KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Donatas Gogys

Programų biznio sluoksnio karkaso kūrimo ir panaudojimo tyrimas

Magistro darbas

Darbo vadovas:

doc. Tomas Blažauskas

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Donatas Gogys

Programų biznio sluoksnio karkaso kūrimo ir panaudojimo tyrimas

Magistro darbas

Recenzentas

doc. Vytautas Pilkauskas

2010-05-27

Vadovas

doc. Tomas Blažauskas

2010-05-27

Atliko

IFM-4/2 gr. stud

Donatas Gogys

2010-05-27

Kaunas, 2010

TURINYS

1. ĮVADAS

1.1. Dokumento paskirtis

Šis dokumentas yra Kauno technologijos universiteto Informatikos fakulteto Programų inžinerijos katedros Programų sistemų inžinerijos magistro studijų magistro darbas. Šiame dokumente yra nagrinėjama ir supažindinama su naujausiomis programavimo technologijomis skirtomis kurti bei projektuoti programų karkasą, kuris būtų kaip naujos kuriamos paskirstytos programinės sistemos architektūrinis pagrindas. Šio dokumento tikslas suteikti skaitytojui naudingą informaciją apie programų karkasų architektūrinį projektavimą ir karkaso susiejimą su duomenų baze, taip pat išryškinti įvairias problemas, kurios gali atsirasti blogai sukūrus naujos sistemos architektūrą, taip sumažinant ateities problemas.

Kadangi dokumente yra pateikiamos UML diagramos ir naudojama programavimo inžinerijos teorija ir terminija, skaitytojas turi turėti programavimo inžinerijos teorinių žinių.

1.2. Santrauka

Darbo tikslas.

Ištirti, kaip kuo daugiau supaprastinti ir apibrėžti programinės įrangos karkaso architektūros projektavimo darbus bei nustatyti dažniausiai pasikartojančius darbus kuriant naują programinę įrangą.

Tiriamasis objektas.

- Programinės įrangos architektūros supaprastinimas kuriant naują sistemą operacijų pagrindu.
- Kokybiškos programinės įrangos kūrimui pagrindo sudarymas.
- Trečiųjų šalių komponentų tinkamumo tyrimas karkasui, kurių funkcionalumas panaudojamas operacijų pagrindu.

Tiriamojo objekto analizės tvarka.

Darbe pirmiausia yra analizuojamas tiriamasis objektas ir literatūra, kuri yra susijusi su šiuo objektu. Nagrinėjama literatūra susijusi su objektiniu programų projektavimu, programų karkasų kūrimu ir kylančiomis problemomis kuriant programų karkasus, tarptautiniai programinės įrangos kūrimo standartai, trečiųjų šalių komponentų aprašų literatūra. Analizuojamos nusistovėjusios tendencijos programų karkasų kūrime ir suteikiama svarba karkasui, kuriant didelius projektus.

Taip pat yra aprašomi programinės įrangos, kuri buvo sukurta magistrantūros studijų metu, esminei aspektai, bei atliekamas šios sukurtos programinės kokybės, galimų ateities pakeitimų ir pritaikymo tyrimas. Toliau yra atliekamas sukurtos programinės įrangos panaudojimo ir našumo eksperimentinis tyrimas.

1.3. Dokumento struktūra

Skyrius 1. Įvadas. Šiame skyriuje pateikiama dokumento paskirtis ir įvadinės mintys.

Skyrius 2. Analitinė dalis. Programų karkasų kūrimo analizė. Šiame skyriuje pateikiama literatūros ir technologijų apžvalga susijusi su programų karkasų projektavimu. Išanalizuojamas programų karkaso architektūros modelio kūrimo pagrindimas.

Skyrius 3. Projektinė dalis. Biznio sluoksniu karkasas. Šiame skyriuje yra pateiktas magistrantūros studijų metu realizuotos sistemos aprašymas ir pateikiami pagrindiniai sukurtos programinės įrangos architektūros aspektai.

Skyrius 4. Tyrimo dalis. Biznio sluoksnio karkaso kokybės tyrimas. Šiame skyriuje atliekamas karkaso galimybių tyrimas. Tyrimas atliekamas analizuojant sukurtą projektą.

Skyrius 5. Eksperimentinė dalis. Biznio sluoksnio karkaso eksperimentinis tyrimas. Šiame skyriuje yra aprašomas studijų metu sukurto karkaso našumo ir panaudojimo eksperimentas.

2. PROGRAMŲ KARKASO KŪRIMO ANALIZĖ

Šiame skyriuje yra analizuojamas programų karkaso reikalingumo ir kūrimo pagrindas. Taip pat suformuluojami architektūrinio projektavimo uždaviniai ir problemos bei aprašomi šių problemų sprendimo būdai.

2.1. Nagrinėjama sritis.

Visų pirma kodėl yra realizuota ir nagrinėjama programų karkaso sritis?

Šiais laikais programavimui yra keliami aukšti reikalavimai, kadangi klientai reikalauja kokybiškos programinės įrangos ir prašo ją sukurti kuo greičiau, dėl to nukenčia programinės įrangos kokybė. Greitas kokybiškų sistemų kūrimas yra aktuali problema programavimo įmonėms ir šią problemą bandoma ištirti šiuo darbu. Geras programos karkasas leidžia programuotojams ir sistemos projektuotojams praleisti daugiau laiko koncentruojantis į kuriamos sistemos dalykinę sritį, o ne į programos architektūrą.

Kuriant naujas sistemas užsakovai dažniausiai nori, kad ji būtų sukurta kaip įmanoma greičiau. Sistemų kūrėjai skubėdami daugiau dėmesio skiria į galutinį sistemos funkcionalumą, o ne į programinį užbaigtumą ar kuriamos sistemos architektūrą. Dėl to stipriai nukenčia sistemos architektūra ir ateities sistemos patobulinimai gali būti sudėtingi ir daug kainuoti, taip pat tokios chaotiškos architektūros sistemos gali turėti savyje daugiau klaidų.

Kas tai yra programų karkasas ir kam jis yra reikalingas?

Kuriant šiuolaikines konkurencingas sistemas yra svarbu kurti jas kiek įmanoma ekonomiškiau ir efektyviau. Vienas iš būdų padidinti kūrimo efektyvumą yra sistemos kompleksiškumo sumažinimas. Kad sumažintume sistemų kompleksiškumą reikia įvesti programinės įrangos tam tikrą šabloną – karkasą.

Programų karkasas yra tam tikros kuriamos programinės įrangos architektūrinis šablonas, kurio pagrindu remiasi visa kuriama nauja sistema. Programų karkase turi būti:

- Sukurti baziniai programos komponentai ir sukurtas bendravimas tarp jų;
- Sukurta terpė naujo funkcionalumo suprogramavimui ir apibrėžtos taisyklės kaip šis funkcionalumas turi būti sukurtas;
- Sukurtas ir suprogramuotas ryšys su duomenų bazėmis;

- Sukurtos funkcijos, kurios dažnai pasitaiko programinės įrangos kūrime (pavyzdžiui: daugiakalbystė, duomenų eksportavimas ir kita).

Programų karkasai reikalingi siekiant:

- Sumažinti programavimo chaosą greitai kuriant naujas sistemas;
- Padidinti programų kūrimo greitį ir ekonomiškai naudoti resursus;
- Padidinti sistemos funkcionalumą ir galimybes pakartotinai panaudojant jau sukurtus komponentus;
- Padidinti kuriamos sistemos išbaigtumą;
- Suteikti programuotojui supaprastintą funkcijų panaudojimą;
- Apibrėžti programuotojui programavimo taisykles (aktualu kai programuotojų yra daug, nes skirtingiems programuotojams bus lengviau skaityti programos kodą);
- Padidinti sistemos lankstumą;
- Sumažinti programinės įrangos tobulinimo kaštus.

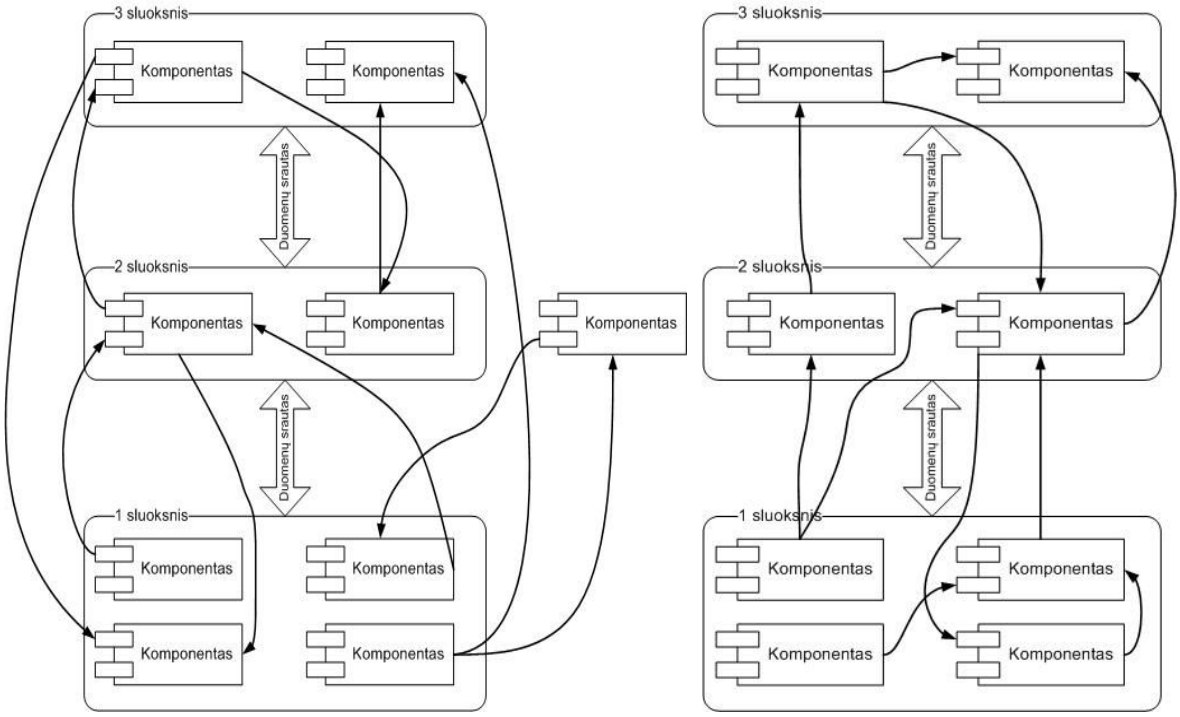
2.2. Architektūros modelis

2.2.1. Daugiasluoksnis architektūros modelis

Programų karkasą bendrą kuo platesniam spektrui sistemų sukurti yra gana sudėtinga, todėl yra pasiribojama tais veiksmais ir funkcijomis, kurias turi dauguma arba privalo turėti kiekviena šiuolaikinė sistema. Karkasas taip kuriamas, kad jis tiktų visoms dalykinėms sritims, turinčioms panašią loginę struktūrą. Ateityje turint konkrečios sistemos specifikaciją, karkasas yra konkretizuojamas, pritaikomas konkrečiam atvejui. Toks sistemų kūrimo būdas kartais dar yra vadinamas parametriniu programavimu [1] t.y. kai sistemoje yra įvedami tam tikri parametrai, kurie gali keisti sistemos funkcionalumą.

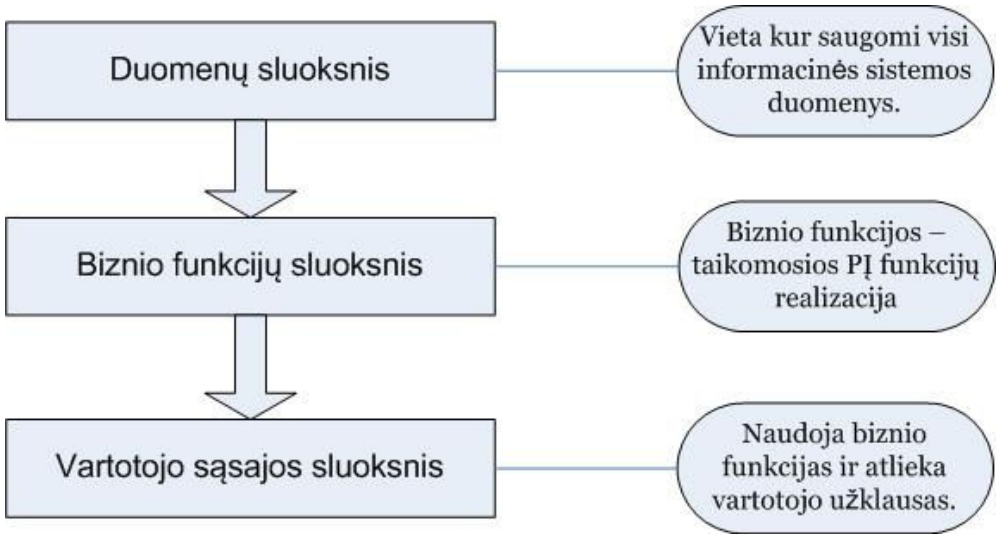
Yra daug būdų projektuoti programinę įrangą, tačiau kuriant dideles sistemas jas reikia skaidyti į mažesnius uždavinius, posistemes ar komponentus, taip padalinti tam tikrus darbus tarp programuotojų. Todėl dažniausiai programinė įranga kuriama naudojant daugiasluoksnę architektūrą [2]. Labai svarbu tiksliai apibrėžti šių sluoksnių bendravimo principus ir sąsajas. Bendravimas tarp sluoksnių yra griežtai ribojamas hierarchijos, kur aukščiausio lygio sluoksnis bendrauja tik su po juo esančiu sluoksniu arba virš jo esančiu

sluoksniu, šis su dar žemesniu ir t.t. Nėra leidžiamas bendravimas apeinant tarpinius sluoksnius. Toks sistemų organizavimas palengvina sistemų palaikomumą ir kokybę.



1 pav. Netaisyklingos(kairėje) ir taisyklingos(dešinėje) daugiasluoksnės architektūros pavyzdys.

Daugiasluoksnė architektūra (dažniausiai ji būna trijų lygių) apjungiama su Modelis-Vaizdas-Valdymas šablonu [3], kuris siūlo išskaidyti programinę įrangą į duomenų sluoksnį, biznio logikos sluoksnį ir vartotojo sąsajos sluoksnį. Pavaizduota optimali sluoksnių diagrama, kuri yra tinkama daugeliui biznio programinių įrangų:



2 pav. Karkaso daugiasluoksnės architektūros diagrama.

Trijų lygių architektūra paremta Modelis-Vaizdas-Valdymas šablonu yra dažniausiai naudojama šiuolaikinėse sistemos, nes tiek vartotojo sąsajos sluoksnis, tiek duomenų sluoksnis praktiškai yra kiekvienoje sistemoje. Tačiau kartais biznio funkcijų sluoksnis yra išskaitomas į kelis kitus bendresnius sluoksnius.

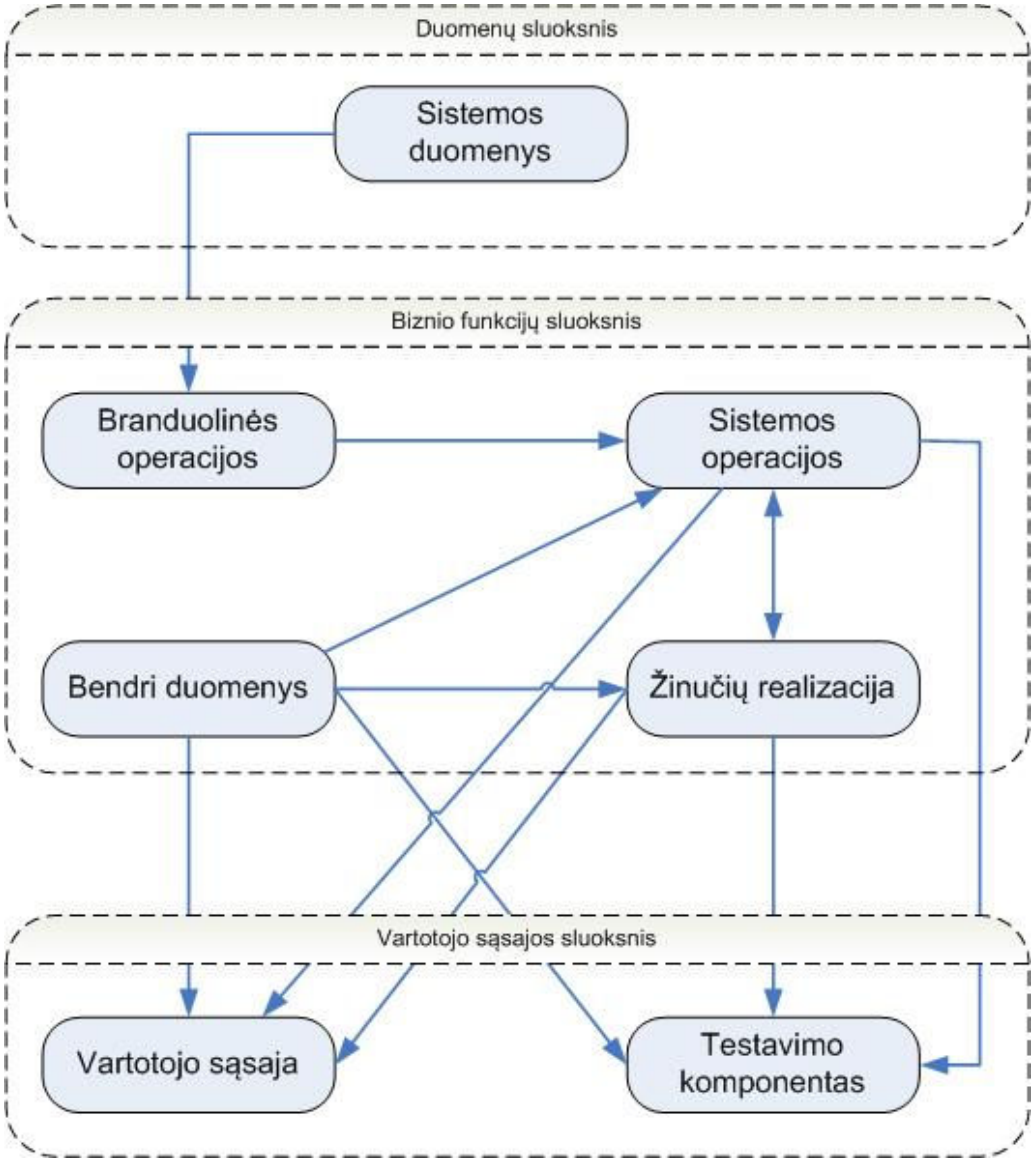
Tokia programinės įrangos trijų sluoksnių architektūra leidžia nesunkiai realizuoti tvarkingą, praplečiamą ir palaikomą programinę įrangą bei lengvai paskirstyti darbus programuotojams.

2.2.2. Siūlomas architektūros modelis

Nustatyta, kad dažniausiai yra naudojama daugiasluoksnė architektūra, todėl ji pasirinkta karkaso realizacijai. Ši architektūra yra pasirinkta dėl:

- Jos sluoksnių apirėžtumo ir bendravimo tarp sluoksnių nustatymo;
- Lengvo programinės įrangos palaikomumo ir jos lankstumo;
- Lengvo pritaikymo kliento/serverio sistemai;
- Sistemos pritaikymo pakartotiniam panaudojimui iš naujo.

