KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ IŽINERIJOS KATEDRA

Rytis Ūsalis

DDD paremto duomenų sluoksnio modelio kūrimas ir tyrimas

Magistro darbas

Darbo vadovas:
doc. dr. Tomas Blažauskas

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ IŽINERIJOS KATEDRA

Rytis Ūsalis

DDD paremto duomenų sluoksnio modelio kūrimas ir tyrimas

Magistro darbas

Recenzentas:
prof. dr. Rimantas Butleris
2010-05

Vadovas:
doc. dr. Tomas Blažauskas
2010-05

Atliko:
IFM – 4/2 gr. stud.
Rytis Ūsalis
2010-05-31

Kaunas, 2010

DDD based data layer model implementation and research

RYTIS ŪSALIS

SUMMARY

The Conventional software developing process consists of several stages: requirements gathering, analysis, implementation, verification and deployment. Passing through these stages takes a lot of time. Automation of some stages could significantly speed up software development process. In Master Thesis introduced and developed data layer model lets designed and responsible for data entities transform into components. This step automates implementation, verification and deployment phases.

Experimental results showed that even designing a system with a small number of classes and using data layer model, process takes more than three times faster than time of developer. When number of classes increase, the time difference becomes much bigger.

Anotacija

Tradicinis programinės įrangos kūrimo procesas apima keletą žingsnių: reikalavimų surinkimas, analizė, įgyvendinimas, patikra, diegimas. Šiuos etapus pereiti užtrunka daug laiko. Automatizavus kelis šio proceso etapus stipriai pagreitėtų programinės įrangos kūrimas. Magistro darbe pasiūlytas ir sukurtas duomenų sluoksnio modelis leidžia suprojektuotas ir už duomenis atsakingas sistemos esybes transformuoti į komponentus. Šis žingsnis leidžia automatizuoti įgyvendinimo, patikros ir diegimo fazes.

Eksperimentiniais tyrimais parodoma, kad net esant nedideliam klasių skaičiui, projektuojant sistemą ir naudojantis duomenų sluoksnio modelio pagalba procesas užtruko daugiau nei trimis kartais trumpiau nei programuotojas. Klasių skaičiui didėjant, šis laiko skirtumas tampa dar didesnis.

Turinys

Lentelių turinys

Paveikslų turinys

1. ĮVADAS

Šiuo metu viena iš greičiausiai besivystančių mokslo ir pramonės šakų – informacinės technologijos. Ji skaidoma į keletą sričių, tačiau žinomiausia iš jų yra kompiuterių mokslas, kurios vėlgi žinomiausia sritis – programinės įrangos inžinerija. Ši sritis apima programinės įrangos modeliavimo, kūrimo bei kokybės užtikrinimo problemas.

Tradicinis programinės įrangos kūrimo procesas apima keletą žingsnių: reikalavimų surinkimas, analizė, įgyvendinimas, patikra, diegimas. Šiuos etapus pereiti užtrunka daug laiko. Šio proceso kai kurių etapų automatizavimas stipriai pagreitintų programinės įrangos kūrimą. Magistro darbe kuriamas duomenų sluoksnio modelis leistų suprojektuotas ir už duomenis atsakingas sistemos esybes transformuoti į komponentus. Tokiu atveju būtų automatizuotos įgyvendinimo, patikros ir diegimo fazės.

Magistro darbe tiek kiekybiniais, tiek ir kokybiniais kriterijais bus ištirtas duomenimis paremtos architektūros duomenų sluoksnio modelis. Eksperimento rezultatai apibendrinami, bus pateikiama atrasti pranašumai bei trūkumai, aprašomi galimi sistemos patobulinimai.

1.1. Paskirtis

Šio darbo paskirtis išanalizuoti probleminę duomenų sluoksnio modelio projektavimo bei įgyvendinimo sritį. Vėlesniuose etapuose jį realizuoti bei atlikti eksperimentus, pateikti savo vertinimus bei pasiūlyti patobulinimus.

1.2. Santrauka

Antrame skyriuje pateikiamas trumpas duomenimis paremtos architektūros modelio aprašymas, iškeliamos problemos ir suformuluojami magistro darbo tikslai ir uždaviniai, pateikiama priimto sprendimo pagrindimas. Trečioje dalyje apžvelgiamos bendros projektavimo ir diegimo įrankio problemos, iškeliama hipotezė, suformuluojami darbo tikslai ir uždaviniai, pažvelgiami panašūs sprendimai. Ketvirtame skyriuje pateikiama magistro darbo metu sukurtos programinės įrangos projektinės ir techninės dokumentacijos esminiai aspektai. Penktojoje dalyje pateikiama projektavimo ir diegimo įrankio esminė projekto dokumentacija. Šeštame skyriuje atliekamas sukurtos programinės įrangos tyrimas.. Septintoje dalyje atliekamas sukurtos bei įdiegtos programinės įrangos eksperimentinis tyrimas. Aštuntojoje dalyje atliekamas esminis

projektavimo ir diegimo įrankio greičio eksperimentas. Devintojoje glaustai išdėstomos išvados: projektavimo metu priimti sprendimai ir tyrimų bei eksperimentų rezultatai.

2. DUOMENŲ SLUOKSNIO MODELIO ANALIZĖ

2.1. Darbo tikslai ir uždaviniai

Pagrindinis darbo tikslas – sukurti duomenų sluoksnio variklį gebantį transformuoti modelį į komponentus.

- Išanalizuoti architektūrines posistemės problemas bei pateikti sprendimus;
- Išanalizuoti grafinio metamodelių aprašymo kalbas bei pasirinkti tinkamą sistemos realizavimui;
- Veikti nepriklausomai nuo taikymo srities;

2.2. Hipotezė

Modeliuojant programinę įrangą su projektavimo įrankiais ir ją transformuojant tiesiai į kodą, pasiekiamas žymiai didesnis programuotojų našumas, bei padidinamas veiklos efektyvumas.

2.3. Posistemės paskirtis

Duomenų sluoksnio modelio paskirtis suprojektuotas esybes transformuoti į konkrečioje aplinkoje veikiančius komponentus. Toks duomenų sluoksnio modelis leistų greitai sukurti ir pateikti suprojektuotą komponentą kitai posistemei bei užtikrintų, kad pateiktas komponentas yra korektiškas.

2.4. Projekto kūrimo pagrindimas

Kadangi jau realizuotoje sistemoje yra probleminis dalykas greitai atnaujinti komponento funkcionalumą, o taip pat reikalingas programuotojo įsikišimas pridedant objektui naujų savybių, todėl buvo nuspręsta realizuoti duomenų sluoksnio modelį, kuris būtų atsakingas už automatinę duomenų transformaciją iš modelio į veikiantį komponentą produkcijos aplinkoje.

2.5. Programinės įrangos modeliavimo metodų analizė

2.5.1. Modeliavimas ir jo svarba duomenų sluoksnio modelyje

Programinės įrangos modeliavimas – tai susitarta notacija aprašyta programų sistemos architektūra ir/arba jos veikimo principas.Kaip jau buvo minėta ankstesniame skyriuje (2.3) modelis reikalingas esamų ir naujų esybių aprašymui ir kad iš jo būtų galima automatiškai sukurti veikiančius komponentus.

2.5.2. Modeliavimo metodams keliami reikalavimai

Kadangi viso darbo esmė yra palengvinti ir pagreitinti visą programavimo bei diegimo procesą, todėl modeliavimo metodams keliami tokie reikalavimai:

- privalo turėti tiek tekstinę, tiek ir grafinę notaciją. Šis reikalavimas grindžiamas tuo, kad vartotojui yra patogiau ir aiškiau matyti modelį grafinėje notacijoje, o tekstinė notacija būtų apdorojama duomenų sluoksnio modelio.
- turi būti visuotinai pripažintas. Šis reikalavimas reikalingas grindžiamas tuo, kad modelio metodas būtų nuolat palaikomas ir laikui bėgant neišnyktų.
- privalo turėti mažiausiai 2 modeliavimo įrankius.
- turi būti orientuotas į objektinį programavimą. Šis reikalavimas grindžiamas tuo, kad projektuojant duomenų struktūras duomenų modelyje yra remiamasi objektiškai orientuoto programavimo principu, dėl to šis reikalavimas itin svarbus.

2.5.3. Reikalavimus atitinkantys modeliavimo metodai

Buvo surasta keletas modeliavimo metodų, tačiau analizei ir nagrinėjimui pasirinkti šeši populiariausi: *Objekto-vaidmenų modeliavimas*[1]*, Verslo procesų modeliavimo notacija, Vieninga modeliavimo kalba*[2]*, Srauto diagrama, Petri tinklai, Lepus3.*

Objekto-vaidmenų modeliavimas – grafinis modeliavimo metodas, tačiau gali būti aprašomas ir tekstiniu formatu, yra ypač populiarus aprašant duomenų bazių schemas bei ryšius tarp jų, tinka objektiškai orientuotoms programoms kurti. Šis modeliavimo metodas turi keletą įrankių, iš kurių populiariausi *VisioModeler, DogmaModeler, Visio for Enterprise Architects, CaseTalk.*

[1] Object-Role Modeling
[2] Unified Modeling Language (UML)

Verslo procesų modeliavimo notacija – šis grafinis modeliavimo būdas neturi tekstinės notacijos ir daugiau tinka procesams, o ne duomenų struktūroms, aprašyti, todėl jis nenaudojamas objektiniam orientuotų sistemų projektavimui. Įrankių yra daug, keletas iš jų: atviro kodo *BPMN2BPEL, AgilPro, MagicDraw*.

Vieninga modeliavimo kalba – vienas iš populiariausių grafinių modeliavimo būdų, turintis ir tekstinį modelio parašymą, tinkantis objektiškai orientuotų sistemų projektavimui. Rinkoje egzistuoja daug įrankių šiam modeliui apibrėžti: *MagicDraw, atviro kodo DIA, ArgoUML, Enterprise Architect, PowerDesigner, Microsoft Visio.*

Srauto diagramos notacija – ši grafinė notacija tinka modelio veiksmų scenarijams projektuoti, tačiau visiškai netinka duomenų struktūrų apibrėžimui, taip pat nėra tekstinės notacijos. Įrankių gausa didelė: *MagicDraw, SmartDraw, Edraw Max.*

Petri tinklų notacija – grafinė matematinė modeliavimo notacija, skirta paskirstytų sistemų projektavimui. Nėra tekstinio modelio aprašymo. Ši notacija netinka objektiniam duomenų struktūrų modeliavimui, tačiau pasižymi didele įrankių gausa. Rinkoje egzistuojantys sprendimai: *AlPiNA, Artifex, F-net, JPetriNet.*

LePUS3 – grafinė modeliavimo notacija, skirta projektuoti objektiškai orientuotoms sistemoms. Pasižymi didele raiška, tačiau neturi tekstinio modelio aprašymo. Nepavyko rasti nė vieno rinkoje egzistuojančio įrankio.

Surašius visus modeliavimo metodų reikalavimus į vertinimo lentelę (žiūrėti žemiau Lentelė Nr. 1 Modeliavimo metodų vertinimas) tolimesnei analizei atrinkti du: *Vieninga modeliavimo kalba* ir *Objekto-vaidmenų modeliavimas.*

Lentelė Nr. 1 Modeliavimo metodų vertinimas

Modeliavimo metodas	Tekstinė notacija	Grafinė notacija	Populiarumas	Įrankių gausa	Orientuotas į objektinį programavimą
Objekto-vaidmenų modeliavimas	✓	✓	✓	✓	✓
Verslo procesų modeliavimo notacija	×	✓	✓	✓	×
Vieninga modeliavimo kalba	✓	✓	✓	✓	✓
Srauto diagramos notacija	×	✓	✓	✓	×
Petri tinklų notacija	×	✓	×	✓	×
Lepus3	×	✓	×	✓	✓

2.5.4. Modeliavimo metodo pasirinkimas ir jo pagrindimas

Atrinkus iš keleto modeliavimo metodų du, nuspręsta detaliau išsiaiškinti jų privalumus ir trūkumus dėl galimo panaudojimo duomenų sluoksnio modelyje. Kiekvienas iš šių modeliavimo metodų bus vertinamas pagal žemiau apibrėžtus kriterijus:

- Griežtas (standartizuotas) modelio aprašymo tekstu formatas. Šis kriterijus yra itin svarbus tuo, kad nuo jo priklauso duomenų sluoksnio modelio darbo greitaveika bei projektavimo sudėtingumas. Idealiu atveju modeliavimo metodui egzistuoja vienas standartizuotas tekstinis formatas, nesunkiai apdorojamas algoritmiškai.
- Įrankių pasirinkimo įvairovė ir jų svoris. Šis kriterijus svarbus tuo, kad jis tiesiogiai veikia vartotojo darbo greitį bei patogumą. Idealiu atveju egzistuoja keletas nuolat palaikomų įrankių.
- Taisyklių ir apribojimų kalbų įvairovė ir svoris. Šis kriterijus yra svarbus dėl galimo duomenų sluoksnio modelio tobulinimo ateityje. Idealiu atveju egzistuoja viena, nesunkiai įsisavinama ir standartizuota kalba.
- Išraiškingumas. Šis kriterijus parodo modeliavimo metodo aprašymo išraiškingumą t.y. ką tuo modeliu norima pasakyti. Idealiu atveju modeliavimo metodas gali nusakyti bet kokią ypatybę tiesiogiai susijusią su dalykine sritimi[1].
- Aiškumas. Šis kriterijus parodo kiek modelį aprašančios išraiškos yra suprantamos ir panaudojamos. Idealiu atveju metodo aprašymo aiškumas turėtų būti

nedviprasmiškas ir tekstinės bei grafinės išraiškos intuityviai suprantamos ir lengvai prisimenamos[2].

- Semantinis stabilumas. Šis kriterijus parodo kaip gerai modeliai išreiškiami ir kiek stabiliai išlaiko originalią savo prasmę pasikeitus taikymui. Kuo daugiau priverstinių keitimų daroma modelyje dėl pasikeitusio taikymo, tuo semantinis stabilumas yra mažesnis[2].
- Semantinis relevantiškumas. Šis kriterijus reikalauja, kad tik abstrakčios savybės būtų modeliuojamos. Bet kokie pertekliniai aspektai (pvz. išankstiniai realizavimo būdų pasirinkimai, reikalavimai techninės įrangos greitaveikai) turėtų būti vengiami[2].
- Patikros mechanizmas. Šis kriterijus apsprendžia srities specialistų galimybes patikrinti modelio atitikimą taikymui. Pavyzdžiui, statinės sistemos galimybės galėtų būti tikrinamos verbalizacijos arba daugkartinės inicializacijos metodu, dinaminės galimybės galėtų būti tikrinamos simuliacijos metodu[2].
- Abstrakcijų mechanizmas. Šis kriterijus svarbus tuo, kad leidžia modelio metode paslėpti nenorimas matyti detales. Tai ypač svarbu sudėtinguose modeliuose[3].

