KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PRAKTINĖS INFORMATIKOS KATEDRA

Artūras Katutis

Ilgalaikės nuomos prekybos sistemos priežiūros modeliai

Magistro darbas

Darbo vadovas
doc. dr. A. Riškus

Kaunas, 2004

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PRAKTINĖS INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. D. Rubliauskas
2004 05 25

Ilgalaikės nuomos prekybos sistemos priežiūros modeliai

Magistro darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. J. Mikelionienė
2004 05 25

Recenzentas
doc. dr. E. Karčiauskas
2004 05 25

Vadovas
doc. dr. A. Riškus
2004 05 25

Atliko
IFM-8/2 gr. stud.
Artūras Katutis
2004 05 25

Kaunas, 2004

Turinys

SUMMARY

Software engineering is defined as the application of the systemic, disciplined, quantifiable approach to the development, operation, and maintenance of software. It is the application of engineering to software. The classic life-cycle paradigm for software engineering includes: system engineering, analysis, design, code, testing, and maintenance [1].

Maintenance is needed to ensure that the system continues to satisfy user requirements. The system changes due to corrective and non-corrective maintenance. According to Martin and McClure [8], maintenance must be performed in order to:

- Correct errors;
- Correct design flaws;
- Interface with other systems;
- Make enhancements;
- Make necessary changes to the system;
- Make changes in files or databases;
- Improve the design;
- Convert programs so that different hardware, software, system features, and telecommunications facilities can be used.

Accordingly, software must evolve and be maintained.

This paper addresses the maintenance portion of PAT life-cycle. There are several improved architecture models presented witch make maintenance and evolution of PAT easier. Among the presented architecture models there is one model (with five layers) witch was adapted to the system. The new system's efficiency was compared with the old one's.

This paper also contains a new architecture model witch could be adapted to the new systems. The main purpose of this architecture is to separate graphical user interface from system's business logics and make software maintenance very easy in the future.

1 ĮVADAS

Sparčiai populiarėjant nuomos (lizingo) kompanijų paslaugom, įmonėms, teikiančioms tokias paslaugas, atsirado poreikis įsigyti programines įrangas kurios, būtų pritaikytos prie jų verslo plano. Tačiau kiekvienoje įmonėje darbo tvarka ir stilius skiriasi. Tad reikalavimai programinei įrangai irgi skirtingi. Daugelis įmonių jau turi programines įrangas, į kurias norėtų integruoti naujas sistemas. Masiniam vartotojui skirtos programinės įrangos tokioms įmonėms gali tikti tik iš dalies arba iš viso netikti dėl nepakankamo programinės įrangos universalumo. Tokiu atveju galima būtų pasisamdyti įmonę, kuri sukurtų ir įdiegtų naują programinę įrangą.

Turint programinę įrangą iškyla jos priežiūros problema. Kas programinę įrangą prižiūrės ir kaip priežiūros darbai bus atliekami? Progreaminės įrangos (PĮ) priežiūra yra nedaloma dalis jos gyvavimo ciklo. Tačiau neretai jai skiriame mažiau dėmesio negu kitoms PĮ kūrimo fazėms. Istoriškai daugelyje organizacijų didžiausias dėmesys buvo skiriamas projektavimui. Dabar situacija keičiasi, kadangi organizacijos stengiasi surasti būdus, kaip geriausiai kompensuoti projektavimo išlaidas išlaikant programas eksploatacijoje kiek galima ilgiau. Priežiūra yra taip pat brangi. Dėl tos priežasties atsiranda galimybė toliau plėtoti mokslinius tyrimus, kad būtų padidintas priežiūros našumas [15].

Šio darbo pagrindinis tikslas yra pateikti programinės įrangos (PĮ) priežiūros modelius kurie garantuotų sėkmingą PĮ PAT (Proceed Asset Trading) priežiūrą ir tobulinimą. Iš to seka kiti darbo uždaviniai:

a) suprojektuoti ir suprogramuoti „Ilgalaikės nuomos prekybos sistemą“;

c) pasinaudojant praktinės sistemos kūrimo metu sukauptomis žiniomis ir identifikuoti pagrindinius sistemos trūkumus, kurie apsunkina tolimesnę PĮ priežiūrą ir tobulinimą;

d) sukurti programinės įrangos programavimo inžinerijos metodus, kurie palengvintų pastarojo modelio taikymą būsimiems projektams;

f) įvertinti siūlomų modelių taikymo efektyvumą ir sprendimų pagrįstumą.

2 ILGALAIKĖS NUOMOS PREKYBOS SISTEMOS PRIEŽIŪROS MODELIAI

2.1 Palaikymo koncepcija

Programinės įrangos priežiūra yra apibrėžiama kaip programinės įrangos produkto modifikavimas po jo pristatymo tam, kad būtų ištaisytos aptiktos klaidos, pagerintas našumas arba pritaikytas prie pasikeitusios aplinkos [2]. Programinės įrangos prižiūrėtojas yra organizacija, kuri atlieka priežiūros darbus [3].

Dažniausiai galvojama, kad priežiūra yra tik aptiktų klaidų taisymas. Tačiau ilgametė patirtis parodė, kad didžioji dalis, daugiau kaip 80 proc., priežiūros darbų yra ne klaidų taisymas [4], [5], [6].

Programinės įrangos kūrimo metu dėmesys sutelkiamas į kodo generavimą, kuris įgyvendina nurodytus reikalavimus ir veikia teisingai. Tačiau programinės įrangos priežiūra yra visai kas kita [7]. Prižiūrėtojai turi žvelgti atgal į sukurtus produktus, tuo pačiu metu žiūrėti į dabartį dirbant su vartotojais ir operatoriais. Prižiūrėtojai, taip pat bando nuspėti problemas ir apgalvoti funkcinius pasikeitimus. Taigi versijų valdymas yra svarbus priežiūros ir tobulinimo aspektas [15].

2.2 Priežiūros poreikis

Priežiūra užtikrina, kad sistema toliau tenkins vartotojo reikalavimus. Pasak Martin ir McClure [8], priežiūra turi būti atliekama tam, kad:

- Ištaisyti klaidas;
- Ištaisyti projektavimo srautus;
- Sujungti su kitomis sistemomis;
- Atlikti patobulinimus;
- Atlikti reikalingus sistemos pakeitimus;
- Atlikti pakeitimus bylose ar duomenų bazėse;
- Patobulinti dizainą;
- Pakeisti programas taip, kad skirtinga aparatūra, programinė įranga, sistemos savybės ir telekomunikacinės priemonės galėtų būti panaudotos.

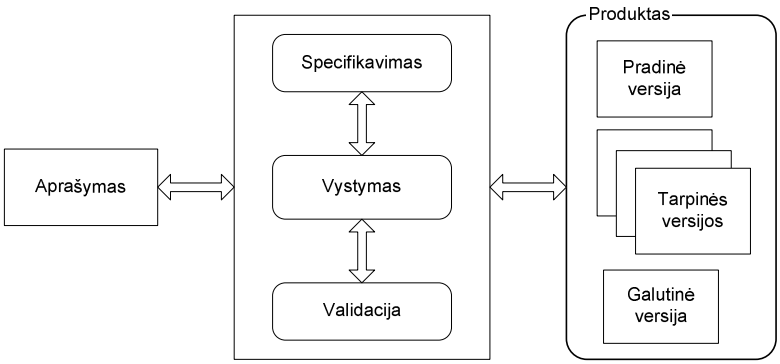
Pagrindiniai keturi priežiūros aspektai yra:

- Kasdienė sistemos funkcijų kontrolė;
- Priežiūros valdymas modifikuojant sistemą;
- Priimtinų egzistuojančių funkcijų gerinimas;
- Neleisti sumažėti sistemos našumui iki nepriimtino laipsnio.

2.3 Detalus praktinis uždavinys

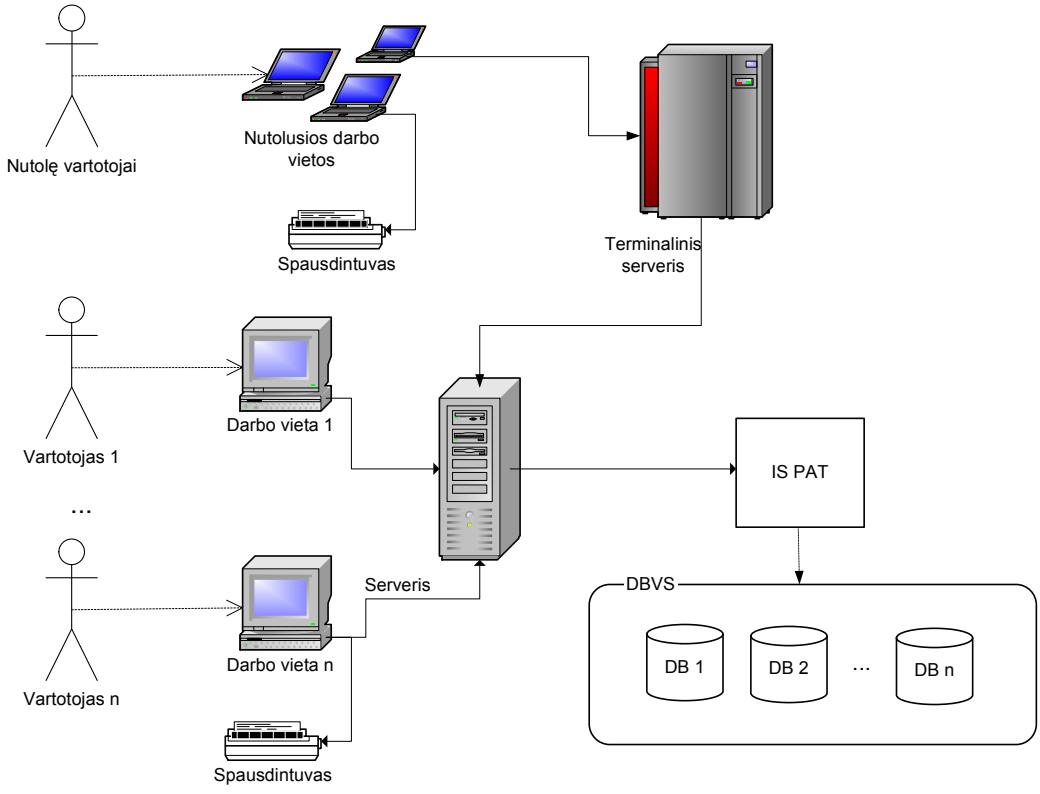
Reikia patobulinti PAT architektūrą taip, kad jos priežiūros metu būtų sunaudota kuo mažiau resursų ir laiko. Tuo pačiu suteikti kuo daugiau universalumo naujai architektūrai, kad ją būtų galima taikyti naujai kuriamoms sistemoms. Taip pat sistemoms, kurios bus integruotos i PAT sistemą.

PAT sistema buvo kuriama pagal evoliucinį modelį (žr. 1 pav.). Produkto kūrime dalyvavo pats užsakovas. Taip buvo išaiškinti visi nesusipratimai tarp užsakovo ir kūrėjo. Užsakovas gavo kuriamos programinės įrangos prototipą jau projekto gyvavimo ciklo pradžioje.



1 pav. Evoliucinis projektavimo modelis

Informacinė sistema PAT įdiegta serveryje (žr. 2 pav.). Kiekviena darbo vieta turi būti prijungta prie vietinio tinklo, prie kurio yra prijungtas ir pats serveris. Darbo vietos neprijungtos prie vietinio tinklo (nutolusios darbo vietos) turės jungtis prie sistemos naudojantis terminaliniu serveriu.



2 pav. IS PAT diegimo diagrama

PAT sistemai buvo iškelti šie pagrindiniai reikalavimai:

a) Autorizacijos sistema – funkcijos turi remtis vartotojų rolėmis, korektiškai tiekti sistemos galimybes vartotojams;
b) Objektų registravimas, redagavimas ir paieška turi būti greita ir patogi;
c) Programa turi funkcionuoti keliomis kalbomis;
d) Programinė įranga turi minimaliai apkrauti serverį;
e) Sistema turi efektyviai naudoti resursus ir atlaisvinti juos darbo pabaigoje;
f) Turi būti numatytas sistemos palaikymas ir priežiūra.

2.4 Galimi sprendimai

Tinkamai suprojektavus sistemos architektūrą, išvengiame nemažai sistemos priežiūros ir tobulinimo problemų. Dažniausiai, projektuojant sistemas, į tai neatkreipiamas dėmesys. Galimi architektūros pakeitimai nagrinėjami skyriuje „Naujos sistemos architektūros palengvinančios priežiūrą bei tobulinimą“ (žr. 25 psl.).

Kiti galimi sprendimai nagrinėja išorinių sistemų taikymus. Kiekvienas siūlomas sprendimas susideda iš mažesnių pasirinkimų: kokią pasirinksime sistemos architektūros projektavimo priemonę, išeities tekstų versijų kontrolės sistemą ir kokia bus naudojama duomenų bazių valdymo sistema.

2.4.1 Sistemos architektūros projektavimo priemonės

Renkantis sistemos projektavimo priemonę, pagrindinis dėmesys buvo skiriamos produkto kainai bei .NET C# palaikymui, kadangi sistema bus realizuojama būtent šia programavimo kalba. Sužinojus Rational Rose paketo kainą, šis produktas nebuvo toliau nagrinėjamas. Tad Analizei pasirinkti du projektavimo paketai: **Microsoft® Visio®** ir **Poseidon for UML.** Kiekvienos šių priemonių UML diagramos buvo palygintos, įvertintas atvirkštinės inžinerijos palaikymo lygis bei dokumentacijos generavimo galimybės.

Microsoft® Visio®

Microsoft® Visio® yra JAV kompanijos Microsoft produktas. Visio – tai ne vien UML projektavimui skirta priemonė, bet diagraminio vaizdavimo sprendimas, skirtas tiek technikos specialistams, tiek ir verslininkams.

Oficialioje programos Visio Interneto svetainėje jau pristatoma Visio 2003 versija, tačiau Lietuvos įmonės vis dar siūlo Visio 2002 versiją.

Skiriamos dvi Visio 2003 paketo redakcijos (*editions*): Microsoft® Visio® Standard 2003 ir Microsoft® Visio® Professional 2003. Lentelėje 1 pateikiamas šių redakcijų palyginimas.

1 lentelė. Visio Standart 2003 ir Visio Professional 2003 palyginimas

Savybės	Visio Standart 2003	Office Visio Professional 2003
Prieinama 17 kalbų	+	+
Blokinės (block) diagramos	+	+
Sąmonės srauto (brainstorming) diagramos	+	+
Statybų planai		+
Verslo procesų diagramos	+	+
DB schemos su atvirkštine inžinerija		+
Žemėlapiai	+	+
Diagramų eksportas į labiausiai paplitusius grafinius formatus	+	+
XML Web servisų integracija	+	+
Duomenų srautų (flowchart) diagramos	+	+
Microsoft .Net palaikymas	+	+
Microsoft Visual Basic palaikymas	+	+
Organizacijos modelis	+	+
Diagramų generavimas su atvirkštine inžinerija		+

Lentelėje 2 pateikiamos Visio kainos JAV ir Lietuvos vartotojams. JAV kainos buvo paimtos iš oficialios Visio Interneto svetainės, o Lietuvos – iš UAB “Sonex” kainininko.

2 lentelė. Microsoft® Visio® kainos

Versija	Kaina JAV vartotojui	Kaina Lietuvos vartotojui
Microsoft® Visio® Standard 2003	199 USD	798,23 Lt.
Microsoft® Visio® Professional 2003	499 USD	2007,39 Lt.

Norint išnaudoti visas programos Visio teikiamas galimybes, reikia, kad darbo vieta atitiktų tokius reikalavimus:

- **Operacinė sistema:** Windows 2000 arba Windows XP
- **Procesorius:** Pentium 233 MHz, rekomenduojama Pentium III
- **Operatyvinė atmintis (RAM):** 128 MB arba daugiau
- **Kietasis diskas:**
  - **Visio Standard 2003.** 160 MB laisvos vietos, įskaitant 75 MB, kuriuos užima instaliuota operacinė sistema.
  - **Visio Professional 2003.** 210 MB laisvos vietos, įskaitant 75 MB, kuriuos užima instaliuota operacinė sistema.
- **Monitorius:** Super VGA (800X600) arba aukštesnės raiškos.

Poseidon for UML

Poseidon for UML produktas sukurta vokiečių įmonės Gentleware. Ja naudojasi apie 400000 vartotojų įvairiose Europos, Šiaurės Amerikos bei Azijos šalyse. Poseidonas buvo sukurtas remiantis atviro kodo programa ArgoUML. Šiuo metu pristatoma Poseidon 2.0 versija įvairiomis redakcijomis: Community Edition 2.0, Standard Edition 2.0, Professional Edition 2.0 ir Embedded Edition 2.0. Pagal oficialioje šios programos Interneto svetainėje pateikiamą informaciją, netrukus turi pasirodyti Enterprise Edition, kuri palaikys darbą komandoje, versijų kontrolę, kliento-serverio architektūrą. Lentelėje 3 pateikiamas trumpas šių redakcijų palyginimas.

3 lentelė. Poseidon for UML redakcijų palyginimas

Savybės	Community Edition 2.0	Standard Edition 2.0	Professional Edition 2.0	Embedded Edition 2.0
Vandens ženklas atliekant grafinį eksportą	+			
Paprasta instaliacija WebStart	+			
Nepriklausomas nuo platformos	+	+	+	+
Visos 9 UML diagramos	+	+	+	+
XMI palaikymas	+	+	+	+
Eksportas į grafinius failus	+	+	+	+
Tiesioginė (forward) inžinerija Java	+	+	+	+
Atvirkštinė inžinerija Java		+	+	+
Tiesioginė inžinerija Corba IDL			+	
Tiesioginė inžinerija C#			+	
Tiesioginė inžinerija VB.net			+	

Lentelėje 4 pateikiamos aukščiau aprašytų Poseidon for UML redakcijų kainos, pateiktos oficialiame šio įrankio puslapyje.

