KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Arūnas Žukauskas

Objektininių struktūrų atvaizdavimo į reliacinę struktūrą modelis

Magistro darbas

Darbo vadovas
prof. Rimantas Butleris

Kaunas, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Objektininių struktūrų atvaizdavimo į reliacinę struktūrą modelis

Magistro darbas

Vadovas
prof. Rimantas Butleris
2006-12

Recenzentas
dr. Dalius Makackas
2007-01

Atliko
IFM-1/4 gr. stud.
Arūnas Žukauskas
2006-12-15

Kaunas, 2007

SANTRAUKA

Šiame darbe nagrinėjamos problemos, iškylančios dėl semantinės spragos tarp reliacinio ir objektinio principų, bei aptariamas objektinio-reliacinio susiejimo taikymas šios problemos sprendimui. Išanalizavus objektinio-reliacinio susiejimo karkasams būdingas savybes bei egzistuojančius karkasus pasiūlomas modelis, kuris, panaudodamas MVC[1] principus, įgalina realizuoti objektinį-reliacinį susiejimą tokiu būdu, kuris išlaiko dalį abiejų modelių privalumų ir gerai tinka esamų reliacinių struktūrų palaipsniniam pertvarkymui į objektines. Aprašomas suprojektuotas ir realizuotas karkaso prototipas, kuris pritaiko siūlomus principus bei remiantis siūlomu modeliu pademonstruoja eksperimento pagalba fiktyvios dalykinės srities realizaciją.

[1] MVC – Model View Controler – tai principas, kuris suskaido taikomąją programą ar jų sistemą į dalis, atsakingas atitinkamai už duomenų saugojimą, pateikimą ir apdorojimą. Architektūra, kuri yra paremta MVC, vadinama trijų sluoksnių architektūra.

SUMMARY

This work is analyzing problems, arising because of sematical gap between relational and object-oriented approaches and discusses how to utilize object-relational mapping for solving this problem. After analysis of object-relational mapping framework (further – ORM) principles and features of existing ORM frameworks a model is suggested, that allows to implement ORM by utilizing MVP principles in a way that retains major portion of both approach pros and is perfect for transitioning existing relational structures to object-oriented. Further, project and implementation of prototype framework, that uses the following principles and suggested model, is described and utilized as for implementing imaginary domain ORM mapping example application.

TURINYS

1. ĮVADAS

Šiuo metu dauguma taikomųjų programų, skirtų verslo reikmėms, yra kuriamos remiantis objektinio programavimo technologijomis, naudojant tokias aukšto lygio programavimo kalbas kaip C++, Java, C#. Duomenų saugojimui vis dar populiariausiomis išlieka reliacinės duomenų bazės. Taigi, šiuolaikiniams programuotojams yra aktuali objektų išsaugojimo reliacinėse duomenų bazėse problema.

Egzistuoja problema – tarp objektinio programavimo ir reliacinių duomenų bazių yra semantinė spraga [1,1-8 psl.]. Objektinės struktūros yra pagrįstos programinės įrangos inžinerijos principais. Objektai yra klasifikuojami į skirtingus tipus (ang. *coupling*), siejasi ir komunikuoja tarpusavyje (angl. *cohesion*), apgaubia turinį ir funkcionalumą, įgalinant prieiti prie objekto atributų ir veiksmų per interfeise deklaruotas priemones (angl. *encapsulation*). Praktiškai visos aukšto lygio programavimo kalbos palaiko ir kitas svarbias OOP[2] savybes – paveldėjimą ir polimorfizmą. Reliacinės struktūros pagrįstos matematiniais principais tokiais kaip aibių teorija. Šie du požiūriai sprendžiant tam tikras užduotis turi skirtingus privalumus ir trūkumus. Objektinės struktūros įgalina konstruoti taikomąsias programas iš objektų, kurie turi tiek duomenų struktūras, tiek veiksmų, taikomų jiems, aibę, tuo tarpu reliacinės struktūros aprašo tik duomenų saugojimo savybes. Tačiau tai yra tik paviršinis modelių skirtumas.

Kad geriau suvoktume problemą, reikia aptarti pačios semantinės spragos sąvokos prasmę. Sistemų teorijoje semantinė spraga reiškia esybės aprašymo skirtingose semantinėse reprezentacijose skirtumus. Puikus semantinės spragos problemos pavyzdys yra teksto atpažinimas. Programos, kurios atlieka teksto atpažinimą operuoja kitame abstrakcijos lygmenyje, apdorodamos taškus ir jų kombinacijas ir bandydamos nustatyti kokius simbolius tos taškų kombinacijos atitinka. Taigi galima sakyti kad tarp tokių programų darbinių duomenų struktūrų ir nuskaitomo teksto egzistuoja semantinė spraga.

Tiek objektinis, tiek reliacinis modelis yra tam tikros realaus pasaulio dalykinės srities abstrakcija, kuri atvaizduoja tam tikras dominančias esybes ir jų savybes. Šie du modeliai dalykinės srities modeliavimą atlieka ne vienodame semantiniame lygmenyje, galima sakyti kad reliacinis modelis yra arčiau kompiuterio aparatūros nei objektinis modelis.

Šis skirtumas tarp požiūrių matomas susijusių esybių atrinkimo principų skirtumuose: susiję objektų egzemplioriai pasiekiami per ryšius, taigi objektai ir ryšiai tarp jų sudaro grafą, tuo tarpu relia-

[2] OOP – Object Oriented Programming – tai programavimas, kuris remiasi objektais ir jų tarpusavio komunikavimu, skirtingai nei senesnis struktūrinis programavimas, kuris remiasi algoritmais ir duomenų struktūromis.

cinėse duomenų bazėse duomenys randami apjungiant lentelių turinius (angl. *joins*) ir filtruojant rezultatų aibę pagal raktines reikšmes, taigi duomenys saugomi įrašų pavidalu lentelėse ir tam tikri laukai ar jų kombinacijos tarnauja kaip raktai, skirti identifikuoti unikalią eilutę lentelėje.

Šių skirtumų dėka abiejų požiūrių apjungimas ir panaudojimas kartu nėra triviali ir paprastai realizuojama operacija, tam kad sėkmingai atlikti efektyvų objektinių struktūrų atvaizdavimą į reliacines struktūras būtina suvokti abiejų principų esmę ir parinkti priimtinus kompromisus, įgalinančius realizuoti reikiamą taikomąją programą be didesnių trikdžių.

Akivaizdu, jog reikalinga tam tikra programinė aplinka, kuri tarnautų kaip tarpininkas tarp taikomosios programos ir reliacinės duomenų bazės. Žinoma, tam tikroms užduotims galbūt yra prasminga panaudoti objektinę duomenų bazę, tačiau reliacinių duomenų bazių panaudojimas užtikrina suderinamumą su liktinėmis sistemomis ir dėl reliacinių duomenų bazių populiarumo taikomoji programa gali dalintis duomenimis su didesne aibe kitų sistemų.

Tokia aplinka turėtų rūpintis objektų būsenos išsaugojimu (angl. *persistency*), objektų kūrimo ir šalinimo valdymu, konkurencinės prieigos kontrole, prieigos teisių nustatymu pagal vartotojo vaidmenis, egzempliorių identifikatorių ir sąryšio tarp objektų ir jų klasių valdymu. Objektinių ir reliacinių sistemų integracija taip pat gali sąlygoti tokias pozityvias išdavas: tam tikro duomenų bazių funkcionalumo, tokio kaip užklausos ir transakcijų palaikymas, panaudojimas, naudojant tam tikrus atvaizdavimo sprendimus dar organizuotesni duomenų saugojimas duomenų bazėje.

Šiame darbe kaip tik ir bus nagrinėjami objektinių struktūrų atvaizdavimo į reliacines struktūras karkaso principai (angl. *object-relational mapping framework* (toliau ORM) *principles*), suprojektuotas ir realizuotas prototipinis karkasas kuris demonstruos kaip ORM gali būti panaudotas praktiškai čia aprašytai problemai spręsti. Taip pat bus pasiūlytas objektinio-reliacinio susiejimo modelis, apjungiantis teigiamas savybes iš abiejų modelių.

2 skyriuje atliekama ORM paskirties ir metodų analizė.

3 skyriuje nagrinėjamas siūlomas objektinio-reliacinio susiejimo modelis ir aprašomas projektuojamas ORM karkasas.

4 skyriuje aprašoma prototipinė ORM karkaso realizacija

5 skyriuje aprašoma eksperimentinė programa, naudojanti sukurtą ORM karkasą.

6 skyriuje analizuojami darbo rezultatai ir pateikiamos išvados.

2. ORM PASKIRTIES IR METODŲ ANALIZĖ

Šiame skyriuje nagrinėjamos problemos, iškylančios derinant reliacinį ir objektinį modelius, pagrindžiama kodėl objektinis modelis geriau tinka projektuojant programinę įrangą, trumpai apžvelgiamos reliacinės ir objektinės duomenų bazės, analizuojama kaip panaudoti reliacines duomenų bazes objektams saugoti naudojant ORM karkasą kaip sąsają, įgalinančia dirbti su reliacinėmis duomenų bazėmis tarsi jos būtų objektinės.

2.1. Semantinė spraga tarp objektinio ir reliacinio modelių

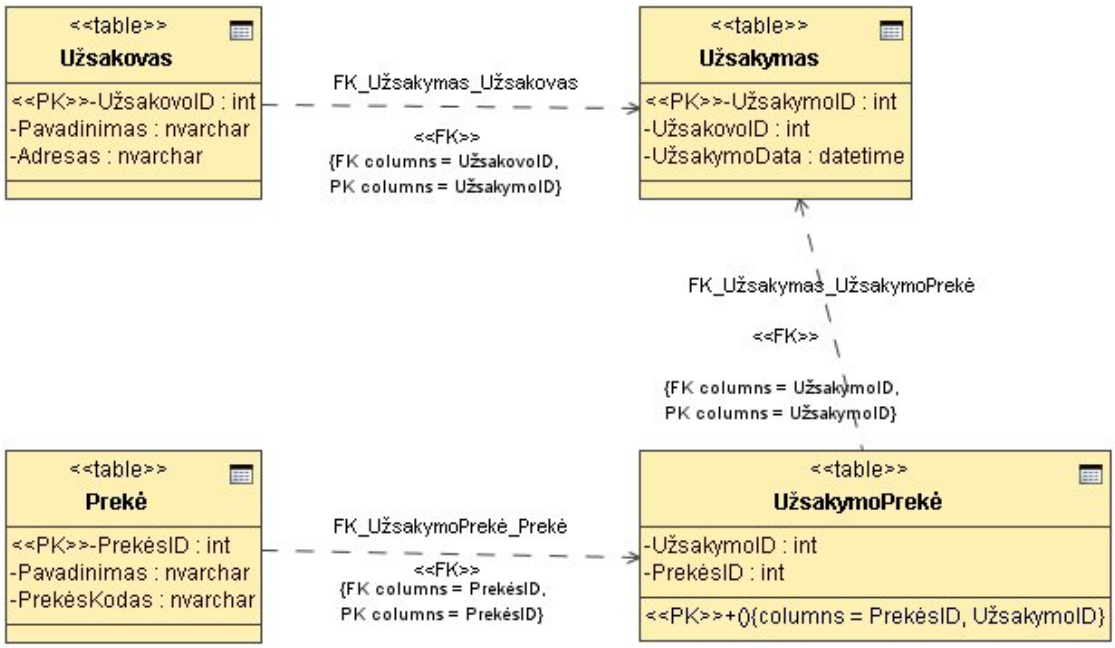
Trumpai apibendrinant įvade paminėtus teiginius reliaciniai modeliai remiasi matematiniais principais, objektiniai – programavimo inžinerijos metodais. Nors ir iš pirmo žvilgsnio gali atrodyti kad šie požiūriai yra panašūs, giliau paanalizavus juos tampa aišku kad jų pagrindai yra fundamentaliai skirtingi.

Šis skyrius remiasi Ambler pastebėjimais, publikuotais IBM programuotojų portale [2]. Ten, kur remiamasi kitų autorių mintimis ar jų nuomonė skiriasi, yra pažymima atskirai.

Panagrinėsime du modelius - UML fizinį duomenų modelį[3] ir klasių diagramą.

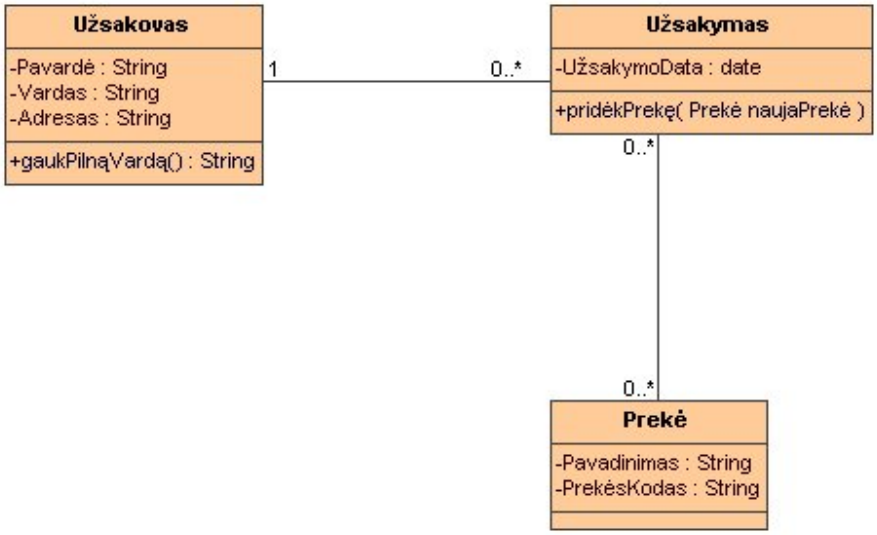
Panagrinėkime tipinę dalykinę sritį - užsakymų registravimą. Užsakymas yra pateikiamos užsakovo, jame nurodant kelias prekes. Šios dalykinės srities principinis reliacinis modelis pateikiamas 1 pav.

[3] Kadangi duomenys fiziškai saugomi reliacinėje duomenų bazėje, šis modelis čia atitiks duomenų bazių schemos diagramą. Toliau darbe ir bus naudojama ši sąvoka.



1 pav. Fizinis duomenų modelis - reliacinė schema

Ta pati dalykinė sritis atvaizduota objektiniame modelyje pateikiama 2 pav.



2 pav. Klasių diagrama

Požiūris, jog taikomosios aplikacijos kūrimas turi remtis duomenų modeliu, vis dar yra paplitęs. Reliacinis modelis ne visuomet yra suprojektuojamas idealiai pagal reliacinius principus, daugeliu atveju dėl praktinių sumetimų yra padaroma tam tikrų pažeidimų ar denormalizavimų. Šiame darbe bus remiamasi praktiniais reliacinių duomenų bazių schemų patarimais, aprašytais [3, 13 psl]. Tačiau agile development principams toks požiūris nelabai tinka [1]. Objektai yra laisvesnė konstrukcija nei duomenų struktūros ir objektiniai modeliai yra atsparesni reikalavimų kitimui. Kuriant aplika-

cijas duomenų pagrindu būtina pradžioje numatyti visus duomenis ir suprojektuoti konceptualų duomenų modelį ir išreikšti jį pvz. ER diagrama[4]. Duomenų srautų diagramos taip pat glaudžiai siejasi su operuojamų duomenų tipais. Paprastai konceptualus duomenų modelis programinės įrangos kūrimo proceso eigoje transformuojasi į fizinį duomenų modelį pvz. reliacinėje duomenų bazėje ar failuose. Natūralu, jog tokių taikomųjų programų klasių diagramos remiasi duomenų modeliais, dažnai net kiekviena lentelė turi save atitinkančią klasę.

Tuo tarpu Agile programavime[5] remtis į duomenų struktūras nepatartina. Objektai suteikia didesnę abstrakcijos laisvę, taigi reikalavimai sistemai turi daugiau erdvės kitimui. Be to pritaikant laiko patikrintus projektavimo šablonus (angl. *Design patterns*[6]) galima suprojektuoti lankstesnę ir aiškesnę architektūrą.

Panagrinėkime panašumus tarp dviejų diagramų. Abi jos apibrėžia panašią struktūrą, duomenų bazės schema (1 pav.) vaizduoja duomenų bazės lenteles ir ryšius tarp jų, klasių diagrama (2 pav.) klases ir ryšius tarp jų. Tačiau klasių diagramoje taip pat galima laisvai aprašyti veiksmus, kuriuos galima atlikti su klasių objektais kai tuo tarpu duomenų bazės diagramoje tai padaryti nėra taip paprasta, nors išsaugotos DBVS[7] procedūros gali atlikti operacijas su lentelėmis, procedūros nepriklauso lentelėms kaip kad metodai priklauso klasėms.

Kokie esminiai skirtumai gaunasi tarp diagramų [5]:

- Projektuojant klasių diagramą ir remdamasis objektinio programavimo principais tokiais kaip projektavimo šablonai (angl. *Design Patterns*) analitikas sukuria modelį, kuris turi elementų kuriuos negalima ar neprasminga atvaizduoti duomenų modelyje.
- Reliacinėms schemoms išgryninti naudojamas duomenų normalizavimas, klasių diagramoms – objektinis normalizavimas.
- Klasių diagramos gali atvaizduoti sudėtingesnius ryšius, pvz. Daug-Su-Daug turi būti išreiškiamas reliacinėje schemoje Vienas-Su-Daug ryšiais ir tarpinės lentelės pagalba.

[4] ER diagrama – santr. Entity Relationship diagram – tai diagrama, kuri vaizduoja esybes ir santykius tarp jų. ER diagrama yra abstraktesnė ir labiau koncepcinė nei klasių ir duomenų bazės diagramos.
[5] Agile programavimas – tai programavimo metodologija, pasižyminti lankstumu, atsparumu reikalavimų pokyčiais, orientuota į vartotoją o ne į projekto planą.
[6] Design patterns – projektavimo šablonai. Tai nusistovėję objektinio programavimo šablonai ir praktiniai sprendimai, panaudojami dažniausiai pasitaikančioms problemoms spręsti.
[7] DBVS – santr. Duomenų Bazių Valdymo Sistemos

- Paprastai klasių diagramose objektai neturi jokių dirbtinių identifikatorių[8]. Objektas gali būti identifikuojamas programoje nenaudojant laukų, tuo tarpu norint sėkmingai išsaugoti įrašą į duomenų bazę yra reikalinga informacija, kuri unikaliai identifikuoja įrašą.

Kaip spręsti šią dilemą?

Kadangi tiek vienas tiek kitas modelis yra reikalingi, būtina panaudoti priemonę, kuri tarnauja kaip tarpininkas tarp jų, ir kiek galima labiau automatizuotai atlieka visas susiejimo operacijas. Tokia priemonė sumažintų semantinę spragą tarp šių modelių ir suteiktų tam tikrų privalumų dėl galimybės panaudoti abiejų modelių stipriąsias puses. Tačiau kaip taisyklė papildomų sluoksnių įvedimas į sistemą didina jos suminį sudėtingumą ir klaidų tikimybę.

2.2. Objektų atvaizdavimo į reliacinę duomenų bazę ypatumai

Paanalizuosime kaip praktiškai atliekamas objektų atvaizdavimas į reliacinę duomenų bazę.

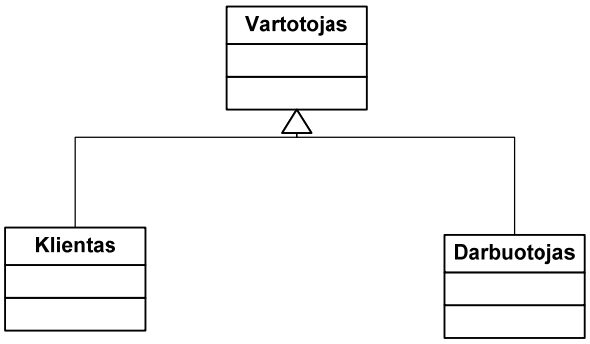
2.2.1. Atributų atvaizdavimas į laukus

Pirmoji problema yra atributų atvaizdavimas į laukus. Klasės atributas gali atsivaizduoti į vieną ar kelis laukus arba būti neatvaizduojamas iš viso – kai kurie atributai, kurie priklauso nuo kitų atributų, grąžina apskaičiuotą vykdymo metu reikšmę. Be to kai kurių atributų tipas gali būti taip pat objektas (Shlegelmilch panagrinėja šią objektų ypatybę ir jos poveikį reliacinei schemai detaliau [6, 3 psl]). Sudėtingų atributų reikšmės gali būti saugomos net keliose lentelėse (pvz. HTML dokumento tipo atributas gali atsivaizduoti į kelis įrašus failų lentelėje). Galimas ir atvirkštinis variantas kai kelių atributų reikšmės saugomos viename atribute bet keliuose įrašuose.

[8] Reliacinėms duomenų bazėms būtina, kad kiekvienas įrašas turėtų lauką ar laukus, saugantį unikalią reikšmę, šis laukas tarnauja kaip įrašo identifikatorius. Net ir tokiu atveju kai dalykinėje srityje objektas neturi identifikuojančio atributo jo sėkmingam panaudojimui DBVS būtinas raktinis laukas; tokiu atveju kai raktas neturi prasmės dalykinėje srityje jis vadinamas dirbtiniu.

2.2.2. Paveldėjimo realizavimas

Antroji problema yra paveldėjimo realizavimas. Yra trys galimi paveldėjimo realizavimo būdai, panagrinėsime kiekvieną iš jų. Nuo to, kaip realizuojamas paveldėjimas, priklauso visos sistemos dizainas, taigi apsisprendimas kokį principą panaudoti turi didelę įtaką ir vėliau pakeisti būdą nėra paprasta. Panagrinėkime primityvią klasių hierarchiją, pateiktą 3 pav.



3 pav. Klasių diagrama objektų atvaizdavimui į duomenų bazę iliustruoti

2.2.2.1. Pirmas būdas: Vienos lentelės panaudojimas visai iš vienos bazinės klasės paveldinčių klasių hierarchijai

Jeigu naudojama viena lentelė visai klasių hierarchijai, tuomet vienas lentelės laukas naudojamas klasės tipui saugoti.

Privalumai:

- paprasta realizuoti;
- lengva įgyvendinti polimorfizmą ir pakeisti objekto klasę;
- lengva nustatyti objekto klasę;
- visi duomenys randasi vienoje lentelėje todėl paprastesnė duomenų paieška.

Trūkumai:

- lentelė turi turėti visų hierarchijos klasių atributus;
- atributai, kurie neturi prasmės tam tikriem objektam, liks neužpildyti;
- lengviau padaryti klaidą dirbant su atributais naudojant SQL nes galima netyčia apdoroti ne to tipo egzempliorius kurio norima;
- sudėtinga realizuoti variantą kai klasės gali paveldėti kelias superklases.

4 pav. paveiksle pateikiamas pavyzdys, kuris atvaizduoja duotąją klasių hierarchiją į duomenų bazės schemą naudojant šį būdą:

vartotojas	
PK	**id**
	objekto tipas vartotojo vardas darbuotojo tabelio numeris kliento firma

4 pav. Duomenų bazės schema naudojant kompozicinę lentelę

2.2.2.2. Antras būdas: Po vieną lentelę kiekvienai galinei klasei

Galima kiekvieną galutinę klasę[9] atvaizduoti į atskirą lentelę su savo ir savo superklasių atributais.

Privalumai:

- paprastai galima išgauti duomenis;
- skirtingų tipų objektų raktai priklauso atskirtoms aibėms.

Trūkumai:

- kadangi bazinių klasių atributai išbarstyti per lenteles, prireikus papildyti atributais superklasę reikėtų apdoroti visas lenteles;
- keičiant objekto tipą reikės perkelti duomenis iš vienos lentelės į kitą;
- sunku užtikrinti duomenų integralumą (problemos su objektais, priklausančiais kelioms klasėms).