Lentelė Nr. 2. Modeliavimo metodų palyginimas

Modeliavimo metodas / **Kriterijus**	***Objekto-vaidmenų modeliavimo notacija***	***Vieninga modeliavimo kalba***
Standartizuotas tekstinis formatas	Vieningas formatas neegzistuoja	Egzistuoja tekstinis standartizuotas *metaduomenų apsikeitimo formatas*[3].
Įrankių pasirinkimo įvairovė ir jų svoris	Egzistuoja daugybė įrankių: Didžiausią svorį turintys įrankiai: • Visio for Enterprise Architects (VEA)	Egzistuoja daugybė įrankių. Didžiausią svorį turintys įrankiai: • MagicDraw • IBM Rational Software
Taisyklių ir apribojimų kalbų įvairovė ir svoris	Egzistuoja keletas kalbų: • RIDL • PSM • ConQuer	Egzistuoja viena standartizuota kalba: • OCL[4]
Išraiškingumas	Didelis, panaudojus papildomus išplėtimus.	Didelis per panaudos atvejų, elgesio ir įgyvendinimo diagramas.
Aiškumas	Didelis	Vidutinis
Semantinis stabilumas	Didelis	Vidutinis
***Semantinis* relevantiškumas**	Didelis	Didelis
Patikros mechanizmas	Turi	Turi
Abstrakcijų mechanizmas	Turi	Turi

Lyginant modeliavimo metodus pagal „*Standartizuotas tekstinis formatas*“ kriterijų *Objekto-vaidmenų modeliavimo notacija* šioje srityje atsilieka, kadangi neturi jį aprašančio vieningo formato. Tiesa, egzistuoja pasiūlymas kaip būtų galima šį metodą aprašyti XML kalba [4], tačiau jis kol kas yra tik rekomendacinio pobūdžio. Skirtingai nei *Objekto-vaidmenų modeliavimo notacija, Vieninga modeliavimo kalba* turi standartizuotą *metaduomenų apsikeitimo* formatą sukurtą XML pagrindu ir kuris aktyviai palaikomas W3C. Įvertinus abu modeliavimo metodus pagal antrąjį kriterijų („*Įrankių pasirinkimo įvairovė ir jų svoris*“), geriau pasirodė *Vieninga modeliavimo kalba,* kadangi ji turi daugiau galingų įrankių, kuriuos nuolat tobulina tokios žinomos kompanijos kaip IBM, No Magic Inc. Pagal trečiąjį kriterijų „*Taisyklių ir apribojimų kalbų įvairovė ir svoris*“ įvertinus abu metodus, čia taip pat nepralenkiama buvo *Vieninga modeliavimo kalba*, kadangi turi IBM sukurtą deklaratyvią apribojimų kalbą (OCL)[5]. *Objekto-vaidmenų modeliavimo notacija* taip pat turi keletą apribojimo kalbų (pvz. RIDL, PSM,

[3] Extensible markup language (Išplėsta žymėjimo kalba)
[4] Object Constraint Language (OCL)

ConQuer) iš kurių žinomiausia ConQuer[6], tačiau dėl matematinės notacijos yra sudėtingesnė nei OCL. Lyginant modeliavimo metodus pagal „*Išraiškingumas*“ kriterijų *Objekto-vaidmenų modeliavimo notacija* nusileido tuo, kad jis pasiekiamas per papildomas notacijos išplėtimus, o *Vieningoje modeliavimo kalboje* jis pasiekiamas per panaudos atvejų, elgesio bei įgyvendinimo diagramas. Įvertinus abu modeliavimo metodus pagal kriterijų „*Aiškumas*“ *Objekto-vaidmenų modeliavimo notacija* yra žymiai pranašesnė prieš konkurentą dėl savo panašumo į natūralią kalbą. Kriterijaus „*Semantinis stabilumas*“ atžvilgiu *Vieningoji modeliavimo kalba* nusileidžia savo konkurentui dėl to, kad modelio notacijoje objektų savybės apibrėžiamos atributais, kurie neaišku kokiais semantiniais ryšiais susiję su objektu. Įvertinus abu metodus pagal kitus tris kriterijus „*Semantinis relevantiškumas*“, „*Patikros mechanizmas*“, „*Abstrakcijų mechanizmas*“ metodai yra panašūs dėl to metodo pasirinkimui įtakos neturės.

Apžvelgus abu modeliavimo metodus, išryškėjo tiek vieno, tiek kito privalumai ir trūkumai. Nepaisant *Objekto-vaidmenų modeliavimo notacijos* pranašumų dėl aiškumo, semantinio stabilumo ir semantinio relevantiškumo, modeliui aprašyti buvo pasirinkta *Vieningoji modeliavimo kalba*. Šį pasirinkimą įtakojo keletas aspektų: didelis rinkoje pirmaujančių kompanijų palaikymas (IBM, No Magic Inc., Microsoft), didelė įvairovė įrankių, galimybė plėsti duomenų sluoksnio modelį pasitelkus standartizuotą apribojimų kalbą (OCL).

2.6. Iškilusios problemos ir pasiūlyti sprendimo būdai

Siekiant išspręsti darbo vietos mobilumo problemą buvo nuspręsta grafinį scenarijų įrankį realizuoti žiniatinklyje. Šį žingsnį lėmė per paskutinį dešimtmetį išaugusi interneto skvarba bei smarkiai patobulėjusios žiniatinklio technologijos. Tiesa, toks pasirinkimas ne tik išsprendė mobilumo problemą, tačiau iškėlė papildomų problemų: kaip suprojektuoti paskirstytą sistemą, kokios technologijas naudoti, kaip užtikrinti greitą duomenų apsikeitimą tarp posistemių.

Kadangi duomenų sluoksnio modelis turi atlikti modelio transformavimą į veikiančius komponentus (resursams imli operacija), nuspręsta jo visas komponentes laikyti vienoje tarnybinėje stotyje.

Ankstesniame skyriuje (2.6) buvo analizuojama kokį modeliavimo metodą pasirinkti modeliui sudaryti - pasirinkta modeliavimo kalba UML. Ši kalba turi savo tekstinį užrašymo formatą XMI[7]. Nors šis formatas yra standartas, tačiau skirtingos kompanijos (IBM ir MagicDraw) savaip interpretuoja kaip jį naudoti modeliui užrašyti. Dėl šios priežasties suprojektavus tą patį modelį su IBM Rational Rose ir NoMagic Inc. MagicDraw modelių

rinkmenos nesutampa. Kadangi Kauno Technologijos Universitete plačiai taikomas modeliavimo MagicDraw įrankis, todėl modeliui transformuoti naudosime būtent šio įrankio formatą.

Kadangi posistemės yra silpnai susietos ir gali būti pakankamai nutolusios viena nuo kitos, iškilo problemos: kaip užtikrinti pakankamai greitą duomenų apsikeitimą bei kokiu protokolu bus vykdomas duomenų apsikeitimas. Pirmąją problemą sprendžia interneto pralaidumo didinimas ir HTTP glaudinimo protokolo naudojimas. Antrąją problemą sprendžia interneto žiniatinklio paslaugų protokolas SOAP, kuris užtikrina suderinamumą protokolų lygiu. Žiniatinklio paslaugų kūrimui bus naudojama .NET karkaso taikomųjų programų kūrimo sąsają WCF. Ši sąsaja užtikrina greitą kūrimo procesą bei lengvą diegimą.

Kadangi duomenų sluoksnio modeliui kurti naudojamos Microsoft technologijos, tai modelio transformavimo į komponentus posistemės realizacijai bus panaudota C# 3.0 versijos programavimo kalba. Ši moderni objektinė programavimo kalba turi didelę raišką, kodo pernešamumą, palyginti lengva išmokti, palaiko internacionalizaciją [8].

3. PROJEKTAVIMO IR DIEGIMO ĮRANKIO ANALIZĖ

3.1. Problema

Tradicinis programinės įrangos kūrimo ir diegimo procesas, kai iš pradžių suprojektuojama užduotį sprendžianti sistemos architektūra, o vėliau programuojamas jos funkcionalumas bei kuriama duomenų saugojimo infrastruktūra yra lėtas, sudėtingas ir daugeliu atžvilgiu neefektyvus procesas. Jeigu sistema yra didelės apimties, toks programinės įrangos kūrimas ne tik didina klaidų atsiradimo tikimybę ir komplikuoja sistemos testavimą, bet ir sukelia papildomų rūpesčių diegiant sistemą vartotojui.

Didelės sistemos reikalauja kitokio, pažangesnio, labiau automatizuoto projektavimo, programavimo ir diegimo proceso. Tokio, kuris pasirūpintų automatiniu duomenų sluoksnio (duomenų bazės, žiniatinklio paslaugų) sukūrimu, sprendimo pateikimu keliomis technologijomis (programavimo kalbomis, skirtingomis architektūrinėmis realizacijomis) bei integruota kūrimo aplinka, leidžiančia vizualiai kurti sprendimo panaudos atvejus (scenarijus).

3.2. Hipotezė

Perėjus nuo tradicinio sprendimo kūrimo tekstiniu redaktoriumi (programavimo) prie duomenimis paremto projektavimo, kai panaudos atvejai kuriami scenarijais grafiniame

redaktoriuje, o duomenų infrastruktūra generuojama remiantis modeliu, pavyktų pasiekti didesnį sprendimo vystymo efektyvumą.

Pasiūlyta lanksti architektūra taip pat leistų sprendimo kodą generuoti skirtingomis technologijomis.

3.3. Darbo tikslai ir uždaviniai

Pagrindinis šio darbo tikslas yra sukurti projektavimo ir diegimo įrankio modelį, kuris palengvintų ir paspartintų programinės įrangos kūrimą.

Siūlomas architektūrinis modelis turėtų:

1) grafiškai modeliuoti scenarijus (panaudos atvejus);
2) generuoti duomenų infrastruktūrą (duomenų bazę, žiniatinklio paslaugas);
3) generuoti kodą skirtingoms programavimo kalbomis (architektūrinėms realizacijoms);
4) veikti sistemos architektūros modelio pagrindu (klasių diagramos);
5) veikti nepriklausomai nuo taikymo srities;
6) palaikyti dažniausiai naudojamus projektavimo standartus;

3.4. Darbo metodai ir priemonės

Teoriniai tyrimai atlikti panaudojant metodus, sąvokas ir kitas žinias iš informatikos, matematikos bei programavimo teorijos.

3.4.1. Grafinių modelių redaktorius

Posistemė sukurta „Microsoft Visual Studio 2008“ kūrimo aplinkoje naudojant „Microsoft Silverlight“ karkasą.

3.4.2. Kodo generatorius

Posistemė realizuota PHP programavimo kalba ir talpinama dedikuotame serveryje. Kūrimo aplinka – „NuSphare PhpED 5.9 profesional“.

3.4.3. Duomenų sluoksnio modelis

Posistemė įgyvendinta Microsoft Visual Studio 2008 programų kūrimo aplinkoje, panaudojant .NET Framework 3.5 SP1. Posistemė įdiegta į Windows 7 operacinę sistemą su MS SQL Server 2008 ir Internet Information Service 7.0.

3.4.4. Hipertekstinės grafinės vartotojo sąsajos biblioteka

Posistemė sukurta deklaratyvia XML (HTML) bei imperatyvia JavaScript programavimo kalba, naudojant Microsoft Visual Studio 2008 integruotą programų kūrimo aplinką.

3.5. Panašių sprendimų apžvalga

Šiuo metu analogiškų projektavimo įrankių pasaulyje nėra. Tačiau galima apžvelgti panašius įrankius, kurie palaiko Modeliu Paremtos Architektūros[5] principus. Populiariausi ir daugiausiai naudojami yra šie:

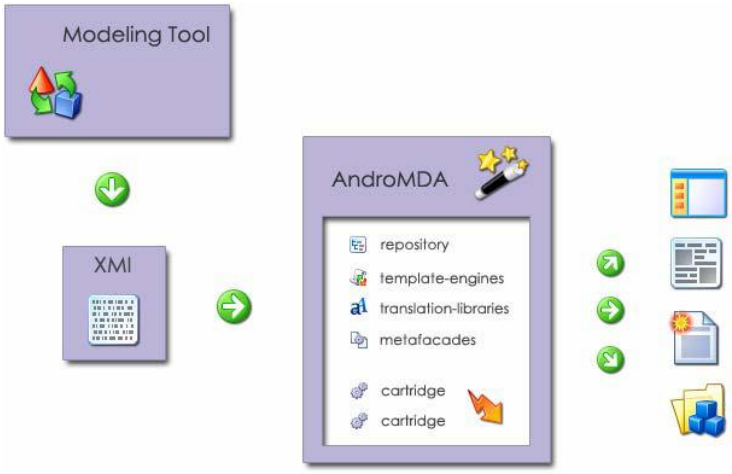
- „AndroMDA“
- „Rational XDE MDA Toolkit“
- „ArcStyler 4.0“
- „OpenMDX“
- „OptimalJ“

3.5.1. „AndroMDA“

„AndroMDA[9]“ yra atviro kodo programinė įranga. Šiuo įrankiu galima generuoti J2EE[6] projektus, realizuotus Java programavimo kalba iš UML modelių. Generuojami žiniatinklio projektai pritaikyti Hibernate, EJB, Struts, Spring ir žiniatinklio paslaugų technologijoms.

Principinė „AndroMDA“ veikimo schema pavaizduota paveikslėlyje (Pav. 1. „AndroMDA“ veikimas).

[5] Model Driven Engineering (MDA) – modeliu paremta architektūra
[6] J2EE - standartinė daugialyčių programų kūrimo Java kalba platforma



Pav. 1. „AndroMDA“ veikimas

Programų sistemos generavimas naudojant „AndroMDA“ veiksmų seka:

- UML įrankiu („MagicDraw“, „Poseidon UML“, „Rational Rose“) sukuriamas sistemos nuo Platformos Nepriklausomas Modelis[7] ir eksportuojamas į XMI dokumentus.
- Iškviečiama „AndroMDA“ programa kuri atlieka reikalingas transformacijas.

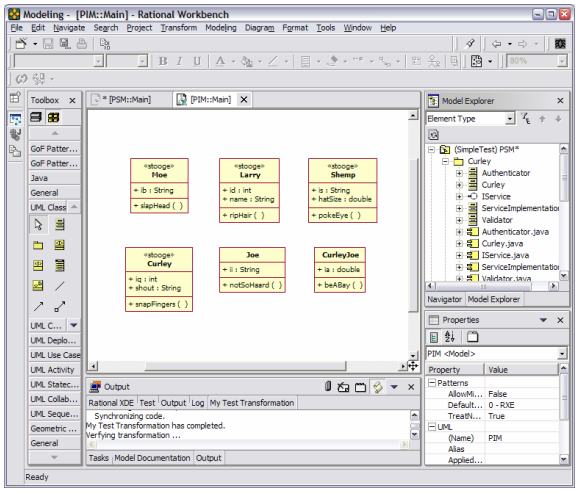
Transformacijos aprašomos įskiepių[8] sistema. Todėl vartotojas turi galimybę kurti savo transformacijos modelius. Šie modeliai aprašomi šabloniniu principu Java programavimo kalba.

Darbas su įrankiu vyksta terminalo[9] pagalba, todėl valdymas labai nepatogus, tačiau kaip kūrėjai teigia itin lankstus.

3.5.2. „Rational XDE MDA Toolkit“

„Rational XDE MDA Toolkit“ yra „Rational Rose XDE“ papildinys. Tai yra tiesiog modeliavimo įrankio papildinys leidžiantis atlikti kodo generavimą iš UML modelių. Bendrą paketą sudaro modeliavimo įrankis, MDA transformacijų įrankis ir kodo generavimo įrankis (Pav. 2. „Rational XDE MDA Toolkit“ vaizdas).

[7] Platform Independent Model (PIM) – nuo platformos nepriklausomas modelis
[8] Cartridge – įskiepis
[9] Console – terminalas



Pav. 2. „Rational XDE MDA Toolkit“ vaizdas

Įsidiegus „Rational XDE MDA Toolkit“ įrankį pastebėta kad nėra paruoštų transformacijų bibliotekų tarp modelių, jas reikia pasiruošti patiems. Šis įrankis nėra iki galo išbaigtas.

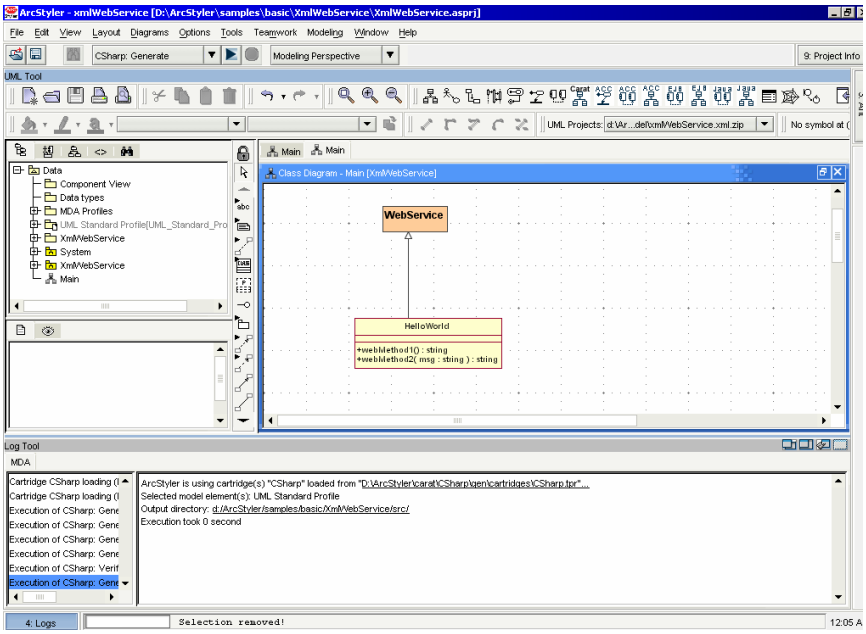
3.5.3. „Arcstyler 4.0“

Tai vienas iš labiausiai išvystytų MDA įrankių. Šis įrankis naudoja „MagicDraw“ kaip modeliavimo variklį. „Arcstyler“ leidžia kurti verslo modelius, UML modelius, įgyvendinimo kodą. Įrankyje taip pat integruotas programinės įrangos vystymo bei testavimo įrankis. „Arcstyler“ turi ypatingą įskiepių sistemą „MDA-Cartridges“, kurioje galima talpinti funkcionalumą reikalingą transformacijoms modelis-modelis arba modelis-infrastruktūra.

