KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Kęstutis Valinčius

DDD metodologija paremto projektavimo įrankio kodo generatoriaus kūrimas ir tyrimas

Magistro darbas

Darbo vadovas:
doc. dr. Tomas Blažauskas

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Kęstutis Valinčius

DDD metodologija paremto projektavimo įrankio kodo generatoriaus 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.
Kęstutis Valinčius
2010-05-28

Kaunas, 2010

DDD methodology based design tool‘s code generator development and research

Kęstutis Valinčius

SUMMARY

Data Driven Design methodology is widely used in various program systems. This methodology aim is to distinguish and parallel software developer and scenario designer’s work. Core functionality is implemented via interfaces and dynamics via scenario support. This introduces a level of abstraction, which makes software product more flexible easily maintained and improved, in addition these actions can be performed in parallel.

The main aim of this work was to create automatic code generator that transforms graphically modeled scenario to software code. Automatically generated software code restricts probability of syntactic and logical errors, all depends on scenario modeling. Code is generated instantly and no need software developer interference. This aim is achieved by moving business logic designing to scenario designing process and code generator service making as a “Web service”. Using cartridge based system code is generated not attached to a specific architecture, technology or application domain. In graphical scenario modeling tool scenario is modeled and transformed to metalanguage, from which software code is generated. Metalanguage – with specific rules defined “XML” language.

Experimental system was developed with no major problems. New project modeling with our modeling tool speeded the development process by seven times. This proves modeling tool advantage over manual programming.

Santrauka

Data Driven Design[1] metodologija plačiai naudojama įvairiose programinėse sistemose. Šios metodologijos tikslas – atskirti bei lygiagretinti programuotojų ir projektuotojų 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.

Darbo tikslas buvo sukurti automatinį kodo generatorių, kuris transformuotų grafiškai sumodeliuotą scenarijų[2] į programinį kodą. Generuojant programinį kodą automatiškai ženkliai sumažėja sintaksinių bei loginių klaidų tikimybė, viskas priklauso nuo sumodeliuoto scenarijaus. Kodas sugeneruojamas labai greitai ir visiškai nereikalingas programuotojo įsikišimas. Šis tikslas pasiektas iškėlus biznio logikos projektavimą į scenarijaus projektavimą, o kodo generavimo posistemę realizavus žiniatinklio paslaugos[3] principu. Kodas generuojamas neprisirišant prie konkrečios architektūros, technologijos ar taikymo srities panaudojant įskiepių sistemą[4]. Grafiniame scenarijų kūrimo įrankyje sumodeliuojamas scenarijus ir tada transformuojamas į metakalbą[5], iš kurios ir generuojamas galutinis programinis kodas. Metakalba – tam tikromis taisyklėmis apibrėžta „XML[6]“ kalba.

Realizavus eksperimentinę sistemą su didelėmis problemomis nebuvo susidurta. Naujos sistemos modeliavimas projektavimo įrankiu paspartino kūrimo procesą septynis kartus. Tai įrodo modeliavimo pranašumą prieš rankiny programavimą.

[1] Data Driven Design (DDD) – duomenimis paremta architektūra.
[2] Script – scenarijus.
[3] WEB service – žiniatinklio paslauga. [11]
[4] Cartridge system – įskiepiais paremta sistema.
[5] Metakalba - specialios paskirties kalba, skirta kitų kalbų aprašymui.
[6] XML (Extensible Markup Language) – bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba.

Turinys

Paveikslėlių turinys

Lentelių turinys

1. ĮVADAS

1.1. Tiriamoji problema

Šiame darbe nagrinėjamas įskiepiais paremtas programinio kodo generatorius transformuojantis scenarijų iš metakalbos į taikymo srities programinį kodą.

1.2. Temos aktualumas

Kompiuterių mokslui tobulėjant programavimo procesas tampa vis brangesnis laiko atžvilgiu, dėl vis sudėtingesnių sistemų kūrimo, todėl stengiamasi kaip įmanoma automatizuoti šį procesą įvairiais automatizavimo metodais, vienas iš jų – naudoti programinio kodo generatorius.

Kodo generavimo procesas vyksta transformuojant modelį iš vieno abstrakcijos lygmens į kitą. Taip sumažinamas sistemos kūrimo detalumas. Kuo aukštesnis abstrakcijos lygmuo tuo mažiau galvojama apie programavimo ypatybes, bet daugiau laiko skiriama biznio logikos išbaigtumui ir projektavimui.