5 pav. pateikiamas klasių hierarchijos atvaizdavimas į duomenų bazės schemą taikant šį metodą ir laikant kad klasė vartotojas yra abstrakti.

klientas	
PK	**id**
	vartotojo vardas firma

darbuotojas	
PK	**id**
	vartotojo vardas tabelio numeris

5 pav. Duomenų bazės schema naudojant hierarchijos išplokštinimą

2.2.2.3. Po vieną lentelę kiekvienai klasei

Efektyviausias sprendimas yra kiekvieną hierarchijos klasę atvaizduoti į atskirą lentelę. Kiekviena lentelė saugo objekto pirminius raktus bei bazinės klasės egzemplioriaus dalies raktus arba išvestinių klasių dalies egzempliorių raktų aibės gali būti bazinių klasių egzempliorių dalies raktų poaibis. Objektai gali priklausyti keliom hierarchijos klasėm šių klasių duomenis saugant atskirose lentelėse. Pagrindinis šio sprendimo privalumas yra tas, kad jis yra artimiausias objektiniam variantui ir

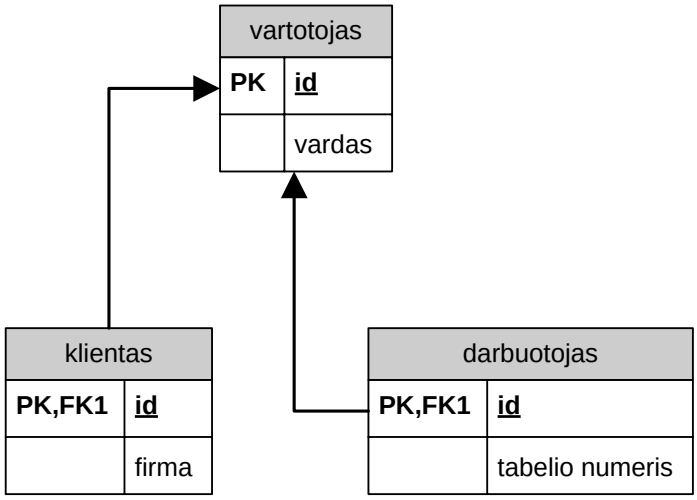
[9] Galutinė klasė – tai klasė, kuri netarnauja kaip bazinė klasė kitai klasei, kitaip tariant hierarchijos lapas. Ši sąvoka naudojama šiame kontekste aprašyti klasei, kuri paveldi iš bazinės klasės ir gali turėti egzempliorius.

natūraliausiai realizuoja objektinio modelio elementus. Šiuo atveju polimorfizmas nesunkiai įgyvendinamas, nesunku pridėti ir šalinti klases.

Trūkumai:

- duomenų bazėje duomenys išbarstyti dideliame kiekyje lentelių, dėl to sulėtėja informacijos skaitymas ir rašymas,
- sunkiau išgauti konkretaus objekto parametrus.

6 pav. pateikiama duomenų bazės schema, kuri realizuoja duotąją klasių diagramą panaudojant šį būdą.



6 pav. Duomenų bazės schema kiekvieną klasę atvaizduojant į atskirą lentelę

2.2.3. Ryšių atvaizdavimas

Vien objektų išsaugojimo į duomenų bazę nepakanka – reikia dar išsaugoti ryšius tarp objektų kad prireikus būtų galima atstatyti visą objektų grafą. Yra keturi galimi ryšių tarp objektų tipai: paveldėjimas, sąryšis, agregacija ir kompozicija (Ne visi autoriai išskiria šiuos ryšius vienodai, G.Booch neišskiria agregacijos ir kompozicijos [9, 95-101 psl]). Paveldėjimo ryšį jau paanalizavome.

Kompozicija ir agregavimas iš esmės skiriasi tik objekto gyvavimo ciklo valdymu. Jeigu objektas A yra įkomponuotas į objektą B, jis sukuriamas kartu su objektu B ir šalinamas kartu su objektu B. Taigi duomenų bazėje B lentelių įrašų manipuliavimas turi iššaukti kaskadines operacijas A lentelėms. Agregacijoje objektas B saugo tik nuorodą į objektą A ir šie objektai gali egzistuoti nepriklausomai. Sąryšiai apibrėžia dalykinės srities ryšius, kurie neapibrėžia koks objektas į kokį objektą įeina. Sąryšiai nebūtinai atvaizduojami į reliacinės duomenų bazės ryšius.

Reliacinės duomenų bazės palaiko tik one-to-many (1..n) tipo ryšius, none-or-one-to-many (0..n) ryšiai yra dalinis atvejis kuris išgaunamas leidžiant raktui įgyti NULL reikšmę. Kadangi many-to-many ryšio reliacinėje duomenų bazėje tiesiogiai įgyvendinti negalima, tam yra sukuriama ir panaudojama asociacijos lentelė (Išsamiau ši problema analizuojama [10, 3.25-3.27]). Karkasas gali suge-

bėti automatiškai naudotis ryšio lentelėmis realizuojant objektų many-to-many ryšius, kitaip tariant tarpinės lentelės gali neatsivaizduoti į jokius objektus.

2.3. Alternatyvos ORM ir jų tinkamumas šiai problemai spręsti

Jeigu nenaudojamas ORM, tuomet galima sumažinti semantinę spragą naudojant objektinę duomenų bazę. Tačiau objektinės duomenų bazės dar nėra paplitusios ir jų sąsajos ar OQL dar nėra standartizuota (Objektinių DB savybės aprašomos [11]). Dėl reliacinių duomenų bazių populiarumo objektinės duomenų bazės nėra labai geras pasirinkimas, nebent duomenų bazė būtų skirta tik vienai taikomajai programai. Kai kurios reliacinės duomenų bazės turi tam tikrus sluoksnius, emuliuojančius objektinės duomenų bazės funkcionalumą. Šis sprendimas yra patrauklesnis, nes abu panaudojimo principai išlieka pasiekiami. Tačiau tokiais atvejais, kada tokio funkcionalumo nėra, reikalingas programinis sprendimas, išpildytas ORM karkaso pavidalu. Šiame darbe panagrinėsime tokio sprendimo specifiką, nes kiti sprendimai rečiau paplitę. Sprendimų skirtumai (remiantis [7, 25 psl]):

1 lent. OO sąsajos suteikimo sprendimai

Savybė	Reliacinė DB	Objektinė DB	Reliacinė DB su objektinės DB sąsajos palaikymu	Programinis sprendimas (ORM karkasas)
Galima naudotis esamais reliaciniais duomenimis?	✓Taip	×Ne	✓Taip	✓Taip
Palaikomas reliacinis API (SQL)?	✓Taip	×Ne	✓Taip	✓Taip
Reikalingas susiejimo aprašas?	✓Taip	×Ne	×Ne	✓Taip
Objektinis panaudojimas yra vartotojui trivialus?	×Ne	✓Taip	✓Taip	✓Taip
Egzistuoja vykdymo laiko apkrova?	✓Taip	×Ne	×Ne	✓Taip
Tik viena loginė DB/Schema?	✓Taip	✓Taip	×Ne	✓Taip
Išaugę reikalavimai saugojimo apimčiai?	×Ne	×Ne	✓Taip	×Ne

2.4. Svarbiausių ORM karkaso funkcijų analizė

Paanalizuosime standartinio ORM karkaso sudėtines dalis ir jų atliekamas funkcijas, bei kaip šios funkcijos pasitarnauja objektinio-reliacinio susiejimo problemai spręsti (Pagal [4]).

Esminės ORM karkaso funkcijos yra tokios:

- Išsaugojimo (persistency) sluoksnis įgalina objektus išsaugoti reliacinėse duomenų bazėse. Šis sluoksnis tarnauja kaip API. Tai karkaso sąsaja, kuri įgalina pasinaudoti visomis karkaso galimybėmis.
- CRUD operacijų valdiklis apdoroja API funkcijas, kuriomis prašoma atlikti objektų sukūrimo, išgavimo, išsaugojimo ir trynimo veiksmus. Visos CRUD operacijos turi būti palaikomos kiekvienam objektui ir sąlygoti atitinkamas duomenų bazės operacijas. CRUD operacijų valdiklis pasinaudoja SQL kodo generatoriumi. Nebūtinai visos operacijos turi atsispindėti į duomenų bazę – CRUD valdiklis apdoroja ir įvairius kitus scenarijus, tokius kaip kad prašymas grąžinti jau užkrautą objektą arba jau ištrinto objekto pakartotinis trynimas kuriems išpildyti kreipiniai į DBVS yra nereikalingi.
- SQL kodo generatorius generuoja atitinkamus INSERT, UPDATE ir kt. SQL sakinius CRUD operacijoms realizuoti tam tikrai DBVS.
- Objektų atributų susiejimo su duomenų bazės lentelių laukais valdiklis skaito susiejimo konfigūracijas ir remiantis jomis objektų ryšių apdorojimo logika gali atrinkti susijusius objektus.
- Tipų konvertavimo valdiklis užtikrina kad DBVS duomenų tipai būtų atvaizduoti į atitinkamus programinius tipus.
- Pakeitimų valdiklis registruoja pakeitimus objektuose ir pagal tai generuoja atitinkamas CRUD operacijas. Šis valdiklis priskiria objektams atitinkamas būsenas (Current, New ir t.t.).
- Identifikavimo valdiklis susieja objektus su jų identifikatoriais.
- Transakcijų valdiklis įgalina panaudoti transakcijas saugant objektus.

Panagrinėsime šias funkcijas detaliau atskiruose skyriuose.

2.4.1. Išsaugojimo sluoksnis

Išsaugojimo sluoksnis programuotojui turi suteikti tam tikrą sąsają, įgalinančią atlikti visas reikiamas funkcijas. Ši sąsaja turi nepriklausyti nuo duomenų bazės, kuri yra naudojama duomenims saugoti, kitaip tariant, ji turi būti universali. Dar daugiau, aplikacija gali naudoti kelias skirtingas duomenų bazes per jos gyvavimo laiką. Būtina taip pat atkreipti dėmesį į tai, jog reikalavimai aplikacijai gali kisti ir šie reikalavimų kitimai gali sąlygoti duomenų modelio kitimą, taigi turės kisti ir schema, ir objektų klasės. Vadinasi išsaugojimo sluoksnis turi sugebėti dinamiškai prisitaikyti prie schemos kitimo, todėl būtina kiek įmanoma daugiau aprašų iškelti už aplikacijos ribų kad jie nebūtų įkompiliuojami o nuskaitomi programos vykdymo metu.

Kadangi operacijos su duomenų baze yra lėtesnės nei su atmintimi, būtina minimizuoti informacijos mainų su duomenų baze apimtį ir stengtis siųsti tik tą informaciją kuri yra būtina ir sumažinti SQL užklausų kiekį iki minimumo. Teoriškai DBVS užklausą gali vykdyti gana ilgai, todėl išsaugojimo sluoksnis sistemose, kuriose reikalingas didelis našumas, turi būti panaudojimas asinchroniškai o tai komplikuoja visą mechanizmą.

2.4.2. CRUD valdiklis

Išsaugojimo mechanizmui reikalinga papildoma informacija apie objektus, kuri yra būtina jų sėkmingam išsaugojimui, taip pat jis suteikia funkcijas CRUD operacijoms atlikti. Priklausomai nuo pasirinktos programavimo aplinkos galimybių galimi keli variantai. Galima visas klases, kurios pasinaudos išsaugojimo mechanizmo funkcijomis, paveldėti iš tam tikros bazinės klasės pvz. PersistentObject, kuri ir atliks visas išsaugojimo mechanizmui reikalingas funkcijas. Jeigu programavimo aplinka suteikia priemones išgauti klasių ir objektų metaduomenis (angl. *reflection*) tuomet išsaugojimo mechanizmas gali visą reikalingą informaciją saugoti savo vidinėse struktūrose reflekcijos pagalba susiedamas objektus su jomis. Tokiu atveju išsaugojimo mechanizmo realizacija neiškelia jokių reikalavimų išsaugojamų klasių struktūrai bet klasės neturi jokių CRUD metodų išsaugojimo operacijoms atlikti, tam naudojamos išorinės valdiklių klasės.

Deja tam tikram funkcionalumui realizuoti kai kada būtina jog klasės informuotų karkasą apie savo savybes ar naudotų karkaso duomenų tipus. Tai dažniausiai reikalinga realizuojant ryšio objektus. Ryšio objektais šiame dokumente vadinsime objekto atributus kurių tipas yra kitas išsaugojamasis objektas (dažniausiai tėvinis objektas) arba jų kolekcijas (dažniausiai vaikiniai objektai). Tokiu atveju yra du sprendimai: arba klasės naudoja karkaso funkcionalumą (kitaip tariant yra *framework aware*) arba karkasas generuoja proxy[10] objektus kas yra sudėtingai realizuojama (Ramanathan įrodo kad įmanomas automatinis generavimas, kuris pertransformuoja reliacinį modelį į objektinį teisingai [7,4 psl]).

CRUD operacijų realizavimas taip pat iškelia šiek tiek iššūkių. Kad būtų paprasčiau valdyti kodą būtina realizuoti SQL generavimą viename kodo modulyje ir užtikrinti jo universalumą. CRUD valdiklis turi sugebėti sugeneruoti SQL užklausas bet kokiai išsaugomai klasei, taigi jis turi turėti visus reikiamus metaduomenis, kurie būtini sėkmingam šios operacijos atlikimui. Deja, universalumas dažnai turi įtakos spartumui – išgautos SQL užklausos gali būti neoptimizuotos nes jos bus generuojamos pagal tam tikrą fiksuotą algoritmą, norint šias užklausas optimizuoti reikia smarkiai išplėsti CRUD valdiklį tuo padidinant jo sudėtingumą. Problema yra ta, jog ne visuomet yra dirbama su vie-

[10] Proxy yra projektavimo šablonas, kurio esmė yra klasių apgaubimas kitos klasės viduje siekiant suteikti tam tikrą sąsają.

nu objektu. Jeigu karkaso yra prašoma atrinkti objektų, atitinkančius tam tikrus kriterijus, aibę, jis turi sugebėti taip suformuoti SQL SELECT operacijai vykdyti kad šią paiešką atliktų DBVS. Tokios užklausos sukelia problemų kai yra naudojamas objektų kešavimas – vieni tinkami objektai jau bus atmintyje o kiti dar ne, jeigu nėra įmanoma nustatyti kurie tinkami objektai jau yra užkrauti o kurie ne gali tekti atisiųsti iš duomenų bazės juos visus iš naujo.

Vis dėlto CRUD operacijų realizavimas naudojant fiksuotą sąsają suteikia didžiulį privalumą – programuotojui nebūtina rūpintis išsaugojimo subtilybėmis ir jis gali patikėti šią funkciją karkasui. Dar daugiau, polimorfizmo dėka galima CRUD tas pačias operacijas iškviesti skirtingų klasių objektams nežinant konkrečių klasių, kitaip tariant, jeigu objektas yra išsaugojamas norint atlikti CRUD operacijas yra nesvarbu kokia jo klasė.

2.4.3. SQL valdiklis

INSERT, UPDATE, DELETE dažniausiu atveju yra paprastos operacijos kurias galima įvykdyti nesigilinant į klasės struktūrą. SELECT, kita vertus, dažnai naudojama atrinkti objektus, tenkinančius tam tikrus kriterijus. Kyla natūralus klausimas – kokiu gi būdu reikia aprašyti šias užklausas.

Galima naudoti SQL, bet SQL yra per daug priklausoma nuo fizinės duomenų bazės schemos. Be to SQL nėra objektinė kalba.

Dažnai profesionalūs karkasai suteikia galimybę naudotis OQL ar panašia kalba.

OQL (Object Query Language) yra specializuota užklausų kalba, skirta objektiškai orientuotoms duomenų bazėms. OQL yra sukurta SQL pagrindu. Kadangi OQL yra labai sudėtinga, joks gamintojas nepalaiko visų jos galimybių, todėl yra daug dalinių OQL variantų (bendros problemos, formuojant užklausas objektinėms duomenų bazėms analizuojamos [12]).

Smulkesni karkasai gali suteikti galimybę suformuoti užklausas pasinaudojant specialia mini kalba ar suformuoti jų DOM iš užklausos objektų (loginių sąlygų ir jų junginių).

2.4.4. Susiejimo valdiklis

Ne visuomet tarp objekto atributų ir lentelės laukų yra 1:1 pobūdžio sutapimas. Kai kada ir klasės gali apimti kelias lenteles ir lentelė gali saugoti kelių klasių duomenis. Apskritai, automatinis objektų ir lentelių susiejimas yra labai ribotas, todėl yra prasminga kažkokiu tai būdu deklaruoti kaip objektų elementai siejasi su tam tikrais duomenų bazės elementais.

Yra du sprendimai: deklaratyvūs metaduomenys ir išoriniai deklaravimo failai (dažniausiai XML). Metaduomenys panaudojami programavimo aplinkose kaip kad .NET kurios įgalina priskirti papildomą informaciją prie kodo elementų tokių kaip klasių ir jų laukų. Išoriniai deklaravimo failai yra universalesnis sprendimas, be to, susiejimą atlikus tokiu būdu galima pakitus fizinei schemai pakoreguoti deklaravimo failus neperkompiliuojant programos.

Karkasui būtina žinoti ne tik kaip lentelių laukai transformuojasi į klasės atributus, bet taip pat kaip objektai yra identifikuojami ir kaip siejasi tarpusavyje. Kadangi tiesiogiai visais atvejais bet kokio tipo ryšių transformuoti į reliacinių duomenų bazių palaikomus 1:n tipo ryšius negalima, karkasui reikia nurodyti kokios tarpinės lentelės saugo papildomą ryšio informaciją ir kaip šis ryšys veikia.

Paprastai objektai gali turėti atributus, kurių tipas yra kita išsaugoma klasė arba jų kolekcija. Karkasas turi sugebėti automatiškai rasti ir užkrauti susijusius objektus. Čia ir vėl yra du galimi sprendimai. Galima užkraunant objektą užkrauti ir susijusius jo objektus. Šis sprendimas gali būti bendru atveju nenašus. Kitas sprendimas – užkrovimas pareikalavus (angl. *lazy loading*) – yra našesnis bet sudėtingiau realizuojamas, kadangi tada būtina jog kreipiantis į objekto atributą būtų užkraunamas susijęs objektas. Norint tokį funkcionalumą realizuoti reikia kad šis atributas būtų pakankamai gudrus ir suprastų kad reikia surasti ir sugrąžinti susijusį objektą, taigi ši klasė turi žinoti apie karkaso egzistavimą (angl. *framework aware*).

.NET karkase tai padaryti yra lengviau nes galima čia pritaikyti sintaksinę struktūrą property, kuri panaudojimo atveju yra atributas, o realizacijos atžvilgiu metodas.

Daugiau problemų sukelia objektų kolekcijos. Užkrauti jų turinį nepakanka. Gali tekti atfiltruoti kolekcijos objektus pagal tam tikrus kriterijus ar modifikuoti jos turinį. Jeigu padedame objektą į kolekciją ar šaliname iš jos tai realiai kuriame ir šaliname ryšius tarp šių objektų. Vadinasi, saugojimo metu karkasas turi žinoti kad šios operacijos buvo atliktos ir atitinkamai atvaizduoti jas į duomenų bazę. Taigi, kad šį funkcionalumą realizuoti patogu turėti specializuotą objektų kolekcijų klasę. Čia vėl yra problema – klasės palieka priklausomos nuo karkaso ir negali egzistuoti vienos pačios.

Ne visuomet ir atributų laukų reikšmes galima išsaugoti į lenteles tiesiogiai. Gali būti jog tipai, naudojami duomenų bazėje ir programoje yra nesuderinti ar neturi atitikmenų. Pavyzdžiui, vektorinis tipas gali būti apibrėžtas programoje o duomenų bazėje nieko panašaus nėra. Viena iš dažniausiai pasitaikančių problemų šioje srityje kuomet naudojamos reliacinės duomenų bazės – NULL reikšmė. Reliacinės duomenų bazės leidžia bet kokio tipo laukams įgyti NULL reikšmes, tuo tarpu ne visi baziniai programiniai tipai gali reprezentuoti šią reikšmę. Pavyzdžiui, sveikųjų skaičių tipas *integer* apima visus sveikuosius skaičius nuo *–MaxInt* iki *+Maxint* ir NULL į šią aibę neįeina. Čia ir vėl yra du galimi sprendimai – arba apibrėžti išvestinį tipą, kuris gali palaikyti NULL reikšmę, arba saugoti statusą ar atributo reikšmė yra NULL atskirai.

Karkasas turėtų realizuoti tipų konvertavimo/apdorojimo valdiklį kuris spręstų šias problemas. Priedo šis valdiklis turėtų pasirūpinti objektų laukų inicializavimu pradinėmis reikšmėmis ir apribojimų, taikomų duomenų bazės laukams užtikrinimui programiniame lygmenyje.

Viena iš sudėtingiausių karkaso problemų yra pakitimų aptikimas ir išsaugojimas [8, 4 skyrius]. Ir vėl čia yra du galimi sprendimai: saugoti pakitimus pareikalavus arba iš karto. Paprastai pakeitimų saugojimas pareikalavimus yra prasmingesnis – galima vienu ypu išsaugoti visus pakitimus objekte, taip pat saugojimo sukoncentravimas į vieną operaciją įgalina tinkamiau informuoti vartotoją apie tai ką daro programa. Pažymėjimą, jog yra pakitimų, gali inicijuoti pati pakitimo operacija arba saugojimo valdiklis gali juos nustatyti sulyginęs objektą su originalia kopija. Pats pakitimų nustatymas gali būti nevienodo detalumo. Galima sekti kiekvieno atributo pakeitimus arba bendrai viso objekto, arba taikyti abu šiuos požiūrius. Svarbūs aspektai, kurie reikšmingi projektuojant pakeitimų valdiklį:

- Dauguma objektų yra modifikuojami labai retai;
- Nepakitusių objektų išsaugojimas yra beprasmis laiko švaistymas;
- Labai lengva pamiršti išsaugoti objektus. Todėl patogu jog pakitimų valdiklis bandytų saugoti visus objektus, ir aptikęs kad tam tikri objektai nepakitę juos paprasčiausiai praleistų;
- Turi būti galimybė atšaukti objekto pakitimus. Tam reikalinga, kad karkasas saugotų originalius objektus.

2.4.5. Identifikavimo valdiklis

Kita problema yra objekto identifikavimas. Programavimo aplinkoje objektams identifikatoriai nereikalingi – vykdymo aplinka (runtime) atskiria objektus pagal jų saugojimo atmintyje vietą. Tuo tarpu duomenų bazėms reikalinga skaliarinė reikšmė, kuri unikaliai identifikuoja objektą. Jeigu dalykinėje srityje tam tikras atributas tarnauja kaip identifikatorius, natūralu tokius atributus naudoti kaip pirminius raktus. Tačiau šis požiūris yra neskalabilus. Pirma, toks atributas ne visada gali tarnauti kaip identifikatorius, ateityje gali atsirasti papildomų atributų, kurie įeina į rakto sudėtį. Paprasčiausias pavyzdys kada tokia situacija susidaro – auditavimo įvedimas. Tuomet unikalus objektas identifikuojamas tokio atributo ir revizijos numerio kombinacija. Antra, jeigu raktas turi prasmę dalykinėje srityje, gali prireikti keisti jo reikšmę. Trečia, jeigu raktas nėra skaičius kris sistemos našumas. Taigi prasmingiausias sprendimas yra dirbtinis fiksuoto tipo raktas, kuris net nebūtinai turi turėti atitikmenį klasėje – jis gali būti saugomas karkase prie objekto metaduomenų. Automatiniai raktai gali būti dviejų tipų – *identity* ir *guid*. Identity yra automatiškai didėjantys skaičiai, guid – pseudoatsitiktinės binarinės reikšmės. Guid yra lėtesni, bet jie įgalina apjungti duomenis iš kelių duomenų bazių išvengiant raktų persiklojimo.

2.4.6. Transakcijų valdiklis

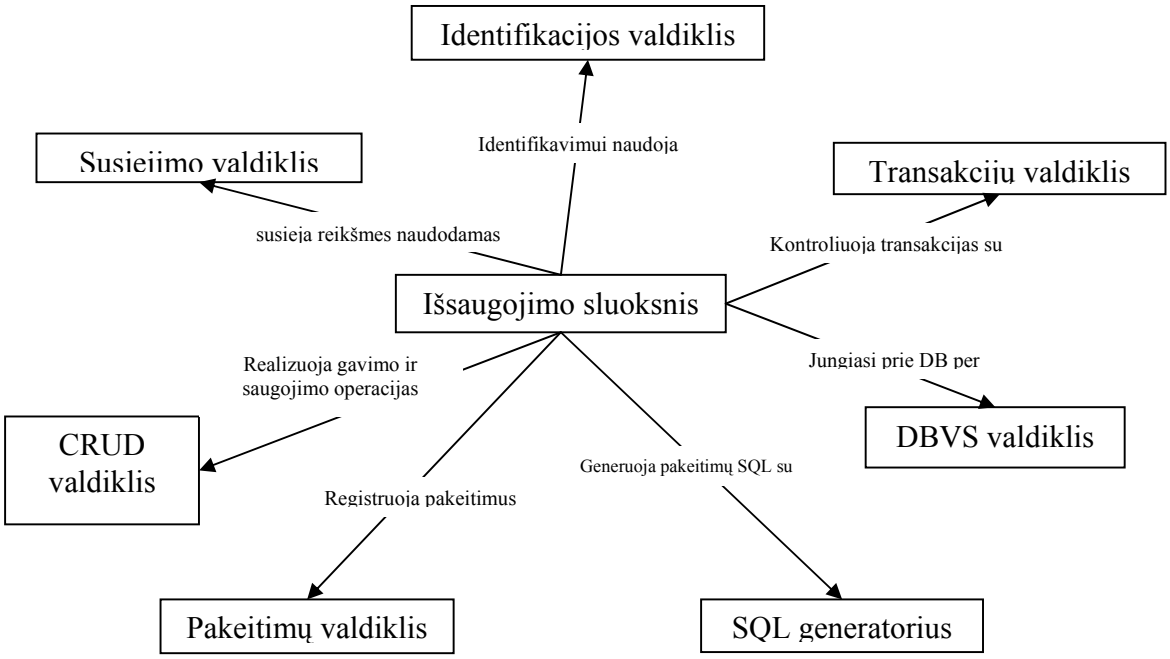
Transakcijų palaikymas yra svarbi karkaso savybė. Transakcijos užtikrina kad susiję objektai yra arba visi tvarkingai išsaugomi arba visi neišsaugomi. Kai kada sunku nustatyti kurie objektų ryšiai

sąlygoja transakcijų naudojimą o kurie ne. Savaime suprantama, tokios situacijos kaip tik pusės objekto išsaugojimas turi būti neleistinos bet kuriuo atveju.