4 lentelė. Poseidon for UML redakcijų kainos

	Community Edition 2.0	Standard Edition 2.0	Professional Edition 2.0	Embedded Edition 2.0
Kaina	EUR 0,-	EUR 199,-	EUR 699,-	EUR 1249,-

Reikalavimai darbo vietai, naudojant Poseidon for UML, nebuvo pateikti, todėl būtų galima įvardinti tik du, pastebėtus atliekant bandymus su šiuo produktu:

1) prieš įdiegiant Poseidon for UML, kompiuteryje turi būti įdiegta virtuali Java mašina;
2) ši programa reikalauja nemažai RAM (turint tik 64 MB su ja dirbti praktiškai neįmanoma).

Atvirkštinė inžinerija

Poseidon for UML palaiko kodo generavimą į tokias programavimo kalbas: Java, CSharp, CorbaIDL, PHP, VB.Net. Taip pat sugeneruoja UML dokumentą iš nubraižytų diagramų. Kalbų sąrašą galima papildyti, pavyzdžiui, C++ arba XML.

Neturint diagramų, jas galima gauti, jei turime prieinamus Java kalba parašytus failus, o taip pat jei kompiliuojant programą nebuvo rasta klaidų.

Microsoft Visio leidžia generuoti C++, Java ir Microsoft Visual Basic kodą iš visų UML diagramų, kuriose naudojamos klasės. Generuojant kodą iš diagramų gaunami šie failų tipai:

- Visual Basic - .cls, .bas, ir frm failai;
- C++ - .cpp ir .h failai;
- Java - .java failai;
- C# - .cs failai.

Kaip ir kodo generavimas, diagramos gali būti gaunamos iš C++, C#, Visual Basic ir Java programavimo kalbų. Deja, Microsoft Visio palaikomų kalbų sąrašo negalima papildyti.

Dokumentacijos generavimas

Dokumentaciją su Paseidon for UML įrankiu galima sugeneruoti pasirinkus generavimo meniu punktą bei pažymėjus, kad norime generuoti HTML formato dokumentaciją. Dokumentacijoje išsamiai pateikiami visi modeliai, jų elementai, ryšiai (pvz., klasės diagramos atributų ypatybės, operacijų ypatybės ir t.t.).

Microsoft Visio suteikia galimybę generuoti dokumentaciją veiklos, komponentų, įdiegimo, būsenų, statinės struktūros diagramoms. Šis paketas leidžia nurodyti kokią informaciją pageidaujame įtraukti į dokumentaciją (pvz., jei darome ataskaitą veiklos diagramai, galime nurodyti, kad aprašytų tik būsenas), taip pat nurodyti kokias UML elementų savybes reikia dokumentuoti (*constraints, documentation, tagged values*). Vartotojas gali pasirinkti dokumentacijos pateikimo būdą: peržiūrą ekrane, atspausdinimą ar eksportavimą į *rtf* failą. Dokumentacijos generavimas atliekamas pasirinkus meniu *UML – Report.*

XMI palaikymas

Microsoft Visio 2002 Professional ir Microsoft Visual Studio .NET Enterprise (Architect Visio for Enterprise Architects) versijos palaiko UML diagramų eksportavimą į XML failus. Ši funkcija yra realizuota atskirame dll faile *XMIExport.dll.* Naudojantis XMI eksportavimo moduliu galima eksportuoti informaciją iš statinės struktūros, įdiegimo ir komponentų diagramų į XML failą, kuris atitinka XMI standartus.

Paseidon for UML gali išsaugoti diagramas XMI formatu, tačiau detalesnės informacijos nepavyko rasti.

Išvados

Pliusai:

- Poseidon for UML
  - Yra nemokama Community edition versija;
  - Yra nepriklausomas nuo platformos;
  - Palaiko visas 9 UML diagramas;
  - Dokumentacijos HTML formatu generavimas;
  - Tiesioginės ir atvirkštinės inžinerijos palaikymas;
  - Diagramų eksportavimas į grafinius failus (*.gif, *.jpg ir t.t.).
- Microsoft Visio
  - Diagramų semantinė kontrolė;
  - Palaiko ne tik UML bet ir įvairius kitus modelius (organizacijos, tinklų inžinerijos, esybių ryšių diagramas ir t.t.);
  - Dokumentacijos generavimas į *.rtf tipo failus;
  - Tiesioginės ir atvirkštinės inžinerijos palaikymas;
  - Generuoja ne tik programos kodą, bet ir duomenų bazės loginę schemą;

Minusai:

- Poseidon for UML
  - Reikalauja labai daug kompiuterio resursų;
  - Dedami vandens ženklai eksportavus diagramą į grafinį failą iš Community edition versijos;
  - Nėra diagramų generavimo vienų iš kitų;
  - Nėra semantinės diagramų kontrolės;
  - Reikia įdiegti Java virtualią mašiną norint dirbti su programa.
- Microsoft Visio
  - Dirba tik operacinės sistemos Windows aplinkoje;
  - Labai brangus paketas;
  - Negeneruoja diagramų vienų iš kitų.

2.4.2 Išeities tekstų versijų kontrolės sistemos

Prieš pradedant sistemos kūrimo darbus reikėjo pasirinkti priemones, užtikrinančias išeities tekstų versijavimą. Atmetus nuosavos išeities tekstų kontrolės sistemos kūrimą, kuris yra beprasmis, nes rinkoje yra didelis tokių produktų pasirinkimas, lieka išsirinkti iš egzistuojančių geriausių pasirinkimų:

a) Concurrent Versions System 1.11 (CVS) [18]

b) Microsoft Visual SourceSafe 6.0 [19]

Abi sistemos buvo lyginamos pagal šiuos kriterijus: funkcionalumo turinys, priėjimas prie saugyklos, saugyklos dydis, IDE palaikymas, sistemos reikalavimai, priedų (add-ons) palaikymas, saugumas ir kaina.

Funkcionalumo turinys

	VSS	**CVS**
Saugo visų failų pakeitimų istoriją	Taip	Taip
Versijų išsišakojimas (branching)	Taip	Taip
Iššakotų versijų suliejimas	Taip	Taip
Galimybė keliem programuotojam taisyti tą patį failą	Taip	Taip
Galimybė papildyti priedais(add-ons)	Ne	Taip
Versijų pasikeitimų perspėjimai	Ne	Taip
Priėjimas per Internetą	Ne	Taip
Integracija su issue/bug tracking tools	Ne	Taip

Priėjimas prie saugyklos

VSS – tai nėra kliento/serverio platforma. Tai klientinė - failų sistemos platforma. Vietiniame tinkle priėjimui prie saugyklos gali būti naudojamas failų dalinimas (*file sharing*). Tačiau esant mažam duomenų pralaidumui šis būdas netinka. Tam yra naudojamas Source-off-Site programinis paketas ($200 viena licensija).

CVS – tai tikra kliento/serverio sistema. Saugyklą galima pasiekti VPN, LAN ar per internetą. Galima naudoti kelis protokolus – saugų ir nesaugų, su kompresija ar be jos. Serveriai veikia ir Windows ir Unix operacinėse sistemose.

Saugyklos dydis

VSS saugo visą failą kiekvienai versijai. Pakeitus bent vieną eilutę faile, saugykloje sukuriamas naujas failas. Saugyklos dydis kinta geometrine progresija.

CVS saugo eilučių skirtumus kiekvienai failo versijai. Tai reiškia, pridedant vieną eilutę faile, saugykla padidėja tik keliomis eilutėmis.

IDE palaikymas

VSS palaiko tik tuos IDE, kurios turi integracijas su SCC API.

CVS yra standartiškai palaikomas daugumos IDE. Toms IDE, kurios palaiko tik SCC API integraciją galima naudoti CVS priedą (*add-ons*), tuomet CVS veiks kaip VSS.

Sistemos reikalavimai

VSS – Microsoft Windows 95/98, Microsoft Windows NT 4.0 SP3 arba naujesnė.

Atmintis diske:

- Klientui: priklausomai nuo įdiegimo tipo nuo 59 Mb iki 79 Mb;
- Serveriui: tipinis įdiegimas – 128 Mb; pilna įdiegimas – 141 Mb.

Monitorius: VGA ar geresnės rezoliucijos.

CVS - Windows NT/95, OS/2, VMS bei daugelyje Unix operacinių sistemų.

Serveriui užtenka 32 Mb, tačiau turėtų veikti ir mažiau atminties turinčiame kompiuteryje.

Priedų (add-ons) palaikymas

Priedus palaiko tik CVS. Naudojant šiuos priedus CVS funkcionalumas yra labai praplečiamas. Reikalui esant, galima ir pačiam susikurti tokius priedus ir pritaikyti juos organizacijos veiklai.

Saugumas

VSS naudoja kliento/serverio protokolą pagrįstą Microsoft RPC.

CVS gali naudoti įvairius prisijungimo būdus. Tai gali būti nesaugus protokolas – pserver (gali veikti naudojant rsh – *remote shell*) ir naudojant ssh – kuris yra saugus būdas.

Kaina

VSS – pilna versija kainuoja $549. Jeigu turėjote senesnę versiją, tai jums kainuos tik $279. Daugiau informacijos galima rasti http://msdn.microsoft.com/ssafe/.

CVS – nieko nekainuoja, tai yra atviro kodo programinė įranga, kurią galima parsisiųsti iš http://www.cvshome.org/ svetainės. Čia galima parsisiųsti išeities tekstus, kuriuos reikia pačiam susikompiliuoti.

Išvados

Naudojant Microsoft sukurtą IDE patartina naudoti ir Microsoft Visual SourceSafe, kadangi tos pačios firmos sukurti produktai komunikuos tarpusavyje geriau. Priešingu atveju patartina naudoti CVS. Ji yra nemoka, pakankamai stabili, gali veikti tiek Unix tiek Windows operacinėse sistemose. CVS funkcionalumą galima papildyti CVS priedais (*add-ons*).

2.4.3 Duomenų bazių valdymo sistemos

Kadangi užsakovo organizacijoje visos įdiegtos sistemos naudoja Microsoft SQL Server 2000, tai ir naujai kuriama sistema turės naudoti šią DBVS. Todėl galimų variantų nagrinėjimas nebuvo atliktas.

2.5 Pasirinktas sprendimas

Renkantis tarp galimų sprendimų lemiamą įtaką darė situacija užsakovo organizacijoje. Reikėjo atsižvelgti į jau egzistuojančias darbo rutinas ir naudojamą programinę įrangą. Todėl buvo nuspręsta:

a) Naudoti Microsoft Visio paketą sistemos projektavimui, kadangi šis produktas yra platinamas kartu su Microsoft Visual Studio paketu ir atskirai įsigyti jo nereikia;
b) Naudoti Microsoft Visual SourceSafe paketą išeities tekstų versijų saugojimui, kadangi šis produktas taip pat yra platinamas kartu su Microsoft Visual Studio paketu. Be to šis produktas jau gana senai naudojamas užsakovo kompanijoje, todėl nereikalaus papildomo darbuotojų apmokymo;
c) Duomenim saugoti naudoti Microsoft SQL Server 2000, nes užsakovo organizacijoje yra naudojama būtent ši duomenų bazių valdymo sistemą, be to, naujai diegiama sistema yra integruojama į jau egzistuojančias užsakovo sistemas;
d) Biznio komponentų realizacijai naudoti COM+ technologiją, nes ši technologija palaiko daug savybių darbui su duomenų bazėmis, kurių viena pagrindinė yra – transakcijų palaikymas.

3 ILGALAIKĖS NUOMOS PREKYBOS SISTEMA

3.1 Apribojimai

3.1.1 Produkto reikalavimai

- Sistema yra pakankamai greita ir patogi vartotojui.
- Sistemos duomenims kaupti pasirinkta SQL Server 2000 DBVS.
- Sistemos patikimumas yra pakankamai aukštas.
- Sistemos duomenys yra lengvai perkeliama į kitą vietą. Sistemos duomenų pasikeitimo vieta neturi įtakos vartotojo darbui. Pasikeitus jai, vartotojui tereikia nurodyti vietą, į kurią sistemos duomenys buvo perkelti.
- Sistemą galimą pritaikyti kitoms, panašaus profilio, įmonėms, atlikus nedidelius pakeitimus.

3) Reikalavimai kūrimo procesui

- Kadangi vizualinis sistemos vaizdas turi didelę įtaką vartotojų įspūdžiui apie produktą bei daro įtaką darbą su sistema, projektuojant vartotojo sąsają būtina laikytis standartų. Vartotojo sąsają buvo projektuojama laikantis MS Windows grafinės sąsajos standartų [12].
- Ši sistema yra suderinama su MS Windows 2000/XP operacinėmis sistemomis.
- Programavimo kalba – MS Visual C# .NET.
- Sistemai kurti buvo naudojamas evoliucinis projektavimo modelis. Vartotojas dalyvavo visuose projektavimo etapuose.
- Yra paruošti šie dokumentai:
    1. Sistemos aprašymas;
    2. IS įdiegimo vadovas;
    3. Sistemos administravimo vadovas;
    4. Vartotojo vadovas.

3.1.2 Bendradarbiaujančios sistemos

Visi kompiuteriai sujungti į vietinį tinklą ir turi prieigą prie serverio. Serveryje yra įdiegtas DBVS MS SQL Server 2000 bei informacinė sistema PAT.

IS PAT su DBVS bendrauja naudodama COM+ komponentų.

SQL serveryje esančioje duomenų bazėje yra saugoma visa informacija. Šiuo metu sistemoje egzistuoja aibė vartotojų, suskirstytų į tam tikras grupes.

3.1.3 Komerciniai specializuoti programų paketai

Sistemos darbui gali būti panaudoti tik licencijuoti produktai. Darbo metu prieinami tokie programinių paketų resursai:

- Microsoft SQL Server 2000;
- Microsoft Visual SourceSafe;
- Microsoft .NET Visual C#;
- Microsoft Viso for Enterprise Architects;
- Microsoft Word 2000.

3.2 Sistemos architektūra

Sistema yra realizuota trisluoksne architektūra (žr. 3 pav.), kurią sudaro:

- vartotojo sąsaja;
- biznio komponentai;
- duomenų bazių valdymo sistema (DBVS).

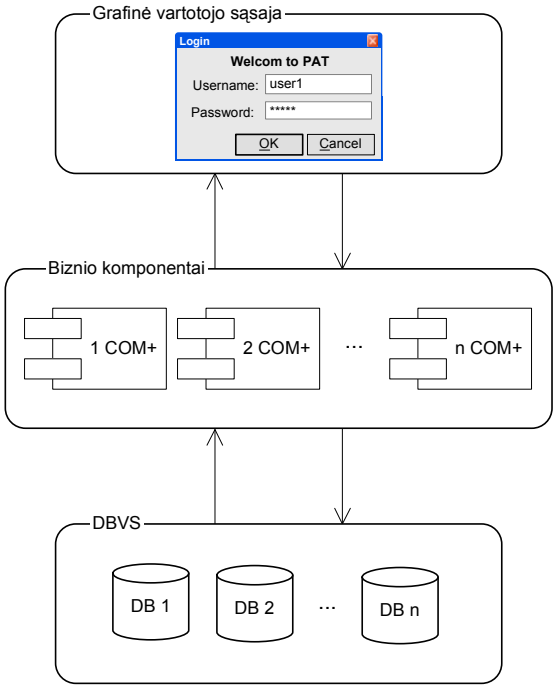
Grafinė vartotojo sąsaja bus naudojama informacijai įvesti bei išvesti. Siekiant išlaikyti korektišką architektūrą komponentas tiesiogiai neatlieka veiksmų su duomenų bazėmis. Tai atlieka tarpininkaujantieji COM+ komponentai. Naudojantis šiais komponentais vartotojas galės atlikti sistemos panaudojimo atvejų funkcionalumą[1]:

1. Sistemos vartotojo autentifikavimas;
2. Sistemos vartotojo sukūrimas;
3. Teisių priskyrimas sistemos vartotojui;
4. Naujo kliento informacijos įvedimas;
5. Kliento informacijos redagavimas;
6. Naujo objekto sukūrimas;
7. Objekto įvertinimas;
8. Objekto informacijos redagavimas.

Duomenų bazėse bus saugoma visa informacija apie įvertinamus objektus, klientus bei sistemos vartotojus ir jų teises.