Labai svarbu modelio transformavimą įgyvendinti neprisirišant prie konkrečios architektūros ar programavimo kalbos, taip pasiekiamas ilgesnis projekto gyvavimo ciklas ir išvengiama didelių projekto palaikymo ir tobulinimo kaštų. Sujungus modeliavimo įrankį su automatiniu kodo generavimu, nepririštu prie konkrečios architektūros, gaunamas galingas projektavimo įrankis užtikrinantis patogų didelių projektų palaikymą ir greitą jų vystymą.

2. PROJEKTAVIMO IR DIEGIMO ĮRANKIO ANALIZĖ

2.1. Bendra sistema

2.1.1. 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).

2.1.2. 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.

2.1.3. 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;

2.1.4. Darbo metodai ir priemonės

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

2.1.4.1. Grafinio modelių redaktorius

Posistemė sukurta „Microsoft Visual Studio 2008“ kūrimo aplinkoje naudojant „Microsoft Silverlight[7]“ karkasą.

2.1.4.2. Kodo generatorius

Posistemė realizuota PHP[8] programavimo kalba ir talpinama dedikuotame serveryje. Kūrimo aplinka – „NuSpharePhpED 5.9 profesional“.

2.1.4.3. 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“.

[7] Silverlight – žiniatinklio sistemų karkasas.
[8] PHP (Hypertext Preprocessor) – dinaminė interpretuojama programavimo kalba.

2.1.4.4. Hipertekstinės grafinės vartotojo sąsajos biblioteka

Posistemė sukurta deklaratyvia XML (HTML[9]) bei imperatyvia JavaScript[10] programavimo kalba, naudojant „Microsoft Visual Studio 2008“ integruotą programų kūrimo aplinką..

2.1.5. Panašių sprendimų apžvalga

Šiuo metu analogiškų projektavimo įrankių pasaulyje nėra. Tačiau galima apžvelgti panašius įrankius, kurie palaiko Modeliu Paremtos Architektūros[11] principus. Populiariausi ir daugiausiai naudojami yra šie:

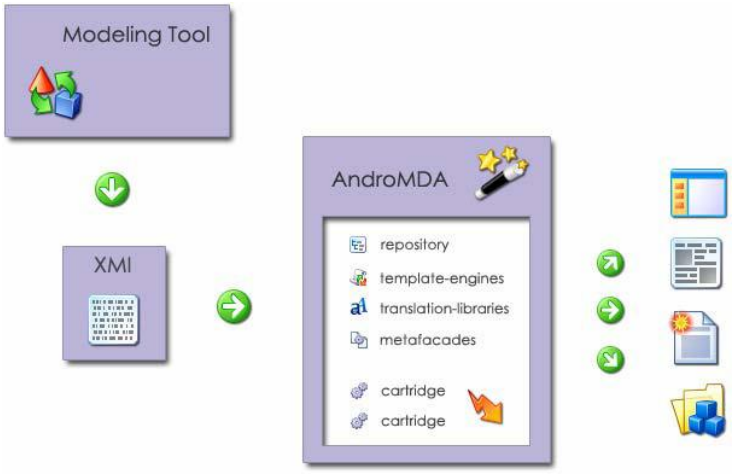
- „AndroMDA“
- „Rational XDE MDA Toolkit“
- „ArcStyler 4.0“
- „OpenMDX“
- „OptimalJ“

2.1.5.1. „AndroMDA“

„AndroMDA“ yra atviro kodo programinė įranga. Šiuo įrankiu galima generuoti J2EE[12] projektus, realizuotus Java programavimo kalba iš UML modelių. Generuojami žiniatinklio projektai pritaikyti Hibernate, EJB, Struts, Spring ir WebServices technologijoms.

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

[9] HTML – Hiperteksto žymėjimo kalba.
[10] JavaScript – objektiškai orientuota skriptų programavimo kalba.
[11] Model Driven Engineering (MDA) – modeliu paremta architektūra.
[12] Java 2 Platform, Enterprise Editon (J2EE) – standartinė daugialyčių programų kūrimo Java kalba platforma.



Pav. 1. „AndroMDA“ veikimas

Programų sistemos generavimas naudojant „AndroMDA“ veiksmų seka:

- UML[13] įrankiu („MagicDraw“, „Poseidon UML“, „Rational Rose“) sukuriamas sistemos nuo Platformos Nepriklausomas Modelis[14] ir eksportuojamas į XMI[15] dokumentus.
- Iškviečiama „AndroMDA“ programa kuri atlieka reikalingas transformacijas.