Jeigu duomenų bazė neturi transakcijų palaikymo realizuoti jas programiškai yra sudėtinga.

2.4.7. Apibendrinimas

Apibendrinsime karkaso savybių apžvalgą. Karkaso esminių elementų diagrama pavaizduota 7 pav.



7 pav. ORM karkaso pagrindinės funkcijos

2.5. ORM karkasų palyginimas

Panagrinėsime kelis populiariausius ORM karkasus ir pagrįsime kodėl šis prototipinis ORM karkasas turi būti kuriamas. Palyginsime kelis karkasus, kurie visi yra skirti .NET (prasmingiausia panagrinėti kaip toje pačioje platformoje skirtingi ORM karkasai realizuoja objektų saugojimo priemones).

2 lent. pateikiamas skirtingų ORM karkasų palyginimas su šiame darbe kuriamu karkasu.

2 lent. Skirtingų ORM karkasų galimybių palyginimas

	DataObjects.NET v3.5	Gentle.NET v1.2.5	Nhibernate v0.9	OJB.NET v0.8.2419	ORM.NET v1.7.2	Šio darbo Prototipinis karkasas
Objektiškai orientuotų klausų palaiky-	✓Taip	×Ne	✓Taip	✓Taip	×Ne	✓Taip

mas						
Internal/Attribute susiejimo palaikymas	✓Taip	×Ne	×Ne	×Ne	✓Taip	✓Taip
Databinding palaikymas	✓Taip	×Ne	✓Taip	×Ne	×Ne	✓Taip
Išsaugotų procedūrų palaikymas	✓Taip	×Ne	×Ne	×Ne	✓Taip	×Ne
Susiejimo konfigūravimo įrankis	×Ne	×Ne	×Ne	×Ne	✓Taip	✓Taip
Generic palaikymas	×Ne	×Ne	×Ne	×Ne	×Ne	✓Taip
Tingus krovimas	✓Taip	✓Taip	✓Taip	✓Taip	×Ne	✓Taip
Kodo generavimas	✓Taip	×Ne	×Ne	✓Taip	✓Taip	✓Taip
Kešavimas	×Ne	✓Taip	✓Taip	✓Taip	×Ne	✓Taip

Aptarsime palyginimo kriterijus ir pagrįsime kodėl kursime specializuotą karkasą.

Objektiškai orientuotų užklausų palaikymas – tai galimybė formuoti užklausas, naudojant objektinio lygmens sąvokas, skirtingai nei reliacinio.

Internal/Attribute susiejimo palaikymas reiškia, jog karkasas įgalina susieti laukus su atributais ne taip, kad kai kurie laukai neatsispindi į klasės atributus ir kai kurie atributai neatsispindi į lentelės laukus, tačiau karkasas išlaiko sugebėjimą operuoti neatspindimais elementais.

Databinding yra .NET karkaso *Windows Forms* ir *Web Forms* priemonė, leidžianti susieti vartotojo sąsajos komponentus su objektų atributais.

Išsaugotų procedūrų palaikymas – tai galimybė panaudoti SQL išsaugotas procedūras vietoje SQL generavimo.

Susiejimo konfigūravimo įrankis – tai priemonė, kuri sugeba generuoti susiejimo konfigūraciją iš duomenų bazės arba objektinio modelio su minimaliu vartotojo įsikišimu.

Generic collections palaikymas – tai .NET *generic* tipo kolekcijų palaikymas.

Kodo generavimas – tai galimybė generuoti kodą remiantis duomenų bazės schema su minimaliu vartotojo įsikišimu.

Kešavimas – tai objektų būsenų saugojimas atmintyje siekiant sumažinti kreipinių į duomenų bazę skaičių.

Kaip matome iš lentelės, kuriamas karkasas realizuos nemažai savybių lyginant su kitais karkasais.

2.6. Analizės apibendrinimas

Iš analizės rezultatų matome, kad tikrai egzistuoja semantinė spraga tarp reliacinių duomenų bazių struktūrų ir objektinių struktūrų ir kad šiai problemai spręsti pasitarnauja objektinio-reliacinio susiejimo karkasai. Kiti sprendimai, tokie kaip kad objektinių duomenų bazių naudojimas, nėra paplitę. Taip pat remiantis analize galima daryti išvadas kad universalaus ir optimaliausio sprendimo nėra, įvairūs inžineriniai sprendimai turi savitų privalumų ir trūkumų. Viena iš aktualiausių sistemų analitikui problemų yra tinkamas informacinės sistemos suprojektavimas, kuris subalansuoja abu modelius ir išlieka praktiškas. Karkasai ir įrankiai patys vieni neišsprendžia problemos, nes semantinės spragos esmė glūdi ne realizacijoje, o duomenų bazės ir klasių diagramos modeliuose. Verslo logikos klasių projektavimas labai smarkiai priklauso nuo pasirinktos ideologijos ir todėl reikalingos tam tikros gairės, padedančios suprojektuoti klases, maksimaliai išnaudojančias ORM karkasų teikiamas galimybes tačiau sugebančios egzistuoti nepriklausomai nuo jų. Sekančiose magistrinio darbo dalyse aptarsime siūlomą objektinio-reliacinio susiejimo modelį ir remiantis juo suprojektuosime prototipinį ORM karkasą. Pagrindinis mūsų uždavinys yra išsiaiškinti kaip efektyviau išspręsti šiame darbe nagrinėjamą problemą, ir, nors darbas nepretenduoja į optimaliausio rezultato siūlymą, išsiaiškinti esminius niuansus, kurie yra aktualūs projektuojant objektiškai orientuotas taikomąsias programas, naudojančias reliacines duomenų bazes, pasinaudojant šio magistrinio darbo rezultatų patirtimi.

3. OBJEKTINIO-RELIACINIO SUSIEJIMO MODELIS IR DEMONSTRACINIO KARKASO PROJEKTAS

3.1. Skirtumai tarp objektiškai orientuoto ir reliacinę schemą atspindinčio modelio

Kaip jau patyrinėjome 2.1 skyriuje yra nemažai skirtumų tarp reliacinio ir objektinio modelio.

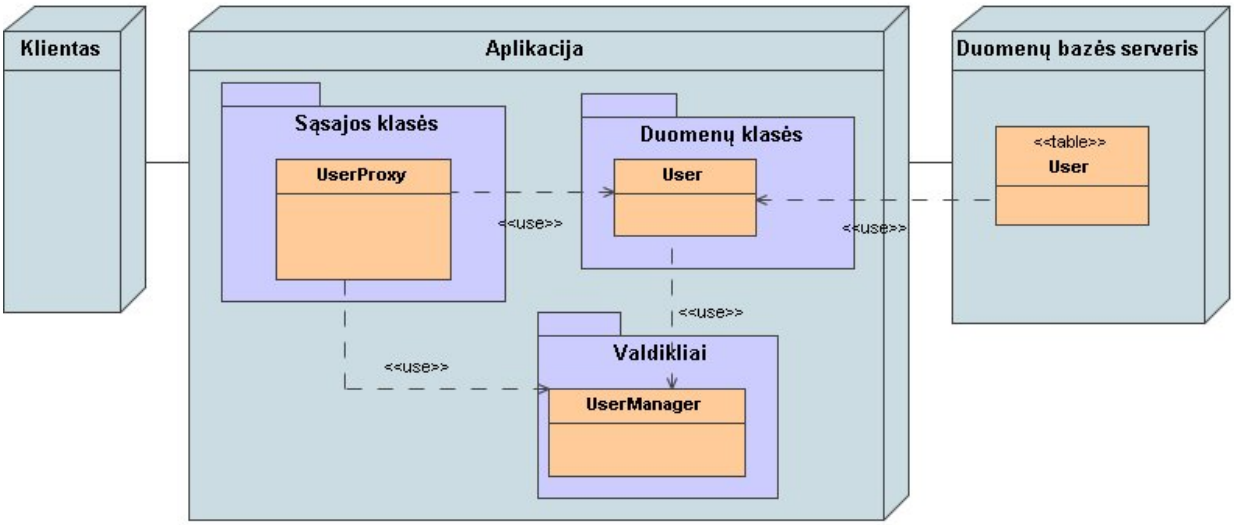
Jeigu klasių diagrama yra suprojektuojama taip, kad klasės atspindi duomenų bazės lentelių schemą tuomet tokią klasių diagramą vadinsime reliacinę schemą atspindinčia. Jeigu klasių diagrama yra labiau orientuota į dalykinės srities aprašymą o ne duomenų bazę tokią klasių diagramą vadinsime objektiškai orientuota.

Taip pat šiame skyriuje išanalizuojami ORM karkasų ypatumai ir parengiamas demonstracinio karkaso projektas. Tiriama kaip objektai išreiškiami aukšto lygio programavimo kalbomis ir kaip ši informacija kartu su pačiu objekto turiniu gali būti išsaugoma išorinėse duomenų saugyklose, aptariamos vykdymo aplinkos priemonės, kurios yra reikalingos realizuoti ORM karkasus, detalizuojamos ORM karkasams būdingos funkcijos ir savybės, iškeliami reikalavimai demonstraciniam prototipui, analizuojama demonstracinio ORM karkaso architektūra ir veikimo principai.

3.2. Kompromisinis sprendimas – MVC principų pritaikymas klasių modeliui

Model View Controler principai gali būti sėkmingai pritaikyti ORM susiejimo kompromisiniam sprendimui. Duomenų klasės (*Model*) gali būti suprogramuotos taip, kad atitiktų reliacinės duomenų bazės lenteles, tuo tarpu sąsajos klasės (*View*) atitiktų pageidaujamus dalykinės srities pjūvius. Valdiklių klasės (*Controler*) rūpintųsi sinchronizacija tarp duomenų ir sąsajos klasių ir kitais objektų gyvavimo ciklo valdymo veiksmais bei dalykinės srities procedūromis.

Klasių išskaidymo pagal MVC principinė diagrama pateikiama 1 pav.



8 pav. Klasių išskaidymas pagal MVC

Panagrinėsime visus išskaidymo modelius detaliau.

3.2.1. Duomenų klasių modelis

Duomenų klasės tiesiogiai siejasi su duomenų bazės lentelėmis. Net ryšio lentelės atsivaizduoja į atitinkamas klases. Dėl to semantinė spraga tarp duomenų bazės ir programinio kodo smarkiais sumažėja, deja duomenų klasės prasčiau reprezentuoja dalykinę sritį.

Duomenų klasių modeliui svarbi identifikatorių problema. Paprastai identifikatoriai, kurie neturi prasmės dalykinėje srityje, klasių interfeisuose turi būti nematomi. Taip pat ryšio raktai turi būti keičiami nuorodomis į susijusius objektus. Šios savybės supaprastina klasių diagramą, padaro klases tvarkingesnes OOP atžvilgiu. ORM karkasas turi pats pasirūpinti identifikavimo ir susiejimo problemomis.

Duomenų klasių modelis gali būti labai lengvai sugeneruotas automatiškai bei pergeneruotas iškilus poreikiui, tačiau rimtesni pakeitimai dalykinėje srityje gali sukelti drastiškus pakitimus duomenų klasių modelyje taigi šis modelis yra mažiau atsparus pakeitimams.

3.2.2. Sąsajos klasių modelis

Sąsajos klasės skirtos būti panaudojamos atvaizdavimo reikmėms. Jų sąsajos labiau atitinka dalykinės srities esybes. Sąsajos klasės apgaubia duomenų klases, iškeldamos reikiamus interfeiso elementus iš duomenų klasių interfeisų į savo ir praplėsdamos trūkstamais elementais. Sąsajos objektai negali egzistuoti be atitinkamo duomenų objektų su kuriais jie susieti nes jie patys nesaugo informacijos. Sąsajos klasės gali agreguoti kelių duomenų klasių duomenis arba atvirkščiai kelios sąsajos klasės pateikti vienos duomenų klasės sąsają dalimis.

Sąsajos klasės įgalina panaudoti duomenų klases bet kokiame kontekste ir pritaikyti išorinių karkasų panaudojimui. Pavyzdžiui, sąsajos klasės gali būti orientuotos į tam tikrų *ObjectDataSource*[11] aptarnavimą.

Kintant reikalavimams sąsajos klasės iš dalies gali sukelti problemų jeigu reikia nemažai jų atnaujinti iš dalies gali apsaugoti nuo pakeitimų – sąsajos klasių sąsajas išlaikyti yra lengviau nei duomenų klasių.

3.2.3. Valdiklių klasių modelis

Valdiklių klasės susieja duomenų ir sąsajos klases bei atlieka dalykinės srities operacijas. Valdikliai yra ganėtinai specializuoti ir todėl bendrų nurodymų kaip juos projektuoti nėra. Galima remtis pasiteisinusiais projektavimo šablonais kad sumažinti architektūrinių klaidų skaičių.

3.2.4. Apibendrinimas

Apibendrinus visus šio skyriaus teiginius galima padaryti tokias išvadas:

Šitoks objektinių struktūrų atvaizdavimo į reliacines modelis leidžia realizuoti duomenų klases atitinkančias reliacinės duomenų bazės lenteles ir atvaizdavimo klases, orientuotas į reikiamus dalykinės srities pjūvius. Minėti principai leidžia turėti objektus, glaudžiai orientuotus į duomenų bazę, dėl to sumažėja duomenų bazės schemos ir objektinės struktūros derinimo problemos, bei objektus, orientuotus į vartotoją, kurie tvirčiau surišti su vartotojo reikalavimais bei padidina sistemos atsparumą kitimams, nekeičiantiems sąsajos. Pagrindiniai modelio trūkumai yra padidėjęs sudėtingumas kuris gali sąlygoti didesnį klaidų ir pakeitimų valdymo problemų skaičių bei sumažėję sistemos našumo rodikliai.

3.3. Objektų išreiškimo aukšto lygio programavimo kalbose ypatumai

Pirmiausia, pasiaiškinkime termino „aukšto lygio programavimo kalba“ sampratą. Aukšto lygio programavimo kalba yra nuo mašinos nepriklausoma programavimo kalba, kuri leidžia programuotojui susikoncentruoti ties problemos sprendimu. Objektinės aukšto lygio programavimo kalbos leidžia programuotojui spręsti problemą išreiškiant ją objektais ir jų sąveikomis. Dauguma aukšto lygio objektinių programavimo kalbų leidžia realizuoti visus pagrindinius objektinio programavimo aspektus: paveldėjimą, polimorfizmą, inkapsuliaciją ir pan. Kadangi mūsų kuriamas karkasas yra programuotojo produktyvumo įrankis, programavimo kalbos pasirinkimas turi didelę reikšmę.

---

[11] ObjectDataSource – tai .NET aplinkos klasė, skirta dirbti su objektais analogiškai darbui su duomenų bazės įrašais.

Aukšto lygio objektinės programavimo kalbos nežymiai skiriasi objektų atvaizdavimu. Vienas iš tokių dažnai pasitaikančių skirtumų yra objektų atributų realizavimas. Atributas čia nelygus objekto laukui – atributas yra objekto sąsajos dalis. Kadangi atributas yra objekto sąsajos dalis, objekto vartotojas negali daryti jokių prielaidų apie tai kaip šis atributas viduje objekto funkcionuoja. Ne visos objektinės programavimo kalbos palaiko atributus. Tokiu atveju analogiškas funkcionalumas turi būti realizuotas funkcijomis, kurios dažniausiai pradedamos žodžiais *get* arba *set*, skirtais identifikuoti kokią operaciją, atributo nuskaitymą ar priskyrimą jos atlieka.

3.4. Metaduomenų apie objektus saugojimas

Bandant atlikti susiejimą tarp objektinio ir reliacinių modelių iškyla klausimas kaip ir kur aprašyti susiejimo taisykles. Reliacinėse bazėse tokią informaciją saugoti neparanku nes duomenų bazės gali būti naudojamos ir kitose aplikacijose todėl nėra prasmės užteršti jas karkasų specifika, be to reliacinėse bazėse tai padaryti standartizuojamu būdu sunkiau. Kitas būdas yra susiejimą pažymėti objektinėje pusėje, bet čia yra du galimi variantai priklausomai nuo kalbos palaikomų funkcijų. Jeigu kalba palaiko standartizuotą metaduomenų prikabinimą prie kodo elementų tuomet viskas paprasta, tačiau jeigu ne reikia tenkintis įvairiomis gudrybėmis (kaip kad per specialų interfeisą išgaunama informacinė struktūra). Susiejimą galima aprašyti ir atskirame, dažniausiai XML formato, faile. To privalumas yra tas kad susiejimo taisykles galima koreguoti atskirai nuo duomenų bazės ir programinio kodo.

3.5. Vykdymo aplinka ir objektų saugojimas

Vykdymo aplinka (*runtime environment*) turi palaikyti reflekciją. Reflekcija – tai vykdymo aplinkos savybė, įgalinanti programos vykdymo metu išgauti informaciją apie objektų tipus ir sąsajas bei atlikti operacijas remiantis tomis informacinėmis struktūromis.

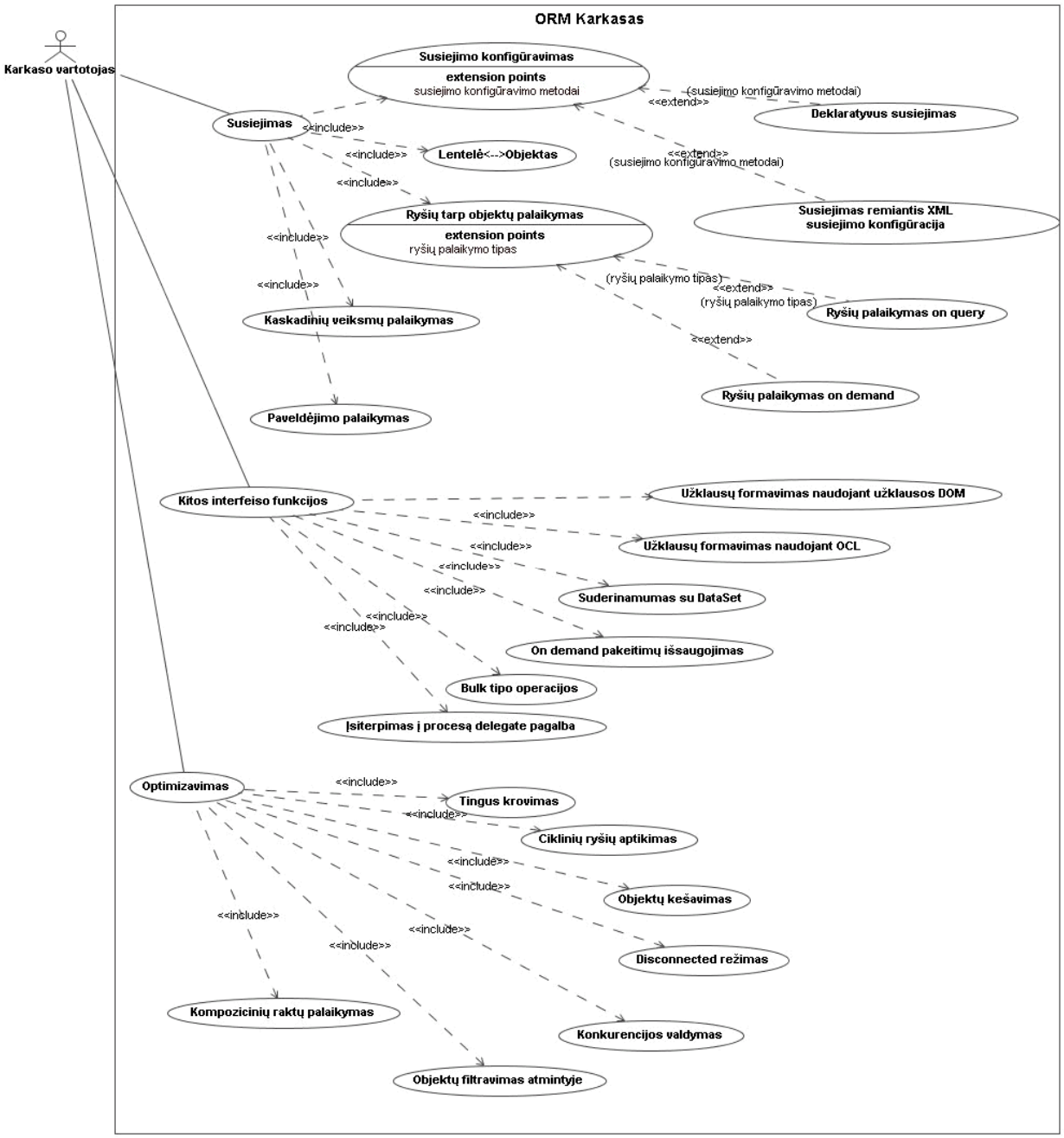
Ši savybė reikalinga tam, kad karkasas vykdymo metu galėtų pritaikyti išoriniuose failuose aprašytas susiejimo taisykles.

3.6. Reikalavimai demonstraciniam prototipui

Panagrinėsime kokie funkciniai ir nefunkciniai reikalavimai keliami projektuojamas karkasui.

3.6.1. Atliekamos funkcijos

Nors objektinio-reliacinio susiejimo karkasas turi išpildyti tik vieną esminį funkcinį reikalavimą – įgalinti objektams išsisaugoti duomenų bazėje – yra nemažai kitų funkcinių reikalavimų, kuriais mes užtikriname tam tikrą efektyvumo ir našumo lygmenį. Pateiksime juos 9 pav.



9 pav. ORM karkaso atliekamos funkcijos

Panagrinėsime ką kiekviena funkcija reiškia.

3.6.1.1. Susiejimo funkcijos

Susiejimo konfigūravimas – deklaratyvus

Deklaratyvaus konfigūravimo esmė – metaduomenų apie susiejimą prikabinimas ant klasės elementų pasinaudojant .NET *System.Attribute*[12] priemonėmis. Privalumas: galima kuriant klasę susieti ją su pageidaujamais duomenų bazės elementais čia pat vietoje, galima matyti į kokį duomenų bazės elementą atsivaizduoja klasės elementas. Trūkumas: klasės kodas apkraunamas metaduomenimis, kadangi susiejimas atliekamas kode norint jį pakeisti reikalingas klasės kodo perkompiliavimas. Šios funkcijos realizavimo prioritetas: Mažas

Susiejimo konfigūravimas – remiantis XML pavidalo ORM susiejimo failais

Deklaratyvaus susiejimo konfigūravimo trūkumų galima išvengti naudojant specializuotus XML pavidalo ORM susiejimo failus. Šiuose failuose galima nusakyti koks klasės elementas siejasi su kokiais duomenų bazės lentelės elementais, kokie yra ryšiai tarp klasių, kokie yra apribojimai, bei daug kitų susiejimo parametrų. Pakitus duomenų bazės schemai daugumoje atvejų nėra poreikio perkompiliuoti klasės kodą – užtenka paredaguoti ORM susiejimo failą.

Karkasas turėtų leisti susiejimus aprašyti ne viename ORM susiejimo faile bet keliuose, tuo užtikrinant galimybę skirtingų posistemių susiejimus aprašyti atskirai.

Hibernate susiejimą taip pat aprašo pagrinde išoriniais susiejimo failais ([14, 2.1 sk.]).