Naudojantis daugiasluoksnės architektūros modeliu ir Modelis-Vaizdas-Valdymas šablonu buvo pasirinktas toks karkaso architektūros modelis:



3 pav. Karkaso architektūros modelis.

1 lentelė. Kiekvieno karkaso sluoksnio ir jo dalių aprašymai.

Duomenų sluoksnis	
Sistemos duomenys	Apibrėžiami visi karkaso duomenys. Jie gali būti įvairių tipų: duomenų bazės, tekstiniai, konfigūraciniai, XML ir duomenų failai.
Biznio funkcijų sluoksnis	
Branduolinės operacijos	Saugomos pamatinės karkaso operacijos, kurios yra nekintamos ir sudaro visą karkaso branduolį (Pavyzdžiui duomenų saugojimui ir ištraukimui iš duomenų bazės, veiksmų registravimui, bendravimui tarp komponentų ir kita).
Sistemos operacijos	Realizuojamos operacijos, kurios nėra pamatinės karkaso operacijos, o yra skirtos konkrečiai sistemai. Tai gali būti konkrečių duomenų ištraukimui iš duomenų bazės, daugiakalbystė, duomenų eksportas ir visos kitos operacijos, kurios vykdo sistemos dalykinės srities funkcijas. Visos šios operacijos gali būti panaudotos iš naujo. Šios operacijos yra kuriamos pagal tam tikrą šabloną, kurį apibrėžia branduolinės operacijos.
Bendri duomenys	Saugomi konkrečios sistemos duomenų ir konfigūracijų objektai, kuriais operuoja visa sistema.
Žinučių realizacija	Saugomi konkrečios sistemos žinučių realizacijos objektai, kuriais operuoja visos sistemoje esančios operacijos.
Vartotojo sąsajos sluoksnis	
Vartotojo sąsaja	Tai yra dalis, kuri naudojasi operacijomis ir suteikia sistemos vartotojui tam tikras funkcijas.
Testavimo komponentas	Tai yra dalis, kurioje aprašyti visi operacijų vienetų testiniai atvejai, sukuriant naują operacija jai taip pat sukuriamas testas. Taip galima bet kada šiuos testus paleisti ir patikrinti sistemos korektiškumą.

Kaip matome iš siūlomo karkaso architektūros modelio ir jo aprašymo, yra naudojama šiek tiek modifikuota daugiasluoksnė architektūra. Modifikacijos esmė – konkretus architektūrinis sluoksnis nesinaudoja aukštesnio architektūrinio sluoksnio teikiamomis funkcijomis, o naudojasi tik žemesnio lygio architektūrinio sluoksnio

funkcionalumu. Šios hierarchinės architektūros pasirinkimo tikslas – didesnė tikimybė, kad kuriamoje sistemoje bus mažiau klaidų, nes architektūra yra labiau konkretizuota.

2.2.3. Projektavimo šablonų analizė

Programinės įrangos kūrime projektavimo šablonai yra dažniausiai apibrėžiami kaip projektavimo problemų sprendimai, kurie gali būti pakartotinai panaudoti daugelyje panašių atvejų. Kitaip sakant, projektavimo šablonai yra būdas leidžiantis pakartotinai panaudoti abstrakčias žinias apie problemą ir jos sprendinį. Taigi paplitus objektinėms programavimo kalboms populiariausi yra objektiškai orientuoti projektavimo pavyzdžiai.

Programinės įrangos be projektavimo šablonų architektūra yra skurdi, todėl gali stipriai išaugti programinės įrangos klaidų kiekis, programinės įrangos pakeitimo kaštai, taip pat programinė įranga bus nelanksti pakeitimams. Todėl projektavimo šablonai yra labai svarbu karkaso architektūrai.

Knygoje „*Design Patterns: Elements of Reusable Object-Oriented Software*“ [4] aprašyti dauguma pagrindinių projektavimo šablonų modelių, kurie yra plačiai naudojami. Projektavimo šablonai yra skirstomi į tris tipus:

- Kūrimo projektavimo šablonai.
- Struktūriniai projektavimo šablonai.
- Elgsenos projektavimo šablonai.

Analizei pasirinkti projektavimo šablonai, kurie, mano manymu, yra aktualiausi bendro naujai kuriamoms programinėms įrangoms karkaso kūrimui.

Vienturtis (angl.: Singleton) projektavimo šablonas [4].

Projektavimo šablonas „Vienturtis“ yra kūrimo tipo. Šis šablonas užtikrina, kad klasė turėtų tik vieną bendrą objektą ir suteikia globalų priėmimą prie jo.

4 pav. Projektavimo šablonas „Vienturtis“.

Klasės dalyvaujančios vienturčio šablone:

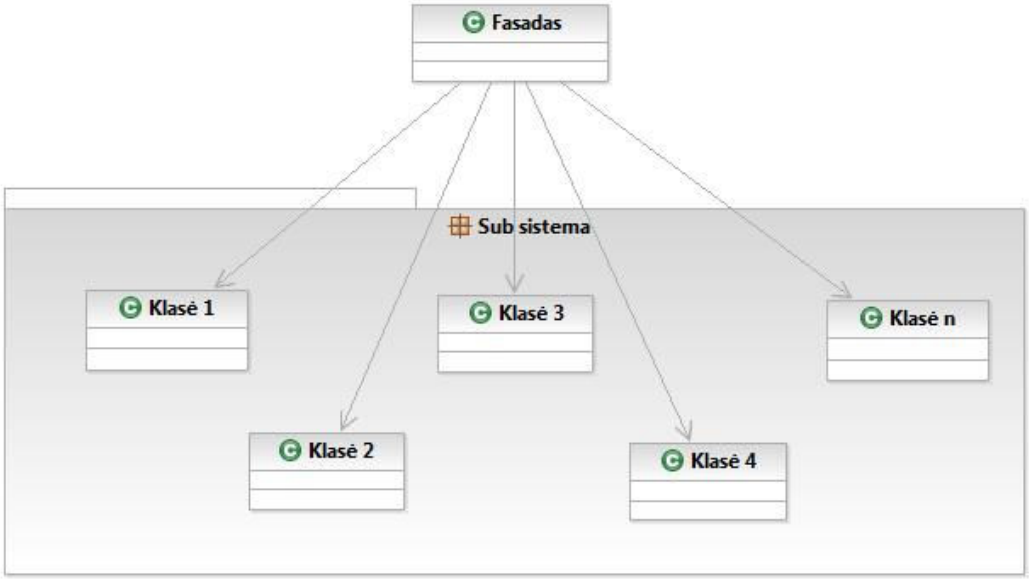
Klasė „Vienturtis“ – saugo tam tikrą unikalią objekto operaciją, kurią yra leidžia pasiekti jos užklausėjui. Taip pat klasė atsako už savo unikalaus objekto sukūrimą ir išsaugojimą.

Programinėse įrangose dauguma egzistuojančių objektų atsako už savo darbą ir dirba su duomenimis, esančiais tam objektui skirtoje srityje. Tačiau yra tokių objektų, kurie papildomai yra atsakingi už globalias operacijas, tokias kaip ribotų resursų valdymas, unikali prisijungimo informacija ir pan. Iš tokių klasių yra reikalaujama, kad būtų tik vienas ir unikalus šios klasės objektas. Dažniausiai problemų kyla tada, kai objektas kuriamas vis iš naujo ir jis išlieka nepriklausomas. Taigi naudojant šį šabloną yra pašalinamas poreikis vis iš naujo kurti ir naikinti objektus. Šio šablono paskirtis yra užtikrinti vieną objektą vienai klasei ir sudaryti galimybę globaliai pasiekti šį objektą t.y. centralizuojamas viename objekte esančio tam tikro resurso valdymas. Vienturčio projektavimo šablonas yra būdas globaliai naudojamų kintamųjų laikymui ir panaudojimui, nes paprastų globalių kintamųjų panaudojimas yra prasto programavimo požymis.

Pavyzdžiui, šiuo principu gali būti suprogramuotas prisijungimas prie duomenų bazės. Tai yra vykdant užklausas naudojamas vienas ir tas pats objektas ir nereikia prisijungti prie duomenų bazės iš naujo kiekvieną kartą vykdant vis kitas užklausas.

Fasadas (angl.: Facade) projektavimo šablonas [4].

Projektavimo šablonas „Fasadas“ yra struktūrinio tipo. Šis šablonas sudaro vieningą sąsają tarp antrinės sistemos sąsajos ir aukštesnio lygio sąsajos. Dėl to galima paprastai ir struktūrizuotai naudoti antrinę sistemą.



5 pav. Projektavimo šablonas „Fasadas“.

Klasės dalyvaujančios fasado šablone:

Klasė „Fasadas“ – tam tikrą užduotą vykdyti operaciją priskiria atitinkamiems antrinės sistemos objektams ir žino, kurios antrinės sistemos klasės atsako už šią operaciją.

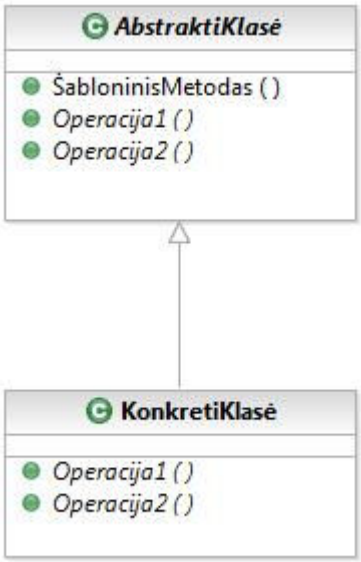
Antrinės sistemos klasės – saugo tam tikrą antrinės sistemos funkcionalumą (operacijas) ir vykdo pagrindinės klasės „Fasadas“ nurodytas operacijas. Šios klasės neturi nuorodos į „Fasado“ klasę.

„Fasadas“ klasė yra kaip sąsaja tarp tam tikrų operacijų ir savybių leidžiančių struktūrizuotai naudoti antrinės sistemos suteikiamą funkcionalumą. Fasado projektavimo šablonas yra paprastai suprantamas ir panaudojamas, taip pat jis yra labai dažnai naudojamas projektuojant daugiasluoksnės architektūros modelio programines įrangas [5]. Fasado projektavimo šablono tikslas yra sudaryti aukšto lygio programos architektūrą, leidžiančią lengvai naudoti antrinės sistemos funkcionalumą ir jos suteikiamas operacijas. Fasado projektavimo šablonas taip pat padeda paslėpti sudėtingą operacijų realizaciją ir suteikia programuotojui paprastesnį tam tikrų operacijų panaudojimo būdą per „Fasado“ klasę. „Fasado“ klasė parodo programuotojui kokias operacijas (ir kaip) jis gali naudoti.

Fasado projektavimo šablono principas labiausiai tinka biznio sluoksnio karkaso realizacijai, kuris yra paremtas operacijų principu.

Šabloninis metodas (angl.: Template method) projektavimo šablonas [4].

Projektavimo šablonas „Šabloninis metodas“ yra elgsenos tipo. Šis šablonas apibrėžia programinės įrangos tam tikro funkcionalumo griaučius, bet ne patį programinės įrangos funkcionalumą. Dėl to galima kurti aukštesnį programinės įrangos lygmenį, kai dar nėra sukurtas žemesnio lygmens funkcionalumas.



6 pav. Projektavimo šablonas „Šabloninis metodas“.

Klasės dalyvaujančios šabloninio metodo projektavimo šablone:

Klasė „AbstraktiKlasė“ – apibūdina abstrakčias primityvias operacijas, kurias „KonkretiKlasė“ klasė privalo įgyvendinti. Ši klasė saugo tam tikro algoritmo sudėtinių dalių griaučius.

Klasė „KonkretiKlasė“ – saugo tam tikrų operacijų ir algoritmų dalių įgyvendinimo funkcionalumą.

Šabloniniu metodu pagrįstas programų projektavimas suteikia programinės įrangos tam tikrų jos dalių pakartotinio panaudojimo galimybę, kas yra aktualu kuriant programų karkasus. Smulkūs darbai yra išskaitomi į atskiras klases, o sujungus tam tikras klases į visumą gaunamas kažkoks konkretus funkcionalumas.

Šabloninis metodas yra naudojamas:

- Konkretaus funkcionalumo įgyvendinimui dalimis;
- Dėl programinio kodo dubliavimo vengimo;

- Dėl veiksmų kontrolės konkretaus funkcionalumo klasėms.

Šio projektavimo šablono principas tinka biznio sluoksnio karkasui įgyvendinti dėl suteikiamo pakartotinio panaudojimo galimybės.

Projektavimo šablonų apibendrinimas

Išnagrinėti projektavimo šablonai, kurie yra orientuoti į trijų sluoksnių architektūrą ir pakartotinį panaudojimą, kas yra labai svarbu bendro, naujai kuriamoms programinėms įrangoms, biznio sluoksnio karkasui.

Projektavimo šablonų pasirinkimo pagrindimas:

- Vienturčio projektavimo šablono pasirinktas dėl globalių kintamųjų panaudojimo galimybės ir unikalaus objekto naudojimo galimybės;
- Fasado projektavimo šablonas pasirinktas dėl daugiasluoksnės architektūros palaikymo galimybės;
- Šabloninio metodo projektavimo šablonas pasirinktas dėl pakartotinio tam tikrų sistemos dalių panaudojimo galimybės.

2.3. Standartų analizė

Siekiant sukurti kokybišką produktą reikia išanalizuoti ne tik architektūrinius modelius ar programavimo platformas, bet ir tarptautinius kokybiškos programinės įrangos kūrimo standartus. Analizuojami du aktualiausi karkaso kūrimui standartai:

- ANSI/IEEE 1471-2000 standartas rekomenduojamos architektūros aprašymo praktikos programine įranga pagristomis sistemomis.
- ISO/IEC 9126 programinės įrangos kokybės vertinimo standartas.

2.3.1. ANSI/IEEE 1471-2000 standartas

ANSI/IEEE 1471-2000 standartas yra rekomendacijos architektūros aprašymui programine įranga pagristomis sistemomis [6]. Šis standartas yra vystomas IEEE darbo grupės remiantis kitais tarptautiniais standartais, industrine ir akademine patirtimi. ANSI/IEEE 1471-2000 standarte yra akcentuojamas konceptualaus programų karkaso architektūrinis apibūdinimas ir būdai kaip šią koncepciją galima nustatyti.

Pasak standarto, architektūra yra fundamentalus sistemos organizavimas, kuris nusakomas jos komponentais, jų ryšiu vienas su kitu ir su aplinka bei principais, kuriais

pagristas projektas ir jo evoliucija. Standarto ANSI/IEEE 1471-2000 aprašymai turėtų būti naudojami:

- Programinės įrangos išraiškai ir jos raidai;
- Bendravimo tarp programinės įrangos suinteresuotų subjektų nustatymui;
- Nuosekliam architektūros vertinimui ir palyginimui;
- Sistemos planavimui, valdymui ir kūrimui organizuoti;
- Architektūros principų suformavimui, kuriais būtų galima keisti ir kurti programinę įrangą;
- Programinės įrangos įgyvendinimo tikrinimui ar laikomasi pradinių architektūrinių idėjų.

Standarto ANSI/IEEE 1471-2000 panaudojimo esminiai aspektai.

- Programinės įrangos aplinka arba jos kontekstas gali įtakoti pačią sistemą. Aplinką gali sudaryti kitos sistemos, kurios gali sąveikauti su kuriama sistema per interfeisus arba tiesioginiai. Taigi turi būti nustatytos ribos kaip kitos sistemos gali sąveikauti su kuriama sistema.
- Programinė įranga visada turi vieną arba daugiau suinteresuotų šalių ir kiekviena suinteresuota šalis dažniausiai turi skirtingus kuriamos sistemos tikslus.
- Programinė įranga yra kuriama, kad įvykdytų vieną ar kelias tam tikras operacijas. Operaciją gali sudaryti keletas smulkesnių veiksmų. Operacijos yra skirtos suinteresuotų šalių tikslams įvykdyti.
- Kiekviena programinė įranga turi architektūrą ir ji turėtų būti kuriama pagal architektūros aprašą.
- Architektūros aprašas yra suskirstomas į vieną arba daugiau sudedamųjų dalių. Kiekviena dalis yra skirta suinteresuotų šalių vienam arba keliems tikslams išpildyti ir yra dalinis architektūros atvaizdas.
- Architektūros atvaizdai sąlygoja naudojamą programavimo kalbą, projektavimo ir architektūros modelius, modeliavimo metodus, analizės metodus ir kita.

Pagal architektūros atvaizdus yra sudaromas rinkinys modelių aprašančių sistemą tam, kad išpildytų suinteresuotų šalių tikslus. Taigi pagal šiuos modelius galima sukurti programinės įrangos architektūrą paremtą, pavyzdžiui, sluoksniniu vaizdavimo būdu.

2.3.2. ISO/IEC 9126 standartas

ISO/IEC 9126 yra programinės įrangos kokybės vertinimo standartas [7]. Pagrindinis šio standarto tikslas yra apibrėžti daromus nukrypimus kuriant programinę įrangą, kurie gali nepalankiai įtakoti programinės įrangos kūrimo kokybę. Taip pat išsiaiškinti projekto prioritetus ir vėliau šiuos abstrakčius prioritetus paverti į išmatuojamas vertes. ISO/IEC 9126 standarte yra bandoma išaiškinti bendrą projekto objektų ir tikslų supratimą.

Šis standartas yra sudarytas iš 4 dalių:

- Kokybės modelio (ISO/IEC 9126-1)
- Išorinės metrikos (ISO/IEC 9126-2)
- Vidinės metrikos (ISO/IEC 9126-3)
- Kokybės metrikos naudojime (ISO/IEC 9126-4)

Kokybės modelis (ISO/IEC 9126-1)

Šioje dalyje yra apibrėžtas struktūrizuotas programinės įrangos kokybės charakteristikų ir subcharakteristikų rinkinys. Šis standartas turi šešis pagrindines charakteristikas, kurių aprašymai yra pateikti 2 lentelėje.

2 lentelė. Kokybės modelio charakteristikų paaiškinimai.

Funkcionalumas	
Tinkamumas	Operacijų atliekančių užduotis pilnumas ir atitikimas reikalavimams.
Tikslumas	Veikimas grąžinant teisingus arba norimus rezultatus.
Bendradarbiavimas	Sąveika su kitomis sistemos.
Funkcinis atitikimas	Atitikimas programinės įrangos kūrimo standartams, konvencijoms, įstatymams ir pan.
Saugumas	Galimybė apsaugoti nuo neteisėto panaudojimo ir priėjimo.
Patikimumas	
Brandumas	Nusako nesėkmingo veikimo ir gedimų dažnį programinėje įrangoje.
Klaidų tolerancija	Programinės įrangos galimybė išlaikyti nustatytą funkcionavimo lygį atsiradus gedimams.
Atkuriamumas	Programinės įrangos gebėjimas atstatyti funkcionavimo lygį ir prarastus duomenis, nesėkmingos operacijos atlikimo atveju, nustatytose laiko ir kaštų ribose.