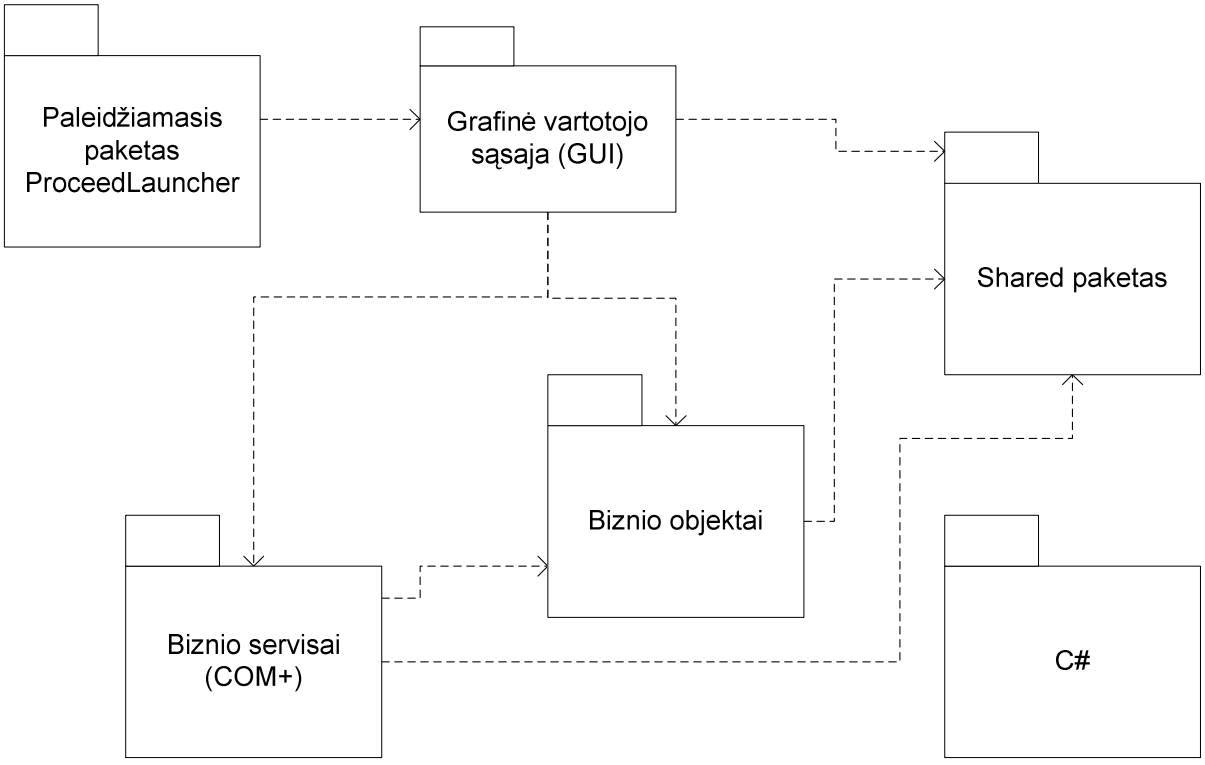
Naudojant tokią architektūrą grafinė vartotojo sąsaja yra atskiriama nuo sistemos biznio logikos. Tai padidina sistemos pakartotinį panaudojamumą.

[1] Pateikti tik pagrindiniai sistemos panaudojimo atvejai.



3 pav. Trisluoksnė IS PAT architektūra

Visi IS PAT komponentai yra sudėlioti į atskirus paketus. Bendra paketų diagrama pateikta 4 paveikslėlyje.



4 pav. Sistemos paketų diagrama

3.2.1 Architektūriniai apribojimai

Yra išskirti šie svarbūs reikalavimai ir sistemos apribojimai, kurie daro didelę įtaką sistemos architektūrai:

1. IS PAT turi būti kuriama, naudojant vieningą informacijos objektų ir jų savybių identifikavimo sistemą, kuri sudaro pagrindą duomenų bazių integravimui ir duomenų apsikeitimui.

   Naudojant deklaratyvius duomenų bazės apribojimus bei mechanizmus, užtikrinančius sudėtingą vientisumo taisyklių realizavimą (duomenų bazės procedūras, trigerius), būtina tikrinti:

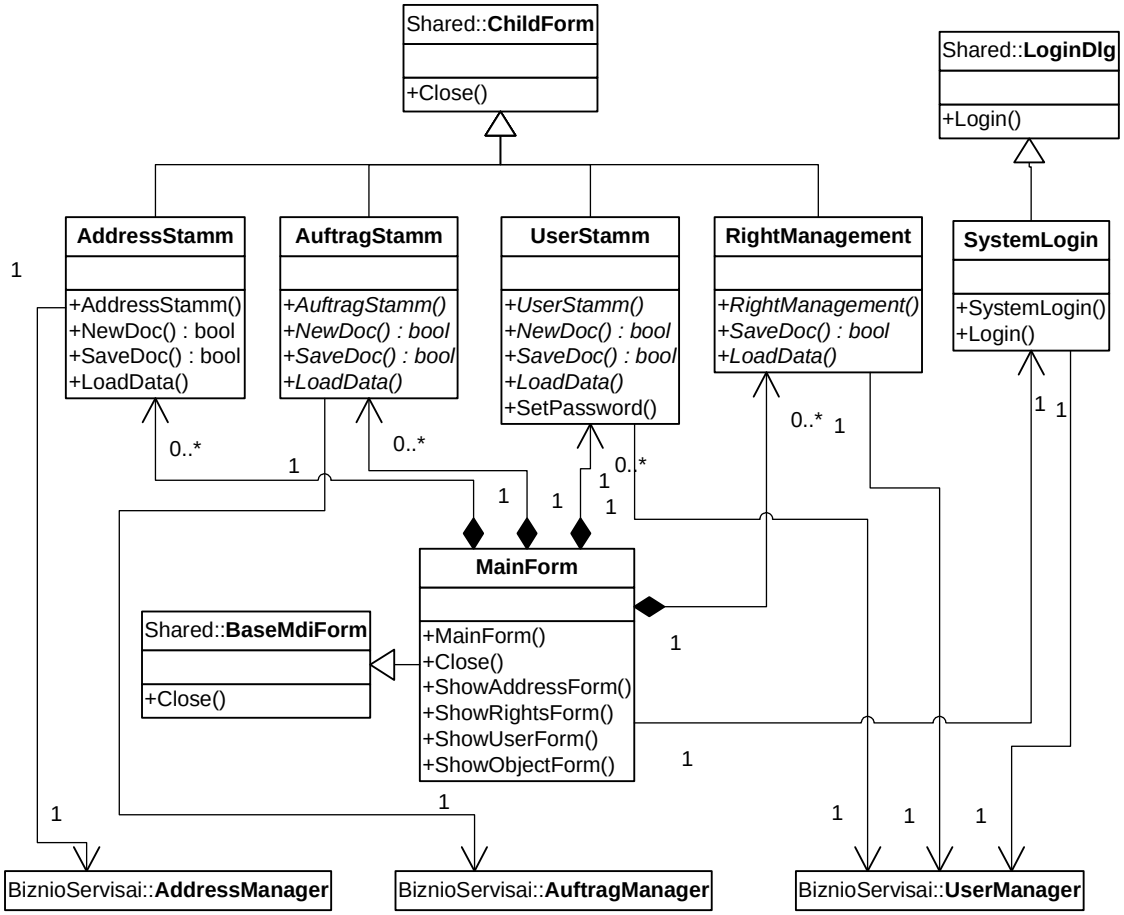
   - nuorodų tenkinimą (*foreign key*);
   - įrašų identifikuojančių laukų (*primary/unique key*) unikalumą;
   - leistinas įrašo laukų reikšmes;
   - atributų būtinumą.
2. Programine įranga galės naudotis tik tie vartotojai, kurie bus užregistruoti sistemoje ir jiems suteiktas prisijungimo vardas bei slaptažodis. Tik autentifikuotas vartotojas galės dirbti su sistema. Priklausomai nuo to, kokios teisės bus suteiktos vartotojui, jis galės atlikti vienus ar kitus veiksmus su sistema.
3. Atliekant veiksmus su duomenų baze, turi būti kuo mažiau apkraunamas tinklas, kad kuo mažiau stabdytų tinklu keliaujančius kitus duomenis.

3.3 Sistemos paketai

Kai kurių klasių, pateiktų klasių diagramose, pavadinimai yra dalinai vokiški. Taip yra, todėl, kad sistemos užsakovas yra Vokietijos įmonė, kuri pageidavo, jog klasių pavadinimai būtų suprantami ir Vokietijoje dirbantiems darbuotojams.

3.3.1 Grafinės vartotojo sąsajos paketas

Grafinės sąsajos paketas susideda iš daugelio klasių. Visos jos yra paveldėtos iš tėvinių *.NET C#* klasių. Pagrindinė paketo klasė yra *MainForm* – tai pagrindinė sąsajos forma, iš kurios galima pasiekti visas kitas formas ar dialogo langus. Jos pagrindinį funkcionalumą galima apibūdinti naudojant panaudojimo atvejų diagramą (ši diagrama buvo pateikta su vartotojo reikalavimų, bei reikalavimų specifikacijos dokumentais). Sistemos grafinės vartotojo sąsajos paketo klasių diagrama yra pateikta 5 pav. Ši diagrama yra pateikta abstrakčiame lygyje, nenorint apkrauti skaitytojo bereikalinga informacija.



5 pav. Grafinės vartotojo sąsajos paketo klasių diagrama

Shared::BaseMdiForm klasė atspindinti bazinę pagrindinę sistemos formą, kurią kituose projektuose reikia paveldėti, tam kad būtų išlaikoma visų projektų vienoda vartotojo sąsaja bei vienodas funkcionalumas. Tokio tipo langas gali turėti savyje daug kitų formų (ne MDI tipo).

Visos formos į kurias vartotojas galės įvesti informaciją yra paveldėtos iš *Shared::ChildForm* klasės.

Shared::LoginDlg klasė atspindinti bazinį, prisijungimo prie sistemos, langą, kurią kituose projektuose reikia paveldėti, tam kad būtų išlaikoma visų projektų vienoda vartotojo sąsaja bei vienodas funkcionalumas.

GUI::AddressStamm klasė yra skirta kliento duomenims kurti bei modifikuoti. Ji yra paveldėta iš *Shared::ChildForm* klasės.

GUI::AuftragsStamm klasė yra skirta objekto duomenis sukurti bei modifikuoti. Ji yra paveldėta iš *Shared::ChildForm* klasės.

GUI::UserStamm klasė yra skirta sistemos vartotojams sukurti bei modifikuoti. Ji yra paveldėta iš *Shared::ChildForm* klasės.

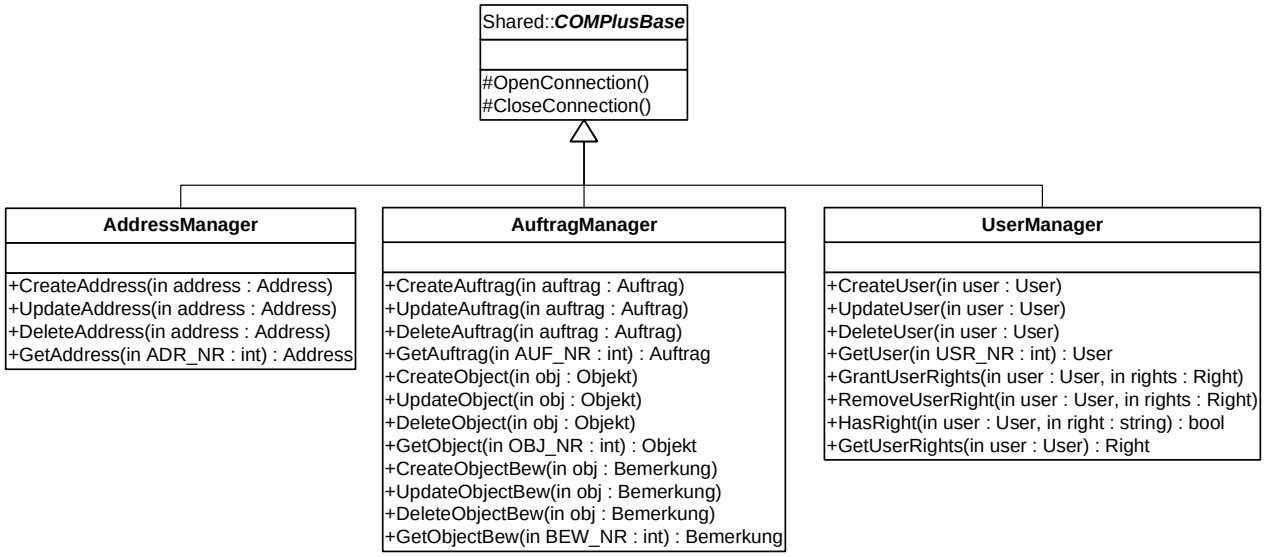
GUI::RightManagement klasė yra skirta sistemos vartotojų teisėm priskirti bei modifikuoti. Ji yra paveldėta iš *Shared::ChildForm* klasės.

GUI::SystemLogin klasė yra skirta prisijungimui prie sistemos. Klasė yra paveldėta iš Shared::LoginDlg klasės.

GUI::MainForm - tai klasė atspindinti pagrindinį sistemos langą. Šis langas yra MDI tipo, tad visi kiti sistemos langai bus atidaromi šio, pagrindinio, lango viduje. Šis langas turi pagrindinį meniu bei įrankių juostą. Ši klasė yra paveldėta iš *Shared::ChildForm* klasės.

3.3.2 Biznio servisų paketas

Biznio servisų paketas susideda iš trijų pagrindinių klasių. Visos jos yra paveldėtos iš *Shared::COMPlusBase* klasės, kurioje yra sudėtas bendras funkcionalumas, kad paveldėtos klasės galėtų funkcionuoti, kaip Microsoft Windows COM+ objektai. Sistemos biznio servisų paketo klasių diagrama yra pateikta 6 pav. Ši diagrama yra pateikta abstrakčiame lygyje, nenorint apkrauti skaitytojo bereikalinga informacija.



6 pav. Biznio servisų paketo klasių diagrama

Shared::COMPlusBase yra bazinę klasė skirta COM+ objektams aprašyti. Ją turėtų paveldėti visos klasės, kurios turi būti sistemos biznio objektai.

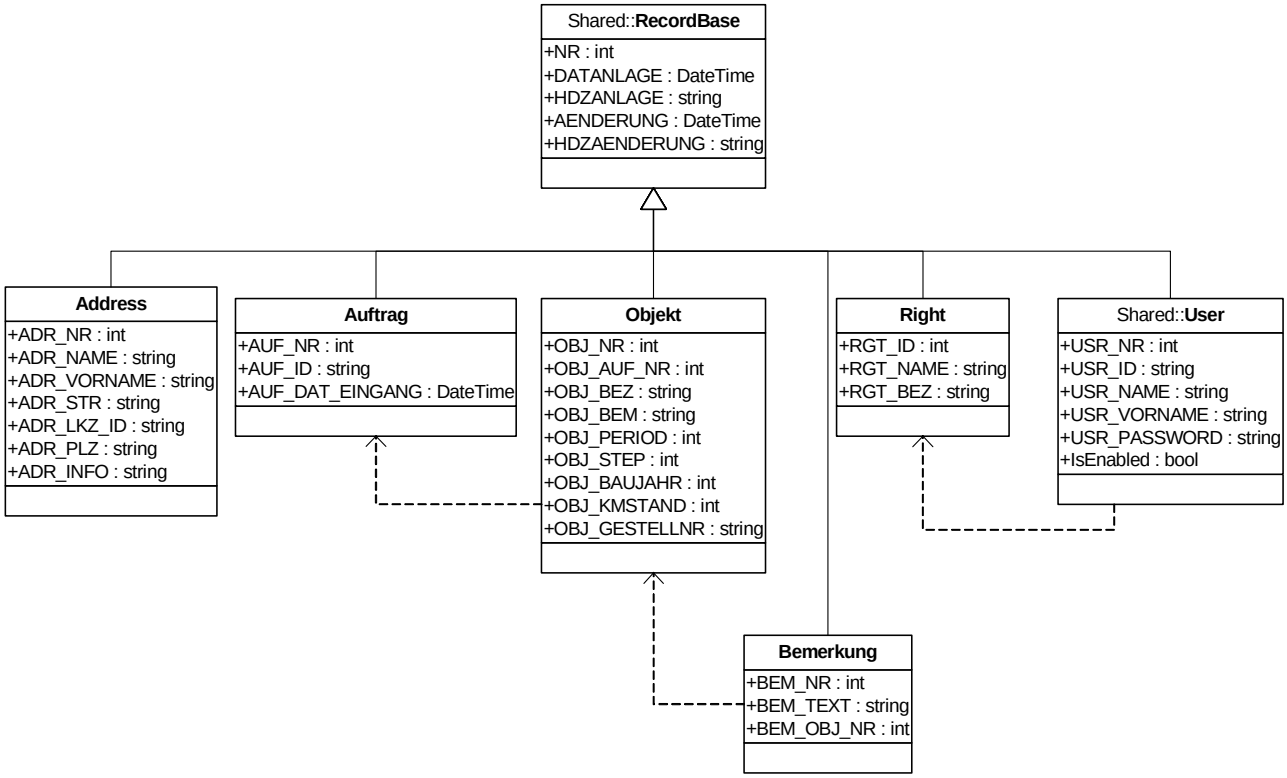
AddressManager - *t*ai biznio serviso klasė. Ši klasė yra skirta kliento duomenims sukurti, modifikuoti bei nuskaityti iš duomenų bazės. Ji yra paveldėta iš *Shared::COMPlusBase* klasės.

AuftragManager - tai biznio serviso klasė. Ši klasė yra skirta objekto duomenims sukurti, modifikuoti bei nuskaityti iš duomenų bazės. Taip pat skirta objekto įvertinimams saugoti bei modifikuoti. Ji yra paveldėta iš *Shared::COMPlusBase* klasės.

UserManager - tai biznio serviso klasė. Ši klasė yra skirta sistemos vartotojo duomenis sukurti, modifikuoti bei nuskaityti iš duomenų bazės. Taip pat skirta sistemos vartotojų teisėm priskirti bei modifikuoti. Ji yra paveldėta iš *Shared::COMPlusBase* klasės.

3.3.3 Biznio objektų paketas

Šiame pakete yra klasės skirtos duomenų mainams tarp grafinės vartotojo sąsajos ir biznio komponentų. Šitos klasės neatlieka jokių skaičiavimų tik saugo informaciją. Visose klasėse yra *ToString* metodas, kuris gražina „žmogišką“ (user-friendly) informaciją apie objekte saugomą informaciją. Biznio objektų klasių diagrama pateikta 7 pav. Ši diagrama yra pateikta abstrakčiame lygyje, nenorint apkrauti skaitytojo bereikalinga informacija.



7 pav. Biznio objektų paketo klasių diagrama

Shared::RecordBase - tai klasė atspinditi duomenų bazės įrašo bazinę klasę. Šioje klasėje saugoma įrašo kūrimo bei modifikavimo laikai ir vartotojai.

Shared::User – tai klasė atspinditi sistemos vartotojų lentelės įrašą duomenų bazėje. Ji yra paveldėta iš *Shared::RecordBase* klasės.