Šios funkcijos realizavimo prioritetas: Didelis

Lentelė <->Objektas susiejimas

Bet koks ORM susiejimas turi palaikyti šį esminį funkcionalumą – leisti objektui išsisaugoti duomenų bazėje. Žymiai parasčiau kai objektas gali išsisaugoti tik vienoje lentelėje ir atvirkščiai. Nėra poreikio jog kiekvienas objekto elementas atsivaizduotų į lentelę ir atvirkščiai. Lentelėje gali būti laukų, kurie svarbūs ORM karkaso funkcionavimui (raktų laukai, kurie neturi semantinės prasmės, *timestamp* tipo laukai ir pan.) bet nepriklauso klasės aprašui; tokius laukus ORM karkasas saugo savo vidinėse struktūrose ir susieja juos su atitinkamais objektais, naudodamas jų reikšmes esant poreikiui.

Šios funkcijos realizavimo prioritetas: Kritinis

[12] System.Attribute klasės egzemplioriai naudojami metaduomenims prie kodo elementų prisegti.

Ryšių tarp objektų palaikymas – on demand

Patogu kai karkasas pats žino kokie yra ryšiai tarp objektų ir sugeba pats išgauti ir pateikti susijusius objektus. Principo *on demand* esmė – susijusių objektų išgavimas ir pateikimas pareikalaujant juos per objekto kompozicines savybes. Kitaip tariant, jeigu turime objektą tipo Autorius ir norime gauti jo adresus (kurių kiekvieno tipas yra Adresas), kreipiamės į objekto savybę Adresai kurios tipas yra Adresas tipo objektų rinkinys. Toks principas yra labai lengvai suvokiamas, paprastai įsisavinamas ir apskritai patogus.

Šios funkcijos realizavimo prioritetas: Didelis

Ryšių tarp objektų palaikymas – on query

Kai kada patogu užklausos metu nurodyti kokie susiję su pagrindiniu mūsų užklausiamu objektu objektai mus domina ir turi būti iš karto užkrauti. Taip pat mes galime pareikalauti tuos susijusius objektus atfiltruoti pagal mus dominančius kriterijus. Deja, tai ženkliai padidina užklausimo sudėtingumą lyginant su vieno objekto tipo užklausomis.

Šios funkcijos realizavimo prioritetas: Mažas

Kaskadinių veiksmų palaikymas

Jeigu dirbame su tėviniu objektu, kai kurios operacijos liečia ir vaikinius objektus ir tam tikrais atvejais turi būti atliktos ir jiems. Jeigu mes norime atlikti objekto trynimą pirmiausiai turime ištrinti vaikinius objektus. Jeigu buvo atliktas objekto išsaugojimas ir pirminio rakto elementai pakito tai turi atsispindėti vaikinių objektų išoriniuose raktuose. Taigi, karkasas turi automatiškai atlikti reikiamus kaskadinius veiksmus.

Šios funkcijos realizavimo prioritetas: Kritinis

Paveldėjimo palaikymas

Tarkime turime objektą kurio tipas yra Vartotojas ir objektą, kurio tipas yra Autorius. Akivaizdu, jog Autorius yra Vartotojo subklasė ir Autoriaus egzempliorius turi ne tik Autoriui būdingus laukus bet ir Vartotojo laukus. Taigi, susiejant objektą, kurio tipas Autorius, su duomenų baze, mums reikalinga atlikti ir su Vartotojo objekto susiejimu susijusias operacijas. Saugant autorių duomenų bazėje yra pirmiausiai saugoma varotojo informacija į vartotojo lentelę, po to autoriaus informacija. Autoriaus lentelėje egzistuoja išorinis raktas į vartotojo lentelę. Paveldėjimas iš esmės yra superklasės objekto specializuotas agregavimas subklasės objekte, tik subklasės interfeisas paveldi superklasės interfeiso elementus.

Šios funkcijos realizavimo prioritetas: Mažas

3.6.1.2. Kitos interfeiso funkcijos

Be esminių ORM operacijų karkasas gali leisti per jo API atlikti ir kitas operacijas, kurios palengvina tam tikrų veiksmų atlikimą.

Užklausų formavimas naudojant DOM

Užklausas formuojant DOM principu jos yra sumontuojamos kuriant atitinkamus užklausos elementus ir priskiriant juos užklausos objektui (pvz. Sąlygos objektą, rikiavimo objektus). Toks principas įgalina panaudoti stiprų tipų tikrinimą ir sumažina klaidų tikimybę, tačiau užklausos formavimas gali gautis labiau gremėzdiškas API vartotojo atžvilgiu.

Šios funkcijos prioritetas: Didelis

Užklausų formavimas naudojant OQL

Naudojant OQL užklausa aprašoma eilutės pavidalu ir išanalizuojama OQL variklio. Privalumai: paprastesnis užklausų formavimas; galimybė užklausas patogiai saugoti; gali būti paprastesnis metodas išmanantiems OQL API vartotojams; Trūkumai: didesnė tikimybė padaryti klaidas, užklausos analizavimas tampa sudėtingesnis; visų OQL galimybių išpildymas gali būti nereikalingas.

Šios funkcijos prioritetas: Mažas.

Suderinamumas su DataSet

Jeigu karkasas realizuojamas .NET aplinkoje, tuomet gali būti patogu gauti užklausų rezultatus DataSet pavidalu, kad būtų galima pritaikyti į DataSet orientuotas priemones.

Šios funkcijos prioritetas: Mažas.

On demand pakeitimų išsaugojimas

Patogiau, kai visos atliktos operacijos su objektais ne iškart atliekamos duomenų bazėje jas atlikus, bet išsaugomos API vartotojui pareikalavus. Taip suteikiama galimybė išsaugoti tik norimus pakitimus ar įterpti kitokią logiką.

Šios funkcijos realizavimo prioritetas: Didelis

Bulk tipo operacijos

Bulk operacijos – tai tokios operacijos, kurios dirba su tiek daug objektų, jog atliekamos neužkraunant tų objektų. Paprastai tai būna tiesioginis priėjimas prie duomenų bazės SQL galimybių. Patogiau kai tokios operacijos realizuotos į objektus orientuotu API.

Šios funkcijos realizavimo prioritetas: Mažas.

Įsiterpimas į procesą delegate pagalba

Delegate(funkcijų rodyklės) įgalina API kviesti aprašytos sąsajos vartotojo kodo paprogrames kurios įsiterpia į ORM procesus ir suteikia priemones vartotojui pakoreguoti ORM funkcionavimą ar išgauti jo veikimo operaciją.

Šios funkcijos realizavimo prioritetas: Mažas

3.6.1.3. Optimizavimas

Kad karkasas funkcionuotų sparčiau ir našiau atliktų API vartotojo operacijas prasminga realizuoti tam tikras priemones, padidinančias jo našumą ar kitaip optimizuojančias karkaso veikimą.

Tingus krovimas

Tingaus krovimo principas – „kol ko nors nereikia, tol nekrauname“. Tingiam krovimui reikalingos specializuotos sąsajos savybės, kurios kreipiasi į karkaso priemones. API vartotojo atžvilgiu klasės išlieka tokios pačios, tačiau susijusių objektų užkrovimu rūpinasi karkasas, visiškai išlaisvindamas API vartotoją nuo rankinio jų suradimo, užkrovimo ir priskyrimo.

Šios funkcijos realizavimo prioritetas: Didelis.

Ciklinių ryšių aptikimas

Cikliniai ryšiai yra dažni objektiniame programavime. Tėvinis objektas turi vaikinių objektų sąrašą o vaikiniai objektai – nuorodą į tėvinį objektą. Jeigu karkasas neatpažintų kad operacija su tam tikru objektu jau atlikta jis patektų į begalinį rekursinį ciklą. Paprastas sprendimas norint išvengti tokios situacijos – operacijos keičia objektų būsenas, jeigu objekto būsena yra einamoji – jokie veiksmai su objektu neatliekami.

Šios funkcijos realizavimo prioritetas: Kritinis.

Objektų kešavimas

ORM vienas iš esminių aspektų, garantuojantis didesnį našumą dirbant su pavieniais objektais lyginant su reliaciniais metodais – galimybė išsaugoti objektus atmintyje ir nepakitus užklausai grąžinti objekto kopiją iš kešo. Esminė problema su šiuo principu – nustatymas kad objekto kopija yra keše ir reikalauja atnaujino. Šios problemos sprendimas priklauso nuo naudojamos DBVS.

Šios funkcijos realizavimo prioritetas: Mažas.

Disconnected režimas

Laikyti prisijungimą prie duomenų bazės atidarytą kai neatliekamos jokios operacijos su ja yra netikslinga. Jeigu objektai nekinta tarp kreipimųsi į duomenų bazę tikslinga juos saugoti atmintyje. Šiuo aspektu ORM karkasas primena ADO.NET karkasą.

Šios funkcijos realizavimo prioritetas: Didelis.

Konkurencijos valdymas

Kai duomenų baze dirba ne vienas vartotojas vienu metu, gali iškilti tokia situacija, jog turimas objektas tampa nebegaliojantis nes kitas vartotojas jį pakeitė ir išsaugojo pakeitimus į duomenų bazę. Yra du metodai konkurencijos sprendimui: objektų rakinimas ir objektų versijavimas. Rakinimo esmė tokia: jeigu objektas paimamas redagavimui, tai pažymima duomenų bazėje ir tolesnės operacijos su juo draudžiamos iki rakinimas nuimamas. Tam tikrais atvejais tai gali sukelti įvairių problemų (pvz. Nulūžus programai rakinimas niekada nenuimamas). Versijavimo esmė: saugojimo metu pažiūrima ar objekto versija (timestamp) nebuvo pakeista. Jeigu taip, vadinasi objektą pakeitė kitas vartotojas, reikia generuoti klaidos pranešimą ir liepti vartotojui pakartoti veiksmus su einamąja objekto versija.

Šios funkcijos realizavimo prioritetas: Mažas.

Objektų filtravimas atmintyje

Kai turimas vaikinių objektų sąrašas atmintyje patogu juos atfiltruoti pagal tam tikrus objektinius kriterijus. Tam reikalingos priemonės, kurios panašios į objektų užklausimo iš duomenų bazės priemones.

Šios funkcijos realizavimo prioritetas: Mažas.

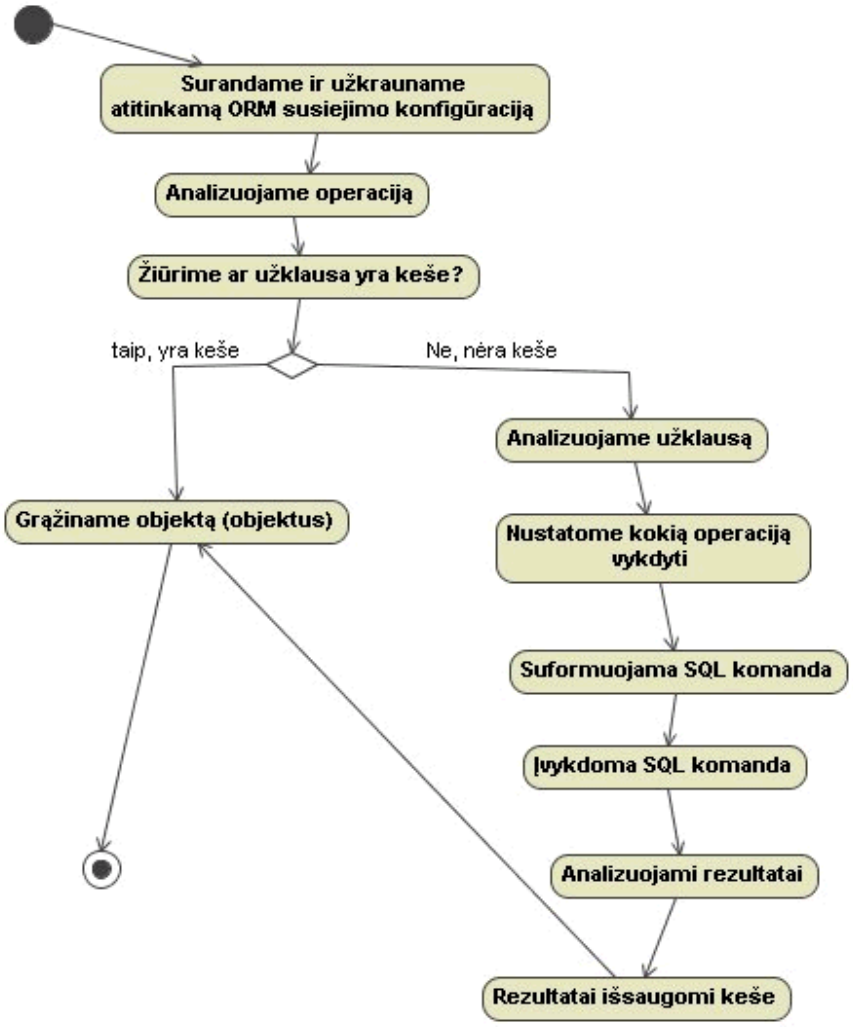
Kompozicinių raktų palaikymas

Lentelių raktai gali būti kompoziciniai – susidėti ne iš vieno įrašo. Taip pat ne visi raktiniai laukai gali turėti atitikmenį klasės aprašyme. Jeigu raktas dirbtinis jis gali būti IDENTITY arba GUID tipo. IDENTITY tipo raktas yra skaičius, kuris automatiškai didinamas. GUID tipo raktas yra pseudoatsitiktinis binaras. Šie raktai reikalauja atitinkamo apdorojimo darbo su duomenų baze metu.

Šios funkcijos realizavimo prioritetas: Didelis.

3.6.2. Funkcionavimo diagrama

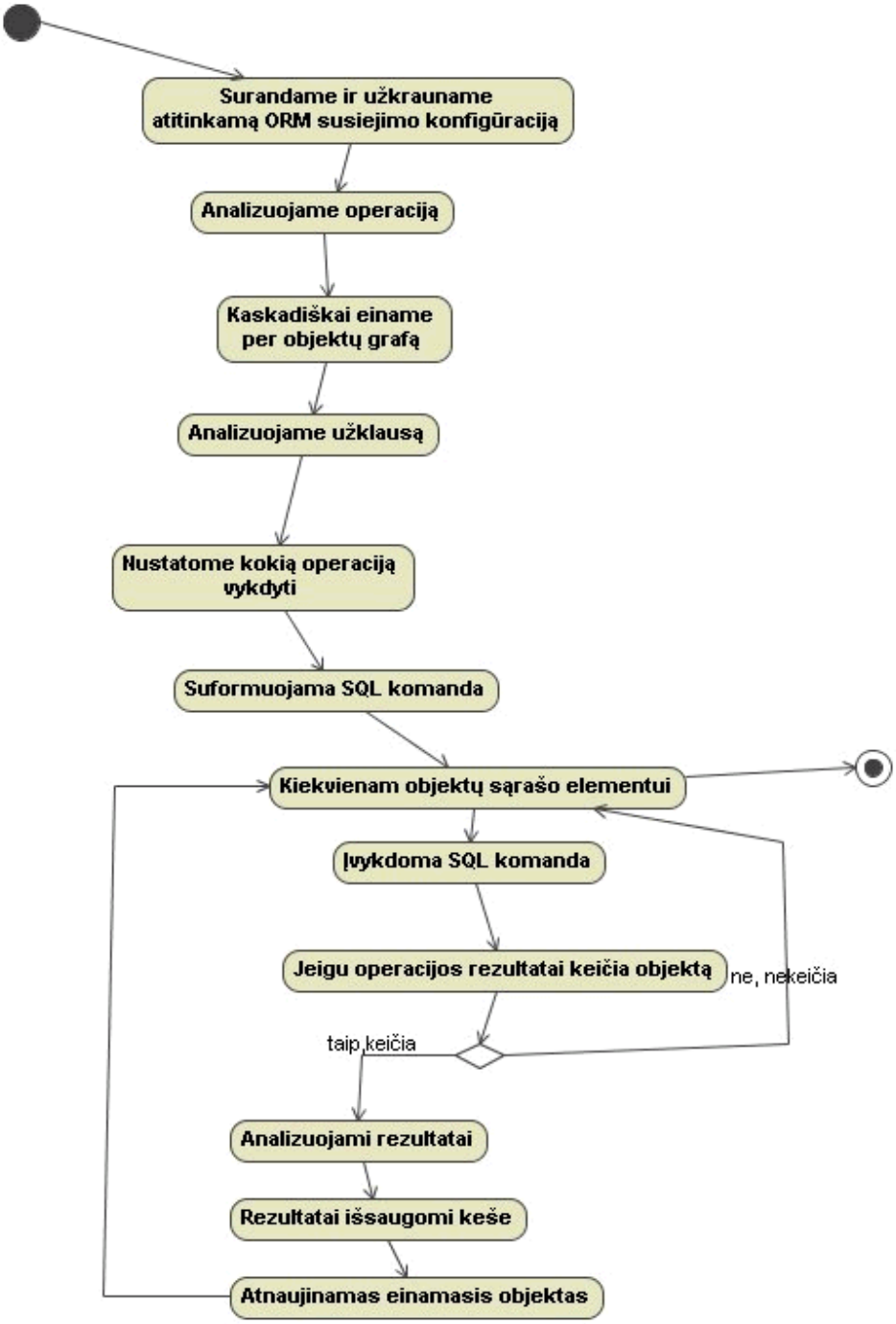
Sekančiame paveiksle (10 pav.) pateikiama karkaso veiksmų seka atliekant objekto išgavimo iš duomenų bazės operaciją.



10 pav. karkaso veiksmų seka atliekant objekto išgavimo iš duomenų bazės operaciją

Pateiktoje diagramoje pavaizduota karkaso veiksmų seka atliekant objekto (ar objektų rinkinio) išgavimo iš duomenų bazės veiksmų schema. Kaip matyti iš diagramos, pirmiausiai yra žiūrima ar rezultatai yra keše. Jeigu ten rezultatai nerandami, tuomet yra atliekamas užklausos vykdymas. Išgavimo operacijai kaskadiniai veiksmai nereikalingi, dirbama tik su vienos klasės objektais ir yra grąžinami visi objektai, atitinkantys nurodytus kriterijus.

Sekančiame paveiksle (10 pav.) pateikiama karkaso veiksmų seka atliekant objekto išsaugojimo operaciją.

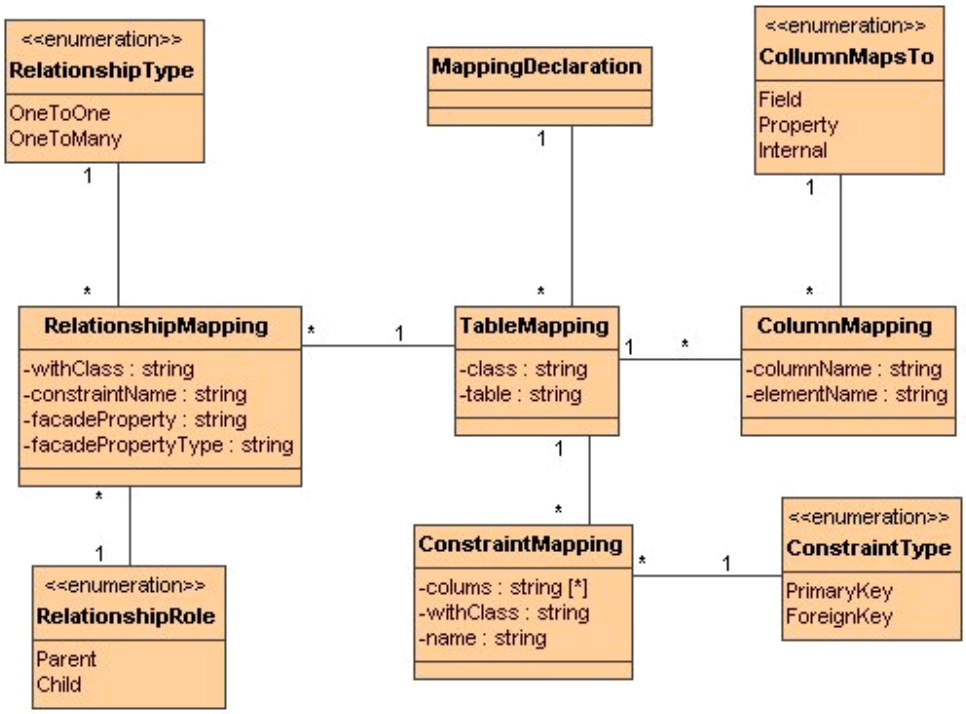


11 pav. Karkaso veiksmų seka atliekant objekto išsaugojimo operaciją

Pavaizduotoje diagramoje matome, kad atliekant objektų išsaugojimą operacijos objektams atliekamos kaskadiškai. Tam tikros operacijos gali modifikuoti objektus, todėl po jų reikia nuskaityti pakeistus objektus ir atšviežinti esančias keše bei darbinę jų kopijas (pvz. naujo objekto saugojimo metu DBVS parenka dalies elementų reikšmes).

3.6.3. Koncepcinis klasių modelis

Koncepcinis klasių modelis pateikiamas 12 pav.



12 pav. Koncepcinis susiejimo klasių modelis

Iš matome jog pagrindinė susiejimo aprašo klasių struktūra nėra sudėtinga. Aptarsime klasių prasmes:

MappingDeclaration

Ši klasė aprašo konstantas, bendras visam ORM susiejimo failui bei saugo vieną ar daugiau *TableMapping*.

TableMapping

Aprašo Klasės<->Lentelės susiejimą: stulpelių susiejimą, apribojimus ir ryšius.

ColumnMapping

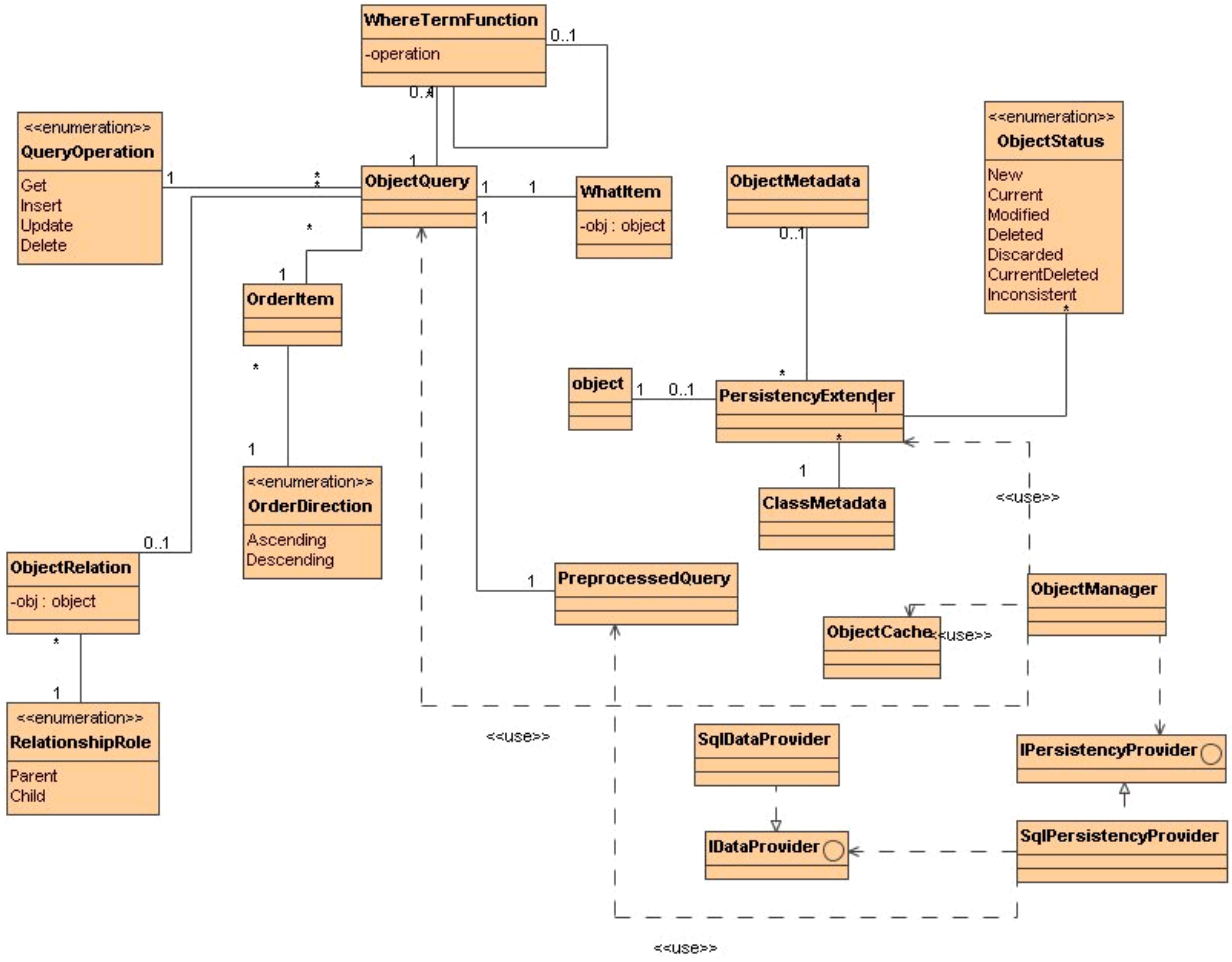
Aprašo lentelės lauko susiejimą su klasės elementu. Laukas gali sietis su klasės lauku (*field*), klasės savybe (*property*) ar nesisieti su jokiu klasės elementu, bet būti išsaugomas ORM karkaso struktūrose (*internal*).