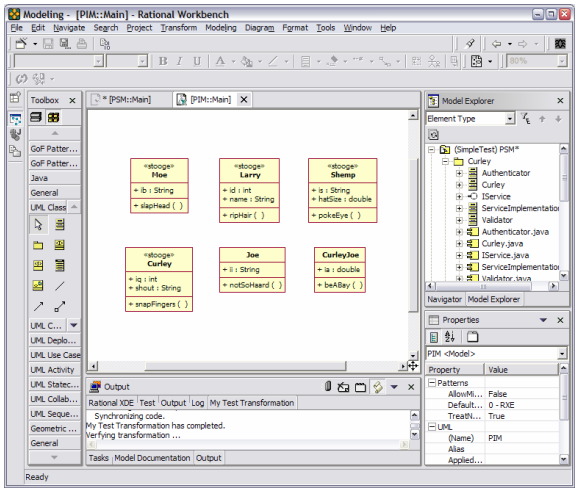
Transformacijos aprašomos įskiepių sistema. Todėl vartotojas turi galimybę kurti savo transformacijos modelius. Šie modeliai aprašomi šabloniniu principu Java[16] programavimo kalba.

Darbas su įrankiu vyksta terminalo[17] pagalba, todėl valdymas labai nepatogus, tačiau kaip kūrėjai teigia itin lankstus.

2.1.5.2. „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).

[13] Unified modeling language (UML) – Unifikuota Modeliavimo kalba.
[14] Platform Independent Model (PIM) – nuo platformos nepriklausomas modelis.
[15] XMI – meta modelio aprašomoji kalba.
[16] Java – objektiškai orientuota programavimo kalba.
[17] 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.

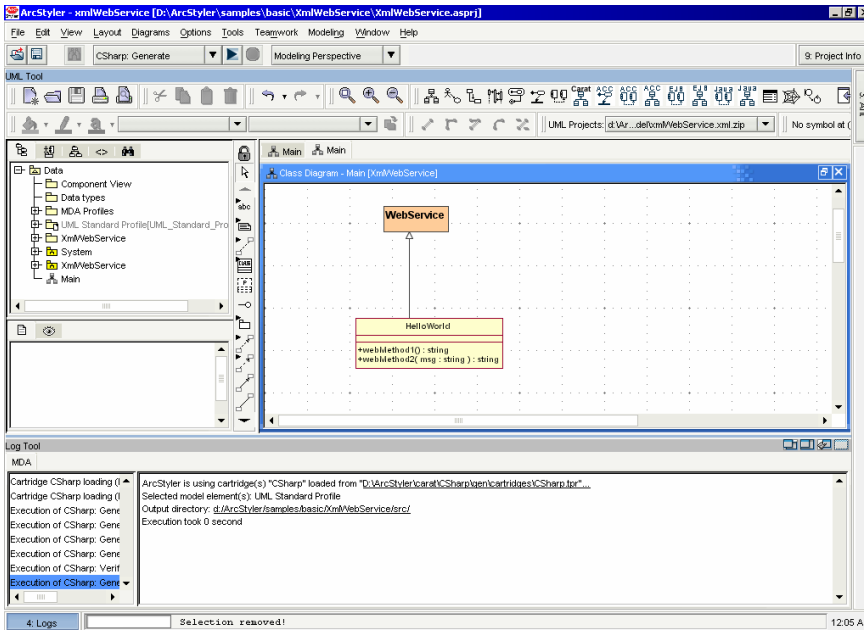
2.1.5.3. „Arcstyler 4.0“

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 [1].

„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

2.1.5.4. „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[18], J2EE, CORBA[19], .NET platformose.
- Žinomiausių modeliavimo įrankių palaikymas („Rational Rose“, „Poseidon UML“, „MagicDraw“).
- Aspektais paremto programavimo palaikymas.

[18] Java 2 Platform, Standart Editon (J2SE) – Java bazinės bibliotekos.
[19] CORBA – standartų sistema koordinuotam kelių programų darbui internete.

2.1.5.5. „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[20], vadinami šablonais[21]. „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.

2.1.5.6. MDA įrankių įvertinimas

Įvertinsime aukščiau aprašytus modeliavimo įrankius pagal šiuos kriterijus:

- PIM palaikymas. Ar yra galimybė sudaryti PIM modelį.
- PSM[22] palaikymas. Ar yra galimybė sudaryti PSM modelį.

[20] Framework – karkasas.
[21] Pattern – šablonas.

- 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[23] 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.
- UML įrankis naudojamas „Poseidon UML“. Įrankis priima modelius iš „Poseidon UML“ modeliavimo įrankio.

[22] Platform Specific Model (PSM) – nuo platformos priklausomas modelis.
[23] Refactoring – automatinis programinio kodo fragmentų pervadinimas.