„Arcstyler“ turi galimybę susieti objektus su verslo procesais. Pagrindiniai šio įrankio privalumai:

- Sumažinti kūrimo proceso laiką. Kūrimo proceso laikas sumažėja dėl naudojamo vizualaus modeliavimo bei kodo generavimo.
- Aukštesnis kokybės laipsnis – aiškios architektūros reikalavimas, realus sistemos proceso dokumentavimas, iš anksto nustatytas testavimas ir patikra dėl atitikimo specifikacijai.
- Lankstus valdymas – greitas ir lengvas pakeitimų valdymas leidžia sumažinti kaštus bei laiką.
- Didelis plečiamumas – modeliavimas ir kodo generavimas gali būti lengvai pritaikomas konkretiems poreikiams.

- MDA pritaikymas sukurtoms programoms – automatinis apgrąžos inžinerijos pritaikymas Java programoms.



Pav. 3. „Arcstyler 4.0“ vaizdas

3.5.4. „OpenMDX“

„OpenMDX“ yra atviro kodo MDA principus atitinkanti programinė įranga. Šis įrankis pritaikytas iš UML modelių generuoti Java kalba realizuotus projektus. Įrankyje realizuotos transformacijos vadinamos įskiepiai. Šių įskiepių sistemos pagalba vartotojas gali kurti savo transformacijas.

Pagrindiniai privalumai:

- Atviras kodas.
- Galimybė kurti plečiamas organizacijų sistemas.
- MDA principų taikymas .
- J2SE[10], J2EE, CORBA[11], .NET[12] platformose.
- Žinomiausių modeliavimo įrankių palaikymas („Rational Rose“, „Poseidon UML“, „MagicDraw“).

[10] J2SE – platforma naudojama JAVA programoms kurti
[11] CORBA – standartų sistema koordinuotam kelių programų darbui internete.
[12] .NET - Microsoft Windows operacinės sistemos komponentas

- Aspektais paremto programavimo palaikymas.

3.5.5. „Optimal J“

„Optimal J“ suteikia paprastą ir palyginti lengvą būdą sukurti paskirstytą Java programą be kūrėjų įtraukimo į sudėtingą J2EE architektūrą. Kitaip tariant, žmonių dėmesys sutelkiamas į tai ką sukurti, o ne kaip tai padaryti.

„Optimal J“ modeliu paremtas metodas įgalina lengvai sukurti vizualų programos modelį. Kūrimas modeliu paremtoje aplinkoje teikia keletą privalumų:

- Aukšto lygio programos peržiūra.
- Pakartotinis objektų ir taisyklių panaudojimas.
- Derinimas kūrimo metu.

Vartotojo apibrėžtomis verslo taisyklėmis paremtas „Optimal J“ leidžia kūrėjams pritaikyto atskiriems vartotojams programas greičiau. Naudojant šį redaktorių, galima įtraukti ir statines ir dinamines veiklos taisykles į atitinkamą modelio lygį. Dinaminės veiklos taisyklės yra saugomos taisyklių bazėje, kas leidžia daryti pakeitimus programoje viso kūrimo metu – be programos kodo keitimo. „Optimal J“ paverčia veiklos taisykles į Java kodą ir jį realizuoja atitinkamame programos taške.

„Optimal J“ centre yra projektavimo ir realizavimo karkasai, vadinami šablonais[13]. „Optimal J“ naudoja šablonus visos vykdymo programos kodo generavimui. Šablonai įtraukia geriausius kodo pavyzdžius į J2EE specifikacijas.

„Optimal J“ sinchronizuoja Java kodą su programos modeliu, taigi modelis tiksliau vaizduoja programą bet kuriuo metu. Tai leidžia pakeisti programą lengvai modifikuojant elementus bet kuriame modelio lygyje. „Optimal J“ užtikrina, kad visi pakeitimai bus suderinti su esančia programos architektūra.

3.5.6. MDA įrankių įvertinimas

Įvertinsime aukščiau aprašytus modeliavimo įrankius pagal šiuos kriterijus:

[13] Pattern – šablonas

- PIM palaikymas. Ar yra galimybė sudaryti PIM modelį.
- PSM[14] palaikymas. Ar yra galimybė sudaryti PSM modelį.
- Ar gali generuoti skirtingus PSM. Ar įrankis leidžia generuoti įvairius PSM iš to pačio PIM modelio.
- Modelių integracija. Galima dirbti su keliais modeliais iš kurių generuojama viena programa.
- Programinės įrangos vystymas. Įrankis palaiko programinės įrangos vystymą.
- Bendravimas su kitais įrankiais. Įrankis gali bendrauti su kitais įrankiais, importuoti eksportuoti modelius.
- Galimybė kurti savo transformacijas. Įrankis leidžia kurti savo transformacijas, t.y. nepasitenkinama tik paruoštomis transformacijomis.
- Korektiškumas. Įrankis teikia priemonę patikrinti modelių korektiškumą, ar jie atitinka nustatytas taisykles.
- Išraiškingumas. Įrankis pateikia pakankamai išraiškingas PIM, PSM notacijų priemones, kuriomis galima pilnai išreikšti sistemos modelį.
- Šablonai ir apibendrinimai. Įrankis leidžia kurti modelio elementų šablonus.
- Keitimo[15] palaikymas. Pakeitimai PIM perduodami iš kart į PSM ir atvirkščiai.
- Ryšiai tarp modelių. PIM, PSM ir kodo modeliai išlaiko ryšius.
- Programinės įrangos gyvavimo ciklo palaikymas. Įrankis leidžia palaikyti programinės įrangos kūrimo ciklą (analizė, projektavimas, testavimas, diegimas, palaikymas).
- Standartizacija. XMI, UML palaikymas.
- Transformacijų kryptis iš PIM į PSM. Ar yra galimybė iš PIM modelio generuoti PSM modelius.
- Transformacijų kryptis iš PSM į kodą. Ar yra galimybė iš PSM modelio generuoti programos kodą.
- UML įrankis naudojamas vidinis. Įrankis naudoja vidinį modeliavimo įrankį.
- UML įrankis naudojamas „MagicDraw“. Įrankis priima modelius iš „MagicDraw“ modeliavimo įrankio.
- UML įrankis naudojamas „Rational Rose“. Įrankis priima modelius iš „Rational Rose“ modeliavimo įrankio.

[14] Platform Specific Model (PSM) – nuo platformos priklausomas modelis
[15] Refactoring – keitimas

- UML įrankis naudojamas „Poseidon UML“. Įrankis priima modelius iš „Poseidon UML“ modeliavimo įrankio.

UML įrankių įvertinimo rezultatai pateikiami lentelėje (Lentelė Nr. 3. UML įrankių įvertinimo rezultatai).

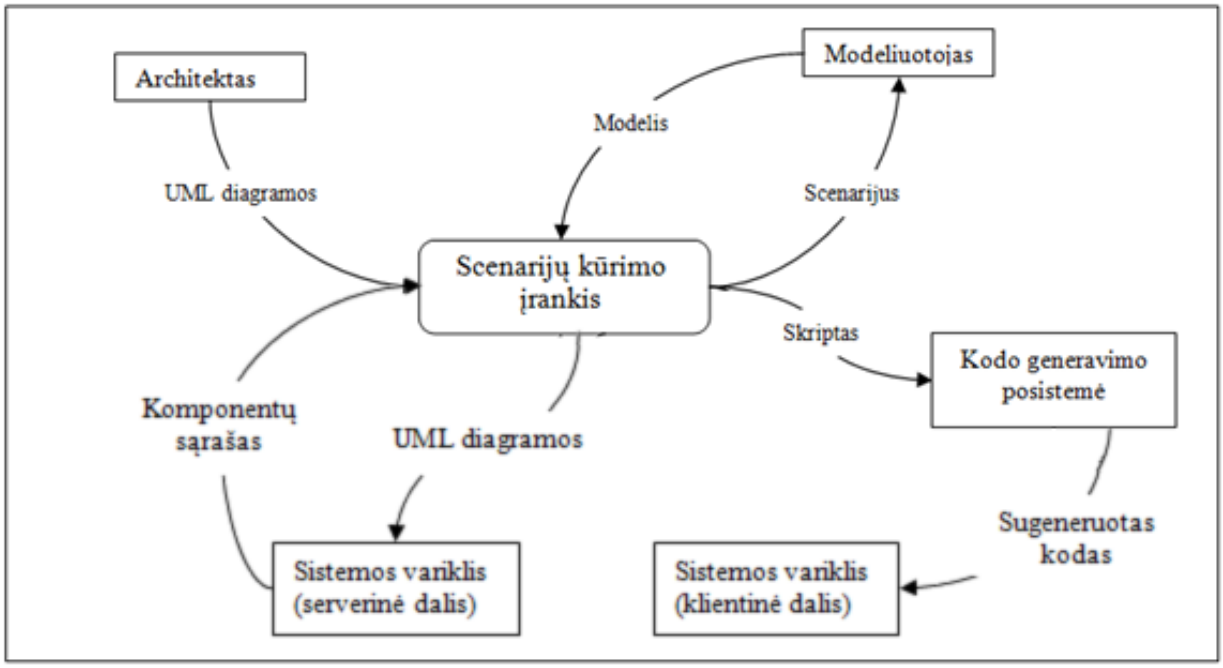
Lentelė Nr. 3. UML įrankių įvertinimo rezultatai[10]

Kriterijus	Andro MDA	Rational XDE MDA Toolkit	ArcStyler	Open MDX	Optimal J
PIM palaikymas	✓	✓	✓	✓	✓
PSM palaikymas	✓	✓	✓	✓	✓
Ar gali generuoti skirtingus PSM	✓	✓	✓	✓	×
Modelių integracija	×	✓	✓	×	✓
Programinės įrangos vystymas	×	✓	✓	×	✓
Bendravimas su kitais įrankiais	✓	✓	✓	✓	✓
Galimybė kurti savo transformacijas	✓	✓	✓	×	✓
Korektiškumas	✓	✓	✓	✓	✓
Išraiškingumas	✓	✓	✓	✓	✓
Šablonai ir apibendrinimai	✓	×	✓	✓	✓
Keitimo palaikymas	×	✓	✓	×	✓
Ryšiai tarp modelių	×	×	✓	×	✓
Programinės įrangos gyvavimo ciklo palaikymas	✓	✓	✓	✓	✓
Standartizacija	✓	✓	✓	✓	✓
Transformacijų kryptis iš PIM į PSM	✓	✓	✓	✓	✓
Transformacijų kryptis iš PSM į kodą	✓	✓	✓	✓	✓
UML įrankis naudojamas vidinis	×	✓	✓	×	✓
UML įrankis naudojamas „MagicDraw“	✓	✓	×	✓	×
UML įrankis naudojamas „Rational Rose“	✓	×	×	✓	×
UML įrankis naudojamas „Poseidon UML“	✓	×	×	✓	×

Išanalizavus populiariausius ir dažniausiai naudojamus MDA įrankius pastebėta idealaus įrankio, kuris tiktų visiems gyvenimo atvejams ir būtų patogus naudoti, nėra. Tačiau galima teigti kad šiuo metu iš pasaulyje esamų geriausias yra „ArcStyler“.

3.6. Veiklos sfera

Sistemos veikimui pakanka, kad architektas apibrėžtų taikymo srities funkcionalumą UML diagramomis. Tai atlikus, sistemos variklis automatiškai sugeneruoja duomenų struktūras ir jų saugojimo infrastruktūrą, o modeliuotojas gali sudaryti scenarijus grafiniu redaktoriumi. Nubraižytas scenarijus siunčiamas kodo generavimo posistemei, kur skriptas transformuojamas į kodą, kurį vartotojui užklausus įvykdo pagrindinis sistemos variklis. Veiklos sferos diagrama pateikta paveiksle (Pav. 4. Bendros sistemos veiklos sfera), o aprašymas lentelėje (Lentelė Nr. 4. Veiklos įvykių sąrašas).

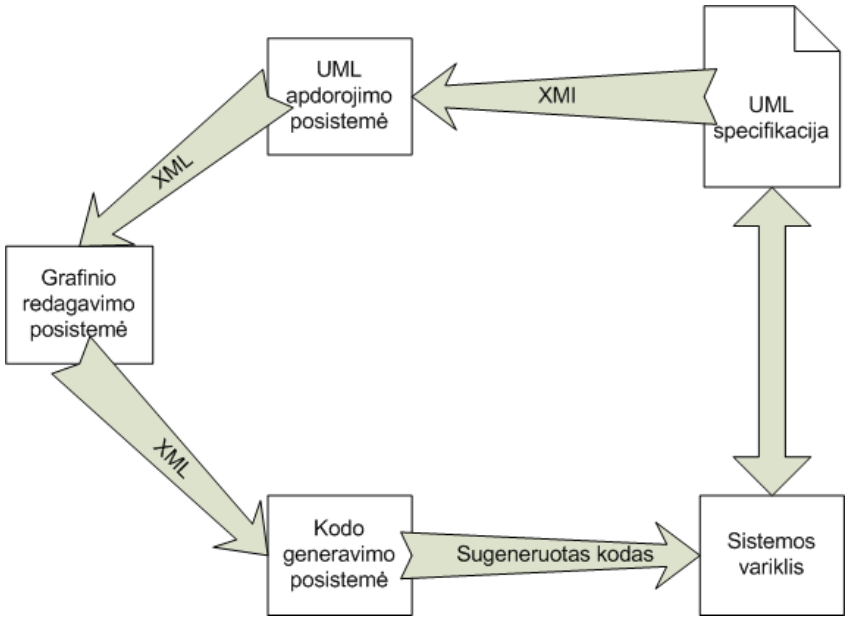


Pav. 4. Bendros sistemos veiklos sfera

Lentelė Nr. 4. Veiklos įvykių sąrašas

Eil. nr.	Etapo aprašymas	Įeinantys / išeinantys informacijos srautai
1.	Modeliuotojas perteikia norimą sistemos modelį, kuris atvaizduojamas kaip scenarijus	Modelis (į) Scenarijus (iš)
2.	Scenarijus užrašomas skriptu ir perduodama į kodo generavimo posistemę	Skriptas (į)
4.	Sistemos architektas apibrėžia sistemos funkcionalumą UML diagramomis	UML diagramos (į)
5.	Sistemos funkcionalumas perduodamas kodo generavimo varikliui	UML diagramos (iš)
6.	Paprastam vartotojui perduodama informacija kurią jis gali keisti (aprašymai ir t.t.)	Informacija (iš) Informacija (į)

3.7. Veiklos pasidalijimas



Pav. 5. Įrankio architektūros diagrama

3.7.1. Grafinis scenarijų kūrimo įrankis

Grafinis scenarijų kūrimo įrankis grafiškai modeliuoja scenarijus. Modelis transformuojamas į tarpinę kalbą.

3.7.2. Kodo generavimo posistemė

Scenarijaus transformavimas į galutinį programinį kodą. Posistemė paremta įskiepiais. Kodas gali būti generuojamas skirtingoms programavimo kalboms ir architektūros realizacijoms.

3.7.3. Duomenų sluoksnio modelio kūrimas ir tyrimas

Esybių transformavimas į konkrečioje aplinkoje veikiančius komponentus. Duomenų sluoksnio modelis greitai sukuria ir pateikia suprojektuotą komponentą kitai posistemei bei užtikrinta, kad komponentas yra korektiškas.