ConstraintMapping

Aprašo apribojimą, taikomą lentelei. Gali būti pirminio ar išorinio rakto apribojimas (kiti nepalaikomi). Apribojimas aprašomas išvardijant į jį įeinančius lentelės laukus.

RelationshipMapping

Aprašo ryšį tarp objektų, objekto rolę ryšyje (tėvas ar vaikas), ryšio pobūdį (pvz. vienas su daug), kaip ryšys atsispindi šiame objekte – savybės pavadinimas (*FacadeProperty*) bei klasės tipas (*FacadePropertyType*)



13 pav. Koncepcinis karkaso klasių modelis

Pasiaiškinsime šių klasių paskirtis:

ObjectQuery

Skirta objektų užklausai aprašyti. Užklausa nusakoma per šios klasės savybes.

WhereTermFunction

Skirta apribojimams, taikomiems objektams, aprašyti. Palaiko *WhereTermFunction* parametrus, įgalinant aprašyti apribojimus medžio struktūros pavidalu.

QueryOperation

Nusako kokią operaciją turi atlikti užklausa.

WhatItem

Nusako kokio tipo objektai ar kokie objektai turi būti apdorojami užklausoje.

OrderItem

Skirta objektų išrikiavimui aprašyti.

ObjectRelation

Skirta apribojimams, kurie remiasi ryšiais su kitais objektais, aprašyti.

PreprocessedQuery

Išanalizuota užklausa, kuri saugo DBVS specifinę informaciją skirtą jai įvykdyti.

PersistencyExtender

Aprašo objekto ORM savybes ir reikiamus metaduomenis. Kiekvienam objektui, kuriam reikalingas ORM funkcionalumas, sukuriamas šios klasės egzempliorius.

ClassMetadata

Saugo metaduomenis, būdingus visai objekto klasei (paprastai susiejimo metaduomenis).

ObjectMetadata

Saugo metaduomenis, būdingus tam tikram objektui (paprastai *internal* savybių reikšmes).

ObjectStatus

Kokia yra objekto būsena ORM atžvilgiu.

ObjectManager

Suteikia priemones manipuliuoti objektais: juos išgauti ir išsaugoti.

ObjectCache

Saugo objektų kopijas atmintyje.

IDataProvicer ir SqlDataProvider

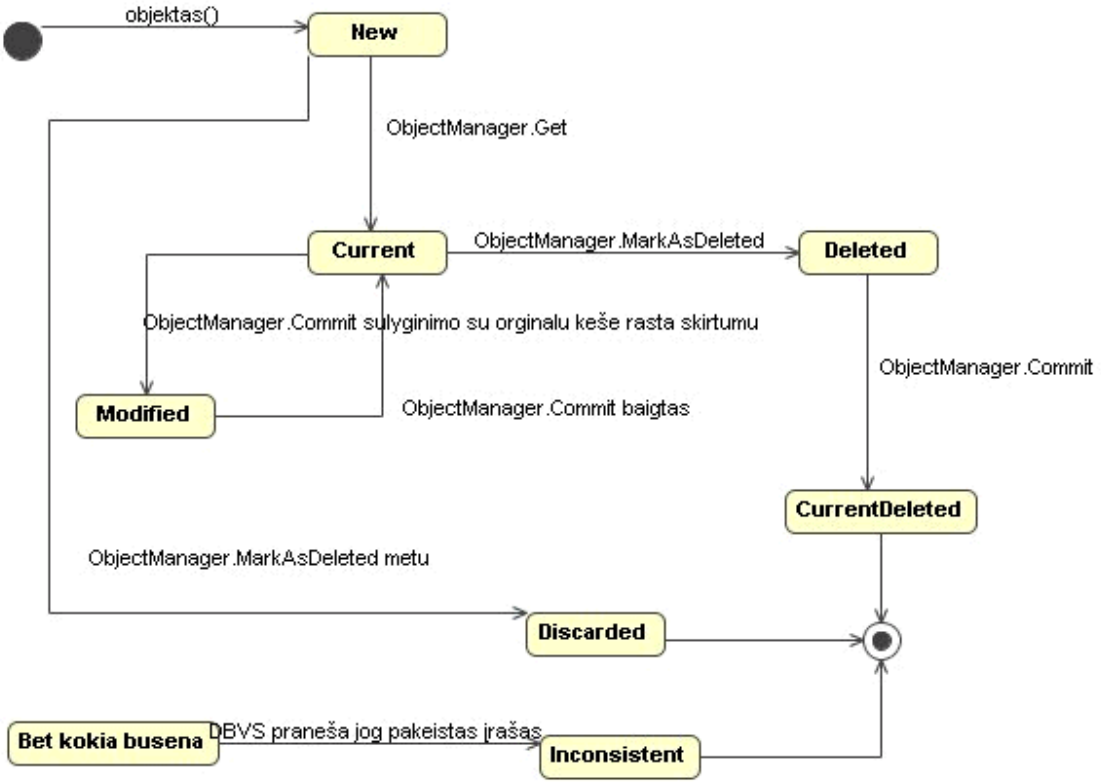
Suteikia priemones prieiti prie tam tikros DBVS.

IPersistencyProvider ir SqlPersistencyProvider

Suteikia priemones ORM realizuoti tam tikrai DBVS.

3.6.4. Sistemos dinamikos aprašymas

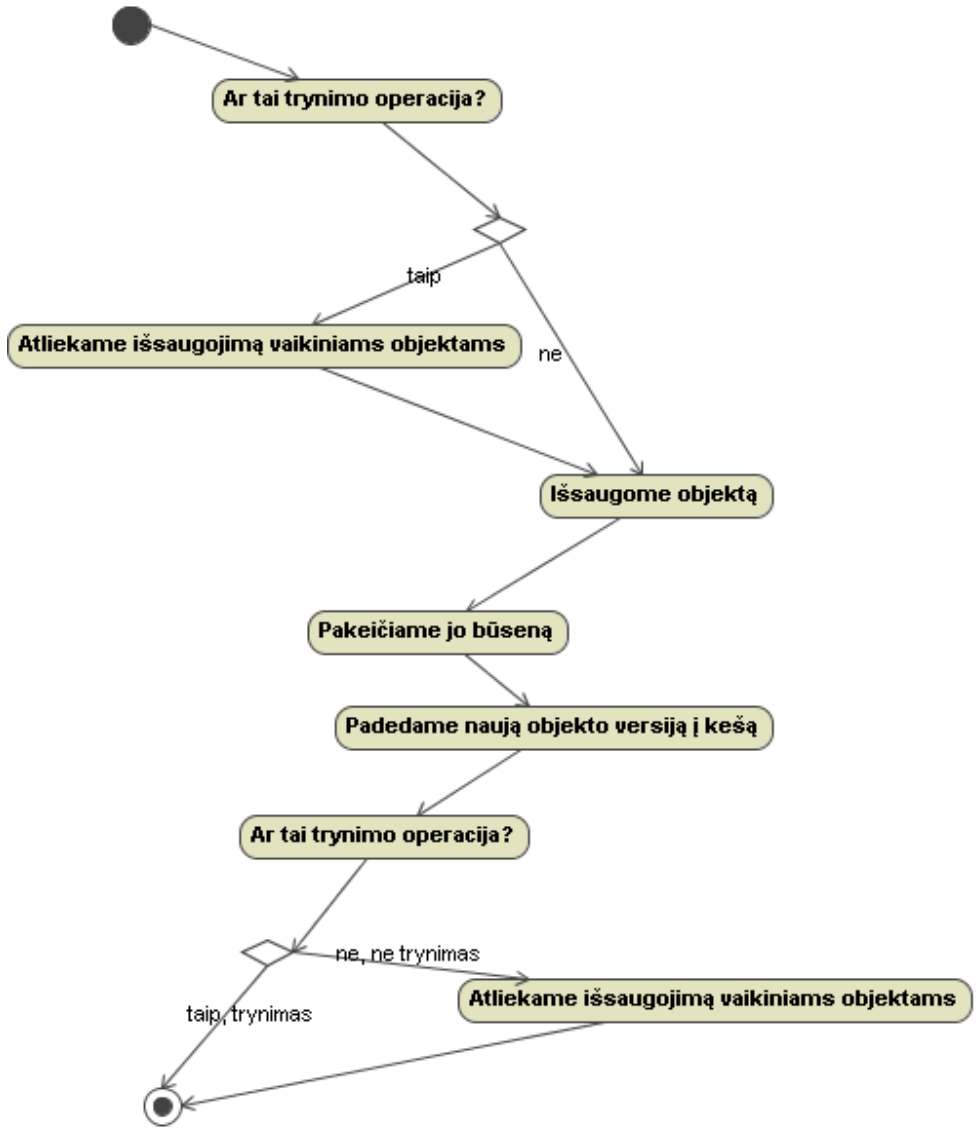
Panagrinėkime objekto būsenų diagramą 14 pav.



14 pav. Objekto būsenų diagrama

Kaip matyti iš 14 pav. visi objektai pradžioje būna *New* (t.y. naujas) būsenos. Kai objektas gaunamas iš duomenų bazės, jo būsena patampa *Current* (t.y. einamoji). Jeigu objektas buvo pamodifikuotas, saugant jį pastebėjus jog jis skiriasi nuo esančios keše versijos jo būsena pakeičiama į *Modified* (pakeistas). Išsaugojus tokios būsenos objektą į duomenų bazę jo būsena vėl patampa *Current.* Jeigu paprašoma objektą ištrinti ir jis buvo *Current* būsenos jis pažymimas kaip *Deleted* (t.y. ištrintas). Po to saugojimo metu jo būsena pakeičiama į *CurrentDeleted* (t.y. ištrintas ir išsaugotas). Kai objekto paprašoma ištrinti esant jam *New* būsenos jis pakeičiamas į *Discarded* (atmestas). Esant bet kurioje būsenoje DBVS pranešus jog įvyko pakitimų lentelėje ir pakitus įrašui turima objekto versija tampa negaliojanti ir jis pakeičiamas į *Inconsistent* (t.y. išderintas).

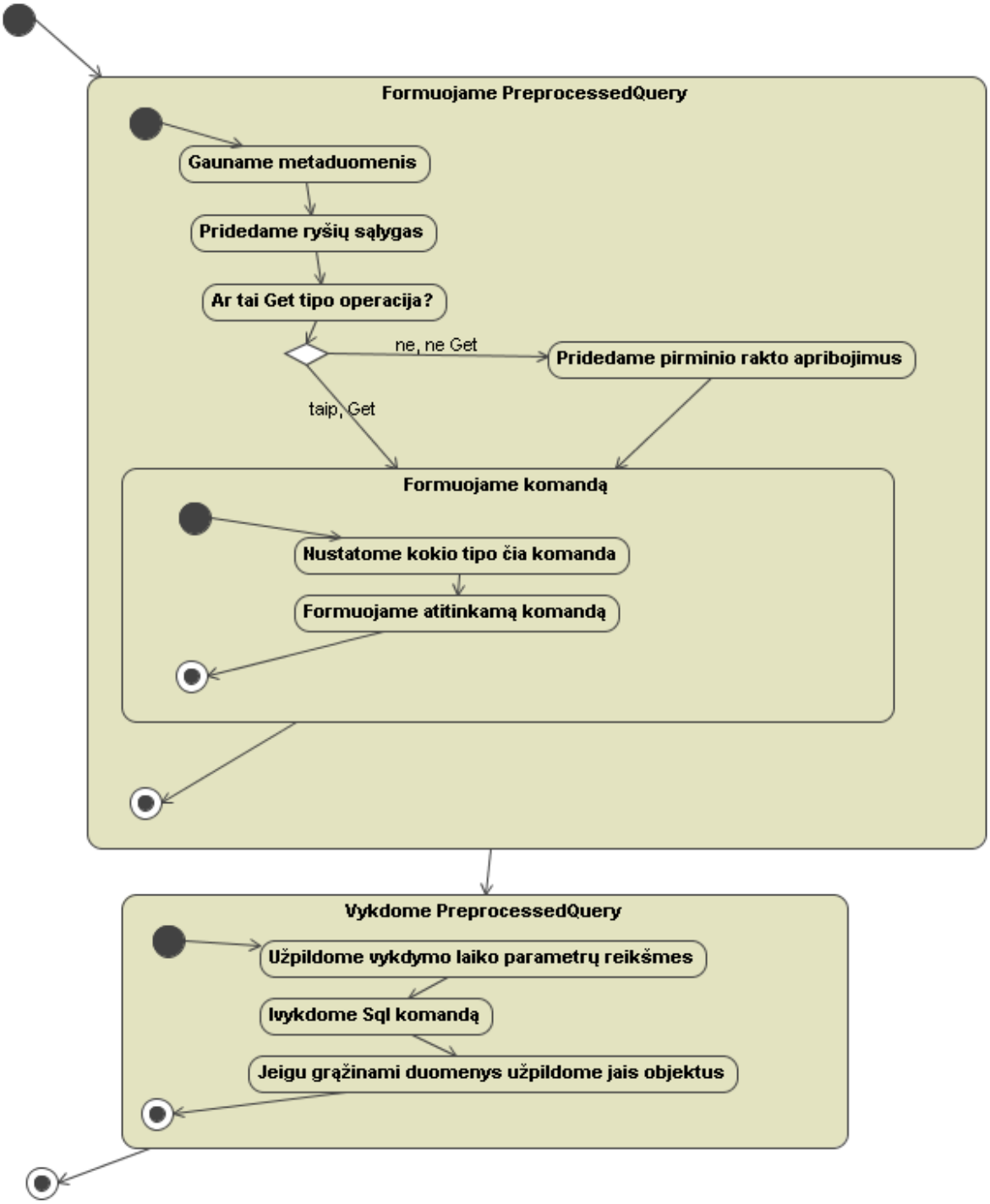
Sekančiame paveiksle (15 pav.) pateikiamas objekto pakeitimų saugojimo algoritmas. Kaip matyti iš jo, vaikinių objektų kaskadinis saugojimas yra atliekamas automatiškai. Jeigu objektas yra trinamas tuomet vaikiniai objektai turi būti ištrinti pirma, kad nebūtų išorinių raktų rodančių į nebeegzistuojantį tėvą.



15 pav. Objekto pakeitimų saugojimo algoritmas

Kitas svarbus algoritmas yra užklausos analizė ir vykdymas, pateiktas 16 pav.

Pirmas algoritmo žingsnis yra reikiamų metaduomenų išgavimas. Šis žingsnis atliekamas tik vieną kartą. Toliau yra analizuojama užklausa. Jeigu vykdoma ne išgavimo operacija darbinių objektai išanalizuojami išgaunant pirminių raktų reikšmes. Toliau yra suformuojama komanda ir įvykdoma. Jeigu tai buvo išgavimo operacija DBVS grąžinami duomenys padedami į grąžinamus objektus.



16 pav. Užklausos analizės ir vykdymo algoritmas

3.6.5. Nefunkciniai reikalavimai

Kai kurie karkaso funkciniai reikalavimai sistemoje, kuri jį naudoja, transformuojasi į tos sistemos nefunkcinius reikalavimus.

Aptarsime standartinius nefunkcinius reikalavimus ir jų įtaką

Našumas

Tai labai svarbus reikalavimas, kadangi vienas iš ORM panaudojimo tikslų ir yra išgauti didesnį našumą operuojant su objektais duomenų bazėje. Šis nefunkcinis reikalavimas pasiekiamas tam tikra .NET programavimo filosofija, tingiu krovimu/egzemplioriaus sukūrimu, kešavimo panaudojimu.

Saugumas

Karkasas nesirūpina saugumo aspektais. Operacijos su duomenų baze atliekamos duoto duomenų bazės vartotojo teisėmis.

Stabilumas

Tai taip pat labai svarbus nefunkcinis reikalavimas. Jis gali būti pasiekiamas tik kruopščiu karkaso derinimu, atsistatymo iš klaidingų situacijų priemonėmis, protokolavimo palaikymu.

Praplečiamumas

Kai kada svarbu jog karkasą būtų galima pritaikyti praplečiant ar papildant funkcionalumą. Tam yra panaudojami delegate skirti įsiterpti į procesus bei objektiškai orientuota architektūra, įgalinanti dėl izoliuotumo modifikuoti kodo dalis nekeičiant interfeisų. Taip pat DBVS specifinis funkcionalumas yra iškeltas į atskirus komponentus.

Patogumas vartoti

Karkaso API yra labai paprastas, jo vartotojui nebūtina žinoti daug funkcijų kad atlikti esmines operacijas. ORM susiejimo failams sukurti yra panaudojamas generatorius. Kadangi karkasas skirtas programuotojui, būtina, jog šis programuotojas turėtų neblogą .NET ir ADO.NET išmanymą.

3.7. Demonstracinio prototipo architektūra ir veikimo principai

3.7.1. Sistemos koncepcijos aprašymas

ORM karkasas suprojektuotas taip, kad galėtų egzistuoti nepriklausomai nuo kitų sistemos modulių, minimaliai įtakodamas jų kodą ir neįtakodamas jų sąsajų.

Klasės, kurios naudoja ORM funkcionalumą gali būti bet kokios, tačiau norint pasinaudoti susijusių objektų tingiuoju užkrovimu, ryšio savybėse turi būti kviečiamos atitinkamos *ObjectManager* funkcijos.

Su objektais tol kol nereikalingas jų išsaugojimas dirbama įprastiniu būdu. Prireikus jų išsaugojimo kviečiamas atitinkamas *ObjectManager* metodas. Jeigu norima gauti jau egzistuojančius objektus iš duomenų bazės taip pat naudojamos atitinkamos *ObjectManager* funkcijos. Karkaso vartotojas yra visiškai išvaduojamas nuo objektų susiejimo su atitinkamomis duomenų bazės lentelėmis bei ryšio objekto gavimo rankiniu būdu – karkasas pirmą kartą užklausus ryšio objekto per objekto savybę suformuoja atitinkamą užklausą ir grąžina jos rezultatus.

Pats karkasas turi tenkinti tokius koncepcinius reikalavimus:

- Turi būti suskaidytas į nepriklausomus komponentus

- Turi būti atskirtas bendras ir specifinis DBVS funkcionalumas
- Paprastam karkaso panaudojimui turi užtekti *ObjectManager* ir *ObjectQuery* klasių paslaugų
- Kuo mažiau karkaso specifikos turi matytis jo sąsajoje ir jo panaudojime

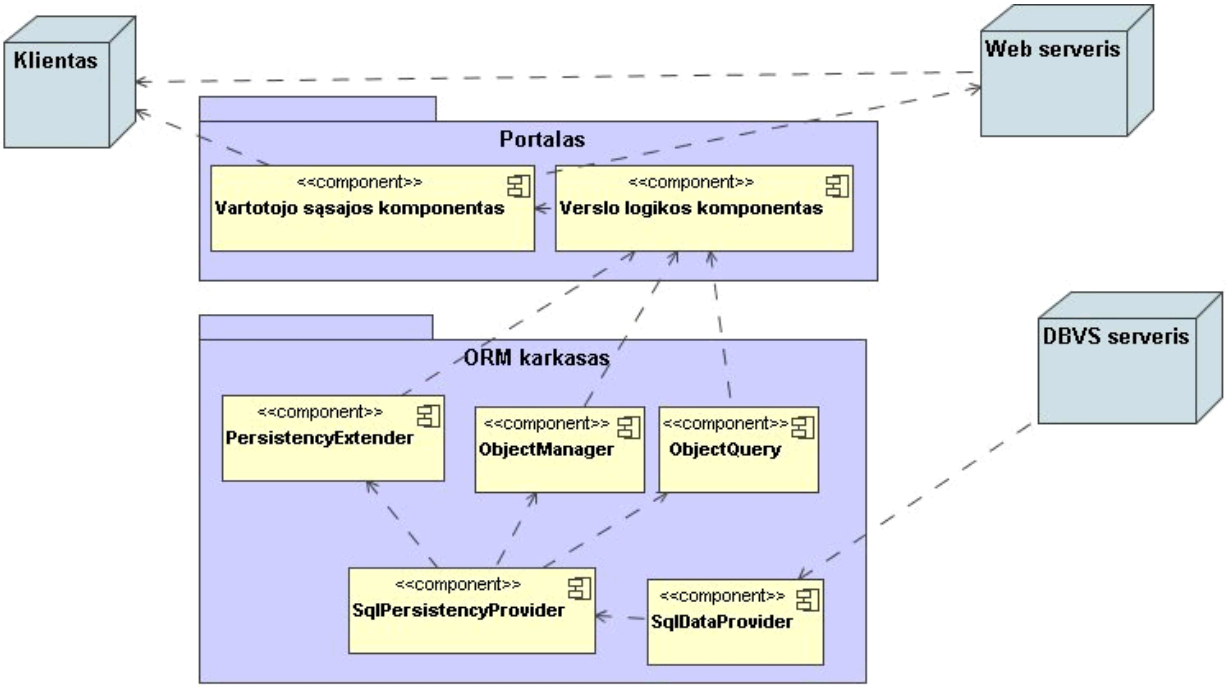
Taip pat keliami tokie bendri reikalavimai:

- Jeigu klasei reikalingas tik vienas egzempliorius naudojimo metu ji turi tenkinti *Singleton* projektavimo šabloną; tokios klasės yra *ObjectManager*, *ObjectCache* ir pan. Šios klasės suteikia statinę savybę *Instance* kuri leidžia prieiti prie vienintelio šios klasės egzemplioriaus. Naudojama specifinė *Singleton* realizacija, užtikrinantį tingų egzemplioriaus kūrimą
- Jeigu klasė yra duomenų, ji neturi jokių logikos metodų;
- Jeigu klasė yra valdiklis ji neturi jokių savybių
- Jeigu klasė atlieka du skirtingus funkcionalumus ji turi būti skeliama per pus
- Valdikliai turi būti saugūs gijos atžvilgiu

Kadangi šis karkasas yra ORM susiejimo demonstracija, nėra keliami specifiniai koncepciniai reikalavimai, jis gali būti bet koks išpildantis praeitame žingsnyje nurodytus reikalavimus; teisingų ORM karkaso sprendimų yra nemažai, jeigu tai neįtakoja programavimo dalykinėje srityje tai neturi didelės esmės.

3.7.2. Architektūros modelis

Architektūros modelis pateikiamas 17 pav.



17 pav. Sistemos architektūra

Jeigu laikysime jog sistema yra trijų sluoksnių (prezentacijos, logikos ir duomenų lygmenys) tai ORM karkasas suteiks priemones duomenų lygmens klasėms susirišti su DBVS serveriu.