Panaudojamumas	
Suprantamumas	Vartotojo pastangos reikalingos programos loginio konteksto supratimui.
Išmokstamumas	Vartotojo pastangos reikalingos išmokti dirbti su programa.
Veiksnumas	Vartotojo pastangos reikalingos programos operacijų atlikimui ir jų valdymui.
Patrauklumas	Vartotojo noras dirbti su programine įranga.
Efektyvumas	
Laikinė elgsena	Programinės įrangos atsako ir veikimo laikas.
Išteklių panaudojimas	Programinės įrangos naudojamų resursų apimtis ir panaudojimo trukmė.
Palaikomumas	
Analizuojamumas	Reikalingas pastangų kiekis siekiant nustatyti trūkumus, klaidų priežastis, modifikuotinos programinės įrangos dalies radimui.
Keičiamumas	Reikalingas pastangų kiekis programos modifikacijoms, klaidų pašalinimui arba funkcionavimo aplinkos pakeitimui.
Stabilumas	Rizikos dydis susijęs su nenuspėjamu funkcionavimu po programinės įrangos modifikacijų.
Testuojamumas	Reikalingas pastangų kiekis siekiant ištestuoti programinę įrangą.
Perkeliamumas	
Pritaikomumas	Programinės įrangos pritaikymo galimybės skirtingoms aplinkoms
Įdiegiamumas	Pastangų kiekis reikalingas įdiegti programinę įrangą nustatytoje aplinkoje.
Atitikimas	Programinės įrangos atitikimas perkeliamumo standartams ir konvencijoms.
Pakeičiamumas	Galimybė panaudoti karkasą vietoje kitos sukurtos programinės įrangos jos aplinkoje.

Išorinės metrikos (ISO/IEC 9126-2)

Ši dalis aprašo išorines metrikas skirtas matuoti programinės įrangos kokybės charakteristikas. Išorinės metrikos tai yra kiekybinis matavimas ir matavimo metodai, kurie gali būti naudojami programinės įrangos atributų arba tam tikrų charakteristikų matavimui, žiūrint į sistemą iš išorės. Išorinės metrikos yra naudojamos testuojant jau sukurtą arba paskutinėje kūrimo stadijoje esančią programinę įrangą.

Vidinės metrikos (ISO/IEC 9126-3)

Ši dalis aprašo vidines metrikas skirtas matuoti programinės įrangos kokybės charakteristikas. Vidinės metrikos tai yra kiekybinis matavimas ir matavimo metodai, kurie gali būti naudojami programinės įrangos atributų arba tam tikrų charakteristikų matavimui, žiūrint į sistemą tiesiogiai arba netiesiogiai iš vidaus t.y kokybiniai matavimai yra gaunami ne iš sistemos elgsenos. Vidinės metrikos yra naudojamos programinės įrangos kūrimo ir projektavimo procese.

Kokybės metrikos naudojime (ISO/IEC 9126-4)

Ši dalis aprašo kokybės matavimo metrikas efektyvumui, produktyvukui, saugumui ir pasitenkinimo programine įranga apskaičiavimui. Šias metrikas naudoti galima tik tada, kai yra sukurtas galutinis produktas.

2.3.3. Standartų analizės išvados

Standartų analizei buvo pasirinkti analizuoti didžiausią įtaką karkaso architektūros kokybei turintis standartai.

Naudojantis ANSI/IEEE 1471-2000 standartu yra atsižvelgiama projektuojant karkaso architektūrą. Šis standartas nusako pagrindinius kriterijus kaip yra sudaromi programų architektūros modeliai. Todėl buvo pasirinkta operacijų realizacija ir sluoksninė architektūra, nes ateityje galima prijungti papildomus architektūrinius sluoksnius naudojant tokį architektūros būdą.

Analizuojant ISO/IEC 9126 standartą didžiausias dėmesys yra atkreipiamas į kokybės modelį, kur visos šešios charakteristikos yra svarbios karkaso kokybei. Sudarant karkaso architektūrą ypatingai yra atsižvelgta į „Palaikomumas“ ir „Perkeliamumas“ charakteristikas, kadangi karkasas nėra skirtas konkrečiai sistemai, todėl šių kokybės charakteristikų įvertinimas turi būti aukščiausias.

2.4. Programavimo technologijų analizė

2.4.1. Microsoft .NET platforma ir C# programavimo kalba

Microsoft .NET Framework platforma.

Microsoft .NET Framework nėra programavimo kalba, MS .NET Framework yra viena pažangiausių Microsoft technologijų skirtų šiuolaikinės programinės įrangos, servisų bei komunikacijos priemonių kūrimui [8]. Sunku yra lyginti kai nėra su kuo lyginti, kadangi visos kitos sistemų kūrimo priemonės (C++, JAVA, PHP ir kita) yra tik programavimo kalbos. MS .NET nuo pat pradžių buvo vystoma kaip pažangi programavimo technologija, orientuota į šiuolaikinio verslo poreikius ir tų poreikių greitą augimą. MS .NET technologija yra sukurta kaip labai lanksti strategija, apimanti labai įvairius ir plačius poreikius, kurie gali keistis, augti ir vystytis. Taip pat MS .NET yra nepriklausoma nuo programavimo kalbos, todėl ja paremtus sprendimus, atsižvelgiant į jų specifiką, galima kurti naudojant įvairias programavimo kalbas: ASP.NET, C#, C++, VB, Delphi, netgi JAVA (J#). MS .NET užtikrina sprendimų, sukurtų skirtingomis programinėmis priemonėmis .NET terpėje, suderinamumą tarpusavyje ir tai padeda sutaupyti laiko ir lėšų atnaujinant informacines sistemas bei suderinant jas su senesnėmis informacinėmis sistemomis.

Pagrindiniai Microsoft .NET technologijų privalumai [9]:

- Pilna, greita ir patogi integracija su visais Microsoft produktais, taip pat geras suderinamumas tarp Microsoft produktų.
- .NET yra daug saugiau. Šios technologijos saugumo politika paremti sprendimai yra gerokai saugesni vien dėl to, kad jie naudoja bendrą integruotą saugumo užtikrinimą.
- .NET technologija yra ir bus vystoma bei palaikoma Microsoft kompanijos, todėl šia technologija remiantis bus galima kurti naujus sprendimus integruojant juos į veikiančias, seniau sukurtas sistemas nekuriant jų iš naujo.
- Naujų sistemų kūrimo paprastumas. To paties funkcionalumo sprendimui realizuoti MS .NET technologijoje (ASP.NET, C#) užtenka tik 30% kodo lyginant su PHP ar JAVA. Žymiai mažiau kodo – atitinkamai mažesnė tikimybė įsivelti programinėms klaidoms, o tai tiesiogiai susiję su programinio sprendimo veikimo patikimumu.

- Svarbiausias privalumas, kad Microsoft .NET technologijose realizuota daug sprendimų: susijusių su internetinėmis technologijomis (ASP.NET), saugumu (assemblies, CAS – Code Access Security), CLR – Common Language Runtime, turi įvairius testus), metaduomenys (.NET metadata), bendrą .NET klasių biblioteką, kurioje realizuota daug pagrindinių funkcijų (FCL - Framework Class Library), tiesioginis darbas su duomenų bazės objektais (ADO.NET), komponentinis programavimas (COM/DCOM) ir kita.

C# programavimo kalba.

Microsoft kompanijos sukurta, funkcinė, tipizuota, objektinė, palaikanti bendrumą (angl. *generic*) programavimo kalba [10]. C# sintaksė yra panaši į C++ programavimo kalbos. Kadangi poreikiai tvirtai, ilgaamžiškai programinei įrangai augo, tai tikslas buvo sukurti kuo patogesnę programavimo kalbą, padidinančią programuotojo produktyvumą. Kuriant C# programavimo kalbą buvo remtasi jau esamom, populiariom programavimo kalbom, tokiom kaip C++, JAVA. Tačiau C# turi ne tik daug panašumų, bet ir daug skirtumų nuo savo gimininguų programavimo kalbų. Microsoft vis atnaujina šią kalbą, dabartinė versija yra C# 3.5, kurioje yra daug naujovių lyginant su ankstesnėmis versijomis. Yra sukurta patogi programų kūrimo aplinka skirta būtent C#, taip pat šiai aplinkai sukurta daug pagalbinių priemonių (pvz.: ReSharper).

Žiūrint iš programų kūrimo pusės programos tvirtumą ir ilgaamžiškumą apibrėžia tokios programavimo kalbos savybės[10]:

- stipriai tipizuota;
- išeities teksto portatyvumas;
- automatinis nereikalingų kintamųjų aptikimas ir sunaikinimas;
- masyvo ribų tikrinimo palaikymas;
- aptikti bandomus naudoti sunaikintus kintamuosius.

C# leidžia panaudoti jau sukurtus įvairius programų komponentus, juos sujungti į veikiančią sistemą. C# tinkama kurti paskirstytas sistemas, servisus bei labai paprastiems vartotojo poreikiams tenkinti, skirtas sistemas. Taigi, kuo daugiau galimybių programavimo kalba turi, tuo lengviau sistema yra sukuriama.

2.4.2. JAVA platforma

Java programavimo platformą J2EE (Java 2 Platform, Enterprise Edition) sukūrė Sun Microsystems [11]. J2EE yra standartinė daugialyčių programų kūrimo Java programavimo kalba platforma, paremta moduliniais komponentais, vykdomais serveryje. Ši platforma sukurta tam, kad supaprastintų komplikuotus kūrimo, diegimo ir priežiūros procesus daugialypiuose sprendimuose. Ši architektūra yra pagrįsta tik Java programavimo kalba. Šios platformos privalumai:

- Java programavimo kalba suprogramuota sistema gali būti diegiama į bekurią operacinę sistemą.
- J2EE turi daug nemokamų bei keletą mokamų programavimo aplinkų Eclipse, Netbeans, JBuilder, IntelliJ ir t.t.
- J2EE platformai yra sukurta daug nemokamų trečiųjų šalių komponentų, kurie suteikia papildomą funkcionalumą.

Kuriant programinę įrangą Java pagrindu parašytas programinis kodas yra kompiliuojamas į baitkodą, kurį palaiko daugelis operacinės sistemos platformų ir jis yra tarpinis tarp programos kodo ir mašininio kodo. Tada šis kodas yra paruoštas paleisti su Java virtualia mašina (JRE – Java Runtime Environment).

Viena iš pačių svarbiausių sričių šiuolaikinėje programų inžinerijoje yra duomenų bazių valdymas, kuris yra būtinas daugumai šiuolaikinių informacinių sistemų. Java turi JDBC (Java DataBase Connectivity) biblioteką, kuri yra skirta reliacinių duomenų bazių valdymui pagrįstam SQL kalba ir nepriklausomam nuo duomenų bazių valdymo sistemos. JDBC apibrėžia sąsajas, kurių realizacijos turi būti pateikiamos duomenų bazių sistemos JDBC tvarkyklėse. Taip pat turi daugelį kitų technologijų.

2.4.3. Kitos programavimo platformos

C++ programavimo kalba.

C++ programavimo kalba (C kalbos nauja modifikacija) pasižymi labai dideliu lakoniškumu, sintaksinių struktūrų lankstumu ir universalumu. Taip pat ji yra bendrosios paskirties programavimo kalba. Šia programavimo kalba sukurtos sistemos nėra pririštos prie konkrečios operacinės sistemos. C++ programavimo didžiausias privalumas yra toks, kad ji yra vidutinio lygmens programavimo kalba, kuri yra sudaryta iš aukšto lygio ir žemo lygio programavimo galimybių.

Kitos programavimo kalbos.

Kitos programavimo kalbos, tokios kaip C, Python, Ruby, Gupta, PHP (skirta tik internetiniams projektams) net nebuvo analizuojamos, nes dauguma jų yra pasenusios ir beveik nenaudojamos naujuose projektuose, o jei naudojamos tai tik liktinėse sistemose. PHP nebuvo analizuojama, nes ji tinkama tik internetiniams projektams.

2.4.4. Programavimo platformos pasirinkimo priežastys

Taigi buvo išanalizuotos visos šiuolaikinės programavimo technologijos ir buvo pasirinkta Microsoft .NET Framework ir C# programavimo kalbą. Pagrindinės pasirinkimo priežastys:

- Planuojant kurti informacines sistemas, kurios kuriamos naudojimuisi ne savaitei, ne mėnesiui ir ne vieneriems metams, kol kas nėra pasirinkimo alternatyvos Microsoft .NET technologijai, technologijai kurios užnugaryje ilgalaikės Microsoft milijardinės investicijos ir užtikrinta ateitis.
- MS .NET Framework platforma yra labiausiai išbaigta ir jos pritaikymo galimybės yra didžiausios Windows operacinėms sistemoms (šias operacines sistemas naudoja didžioji dauguma kompiuterių vartotojų ~90% [12]).
- MS .NET Framework platformai yra sukurta daugybė trečiųjų šalių komponentų ir jie yra nuolat palaikomi, kas užtikrina jų kokybę.
- Dauguma biznio sluoksnio programinės įrangos pasaulyje yra kuriama būtent naudojant MS .NET Framework technologijas.

2.4.5. Trečiųjų šalių komponentai

nHibernate

nHibernate yra objektiškas duomenų atvaizdavimas (angliškai: Object-relational mapping (ORM)) skirtas Microsoft .NET technologijoms (Java atitikmuo Hibernate). Pagrindinė šios technologijos paskirtis yra objektiškai orientuotas duomenų atvaizdavimas kuriamoje sistemoje tradicinės reliacinės duomenų bazės [13].

nHibernate – tai technologija, kuri skirta SQL užklausoms generuoti. Ši technologija leidžia sistemai neprisirišti prie kokios nors konkrečios duomenų bazės SQL užklausų kalbos, tai yra nauda sistemos vartotojui – suteikiant jam pasirinkimo laisvę. Tačiau tai kartu yra ir programuotojui nauda, kadangi jam užtenka mokėti tik nHibernate technologiją ir jis galės sukurti sistemas įvairioms duomenų bazių valdymo sistemos. Taip pat naudojant šią

technologiją yra lengvas sistemų plečiamumas, nes naudojant SQL užklausas atsiradus naujam atributui viską reiktų keisti, naudojant šią technologiją to daryti nereikia. Taip pat nHibernate palaiko daug duomenų bazių valdymo sistemų.

Linq

Linq yra objektiškas duomenų atvaizdavimo komponentas, kuris yra Microsoft .NET technologijų priedas [14]. Tai nHibernate atitikmuo tik yra sukurtas Microsoft kompanijos. Linq yra nesenai sukurtas produktas ir yra skirtas tik MS .NET technologijų palaikomoms programavimo kalboms. Linq yra patogus tuo, kad jis gali dirbti su XML ir SQL duomenimis. „LINQ to XML“ užklausos operatoriai yra efektyviai ir lengvai panaudojami, taip pat yra teikiamos XPath/XQuery funkcijos naudojamoje MS .NET programavimo kalboje. „LINQ to SQL“ užklausos operatoriai yra grindžiami reliacine SQL kalba, kurie tiesiog yra rašomi vienos iš MS .NET programos kode. Tai yra naujas ir patogus įrankis, tačiau nHibernate turi daugiau privalumų.

Log4net

Log4net komponentas yra skirtas MS .NET Framework technologijoms ir jis palaiko visas programavimo kalbas, kurias palaiko ir .NET Framework. Log4net yra atviro kodo laisvai platinamas komponentas. Log4net suteikiami privalumai programinės įrangos kūrėjui [15]:

- Šis komponentas tinka tiek paprastoms vieno vartotojo informacinės sistemoms, taip pat ir sudėtingoms daug gijų turinčioms sistemos;
- Informaciją registruoti galima įvairiomis formomis: failuose, duomenų bazėje;
- Nesudėtingas konfigūravimas užsaugomų ir sugeneruotų žinučių;
- Realizuotos skirtingo tipo žinutės: informacinės, perspėjimo, klaidos, kritinės klaidos. Kiekvienai žinutei gali būti realizuoti atitinkami veiksmai;
- Yra galimybė registruojamus veiksmus siųsti elektroniniu paštu. Tai yra naudinga informacinėse sistemose, kurios veiklos sutrikimas gali padaryti didelius nuostolius. Pavyzdžiui atsiradus kritinei klaidai informacinėje sistemoje yra siunčiama žinutė kūrėjui su atliktais veiksmais sukėlusiais šią klaidą. Taip informacinės sistemos kūrėjas gali greičiau ir operatyviau reaguoti.

Šių laikų informacinės sistemos be veiksmų registravimo egzistuoti nebegali. Veiksmų registravimas padeda informacinės sistemos kūrėjui greičiau aptikti klaidas ir netikslumus. Tai yra svarbu didelėse sistemose, kadangi sukurti didelę sistemą be jokių klaidų yra beveik neįmanoma ir ne visada informacinės sistemos vartotojas sugebės atsiminti ir atkartoti veiksmus, kurie sukėlė tam tikrai klaidą ar netikslumą.

nServiceBus

nServiceBus atviro kodo komponentas yra skirtas MS .NET Framework technologijoms. nServiceBus yra skirtas bendravimui tarp sistemos komponentų žinučių principu. Pagrindinis šio komponento tikslas yra palengvinti bendravimą paskirstytose sistemose tarp atskirų šios sistemos komponentų. Šis komponentas suteikia galimybę sistemos komponentams bendrauti tarpusavyje žinučių pagalba MSMQ [16]. Tai yra naudinga tada, kai tarkim vienas sistemos komponentas dėl tam tikrų priežasčių neveikia ir šiam neveikiančiam komponentui kitas komponentas siunčia informaciją. nServiceBus komponentas tokius atvejus apdoroja ir neleidžia prarasti informacijos.

nUnit Tests

nUnit Tests tai yra nemokamas .NET Framework komponentas skirtas testavimo optimizavimui[17]. Naudojant testavimo komponentą galima atlikti vienetų testavimą ir integracinį testavimą. Testavimo atlikimui yra tiesiog rašomi testai, kuriuos galima po vieną arba visus vienu metu paleisti ir taip pasitikrinti, ar sistema veikia korektiškai.

nRhinoMocks

nRhinoMocks yra atviro kodo komponentas skirtas MS .NET Framework technologijoms. Šis komponentas yra skirtas imitacinių duomenų generavimui vietoj neįgyvendintų operacijų. Tai yra imitacinių duomenų generatorius [18], kuris skirtas vartotojo sąsajos testuotojams, galutiniams sistemos klientams parodyti kuriamos sistemos būsimą funkcionalumą ir pan.

CastleWinsdor

CastleWinsdor yra atviro kodo komponentas skirtas MS .NET Framework technologijoms. CastleWinsdor yra skirtas pridėti papildomą sistemos funkcionalumą jos nemodifikuojant [19]. Naudojant konfigūracinius failus galima atlikti sistemos komponentų konfigūravimą atsiejant juos nuo programos kodo. Tai padidina sistemos pakartotinio

panaudojimo vertę, nes norint pakeisti jos konfigūraciją, užtenka redaguoti sistemos konfigūracinius failus ir yra išvengiamas sistemos ar jos dalių perkompiliavimas.