3.7.4. Hipertekstinės grafinės vartotojo sąsajos kūrimas aukšto abstrakcijos lygmens deklaratyvia sintakse

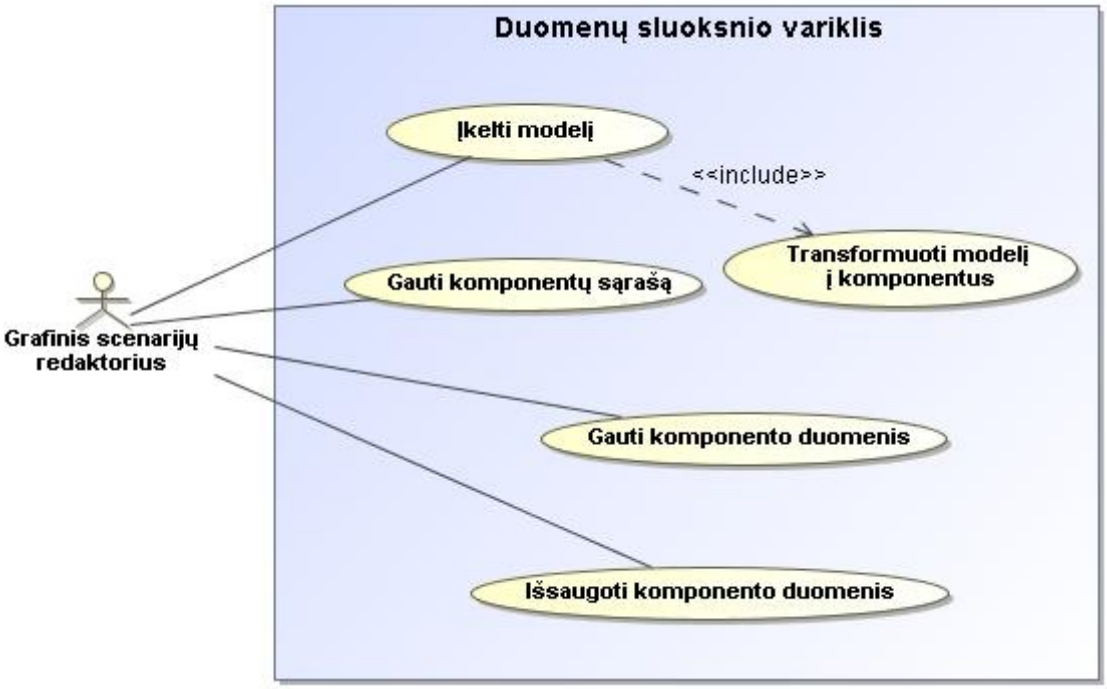
Hipertekstinės grafinės vartotojo sąsajos biblioteka palengvina ir pagreitina vartotojo sąsajos įgyvendinimą bei supaprastina jos plėtojimą. Bibliotekos elementų rinkinį sudaro apie 20 elementų. Architektūra leidžia komponuoti elementus tarpusavyje (agregacija), plėsti elementų funkcionalumą (paveldėjimas) bei kurti elementų hierarchijas. Ši biblioteka yra viena iš grafinio scenarijų kūrimo įrankio taikymo sričių.

4. DUOMENŲ SLUOKSNIO MODELIO PROJEKTAS

Šiame skyriuje bus pateiktas išsamus kuriamos posistemės architektūrinis (duomenų sluoksnio modelio) vaizdas. Posistemės architektūra per UML diagramas: komponentų, paketų ir diegimo diagramas. Taip pat šiame skyriuje bus nustatyti sistemos funkciniai ir nefunkciniai reikalavimai.

4.1. Funkciniai reikalavimai

4.1.1. Panaudos atvejai



Pav. 6.Panaudos atvejų diagrama

Lentelė Nr. 5 Panaudos atvejis „Įkelti modelį“

Panaudos atvejis	**Įkelti modelį**
Tikslas:	Įkelti modelio rinkmeną į duomenų sluoksnio modelio sistemą.
Aktoriai:	Grafinis scenarijų redaktorius
Ryšiai su kitais PA:	Naudoja „*Transformuoti modelį į komponentus*“
Nefunkciniai reikalavimai:	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo.
Prieš-sąlygos:	-
Sužadinimo sąlyga:	Grafinio scenarijaus redaktoriuje pasirenkama modelio rinkmena, spaudžiama įkelti.
Po sąlygos:	Rinkmena įkeliama į tam tikrą aplanką.
Pagrindinis scenarijus:	Grafinio scenarijaus redaktoriuje atidaromas rinkmenos įkėlimo dialogas, kuriame pasirenkama modelio rinkmena spaudžiamas mygtukas „Įkelti“.
Alternatyvūs scenarijai:	Atšaukiamas rinkmenos įkėlimas.

Lentelė Nr. 6. Panaudos atvejis „Transformuoti modelį į komponentus“

Panaudos atvejis	**Transformuoti modelį į komponentus**
Tikslas:	Modelyje esančias esybes transformuoti į kodą, kuris sukompiliuojamas į komponentus ir įdiegiamas.
Aktoriai:	Grafinis scenarijų redaktorius
Ryšiai su kitais PA:	-
Nefunkciniai reikalavimai:	-
Prieš-sąlygos:	Modelio rinkmena turi egzistuoti tam tikrame aplanke.
Sužadinimo sąlyga:	Nuolat tikrinama ar nėra naujos modelio versijos, jei yra inicijuojamas panaudos atvejis.
Po sąlygos:	Pateikiami naudojami komponentai
Pagrindinis scenarijus:	Modelio rinkmena apdorojama, išskiriamos esybės, kurios transformuojamos į kodą, sukompiliuojamos ir įdiegiamos.
Alternatyvūs scenarijai:	-

Lentelė Nr. 7. Panaudos atvejis „Gauti komponentų sąrašą“

Panaudos atvejis	**Gauti komponentų sąrašą**
Tikslas:	Grąžinti sistemoje įdiegtų ir viešai pasiekiamų komponentų adresų sąrašą.
Aktoriai:	Grafinis scenarijų redaktorius
Ryšiai su kitais PA:	-
Nefunkciniai reikalavimai:	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo.
Prieš-sąlygos:	Turi egzistuoti komponentai.
Sužadinimo sąlyga:	Posistemė inicijuoja komponentų sąrašo gavimą.
Po sąlygos:	Grąžinamas komponentų adresų sąrašas.
Pagrindinis scenarijus:	Posistemė inicijuoja komponentų gavimą. Rezultate gražinami adresai, kuriais galima pasiekti
Alternatyvūs scenarijai:	-

Lentelė Nr. 8. Panaudos atvejis „Gauti komponento duomenis“

Panaudos atvejis	**Gauti komponento duomenis**
Tikslas:	Gauti konkretaus komponento duomenis.
Aktoriai:	Grafinis scenarijų redaktorius
Ryšiai su kitais PA:	-
Nefunkciniai reikalavimai:	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo.
Prieš-sąlygos:	Komponentas turi egzistuoti sistemoje.
Sužadinimo sąlyga:	Grafinio scenarijaus redaktoriaus inicijavimas komponento gavimui.
Po sąlygos:	Grąžinama komponento būsenos duomenys
Pagrindinis scenarijus:	Grafinio scenarijaus redaktoriaus inicijuojamas konkretaus komponento duomenų gavimas.
Alternatyvūs scenarijai:	Atšaukiamas komponento gavimas.

Lentelė Nr. 9. Panaudos atvejis „Išsaugoti komponento duomenis“

Panaudos atvejis	**Išsaugoti komponento duomenis**
Tikslas:	Išsaugoti konkretaus komponento duomenis.
Aktoriai:	Grafinis scenarijų redaktorius
Ryšiai su kitais PA:	-
Nefunkciniai reikalavimai:	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo.
Prieš-sąlygos:	-
Sužadinimo sąlyga:	Grafinio scenarijaus redaktoriaus inicijavimas komponento išsaugojimui.
Po sąlygos:	-
Pagrindinis scenarijus:	Grafinio scenarijaus redaktoriuje inicijuojamas konkretaus komponento duomenų išsaugojimas.
Alternatyvūs scenarijai:	-

4.2. Nefunkciniai reikalavimai

4.2.1. Reikalavimai posistemės išvaizdai

Kadangi posistemė nėra tiesiogiai prieinama vartotojui, reikalavimų posistemės išvaizdai nėra.

4.2.2. Reikalavimai panaudojamumui

Kadangi posistemė nėra tiesiogiai prieinama vartotojui, todėl reikalavimų posistemės panaudojamumui nėra.

4.2.3. Reikalavimai vykdymo charakteristikoms

Reikalavimai vykdymo charakteristikoms kiekvienam panaudos atvejui suformuoti žemiau esančia lentele (Lentelė Nr. 10. Panaudos atvejų reikalavimai vykdymo charakteristikoms)

Lentelė Nr. 10. Panaudos atvejų reikalavimai vykdymo charakteristikoms

Panaudos atvejis	**Reikalavimas**
Įkelti modelį	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo
Transformuoti modelį į komponentus	Nėra
Gauti komponentų sąrašą	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo
Gauti komponento duomenis	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo
Išsaugoti komponento duomenis	Turi būti įvykdomas vartotojui nepastebėjus jokio užlaikymo

4.2.4. Reikalavimai veikimo sąlygoms

Posistemė turi būti pasiekiama 99,9% laiko.

4.2.5. Reikalavimai saugumui

Kadangi sistema kuriama tyrimui, saugumo reikalavimai nesuformuoti.

4.2.6. Kultūriniai-politiniai reikalavimai

Kultūriniai ir politiniai reikalavimai nesuformuoti.

4.2.7. Reikalavimai posistemės priežiūrai

Kadangi posistemės veikimas automatinis, specialių reikalavimų posistemės priežiūrai nėra, išskyrus tuos, kurie nurodyti 4.4 skyriuje.

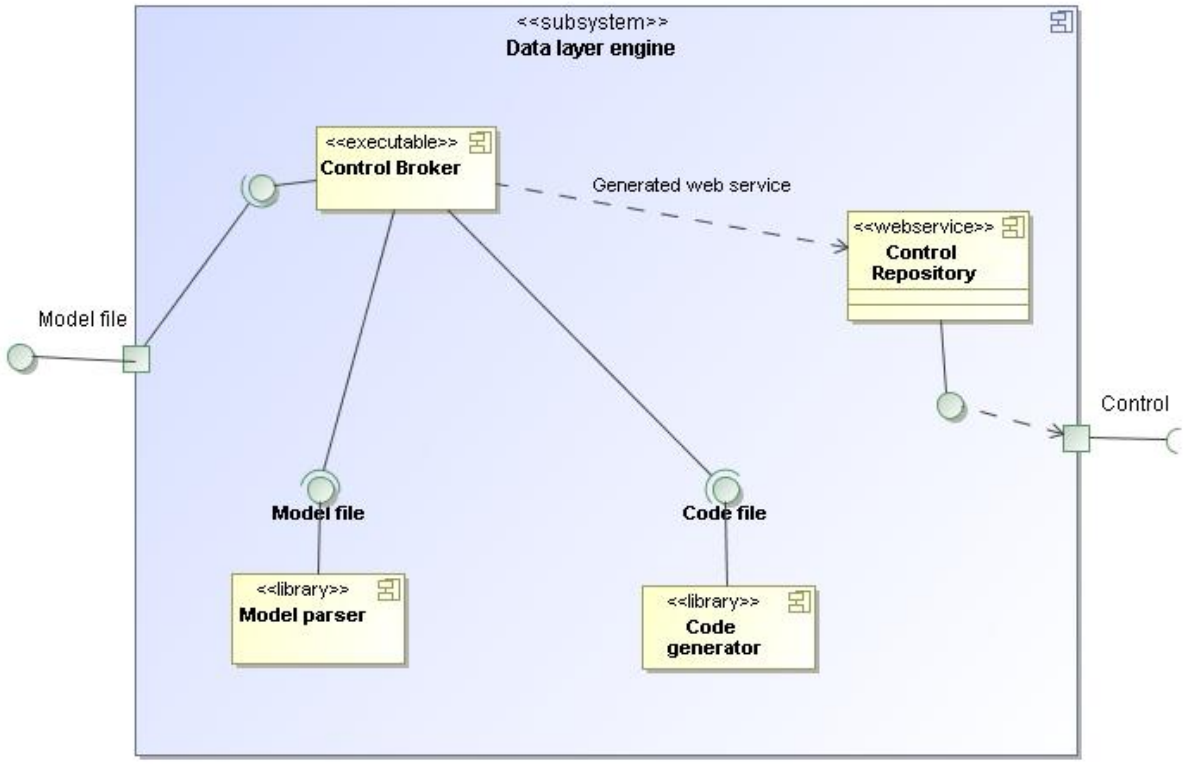
4.2.8. Teisiniai reikalavimai

Visos posistemės dalių veikla turi neprieštarauti Lietuvos Respublikos įstatymams.

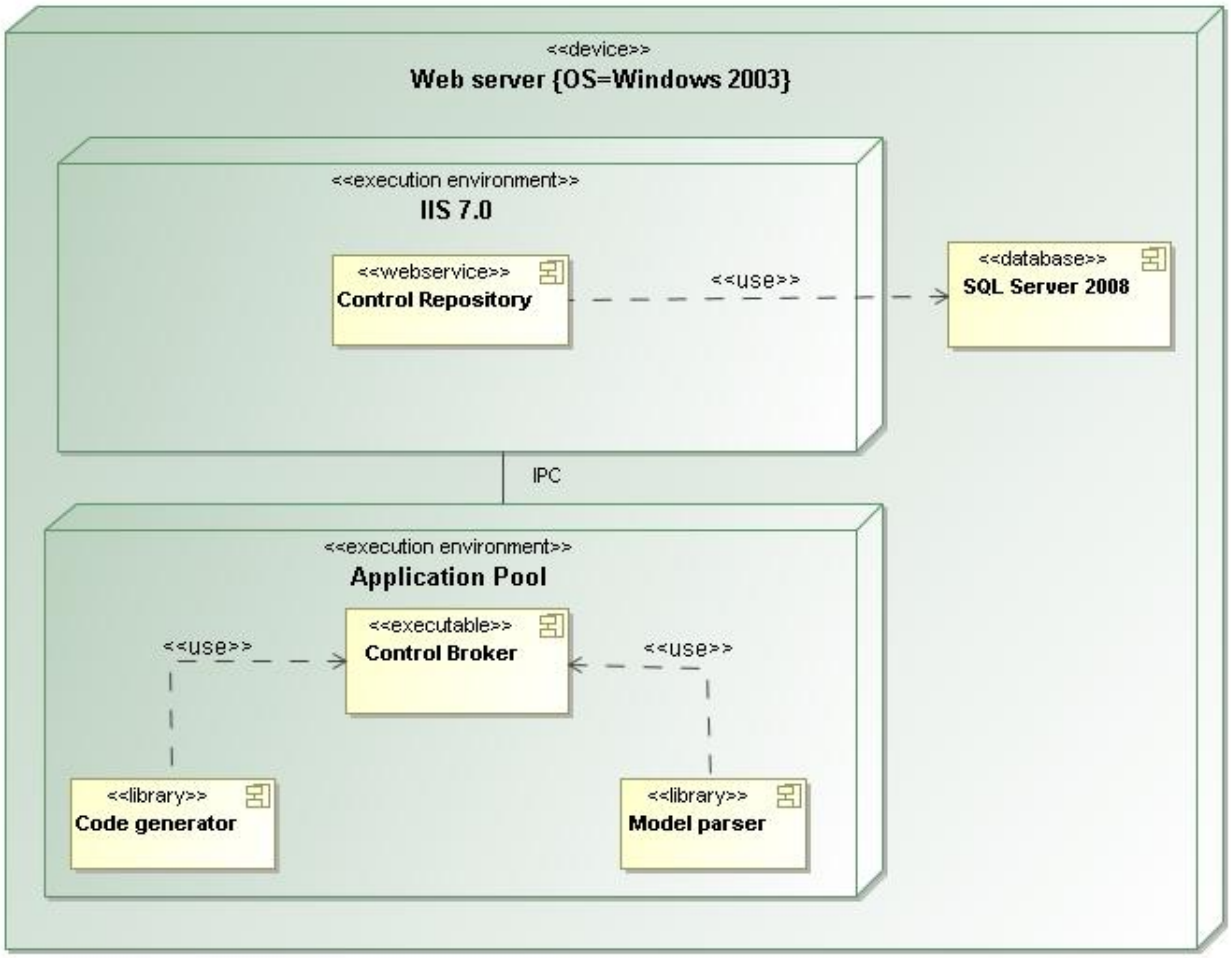
4.3. Architektūros specifikacija

Šiame skyriuje bus pateikta architektūros specifikacija, kurią sudarys trys UML diagramos: komponentų diagrama, išdėstymo diagrama, paketų diagrama.