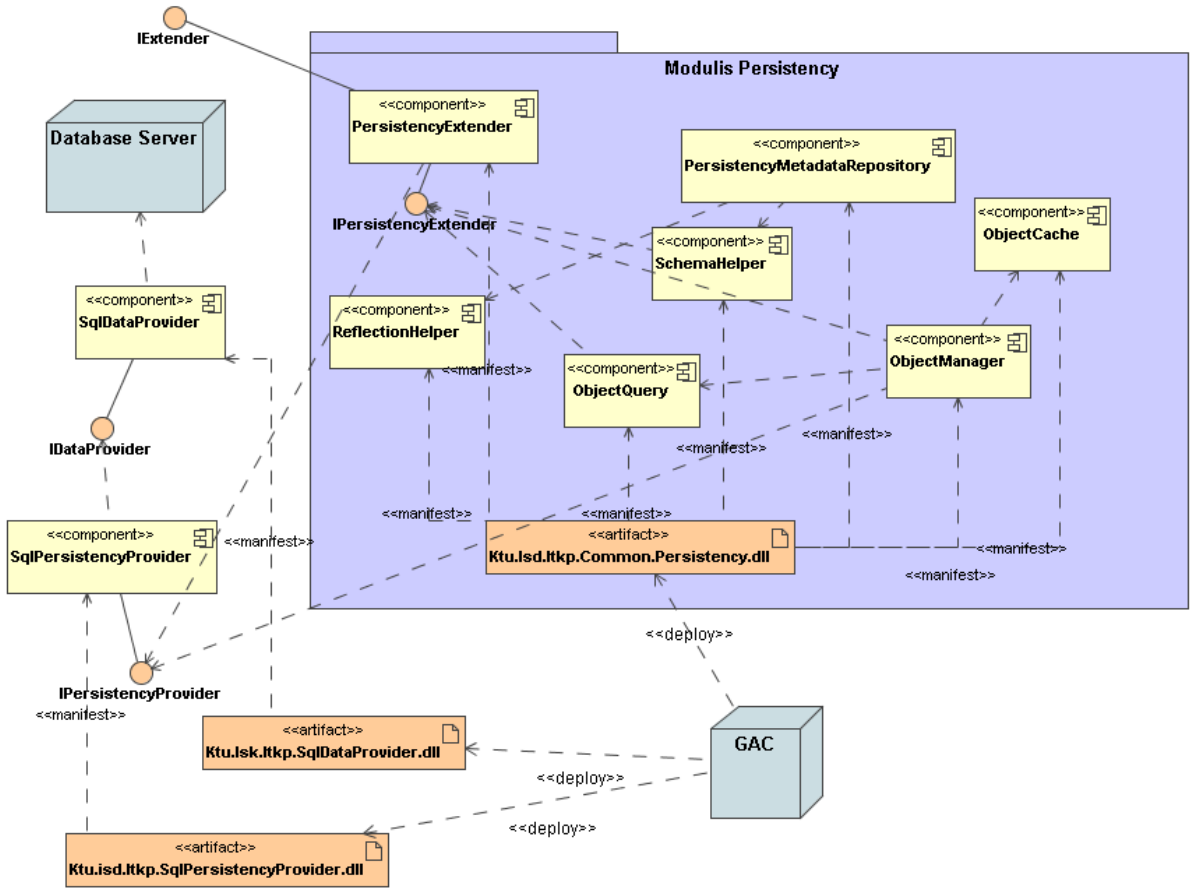
Apskritai, jokių specifinių reikalavimų sistemos architektūrai karkasas nekelia, tačiau yra keli rekomenduotini aspektai:

- Duomenų ir logikos atskyrimas; duomenų klasės neturi turėti metodų kurie atlieka veiksmus su objektu, priedo tokias klases yra lengviau sugeneruoti generatorių pagalba iš duomenų bazės schemos (*reverse engineering*).
- *PersistencyExtender* neturi būti tiesiogiai išgaunamas. Verslo logika neturi kviesti ORM karkaso funkcijų kad prieiti prie paslėptų ORM savybių. Tokiu atveju prasminga realizuoti savybes duomenų klasės aprašuose kurios kreipiasi į karkaso funkcijas, kadangi vienaip ar kitaip duomenų klasės naudoja karkaso funkcijas apraše tingiam krovimui užtikrinti
- Duomenų ir atvaizdavimo atskyrimas yra sudėtingas aspektas jeigu norima pasinaudoti .NET karkaso *DataBinding* funkcionalumu leidžiančiu vartotojo sąsajos elementų savybes susieti su duomenų objektų savybėmis. Rekomenduojama naudoti *ObjectDataSource* bei realizuoti valdiklius, kurie suteikia funkcijas reikalingas *ObjectDataSpace*. Vartotojo

sąsajos elementų susiejimas su duomenų objektų interfeisais yra leistinas, tačiau sprendimas, kuriame šis susiejimas būtų atliekamas konfigūracijos failų pagalba yra rekomenduotinas.

3.7.3. Realizacijos modelis

Realizacijos modelis pateikiamas 18 pav.



18 pav. Realizacijos modelis

Database Server

Saugo reliacinius duomenis. Karkaso kai kurie komponentai suprojektuoti Microsoft Sql Server 2005 DBVS.

GAC

Global Assembly Cache – .NET karkaso nustatyta vieta kur yra padedamos visos asemblės kad būtų pasiekiamos visoms kompiuterio programoms.

PersistencyExtender

Klasė, kuri skirta objektui praplėsti ORM funkcionalumu. Saugo objekto ORM metaduomenis bei suteikia aibę funkcijų kitoms karkaso priemonėms.

PersistencyMetadataRepository

Pasirūpina ORM konfigūracijos failų suradimu ir nuskaitymu, ORM susiejimo aprašymo užkrovimu ir analize

ReflectionHelper

Pagalbinės priemonės, kurios, naudojant *System.Reflection* vardų aibės funkcijas, įgalina atlikti operacijas su objektais nepriklausomai nuo jų tipo – nuskaityti ir priskirti laukų reikšmes, kviesti metodus. Tik *System.Reflection* dėka ORM susiejimas yra įmanomas .NET karkase.

SchemaHelper

Analizuoja ORM susiejimo informaciją ir inicializuoja visus reikiamus metaduomenis

ObjectQuery

Aprašo užklausas, kurios atlieka operacijas su objektais. Įgalina aprašyti kokie objektai turi būti apdorojami, kokius kriterijus jei turi atitikti, su kokiais objektais kokiais ryšiais turi sietis, kokia tvarka išrikiuoti rezultatus.

ObjectCache

Saugo objektų kopijas atmintyje tuo drastiškai padidindama ORM karkaso spartą.

ObjectManager

Suteikia priemones objektams gauti iš duomenų bazės ir išsaugoti.

SqlDataProvider

Realizuoja *ISqlDataProvider* Microsoft SQL 2005 Server DBVS. Suteikia priemones prisijungti prie duomenų bazės ir vykdyti SQL užklausas.

SqlPersistencyProvider

Realizuoja *IPersistencyProvider.* Analizuoja užklausas ir jas vykdo Microsoft SQL 2005 Server DBVS.

3.7.4. Reikalavimai sistemos funkcionavimo palaikymui

Sistemos funkcionavimui reikalinga:

- .NET framework 2.0
- Microsoft SQL Server 2005 su sukonfigūruotu notifikavimu
- Aplikacijos konfigūracijoje nuorodos į ORM susiejimo failus bei kiti ORM karkaso nustatymai
- Korektiški ORM susiejimo failai visoms klasėms, kurioms reikalingas šis funkcionalumas
- Visos reikalingos asemblės GAC:
    - Ktu.Isd.Itkp.Common.dll
    - Ktu.Isd.Itkp.Common.Persistency.dll
    - Ktu.Isd.Itkp.SqlDataProvider.dll
    - Ktu.Isd.Itkp.SqlPersistencyProvider.dll
    - Ktu.Isd.Itkp.Common.Exceptions.dll
- Programuotojas turi turėti darbo su .NET karkasu, C#, ADO.Net patirties.

3.7.5. Sistemos naudojimo instrukcija

Paanalizuosime kas yra reikalinga be paties karkaso parengti testavimo prototipui.

Tam, kad būtų galima pademonstruoti visas karkaso galimybes, reikalinga tokia testinė duomenų bazės schema:

- Bent 5 lentelės
- Bent 1 lentelė kuri turi ryšį į save
- Bent 1 lentelių pora kurios turi tarpusavyje daugiau nei vieną tos pačios krypties ryšį (pvz. *StraipsnioVertimai.OrginaloId*, *VertimoId->Straipsniai*
- Bent 1 lentelė su kompleksiniu raktu

Prototipas turi atlikti tokias funkcijas:

- Panaudoti *Manager* projektavimo šabloną duomenų objektams Turėti vartotojo sąsają kuri siejasi su duomenų objektais per *proxy* objektus mechanizmą
- Su objektais atlikti visas CRUD operacijas
- Atlikti veiksmus iššaukiančius kaskadines operacijas
- Būti realizuotas kaip *Windows Forms* aplikacija.

3.8. Išvados

Šiais aukšto lygio programavimo kalbų laikais programuotojai orientuojasi į dalykinės srities problemų sprendimą o ne technologines programų kūrimo subtilybes. Dėl to karkasai, kurie įgalina trivialias operacijas atlikti kiek įmanoma paprasčiau ir automatizuočiau turi didelę vertę.

Dirbant su objektais visada iškyla problema – kaip išsaugoti jų būseną kad ji būtų prieinama kitoms programoms ir kitiems tos programos egzemplioriams (*instance*). Tokia objektų savybė yra vadinama *Persistency*.

Analizės dalyje pagrindžiamas ORM poreikis ir kūrimo metodai, tai yra esminis šio magistro darbo aspektas. Šioje, projekto dalyje, yra aptariama kaip karkasas atrodo iš projektinės pusės ir kas yra reikalinga prototipui, demonstruojančiam jo panaudojimą, parengti.

4. PROTOTIPINĖ ORM KARKASO REALIZACIJA

4.1. Taikymo apžvalga

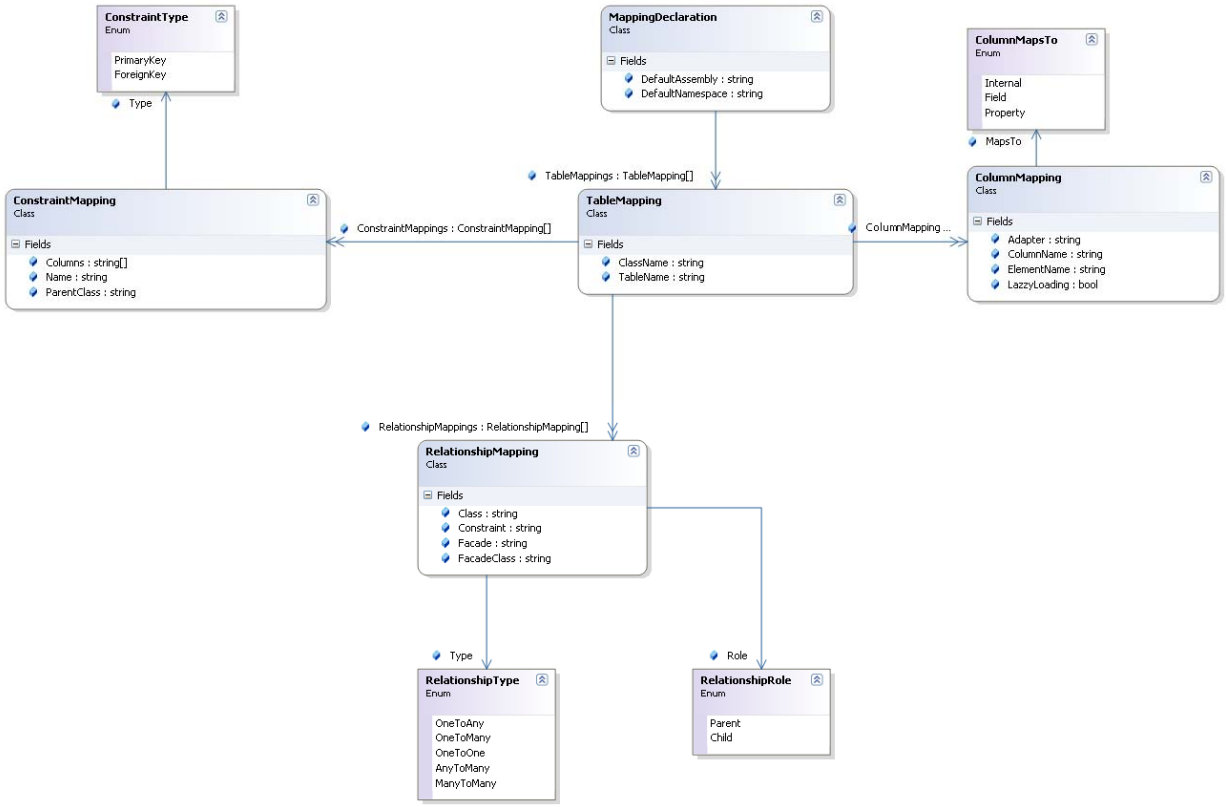
Kaip aprašyta ankstesniame skyriuje, prototipinis karkasas nepalaikys visų galimų funkcijų o tik tokias, kurios būtinos siūlomo modelio realizavimui. Duomenų klasės visiškai atitiks duomenų bazės lenteles o atvaizdavimo klasės tarnaus kaip aukštesnio lygmens sluoksnis.

4.2. Karkaso klasių modelis

Panagrinėsime karkaso klases. Jos yra suskirstytos į kelias vardų aibes pagal atliekamą funkcionalumą.

4.2.1. Susiejimo aprašų struktūros

Sekančioje diagramoje (19 pav.) pavaizduotos susiejimo XML failus reprezentuojančios struktūros kurios yra suprojektuotos taip, kad būtų galima pasinaudoti .NET karkaso XML serializavimo priemonėmis.



19 pav. Susiejimo aprašų struktūros

Trumpai aptarsime šių klasių paskirtis.

ConstraintType išvardijimas apibrėžia galimus apribojimų tipus. PrimaryKey yra skirtas pirminių raktų, ForeignKey – išorinių raktų apribojimams žymėti.

ColumnMapsTo nurodo į kur atvaizduojama duomenų bazės lentelės lauko reikšmė. Internal reiškia kad laukas atvaizduojamas į karkaso vidines struktūras ir duomenų klasių interfeisuose nėra pasiekiamas, Property reiškia kad laukas atvaizduojamas į duomenų klasės savybę ir Field – į duomenų klasės lauką.

RelationshipRole nurodo kokią funkciją ryšyje atlieka klasė. *Parent* reiškia kad klasė šiame ryšyje atlieka tėvo vaidmenį, *Child* – vaiko. Tėvinė klasė gali turėti kelias vaikines klases jeigu ryšys yra vienas-su-daug.

RelationshipType nurodo ryšio tipą. Šiuo metu karkasas palaiko tik vieną ryšio tipą – *OneToMany* – vienas-su-daug.

ColumnMapping aprašo duomenų bazės lentelės lauko atvaizdavimą. *ColumnName* yra lauko pavadinimas, *ElementName* yra klasės elemento pavadinimas į kurį jis yra atvaizduojamas (Internal elementai irgi turi pavadinimus, kurie turi būti unikalūs klasės, su kuria jie susieti, atžvilgiu). Kiti parametrai turi reikšmę tik išimtiniais atvejais ir tik dabartinėje karkaso versijoje.

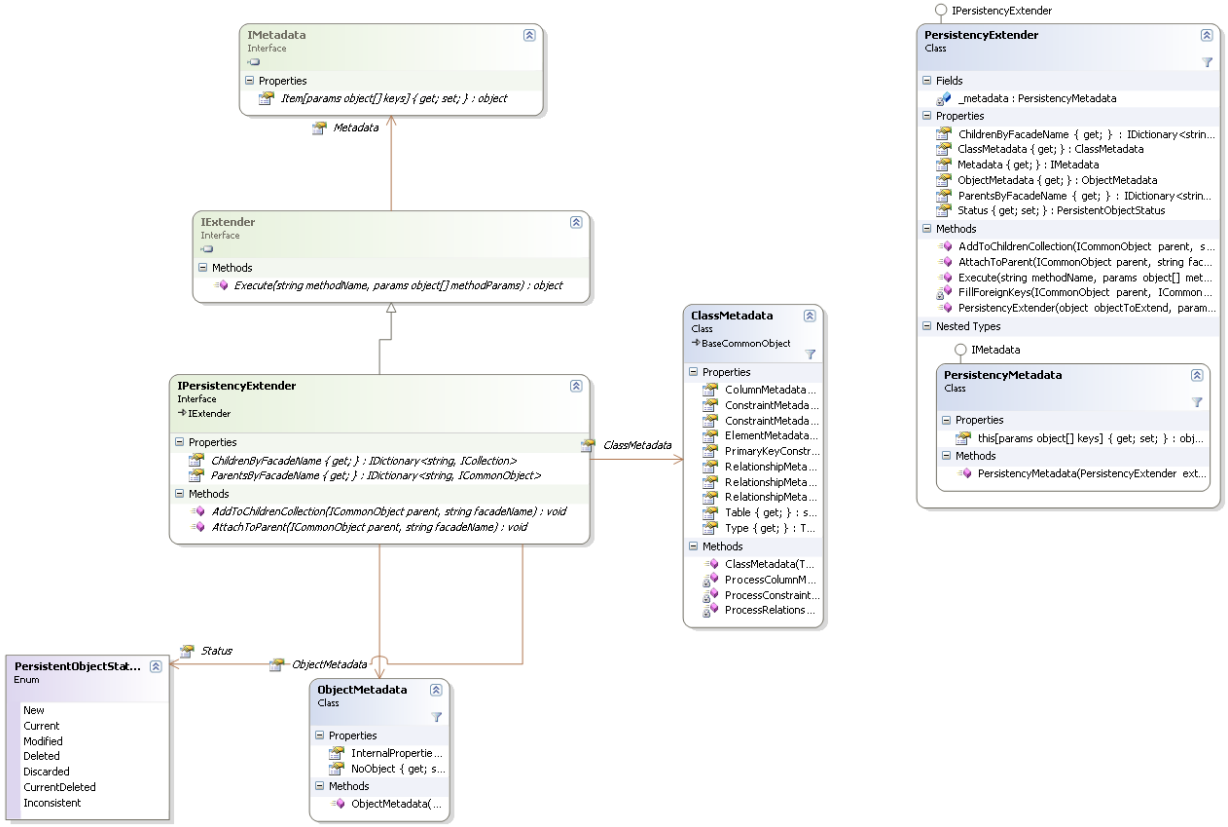
ConstraintMapping aprašo apribojimus. Kiekviena lentelė turi pirminio rakto apribojimą, karkasui būtina žinoti kokie laukai (*Columns*) sudaro pirminio rakto apribojimus, taigi kiekvienos lentelės susiejimo apraše turi būti bent vienas *PrimaryKey* apribojimas. Išorinio rakto apribojimai dar nurodo su kokia klase jie sieja šią klasę (*ParentClass*). Visi apribojimai turi turėti globaliai unikalius pavadinimus (*Name*).

RelationshipMapping klasės skirtos ryšių aprašymui. Kiekvienos ryšyje dalyvaujančioje klasės apraše yra po *RelationshipMapping* skirtą klasės atliekamam vaidmeniui ryšyje aprašyti (*Role*). Yra nurodoma su kokia klase siejasi ryšys (*Class*), koks yra savybės, kuri atstovauja kitą ryšio klasę šioje klasėje pavadinimas (*Facade*) ir klasė (*FacadeClass*) bei koks išorinio rakto apribojimas apibrėžia ryšį.

TableMapping klasė aprašo kokia lentelė (TableName) su kokia klase (ClassName) yra susiejama. TableMapping aprašai yra talpinami MappingDeclaration klasėje, kurios reprezentacija XML dokumente yra šakninis XML dokumento elementas.

4.2.2. Praplėtimo mechanizmas

Praplėtimo mechanizmas yra skirtas praplėsti tam tikras klases norimu funkcionalumu. Kadangi karkasas dar priedo yra IT-Europe portalo dalis tai klasių sąsajose yra nemažai elementų, kurie turi panaudojimo prasmę tik IT-Europe portale. Diagrama pateikiama 20 pav.



20 pav. Praplėtimo mechanizmas

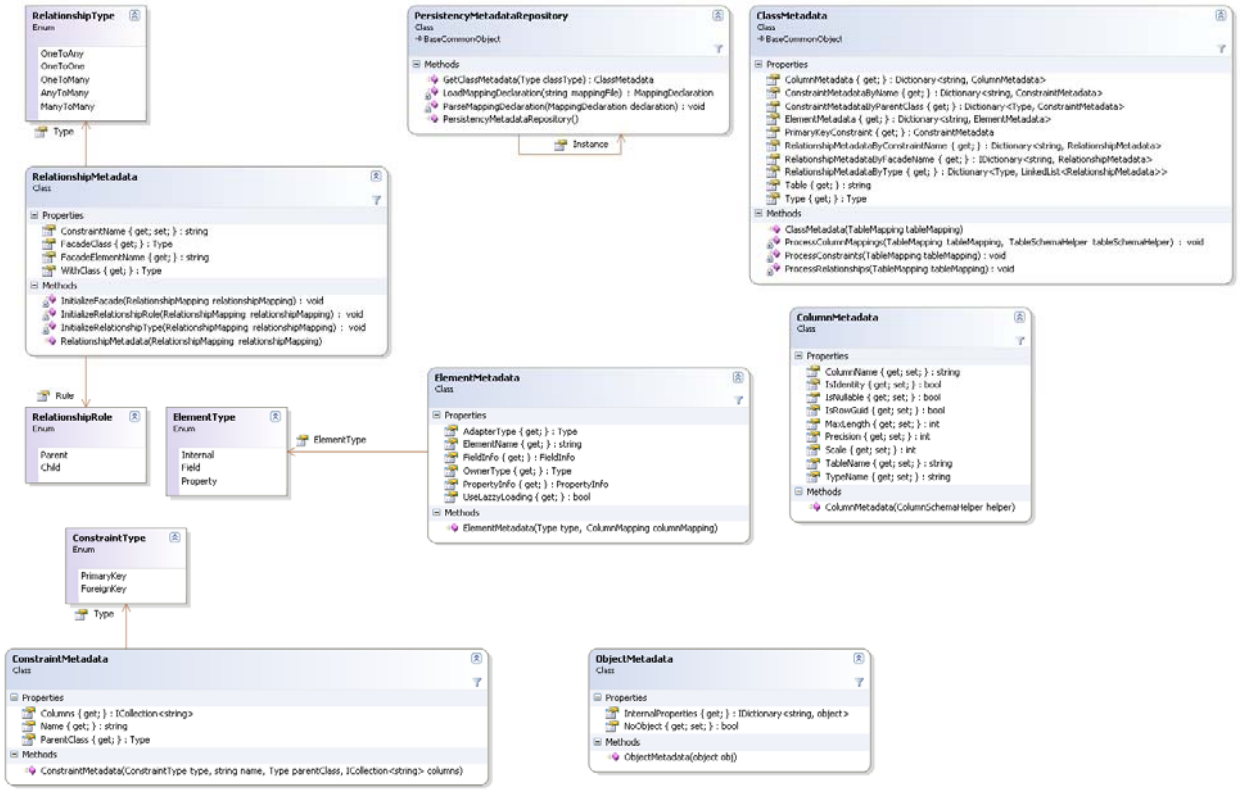
PersistentObjectStatus išvardijimas aprašo visas galimas objekto išsaugojimo būsenas. Naujai sukurtas objektas įgyja būseną *New*. Kuomet jis išsaugomas į duomenų bazę ar išgaunamas iš ten jis būna būsenos Current. Jeigu objekto kuri nors savybė yra pakeičiama objektas įgauna būseną *Modified*. Jeigu objektas pažymimas kaip ištrintas (*Deleted*) ir išsaugomas turima atmintyje objekto būsena patampa *CurrentDeleted* kuri indikuoja kad objektas buvo pašalintas iš duomenų bazės ir tolesnės operacijos su šiuo objektu ignoruojamos. *Discarded* būsena objektui suteikiama kuomet naujas objektas yra pažymimas kaip ištrintas prieš jį išsaugant – tuomet visos saugojimo operacijos ignoruoja objektą. *Inconsistent* būseną objektas įgyja dėl klaidos ar sinchronizavimo problemų, pavyzdžiui, kai kitas vartotojas atnaujino duomenų bazę.

ClassMetadata pagalbinė klasė saugo klasės metaduomenis, kurie turi prasmę karkase, pagrinde nuskaitytas iš XML susiejimo failo struktūras.

ObjectMetadata pagalbinė klasė saugo objekto metaduomenis reikalingus karkasui, pavyzdžiui vidines objekto savybes.

4.2.3. Metaduomenų struktūros karkaso viduje

Sekančioje diagramoje (21 pav.) pateikiamos metaduomenų struktūros, kuriose karkasas saugo informaciją apie klasių susiejimą su lentelėmis bei pagalbinius susiejimo objektus.



21 pav. Metaduomenų struktūros karkaso viduje

Dauguma klasių čia atitinka susiejimo failų struktūras. Išvardijimai yra visiškai identiški.

Valdiklis *PersistencyMetadataRepository* išgauna metaduomenis apie klases iš susiejimo failų, juos išanalizuoja ir apdoroja ir užkešuoja. *GetClassMetadata* metodas grąžina išsaugojamos klasės metaduomenis. *LoadMappingDeclaration* užkrauna susiejimo failą, *ParseMappingDeclaration* išanalizuoja susiejimo failo struktūras.

Klasė *ClassMetadata* saugo konkrečios klasės susiejimo metaduomenis. Šios klasės savybės leidžia surasti norimas laukų ar apribojimų bei kitas struktūras pagal jų pavadinimus ir kitus požymius.