Address – tai klasė atspinditi adreso lentelės įrašą duomenų bazėje. Ji yra paveldėta iš *Shared::RecordBase* klasės.

Auftrag - tai klasė atspinditi užsakymo lentelės įrašą duomenų bazėje. Ji yra paveldėta iš *Shared::RecordBase* klasės.

Objekt - tai klasė atspinditi objekto lentelės įrašą duomenų bazėje. Ji yra paveldėta iš *Shared::RecordBase* klasės.

Bemerkung – tai klasė atspinditi objekto įvertinimo lentelės įrašą duomenų bazėje. Ji yra paveldėta iš *Shared::RecordBase* klasės.

Right - tai klasė atspinditi sistemos vartotojo teisių lentelės įrašą duomenų bazėje. Ji yra paveldėta iš *Shared::RecordBase* klasės.

3.4 Komponentų analizė

3.4.1 Patikimumo įvertinimas

Atlikus atskirų komponentų naudingumo ir patikimumo analizę, buvo nustatyta, jog biznio servisų komponentas yra vienas iš svarbiausių komponentų IS PAT. Priklausomai nuo jo realizavimo bus aišku ar pavyko įvykdyti pagrindinius sistemos projektavimo tikslus. Taip pat labai svarbus yra grafinės vartotojo sąsajos komponentas, kadangi sistemos išvaizda turi didelę įtaką vartotojų įspūdžiui apie produktą bei darbo įtaką su sistema.

Testavimas buvo pradėtas planuoti po pirmojo versijos pristatymo užsakovui. Atskiriems sistemos moduliams buvo taikomas struktūrinis testavimas, o visai sistemai - „black-box“ metodu, tačiau patys testai buvo parinkti analizuojant programos kodą. To pasekoje buvo apjungti du testavimo metodai – „black-box“ ir struktūrinis testavimas. Buvo išanalizuotos galimos kritinės situacijos ir sistema išsamiai patikrinta. Taip pat buvo suplanuotas bandomasis sistemos periodas. Užbaigta sistema buvo patalpinta realiose darbo sąlygose ir vartotojai turėjo galimybę kurį laiką stebėti ir analizuoti sistemos darbą. Tuo metu tikras darbas su sistema nebuvo vykdomas, tačiau buvo stengiamasi imituoti rutinines užduotis. Po sistemos užbaigimo šis bandomasis laikotarpis truko mėnesį. Dėl to, kad testavimas prasidėjo jau po pirmojo sistemos pristatymo, daugelis klaidų buvo ištaisytos iki šio bandomojo laikotarpio. Todėl šiuo laikotarpiu buvo aptiktos nekritinės sistemos klaidos, kurios buvo greitai ištaisytos, ir sistema buvo pradėta vartoti laiku.

Įvertinus nenumatytų situacijų pavojus ir sistemos tikslus, galima teigti, jog projekto sėkmė visiškai priklausė nuo biznio servisų komponento. Todėl pagrindinis dėmesys testuojant buvo skiriamas būtent šiam komponentui. Jis buvo testuojamas tiek struktūriniu tiek „juodos dėžės“ metodais [14].

Kuriant šią informacinę sistemą buvo pasirinkta testavimo strategija, kuri apėmė šias testavimo procedūras:

- Vienetų (*unit*) testavimas;
- Integravimo testavimas;
- Validacijos testavimas;
- High-order testavimas.

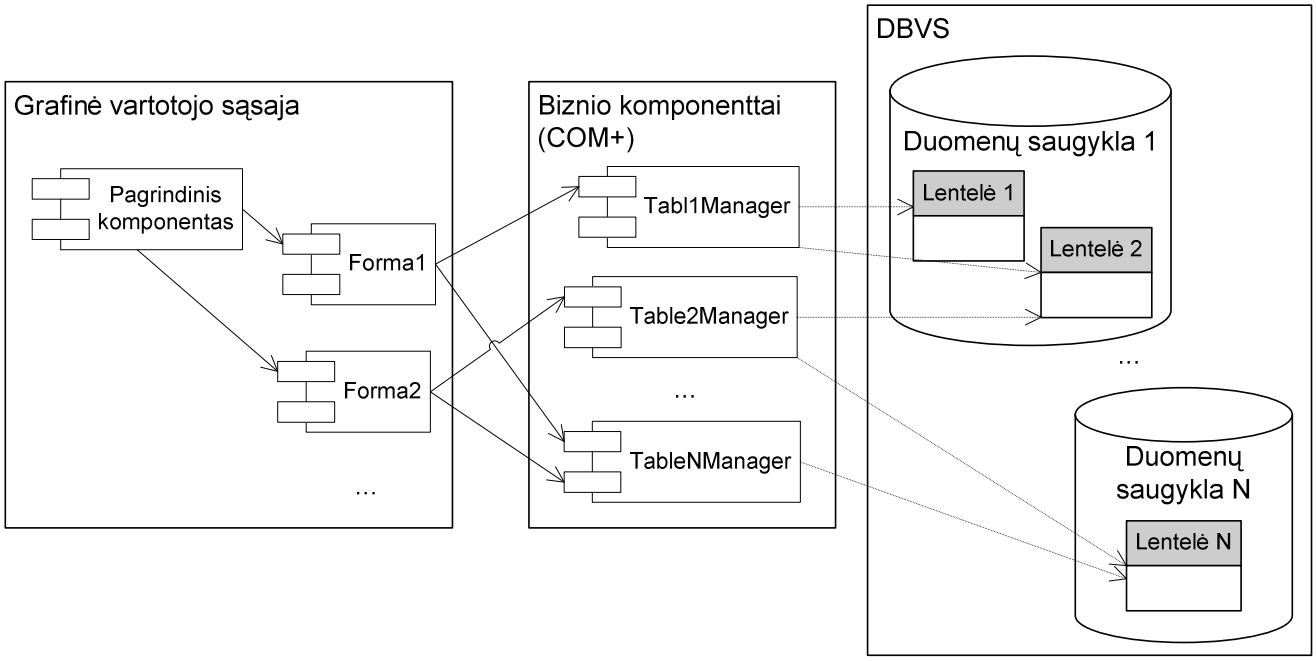
Visi šie testavimai buvo atliekami daug kartų. Kiekvieno testavimo metu aptiktos klaidos buvo ištaisytos.

3.4.2 Panaudojimo inicijavimo planas

Vartotojui bus pateikta sistemos įdiegimo programa, kuri automatiškai įdiegs informacinę sistemą PAT ir ją paruoš darbui. Programa startuos vietiniame tinkle, todėl užteks sistemą įdiegti tik viename, prie tinklo prijungtame kompiuteryje. Rekomenduotina, kad sistema būtų įdiegta kompiuteryje, kuris atitinka rekomenduojamą kompiuterio bei operacinės sistemos konfigūraciją.

4 NAUJOS SISTEMOS ARCHITEKTŪROS PALENGVINANČIOS PRIEŽIŪRĄ BEI TOBULINIMĄ

Išanalizavus sukurtos sistemos architektūrą buvo pastebėta, jog jos pakartotinio panaudojamumas nėra itin aukšto lygio. Taip yra dėl to, kad grafinė vartotojo sąsaja nėra visiškai atskirta nuo biznio komponentų (žr. 8 pav.). Taip pat dauguma biznio komponentų atlieka operacijas ne su viena duomenų bazės lentele, bet su keliomis. Dėl šių priežasčių sistemos palaikymas sudėtingėja bei palaikymo kaštai išauga.



8 pav. Pradinė sistemos architektūra

Norint to išvengti buvo atliktos kelios sistemos architektūros modifikacijos.

4.1 Keturių sluoksnių architektūra

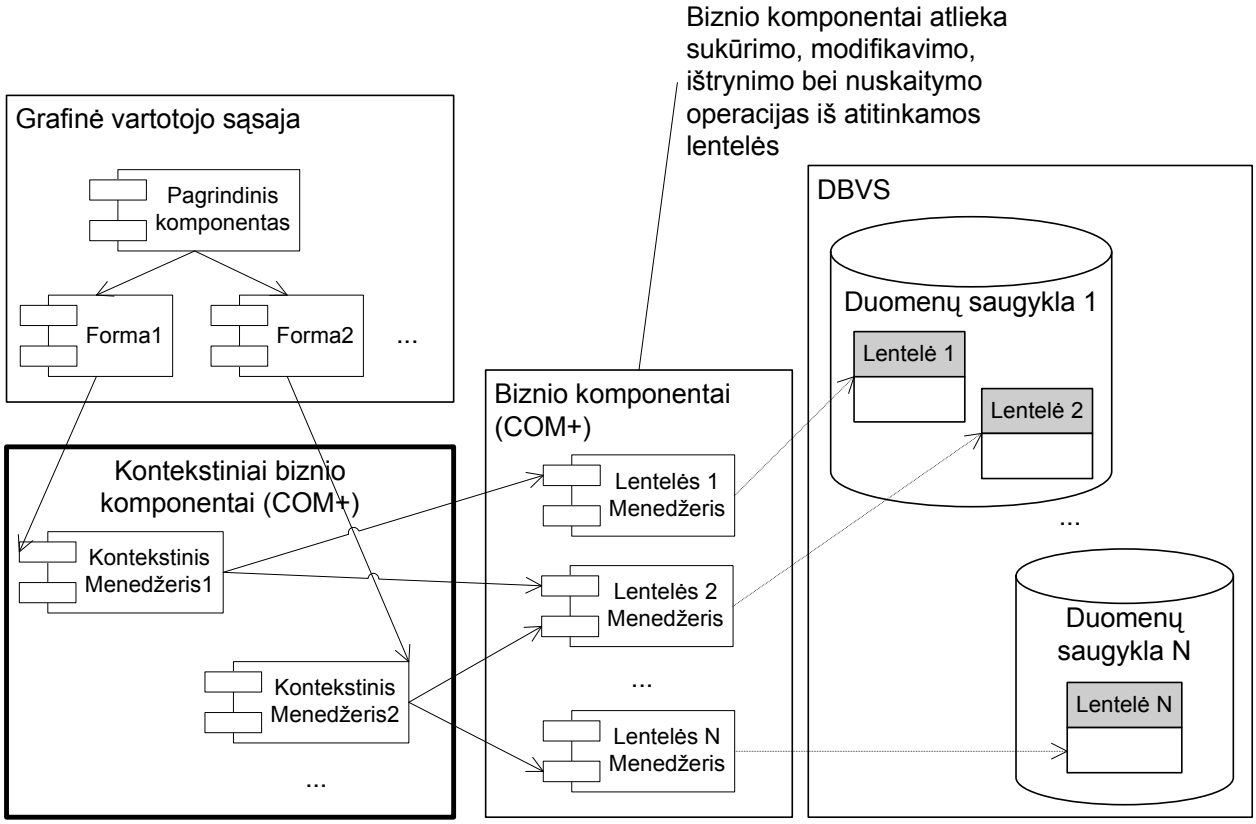
Pradinėje sistemos architektūroje buvo trys pagrindiniai sluoksniai:

- grafinė vartotojo sąsaja;
- biznio komponentai;
- duomenų sluoksnis.

Tačiau grafinei vartotojo sąsajai ir biznio komponentams trūko universalumo. Todėl buvo įtrauktas dar vienas kontekstinių biznio komponentų sluoksnis (žr. 9 pav.). Naujame sluoksnyje yra sudėti kontekstiniai menedžeriai. Jie atlieka tam tikros srities (konteksto) biznio veiklą. Pvz.: kliento informacijos menedžeris sugeba atlikti visas duomenų apdorojimo operacijas susijusias su kliento informacija; sistemos vartotojo menedžeris sugeba atlikti visas duomenų apdorojimo operacijas susijusias su sistemos vartotojo informacija, įtraukiant sistemos vartotojo teises.

Kontekstiniai biznio komponentai tai pat yra realizuoti pagal COM+ technologija. Viena iš priežasčių yra ta, kad atliekant vieną kontekstinio menedžerio operaciją gali būti modifikuojami kelių įrašų ar net kelių lentelių duomenys. Tačiau nepavykus atlikti sėkmingai bent vienos operacijos, visi duomenys turi būti atstatomi (*roll-back*), kad būtų išsaugotas duomenų integralumas. Tam yra naudojamos transakcijos, kurias palaiko COM+ technologija [11], [12].

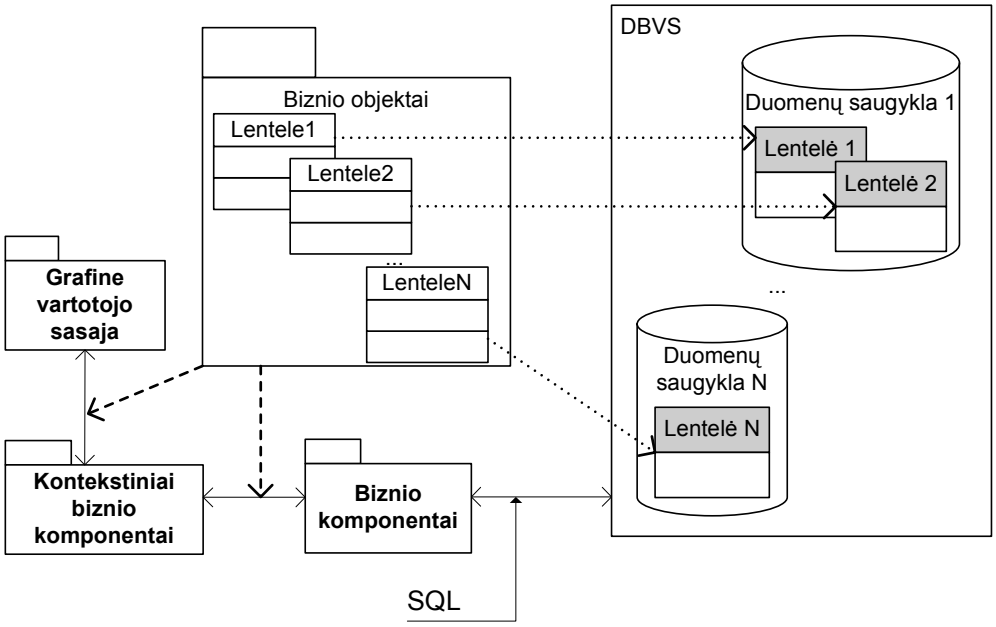
Įtraukus kontekstinių biznio komponentų sluoksnį, pasikeitė biznio komponentų apdorojamos informacijos kiekis. Dabar kiekvienas biznio komponentas apdoroja tik vienos lentelės duomenis.



9 pav. Keturių sluoksnių sistemos architektūra

Biznio objektai visoje sistemoje yra naudojami kaip duomenų konteineriai duomenims keistis tarp grafinės vartotojo sąsajos ir kontekstinių biznio komponentų, bei tarp kontekstinių biznio komponentų ir biznio komponentų (žr. 10 pav.). Kiekvienas biznio objektas gali įgyti tam tikrą būseną, kuri nusako atliekamas operacijas su duomenimis. Biznio objektas gali įgyti šias būsenas:

- nepakeistas;
- naujas;
- modifikuotas;
- ištrintas.



10 pav. Biznio objektų mainų diagrama

4.1.1 Architektūros įvertinimas

Naudojant tokią architektūrą yra aišku ką kiekvienas komponentas atlieka. Pasikeitus duomenų bazės lentelės struktūrai, reikėtų pakeisti ir biznio komponentą, atsakingą už tos lentelės informacijos apdorojimą. Taip pat nereikia pamiršti pakeisti ir biznio objekto, atspindinčio lentelės struktūrą.

Tokios sistemos pakartotinis panaudojamumas yra geresnis negu pradinės architektūros. Biznio komponentai gali būti pakartotinai panaudojami kitose programinėse įrangose.

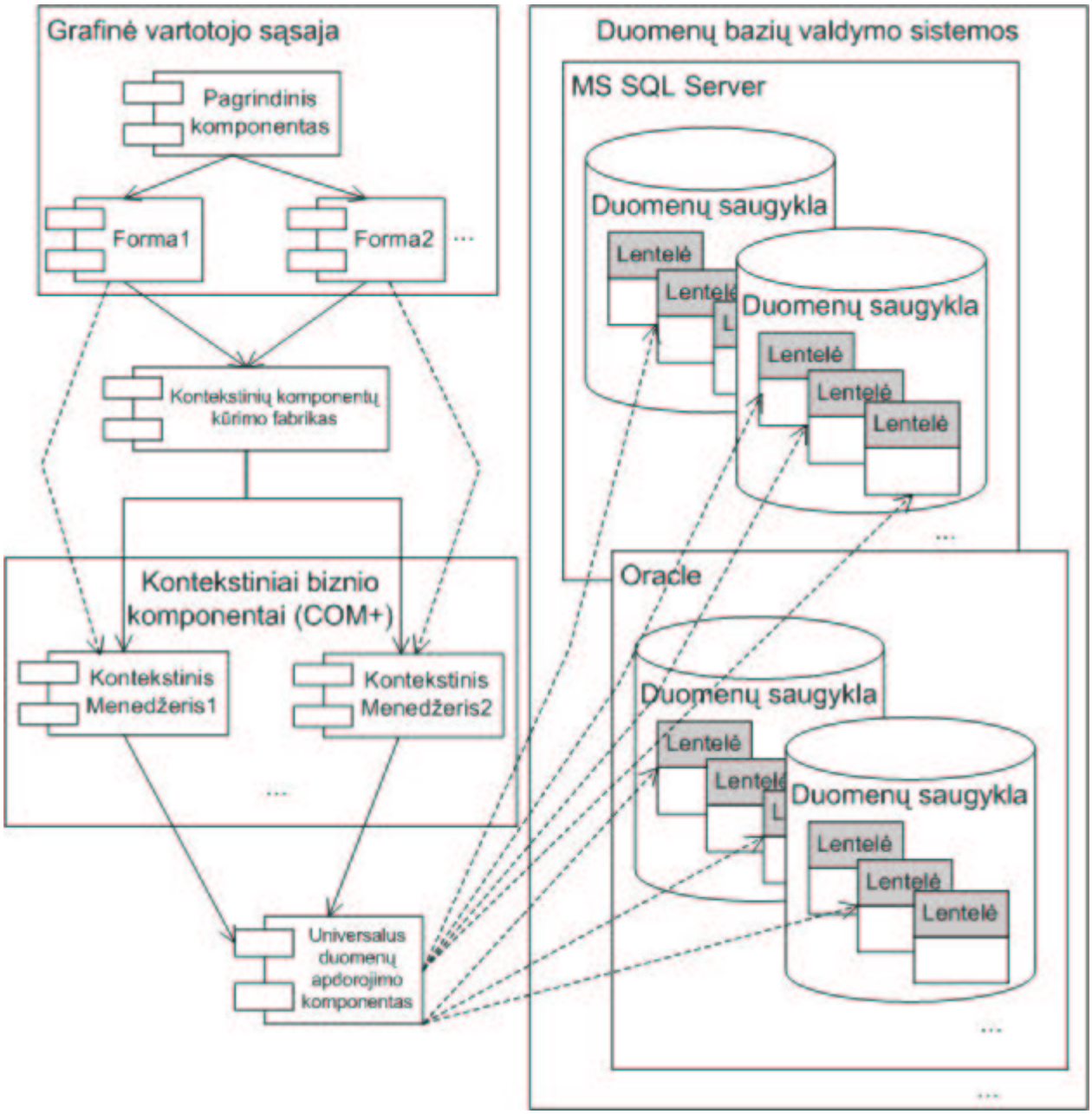
Dalinai atsiribojo grafinė vartotojo sąsaja nuo biznio logikos naudojant kontekstinius biznio komponentus. Tačiau vis dar yra likęs ryšys tarp šių dviejų komponentų.

