės orientacijos kalbai, kurios prototipu galima nurodyti SQL3 kalbą [2]. SQL3 kalba atžvilgiu SQL turi abstrakčių duomenų tipų praplėtimą. Objektinių duomenų tipų panaudojimas vartotojų scenarijų sudarymui iliustruotas [3].

Reliacinių ir objektinių duomenų klasių integracijos poreikis stimuliavo reinžinerijos metodų išvystymą ir panaudojimą. 1 paveiksle parodytos dvi DB schemų transformacijos - RPQ ir Rel-transformacijos. Tiesioginės inžinerijos kryptimi koncepcinė RPQ schema transformuojama į konkrečios reliacinės duomenų bazių valdymo sistemos (DBVS) duomenų aprašymo kalbos (DDL) tekstą. Reinžinerijoje loginė reliacinė schema transformuojama į koncepcinę RPQ schemą. Šia reinžinerijos prasme svarbiausias RPQ transformacijos bruožas yra tas, kad iš poaibių priklausomybių tarp sudėtinių atributų reikšmių yra nustatomi (išvedami) is_a sąryšiai tarp koncepcinės schemos esybių. Rel-transformacijos uždaviniai pakankamai giliai išnagrinėti [4].

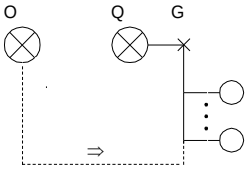


1 pav. Duomenų struktūrų inžinerija ir reinžinerija

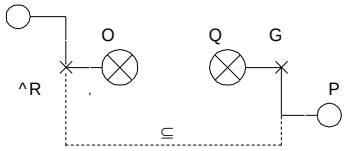
T1)



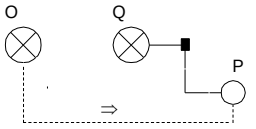
$$\frac{O .\hat{} \ R \subseteq Q.G}{O \Rightarrow Q.G}$$



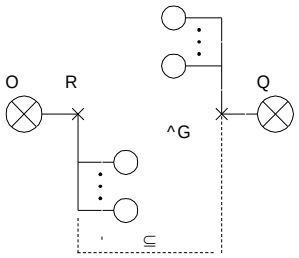
T2)



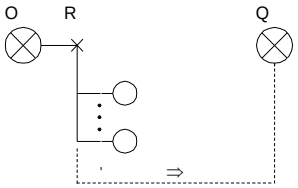
$$\frac{O \subseteq Q.G}{O \Rightarrow Q..P}$$



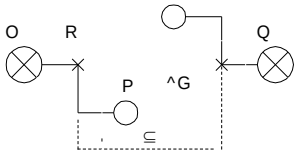
T3)



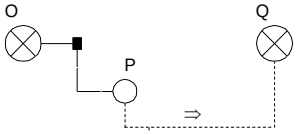
$$\frac{O.R \subseteq Q.\hat{}\ G}{O.R \Rightarrow Q}$$



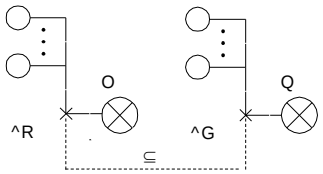
T4)



$$\frac{O.R \subseteq Q}{O..P \Rightarrow Q}$$

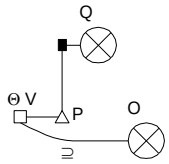


T5)



$$\frac{O.\hat{}\ R \subseteq Q.\hat{}\ G}{O \Rightarrow Q}$$

T6)



$$\frac{O \subseteq Q..P\ \Theta\ V}{O \Rightarrow Q}$$

2 pav. Išorinių is_a sąryšių išvedimo taisyklės

Šiose tezėse pateiktos išorinių (tarp klasių arba jų komponentų) isa-sąryšių išvedimo taisyklės (2 pav.). Išorinių sąryšių (grafiškai jie žymimi $\Rightarrow$) išvedimo prielaidos yra išorinės poaibių priklausomybės tarp sudėtinių atributų reikšmių aibių.

Pateiktose taisyklėse tipų struktūrai specifikuoti panaudota trijų pozicijų sintaksė [3]:

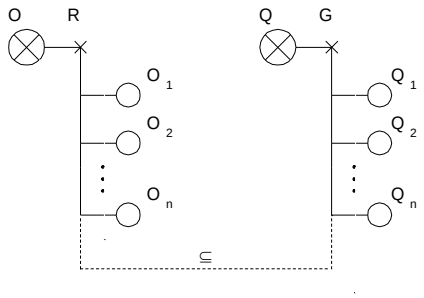
X - klasė,
X.Y - klasės X sudėtinis atributas Y,
X.^Y - klasės X sudėtinis identifikatorius ^Y,
X.Y.Z - sudėtinio atributo (vidinio sąryšio) X.Y komponentinis atributas Z.

Bendrasis poaibio priklausomybės atvejis yra 3 paveiksle. Išorinė poaibio priklausomybė (grafiškai žymima $\subseteq$) $O.R \subseteq Q.G$ koncepcinėje RPQ schemoje specifikuojama tada, kai kiekviena klasės O sudėtinio atributo *R* reikšmė $<O_1, O_2, ..., O_n>$ yra klasės *Q* sudėtinio atributo *G* reikšmė, t. y. iš

$$< O_1, O_2, ..., O_n> \in O.R$$

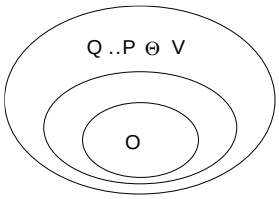
seka

$$< O_1, O_2, ..., O_n> \in Q.G$$



3 pav. Bendrasis poaibio priklausomybės atvejis

Taisyklėje T6 konstruktas *Q..P Θ V* trečioje (atributo) pozicijoje turi aritmetinį palyginimo operatorių *Θ* (=, ≠, <, >, ≤, ≥). Šiuo konstruktu deklaruojamas klasifikacijos sąryšis: klasėje *Q* išskiriamas objektų poaibis, kurio egzempliorių atributas P įgyja reikšmes, užduotas aritmetiniu palyginimo operatoriumi *Θ* atžvilgiu reikšmės *V*. Jeigu *P=V*, tai poaibio kiekvieno egzemplioriaus atributo *P* reikšmė yra lygi *V*. Jeigu $O \subseteq Q..P\ \Theta V$, tai kiekvienas *o*, $o \in O$, bus poaibio $Q..P\ \Theta V$ egzempliorius, o tuo pačiu ir *Q* egzempliorius. Ši situacija grafiškai pavaizduota 4 paveiksle.



4 pav. Taisyklės T6 grafinis atvaizdavimas

Literatūra

[1] B. Paradauskas. Conceptual Object-Relationship-Property Approach: Three Different Interpretations of the Same Entity. *Informatica*, 1995, Vol. 6, No. 4, pg. 497-522.

[2] American National Standards Institute (ANSI) Database Committee (X342). Database Language SQL3. *J. Melton, ed.*, August 1994, 1007 psl.

[3] B. Paradauskas, L. Salelionis. Struktūrinių tipų panaudojimas vartotojų scenarijų sudarymui. *Informacinės technologijos '98. Konferencijos pranešimų medžiaga. Technologija, Kaunas*, 1998, 39-43 psl.

[4] Ch. Fahrner, G. Vossen. A Modular Approach to Relational Reverse Engineering. *University of Muenster, Germany. Bericht*, No. 22/96-I, 43 psl.