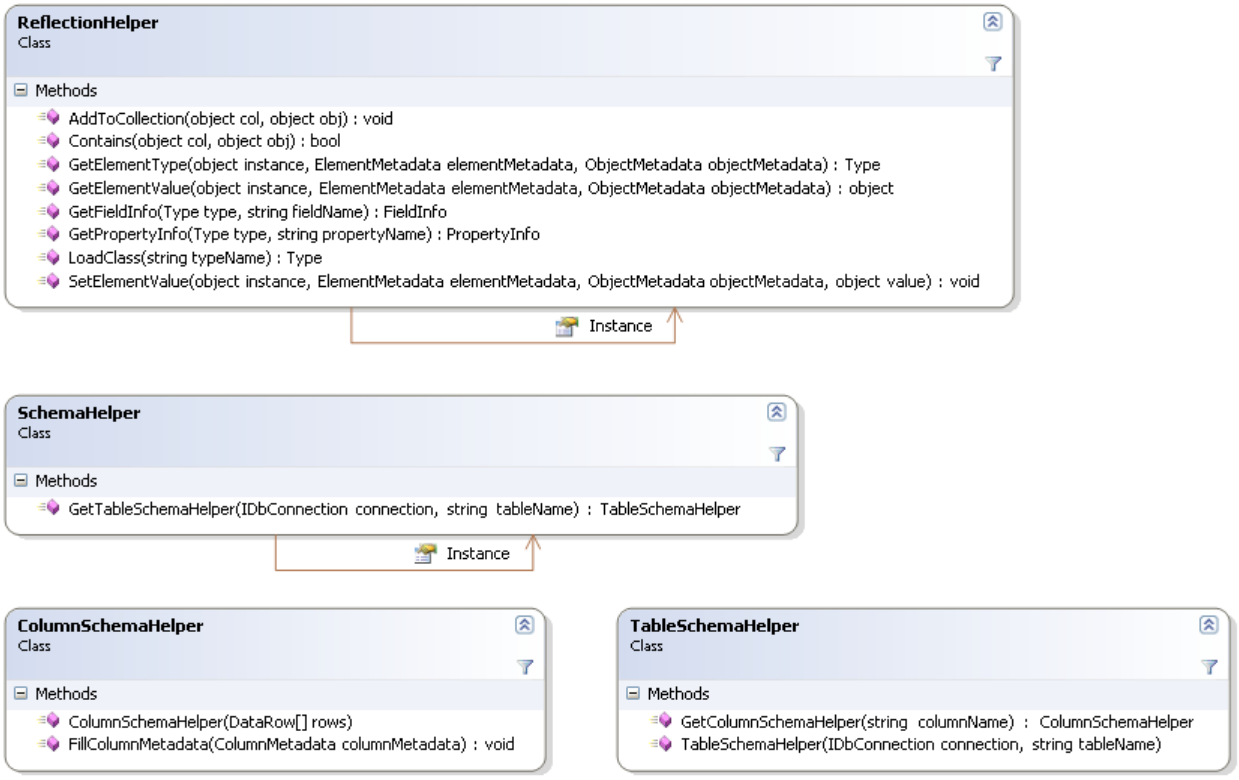
ColumnMetadata saugo informaciją apie lauko atvaizdavimą į objekto struktūras. Karkasas išgauna iš duomenų bazės ir įvairią informaciją apie fizinį lauką, tokią kaip kad jo ilgis ar fizinis tipas ir išsaugo šioje struktūroje.

Visos kitos klasės yra tik susiejimo failo struktūrų wrapperiai. Vienintelis dar svarbus nepaminėtas dalykas yra tas, kad klasės *ObjectMetadata* savybių žodynas *InternalProperties* leidžia

prieiti prie karkase saugojamų vidinių objekto savybių, į kurias atsivaizduoja duomenų bazių lentelių laukai, kurie susiejimo faile pažymimi kaip `mapsTo="Internal"`.

4.2.4. Pagalbinės klasės

Pagalbinių klasių diagrama pateikiama sekančiame paveiksle. Diagrama pateikiama šioje dokumento vietoje todėl kad ji yra logiškai susijusi su 4.2.3 skyriuje aprašyta klasių diagrama.



22 pav. Pagalbinės klasės

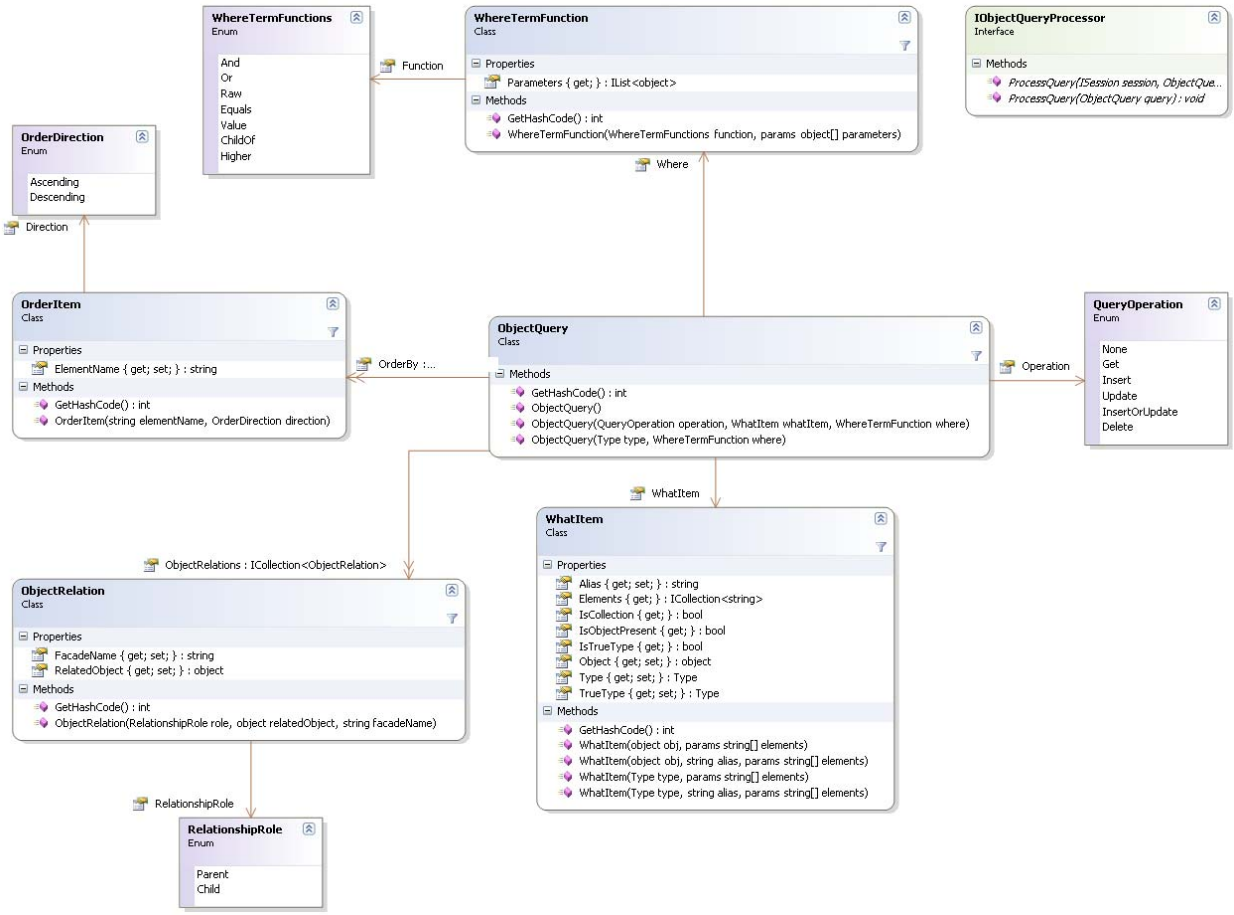
ReflectionHelper suteikia pagalbines priemones, perdengiančias *System.Reflection* funkcijas ir įgalinančias visam karkasui funkcionuoti – galimybė išgauti informaciją apie klasės struktūrą vykdymo metu, gauti ir priskirti objekto elemento reikšmę, padėti objektą į bet kokio tipo kolekciją.

SchemaHelper išgauna informaciją apie lentelę nuskaitydama iš duomenų bazės sisteminių lentelių informaciją. Viena iš pagrindinių *SchemaHelper* funkcijų yra informacijos apie lentelių laukus išgavimas.

TableSchemaHelper ir *ColumnSchemaHelper* yra klasės, kurios atitinkamai atsakingos už lentelės ir lauko sisteminės informacijos išgavimą.

4.2.5. Užklausų formavimo klasės

Sekančioje diagramoje (23 pav.) pateiktos užklausų formavimo klasės. Kuriant šių klasių egzempliorius galima suformuoti įvairias užklausas, jas išsaugoti ir perduoti karkaso valdikliams jas vykdyti.



23 pav. Užklausų klasės

WhereTermFunction yra skirta termams aprašyti. Kai kurios *WhereTermFunction* kaip parametrus gali priimti kitas *WhereTermFunction* (Pvz. *And*, *Or*) taigi galima sukonstruoti sudėtingas užklausas panaudojant šių funkcijų medį. *WhereTermFunctions* išvardijimas nurodo galimas operacijas.

OrderItem aprašo rikiavimo kriterijus. Rikiavimas atliekamas *ObjectQuery.OrderBy* elementų išsidėstymo tvarka.

ObjectRelation skirta aprašyti filtravimams pagal tarpobjektinius ryšius. Galima nurodyti kad būtų atrenkami objektai kurie yra pateikto objekto vaikai arba tėvas.

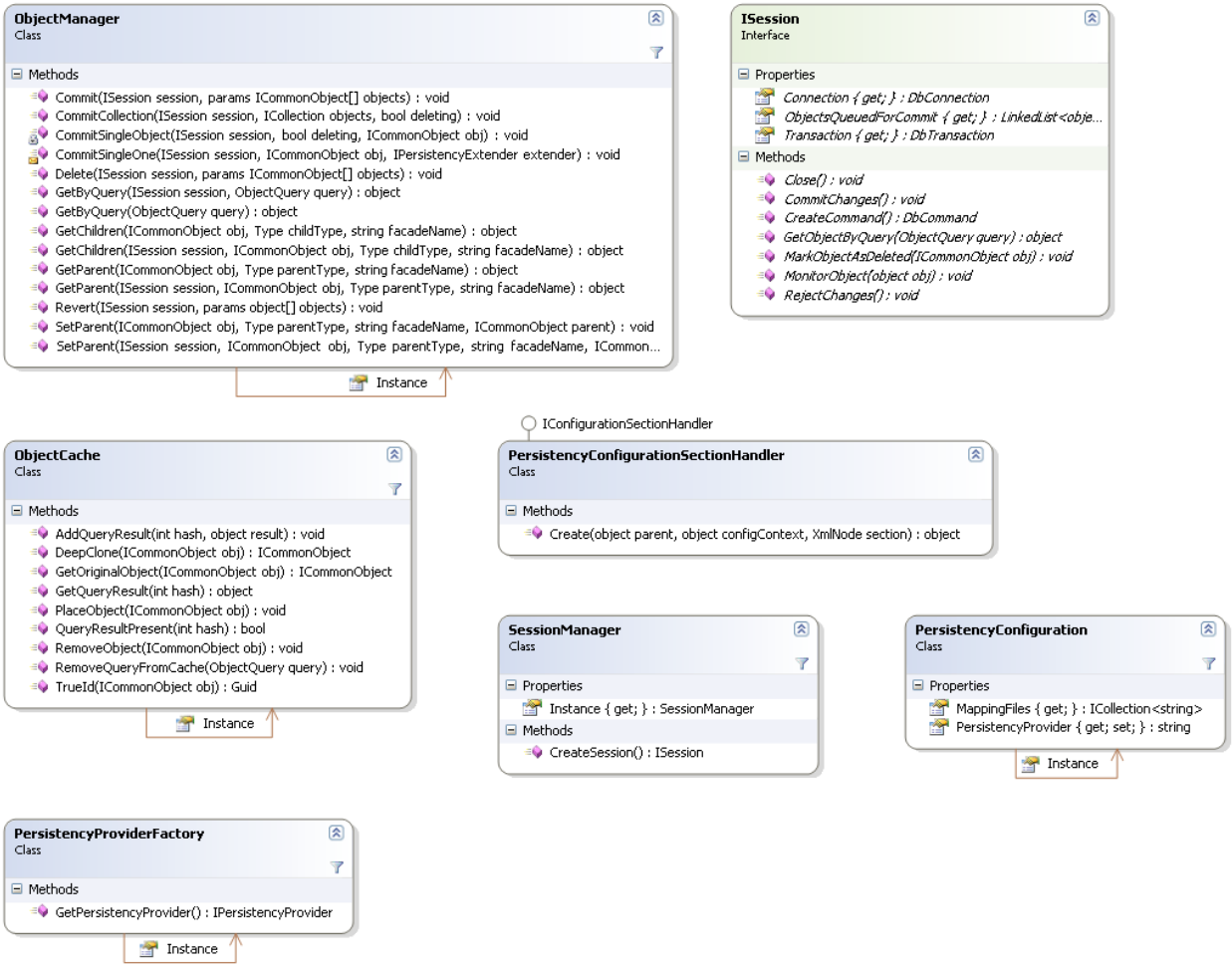
WhatItem nusako užklausos operuojamų objektų tipus ar saugo pačius objektus. Išgavimo operacijai reikia žinoti kokio tipo objektas(ai) turi būti grąžinti. Jeigu paduodama objektų kolekcijos tipas

karkasas bando grąžinti visus sąlygas tenkinančius objektus, o jeigu tik objekto tipas tuomet grąžinamas tik pirmas objektas tenkinantis sąlygą. Jeigu objektų tenkinančių sąlygą nerandama pirmu atveju grąžinama tuščia kolekcija o antru atveju null. Kolekcijos turi būti *generic*. Operacijoms, kurios manipuliuoja jau esančiais objektais reikia paduoti pačius objektus ar jų kolekcijas.

Klasė *ObjectQuery* agreguoja visas kitas užklausos formavimo klases ir saugo visą informaciją apie užklausą ir jos rezultatus.

4.2.6. Karkaso pagrindinių klasių valdiklių modelis

Pagrindinių karkaso klasių diagrama pateikiama sekančiame paveiksle 24 pav.



24 pav. Karkaso pagrindiniai valdikliai

ISession yra pagrindinis interfeisas, per kurį pasiekiamos karkaso funkcijos. Nauja sesija sukuriama *SessionManager.CreateSection*, uždaroma.*ISession.Close*. Per *ISession* galima išgauti objektus (*GetObjectsByQuery*) ir juos užregistruoti sesijoje (*MonitorObject*) nei išsaugoti pakeitimus (*CommitChanges*). Veikimo principas yra analogiškas *Hibernate* ([13, 4 sk.]).

Kiti valdikliai paprastai tiesiogiai nenaudojami. *ObjectManager* realizuoja daugumą ISession funkcijų bei susijusių objektų išgavimo/išsaugojimo operacijas pasiekiamas iš duomenų klasių sa-

vybių get ir set metodų. *ObjectCache* saugo duomenų originalias kopijas, lyginant objektus su originalais nustatoma ar yra pakeitimų. *PersistencySectionHandler* ir *PersistencyConfiguration* nuskaito karkasui skirtą informaciją iš aplikacijos konfigūracijos failo.

4.3. Realizacijos išvados

Magistrinio darbe svarstomas problemas ir siūlomus jų sprendimus iliustruoja realizacija, susidedanti iš demonstracinio ORM karkaso ir demonstracinės programos, kuri pritaiko ORM karkaso funkcijas tam tikros dalykinės srities objektų valdymui. Realizacija praktiškai pademonstruoja ORM karkaso kūrimo subtilybes ir veikimą, demonstracinė programa parodo kaip objektinio-reliacinio susiejimo modelis gali būti suprojektuotas pritaikant MVC principus.

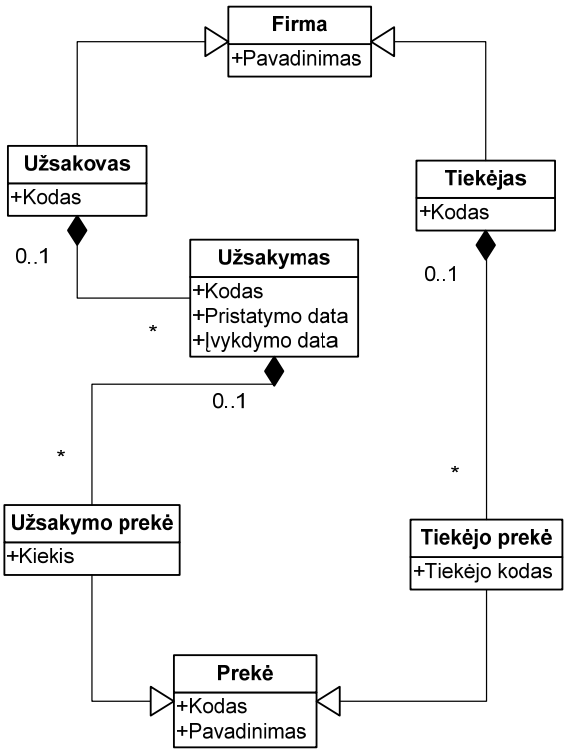
Apibendrinant galima teigti, kad realizacija pateisino jai iškeltus uždavinius.

5. EKSPERIMENTO APRAŠYMAS

5.1. Pavyzdinė dalykinė sritis

Paimkime firmos, kuri užsiima krovinių pervežimu dalykinę sritį. Firma kaupia duomenis apie prekes, jų tiekėjus, užsakovus ir jų užsakymus. Taigi esminės dalykinės srities esybės yra tokios: *Firma, Tiekėjas, Užsakovas*, *Prekė*, *Užsakymas*. Tiek *Tiekėjas* tiek *Užsakovas* yra *Firma*, tariame kad su individualiais asmenimis paprastumo dėlei nebendraujama. Tiekėjas turi tiekiamų prekių sąrašą ir prekės gali būti pristatytos įvairiems užsakovams įvairiais kiekiais. Jeigu ryšys tarp objektų turi tam tikrus papildomus atributus, saugančius informaciją, jis turi būti atvaizduojamas ir objektiniame modelyje taip kad juo būtų galima manipuliuoti.

Taigi dalykinės srities koncepcinė klasių diagrama atrodys taip, kaip pavaizduota 25 pav.



25 pav. Dalykinės srities koncepcinė klasių diagrama

Tiek duomenų bazės diagrama tiek dalykinės srities duomenų klasių diagrama bus kuriamos šios diagramos pagrindu tačiau paveldėjimas bus realizuojamas ryšių pagalba ir atsispindės tik atvaizdavimo klasėse.

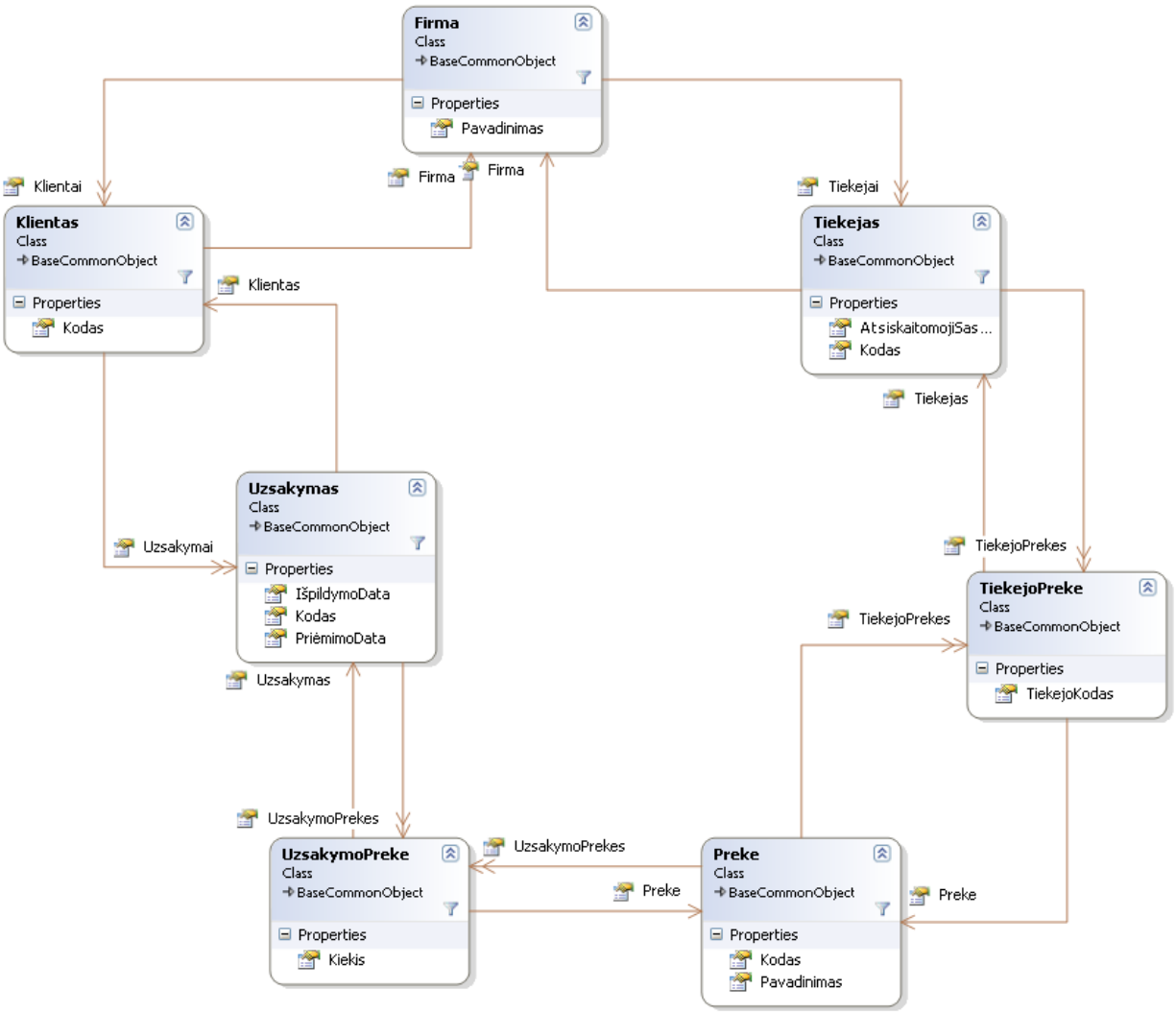
Ta pati firma gali būti tiek tiekėjas, tiek užsakovas, tačiau ji gali atlikti tik po vieną tokią funkciją. Prekės tuo tarpu gali būti gaunamos iš įvairių tiekėjų ir dalyvauti įvairiuose užsakymuose.

5.2. Pavyzdinės dalykinės srities klasių modeliai

Pateiksime dalykinės srities duomenų, atvaizdavimo ir valdiklių klasių modelius.

5.2.1. Dalykinės srities duomenų klasių modelis

Panagrinėsime dalykinės srities duomenų klasių modelio diagramą (26 pav.).