UML įrankių įvertinimo rezultatai pateikiami lentelėje „Lentelė Nr. 1“. [1]

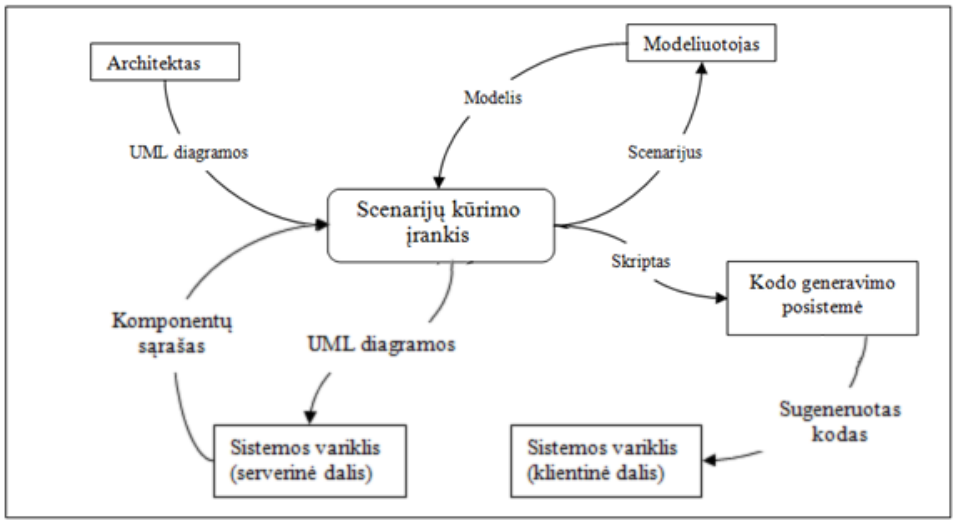
Lentelė Nr. 1. UML įrankių įvertinimo rezultatai

Kriterijus	**Andro MDA**	**Rational XDE MDA Toolkit**	**ArcStyler**	**Open MDX**	**OptimalJ**
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“.

2.1.6. 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“, o aprašymas lentelėje „Lentelė Nr. 2“.



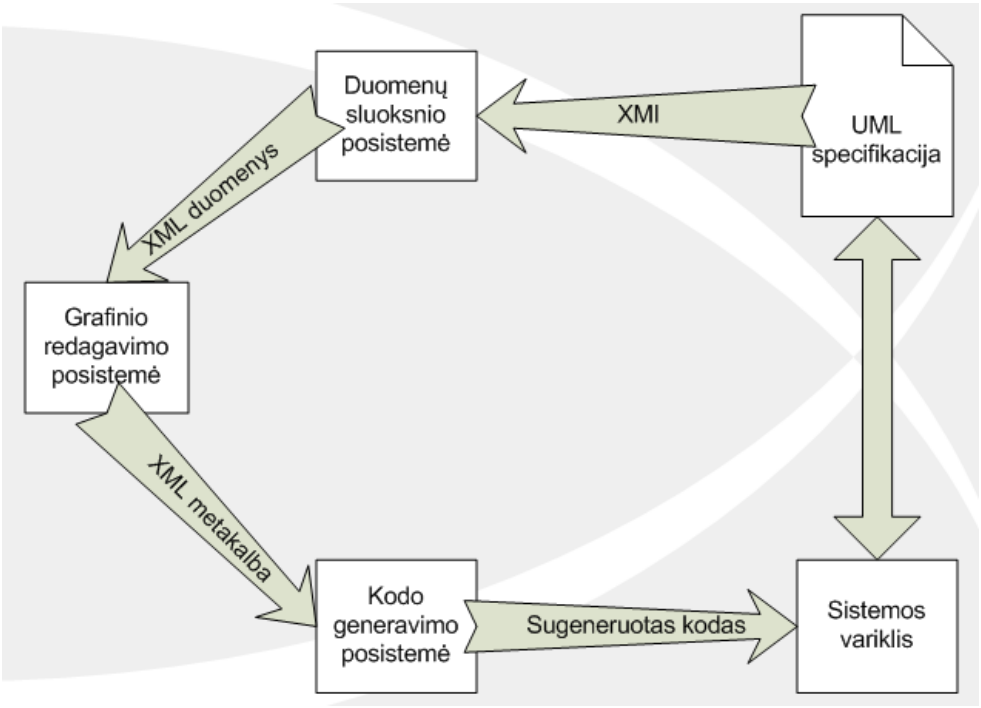
Pav. 4. Bendros sistemos veiklos sfera

Lentelė Nr. 2. 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)
3.	Sugeneruotas kodas perduodamas sistemos variklio klientinei daliai.	Sugeneruotas kodas (į)
4.	Sistemos architektas apibrėžia sistemos funkcionalumą UML diagramomis.	UML diagramos (į)
5.	Sistemos variklio serverinei dalis apdoroja UMS diagramas	UML diagramos (į) Komponentų sąrašas (iš)

2.1.7. Veiklos pasidalinimas

Projektavimo įrankį sudaro keturios posistemės. Bendra įrankio architektūros diagrama pateikta paveiksle „Pav. 5“.