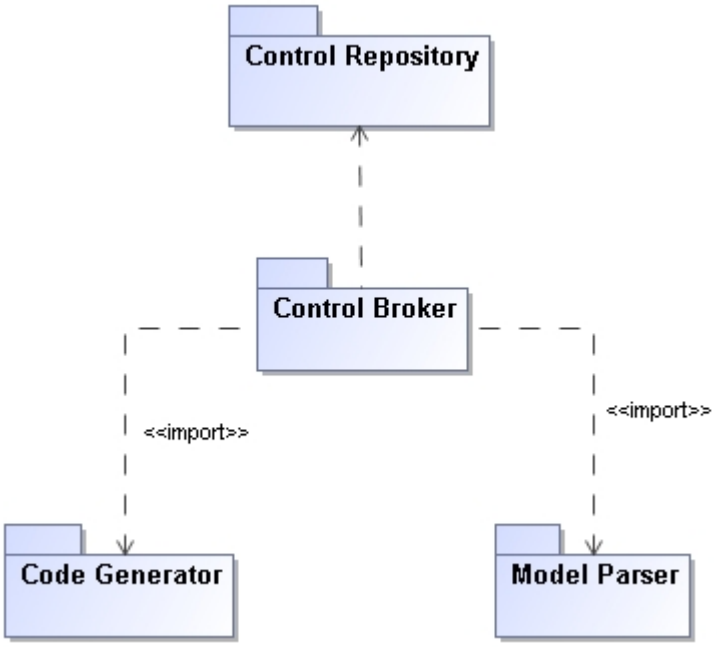
4.3.1. Posistemės statinis vaizdas



Pav. 7. Komponentų diagrama



Pav. 8. Išdėstymo diagrama



Pav. 9. Paketų diagrama

4.4. Reikalavimai programinei įrangai

Lentelė Nr. 11. Reikalavimai programinei įrangai

Pavadinimas	Kiekis
Microsoft Windows Server 2003 (ir aukštesnės versijos)	1
Micorosft Internet Information Service 7.0 (ir aukštesnės versijos)	1
Micorosft .NET Framework 3.5 SP1 (ir aukštesnės versijos)	1
NoMagic Inc. MagicDraw 16.0 (ir aukštesnės versijos)	1

5. PROJEKTAVIMO IR DIEGIMO ĮRANKIO PROJEKTAS

5.1. Architektūros tikslai ir apribojimai

Kuriamos sistemos architektūros tikslai:

- Sistema bus galima naudotis visais kompiuteriais turinčiais interneto naršyklę ir prieigą prie interneto.

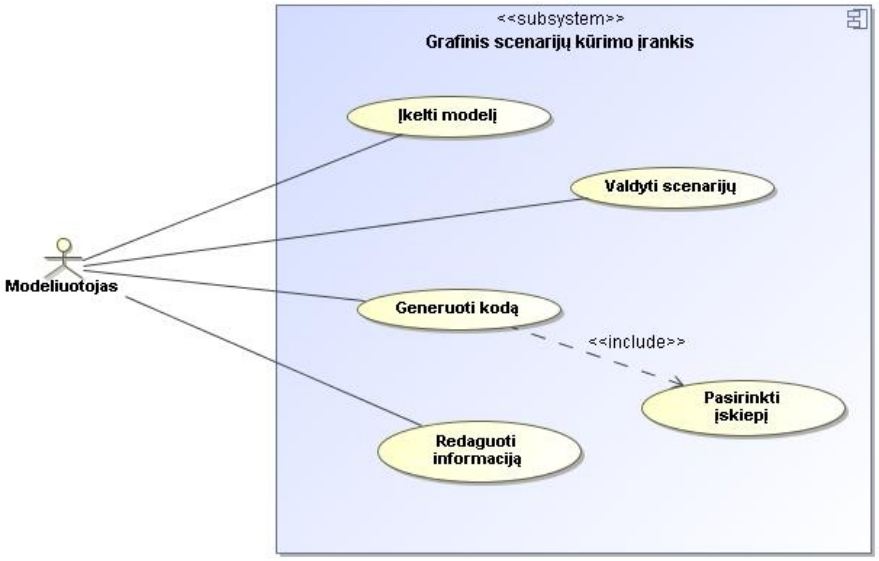
- Duomenimis paremta architektūra. Paduodame UML diagramas ir pagal tai modeliuojame scenarijus grafiniame redaktoriuje.
- Nepriklausoma taikymo sritis. Kiekviena taikymo sritis aprašoma kaip įskiepis kodo generavimo posistemėje.

Kuriamos sistemos apribojimai:

- Įrankis – internetinis. Nėra galimybės dirbti atsijungus nuo interneto.

5.2. Panaudos atvejai

Projektavimo įrankio pagrindiniai panaudos atvejai pateikti paveiksle (Pav. 10), o jų aprašymai lentelėse (Lentelė Nr. 12 – Lentelė Nr. 16).



Pav. 10. Panaudos atvejų vaizdas

Lentelė Nr. 12. Panaudos atvejis „Įkelti modelį“

Įkelti modelį	
Aprašas:	Tai visos sistemos UML specifikacijos įkėlimas.
Vartotojas/Aktorius:	Modeliuotojas.
Prieš-sąlyga:	Sistemos specifikacija yra parengta.
Sužadinimo sąlyga:	Įrankio diegimas į sistemą.
Po-sąlyga:	Įrankiu galima pradėti naudotis.

Lentelė Nr. 13. Panaudos atvejis " Valdyti scenarijų "

Valdyti scenarijų	
Aprašas:	Scenarijaus modeliavimas naudojant grafinį scenarijų kūrimo įrankį.
Vartotojas/Aktorius:	Modeliuotojas.
Prieš-sąlyga:	Nėra.
Sužadinimo sąlyga:	Kilo poreikis sistemoje įgyvendinti naują funkcionalumą arba projektuoti naują sistemą.
Po-sąlyga:	Sistema gali naudoti scenarijuje sumodeliuotą veiklą.

Lentelė Nr. 14. Panaudos atvejis „Generuoti kodą"

Generuoti kodą	
Aprašas:	Generuoja skriptą, pagal sumodeliuotą scenarijų.
Vartotojas/Aktorius:	Modeliuotojas.
Prieš-sąlyga:	Scenarijus yra sukurtas.
Sužadinimo sąlyga:	Baigtas modeliuoti scenarijus.
Po-sąlyga:	Sugeneruotas kodas atitinka scenarijaus veiksmus.

Lentelė Nr. 15. Panaudos atvejis „Pasirinkti įskiepį"

Pasirinkti įskiepį	
Aprašas:	Parenka įskiepį atitinkantį taikymo srities architektūrą.
Vartotojas/Aktorius:	Modeliuotojas.
Ryšys su kitais PA:	Generuoti kodą.
Prieš-sąlyga:	Scenarijus yra sukurtas.
Sužadinimo sąlyga:	Baigtas modeliuoti scenarijus.
Po-sąlyga:	Sugeneruotas kodas atitinka taikymo srities architektūrą.

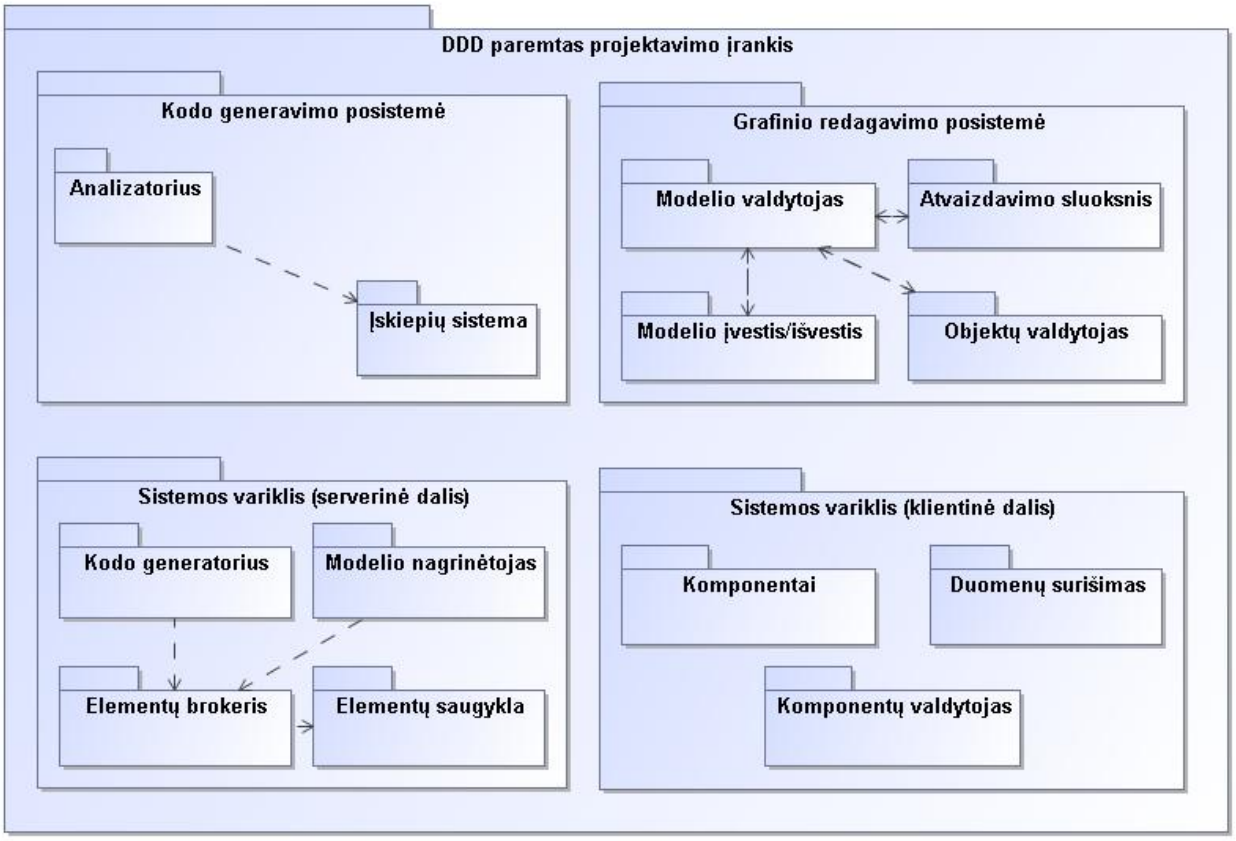
Lentelė Nr. 16. Panaudos atvejis „Redaguoti informaciją”

Redaguoti informaciją	
Aprašas:	Galimybė pakeisti sistemos scenarijaus modelį, neleidžiant pakeisti sistemos loginės veiklos.
Vartotojas/Aktorius:	Modeliuotojas.
Prieš-sąlyga:	Yra bent vienas scenarijus. Sistemos veikla išlieka ta pati, bet pasikeitė aprašymas.
Sužadinimo sąlyga:	Pasikeitė veiklos komponentu aprašas (tekstinė informacija).
Po-sąlyga:	Panaudos atvejo funkcionalumas yra pakankamas atlikti numatytus pakeitimus.

5.3. Sistemos statinis vaizdas

Sistema suskirstyta į šiuos paketus (Pav. 11):

- Grafinio redagavimo posistemė.
- Kodo generavimo posistemė.
- Sistemos variklio serverinė dalis.
- Sistemos variklio klientinė dalis.



Pav. 11. Projektavimo įrankio paketų diagrama

5.3.1. Grafinio redagavimo posistemė

„Model manager“ paketas atsakingas už modelio sąveiką su kitomis posistemės dalimis. Tai yra pagrindinis Grafinės redagavimo posistemės paketas.

„Model IO“ - atsakingas už pradinės sistemos specifikacijos konvertavimą į posistemės objektus. Taip pat šis paketas atsakingas už sukurto scenarijaus saugojimą duomenų bazėje.

„Presentation layer“ paketas atsakingas už grafinėje sąsajoje atvaizduojamus objektus ir ryšius. Taip pat šis paketas inicijuoja paketų veiklą.

„Object manager“ atsakingas už objektų kūrimą modelyje. Šis paketas paremtas „Factory method“ šablonu kuris leidžia praplėsti objekto tipus naudojamus įrankyje.

5.3.2. Kodo generavimo posistemė

Analizatoriaus paketas atsakingas už scenarijaus perskaitymą ir parametrų surinkimą.

Įskiepių sistemos paketas atsakingas už įskiepių parinkimą. Kiekvienas įskiepis atsakingas už programinio kodo generavimą, optimizavimą bei perdavimą taikymo srities klientinei daliai.

5.3.3. Sistemos variklio serverinė dalis

Kodo generatoriaus paketas atsakingas už išeities teksto failų generavimą.

Modelio nagrinėtojo paketas atsakingas už modelio teisingą nuskaitymą.

Elementų brokerio paketas atsakingas už teisingą modelio transformaciją į veikiančius komponentus.

Elementų saugyklos paketas atsakingas už veikiančių komponentų saugojimą ir pateikimą galutiniam vartotojui.

5.3.4. Sistemos variklio klientinė dalis

„Komponentai“ paketas atsakingas už komponentų rinkinį it funkcionalumą.

„Duomenų surišimas“ paketas atsakingas už duomenų tiekimą komponentams.

„Komponentų valdymas“ paketas atsakingas už komponentų atpažinimą ir sukūrimą.

5.4. Diegimo aplinka

Sistemą turi sudaryti dvi atskiros dalys bendraujančios sąsajomis – klientinė bei serverinė. Šios dalys veikia skirtingose aplinkose. Klientinė dalis veikia vartotojo kompiuteryje interneto naršyklėje, čia vyksta informacijos atvaizdavimas bei įvedimas. Serverinė dalis yra nutolusiame kompiuteryje (serveryje), čia saugomi sistemos duomenys, atliekami kodo generavimo veiksmai.

Reikalavimai vartotojo programinei įrangai pateikiami lentelėje „Lentelė Nr. 17“, minimalūs reikalavimai vartotojo techninei įrangai pateikiami lentelėje „Lentelė Nr. 18“. Reikalavimai serverio programinei įrangai pateikiami lentelėje „Lentelė Nr. 19“, minimalūs reikalavimai serverio techninei įrangai pateikiami lentelėje „Lentelė Nr. 20“.

Lentelė Nr. 17. Reikalavimai vartotojo programinei įrangai

Reikalavimai	Parametrai
Operacinė sistema	Nesvarbu kokia
Interneto naršyklė	Turi būti įjungtas „JavaScript“ rėžimas
Papildoma programinė įranga	Įdiegtas „Microsoft Silverlight“ karkasas

Lentelė Nr. 18. Minimalūs reikalavimai vartotojo techninei įrangai

Reikalavimai	Parametrai
Procesorius	Daugiau kaip 1 Ghz
Interneto prieiga	Būtina.Greičio apribojimų nėra.

Lentelė Nr. 19. Reikalavimai serverio programinei įrangai

Reikalavimai	Parametrai
Programinės įrangos paketai	Apache 1.3 ir naujesnis.
	PHP 5.0 ir naujesnis.
	Micorosft Internet Information Service 7.0 ir naujesnis
	Microsoft Windows Server 2003 (ir aukštesnės versijos)

Lentelė Nr. 20. Minimalūs reikalavimai serverio techninei įrangai

Reikalavimai	Parametrai
Procesorius	Daugiau kaip 1 Ghz.
Operatyvioji atmintis	128 MB ir daugiau.
Laisva disko vieta	100 MB ir daugiau.

6. DUOMENŲ SLUOKSNIO MODELIO TYRIMAS

Šioje dalyje bus aprašomas suprojektuoto ir realizuoto duomenų sluoksnio variklio tyrimas: atliekama kokybės analizė ir aprašomos iškilusios problemos, pateikiami siūlymai tobulinti sistemą.