4.1.2 Pagrindiniai privalumai ir trūkumai

+ Kuriant naują projektą ir žinant kurių lentelių reikės, bus lengva atsirinkti ir lentelių menedžerius.
+ Sumažėjęs ryšys tarp grafinės vartotojo sąsajos bei sistemos biznio logikos;
+ Atsirado aiškumas kuris komponentas už ką yra atsakingas;
– atsiradus naujai lentelei duomenų bazėje reikia sukurti naują lentelės menedžerį bei biznio objektą;
– Duomenų bazėje esant daug lentelių, automatiškai atsiranda labai daug lentelių menedžerių ir atitinkamų biznio objektų;

4.2 Penkių sluoksnių architektūra

Pradinėje sistemos architektūroje buvo trys pagrindiniai sluoksniai, o patobulintoje architektūroje – keturi. Pabandykime įdėti dar vieną sluoksnį ir atlikti kelis pakeitimus. Rezultatą galite pamatyti 11 paveikslėlyje.



11 pav. Penkių sluoksnių architektūra

Ši architektūra sudaryta iš penkių sluoksnių (lygių):

- Grafinė vartotojo sąsaja;
- Biznio komponentų kūrimo fabrikas;
- Biznio komponentai;
- Duomenų apdorojimas;
- Duomenų sluoksnis.

4.2.1 Universalus duomenų apdorojimo komponentas

Ši architektūra turi panašumų su keturių sluoksnių architektūra. Biznio komponentai buvo pakeisti universaliu duomenų apdorojimo komponentu (UDAK). Šis komponentas gali atlikti visas biznio komponento operacijas. Jam tereikia perduoti biznio objektą ir iškviesti norimą operaciją (nuskaityti ar apdoroti duomenis). Universalus komponentas išnagrinėjęs biznio objekto struktūrą bei nuskaitęs visų savybių (*properties*) atributus pats žino į kokią duomenų bazių valdymo sistemą kreiptis ir kokioje duomenų bazėje yra lentelė bei pats įrašas. Taip pat žino kokia biznio objekto savybė atitinka lentelės kolonėlę.

Sistemos vykdymo metu (*run-time*) objektų analizę galima atlikti panaudojus .NET reflection metodologiją [11], [12]. Refleksijos metodologiją galima naudoti dinamiškai kuriant objektus pagal nurodytą tipą, susieti tipą su egzistuojančiu objektu ar gauti egzistuojančio objekto tipą. Po to galima iškviesti tipo metodą ar pasiekti laukus bei savybes[1].

Naudojant tokį komponentą, mes išvengiame naujų biznio komponentų kūrimo, atsiradus naujai lentelei duomenų bazėje. Taip pat nebereikia modifikuoti biznio komponentų pasikeitus lentelės struktūrai. Tereikia sukurti ar modifikuoti biznio objektą.

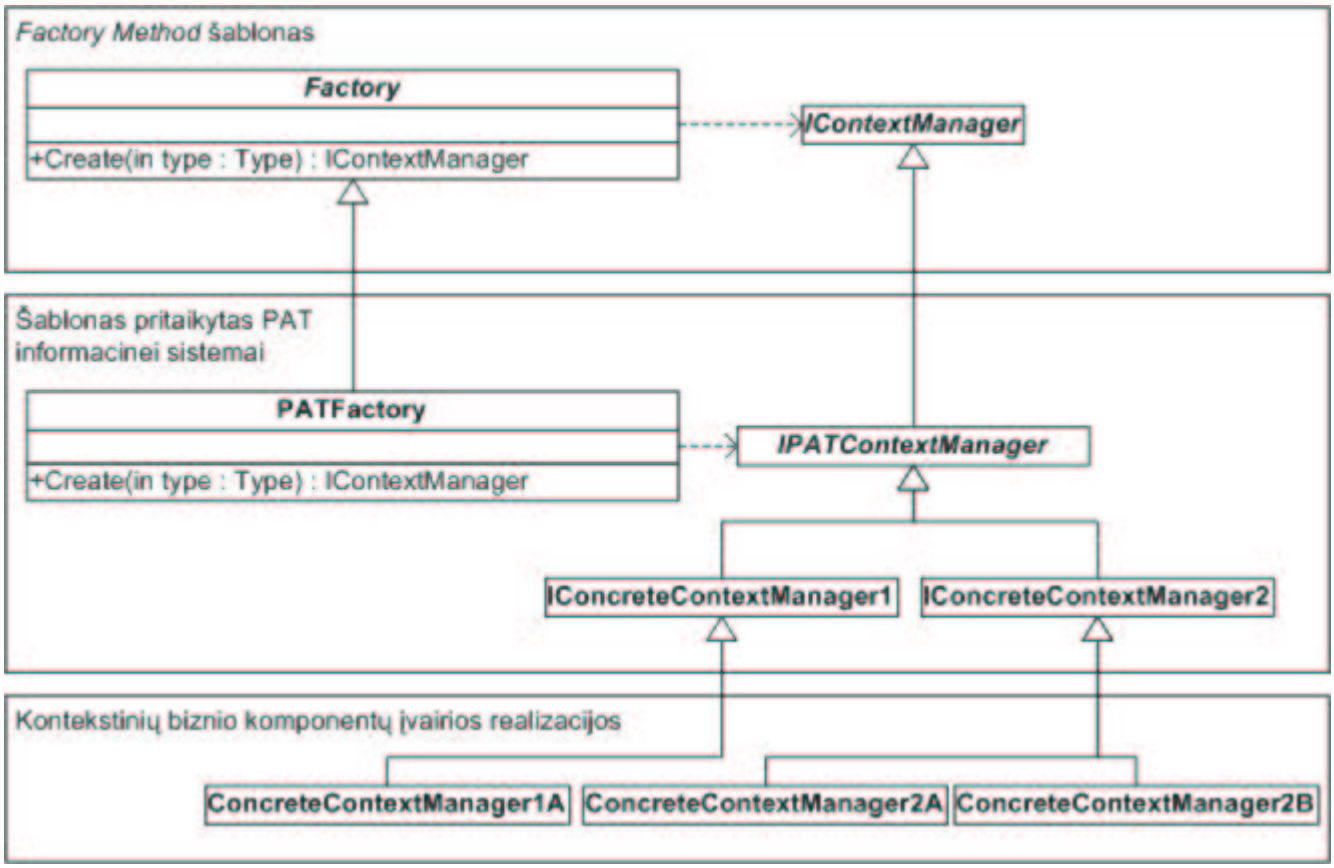
Tas pats universalus duomenų apdorojimo komponentas gali būti naudojamas visose sistemose, kurių architektūra atitinka šią architektūrą. Naudojant šį komponentą, sistemų priežiūra ir tobulinimas labai palengvėja. Norint papildyti duomenų bazę naujomis lentelėmis, tereikia sukurti naujus biznio objektus. Tokių objektų kūrimą galima palengvinti sukūrus papildomą priemonę, kuri tiesiog sugeneruotų *.NET C#* klasę, atitinkančią duomenų bazės lentelės struktūrą. T.y. sukurtų klasę, kuri turėtų savybes (*properties*) atitinkančias lentelės kolonėles. Kiekvienai klasės savybei būtų priskiriami atitinkami atributai nurodantys kolonėlių tipus – raktas (*key*), išorinis raktas (*foreign key*) ar paprasta kolonėlė ir kt. Ši informacija yra reikalinga atliekant operacijas su duomenimis.

Universalus duomenų apdorojimo komponentas operuoja bazinėmis biznio objektų klasėmis. Po kiekvienos atliktos operacijos kontekstiniai biznio komponentai turi konvertuoti (*cast*) UDAK gražinamus objektų tipus į tikruosius. Tačiau atliekant tipų konvertavimą sugaištama šiek tiek laiko [16], [17]. Tačiau konvertavimo metu sugaištas laikas neprilygsta refleksinei objekto analizei. Atliekant objektų analizę sugaištama kur kas daugiau laiko.

[1] You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object. You can then invoke the type's methods or access its fields and properties.

4.2.2 Kontekstinių biznio komponentų fabrikas

Dar viena naujovė penkių sluoksnių architektūroje – kontekstinių biznio komponentų kūrimo fabrikas. Naudojant šį fabriką, maksimaliai atsiejame grafinę vartotojo sąsają nuo sistemos biznio veiklos. Šis fabrikas yra realizuotas pagal *Factory method* šabloną (*pattern*) (žr. 12 pav.).



12 pav. *Factory method* šablonas

Factory method šablonas sudarytas iš dviejų klasių: fabriko ir bazinės klasės (arba sąsajos). Fabriko klasė turi tik vieną metodą *Create* (sukurti). Šis metodas turi tik vieną parametrą, kurio tipas yra *type*. Šis tipas nurodo kokio tipo objektą reikia sukurti. Sukurtas objektas nebūtinai turi atitikti perduoto tipo objektą. Objektas gali būti sukurtas iš paveldėtų klasių. Metodas *Create* gražina jau sukurtą objektą.

Šis fabrikas kuria kontekstinius biznio komponentus pagal nurodytą kontekstinio biznio komponento sąsajos (interface) tipą. Taip atsiribojama nuo konkrečios kontekstinio biznio komponento realizacijos. Naudojant šį šabloną yra inkapsuliuojamas konkrečių kontekstinių biznio komponentų sukūrimas. Mums nėra įdomus pats kūrimo procesas, mums yra svarbiausias rezultatas – sukurtas kontekstinis biznio komponentas atitinkantis nurodytą kontekstinio biznio komponento sąsajos struktūrą. Tai yra viena iš objektinio programavimo savybių. Grafinei vartotojo sąsajai tereikia „žinoti“ kurį kontekstinio biznio komponento sąsają perduoti fabrikui.

Pritaikius šį šabloną kontekstinių biznio komponentų kūrimui, skirtingose sistemos realizacijose galima naudoti skirtingas kontekstinių biznio komponentų realizacijas.

4.2.3 Architektūros įvertinimas

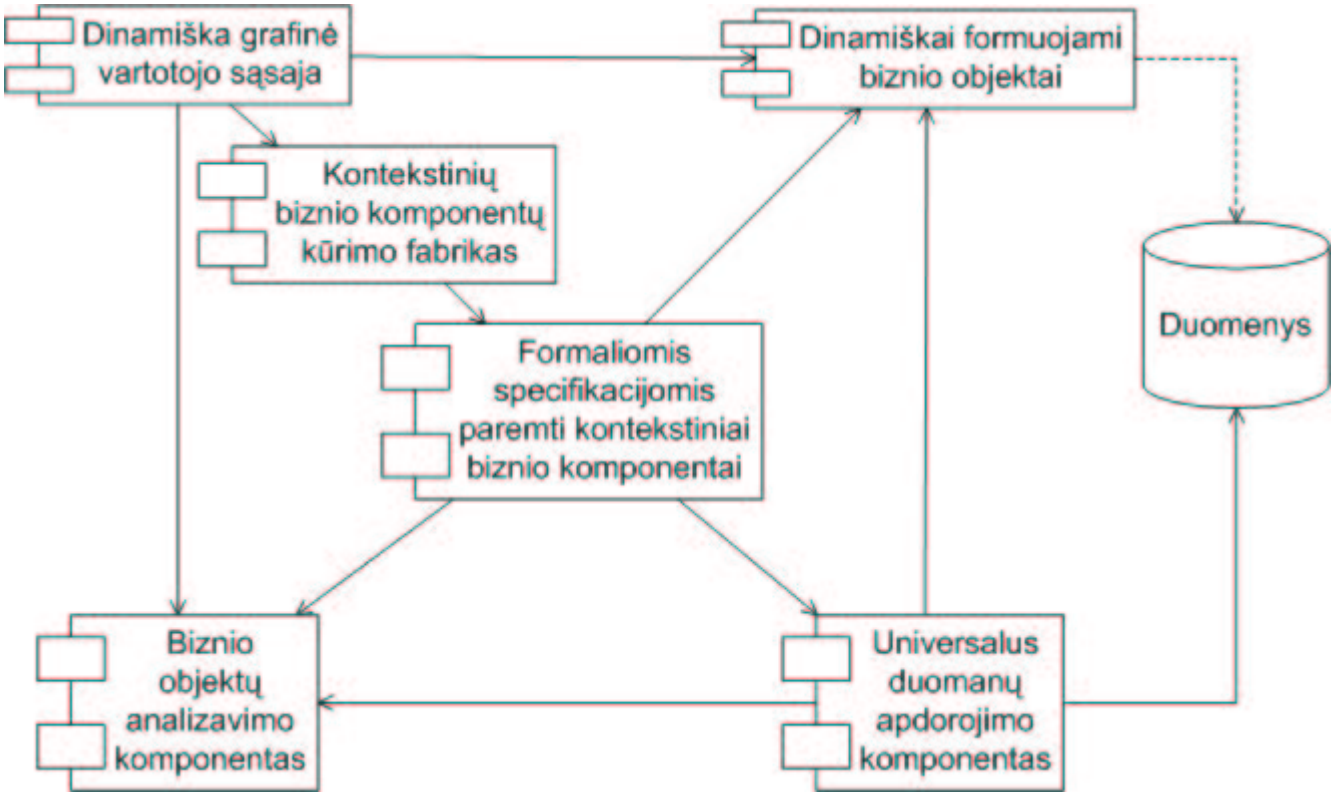
Šios architektūros įvertinimas ir palyginimas su pradinės PAT sistemos architektūra yra pateiktas 5 skyriuje. Ši architektūra buvo pritaikyta naujai PAT sistemai dėl daugelio priežasčių, kurių pagrindinės butų šios:

- atsiradusi galimybė lengvai ir greitai įtraukti bei panaudoti sistemoje naujas duomenų bazės lenteles, naujo tipo informacijai apdoroti;
- susilpnėjęs ryšys tarp grafinės vartotojo sąsajos ir sistemos biznio komponentų.

Aukščiau išvardinti faktoriai labai palengvino sistemos priežiūrą bei tobulinimą.

4.3 Dinamiškos architektūros vizija

Dinamiškai formuojami biznio objektai, dinamiškai kuriama grafinė vartotojo sąsaja, dinamiškai užkraunamos ir pašalinamos iš atminties rinkmenos (*assemblies*), formaliomis specifikacijomis aprašoma sistemos biznio logika – taip galėtų skambėti naujos architektūros apibūdinimas. Norint stipriai atsiriboti nuo sistemos priežiūros, reikėtų susimastyti apie tokios architektūros įgyvendinimą visose sistemose (žr. 13 pav.).



13 pav. Dinamiška architektūra

Jeigu darant prielaidą, kad tokie komponentai visomis sąlygomis veikia teisingai, tai sistemos priežiūros ir tobulinimo darbų kaštai labai sumažėtų. Sistemoms prižiūrėti tereikėtų tik vieno specialisto, sugebančio teisingai sukonfigūruoti dinaminius komponentus. Žinoma, tokių sistemų efektyvumas nebūtų itin aukštas. Tokią architektūrą būtų galima taikyti nekritinėms sistemoms ir toms

sistemos, kurioms nėra itin svarbus labai didelis operacijų atlikimo greitis. Sistemoms, kurios dažniausiai atlieka duomenų kaupimo, redagavimo darbus, būtų galima laisvai taikyti tokią architektūra.

Sistemų priežiūrai reikėtų papildomai sukurti tokios architektūros priežiūros automatizuotas priemones. Naudojant šias priemones, būtų galima keisti sistemos struktūrą, o pati priemonė automatiškai turėtų adaptuoti struktūros pakeitimus sistemai.

Toliau pasistengsiu pateikti dinamiškos architektūros abstraktų vaizdą.