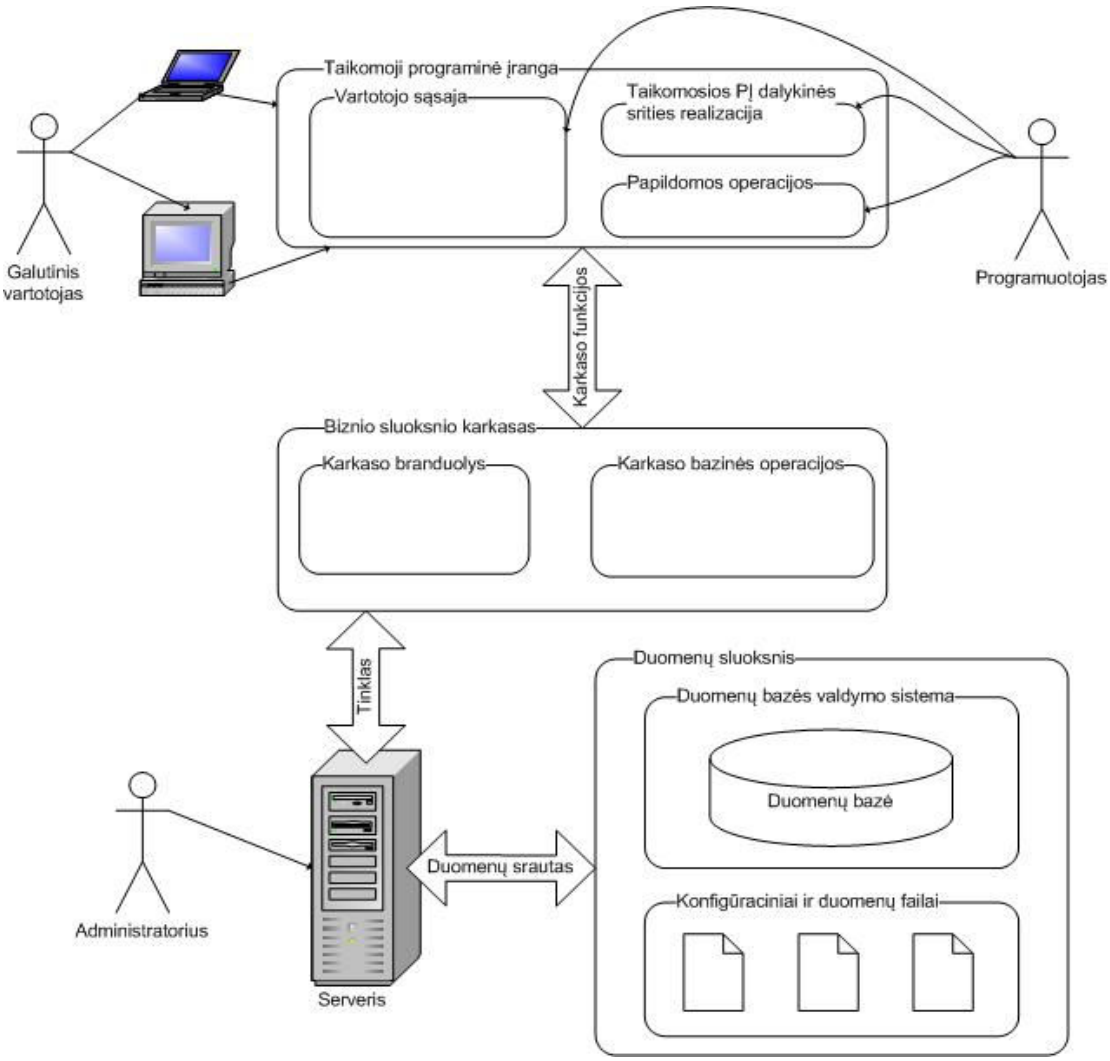
3. BIZNIO SLUOKSNIO KARKASAS

Biznio sluoksnio karkasas, bendras kuriamoms programoms, skirtas naudoti naujai kuriamoms paskirstytoms programinėms įrangoms kaip architektūrinis pagrindas. Pagrindinė šio karkaso paskirtis automatizuoti naujų programų kūrimą ir palengvinti jas kurti įvedant iš anksto suplanuotą busimos programos architektūrą. Šis karkasas apima paskirstytų sistemų biznio funkcijų dalį. Ši programinė įranga sukurta naudojantis šiuolaikinėmis projektavimo ir programavimo technologijomis.

3.1. Karkaso veiklos kontekstas

Sukurtas karkasas visų pirma yra biznio sluoksnio programinės įrangos pamatas. Atskirai jis nėra naudojamas (nebent parodomaisiais, pristatymo, testavimo, eksperimentiniais tikslais). Santykius tarp karkaso bei tam tikros taikomosios programos apibrėžia žemiau pavaizduota diagrama (7 pav.).

Diagramoje matome, jog taikomoji programa, atlikdama biznio sluoksnio funkcijas, naudojasi karkaso komponentais ir programuotojo papildomais suprogramuotais komponentais. Karkasas apibrėžiamas, kaip sluoksnis tarp duomenų bazės bei taikomosios programos.



7 pav. Karkaso veiklos konteksto diagrama.

3.2. Karkaso suteikiamas funkcionalumas

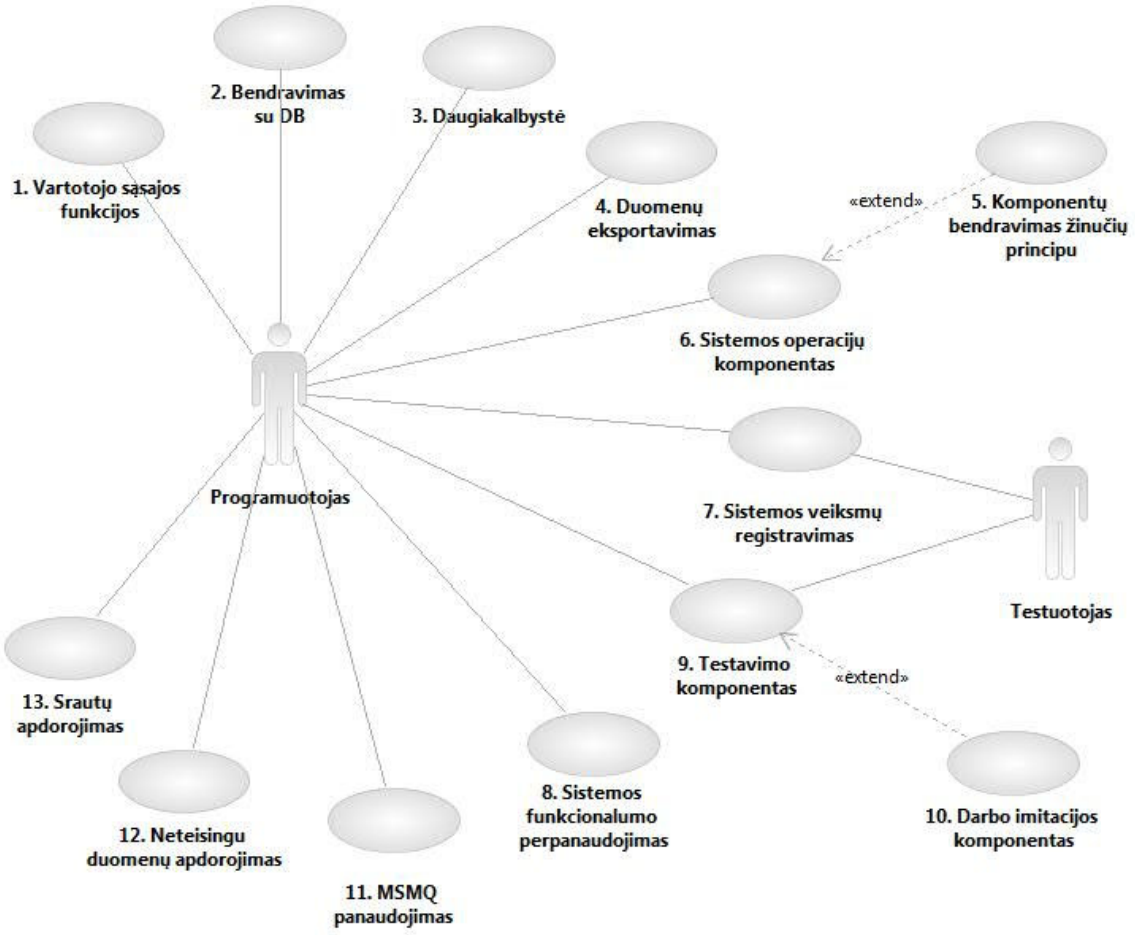
Karkaso funkcionalumui pavaizduoti labiausiai tinkama yra panaudos atvejų diagrama.

Sistemoje identifikuojami du pagrindiniai aktoriai:

- Programuotojas;
- Testuotojas.

Programuotojas – tai aktorius, kuris gali panaudoti visus sistemos panaudos atvejus. Prie programuotojo priskiriamas ir sistemų analitikas, architektas, kadangi kuriant sistemą jie analizuos bei panaudos sistemos karkasą.

Testuotojas, remiantis panaudos atvejų diagrama, panaudoja tik su sistemos testavimu susijusius komponentus, tokius kaip sistemos veikimo įrašu registravimas, tam tikrų operacijų veikimo imitavimas ir pan.



8 pav. Panaudos atvejų diagrama.

Biznio sluoksnio karkasas buvo kuriamas dviejų studentų Donato Gogio ir Žygimanto Šilansko, mano sukurta dalis apima 1-7 panaudos atvejus.

3 lentelė. Panaudos atvejų aprašai.

Panaudos atvejis	Aprašas
1. Vartotojo sąsajos funkcijos	Skirtas karkaso vartotojo sąsajos funkcijoms apibrėžti bei joms atvaizduoti. Taip pat tai gali būti vartotojo sąsajos karkasas, kurio pagrindu veiks vartotojo aplikacija.
2. Bendravimas su DB	Skirtas karkaso bendravimui su duomenų bazės valdymo sistema. Sudarytas iš duomenų trynimo, užsaugojimo, paieškos, duomenų apsikeitimo saugumo ir kitų funkcijų. Šis komponentas nėra priklausomas nuo duomenų bazės valdymo sistemos ir tinkamas visoms DBVS.
3. Daugiakalbystė	Komponentas skirtas programose įgyvendinti daugiakalbystę (pvz. programos vartotojo sąsają galima matyti keliomis kalbomis. Taip pat nesudėtingai įvesti naujas kalbas ir pan.)
4. Duomenų eksportavimas	Naudingas funkcionalumas jei norima atlikti duomenų eksportavimą į MS Word, MS Excel, PDF, tekstinius failus. Biznio sluoksnio sistemose to prireikia dažnai.
5. Komponentų bendravimas žinučių principu	Apibrėžia bendravimą tarp dviejų komponentų ir bendravimą tarp komponentų ir duomenų bazės žinučių (request – response) principu. Tai reiškia, kad visi duomenys yra įrašomi į objektą „request“ ir tik tada perduodami komponentui ar operacijai, o rezultatas įrašomas į „response“ tipo objektą ir grąžinamas.
6. Sistemos operacijų komponentas	Šis panaudos atvejis apibrėžia sistemoje vykdomus veiksmus t.y. visus veiksmus (operacijas) vykdys šis panaudos atvejis bei praneš apie tam tikros operacijos įvykdymą/neįvykdymą. Pvz.: paduodama operacija apie konkretaus įrašo, konkrečioje lentelėje išsaugojimą, ši informacija paduodama „request“ tipo objekte, užsaugojimo informacija pateikta operacijos objekte. Komponentas įvykdęs operaciją gaus „response“ tipo objektą.
7. Sistemos veiksmų registravimas	Nurodytų sistemos veiksmų registravimo (Loging) panaudos atvejis, kuris registruojamą informaciją pateiks į failą ir panašiai. Šis komponentas skirtas palengvinti klaidų aptikimą, taip pat atkurti vartotojo atliktus veiksmus su programine įranga.

Pagrindiniai apribojimai, kuriuos sistema privalo tenkinti, yra skirstomi į šiuos punktus:

- Karkasas turi būti lengvai praplečiamas įdiegiant naujus programinius modulius ir plečiant jau esamus;
- Turi užtikrinti nepriklausomumą nuo naudojamos duomenų bazių valdymo sistemos;
- Turi būti užtikrintas aukštas saugumo lygis bendravimui su duomenų baze ir tarp atskirų sistemos komponentų;
- Karkasas turi registruoti veiksmus, kuriuos vykdo naujai kuriama sistema;
- Naujų sistemų kūrimas turi būti paprastas, greitas ir kokybiškas;
- Programos architektūra leistų nesudėtingai ir greitai pakeisti sistemos funkcionalumą;
- Turėti atskirą automatinių testų modulį.

3.3. Karkaso architektūra

Karkaso architektūros tikslas yra sukurti tokį programinių sistemų karkasą, kad naujų projektų sukūrimas ar jau esančio vystymas būtų kuo paprastesnis ir greitesnis, pareikalaujantis kiek įmanoma mažiau darbo. Taip pat pritaikyti šį karkasą kuo platesniam spektrui naujai kuriamų programinių sistemų, nesvarbu ar tai būtų Windows operacinės sistemos taikomoji programa ar internetinis projektas.

3.3.1. Architektūros tikslai ir apribojimai

Architektūros tikslai:

- Karkaso architektūra turi būti suprojektuota taip, kad ją galima būtų lengvai išplėsti ir lengvai prijungi naujus modulius ar komponentus;
- Karkaso architektūra turi leisti lengvai ir be didelių laiko sąnaudų atlikti pakeitimus sistemoje;
- Karkaso architektūra turi leisti lengvai testuoti sistemoje veikiančias operacijas bei turėti galimybę talpinti automatinius testus;
- Naudoti nemokamus atviro kodo komponentus, kurie suteis sistemai didesnį funkcionalumą.

Kiekvienas karkaso komponentas bus pakartotinai panaudojamas, taigi jie turi būti labai lankstūs.

Karkaso architektūros apribojimai:

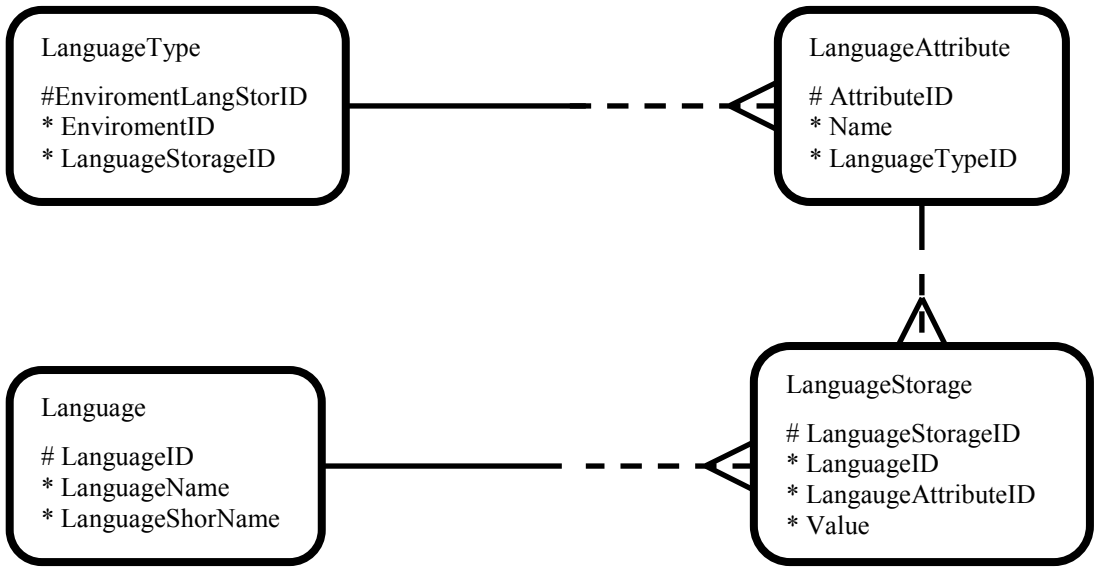
- Bendravimas su duomenų baze turi nepriklausyti nuo duomenų bazės valdymo sistemos, nes nėra žinoma kokią DBVS naudos nauja kuriama sistema.
- Kuriama sistema pateikta kaip atviro kodo, nekomercinė programinė įranga.
- Sistemos architektūra turi būti pritaikyta ne konkrečiai sistemai su konkrečiais reikalavimais, o kiek įmanoma didesniam spektrui naujai kuriamų sistemų.

Kuriant karkaso architektūrą buvo vengiama naudoti komercinius produktus, kadangi jie brangiai kainuoja, o sistemos kūrimo biudžetas yra minimalus. Taigi yra naudojami nemokami ir aukšto patikimumo atviro kodo komponentai: *nHibernate, Log4net, nUnit Tests, Castle Winsdor, nServiceBus, nRhinoMocks*.

3.3.2. Duomenų modelis

Šis projektas visų pirma yra kaip architektūrinis pagrindas naujai kuriamo projekto ir jame nėra apibrėžiami duomenys, kadangi nėra žinomas galutinis vartotojas. Karkasas yra kaip funkcijų rinkinys, kurios yra suteikiamos naudojant šį karkasą naujai kuriant projektą. Taigi pagrindiniai šio karkaso duomenys yra jo suteikiamos funkcijos ir apibrėžta architektūra, taigi – klasių diagrama.

Tačiau yra kaupiami tam tikri duomenys daugiakalbystės duomenims kaupti. Šių duomenų vaizdas atvaizduotas ER diagramoje.



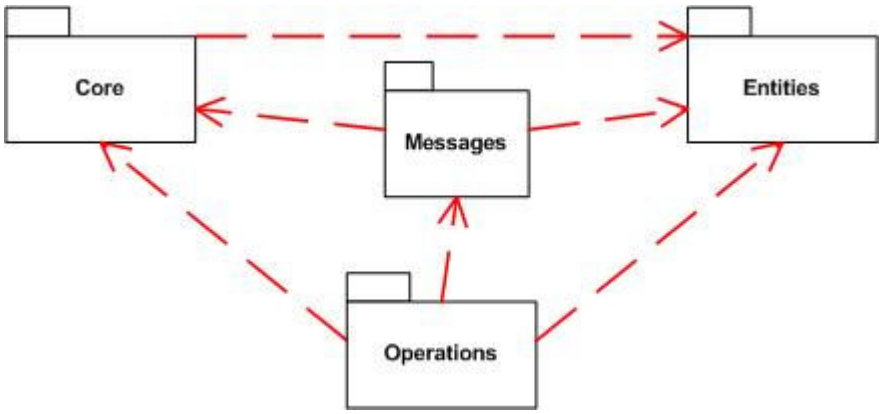
9 pav. Duomenų vaizdo ER diagrama.

4 lentelė. Esybių aprašai.

Esybė	Aprašymas
Language	Saugoma informacija apie kalbas sistemoje.
LanguageStorage	Saugoma informacija apie sistemoje egzistuojančiu atributų tekstą tam tikroje kalboje.
LangaugeAttribute	Saugoma informacija apie sistemoje egzistuojančius atributus.
LanguageType	Saugoma informacija apie sistemoje egzistuojančius atributo tipus.