6.1. Kokybės analizė

6.1.1. Atitikimas specifikacijai

Duomenų sluoksnio variklis atitinka specifikacijoje numatytus funkcinius reikalavimus. Tiesa, dėl iškilusių kai kurių modelio formatų ypatumų, iš anksto buvo atsisakyta padengti visus galimus modelio nagrinėjimo atvejus. Ši atvejį plačiau aptarsime kitame skyriuje.

6.1.2. Modelio saugojimo XMI formatu identiškumo problema

Kaip jau buvo minėta anksčiau (2.6 skyriuje), išanalizavus du rinkoje dominuojančius įrankius (IBM Rational Rose ir NoMagic Inc. MagicDraw) pastebėta, kad nors ir naudojamas universalus XMI formatas modeliui aprašyti, tačiau šios dvi kompanijos skirtingai interpretuoja šio formato teikiamas galimybes. Todėl modelio failas, aprašytas skirtingais įrankiais, nebus identiškas. Dėl šios priežasties negalima bet kokiu įrankiu suprojektuoto modeliu pateikti duomenų sluoksnio varikliui.

6.1.3. Modelio esybių versijavimo nebuvimo problema

Dabar dirbant keliems modeliuotojams su grafiniu scenarijų redaktoriumi iškyla problema, kad vienam iš jų pridėjus/pašalinus klasės atributą ir jį išsaugojus elementų saugykloje, kitas modeliuotojas turėtų nepajusti šio pasikeitimo pasekmių arba būti informuotas apie saugykloje egzistuojančią naują versiją.

6.2. Tobulinimo galimybės

6.2.1. Pritaikymas visų rinkoje egzistuojančių UML įrankių modeliams nagrinėti

Kaip jau buvo minėta 6.1.2 skyriuje, egzistuoja modelio saugojimo formatų identiškumo problema skirtinguose įrankiuose. Kol nėra vieningo susitarimo kaip naudoti XMI formatą,

vienintelė išeitis modelio nagrinėjimo komponentą pritaikyti kiekvienam įrankiui atskirai. Tokiu būdu duomenų sluoksnio variklis taptų universalesnis.

6.2.2. Elementų saugyklos funkcionalumo išplėtimas sesijomis

Projektuojančiam modeliuotojui svarbu, kad atsisėdus į darbo vietą iškart būtų galima vykdyti projektavimo darbus, arba nutrūkus tinklo ryšiui būtų galima atkurti paskutinę išsaugotą projektavimo sesiją. Todėl duomenų sluoksnio variklį siūlyčiau praplėsti sesijų valdymo funkcionalumu. Šio funkcionalumo įdiegimas leistų projektuotojui išsaugoti esamą projektuojamo modelio būseną, o vėliau ją atkurti.

6.2.3. Elementų saugyklos praplėtimas versijų valdymo galimybe

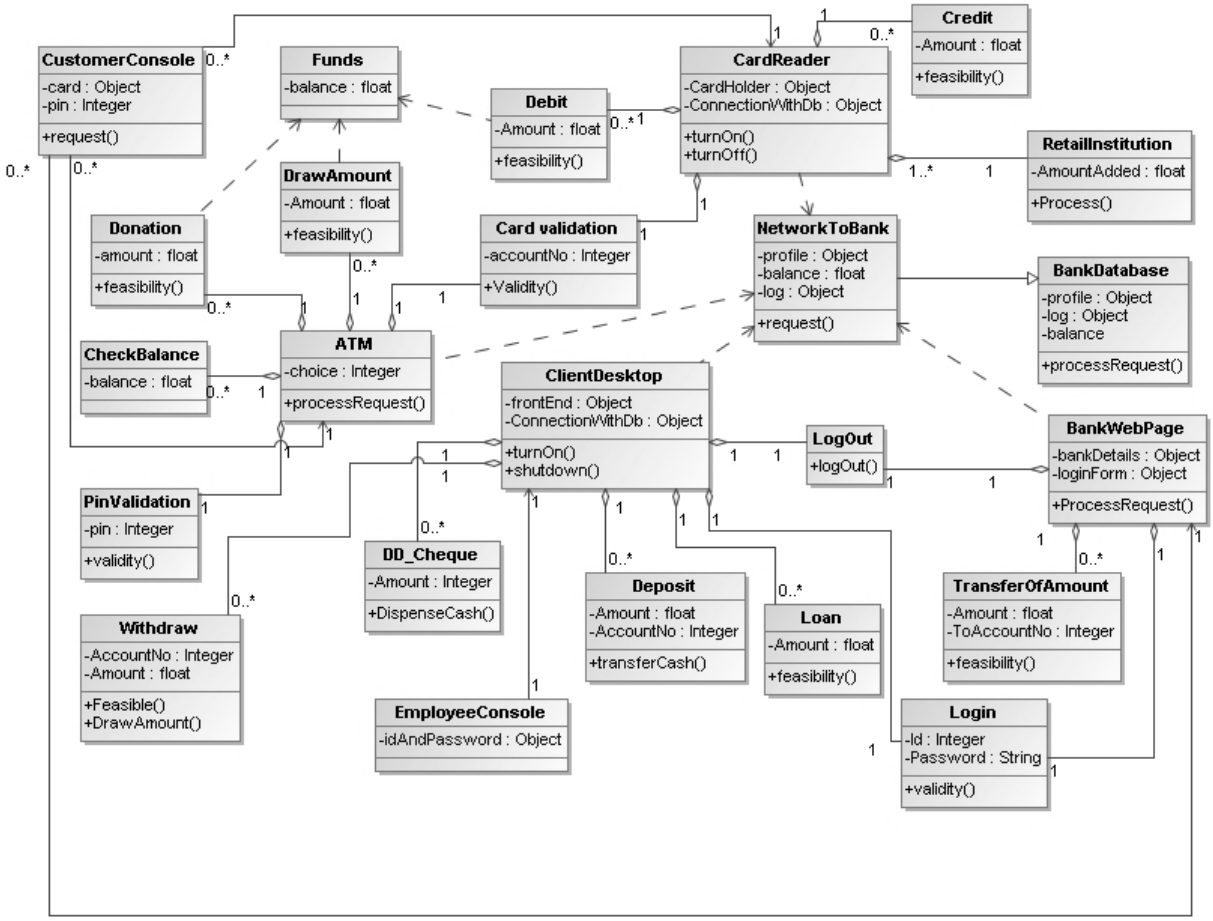
Su tuo pačiu modeliu dirbantiems keliems projektuotojams svarbu, kad išsaugojus modelio pakeitimus į saugyklą, nebūtų prarastas esamas modelio funkcionalumo bei išvengta galimų sutrikimų dėl pasikeitusio modelio. Dėl šios priežasties siūloma duomenų sluoksnio variklį išplėsti modelio versijavimo funkcija.

6.3. Išvados

Realizuoto duomenų sluoksnio variklio tyrimo metu buvo aptikta keletas problemų. Šios problemos buvo išdėstytos bei pasiūlyti sprendimai joms pašalinti.

7. DUOMENŲ SLUOKSNIO MODELIO EKSPERIMENTINĖ DALIS

Šioje dalyje bus aprašytas eksperimentas su realizuotu duomenų sluoksnio varikliu. Rezultatai bus pateikti lentelių bei grafikų pavidalu. Eksperimentas bus atliktas su gerai žinomu ATM modeliu (žiūrėti žemiau Pav. 12. ATM modelis).



Pav. 12. ATM modelis

7.1. Eksperimento vykdymo metodika

Abiejuose eksperimentuose bus įgyvendintas 7 skyriuje minėtas ATM modelis.

7.1.1. Automatinės ir rankinės sistemos realizacijos palyginimo metodika

Šio eksperimento tikslas patikrinti ar automatinis modelio transformavimas į komponentus greitesnis už žmogaus rankinį darbą. Eksperimentas susideda iš dviejų dalių: ATM

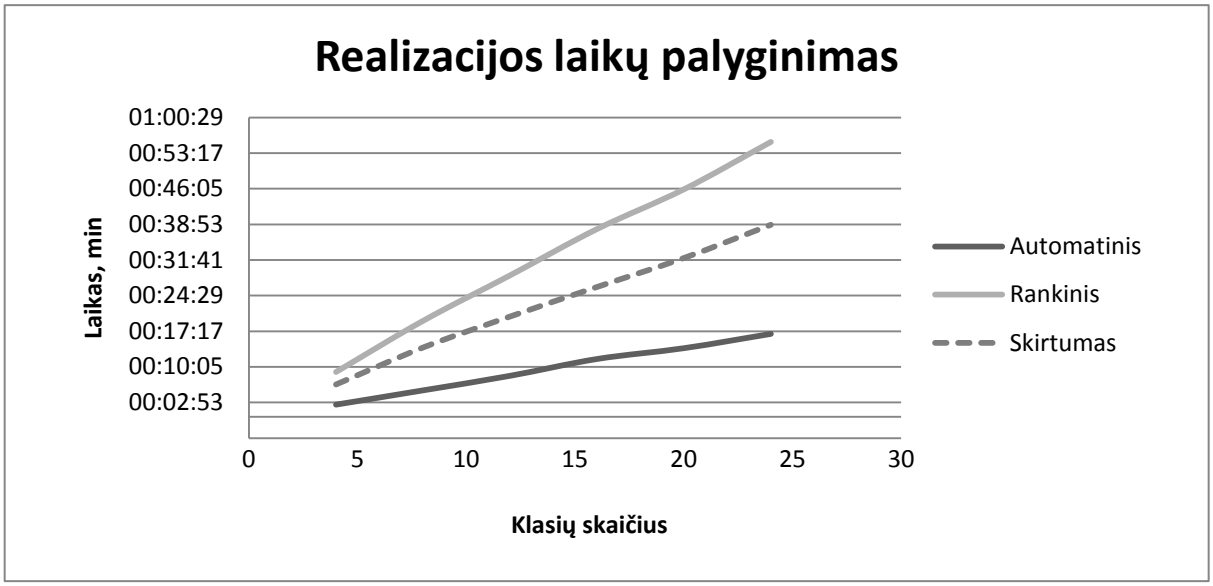
modelio projektavimo ir įkėlimo į duomenų sluoksnio variklį ir ATM modelio programavimo bei diegimo rankiniu būdu. Kiekvienoje dalyje eksperimentas vykdomas su skirtingu klasių skaičiumi (4, 8, 12, 16, 20, 24). Automatinio įgyvendinimo metu laikas stabdomas, kai komponentas atsiranda komponentų saugykloje. Kad būtų užtikrinti korektiški rezultatai, eksperimentą vykdys vienas ir tas pats žmogus (t.y. ir klases programuos ir ATM projektuos vienas asmuo).

7.2. Eksperimento rezultatai

7.2.1. Automatinės ir rankinės sistemos realizacijos palyginimas

Lentelė Nr. 21. Automatinės ir rankinės sistemos realizacijos palyginimas

Klasių skaičius	**4**	**8**	**12**	**16**	**20**	**24**
Automatinis įgyvendinimas, min.	02:28	05:21	08:17	11:41	13:52	16:45
Rankinis įgyvendinimas, min.	09:01	19:21	28:32	37:54	45:56	55:31



Pav. 13. Realizacijos laikų palyginimas

7.3. Išvados

Atlikus automatinės ir rankinės realizacijų palyginimus, gauti duomenys (žiūrėti Lentelė Nr. 21) patvirtino, kad panaudojus įgyvendintą duomenų sluoksnio modelį programavimo darbai užtrunka trumpiau. Ši tendencija dar labiau išryškėja didinant realizuojamų komponentų kiekį

(žiūrėti Pav. 13 liniją „Skirtumas“). Šį efektą galima paaiškinti tuo, kad didėjant komponentų skaičiui, rankinis žiniatinklio paslaugos programavimas bei diegimas užtrunka ypač ilgai.

8. PROJEKTAVIMO IR DIEGIMO ĮRANKIO EKSPERIMENTINĖ DALIS

8.1. Projektavimo ir diegimo įrankio greičio tyrimas

Norint atlikti pilną projektavimo ir diegimo įrankio eksperimentą buvo sumąstytos skirtingo sudėtingumo užduotys (užduotys pateikiamos didėjant sudėtingumui):

- Sumodeliuokite paieškos komponentą, kurį sudaro teksto įvedimo laukas ir paieškos vykdymo mygtukas. Nuspaudus jį, įvestas tekstas turi būti perduotas serveriui ir pereinama į „Rezultatai“ langą.
- Sumodeliuokite meniu komponentą, kuriame rodomi trys mygtukai. Nuspaudus pirmąjį mygtuką pereinama į „Home“ langą, nuspaudus antrąjį pereinama į „Search“ langą. Trečias mygtukas veikia priklausomai nuo prisijungimo būsenos. Jei vartotojas prisijungęs, ant jo rodomas tekstas „Logout“, nuspaudus jį vartotojas atsijungia ir įjungiamas „Home“ langas. Kitu atvejų rodomas tekstas “Login“, o jį paspaudus atsidaro prisijungimo langas „Login“.

Pav. 14. Trečios užduoties pavyzdys

- Sumodeliuokite prisijungimo langą, kuris atrodytų panašiai į Pav. 14. „Dialog“ komponento viduje yra logotipas, klaidos pranešimo laukas, vartotojo vardo žymė, vartotojo vardo įvedimo laukas, slaptažodžio žymė, slaptažodžio įvedimo laukas ir pateikimo mygtukas.
    - Logotipo paveikslui nurodyti „logo.jpg“ failą.
    - Vartotojo vardo žymės tekstą nurodyti „Username“.
    - Slaptažodžio žymės tekstą nurodyti „Password“

- Pateikimo mygtuko tekstą nurodyti „LogIn“. Paspaudus jį, jei vartotojo vardas netinkamas, parodomas klaidos pranešimo laukas su užrašu „Tokio vartotojo nėra“. Taip pat išvalomi abu įvedimo laukai. Jei toks vartotojo vardas egzistuoja, bet netinka slaptažodis parodomas klaidos pranešimas su užrašu „Blogas slaptažodis“. Taip pat ištrinamas slaptažodžio įvedimo laukas, o vartotojo vardas įvedimo lauke paliekamas. Įvedus teisingus prisijungimo duomenis įjungiamas langas „Secret“.

8.2. Nepatyrusio programuotojo eksperimentas

Sukurto projektavimo įrankio įvertinimui buvo atliktas praktinis bandymas. Jo metu komponentams buvo sukurti grafiniai modeliai ir rašomas programinis kodas. Šio bandymo metu norėjome įsitikinti, kiek kodo generavimas iš grafinio modelio bus greitesnis už įprastą kodo rašymą programuojant nepatyrusiam programuotojui. Buvo pasirinktos trys skirtingo sunkumo užduotys, kurios leis nustatyti laiko skirtumo priklausomybę, didėjant projektų sudėtingumui.

8.2.1. Nepatyrusio programuotojo eksperimento rezultatai

Lentelė Nr. 22. Nepatyrusio programuotojo rezultatai

Realizavimo tipas	**Sudėtingumas**		
	Žemas	**Vidutinis**	**Aukštas**
Modeliavimo laikas (min.)	4,5	4,1	5,8
Programavimas laikas (min.)	33,8	21	37,4
Pagreitėjimas (%)	751,11	512,20	644,83

8.2.2. Išvados

Atlikus eksperimentą galima pastebėti, kad didžiausias laiko skirtumas modeliuojant ir programuojant matomas vykdant lengviausią užduotį, vėliau skirtumas sumažėjo, bet išaugo vykdant sudėtingą užduotį. Tai parodo, kad nauji programuotojai įdeda daug pastangų, kol įsigilina į karkaso galimybes ir išanalizuoja dokumentaciją. Skirtumo sumažėjimas atliekant vidutinę ir padidėjimas vykdant sunkią užduotį rodo, kad įrankio efektingumas auga didėjant uždavinio sudėtingumui.

8.3. Patyrusio programuotojo eksperimentas

Tos pačios užduotys buvo duotos patyrusiam, apie sistemos variklį išmanančiam, programuotojui. Po šio bandymo rezultatų bus priimta išvada ar sudėtingumui kylant modeliavimo greitis išliks aukštesnis nei programavimo.