4.3.1 Dinamiškai formuojami biznio objektai

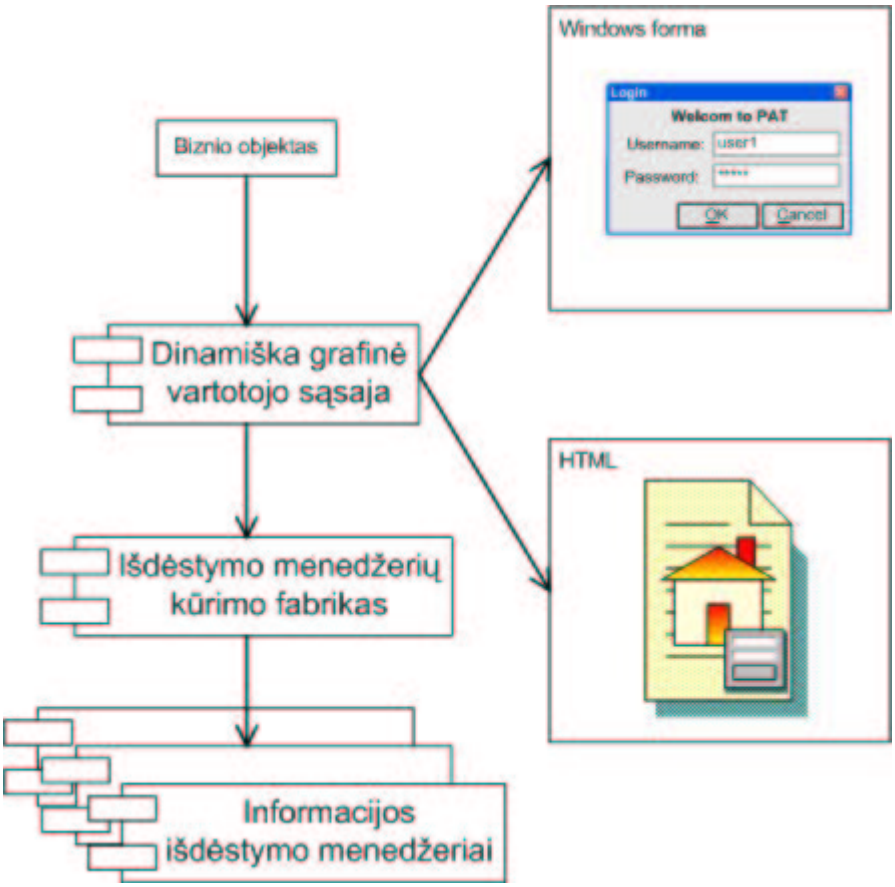
Naudojant automatizuotas priemones, dinamiški biznio objektai būtų adaptuojami prie pasikeitusios sistemos struktūros. Jeigu būtų sukurta nauja lentelė duomenų bazėje, tuomet turėtų būti sukuriamas ir dinamiškas biznio objektas pagal sukurtos lentelės struktūrą bei įtrauktas į sistemą. Po biznio objekto sukūrimo ar modifikavimo sistemos perkompiliuoti nereikėtų dėl to, kad biznio objektai būtų užkraunami panaudojus .NET refleksijos (*reflection*) technologijos galimybes. Atlikus reikiamas operacijas su biznio objektais, jie automatiškai būtų pašalinami iš atminties dėl dviejų priežasčių:

1) užkrauta informacija užima vietą atmintyje; operuojant dideliais informacijos kiekiais, galima apkrauti operacinę sistemą, kas tik dar labiau sulėtintų sistemos darbą arba iš viso sustabdytų;
2) jeigu biznio objektai nebūtų pašalinami iš atminties, tai atlikus sistemos struktūros pakeitimus, naudojant automatizuotas priemones, sistemos veikimas gali sutrikti dėl nevienodų objektų tipų.

Patys biznio objektai nesiskirtų nuo penkių sluoksnių biznio objektų, skirtųsi tik jų kūrimas bei jų panaudojimo būdas.

4.3.2 Dinamiškai kuriama grafinė vartotojo sąsaja

Vienas iš sudėtingiausių komponentų – tai dinamiškos grafinės vartotojo sąsajos komponentas. Šis komponentas turi generuoti tiek *html* puslapius, tiek Windows formas. Bet koks informacijos išdėstymas (formoje ar *html* puslapyje) nėra tinkamas, nes grafinė vartotojo sąsaja turi būti išvaizdi ir patogi vartotojui. Tam naudojami informacijos išdėstymo menedžeriai (žr. 14 pav.).



14 pav. Dinamiškos grafinės vartotojo sąsajos architektūra

Informacijos išdėstymo menedžerių kūrimui panaudotas *Factory method* šablonas, suteikia grafinei vartotojo sąsajai lankstumo ir atsiriboja nuo konkrečios informacijos išdėstymo menedžerio realizacijos.

4.3.3 Formaliomis specifikacijomis aprašoma sistemos biznio logika

Dar vienas architektūros pliusas - formaliomis specifikacijomis aprašoma sistemos biznio logika. Sukūrus automatizuotą pagalbinę priemonę, kurią naudodami galėtume formaliai aprašyti biznio logiką, mes atsiribojame nuo tiesioginio kontekstinių biznio komponentų kūrimo. Pagal formalias specifikacijas, automatizuota pagalbinė priemonė sugeneruotų kontekstinius biznio komponentus ir juos automatiškai įtrauktų į sistemą. Tarkim buvo sukurtas kontekstinis biznio komponentas. Tada mes, naudodami automatizuotą pagalbinę priemonę, atlikome kelis pakeitimus ir sukūrėme kitą kontekstinio biznio komponento realizaciją. Tuomet automatizuota pagalbinė priemonė turi sukonfigūruoti kokį kontekstinį biznio komponentą sistema turi naudoti – t.y. sukonfigūruoti kontekstinių biznio komponentų kūrimo fabriką.

5 PATOBULINTOS ARCHITEKTŪROS ĮVERTINIMAS

5.1 Eksperimento tikslas

Eksperimento tikslas yra nustatyti pradinės PAT sistemos bei pritaikius penkių sluoksnių architektūrą sistemos efektyvumą. Taip pat palyginti ir įvertinti sistemų priežiūros bei tobulinimo galimybes. Remiantis šiais tikslais buvo suformuoti eksperimento uždaviniai:

a) ištirti ir palyginti darbo su duomenų bazėmis efektyvumo skirtumus;
b) ištirti ir palyginti sistemos darbo efektyvumo skirtumus naudojant ir nenaudojant biznio komponentų kūrimo fabriko;
c) ištirti ir palyginti sistemos prezentacinio lygmens (grafinės vartotojo sąsajos) darbo efektyvumo skirtumus;
d) ištirti ir palyginti sistemų priežiūros bei tobulinimo skirtumus.

Šių uždavinių tikslui pasiekti buvo sukurtos kelios bandomosios programos skirtingai atliekančios duomenų apdorojimo operacijas bei naudojančios skirtingas duomenų analizavimo metodologijas.

Duomenų apdorojimo efektyvumo įvertinimas

Duomenų apdorojimo efektyvumo buvo tiriamas vykdant tas pačias operacijas su skirtingomis bandomosiomis programomis. Duomenų apdorojimo efektyvumo įvertinimui buvo atliekami tokie veiksmai:

a) duomenų skaitymas;
b) naujų duomenų įrašymas;
c) egzistuojančių duomenų modifikavimas.

Duomenų nuskaitymui buvo naudojami keli variantai:

a) vieno įrašo nuskaitymas;
b) vieno įrašo nuskaitymas naudojant sudėtingus filtrus;
c) vieno įrašo nuskaitymas naudojant sudėtingus filtrus iš skirtingų lentelių kolonėlių (naudojami *MS SQL* apjungimo filtrai (*Join filter*))
d) visų įrašo nuskaitymas;
e) įrašų nuskaitymas naudojant sudėtingus filtrus;
f) įrašų nuskaitymas naudojant sudėtingus filtrus iš skirtingų lentelių kolonėlių (naudojami *MS SQL* apjungimo filtrai (*Join filter*))

Kadangi operacinėje sistemoje veikė ir kiti procesai, kurie potencialiai galėjo iškreipti matuojamus operacijų vykdymo laikus, tai kiekviena operacija buvo kartojama po 10 kartų. Visi testai

taip pat bvo pakartojami 10 kartų (t.y. kiekviena operacija buvo atlikta po 100 kartų). Rezultatai buvo susumuojami ir padalinti iš atliktų operacijų skaičiaus. Taip buvo gauti rezultatų vidurkiai.

Kaip ir buvo tikėtasi visos patobulintos architektūros efektyvumas yra blogesnis negu senosios. Taip yra dėl to, kad naujoje architektūroje yra naudojama .NET refleksijų (*reflection*) technologija, pagal kurią yra sukurtas universalus duomenų apdorojimo komponentas (UDAK). Ši technologija suteikia UDAK universalumo, tačiau sumažina efektyvumą (žr. priedo A 1 - 8 pav.).

Ši architektūra yra pritaikyta sistemai, kuri nereikalauja labai didelio operacijų atlikimo greičio, tad nereikėtų į tai labai kreipti dėmesio. Ateityje reikėtų peržiūrėti UDAK architektūros realizaciją ir atlikti kodo optimizaciją [16], [17]. Jeigu šis komponentas nėra optimizuotas, tai jo efektyvumą galima padidinti bent 50 proc.

Duomenų nuskaitymui, naudojant sudėtingus filtrus iš skirtingų lentelių kolonėlių, įtakos turėjo ir nevisiškai identiškų filtrų sudarymas – UDAK suformuotuose SQL sakiniuose filtras, tam tikrais atvejais, turėjo perteklinės informacijos. Tačiau gauti rezultatai nuo to nepasikeitė. Yra sudėtinga sukurti universalų komponentą, kuris tiktų visiem gyvenimo atvejams.

Išanalizavus šiuos eksperimentus, galima teigti, jog komponentų universalumo kaina yra jų efektyvumo sumažėjimas. Tačiau efektyvumo sumažėjimas yra labai nežymus ir sistemos vartotojo darbui įtakos neturės.

5.2 Biznio komponentų fabriko efektyvumo įvertinimas

Norint ištirti sistemos efektyvumo įvertinimą, naudojant biznio komponentų fabriką, vienu atvejų buvo kuriami biznio komponentai pagal senąją architektūrą – nenaudojant biznio komponentų fabriko, kitu atveju – naudojant biznio komponentų fabriką.

Panaudojus biznio komponentų fabriką naujoje architektūroje padidiname sistemos universalumą ir nepriklausomumą nuo biznio komponentų realizacijos. Grafinė vartotojo sąsaja tampa maksimaliai atskirta nuo sistemos biznio logikos. Tai yra labai svarbus momentas visoms ateities sistemoms. Taip pat maksimaliai atskyrę grafinę vartotojo sąsają palengviname sistemų priežiūrą ir tobulinimą.

Eksperimento rezultatai parodė, jog padidinus sistemos universalumą – naudojant biznio komponentų fabriką – jos efektyvumas sumažėjo. Tačiau efektyvumo sumažėjimas yra itin mažas – nuo vienos iki dviejų milisekundžių kiekvienam kuriamam biznio komponentui (žr. 9 pav.). Atlikus tyrimą, paaiškėjo, jog naudojant biznio komponentų fabriką, biznio komponentų sukūrimas sulėtėja viena dešimtąja milisekundės kiekvienam sukurtam biznio komponentui.

5.3 Prezentacinio lygmens efektyvumo įvertinimas

Prezentacinio lygmens tyrimas buvo toks pat sudėtingas, kaip ir jo testavimas. Grafinė vartotojo sąsaja - tai daugiausiai darbo reikalaujanti sritis. Tad ir ištirti jos efektyvumą buvo nelengva, nes eksperimentinių situacijų yra itin daug. Todėl buvo nagrinėjamos tik pagrindinės sistemos būsenos, kurios yra svarbiausios sistemos vartotojams:

- matuojamas sistemos startavimo greitis (laikas nuo sistemos paleidimo momento iki sistemos ramybės būsenos (*idle*));
- matuojamas sistemos darbo baigimo greitis (laikas nuo baigimo komandos iškvietimo iki tol, kol operacinėje sistemoje sistemos procesas baigiasi);
- matuojamas paieškos lango sukūrimo bei rezultatų pateikimo greitis (laikas paieškos langų kūrimo iki rezultatų gavimo ir sistemos ramybės būsenos). Naujoje architektūroje paieškos langai formuojami dinamiškai pagal biznio objektą, išanalizavus jį naudojant *.NET* refleksijų technologiją.

Atlikti eksperimentai taip pat patvirtino prielaidą, jog didėjant sistemos universalumui sistemos efektyvumas sumažėja. Naujos architektūros sistemos startavimo efektyvumo sumažėjimui turėjo padidėjusi sistemos apimtis – padaugėjo komponentų kiekis, kuriuos .NET aplinka automatiškai užkrauna sistemai startuojant (žr. 10 pav.). Tačiau efektyvumo sumažėjimas nėra toks didelis, kad sistemos vartotojas galėtų pastebėti – sulėtėji tik dviem šimtosiomis sekundės dalimis.

Taip pat šiek tiek padidėjusiam sistemos baigimo laikui naujoje architektūroje, pirmu atveju, įtakos turėjo naujoje sistemoje padaugėjusių komponentų skaičius. Antru atveju buvo atliekami įvairūs veiksmai prieš užbaigiant darbą su sistema. Naujoje architektūroje yra naudojama duomenų kaupimo (*cache*) technologija. Kartą nuskaičius, nekintančius sistemos duomenis iš duomenų bazės, antrą kartą imami iš sukauptų duomenų saugyklos. Taip yra minimizuojamas DBVS bei tinklo apkrovimas ir taupomas informacijos gavimo laikas. Tačiau sistemai baigiant darbą, yra įvykdomas šiukšlių surinkimo (*Garbage collector*) darbas [11], [12]. Šis procesas yra labai lėtas, ypač tada, kai sistemoje yra prikaupta labai daug bereikalingos informacijos. Visa tai sulėtina sistemos baigimo darbą, bet naudojant šias technologijas yra kompensuojamas sistemos efektyvumas darbo metu.

5.4 Priežiūros ir tobulinimo įvertinimas

Akivaizdu, kad patobulinta architektūra yra daug universalesnė įvairiuose sistemos architektūros sluoksniuose, negu pradinės IS PAT. Nors ir sistemos efektyvumas yra sumažėjęs, tačiau jis nėra labai didelis, tad sistemos vartotojai to tiesiog neturėtų pastebėti. Tuo tarpu padidėjęs universalumas palengvima sistemos priežiūrą bei tobulinimą.

Panaudojus biznio komponentų fabriką naujoje architektūroje padidiname sistemos universalumą ir nepriklausomumą nuo biznio komponentų realizacijos. Grafinė vartotojo sąsaja tampa

maksimaliai atskirta nuo sistemos biznio logikos. Tai yra labai svarbus momentas visoms ateities sistemoms. Taip pat maksimaliai atskyrę grafinę vartotojo sąsają palengviname sistemų priežiūrą ir tobulinimą.

Naudojant universalų duomenų apdorojimo komponentą (UDAK), mes išvengiame naujų biznio komponentų kūrimo, atsiradus naujai lentelei duomenų bazėje. Taip pat nebereikia modifikuoti biznio komponentų pasikeitus lentelės struktūrai. Tereikia modifikuoti biznio objektą. Tas pats universalus duomenų apdorojimo komponentas gali būti naudojamas visose sistemose, kurių architektūra atitinka patobulintąją architektūrą. Naudojant šį komponentą, sistemų priežiūra ir tobulinimas labai palengvėja. Norint papildyti duomenų bazę naujomis lentelėmis, tereikia sukurti naujus biznio objektus. O šių objektų kūrimą galima palengvinti sukūrus papildomą automatizuotą priemonę, galinčią sukurti ar adaptuoti biznio objektą prie pasikeitusios duomenų bazės lentelės.

Įvertinus aukščiau pateiktus faktorius, galima teigti, jog patobulintos architektūros priežiūra ir tobulinimas yra labai palengvintas. Įgyvendinus visiškai dinamiškos architektūros viziją ir pritaikius ją naujų sistemų kūrimui, būtų dar labiau palengvinta sistemų priežiūra bei tobulinimas.

6 TOLIMESNI DARBAI

Reikėtų detaliau išanalizuoti bei suprojektuoti dinamišką architektūrą. Taip pat reikėtų pritaikyti tokią architektūrą IS PAT bei palyginti visų architektūrų efektyvumą. Kadangi architektūra butų labai lanksti, tai galima spėti, kad jos efektyvumas turėtų būti mažesnis negu anksčiau išnagrinėtų architektūrų.

Taip pat reikėtų išanalizuoti universalaus duomenų apdorojimo komponento optimizavimo galimybes bei jas įgyvendinti.

Reikėtų pakeisti biznio objektų analizę stengiantis kuo mažiau naudoti .NET refleksijų technologiją. Dalinai biznio objektų analizę perkelti iš UDAK į pačius biznio objektus. Įgyvendinus šiuos pakeitimus, sistemos efektyvumas turėtų žymiai pagerėti.

7 IŠVADOS

1) Išnagrinėti keli sistemos architektūros patobulinimai;
2) Įgyvendinta sistema, pagal patobulintą architektūrą, kuri palengvino sistemos priežiūrą ir tobulinimą;
3) Pasiūlyta viena sistemos architektūros vizija, pagal kurią sukurtų sistemų priežiūra ir tobulinimas yra itin paprastas;
4) kiekvienas atliktas architektūros patobulinimas, kurio dėka sistemos architektūra tampa universalesnė, šiektiek sulėtina sistemos darbą, tačiau palengvina priežiūros bei tobulinimo darbus;
5) sistemos priežiūros numatymas ir problemų prognozavimas ankstyvosiose projekto gyvavimo stadijose gali būti svarbi projekto sėkmės priežastis;
6) sukurta veikianti sistema, kuri yra naudojama Vokietijoje įkurtoje įmonėje PAT (Proceed Asset Trading), kuri teikia ilgalaikės nuomos (lizingo) paslaugas.

8 PADĖKOS

Dėkoju Kauno technologijos universitetui, už suteiktą galimybę studijuoti ir įgyti bakalauro bei magistro laipsnį.