3.3.3. Architektūros realizacija

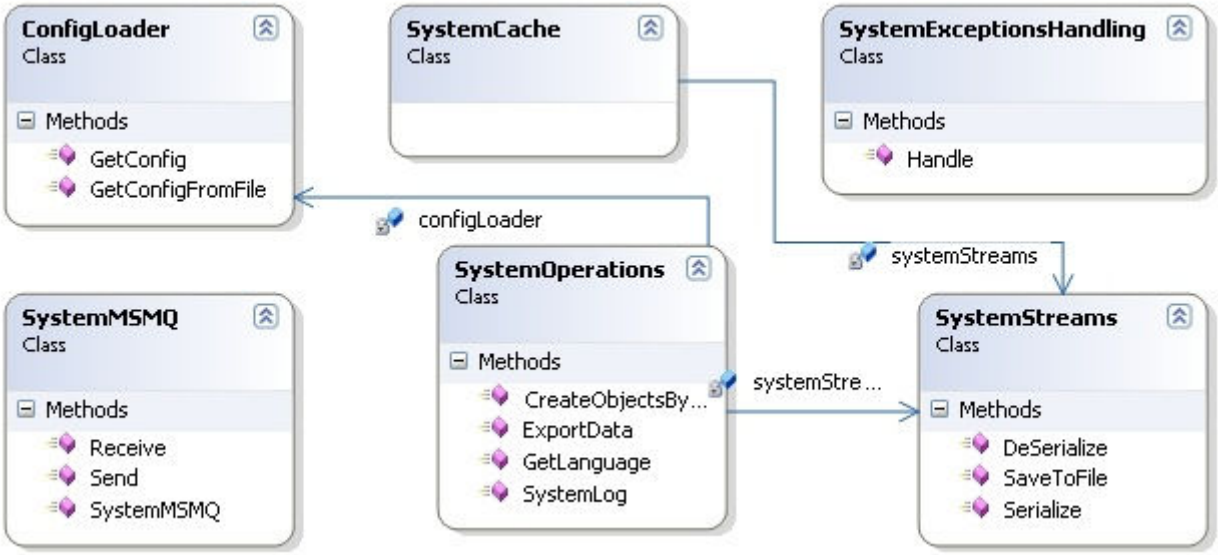
Biznio sluoksnio karkasas, bendras kuriamoms programoms yra suskirstytas į keturis klasių paketus, kurie kartu suteikia sistemai tam tikrą funkcionalumą. Kiekvieną iš šių klasių paketų galima laikyti kaip atskira programa, kur kiekvienas paketas teikia tam tikras funkcijas kitam paketui ir visų šių paketų visuma sudaro programos karkasą. Karkasas yra išskaidomas į 4 paketus. Toliau bus pateikiami kiekvieno paketo esminiai aspektai.



10 pav. Karkaso klasių paketų diagrama.

„Core“ klasių paketas.

Pakete realizuojamos visos sisteminės operacijos, tai yra pats žemiausias sisteminių operacijų lygmuo. Šios klasės naudoja .NET Framework, įvairias Windows bei trečių šalių bibliotekas. Šis paketas tai tarsi karkaso funkcionalumo branduolys.



11 pav. Paketo „Core“ klasių diagrama.

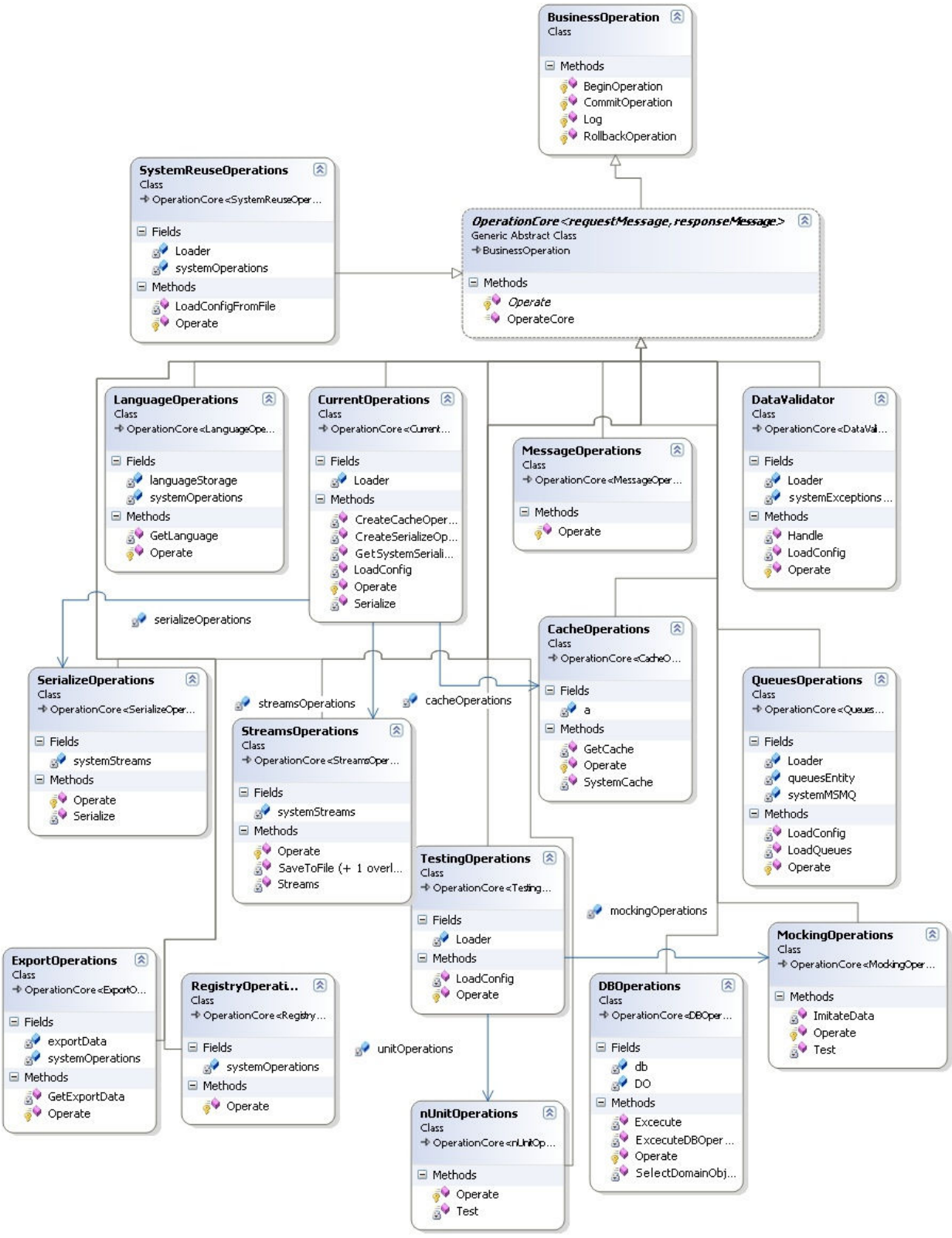
5 lentelė. Paketo „Core“ aprašas.

Klasifikacija
Paketas
Apibrėžimas
Pakete pateikiamos klasės sudaro karkaso branduolį. Jos yra nemodifikuojamos ir nekeičiamos. Paketas „Core“ yra pagrindinis sistemos paketas. Jį sudaro sistemos pagrindinės atominės sistemos funkcijos.
Atsakomybės
Paketas reprezentuoja sistemos branduolį ir vykdo visos sistemos ir jos modulių atomines funkcijas.
Apribojimai
Paketo klasių ir funkcionalumo keisti negali busimų sistemų programuotojai, kurie naudosis šiuo karkasu.
Struktūra
Komponentą sudaro klasės pavaizduotos „Core“ pateiktoje klasių diagramoje.
Sąveikavimas
Paketas sąveikauja su visais kitais karkaso paketais ir teikia jiems pagrindines karkaso funkcijas. Klaidos šiame komponente privers klaidingai veikti visą sistemą.
Resursai
Naudoja *Castle Winsdor, MSMQ, Log4net* komponentus.

Skaičiavimai
Skaičiavimai detalizuojami paketo klasių metodų aprašymuose.
Sąsaja/eksportas
Vartotojo grafinės sąsajos neturi. Turi konfigūracinius failus, kurie gali būti konfigūruojami programuotojo arba sistemos administratoriaus.

„Operations“ klasių paketas.

Čia įgyvendinti visi karkaso ir busimosios sistemos panaudos atvejai. Šiose klasėse bus atliekamos visos operacijos (funkcijos), kurias galės atlikti karkasas ir busimoji programa. Per šias klases bus galima prieiti prie kitų paketų klasių. Taigi, tai yra karkaso funkcionalumas, kuriuo galės naudotis programų kūrėjai ir sąsajos. Bazinė klasė yra „OperationCore“, ji yra abstrakti ir ją paveldi visos kitos operacijų klasės.



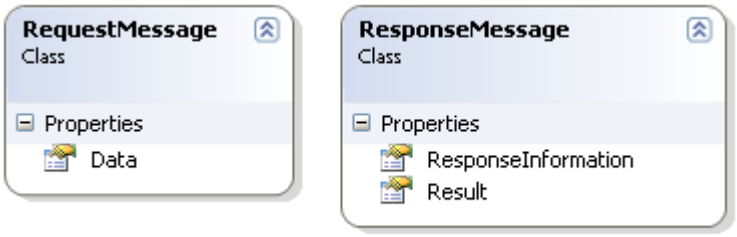
12 pav. Paketo „Operations“ klasių diagrama.

6 lentelė. Paketo „Operations“ aprašas.

Klasifikacija
Paketas
Apibrėžimas
Pakete pateikiamos klasės sudaro karkaso vykdomų operacijų branduolį. Šio paketo suteikiamomis funkcijomis naudosis vartotojo sąsaja per interfeisus.
Atsakomybės
Paketas reprezentuoja klases, kuriomis bus atliekamos visos operacijos (funkcijos), per šias klases bus galima prieiti prie kitų paketų klasių. Tai yra karkaso funkcionalumas, kuriuo galės naudotis programų kūrėjai.
Apribojimai
Paketo klases ir funkcionalumą gali keisti karkaso naudotojai – programuotojai, tačiau jie negalės keisti bazinių šio paketo klasių, kadangi pakeitimai gali iššaukti kritines klaidas sistemoje.
Struktūra
Komponentą sudaro klasės pavaizduotos „Operations“ pateiktoje klasių diagramoje.
Sąveikavimas
Paketas sąveikauja su visais kitais karkaso paketais ir teikia karkaso programuotojui pageidaujamas funkcijas.
Resursai
Duomenų bazė, taip pat naudoja *nHibernate, Log4net, nUnit Tests, nServiceBus, MSMQ, nRhinoMocks* komponentus.
Skaičiavimai
Skaičiavimai detalizuojami paketo klasių ir metodų aprašymuose.
Sąsaja/eksportas
Vartotojo grafinės sąsajos neturi. Pakete yra tik funkcijos suteikiamos vartotojai sąsajai.

„Messages“ klasių paketas.

Paketą sudaro klasės, kurios atsakingos už duomenų bei rezultatų pernešimą. „RequestMessage“ bazinė klasė, kurios objektai perneša duomenis operacijai, rezultatai perduodami per klasės „ResponseMessage“ objektą.



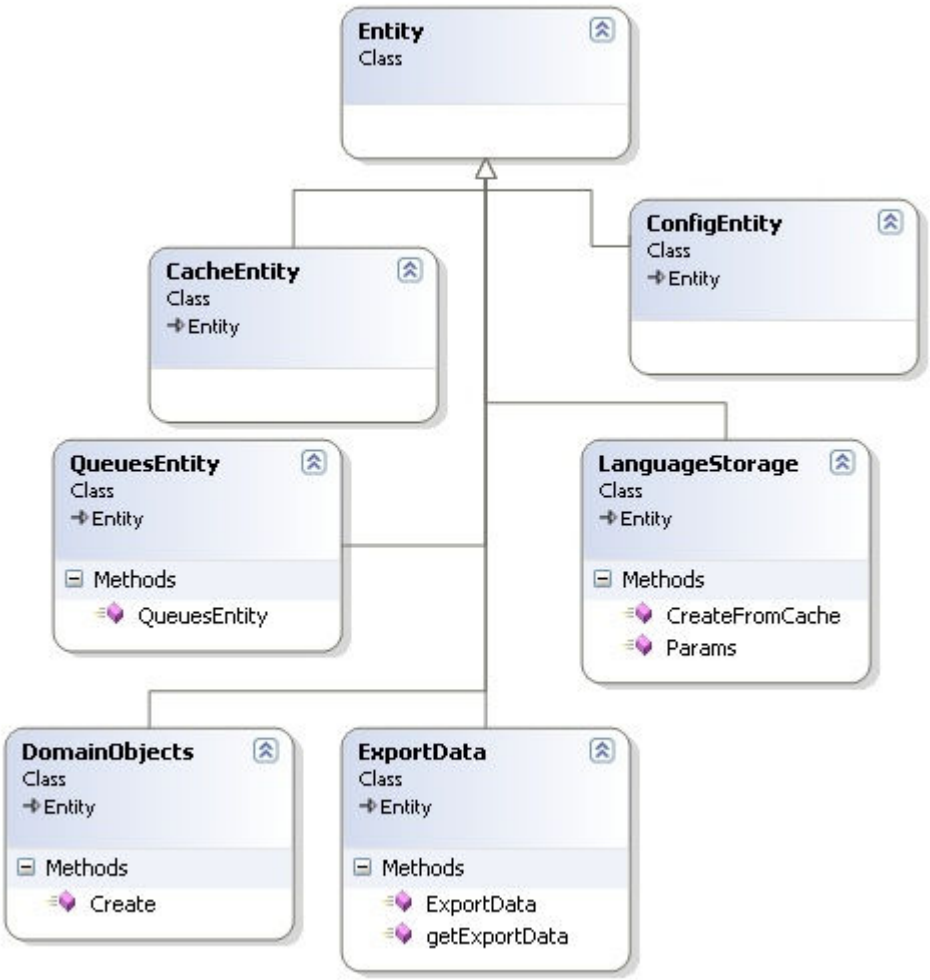
13 pav. Paketo „Messages“ klasių diagrama.

7 lentelė. Paketo „Messages“ aprašas.

Klasifikacija
Paketas
Apibrėžimas
Paketą sudaro klasės, kurios atsakingos už duomenų bei rezultatų pernešimą.
Atsakomybės
Pernešti karkaso duomenis iš vieno komponento kitam, teikti bei gauti duomenis vykdomų operacijų.
Apribojimai
Paketo klasės skirtos tik siunčiamų ir gaunamų duomenų funkcijoms vykdyti.
Struktūra
Komponentą sudaro klasės pavaizduotos „Messages“ pateiktoje klasių diagramoje.
Sąveikavimas
Paketas sąveikauja su visais kitais karkaso paketais ir teikia jiems pagrindines karkaso bendravimo funkcijas. Klaidos šiame komponente privers klaidingai veikti visą sistemą.
Resursai
Karkase naudojami objektai.
Skaičiavimai
Šiame pakete nėra atliekami skaičiavimai, jis yra skirtas tik pernešti informaciją.
Sąsaja/eksportas
Nėra

„Entities“ klasių paketas.

Pakete yra klasės, kurios saugo duomenis – esybes. Esybės paveldi bazinę klasę „Entity“. Ši klasė turi įvairias operacijas, kurios yra susijusios su esybėmis. Šiais duomenimis naudosis kitose paketuose esančios klasės.



14 pav. Paketo „Entities“ klasių diagrama.

8 lentelė. Paketo „Entities“ aprašas.

Klasifikacija
Paketas
Apibrėžimas
Paketą sudaro klasės, kurios atsakingos už duomenų ir esybių saugojimą karkase, kuriais bus operuojama sistemoje.
Atsakomybės
Saugoti karkaso esybes ir pagrindinius duomenų objektus bei atlikti elementarias funkcijas su duomenų objektais.

Apribojimai
Paketo klasės skirtos tik saugoji sistemos esybėms ir paprastiems esybių duomenų pakeitimams.
Struktūra
Komponentą sudaro klasės aprašytos pakete „Entities“ pateiktoje diagramoje ir būsimos sistemos esybės, kurias pridės programuotojai kuriant naują sistemą.
Sąveikavimas
Paketas sąveikauja su visais kitais karkaso paketais ir teikia jiems karkaso esybių objektus ir paprastas jų funkcijas. Klaidos esybių saugojime privers sistemą klaidingai saugoti duomenis, kas šiuolaikinėse sistemose yra jautri nuostoliams sritis.
Resursai
Karkase naudojami esybių objektai, duomenų bazė.
Skaičiavimai
Šiame pakete nėra atliekami skaičiavimai, jis yra skirtas tik saugoti esybių objektus bei pateikti juos įvairia norima forma.
Sąsaja/eksportas
Paketas daugumoje atspindi duomenų bazės vaizdą tik objektų pavidalu.

4. BIZNIO SLUOKSNIO KARKASO KOKYBĖS TYRIMAS

Šiame skyriuje yra atliekamas sukurto biznio sluoksnio karkaso kokybės, panaudojamumo tyrimas. Taip pat yra aprašomi sukurto karkaso privalumai, trūkumai ir galimi ateities patobulinimai.

4.1. Kokybės tyrimas

Vertinant sukurto biznio sluoksnio karkaso kokybę yra atsižvelgiama į studijų metu atlikto karkaso testavimą ir sukurtą testavimo medžiagą. Taip pat yra atsižvelgiama į ISO/IEC 9126 programinės įrangos kokybės vertinimo standartą ir ANSI/IEEE 1471-2000 standartą skirtą architektūros aprašymui programine įranga pagristoms sistemoms.

4.1.1. Kokybės įvertinimas pagal testavimo medžiagą

Vykdant sukurto biznio sluoksnio karkaso testavimą buvo laikomasi apibrėžtų testavimo metodikų ir sukurti automatizuoti testai naudojantis *nUnit Tests* komponentu. Visu projekto vykdymo metu buvo kuriami automatiniai testai kiekvienai sukurtai operacijai, tiek bazinėms karkaso operacijoms, tiek operacijoms skirtoms konkrečiai dalykinei sričiai. Tokia testavimo metodologija ir planas buvo pasirinktas dėl karkaso kokybės užtikrinimo.