8.3.1. Rezultatai

Lentelė Nr. 23. Patyrusio programuotojo rezultatai

Realizavimo tipas	**Sudėtingumas**		
	Žemas	**Vidutinis**	**Aukštas**
Modeliavimo laikas (min.)	2,5	3	4,2
Programavimas laikas (min.)	2,3	4,6	8,1
Pagreitėjimas (%)	92	153,33	192,86

8.3.2. Išvados

Eksperimento rezultatai rodo, kad patyręs programuotojas, realizuodamas nesudėtingą užduotį, ją įgyvendino greičiau nei modeliuojant, tačiau kai užduočių sudėtingumas išaugo modeliavimo pagreitėjimas sparčiai augo. Todėl galime sakyti, kad sudėtingų sistemų modeliavimas yra efektyvesnis už programavimą.

9. IŠVADOS

Šiame darbe buvo nagrinėjama viena iš projektavimo ir diegimo įrankio dalių - duomenų sluoksnio modelis, kuris atsakingas už suprojektuoto modelio transformavimą į veikiančius komponentus. Analizės dalyje buvo nustatytas realizuoti tinkamiausias modeliavimo metodas, kuriuo tapo Vieninga Žymėjimo Kalba (UML) bei apžvelgtos į projektavimo ir diegimo įrankį panašios priemonės.

Atlikus eksperimentinės sistemos automatizuotą bei rankinę realizaciją, gauti eksperimentiniai rezultatai parodė, kad net esant nedideliam klasių skaičiui, dėl duomenimis paremto modelio procesas užtruko daugiau nei trimis kartais trumpiau nei programuotojas. Klasių skaičiui didėjant, šis laiko skirtumas tampa dar didesnis. Eksperimentas pilnai patvirtina analizės dalyje iškeltą hipotezę, kad naudojant projektavimo įrankius ir modelio transformaciją į kodą, pasiekiamas didesnis veiklos efektyvumas.

10. LITERATŪRA

[1]. **Griethuysen, J. van.** *Concepts and Terminology for the Conceptual Schema and the Information Base.* New York : s.n., 1982. ISO/TC97/SC5/WG3-N695.

[2]. *Data modeling in UML and ORM: a comparison.* **Halpin, Terry and Bloesch, Anthony.** 1999, Journal of Database Management, pp. 4-13.

[3]. **Campbell, L., Halpin, T ir Proper, H.** *Conceptual schemas with abstractions: making flat conceptual schemas more comprehensible.* 1996.

[4]. **Bird, Linda, Goodchild, Andrew ir Halpin, Terry.** *Object Role Modelling and XML-Schema.* s.l. : Springer Berlin / Heidelberg, 2000. T. Volume 1920/2000.

[5]. *Introduction To OCL.* **Warmer, J.** s.l. : Technology of Object-Oriented Languages and Systems, 1999. 0-7695-0275-x.

[6]. **Halpin, Terry ir Bloesch, Anthony.** *ConQuer: A conceptual query language.* s.l. : Springer Berlin / Heidelberg, 1996. 978-3-540-61784-6.

[7]. **Kovse, Jernej ir Härder, Theo.** *Object-Oriented Information Systems.* s.l. : Springer Berlin / Heidelberg, 2002. 978-3-540-44087-1.

[8]. **(ECMA), European Computer Manufacturers Association.** *C# Language Specification.* Geneva : European Computer Manufacturers Association (ECMA), 2006.

[9]. **Bohlen, M. and A. Team.** AndroMDA. *AndroMDA.* [Tinkle] http://www.andromda.org/.

[10]. **Packevičius, Šarūnas, Eidukynaitė, Vilma ir Ušaniov, Andrej.** *MDA CASE ĮRANKIŲ ANALIZĖ.* s.l. : Kauno Technologijos Universitetas.

11.TERMINŲ IR SANTRUMPŲ ŽODYNAS

ORM (*Object Role Modelling*) - tai semantinis duomenų modeliavimo būdas, modeliuojamą sritį išreiškiantis per objektus ir jų atliekamus vaidmenis (sąsajas tarp objektų)

OCL (*Object Constraint Language*) - deklaratyvi ribojimų kalba, skirta aprašyti taisykles taikomas UML modeliams.

W3C (*World Wide Web Consortium*) - konsorciumas leidžiantis programinės įrangos standartus žiniatinkliui.

UML (*Unified Modeling Language*) - modeliavimo ir specifikacijų kūrimo kalba, skirta specifikuoti, atvaizduoti ir konstruoti objektiškai orientuotų programų dokumentus.

IBM (*International Business Machines Corporation*) - JAV kompiuterių gamintoja, programinės ir techninės įrangos prekybos bendrovė, teikianti infrastruktūros ir konsultavimo paslaugas.

HTTP (*HyperText Transfer Protocol*) - pagrindinis metodas pasiekti informaciją pasauliniame tinkle.

SOAP (*Simple Object Access Protocol*) – protokolo specifikacija skirta struktūrizuotos duomenų informacijos apsikeitimui per kompiuterinį tinklą.

WCF *(Windows Communication Foundation)* – .NET karkaso programavimo sąsaja skirta kurti į paslaugas orientuotas taikomąsias programas.

SP *(Service Pack)* – programinės įrangos atnaujinimo paketas.

ECMA *(European Computer Manufacturers Association)* – tarptautinė, privati, nesiekianti pelno standartų kūrimo asociacija.

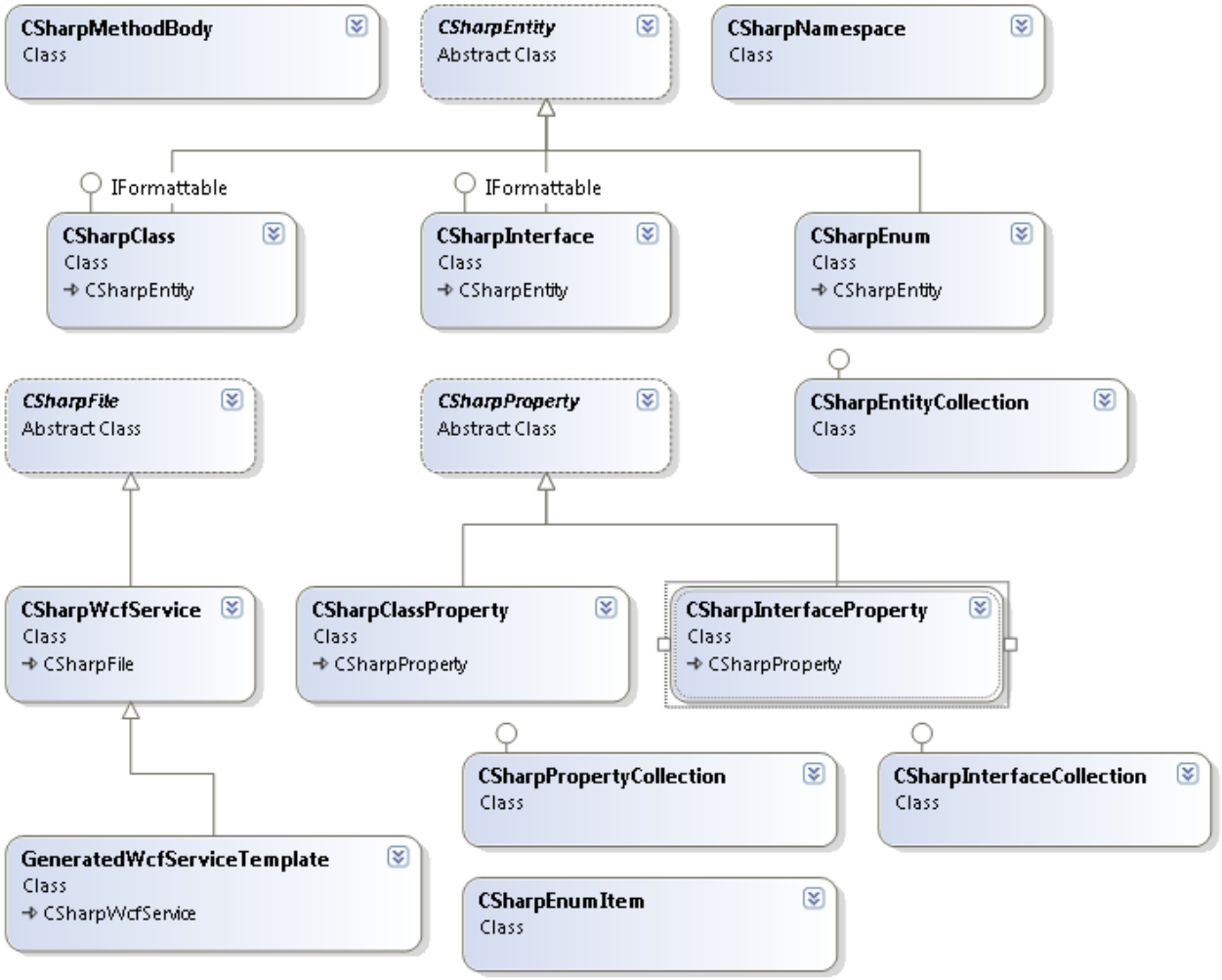
MDA *(Model Driven Engineering)* – modeliu paremta architektūra.

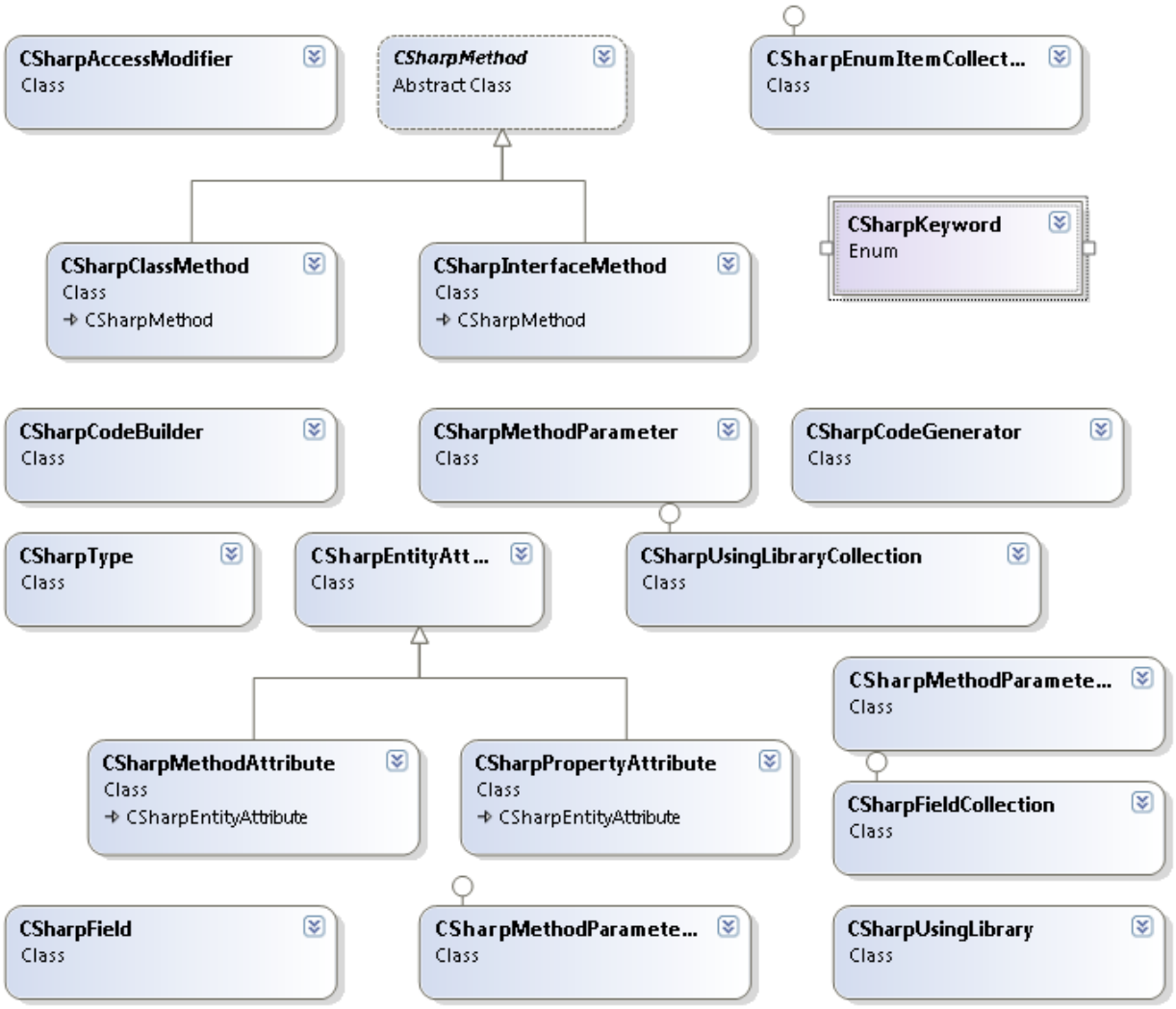
DDD *(Data-driven design)* – duomenimis paremta architektūra.