Už suteiką galimybę tobulėti, siekti praktinių žinių, atliekant praktinį darbą studijų metu bei toliau dirbti įmonėje norėčiau padėkoti UAB „Baltic Software Solutions“ įmonės direktoriui Aidui Kavaliauskui.

Taip pat norėčiau padėkoti magistrinio darbo vadovui doc dr. Aleksui Riškui už patarimus studijuojant magistratūroje bei rašant magistrinį darbą.

9 LITERATŪRA

1. Pigoski T. M. *SWEBOK Knowledge Area Description for Software Evolution and Maintenance* - Technical Software Services (TECHSOFT), Inc.1999.
2. IEEE STD 1219: *Standard for Software Maintenance* [žiūrėta 2004-03-22]. Prieiga per internetą: < http://standards.ieee.org/ >
3. ISO/IEC 12207: *Information Technology-Software Life Cycle Processes* [žiūrėta 2004-03-22]. Prieiga per internetą: < http://www.ieee.org/ >
4. Pressman. R. S. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, fourth edition, 1997.
5. Sommerville I. *Software Engineering*. McGraw-Hill, fifth edition, 1996.
6. Pigoski T. M. *Practical Software Maintenance: Best Practices for Managing your Software Investment*. Wiley, 1997.
7. Pfleeger S. L. *Software Engineering—Theory and Practice*. Prentice Hall, 1998.
8. Martin J. and McClure C. *Software Maintenance: The Problem and its Solutions*. Prentice-Hall, 1983.
9. McConnell S. *Rapid development* – Microsoft Press 1996.
10. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns, Elements of Reusable Object* – Oriented Software – Addison – Wesley 1997.
11. Liberty J. *Programming C#* - O'Reilly & Associates 2002.
12. Microsoft Developer Network (MSDN) Library, Microsoft projektuotojų tinklo biblioteka [žiūrėta 2004-04-11]. Prieiga per internetą: < http://msdn.microsoft.com/library/ >
13. IBM Rational technologinių sprendimų svetainė [žiūrėta 2004-02-11]. Prieiga per internetą: < http://www.rational.com/ >
14. Šeinauskas R. Programų Inžinerijos modulio paskaitų skaidruolės. KTU. [žiūrėta 2004.04.12]. Prieiga per internetą: < http://www.elen.ktu.lt/~rsei/KTU/ >
15. Štuikys V. Programų priežiūra ir tobulinimas modulio paskaitų skaidruolės. KTU. [žiūrėta 2004.04.20]. Prieiga per internetą: < http://www.elen.ktu.lt/~rsei/KTU/ >
16. Katutis A. *Microsoft Visual C# .NET kodo optimizavimas* - Informacinės technologijos 2004.
17. Katutis A. *Microsoft Visual C# .NET Code optimization* - Estonian Winter School in Computer Science 2004.

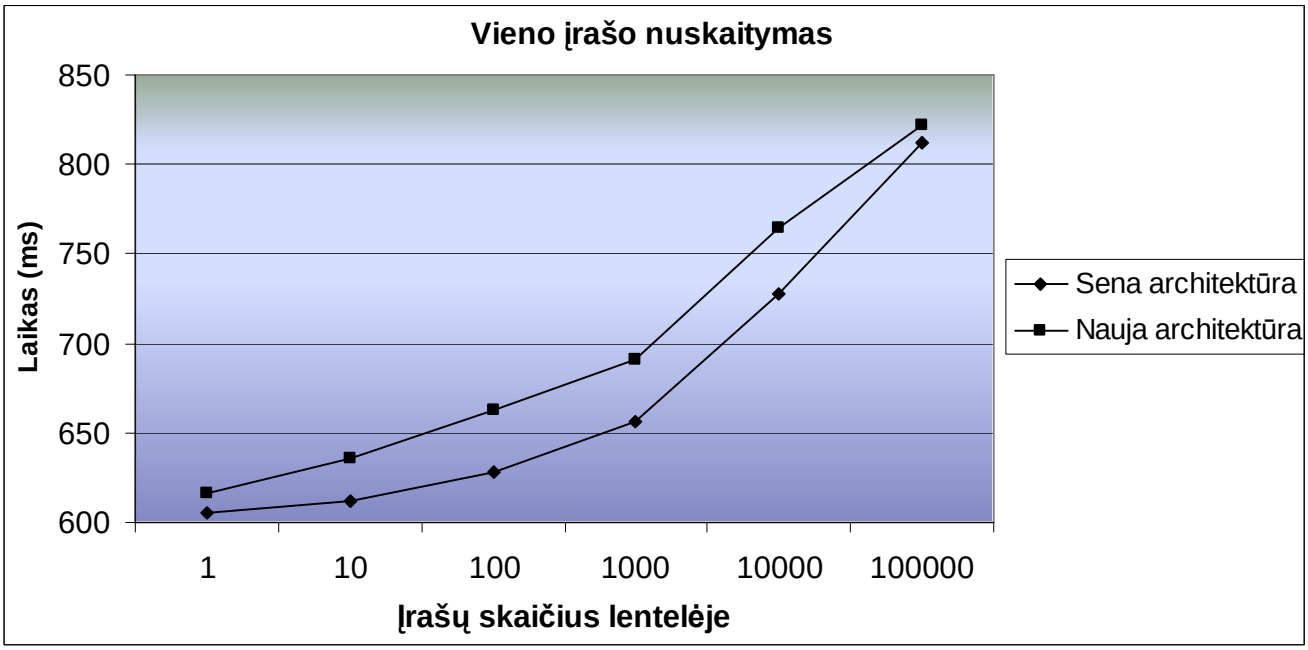
18. WinCVS failų versijų kontrolės sistemos svetainė [žiūrėta 2003.02.03].
    Prieiga per internetą: < http://www.wincvs.org/ >
19. Microsoft Visual SourceSafe failų versijų kontrolės sistemos svetainė [žiūrėta 2003.02.03].
    Prieiga per internetą: < http://msdn.microsoft.com/ssafe/ >

10 TERMINŲ IR SANTRUMPŲ SĄRAŠAS

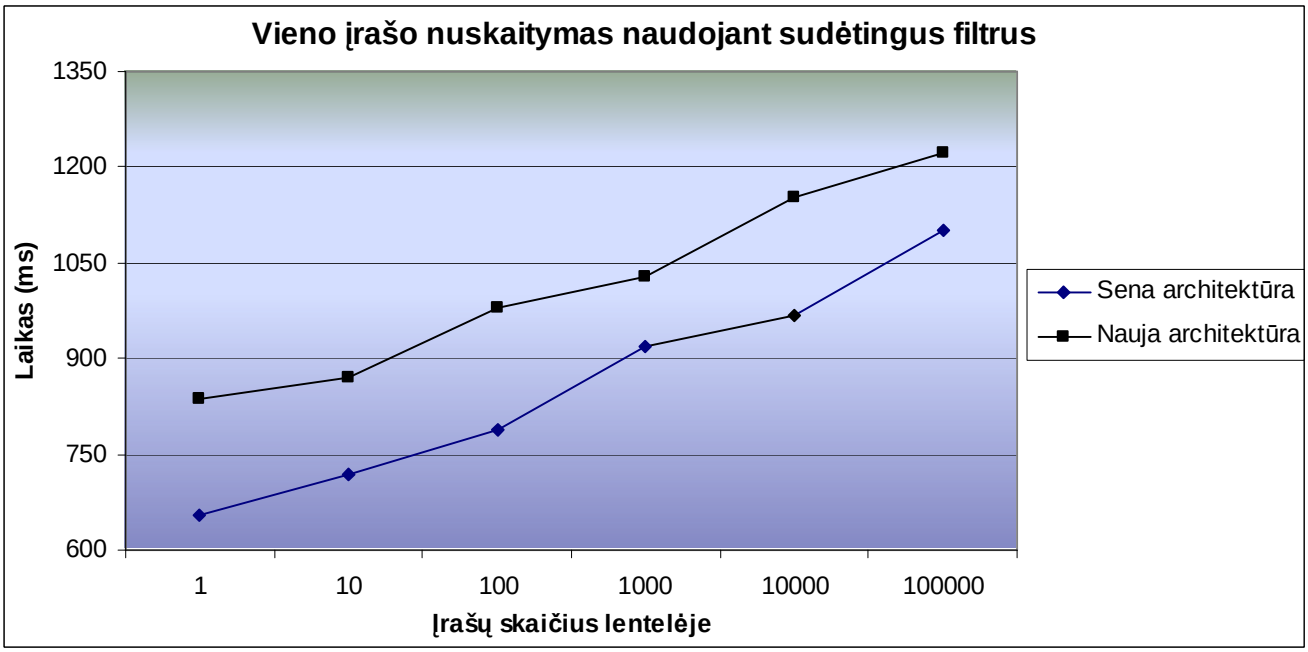
OS	Operacinė sistema
IS	Informacinė sistema
DBVS	Duomenų bazės valdymo sistema
PĮ	Programinė įranga
PĮP	Programinės įrangos priežiūra
PAT	Sukurtos programinės įrangos sutrumpintas pavadinimas („Proceed Asset Trading“)
Factory method	Šablonas, naudojamas kontekstinių biznio komponentų kūrimui

11 PRIEDAI

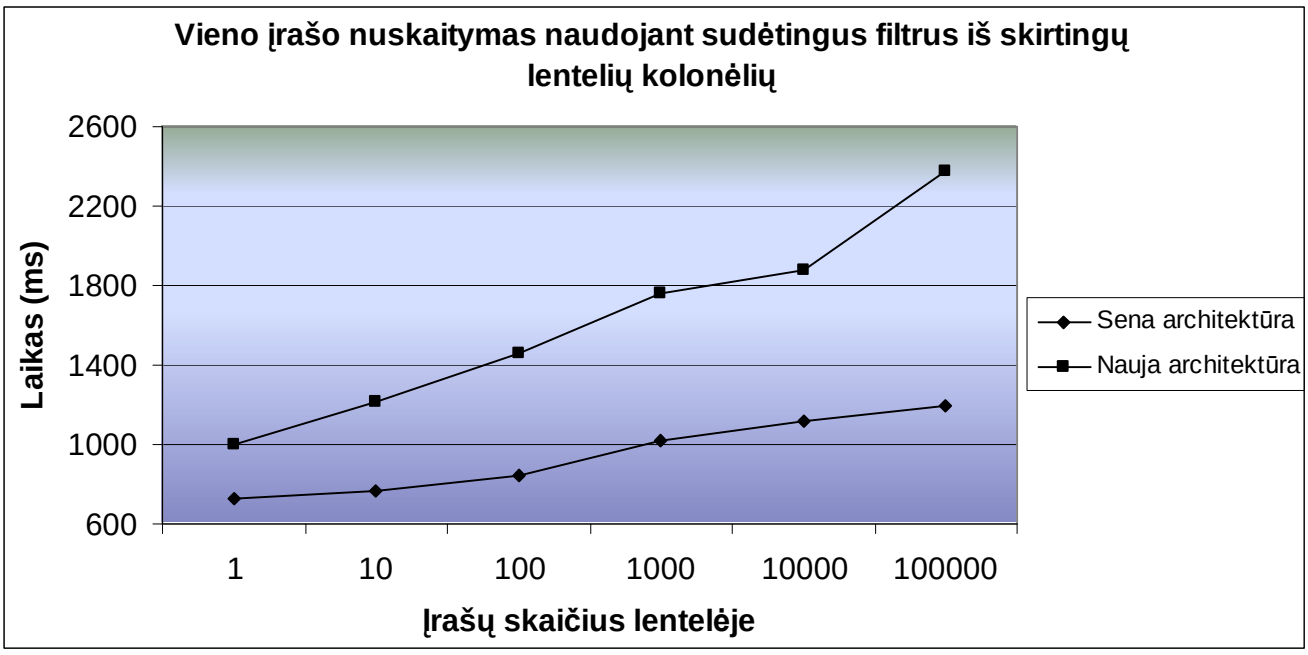
11.1 Priedas A – atliktų eksperimentų rezultatai



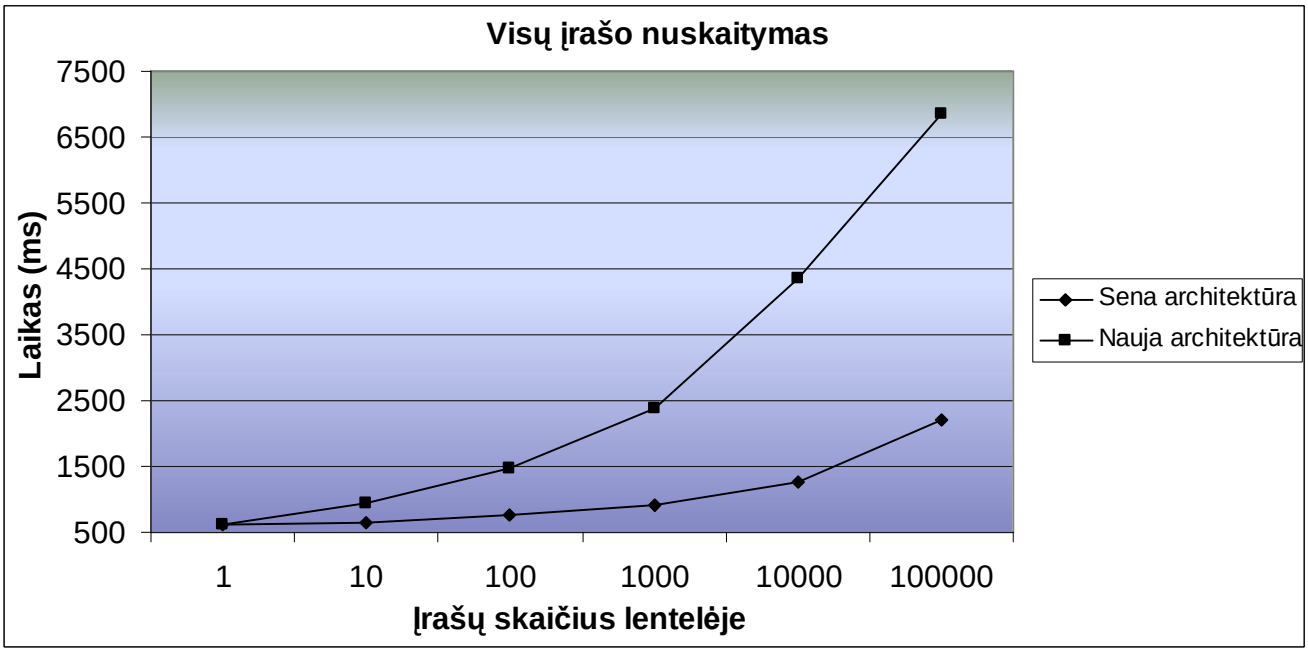
1 pav. Vieno įrašo nuskaitymas esant skirtingam įrašų skaičiui lentelėje



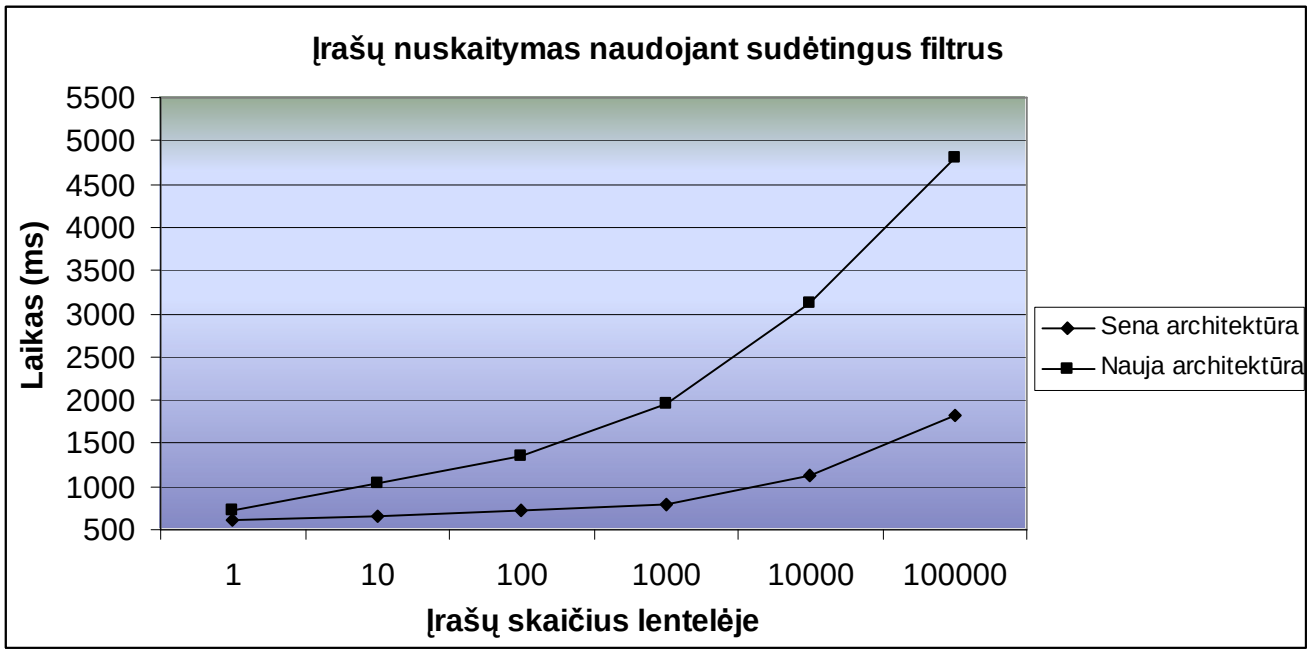
2 pav. Vieno įrašo nuskaitymas naudojant sudėtingus filtrus esant skirtingam įrašų skaičiui lentelėje