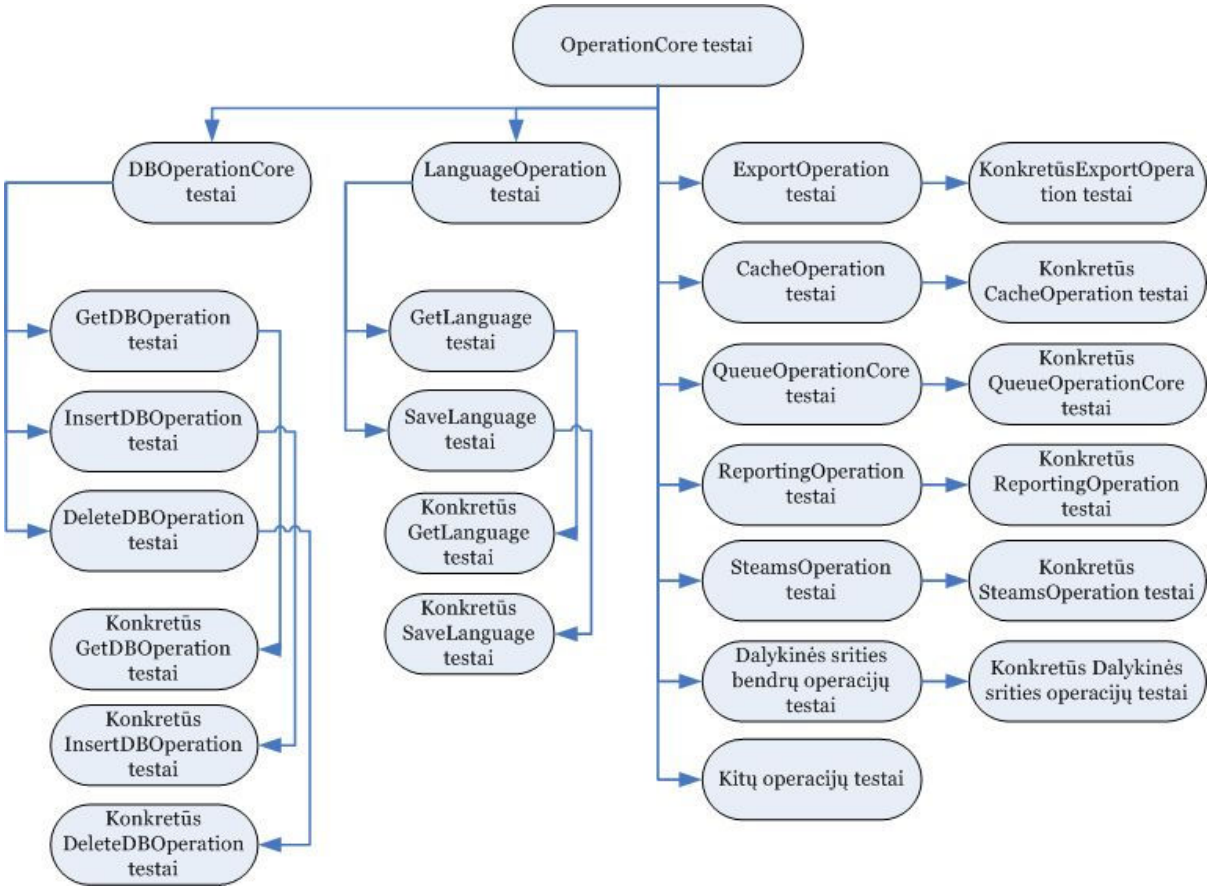
Pats karkaso kūrimas yra testais paremtas programavimas, nes architektūros kokybė programinėje įrangoje yra kritinė sritis, kurioje nėra leistinos jokios klaidos dėl didelių kaštų, nes norint šias klaidas pašalinti vėlesniuose programinės įrangos kūrimo ar palaikymo stadijose tai gali brangiai kainuoti. Testais paremtas programavimas – programavimo technika, kai tam tikras testas yra rašomas programavimo metu ir iš esmės lemia programinės įrangos kūrimą. Kad būtų sėkmingai taikomas testai paremto programavimo metodas, reikia surinkti tam tikras testavimo metrikas. Testavimo kokybei užtikrinti yra naudojamos šios metrikos:

- Testinių atvejų programinio kodo padengimo procentas;
- Testinių atvejų ir patikrinimų kiekis;
- Kodo eilučių kiekis testuose;
- Bendras visų testų atlikimo laikas.

Pati svarbiausia testavimo metrika yra testinių atvejų programinio kodo padengimo procentas. Ši metrika yra vienas iš pagrindinių testo efektyvumo rodiklių, kuris įvertina testavimo išsamumą ir testavimo kokybę. Dažniausiai yra iš anksto nustatomas minimalus kodo padengimo lygis, tačiau šiuo atveju jis turi būti kuo didesnis, nes yra testuojama jautri programinės įrangos sritis. Ši metrika padeda surasti klaidas, o ne parodyti kad jų nėra, nes visų klaidų suradimas yra beveik neįmanomas.

Kitos trys metrikos yra apžvelgiamos kartu. Testinių atvejų ir patikrinimų kiekis yra toks, kad būtų ištestuotas visas karkaso funkcionalumas tam tikromis dalimis. Kodo eilučių kiekis testuose nėra svarbus, svarbu tik tai, kad būtų padengiamas kuo didesnis karkaso funkcionalumo testavimas. Bendras visų testų atlikimo laikas yra iš dalies svarbi metrika, nes kuo ilgiau atliekami testai, tuo didesni nepatogumai sukeliami programuotojui juos paleidžiant. Taip pat stipriai padidėjus testų veikimo laikui gali būti identifikuojamos karkaso klaidos.

Žemiau pateiktoje diagramoje (15 pav.) yra pavaizduota karkaso parašytų automatinių testų hierarchija.



15 pav. Karkaso testų hierarchijos diagrama.

Automatinių testų rašymui buvo skirta tiek pat resursų kaip ir karkaso funkcionalumui sukurti. Todėl buvo sukurtas testavimo komponentas, kurį galima paleisti bet kada ir patikrinti karkaso bei kuriamos sistemos aptiktų klaidų kiekį. Iš pradžių yra paleidžiami hierarchijos viršuje esantys testai, po to paleidžiami šakų testai ir t.t. Jei aptinkama klaida operacijoje, kuri turi papildomų šakų, tai tų šakų testai nėra leidžiami, nes jų testų rezultatai gali būti nekorektiški. Pavyzdžiui, jeigu yra atliekamas testas konkrečios operacijos, kuri atlieka duomenų ištraukimą iš duomenų bazės pagal pasirinktis paieškos kriterijus, kad ši operacija būtų ištestuota turi veikti visi *GetDBOperation*, *DBOperationCore* ir *OperationCore* testai.

Tokio pasirinkto testavimo privalumai:

- Testais paremtas programavimas padeda greičiau sukurti programinę įrangą, nes kokybiškas programos kūrimas yra valdomas kiekviename kūrimo žingsnyje, todėl mažėja klaidų skaičius vėlesniuose kūrimo etapuose;
- Sukurti testavimo atvejai lemia mažesnį klaidų atsiradimą naujame kode, kai naujas programinis kodas priklauso nuo jau sukurto ir ištestuoto programinio kodo;
- Sukurtus testavimo atvejus galima bet kada paleisti ir patikrinti kuriamos sistemos korektiškumą ir klaidų kiekį;
- Testai pagerina programinio kodo dizainą, nes priverčia programuotoją kurti lengviau skaitomą programinį kodą.

Tokio pasirinkto testavimo trūkumai:

- Kokybiškų testinių atvejų rašymas užima labai daug laiko ir kartais gali net ilgiau užtrukti negu testuojamos operacijos sukūrimas.
- Sunku kontroliuoti kokybiškų testų rašymą.

Dauguma defektų buvo pašalinta programavimo fazės metu, o automatiniai testai padėdavo išvengti klaidų modifikuojant operacijas, kurios turi ryšį su kitomis operacijomis. Daugumoje testavimo procedūrų buvo aptikti defektai, kurie buvo sėkmingai pašalinti. Taigi automatiniai testai turi didelę įtaką kokybiškos programinės kūrimui.

4.1.2. Kokybės įvertinimas pagal standartus

Kokybės įvertinimas pagal ANSI/IEEE 1471-2000

ANSI/IEEE 1471-2000 standartas, aprašytas skyriuje 2.3.1, visų pirma yra rekomendacijos architektūros aprašymui programine įranga pagristomis sistemomis. Karkaso architektūros kokybės užtikrinimui šio standarto rekomendacijomis buvo naudojamasi architektūros projektavimo metu surinkus karkaso reikalavimus. Remiantis standartu ir reikalavimų analize buvo sudaryti architektūros atvaizdai ir pagal šiuos atvaizdus sudarytas karkaso architektūros modelis.

Nors šis standartas tiesiogiai neįvertina programinės įrangos ar jos architektūros kokybės, bet padeda sudaryti kuo tikslesnį architektūros modelį. O nuo karkaso architektūros, kuri yra kaip pamatas kuriamos programinės įrangos, didžiąja dalimi priklauso visos sistemos kokybė. Taigi buvo stengiamasi naudoti šį standartą kuriant kokybišką ir reikalavimus atitinkančią architektūrą.

Kokybės įvertinimas pagal ISO/IEC 9126

Sukurto biznio sluoksnio karkaso kokybės įvertinimas atliekamas pagal ISO/IEC 9126 standarto, aprašyto skyriuje 2.3.2, vertinimo charakteristikas. Šis standartas suformuluoja charakteristikas ir subcharakteristikas, kurios tiesiogiai įvertina sukurtos programinės įrangos kokybę. Kokybės subcharakteristikų įvertinimo pagrindas yra aprašytas remiantis reikalavimų analize ir sukurtu produktu. Įvertinimai yra parinkti pačiam tiriant sukurto karkaso kokybę, remiantis vadovo nuomone ir kolegų programuotojų nuomone, kurie buvo supažindinti su karkaso galimybėmis ir funkcionalumu, taip pat remiantis surinkta testavimo medžiaga testuojant karkasą.

Kiekviena kokybės subcharakteristika yra įvertinama dešimtbalėje sistemoje, kur 1 balas reiškia prastą subcharakteristikos kokybės įvertinimą, o 10 balų reiškia aukštą subcharakteristikos kokybės įvertinimą. Žemiau pateiktoje lentelėje (lentelė 9) yra įvertinti biznio sluoksnio karkaso kokybės įvertinimo rezultatai. Kiekviena subcharakteristika yra įvertinama atskirai, po to yra skaičiuojama konkrečios charakteristikos bendras įvertinimo rezultatas. Gale yra apskaičiuojamas visų charakteristikų bendras įvertinimas.

9 lentelė. Karkaso kokybės įvertinimas pagal ISO/IEC 9126 standartą.

Funkcionalumas		
Subcharakteristika	*Įvertinimo pagrindimas*	*Įvertinimas*
Tinkamumas	Realizuoti visi funkciniai ir nefunkciniai reikalavimai.	9
Tikslumas	Karkaso operacijos grąžina norimus rezultatus.	10
Bendradarbiavimas	Karkasas pilnai gali būti sukonfigūruotas dirbti su kitomis programinėmis įrangomis.	10
Funkcinis atitikimas	Karkasas atitinka programinės įrangos kūrimo standartus ir įstatymus.	10
Saugumas	Karkasas yra apsaugotas bazinėmis priemonėmis, tačiau nesudėtingai gali būti atliktas patobulinimas.	8
	Funkcionalumo charakteristikos bendras įvertinimas:	9,4
Patikimumas		
Subcharakteristika	*Įvertinimo pagrindimas*	*Įvertinimas*
Brandumas	Karkase nėra kritinių klaidų, tačiau nenumatytose situacijose yra kviečiamas prieštaravimas (angl.: exception), kas konkrečiose sistemose turi būti patobulinta.	8
Klaidų tolerancija	Atsiradus klaidai karkasas toliau gali funkcionuoti. Visos klaidos yra registruojamos ir apdorojamos.	10
Atkuriamumas	Jeigu yra registruojama kritinė klaida, prarastus duomenis galima atstatyti. Atstatymas gali būti sudėtingas.	8
	Patikimumo charakteristikos bendras įvertinimas:	8,7
Panaudojamumas		
Subcharakteristika	*Įvertinimo pagrindimas*	*Įvertinimas*
Suprantamumas	Programuotojui užtruks laiko perskaityti karkaso dokumentaciją ir suprasti karkaso veikimo principus.	8
Išmokstamumas	Bendra karkaso architektūros logika nėra labai sudėtinga ir lengvai įsimenama.	9
Veiksnumas	Perskaičius vartotojo dokumentaciją karkaso pagrindu nėra labai sudėtinga programuoti.	8
Patrauklumas	Ne visi programuotojai gali noriai priimti tokį programavimo būdą.	7
	Panaudojamumo charakteristikos bendras įvertinimas:	8

Efektyvumas		
Subcharakteristika	*Įvertinimo pagrindimas*	*Įvertinimas*
Laikinė elgsena	Karkaso papildomi veiksmai vykdant operaciją prailgina jos vykdymo laiką.	8
Išteklių panaudojimas	Karkaso pilnas pajėgumas naudojamas tik vykdant operacijas.	9
	Efektyvumo charakteristikos bendras įvertinimas:	8,5
Palaikomumas		
Subcharakteristika	*Įvertinimo pagrindimas*	*Įvertinimas*
Analizuojamumas	Karkaso kodas yra paprastais skaitomas, klaidų pranešimai yra informatyvūs.	10
Keičiamumas	Reikalingas mažas pastangų kiekis karkaso modifikacijoms atlikti.	9
Stabilumas	Po karkaso modifikacijų visi kiti veiksmai yra nuspėjami.	9
Testuojamumas	Sukurtu testavimo komponentu gali lengvai aptikti klaidų kiekį ir lengvai testuoti karkasą	10
	Palaikomumo charakteristikos bendras įvertinimas:	9,5
Perkeliamumas		
Subcharakteristika	*Įvertinimo pagrindimas*	*Įvertinimas*
Pritaikomumas	Karkasas pritaikytas dirbti tik Windows operacinėse sistemose, bet palaiko daugelį duomenų bazių valdymo sistemų.	9
Įdiegiamumas	Karkasą paprasta įdiegti programavimui.	10
Atitikimas	Karkasas atitinka perkeliamumo standartus.	10
Pakeičiamumas	Karkaso pagrindu galima perkurti jau sukurtą programinę įrangą ir panaudoti ją jos aplinkoje	9
	Perkeliamumo charakteristikos bendras įvertinimas:	9,5
	Karkaso kokybės įvertinimas:	9

Kaip matome atlikus kokybės vertinimą pagal ISO/IEC 9126 standartą bendras karkaso kokybės įvertinimas yra lygus 9 balams.

4.1.3. Kokybės tobulinimo galimybės

Biznio sluoksnio karkaso kokybės tyrimas buvo atliktas remiantis: testavimo medžiaga, tarptautiniais standartais ANSI/IEEE 1471-2000 ir ISO/IEC 9126. Pats biznio sluoksnio karkasas nuo pat jo kūrimo ir projektavimo pradžios buvo orientuotas į aukštą kokybę, nes programinės įrangos architektūra yra jautriausia sritis visoje sistemoje ir klaidos architektūroje ateityje gali kainuoti labai daug. Taigi atlikus biznio sluoksnio karkaso kokybės tyrimą buvo pasiektas aukštas kokybės įvertinimas, ko ir buvo siekta viso projekto metu. Tačiau nors ir buvo pasiektas aukštas karkaso kokybės įvertinimas, bet jis nėra pats aukščiausias, tuo galima įsitikinti atlikus kokybės įvertinimą pagal ISO/IEC 9126 standartą.

Pagrindinės galimos karkaso kokybės tobulinimo galimybės:

- Kuriant naują sistemą karkaso pagrindu ir siekiant sukurti kokybišką produktą reikia toliau vystyti karkaso kokybines charakteristikas ir tęsti testavimo komponento kūrimą atsiradus naujoms operacijoms;
- Palaikomumo ir perkeliamumo kokybinės charakteristikos yra pačios svarbiausios karkasui ir jos turi būti maksimaliai gerinamos;
- Panaudojamumo kokybinės charakteristikos privalo būti pagerintos, kadangi jei karkasas nėra visapusiškai panaudojamas tai jo gyvavimo laikas gali sutrumpėti;
- Pagrindinė testavimo idėja yra paremta operacijų vienetų testavimų, tačiau yra mažai testuojama gimininguų operacijų tarpusavio sąveika. Siekiant aukštesnės testavimo kokybės ši sąveika taip pat turi būti ištestuota testavimo komponente.
- Naudoti tik tuos trečiųjų šalių komponentus, kurie yra pilnai ištestuoti ir turi aukštą kokybės įvertinimą.

4.2. Panaudojamumo tyrimas

Šiame skyriuje yra nagrinėjami sukurto biznio sluoksnio karkaso privalumai ir trūkumai.

4.2.1. Karkaso privalumai

Šiame skyrelyje yra išvardinami pagrindiniai karkaso privalumai. Kadangi apie karkaso funkcionalumą jau buvo rašoma yra paminėti tik pagrindiniai karkaso panaudojimo privalumai, o ne jo funkcionalumas.

Kaupiama operacijų ir žinių bazė

Kuriant įvairius projektus karkaso pagrindu galima struktūrizuotai kaupti programinį kodą ar tam tikrus sukurtus operacijų komponentus. Sukurtą funkcionalumą galima vėl nesudėtingai pakartotinai panaudoti ateities projektuose, nes biznio sluoksnio karkasas visų pirma yra orientuotas į pakartotinį panaudojimą. Taip yra kaupiama operacijų ir žinių bazė, kuri didintų įmonės pelną ateityje.

Trumpesnis kokybiško programavimo laikas

Kuriant didelius projektus struktūrizuotas programinės įrangos kūrimas leidžia padidinti kūrimo greitį, kai dalyvauja daug programuotojų, nes programos kodas tampa lengvai skaitomas ir nuspėjamas. Taip pat galima įterpti ir pakartotinai panaudoti tinkamas operacijas iš praeities projektų. Kokybė yra užtikrinama skyriuje 4.1.1 ištirtu testavimo komponentu.

Tinkamas daugeliui kuriamų sistemų

Biznio sluoksnio karkaso architektūra yra skirta verslo valdymo sistemoms realizuoti, kurios yra dažniausiai užsakomi produktai programavimo įmonėse šiuo metu. Taip pat operacijų pagrindu galima realizuoti daug kitų projektų.

Sukurtos visos bazinės funkcijos

Karkase yra realizuota daugelis dažniausiai pasikartojančių operacijų programinės įrangos kūrime, tokių kaip: bendravimas su duomenų baze, daugiakalbystė, duomenų eksportavimas, komunikacija tarp komponentų, veiksmų registravimas ir kt.

Karkaso dokumentacijos panaudojimas

Panaudojant karkaso dokumentaciją ir architektūros modelio pagrindines idėjas galima realizuoti konkrečiai sistemai skirtą architektūrą.

4.2.2. Karkaso trūkumai

Šiame skyrelyje yra išvardinami ir detaliai išanalizuojami pagrindiniai karkaso trūkumai. Pagal trūkumų aprašymus galima padaryti išvadas ar visgi verta naudoti sukurtą biznio sluoksnio karkasą ar kurti konkretų karkasą pritaikyta konkrečiai kuriamai sistemai.

Atviro kodo komponentų panaudojimas

Atviro kodo komponentai yra neatskiriama priemonė šių dienų programinės įrangos kūrime, kadangi jie leidžia sukurti programinę įrangą greičiau ir kokybiškiau bei užtikrina didesnį sistemos funkcionalumą. Tačiau ne visuomet atviro kodo komponentai sutaupo laiką kuriant naują sistemą, jie taip pat gali sukelti didelių problemų, o sukurtame biznio sluoksnio karkase jų naudojama daug. Pavyzdžiui, jei trečiosios šalies komponentas veikia ne taip kaip tikėtasi ir jį sukūrusi kompanija nustoja vystyti šį komponentą, atsiradusios klaidos šiame komponente gali sukelti nepataisomų problemų.