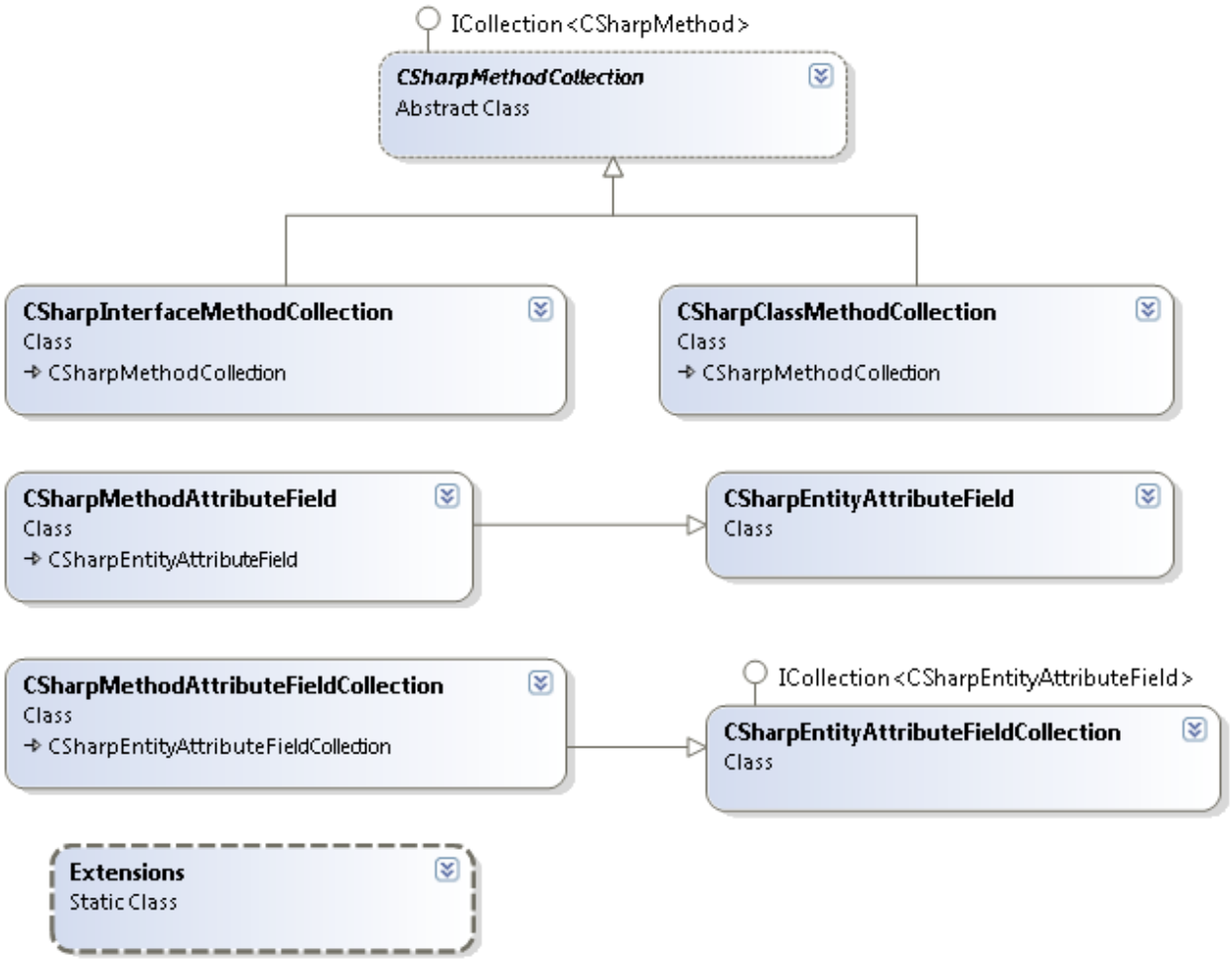
12.PRIEDAI

12.1. Duomenų sluoksnio variklio klasių diagramos

12.1.1. Kodo generatoriaus paketo klasių diagramos



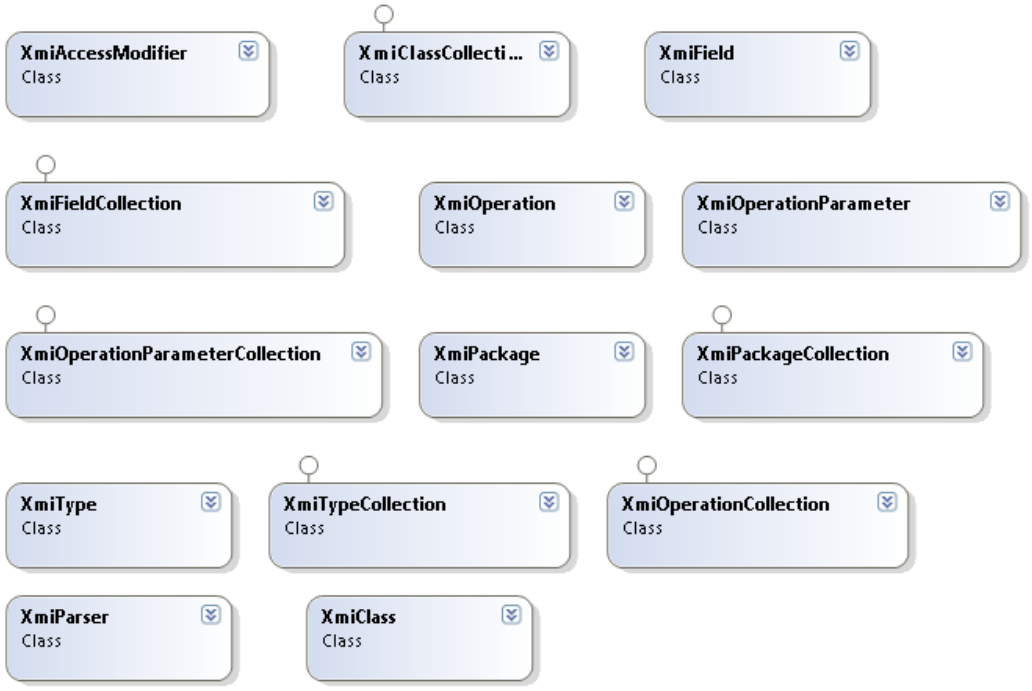

CSharpAccessModifier
Class
CSharpMethod
Abstract Class
CSharpEnumItemCollect...
Class
CSharpClassMethod
Class
CSharpMethod
CSharpInterfaceMethod
Class
CSharpMethod
CSharpKeyword
Enum
CSharpCodeBuilder
Class
CSharpMethodParameter
Class
CSharpCodeGenerator
Class
CSharpType
Class
CSharpEntityAtt...
Class
CSharpUsingLibraryCollection
Class
CSharpMethodParamete...
Class
CSharpMethodAttribute
Class
CSharpEntityAttribute
CSharpPropertyAttribute
Class
CSharpEntityAttribute
CSharpFieldCollection
Class
CSharpField
Class
CSharpMethodParamete...
Class
CSharpUsingLibrary
Class



12.1.2. Komponentų agento paketo klasių diagramos

12.1.3. Komponentų saugyklos paketo klasių diagramos

12.1.4. Modelio nagrinėtojo paketo klasių diagramos



12.2. Straipsnis „Automatinis kodo generavimas naudojant grafinį scenarijų kūrimą remiantis Data driven design šablonu“.

Straipsnis pristatytas 2009 m. gegužės 8 d. 14-oje tarpuniversitetinėje magistrantų ir doktorantų konferencijoje "Informacinės technologijos 2009".

AUTOMATINIS KODO GENERAVIMAS NAUDOJANT GRAFINĮ SCENARIJŲ KŪRIMĄ REMIANTIS DATA DRIVEN DESIGN ŠABLONU

Kęstutis Valinčius, Sigitas Povilaitis, Rytis Ūsalis ir Paulius Paškevičius

Kauno technologijos universitetas, Programų inžinerijos katedra

1. Įvadas

Data Driven Design metodologija plačiai naudojama įvairiose programinėse sistemose. Šios metodologijos tikslas - atskirti bei lygiagretinti programuotojų ir dizainerių veiklą. Sistemos branduolio funkcionalumas yra įgyvendinamas sąsajomis, o dinamika - scenarijų pagalba. Taip įvedamas abstrakcijos lygmuo, kurio dėka programinis produktas tampa lankstesnis, paprasčiau palaikomas ir tobulinamas, be to šiuos veiksmus galima atlikti lygiagrečiai. Kuriant scenarijus grafiškai mažėja klaidų tikimybė, spartėja darbo našumas ir užtenka minimalių programavimo žinių. Tai leidžia darbuotojams dirbti darbą, kurį jis moka geriausiai[1]. Nors duomenimis paremto šablono naudojimas ir yra imlus laikui procesas, bet supaprastinus sudėtingus ar problematiškus etapus sumažinsime riziką[2].

UML (Unified Modeling Language) yra nuosekli kalba skirta specifikuoti ir grafiškai atvaizduoti sistemos komponentus. Programinės įrangos architektai gali naudoti ja apibrėžiant, vaizduojant, konstruojant ir dokumentuojant projektus.

API (Application Programming Interface) leidžia programinę įrangą naudoti kaip komponentą. Tai užtikrina, kad kita sistema galės vykdyti veiksmus įgyvendintus joje aplenkiant grafinę vartotojo sąsają.

2. Problemos sprendimas pasaulyje

Automatinis kodo generavimas iš vizualiai atvaizduotos logikos yra novatoriškas būdas papildyti sistemos galimybes, todėl analogų kuriamai programinei įrangai nėra daug. Produktas „Visustin v5 Flow chart generator“ gali atvaizduoti programinį kodą į diagramas, panašias į UML veiklos diagramas. Taip pat galima atlikti atvirkščią veiksmą.

Šiuo įrankiu galimas automatizuotas programos įgyvendinimo procesas. Užtenka algoritmui suprojektuoti veiklos diagramą ir ji bus realizuota. Ši programa leidžia suprasti programinį kodą nesigilinant į programinės kalbos ypatybes ar sintaksę.

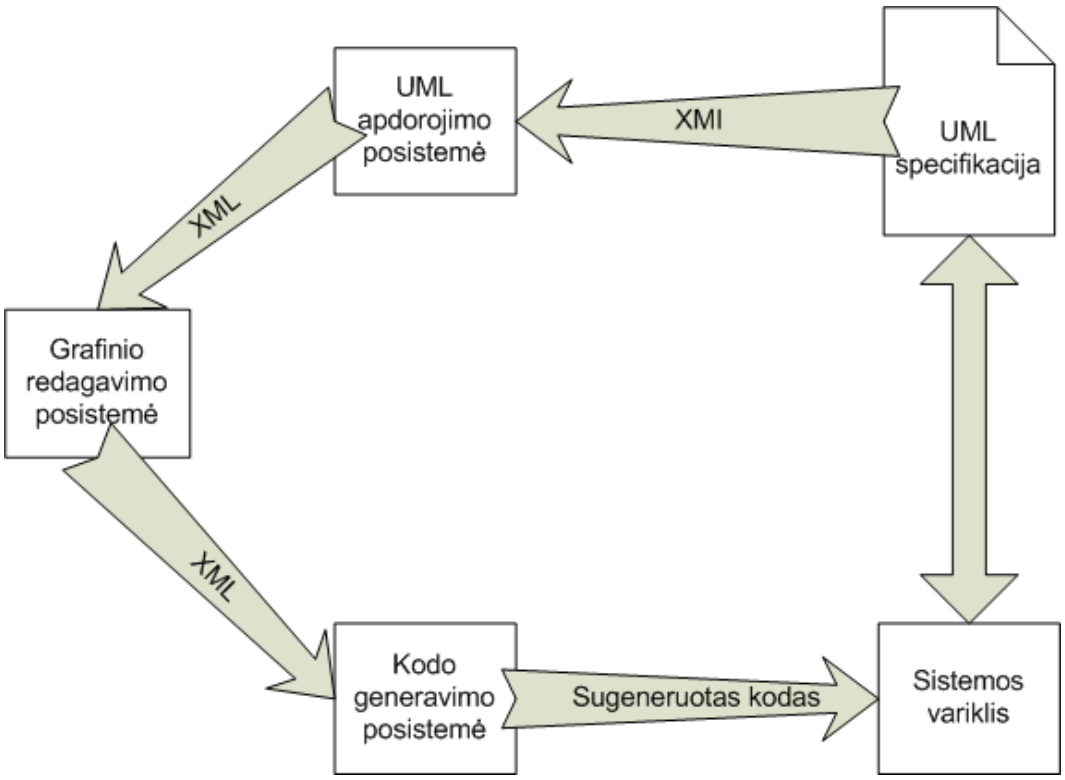
Šis produktas priartina projektavimą prie Executable UML. Executable UML leidžia iš anksto patikrinti programos kodą, sugeba išversti UML modelį tiesiai į efektyvų programinį kodą, ir leidžia atidėti įgyvendinimo sprendimus, iki paskutinės minutės[3].

Tačiau ši programinė įranga labiau pritaikyta ne naudoti jau veikiančios sistemos galimybes, o kurti naujiems algoritmams.

3. Siūlomas sprendimas

Kuriamas įrankis leidžia išplėsti sistemos funkcionalumą turint elementarias programavimo žinias. Naujos sistemos galimybės yra modeliuojamos grafiškai, o įrankis transformuoja modelį į aktyvų sistemos kodą. Šį įrankį galima suskaidyti į tris komponentus (UML apdorojimo, grafinė scenarijų kūrimo ir automatinė kodo generavimo posistemes).

- Sistemos UML apdorojimo posistemė analizuoja klasių diagramą ir atrenka atvirai prieinamas klases bei jų metodus. Šie duomenys yra perduodami į grafinę redagavimo posistemę XML formatu.
- Grafinė scenarijų kūrimo posistemė atvaizduoja sistemos klases ir metodus kaip galimų naudoti objektų aibę. Šia aibe modeliuotojas galės operuoti įgyvendindamas naujus sistemos panaudos atvejus ir pridėti elementarią logiką (sąlygos sakinius, sudaryti ciklus). Šios posistemės išvedami duomenys perduodami į kodo generavimo posistemę XML formatu.
- Kodo generavimo posistemė analizuoja panaudos atvejų modelį atpažindama elementariąją logiką ir transformuoja į galutinį programos kodą. Šios posistemės pagrindinis principas sudaryti programinį kodą architektūriniu požiūriu skirtingoms sistemoms. Tam įgyvendinti naudojama įskiepių technologija, kur įskiepis gali turėti taisykles būdingas specifinei architektūrai ar programavimo kalbai.



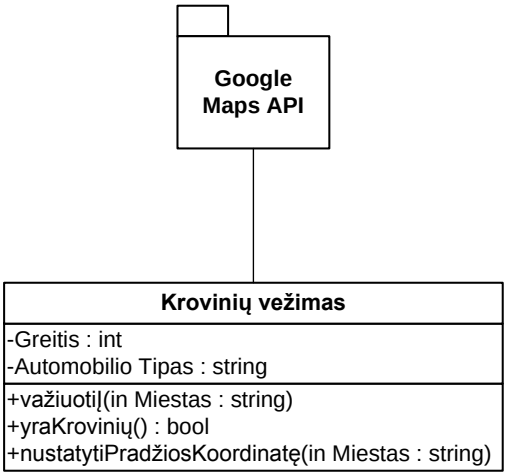
1 pav. Įrankio architektūros diagrama

4. Įrankio veikimo pavyzdys

Pavyzdinė sistema realizuota su „Google maps“ įskiepiu. Trumpas scenarijaus aprašymas:

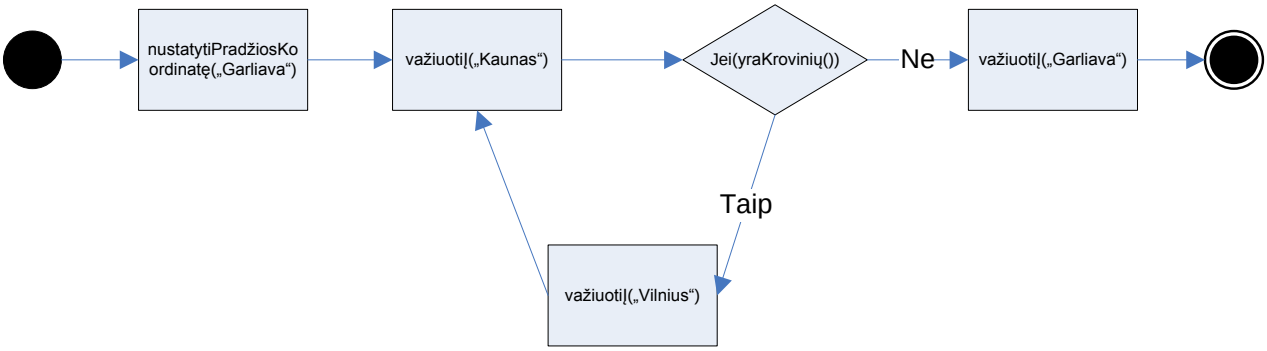
Vartotojui reikia sudaryti krovinio vežimo scenarijaus iš Kauno į Vilnių demonstraciją. Įmonės būstinė yra Garliavoje. Vairuotojas važiuoja į Kauną, kur tikrinama ar yra pervežimo užsakymų. Jeigu užsakymų yra, krovinys vežamas į Vilnių ir grįžtama atgal. Ši procedūra kartojama tol, kol kroviniai baigiasi. Tada vairuotojas grįžta į būstinę.

Sistemos UML specifikacijos pavyzdys:



2 pav. Pervežimų sistemos UML klasių diagrama

Grafinio redaktoriaus sumodeliuotas vaizdas:



3 pav. Krovinio vežimo grafinis modelis

Grafinio redaktoriaus XML išvestis:

```
<Busena id="start">
	<next id="B1" />
</Busena>

<Busena id="B1">
	<function name="nustatytiPradziosKoordinate">
		<param value="Garliava" />
	</function>
<next id="B2" />
</Busena>

<Busena id="B2">
	<function name="vaziuotiI">
		<param value="Kaunas" />
	</function>
	<next id="B3" />
</Busena>

<Busena id="B3">
	<function name="if">
		<param name="yraKroviniu" />
		<true id="B4" />
		<falše id="B2" />
	</function>
</Busena>
```

Sugeneruotas programinis kodas:

```
function B1(){nustatytiPradziosKoordinate("Garliava"); B2();}
function B2(){vaziuotiI("Kaunas"); B3();}
function B3(){
      if(yraKroviniu()) B4();
      else B5();
}
function B4(){vaziuotiI("Vilnius"); B2();}
function B5(){vaziuotiI("Garliava");}
```

5. Išvados

Straipsnyje pateiktas įrankio prototipas leidžia atskirti sistemos programuotojo ir dizainerio darbą. Taip padidinant darbo našumą ir supaprastinant naujų panaudos atvejų kūrimo procesą. Taip pat naudojant įskiepių technologiją užtikrinamas greitas atsakas į sistemos architektūros pakeitimus.

Tačiau tokiu būdu sugeneruotas kodas yra sunkiau skaitomas, todėl veikimo pakeitimai turės būti atliekami tik šio įrankio pagalba.

6. Literatūros sąrašas

[1] **Kyle Wilson, Data-Driven Design** [Žiūrėta 2009 04 03], prieiga internete < http://www.gamearchitect.net/Articles/DataDrivenDesign.html>

[2] **Lost Garden, Managing game design risk: Part II - Data Driven Development** [Žiūrėta 2009 04 03], prieiga internete < http://lostgarden.com/2006/04/managing-game-design-risk-part-ii-data.html>

[3] **Stephen J. Mellor, Executable UML** [Žiūrėta 2009 04 03], prieiga internete <http://www.techonline.com/article/pdf/showPDF.jhtml?id=1931036231>