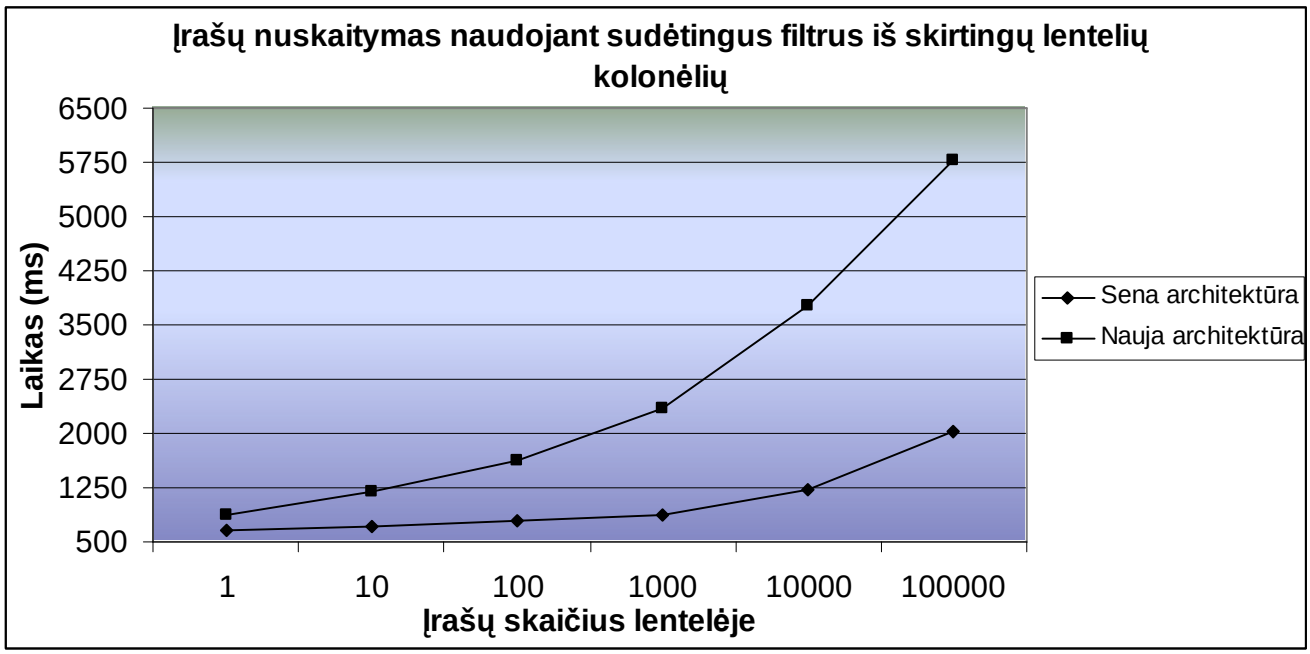
3 pav. Vieno įrašo nuskaitymas naudojant sudėtingus filtrus iš skirtingų lentelių kolonėlių esant skirtingam įrašų skaičiui lentelėje



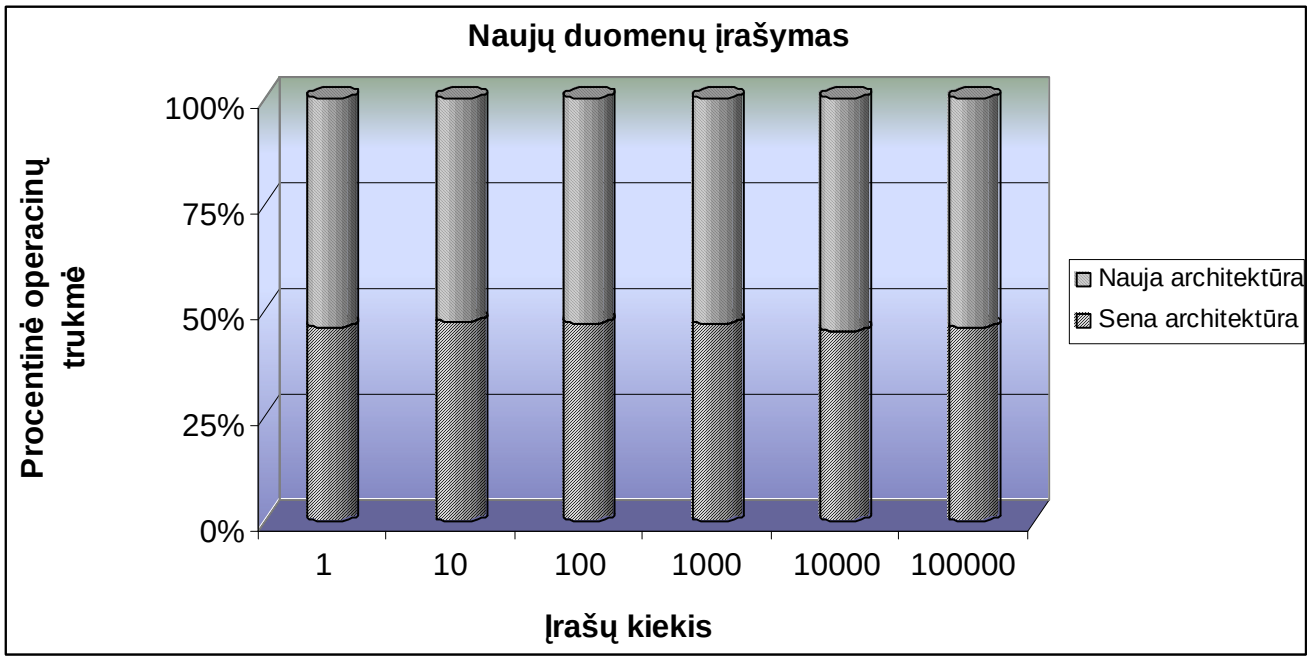
4 pav. Visų įrašų nuskaitymas esant skirtingam įrašų skaičiui lentelėje



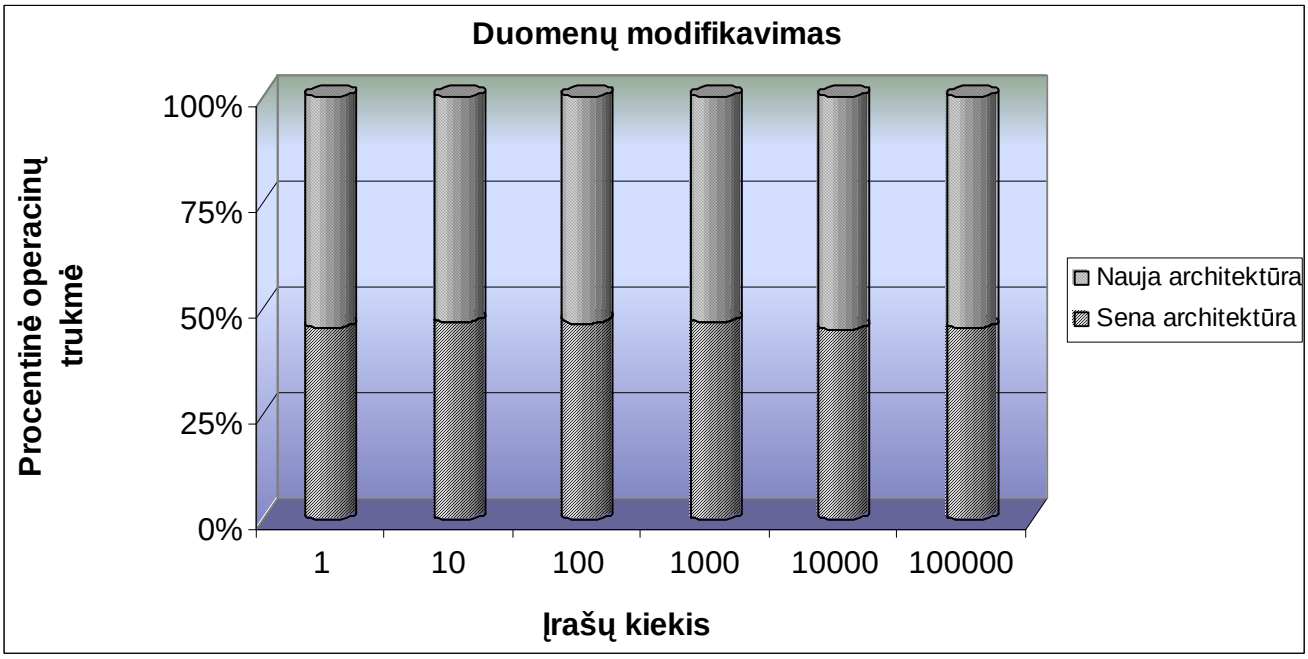
5 pav. Įrašų nuskaitymas naudojant sudėtingus filtrus esant skirtingam įrašų skaičiui lentelėje



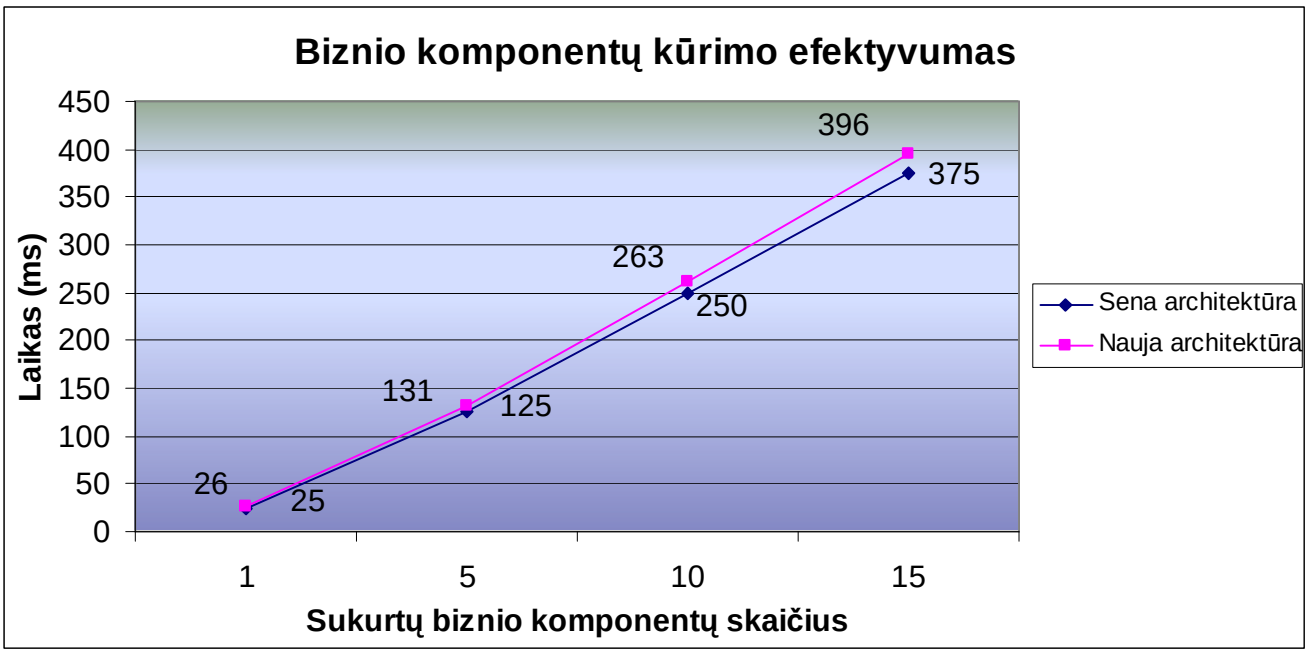
6 pav. Įrašų nuskaitymas naudojant sudėtingus filtrus iš skirtingų lentelių kolonėlių esant skirtingam įrašų skaičiui lentelėje



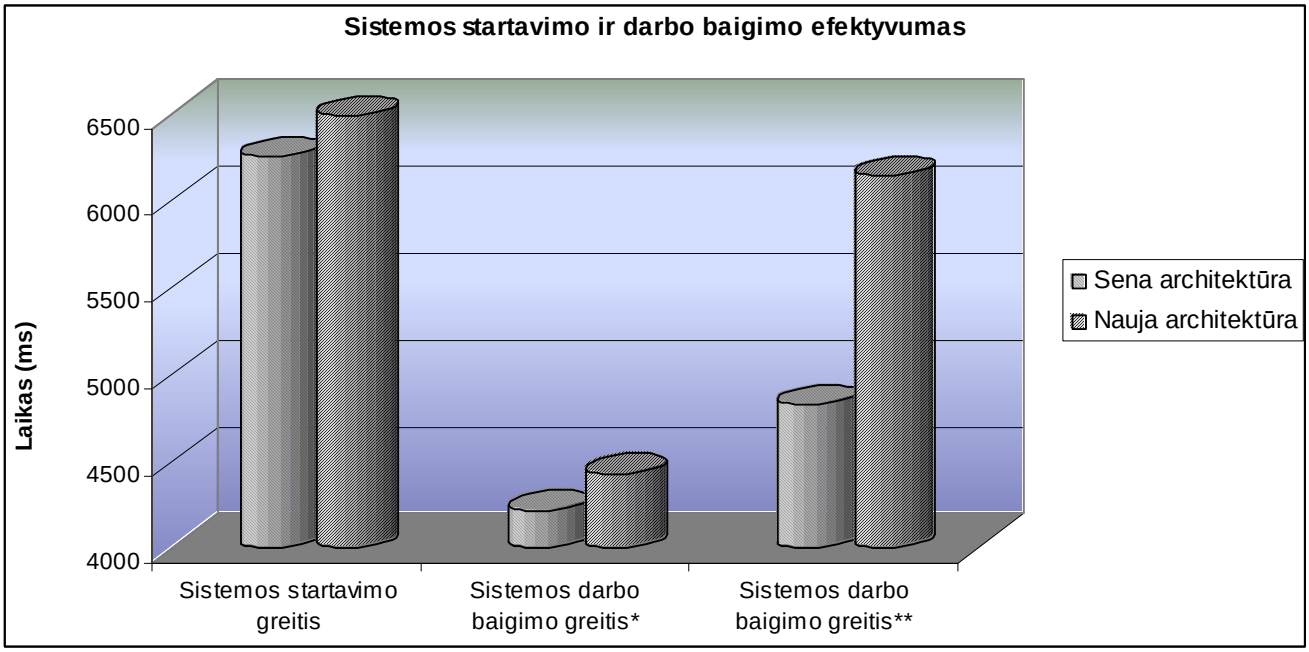
7 pav. Naujų duomenų įrašymo efektyvumo palyginimas



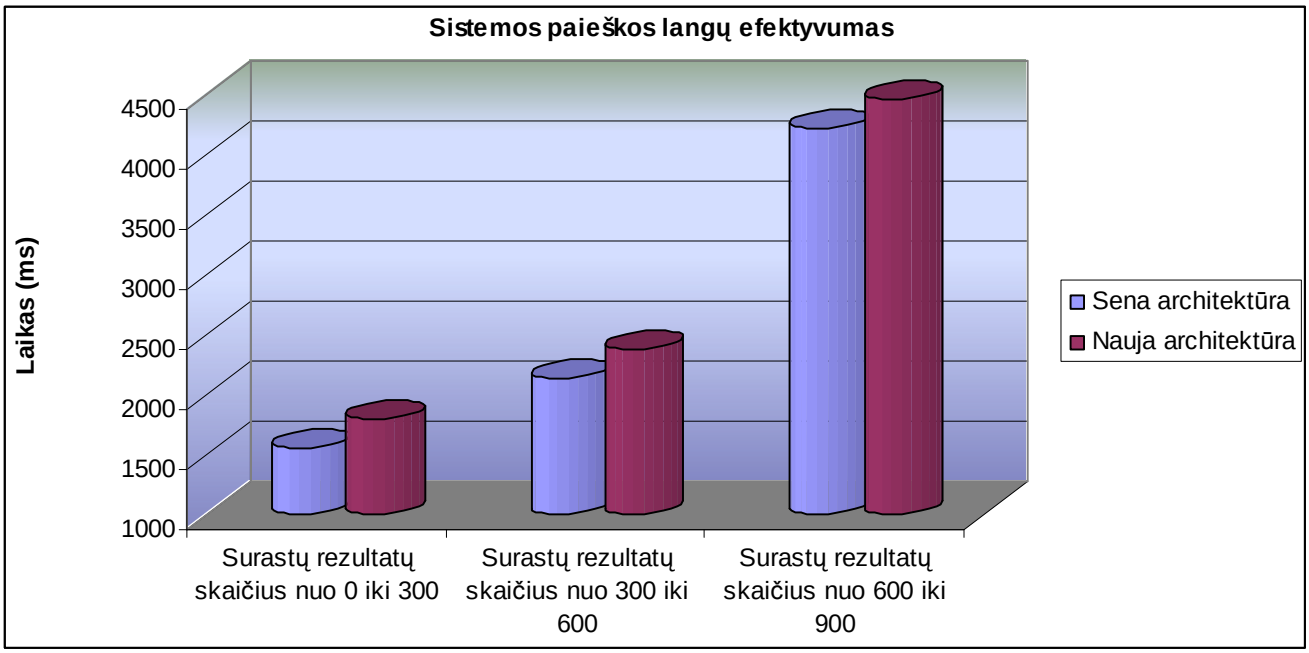
8 pav. Duomenų modifikavimo efektyvumo palyginimas



9 pav. Biznio komponentų kūrimo efektyvumas naudojant ir nenaudojant biznio komponentų kūrimo fabriko



10 pav. Sistemos prezentacinio lygmens startavimo ir darbo baigimo efektyvumas (* - prieš baigiant darbą nebuvo atliekami jokie veiksmai; ** - prieš baigiant darbą buvo vykdomos įvairiuos operacijos)



11 pav. Sistemos prezentacinio lygmens paieškos efektyvumas

11.2 Priedas B - Konferencijose publikuoti straipsniai

11.2.1 „Informacinės technologijos 2004“, 2004 m. sausis

11.2.2 “Estonian Winter School in Computer Science 2004”, 2004 m. kovas