Bendrų reikalavimų sudarymas

Sukurti bendrą programos karkasą, kuris tiktų visoms naujai kuriamoms programinėms įrangoms, yra neįmanoma. Taip pat labai sudėtinga surinkti bendrus visų sistemų reikalavimus. Kiekviena sistema yra skirtinga ir turi skirtingus reikalavimus. Taigi galima tik nuspėti bendrus dalykus kaip: bendravimas su duomenų baze, bendravimas tarp komponentų, sistemos testavimas, neteisingų duomenų apdorojimas, duomenų eksportavimas ir kt. Tačiau neįmanoma nuspėti kokie reikalavimai yra skirti saugumui, bendravimo greičiui tarp komponentų, suderinamumo su jau sukurtomis sistemomis. Taigi karkasas gali būti visiškai netinkamas realizuojant tam tikras sistemas su specifiniais reikalavimais.

Architektūros pasirinkimo problemos

Programinės įrangos architektūros projektavimas vėl gi yra sudėtingas uždavinys. Bendrai sukurta programos architektūra naudojama konkrečiam uždaviniui realizuoti niekada nebus geresnė negu sukurta programos architektūra skirta konkrečiai šiam uždaviniui spręsti. Pagrindiniai trūkumai bendros programinės įrangos architektūros yra greitis ir nepakankamas ar neįmanomas funkcionalumas.

Karkaso gyvavimo laikas

Greitai keičiantis programavimo technologijomis ir atsirandant naujoms trečių šalių komponentų versijomis atsiranda didelė grėsmė karkaso gyvavimo laikui. Yra tikimybė, kad versijų pakeitimuose gali nelikti funkcionalumo, kuris buvo ankstesnėse komponento versijose ir jis yra naudojamas karkase. Senesnių versijų naudojimas gali būti netikslingas dėl to, kad tame komponente gali likti neištaisytų klaidų.

Karkaso veikimo greitis

Programų veikimo greitis visada yra svarbus programinės įrangos parametras. Šiais laikais yra derinami du parametrai: sistemos veikimo greitis ir sistemos sukūrimo laikas. Akivaizdu, kad sistemos veikimo greičio parametras nukenčia naudojant sukurtą biznio sluoksnio karkasą, tačiau pagreitėja kokybiškos sistemos sukūrimo laikas.

4.3. Patobulinimų tyrimas

Biznio sluoksnio karkaso bendro naujai kuriamoms programinės įrangoms patobulinimų sąrašas gali niekada nesibaigti. Visada bus kur tobulinti programų karkasą ar realizuoti papildomą funkcionalumą. Taigi bendro programų karkaso kūrimas yra ilgas ir iteracinis procesas. Siūlomi karkaso patobulinimai yra pateikti 10 lentelėje.

10 lentelė. Karkaso galimų patobulinimų sąrašas.

Operacijų versijavimas
Verslo valdymo sistemose dažnai keičiasi verslo logika. Pavyzdžiui, draudimo sistemose keičiasi įmokų apskaičiavimai, prekybos sistemos keičiasi mokesčių apskaičiavimo metodikos. Taigi tam puikiai pagelbėtų operacijų versijavimas, kuris būtų priklausomas nuo laikinės charakteristikos t.y. karkasas tam tikru laiku vykdytų tam tikrą operacijos versiją.
Duomenų versijavimas
Sukurti paprastą duomenų versijavimo galimybę, nes to dažnai prireikia verslo valdymo sistemose. Pavyzdžiui, prekybos sistemose prekės pavadinimas „Obuolys“ pakeičiamas į „Raudonas obuolys“ ir senose sąskaitose turi likti senas pavadinimas, naujose naujas ir pan. Duomenų versijavimas padėtų automatiškai valdyti tokius atvejus.
Operacijų konfigūracijų valdymas
Sukurti programiškai reguliuojamą operacijų konfigūracijų valdymą. Pavyzdžiui, programiškai atjungti operacijos veiksmų registravimą arba jį apriboti, pasirinkti operacijos vykdymo laiką, pasirinkti operacijos vykdymo prioritetą ir t.t.
Saugumo operacijų realizavimas
Realizuoti skirtingų operacijų vykdymo tam tikro saugumo užtikrinimo galimybę.
Sukurti galimybę lygiagrečiai vykdyti operacijas
Sukurti konfigūruojamą ir paprastą operacijų lygiagretaus vykdymo galimybę pagal prioritetus, kurie būtų programiškai konfigūruojami.

4.4. Perspektyvumo tyrimas

Šiuo metu programavimo verslas Lietuvoje ir pasaulyje turi dideles perspektyvas, kadangi kompiuterinė technika yra ištobulėjusi ir nė viena įmonė negali egzistuoti be kompiuterio ir informacinių technologijų. Taigi vis daugiau įmonių yra reikalingos informacinės sistemos, kurios padėtų atlikti įmonėje vykstančius procesus, realizuotų jų dalykinę sritį ir darbinę veiklą bei būtų pilnai pritaikyta įmonės poreikiams tenkinti. Greitam ir kokybiškam informacinių sistemų kūrimui reikalingos automatizuotos kūrimo priemonės. Taigi greitas sistemų kūrimas yra aktuali problema programavimo įmonėms ir šią problemą bandoma išspręsti šiuo biznio sluoksnio karkasu.

Programavimo įmonės kuriant naujos sistemos architektūrą turi tokias pasirinkimo galimybes:

- Jeigu įmonė turi savo sukurtą programų karkasą, tai jį naudoti ir šio karkaso pagrindu sukurti naują sistemą.
- Kurti specializuotą programų karkasą pritaikytą konkrečiai programiniai įrangai (tai užtrunka daugiau laiko ir kainuoja daugiau pinigų, tačiau sukurtas produktas yra kokybiškiausias).
- Kurti sistemą chaotiškai, kur pagrindinis tikslas – realizuoti kuriamos sistemos reikalavimus ir kuo greičiau patenkinti užsakovo lūkesčius.

Deja, dažniausiai yra pasirenkamas paskutinis variantas, nes naujos kuriamos sistemos biudžetas, kaip taisyklė, yra per mažas. Be to dažniausiai yra prašoma sistemą sukurti kuo greičiau ir pagrindinis užsakovo tikslas yra sistemos funkcionalumas, o ne jos kokybė. Kokybe užsakovas susirūpina vėliau, kai jau būna per vėlu ir pakeitimų kaštai stipriai išauga. Kartais susidaro net tokios situacijos, kai sukurti sistemą iš naujo kainuoja mažiau negu modifikuoti esamą liktinę sistemą.

Šiuo metu rinkoje daugumoje tik didelės programavimo įmonės naudoja iš anksto sukurtą karkasą įgyvendinant naujus projektus. Taip yra dėl to, kad mažosios įmonės neturi pakankamai lėšų investuoti į projektą, kuris per greitą laiko tarpą neatsiperka. Taigi sukurti naują universalų karkasą yra reikalingos investicijos.

Biznio sluoksnio karkasas turi dvi pagrindines perspektyvas:

- Parduoti šį karkasą programavimo įmonėms.
- Panaudoti šį karkasą naujos programinės įrangos kūrimui: kaip dokumentacija ir pagrindinėmis iškeltomis architektūros projektavimo idėjomis arba kaip sukurtu architektūriniu pagrindu.

Abi šios perspektyvos turi savų trūkumų. Pirmoji perspektyva yra sudėtinga, kadangi programavimo įmonės gali pačios susikurti savo karkasus ir vargu investuotų į kitų sukurtą karkasą t.y. gali juo nepasitikėti. Antroji perspektyva yra realesnė, tačiau sunku įsilieti į šiuo metu vyraujančia rinką, kadangi yra daug programavimo įmonių ir didelė konkurencija.

Taigi šis programų karkasas turi dideles perspektyvas kurti kliento poreikiams pritaikytas paskirstytas verslo valdymo sistemas dėl plačių jo panaudojimo galimybių.

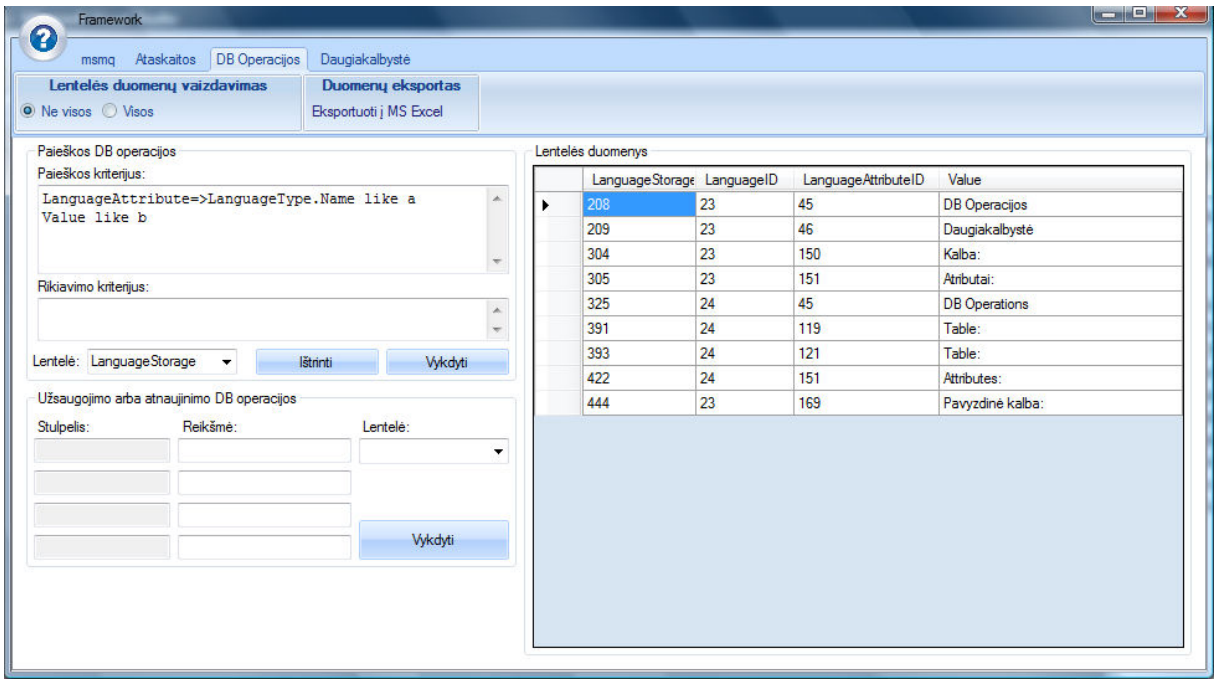
5. BIZNIO SLUOKSNIO KARKASO EKSPERIMENTINIS TYRIMAS

Šioje eksperimentinėje dalyje yra atliekamas magistratūros studijų metu sukurto biznio sluoksnio karkaso eksperimentinis tyrimas ir papildomai atliktų patobulinimų eksperimentinis tyrimas.

5.1. Panaudojimo eksperimentinis tyrimas

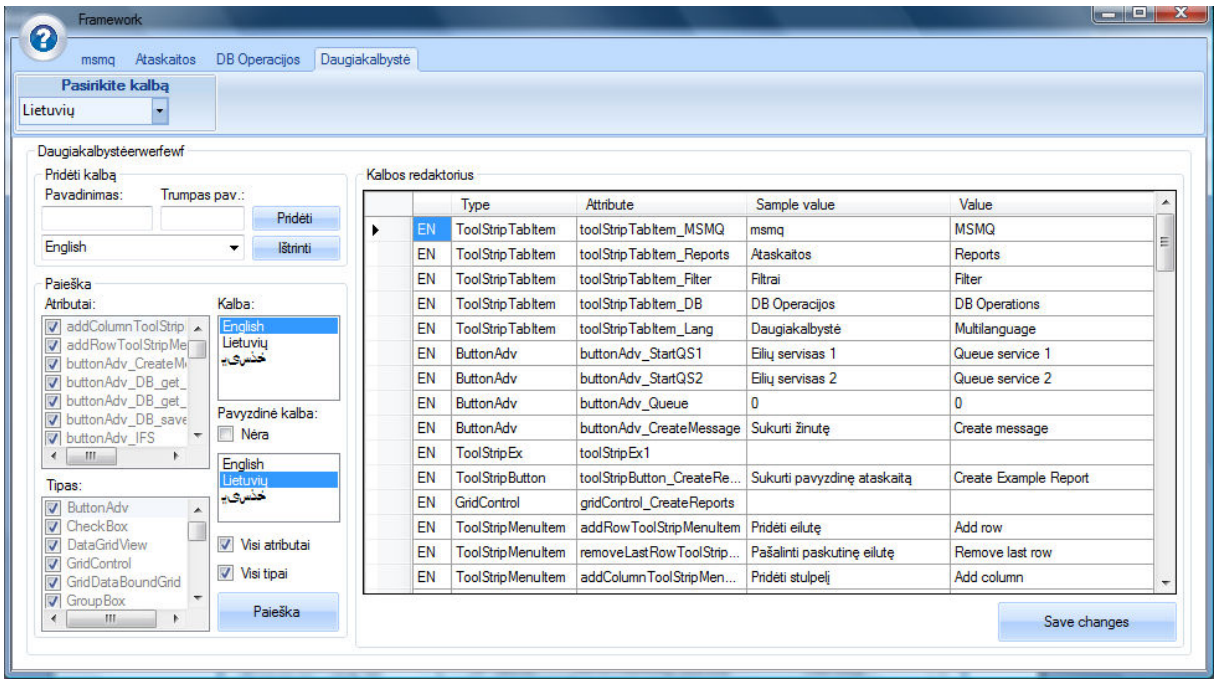
Biznio sluoksnio karkaso eksperimentui atlikti ir funkcionalumui pavaizduoti buvo sukurta vartotojo sąsaja. Vartotojo sąsajos pagalba yra vaizdžiai pavaizduotas operacijų vykdymas ir jų funkcionalumo galimybės.

Duomenų valdymo operacijų eksperimentui atlikti galima naudotis vartotojo sąsaja. Reikia užpildyti paieškos ir rikiavimo masyvus ir taip paeksperimentuoti su karkaso duomenų valdymo operacijomis.



16 pav. Duomenų valdymo operacijų eksperimentas.

Daugiakalbystės operacijos eksperimentui atlikti taip pat yra sukurta vartotojo sąsaja, kurios pagalba galima įvesti naują kalbą, ją pakeisti ir taip patikrinti šios operacijos našumą ir tinkamumą.



17 pav. Daugiakalbystės operacijos eksperimentas.

Kitoms mano realizuotoms operacijoms nebuvo sukurta vartotojo sąsaja, kadangi jų panaudojimas yra paprastas ir jos tik atlieka tam tikrą funkcionalumą. Taip pat eksperimentavimui galima panaudoti testavimo komponentą, kuriame yra testuojamos visos karkaso operacijos. Šiam eksperimentui atlikti yra būtinos programavimo žinios.

Programuotojai arba programų architektai gali nesudėtingai ir vaizdžiai išbandyti sukurtą biznio sluoksnio karkaso funkcionalumą naudojant vartotojo sąsają, taip pat naudojantis testavimo komponentu, ir taip įsitikinti šio karkaso reikalingumu arba visišku jo atmetimu.

5.1.1. Bendravimo su duomenų baze panaudojimo eksperimentas

Bendravimo su duomenų baze karkaso operacija yra atsakinga už duomenų traukimą pagal tam tikrus kriterijus iš duomenų bazės, duomenų užsaugojimą ir trynimą. Tai yra bazinė operacija bendravimui su duomenų baze, kurią turi paveldėti visos operacijos esančios karkase, kurios bendrauja su duomenų baze. Kuriant naują duomenų bazės operaciją (pvz.: *NewDBOperation*) ji privalo paveldėti *DBOperationCore* operacijos klasę.

Pavyzdžiui:

```
public class NewDBOperation : DBOperationCore<NewDBRequestMessage, NewDBResponseMessage>
{
  protected override NewDBResponseMessage Operate(NewDBRequestMessage rM)
  {
        // Veiksmai
  }
}
```

Taip pat reikia sukurti žinutes (pvz.: *NewDBRequestMessage* ir *NewDBResponseMessage*), kurios atitinkamai paveldės bazinę *DBRequestMessage* užklausos (angl.: request) duomenų bazės tipo žinutę ir *DBResponseMessage* atsakymo (angl.: response) duomenų bazės tipo žinutę. Šios žinutės saugo bazinius duomenis skirtus operacijoms su duomenų baze. Dar yra sukuriamas duomenų bazės lentelę atspindintis *Domain* objektas, kur kiekvienas duomenų bazės stulpelis yra šios klasės viešasis kintamasis ir jis saugos reikšmę esančią duomenų bazėje. Kiekvienai duomenų bazės lentelei reikia sukurti XML tipo *nHibernate* aprašą.

Atliekamas duomenų ištraukimo iš duomenų bazės eksperimentas

Sukuriami duomenų bazės operacijos ir žinučių nauji objektai. Užklausos tipo žinutėje perduodami norimų ištraukti duomenų paieškos parametrai, kurie yra aprašyti vartotojo dokumentacijoje. Pavyzdžiui:

```
NewDBOperation operation = new NewDBOperation ();
NewDBRequestMessage requestMessage = new NewDBRequestMessage();
NewDBResponseMessage responseMessage = new NewDBResponseMessage();

requestMessage.SearchCriteria.Add(new string[] { "LanguageAttribute=>LanguageType.Name",
"like" }, new object[] { "form" });
requestMessage.OrderCriteria.Add("Value", "desc");
responseMessage=operation.OperateCore(requestMessage);
```

Duomenų bazės operacijose yra sukurtas funkcionalumas, kurio pagalba programuotojui nereikia naudoti *nHibernate* operacijų, o tik užpildyti suprogramuotą paieškos masyvą *SearchCriteria* ir rikiavimo masyvą *OrderCriteria*.

Rezultatas

Programuotojas nemokėdamas *nHibernate* funkcionalumo gali kurti sistemas paremtas šia technologija, jam tik reikia užpildyti *string* tipo paieškos ir rikiavimo masyvus, XML duomenų bazės lentelės aprašą, kurių užpildymas yra nesudėtingas. Taip pat programuotojas šiuos paieškos masyvus gali nesudėtingai generuoti programiškai, taip dar labiau praplečiamas busimos sistemos funkcionalumas.

Karkaso duomenų bazės operacijos suprogramavimas ir paprastos SQL užklausos parašymas, programos susiejimas su duomenų baze ir įvykdymas, užima maždaug tiek pat laiko. Tačiau karkasas suteikia daug papildomo funkcionalumo ir paieškos galimybių, o SQL užklausą perrašinėti iš naujo.

5.1.2. Daugiakalbystės panaudojimo eksperimentas

Daugiakalbystės operacijos yra skirtos programinės įrangos kalbos nustatymams atlikti. Įdiegus karkasą į jau sukurta ar į tik pradėtą kurti programinę įrangą galima nesudėtingai pakeisti jos vartotojo sąsajos kalbą. Daugiakalbystė yra suderinta su visais *System.Windows.Forms* ir *Syncfusion.Windows.Forms* bibliotekų vartotojo sąsajos komponentais. Tačiau diegiant daugiakalbystę į jau sukurtą programą teks pakeisti išmetamų pranešimų kvietimą, kadangi neįmanoma aptikti šių pranešimų automatinėmis priemonėmis bei tuo labiau juos pakeisti, nes jie dažniausiai yra įrašomi paprastu *string* tipo tekstu.