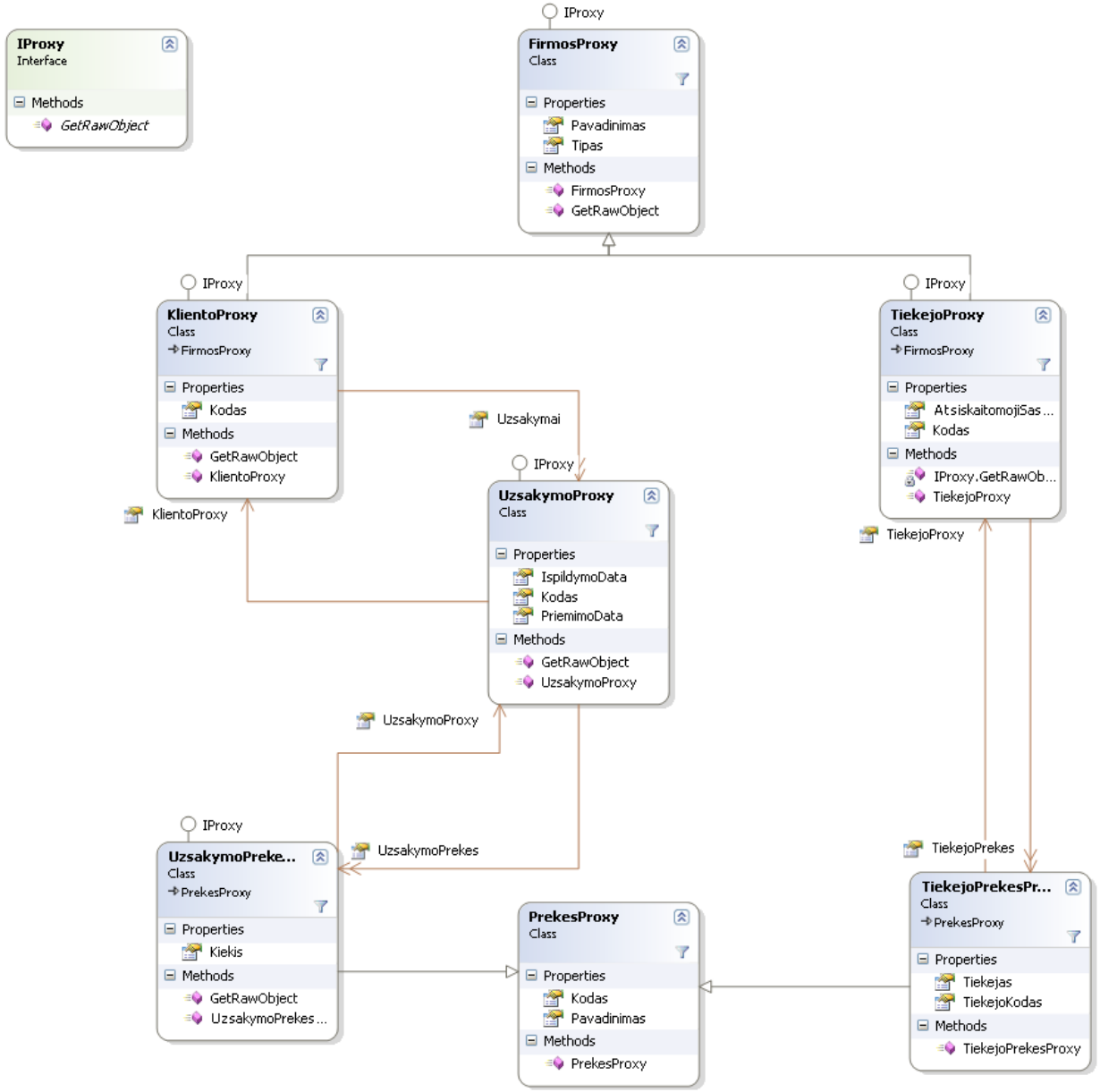
26 pav. Dalykinės srities duomenų klasių modelio diagrama

Kaip matyti iš diagramos, visos susijusios klasės tarpusavyje turi du ryšius – vienas tradicinis vienas-su-daug jungiantis tėvinį objektą su vaikiniu, kitas ryšys yra nuoroda į tėvą vaikinėje klasėje. Nors klasė *Klientas* realiai paveldi iš klasės *Firma*, šias klases sujungiame paprasta asociacija, dalykinėje klasėje klientas turės firmą savo viduje o atvaizdavimo klasėje klientas paveldės iš firmos. Tokie pertvarkymai galimi kadangi paveldėjimas yra ne kas kita kaip automatizuota kompozicija su įkomponuotos klasės interfeiso iškėlimu į konteinerio klasę.

Programiniai objektai visada ryšius realizuos taip kad jie būtų dviejų krypčių: nuoroda į tėvinį objektą ir vaikinių objektų nuorodos. Tai ne visais atvejais turi prasmę, pavyzdžiui firma gali atitikti tik vieną klientinį objektą sistemoje, taigi tam yra reikalingos priemonės apribojimams įvesti o nereikalingi ryšiai gali būti neatvaizduojami atvaizdavimo klasėse.

5.2.2. Dalykinės srities sąsajos klasių modelis

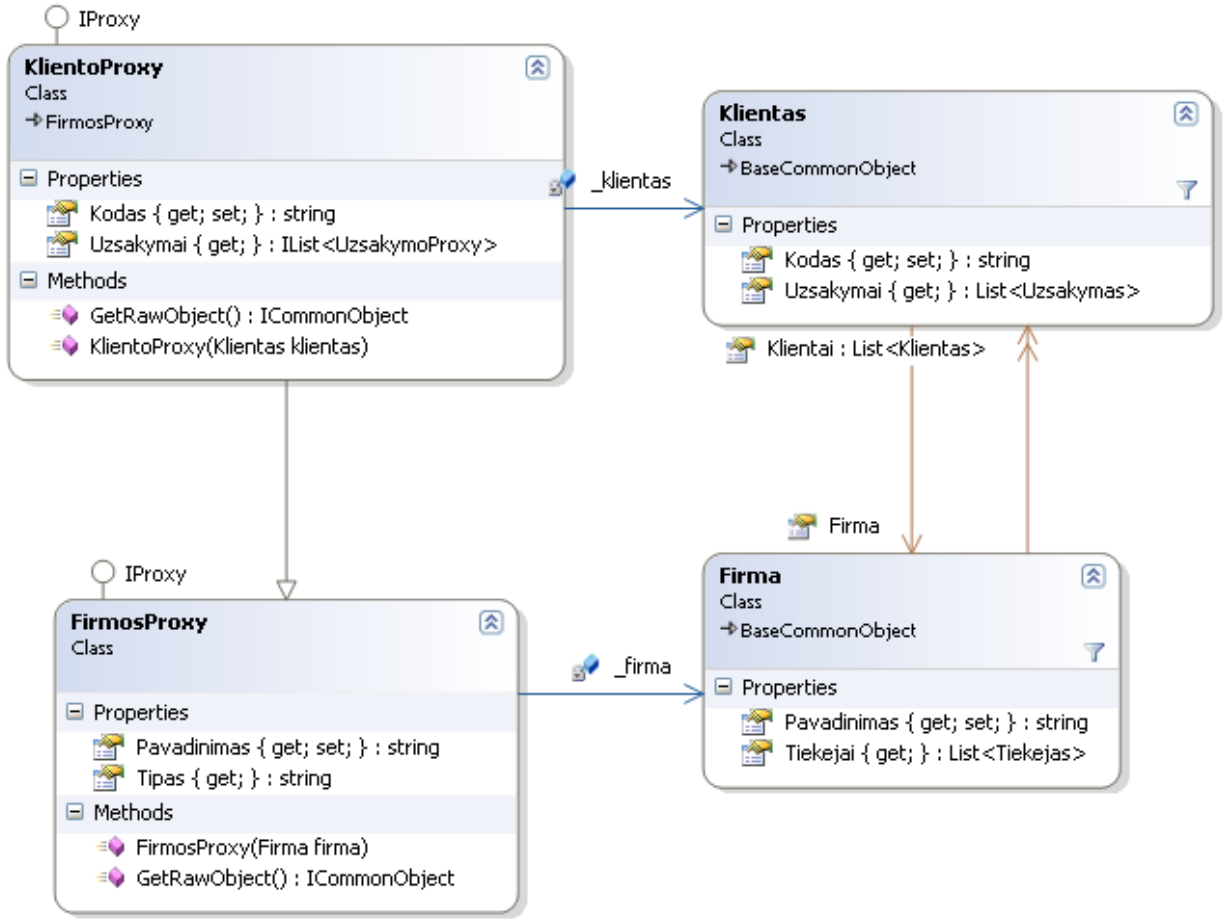
Panagrinėsime dalykinės srities atvaizdavimo klasių modelį, pateiktą 27 pav.



27 pav. Dalykinės srities sąsajos klasių modelio diagrama

Čia pateikiamos atvaizdavimo klasės agreguoja savyje visą reikiamą informaciją atitinkamai esybei atvaizduoti. Demonstraciniame prototipe pasitenkinsime po viena atvaizdavimo klase kiekvienai dalykinei klasei, esant poreikiui dalykinė klasė gali turėti kelias atvaizdavimo klases. Kaip matyti iš diagramos, klasių ryšiai yra panašūs į ryšius koncepcinėje dalykinės srities diagramoje.

Kiekviena atvaizdavimo klasė yra susijusi agregacijos ryšiais su duomenų klase. Sekančioje diagramoje (28 pav.) detalizuojamas atvaizdavimo ir duomenų klasių sąryšis.



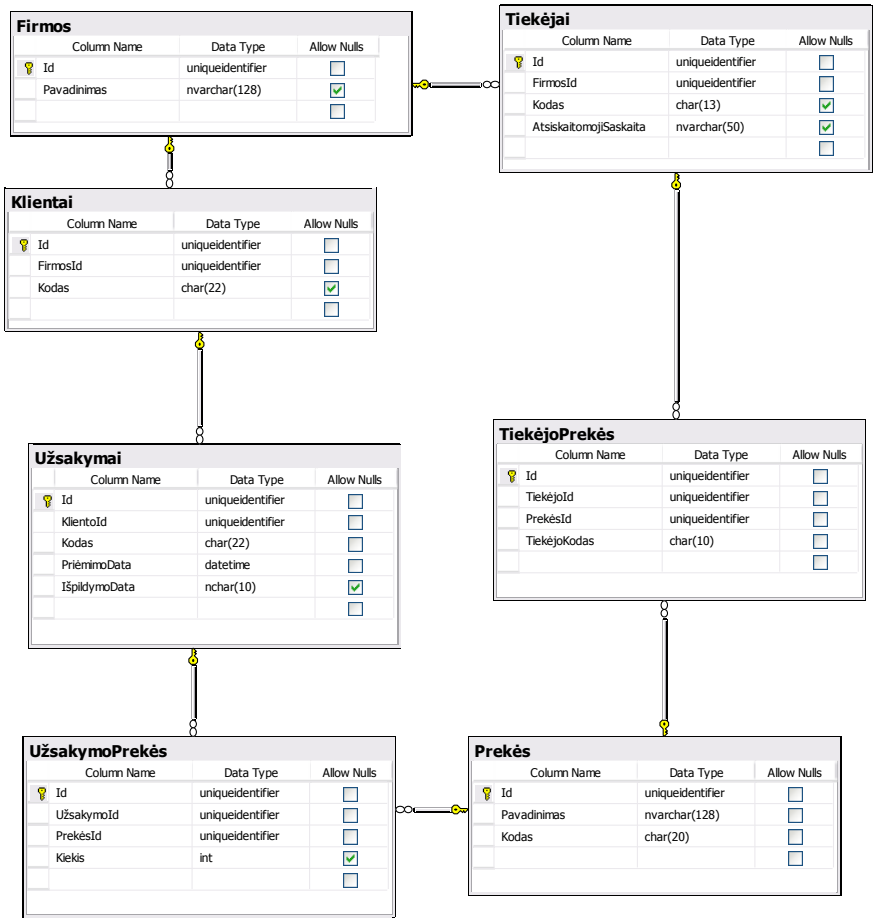
28 pav. Duomenų ir sąsajos klasių sąryšis

Kaip matyti iš diagramos, atvaizdavimo klasės yra susijusios su duomenų klasėmis per agregacinius ryšius. Atvaizdavimo klasės panaudoja paveldėjimą tam, kad panaudojimo atžvilgiu klasių hierarchija atitiktų dalykinės srities esybių hierarchiją.

IProxy interfeisas įgalina išgauti su atvaizdavimo objektu susijusį duomenų objektą. Daugiau su duomenų klasėmis susijusių elementų atvaizdavimo klasių interfeisuose neturi būti, visi kiti veiksmai turi būti atliekami per valdiklių klases, kurios paprastai relizuojamos pagal Singleton projektavimo šabloną.

5.1. Dalykinės srities duomenų bazės modelis

Sekančiame paveiksle (29 pav.) pateikiamas dalykinės srities duomenų bazės modelis. Karkaso veikimui nereikalingos jokios papildomos lentelės.



29 pav. Dalykinės srities duomenų bazės modelis

Kadangi lentelių paskirtys atitinka anksčiau aprašytų esybių paskirtis aptarsime tik jų realizavimo SQL serveryje ypatumus.

Visos lentelės turi po dirbtinį identifikatorių kuris tarnauja kaip pirminis raktas. Šio identifikatoriaus tipas yra *uniqueidentifier* tam, kad esant reikalui būtų galima be problemų sulieti lenteles ir skirtingų tipų objektus programoje.

5.2. Dalykinės srities objektinio-reliacinio susiejimo aprašas

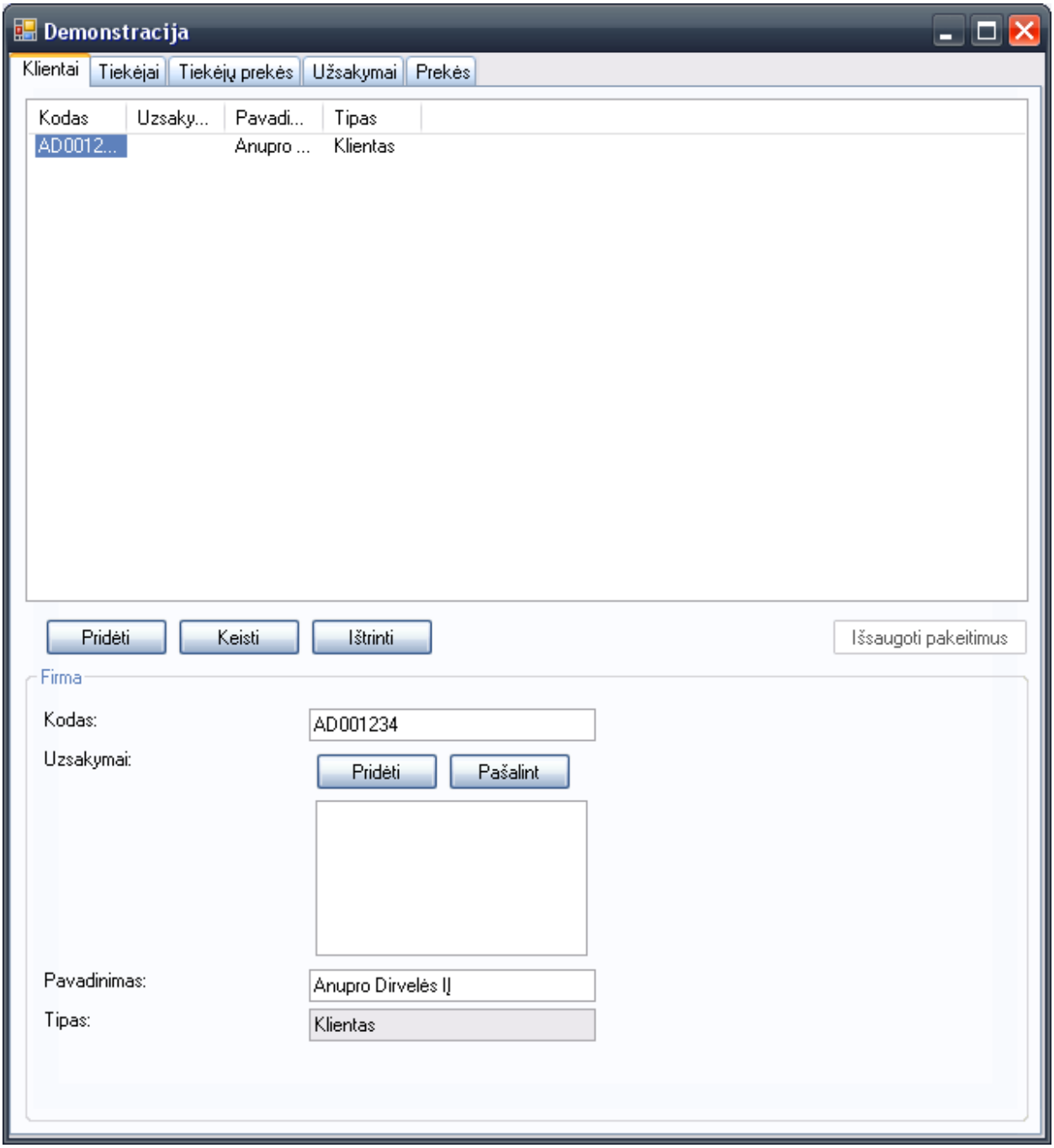
Čia pateikiamas tik epizodas iš aprašo, skirtas klasei Užsakymas:

```
<classToTableMapping className="TestSuite.Data.Uzsakymas,TestSuite" tableName="[dbo].[Užsakymai]">
   <columnMappings>
      <columnMapping columnName="[Id]" elementName="Id" mapsTo="Internal" />
      <columnMapping columnName="[Kodas]" elementName="Kodas" mapsTo="Property" />
      <columnMapping columnName="[PriėmimoData]" elementName="PriėmimoData" mapsTo="Property"/>
      <columnMapping columnName="[IšpildymoData]" elementName="IšpildymoData" mapsTo="Property"/>
      <columnMapping columnName="[KlientoId]" elementName="KlientoId" mapsTo="Internal" />
   </columnMappings>
   <constraints>
      <constraint name="PK_Uzsakymai" type="PrimaryKey">
         <constraintColumns>
            <constraintColumn>[Id]</constraintColumn>
         </constraintColumns>
      </constraint>
      <constraint name="FK_Uzsakymai_Klientai" type="ForeignKey" parentClass="TestSuite.Data.Klientas,TestSuite">
         <constraintColumns>
            <constraintColumn>[KlientoId]</constraintColumn>
         </constraintColumns>
      </constraint>
   </constraints>
   <relationships>
      <relationship role="Child" ofClass="TestSuite.Data.Klientas,TestSuite" facade="Klientai"
facadeClass="TestSuite.Data.Klientas,TestSuite" constraint="FK_Uzsakymai_Klientai" />
      <relationship role="Parent" ofClass="TestSuite.Data.UzsakymoPreke,TestSuite" facade="UzsakymoPrekes"
facadeClass="System.Collections.Generic.List`1[[TestSuite.Data.UzsakymoPreke,TestSuite]]"
constraint="FK_UzsakymoPrekes_Uzsakymai" />
      </relationships>
</classToTableMapping>
```

Susiejimo aprašas atitinka anksčiau pateiktas struktūras, todėl paaiškinsime tik likusius svarbius aspektus. Kaip matyti iš aprašo, duomenų bazės laukų pavadinimai apgaubiami laužtiniais skliaustais tam, kad jiems būtų galima suteikti bet kokius pavadinimus įskaitant ir tokius kurie persidengia su *Transact-SQL* raktiniais žodžiais. Visi tipai čia pateikiami pilna forma (pilnas kelias iki klasės per vardų aibes bei asemblio, kuriame tipas randasi, pavadinimas). Atkreiptinas dėmesys į tai kaip aprašomi *generic* kolekcijų tipai.

5.1. Eksperimentinės programos vartotojo sąsaja

30 pav. pateikiamas tipinis eksperimentinės programos langas. Vartotojas turi galimybę peržiūrėti objektus, pridėti naujus, keisti esamus, ištrinti nebereikalingus, susieti objektus ryšiais ir t.t.



30 pav. Eksperimetinės programos tipinis langas

5.2. Eksperimento realizacijos priemonės. Pasirinktų eksperimento sprendimų privalumai ir trūkumai

Karkasas ir demonstracija yra realizuojami .NET 2.0 aplinkoje. Jokie trečiųjų šalių komponentai nenaudojami. Kaip DBVS naudojamas Microsoft SQL Server 2005. Demonstracija yra Windows Forms aplikacija, tuo tarpu IT-Europe projektas kur taip pat panaudojamas karkasas yra ASP.NET Web aplikacija.

Pasirinkti realizavimo sprendimai įgalina pademonstruoti demonstracinį ORM karkasą, kuris aptarnauja objektinę struktūrą, suprojektuotą pagal šiame darbe siūlomą modelį tačiau demonstracinis prototipas neaprėpia visų galimų karkaso funkcijų ir ORM panaudojimo atvejų.

6. IŠVADOS

1. Nustatyta, kad egzistuoja semantinė spraga, kylanti dėl skirtingo objektinio ir reliacinio modelių dalykinės srities atvaizdavimo pjūvio.
2. Ištirti minėtos semantinės spragos problemos sprendimo būdai: objektinės duomenų bazės panaudojimas, objektinio-reliacinio susiejimo panaudojimas, reliacinės duomenų bazės su objektinės duomenų bazės sąsaja panaudojimas ir kt., jie palyginti tarpusavyje ir nustatyta, kad praktiškai tinkamiausias panaudojimui sprendimas yra objektinio-reliacinio susiejimo karkasas.
3. Išanalizuotos tipinės objektinio-reliacinio susiejimo karkaso funkcijos: SQL generavimas, identifikavimo valdymas, susiejimo valdymas, CRUD operacijų API, pakeitimų nustatymas ir kt.
4. Tarpusavyje palyginti keli populiarūs ORM karkasai remiantis jų teikiamomis funkcijomis, kartu su jais analizuojant teoriškai įmanomomis specialiai kuriamo karkaso galimybes. Nustatyta, kad geriausiai reikiamus poreikius išpildyti gali naujas karkasas.
5. Aprašomas objektinio-reliacinio susiejimo modelio, panaudojančio dalį reliacinio ir dalį objektinio modelių privalumų, principas. Šio modelio esmė yra MVC principų pritaikymas objektiniam modeliui, kurio pasekoje atvaizdavimo klasės yra artimos dalykiniai sričiai, o duomenų klasės artimos reliacinės duomenų bazės schemai.
6. Aptariami objektų išreiškimo aukšto lygio programavimo kalbomis ypatumai, turintys įtakos kuriamam karkasui, nustatomi skirtumai tarp *get/set* metodų ir atributų. Išnagrinėjus kodo metaduomenų pritaikymą ORM karkasų susiejimo konfigūravimui aprašyti, nustatyta, kad kodo metaduomenys patogesni, bet išoriniai metaduomenys lankstesni.
7. Pateikiami funkciniai reikalavimai karkasui, iš kurių būtinos funkcijos yra šios: susiejimo konfigūravimas, lentelių ir klasių susiejimas, ryšių tar objektų palaikymas, kaskadinių operacijų palaikymas, užklausų formavimas, pakeitimų išsaugojimas, tingus krovimas, ciklinių ryšių aptikimas, *disconnected* pobūdžio režimas, kompozicinių raktų palaikymas.
8. Karkasas ir eksperimentinė programa realizuojamos .NET 2.0 aplinkoje, naudojama Microsoft SQL Server 2005 DBVS. Eksperimentinė programa dirba su krovinių pervežimo firmos dalykinės srities esybėmis.
9. Darbo rezultatai gali būti panaudoti sprendžiant panašias problemas, iškylančias realizuojant taikomąsias aplikacijas, naudojančias reliacines duomenų bazes. Darbe analizuojami objektinio reliacinio susiejimo principai ir modelis pasitarnauja pasirenkant tinkamą sprendimą konkrečiai situacijai ir suteikia įžvalgos į komercinių objektinio-reliacinio susiejimo karkasų, kurie įprastinėmis sąlygomis yra juodos dėžės, funkcionavimo pagrindus.

7. LITERATŪROS SĄRAŠAS

[1] TSICHRITZIS, D.; NIERSTRASZ, O. Fitting Round Objects Into Square Databases: *European Conference on Object-oriented Programming.* Geneva : Springer Verlaag, 1988

[2] AMBLER, W. S. *Mapping objects to relational databases - What you need to know and why*. 2000. [žiūrėta 2006-12-15] Prieiga per internetą: http://www-128.ibm.com/developerworks/webservices/library/ws-mapping-to-rdb/

[3] LITWIN, P**.** *Fundamentals of Relational Database Design.* Sybex, 1994

[4] YODER, W. ETC. *Connecting Business Objects to Relational Databases.* Proceedings of he 5th Conference on the Pattern Languages of Programs, Monticello-IL-EUA, August 1998. 16

[5] AMBLER, S.W. *The Fundamentals of Mapping Objects to Relational Databases. Agile database techniques*. John Wiley & Sons Inc, 2003

[6] SCHLEGELMILCH, J. *An Advanced Relationship Mechanism for Object-Oriented Database Systems*. Department of Computer Science, University of Rostock, Germany, 1996

[7] RAMANATHAN, CH. *Providing object-oriented access to existing relational databases*. Department of Computer Science, Missisipi State University. 1997, A Dissertation for PhD of Computer Science

[8] NOCK, C. *Data Access Patterns: Database Interactions in Object-Oriented Applications*. Addison Wesley, 2003. ISBN: 0-13-140157-2 .

[9] BOOCH, G. *Object-oriented analysis and design*. Addison Wesley, 1994. ISBN: 0-8053-5340-2.

[10] SPEELPENNING, J; DAUX, P; GALLUS, J. *Data Modeling and Relational Database Design.* Oracle Press, 2001, Vol. 1. Student Guide

[11] ATKINSON M. *The Object-Oriented Database System Manifesto*.

[12] MESEGUER, J.; QUAIN, X. *A Logical Semantics for Object-Oriented Databases*. Computer Science Laboratory, SRI Internaltional.

[13] BAUER, CH.; KING. G. *Hibernate in Action*. Manning Publications Co., 2005. ISBN: 1932394-15-X.

[14] JAMES, E. *Hibernate - A Developer's Notebook*. O'Reilly, 2004. ISBN: 0-596-00696-9.

[15] MARTIN, F. UML *Distilled: A Brief Guide to the Standard Object Modeling Language*, Third Edition: Addison Wesley, 2003. ISBN: 0-321-19368-7.