Kalba yra saugoma duomenų bazėje, kur yra saugomas vartotojo sąsajos komponento kintamojo pavadinimas ir jo tipas. Bazinę kalbą turi įvesti programuotojas naudodamasis kintamųjų pavadinimais arba bus naudojama suprogramuota kalba esanti vartotojo sąsajoje. Kitas kalbas nesudėtingai gali įvesti bet kas, taip pat galima pakeisti ir bazinę kalbą.

Atliekamas kalbos pakeitimo eksperimentas

Kaip ir kiekvienos operacijos atveju yra sukuriami: daugiakalbystės operacijos objektas, užklausos ir atsakymo tipų žinučių objektai.

```
LanguageOperation operation = new LanguageOperation();
LanguageRequestMessage requestMessage = new LanguageRequestMessage();
LanguageResponseMessage responseMessage = new LanguageResponseMessage();
Language setLanguage = new Language();
```

Užklausos žinutės tipo objektui yra perduodama visa informacija apie norimą kalbos pakeitimą: programos vartotojo sąsajos pagrindinis objektas, kalbos pavadinimas arba trumpas pavadinimas arba kalbos id. Užpildžius užklausos žinutę yra kviečiama bazinė operacijos vykdymo funkcija.

```
setLanguage.LanguageShortName = "LT";
IList<LanguageParams> LanguageParamsList = new List<LanguageParams>();
LanguageParams languageParams = new LanguageParams();
languageParams.ObjectType = Type.GetType("GUI.Form_MAIN");
languageParams.CurrentObject = vartotojoSasajosFormObj;
LanguageParamsList.Add(languageParams);
requestMessage.SetLanguage = setLanguage;
requestMessage.LanguageParamsList = LanguageParamsList;

responseMessage.ResponseInformation=operation.OperateCore(requestMessage);
```

Po šių veiksmų vartotojo sąsajos kalba yra pakeičiama į lietuvių kalbą.

Rezultatas

Programuotojuj nereikia papildomai realizuoti daugiakalbystės vartotojo sąsajoje, be to nereikia ir keisti vartotojo sąsajos programavimo stiliaus, kadangi kalba yra užpildoma

automatiškai (išskyrus išmetamus pranešimus). Šios operacijos panaudojimo principas visiškai nesiskiria nuo kitų karkaso operacijų panaudojimo, todėl programuotojuj yra lengva programuoti. Šios operacijos vykdymas proporcingai išauga nuo vartotojo sąsajos komponentų kiekio.

5.2. Patobulinimų eksperimentinis tyrimas

Biznio sluoksnio karkase papildomai buvo realizuoti ir eksperimentuoti šie patobulinimai: operacijų ir duomenų versijavimas. Operacijų versijavimas yra atliktas kaip bazinės operacijos praplėtimas ir gali būti naudojamas kaip papildomas funkcionalumas. Duomenų versijavimas patobulinimas yra atliktas kaip bazinės duomenų operacijos praplėtimas.

Operacijų versijavimas

Šis funkcionalumas gali būti vykdomas vykdant bet kurią operaciją. Prieš vykdant operaciją galima pasirinkti operacijos versiją pagal datą, versijos numerį. Jei programiškai nurodyti abu parametrai pirmumą turi parametras *version*, jei nevienas nenustatytas tada vykdoma paskutinės versijos operacija. Pavyzdžiui:

```
requestMessage.version = "2";
requestMessage.versionDate = Convert.ToDateTime("03/05/2010");
```

Programuojant operacijų skirtingas versijas tiesiog yra sukuriamos konstantos su versijos numeriu ir operacijos pakeitimo data. Pagal šiuos parametrus karkasas supranta, kurią operacijos versiją reikia vykdyti.

Duomenų versijavimas

Šis funkcionalumas yra vykdomas, jeigu yra vykdoma atnaujinimo duomenų operacija arba duomenų traukimo operacija iš duomenų bazės. Taip pat turi būti nustatyta, kad yra įjungtas duomenų versijavimas t.y. užklausos žinutėje nustatytas atributas (pagal nutylėjimą šio atributo reikšmė *false*):

```
requestMessage.versification = true;
```

Kai versijavimas įjungtas, vykdant duomenų atnaujinimo operaciją (angl.: *update*) atnaujinant yra sukuriamas naujas įrašas neištrinant senojo. Papildomai istorijos lentelėje yra užsaugojama buvusio įrašo id, lentelės pavadinimas, atnaujinimo data ir naujo įrašo id. Kai yra vykdoma duomenų traukimo operacija ir versijavimas yra įjungtas, tada pagal duotą užklausos datą iš istorijos lentelės yra ištraukiamas norimos versijos duomenų įrašas.

Rezultatas

Operacijų versijavimo funkcionalumo realizacija nėra sudėtinga, nes nereikia atlikti tinkamos operacijos versijos išrinkimą, ir laiko atžvilgiu jos vykdymas apčiuopiamai nepailgėja. Taigi karkaso našumui įtakos tai neturi.

Duomenų versijavimo patobulinimą reikia geriau ištestuoti ir patobulinti, nes gali įvykti duomenų konfliktas, jeigu versijavimas programinėje įrangoje yra išjunginėjamas ir įjungiamas tai pačiai duomenų lentelei. Taip pat naudojant duomenų versijavimą ilgą laiko tarpą gali stipriai sulėtėti operacijos vykdymo laikas, kadangi papildomai atliekamos paieškos laikas tolygiai ilgėja.

5.3. Karkaso našumo tyrimas

Bendro kuriamoms programinės įrangos biznio sluoksnio karkasas turi vieną didžiausią trukumą – jo veikimo našumas. Bet kokia bendram funkcionalumui skirta realizacija niekada nebus našesnė negu konkrečiai užduočiai atlikti pritaikyta priemonė. Dėl to yra keliamas klausimas ar tokia priemonė gali būti naudojama tam tikriems uždaviniams spręsti.

5.3.1. MS .NET reflection bibliotekos našumas

Daugiakalbystei realizuoti yra naudojama MS .NET reflection biblioteka, kurios vykdymo našumas nėra didelis. Ši biblioteka realizacijai pasirinkta todėl, kad su ja galima įgyvendinti daugiakalbystę bet kokiai sistemai, net ir nepritaikytai naudoti skirtingas kalbas. Naudojant šią biblioteką nėra privaloma keisti vartotojo sąsajos programavimo stiliaus, kadangi su MS .NET reflection biblioteka galima pagal vartotojo sąsajos pagrindinį objektą nustatyti tam tikrų komponentų vardus ir juos pakeisti.

Rezultatas

Visa daugiakalbystės operacija remiasi vartotojo sąsajos komponentų tipų aptikimu ir jų teksto pakeitimu. Naudojant MS. NET reflection biblioteką paprastam kintamojo reikšmės priskyrimui yra sugaištama 4 kartus daugiau laiko, negu kintamajam reikšmę priskirti tiesiogiai. Be to dar pati daugiakalbystės operacija atlieka duomenų ištraukimą iš duomenų bazės ir išrinkimą kuriam komponentui kurį vardą priskirti. Taigi šios operacijos vykdymas užima daug laiko ir auga su didėjančiu vartotojo sąsajos komponentų kiekiu. Realizavus

eksperimentinę programinę įrangą karkaso funkcionalumui pavaizduoti šios operacijos vykdymo laikas nėra pastebimas vartotojui, nes sistema nėra didelė.

Reikia paminėti, kad ši operacija nėra dažnai vykdoma ir dažniausiai vykdoma paleidžiant programinę įrangą. Taigi sistemos vartotojui tai neturėtų sukelti didelių nepatogumų.

5.3.2. Perteklinis kodas

Perteklinio kodo naudojamas yra viena didžiausių problemų šiuolaikinėse sistemose, nes dažniausiai sistemų kūrėjai greito sprendimo beieškant nukopijuoja tam tikro kodo dalį iš kitos sistemos vietos taip tik padidindami klaidų riziką ateityje. Karkasas yra maksimaliai pritaikytas pakartotiniam kodo panaudojimui, tačiau sistemų kūrėjas turi gerai suprasti jo veikimo principus. Taip pat karkasas vykdo įvairias funkcijas, kurios galbūt nėra reikalingos konkrečios sistemos realizacijai ir į tai taip pat turi būti atsižvelgta, nes tai papildomai gali prailginti vykdymo laiką.

Rezultatas

Karkase vykdomi pertekliniai veiksmai, kurių konkrečiam uždaviniui realizuoti galbūt nėra reikalingi ir tai prailgina užduoties vykdymo laiką. Tačiau karkasą programuotojai gali ir patys redaguoti ir ištrinti tas dalis, kurios jų manymu nėra reikalingos, pavyzdžiui duomenų registravimą arba versijavimą, jei šie funkcionalumai nėra reikalingi.

5.3.3. Užklausų generatorių našumas

Visuose šiuolaikinėse kuriamuose programinėse įrangose, kuriuose vienokiu ar kitokiu būdu yra kaupiami duomenys, yra dažniausiai naudojama duomenų bazė. Susiejimui su duomenų baze ir SQL užklausų generavimui buvo pasirinktas *nHibernate* komponentas.

nHibernate naudojimo pagrindimo eksperimentas

Šios technologijos pagrindinės pasirinkimo priežastys:

- Dažniausiai naudojama technologija objektiškam duomenų atvaizdavimui pasaulyje.
- Didelės užklausų generavimo galimybės.
- SQL užklausos vykdomos tik tada, kada to reikalauja sistema.

- Automatiškai sujungia kelias *insert/update/delete* užklausas į vieną jei jos vykdomos vienu metu ir yra to pačio duomenų objekto. Taip pat niekada nėra vykdoma *insert/update/delete* užklausa nemodifikuoto objekto.
- Užklausoms realizuoti naudojamas *outer join* – efektyviausiai vykdomas duomenų bazių valdymo sistemose.
- Palaiko *many-to-many* lentelių ryšį, *Linq to SQL* to nepalaiko.
- Užklausų generavimo greitis yra panašus į kitų objektiškai orientuotų užklausų generavimo technologijų.
- Generuojamos mažo sudėtingumo užklausos.

Pateiksiu pavyzdį sugeneruotos paprastos užklausos su *Linq to SQL* ir su *nHibernate* technologijomis. Užklausos generavimui yra pasirinkta paprasta ir dažnai pasitaikanti užklausa su *like* atributu. (`LanguageName like '%g%'`)

Linq to SQL sugeneruota SQL užklausa:

```
SELECT [t0].[languageid], [t0].[languagename], [t0].[languageshortname]
FROM [dbo].[Language] AS [t0]
WHERE (
    (CASE
        WHEN (DATALENGTH([t0].[languagename])/ 2) = 0 THEN 0
        ELSE CHARINDEX('g', [t0].[languagename]) - 1
     END)) > 0
```

nHibernate sugeneruota SQL užklausa:

```
SELECT this_.languageid as y0_, this_.languagename as y1_, this_.languageshortname as
y2_
FROM language AS this_
where this_.languagename like '%g%'
```

Taigi pagal pasirinktą konkretų ir dažnai pasikartojantį pavyzdį *Linq to SQL* sugeneruotos užklausos sudėtingumas yra aukštesnis negu *nHibernate*.

Rezultatas

Pasirinktas populiariausias ir dažniausiai naudojamas atviro kodo SQL užklausų *nHibernate* generatorius. Žinoma generatoriaus naudojimas kenkia sistemos greitaveikai, tačiau generatoriai suteikia papildomą funkcionalumą ir pakartotinio panaudojimo galimybę. Tačiau jeigu kuriama sistema skirta vienai duomenų bazių valdymo sistemai ir yra svarbus greičio faktorius tada geriau nenaudoti jokiu SQL užklausų generatorių.

6. IŠVADOS

Magistro darbo nagrinėjama probleminė sritis – programų biznio sluoksnio architektūros projektavimas. Atlikus literatūros analizę ir studijų metu sukurto biznio sluoksnio karkaso tyrimą buvo padarytos šios pagrindinės išvados:

- Karkaso architektūros realizacijai naudoti daugiasluoksnės architektūros modelį apjungiamą su Modelis-Vaizdas-Valdymas šablonu ir išskaidyti programinę įrangą į duomenų sluoksnį, biznio logikos sluoksnį ir vartotojo sąsajos sluoksnį;
- Naudoti fasado projektavimo šabloną dėl daugiasluoksnės architektūros palaikymo galimybės ir šabloninio metodo projektavimo šabloną dėl operacijų pakartotinio panaudojimo galimybės;
- Funkcionalumą realizuoti bendrą struktūrą turinčių operacijų pagrindu, kur kiekviena operacija yra pamatinė operacija arba kitų operacijų sąjunga;
- Kuo dažniau naudoti pakartotinį operacijų panaudojimą;
- Naudoti testavimu paremtą programavimą – sukūrus naują operaciją jai sukurti visapusišką automatinį testą – toks programavimo būdas užtikrina sistemos kokybę;
- Naudoti tik patikimus ir toliau palaikomus trečiųjų šalių komponentus.

Šiame darbe yra išanalizuota kaip sukurti ir kaupti struktūrizuotą programinį kodą bei pritaikyti jį realizuojant įvairius projektus.

7. LITERATŪRA

[1] M. Lynch. Features of parametric programming. [interaktyvus] 1991, Gegužė [Žiūrėta: 2010-04-23]. Prieiga per internetą: < http://findarticles.com/p/articles/mi_m3101/is_n12_v63/ai_10691390/ >.

[2] J. Woodger, MDA. Multi-Tier Architectures. [interaktyvus], [Žiūrėta: 2010-04-23]. Prieiga per internetą: < http://www.woodger.ca/archmult.htm >.

[3] S. Burbeck, Ph.D. How to use Model-View-Controller (MVC). [interaktyvus], [Žiūrėta: 2010-04-23]. Prieiga per internetą: < http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html >.

[4] E. Gamma; R. Helm; R. Johnson; J. Vlissides; *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, 1997. 144p. 208p. 306p. ISBN 0-201-63361-2.

[5] A. Kramek. Design Patterns – The Facade. [interaktyvus] 2007, [Žiūrėta: 2010-05-04]. Prieiga per internetą: < http://weblogs.foxite.com/andykramek/archive/2007/01/21/3165.aspx >.

[6] Software Engineering Standards Committee. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. [interaktyvus] 2000, [Žiūrėta: 2010-05-08]. Prieiga per internetą: < http://www.groeiplatformgea.nl/includes/img.asp/id,68/Artikelen >.

[7] P. Botella; X. Burgués; J.P. Carvallo; X. Franch; G. Grau; J. Marco; C. Quer. ISO/IEC 9126 in practice: what do we need to know? [interaktyvus] 2004, [Žiūrėta: 2010-05-08]. Prieiga per internetą: < http://www.essi.upc.edu/~webgessi/publicacions/SMEF'04-ISO-QualityModels.pdf >.

[8] H. Gilbert Yale University. .NET Framework. [interaktyvus], [Žiūrėta: 2010-05-13]. Prieiga per internetą: < http://www.yale.edu/tp/framework.htm >.

[9] .NET Framework Technologies. [interaktyvus], [Žiūrėta: 2010-05-13]. Prieiga per internetą: < http://msdn.microsoft.com/en-us/netframework/bb513602.aspx >.

[10] The C# Language. [interaktyvus], [Žiūrėta: 2010-05-13]. Prieiga per internetą: < http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx >.

[11] Then J2EE Tutorial. [interaktyvus], [Žiūrėta: 2010-05-15]. Prieiga per internetą: < http://java.sun.com/j2ee/1.4/docs/tutorial/doc/J2EETutorial.pdf >.

[12] OS Platform Statistics. [interaktyvus], [Žiūrėta: 2010-05-15]. Prieiga per internetą: < http://www.w3schools.com/browsers/browsers_os.asp >.

[13] NHibernate Reference Documentation. [interaktyvus], [Žiūrėta: 2010-05-15]. Prieiga per internetą: < http://users.telenet.be/christianr/a_nodig_samen/VBNet/NHibernate%20Reference.pdf >.

[14] Don Box; Anders Hejlsberg. LINQ: .NET Language-Integrated Query. [interaktyvus], [Žiūrėta: 2010-05-20]. Prieiga per internetą: < http://msdn.microsoft.com/en-us/library/bb308959.aspx >.

[15] Nauman Leghari. Using log4net. [interaktyvus] 2003, [Žiūrėta: 2010-05-15]. Prieiga per internetą: < http://www.ondotnet.com/pub/a/dotnet/2003/06/16/log4net.html >.

[16] Overview of nServiceBus. [interaktyvus], [Žiūrėta: 2010-05-16]. Prieiga per internetą: < http://www.nservicebus.com/Overview.aspx >.

[17] Nunit. [interaktyvus], [Žiūrėta: 2010-05-16]. Prieiga per internetą: < http://www.nunit.org/index.php >.

[18] Rhino Mocks. [interaktyvus], [Žiūrėta: 2010-05-16]. Prieiga per internetą: < http://ayende.com/projects/rhino-mocks.aspx >.

[19] Introducing Castle - Part I. [interaktyvus], [Žiūrėta: 2010-05-16]. Prieiga per internetą: < http://www.codeproject.com/KB/architecture/introducingcastle.aspx >.

8. SUMMARY

Program business layer framework development and usage research

Summary

Nowadays, programming has the high requirements, because customers demand high-quality software as soon as possible. Limited creation time can lead to lower software quality and mistakes possibility. Fast development of high-quality systems is the main problem in programming companies.

The main goal of this work is to create program business layer framework and explore how to simplify and define software architecture framework and how to connect it to database. Also identify the most frequently jobs developing the new software.

The main objectives are:

- Software architecture simplifying and developing a new system based on operations;
- Framework architecture linking with the database in compliance with operations;
- Third-party software components suitability for developing software framework, whose functionality is used on the basis of operations.

Systems developers spend more time on the final functionality of the system, rather than programmatic completion or system architecture. As a result, this strongly affects the system architecture and future improvements of the created system can be cumbersome and costly. A good program framework allows developers and system designers spend more time concentrating on the main subjects of system rather then creating the program architecture.

9. TERMINŲ IR SANTRUMPŲ ŽODYNAS

Terminas/santrumpa	Paaiškinimas
PĮ	Programinė įranga.
XML	Duomenų aprašymo kalba (angl.: eXtensible Markup Language).
SQL	Struktūrizuota užklausų kalba (angl.: Structured Query Language).
DBVS	Duomenų bazės valdymo sistema.
IEEE, ISO	Standartus kuriančios tarptautinės organizacijos.
MS	Microsoft akronimas.
C++, JAVA, Delphi	objektinės programavimo kalbos.
C#, J#	.NET framework objektinės programavimo kalbos.
PHP	Dinaminė interpretuojama programavimo kalba.
ASP.NET	MS .NET framework programavimo kalba pritaikyta svetainių kūrimui.
ADO.NET	sudėtine .NET framework dalis, kuri skirta darbui su duomenų bazės objektais.
ORM	Objektinis duomenų atvaizdavimas (angl.: Object-relational mapping).
CLR	Sudėtine Microsoft .NET framework dalis, kuri vykdo programos kodą ir valdo servisus (angl.: Common Language Runtime)
JDBC	Prisijungimo prie duomenų bazės JAVA komponentas (angl.: Java Database Connectivity)
J2EE	Java 2 programavimo technologija (angl.: Java 2 Platform, Enterprise Edition)