Pav. 5. Įrankio architektūros diagrama

2.1.7.1. DDD metodologija paremto projektavimo įrankio grafinio modelių redaktoriaus kūrimas ir tyrimas

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

2.1.7.2. DDD metodologija paremtos sistemos kodo generatoriaus kūrimas ir tyrimas

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

2.1.7.3. DDD metodologija paremtos sistemos duomenų sluoksnio modelio kūrimas ir tyrimas

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

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

Hipertekstinės grafinės vartotojo sąsajos[24] 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.

[24] GUI (Graphical User Interface) – grafinė vartotojo sąsaja.

2.2. Kodo generavimo posistemė

2.2.1. Sistemos paskirtis

Kuriamas projektavimo įrankis turi tikti keletui taikymo sričių, architektūriniu požiūriu skirtingoms sistemoms. Kodo generavimo posistemė atsakinga už galutinio programinio kodo generavimą iš papildomo abstrakcijos lygmens (scenarijaus) nepriklausomai nuo taikymo srities.

2.2.2. Projekto kūrimo pagrindas

Galutinų sistemų, kurioms pritaikomas mūsų kuriamas įrankis, architektūra gali skirtis, net ir programavimo kaba gali skirtis. Todėl reikalinga kodo generavimo posistemę kurti universalią ir nepriristą prie taikymo programavimo kalbos ar architektūros. Šis funkcionalumas užtikrinamas panaudojus įskiepiais paremtą sistemą. Kiekvienas atskiras įskiepis – atskira taikymo sritis. Taip kiekvienas įskiepis tampa nepriklausomas ir atlieka tik tam tikrai taikymo sričiai pritaikytą kodo generavimą. Prireikus įrankį papildyti kita taikymo sritimi – suprojektuojamas ir realizuojamas papildomas įskiepis [3].

2.2.3. Egzistuojantys sprendimai pasaulyje

Analogiškų sistemų pasaulyje nėra, todėl kad kodo generavimo posistemė yra labai specifinė ir graudžiai susieta su bendra sistema ir kaip atskira dalis netenka vertės. Bandymas ją pritaikyti kitos sistemos kontekste reikštų visišką jos perdarymą.

Galima panaudoti pakartotinai[25] tik atskirus komponentus, tokius kaip programinio kodo optimizatorių, taisyklingumo patikrą.

Anksčiau 2.1.5 skyriuje (Panašių sprendimų apžvalga) apžvelgtų MDA įrankių panašų funkcionalumą turi šie įrankiai:

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

[25] Reuse – panaudoti pakartotinai.

2.2.4. Sistemos tikslai

- Užtikrinti nepriklausomą kodo generavimą nuo taikymo srities ir naudojamų technologijų ar programavimo kalbos. Šis funkcionalumas bus užtikrintas panaudojus įskiepių sistemą.
- Lengvas pritaikomumas naujai taikymo sričiai. Naujo įskiepio suprojektavimas ir realizavimas turi būti kaip įmanoma paprastesnis. Statinė kodo analizė scenarijui bus vykdoma prieš kodo generavimą, taip bus užtikrinamas universalumas visiems įskiepiams.
- Tarpinės metakalbos (skripto) specifikavimas. Bus sukuriamas taisyklių rinkinys aprašantis kaip turi būti užrašytas scenarijus metakalba.

2.2.5. Iškilusios problemos

Projektavimo įrankio pagalba sumodeliuojami objektai ir tiems objektams priskiriamas funkcionalumas (sumodeliuojama veiksmų seka[26]). Papildomos problemos kilo sumodeliuoto funkcionalumo generavimo srityje.

2.2.5.1. Kodo generavimui

Papildomos problemos iškilo objektui sumodeliuoto funkcionalumo analizėje (statinė kodo analizė). Reikalinga aptikti šiuos programinio kodo fragmentus:

- Sąlygos sakiniai.
- Ciklo sakiniai.

2.2.5.2. Kodo optimizavimui

Modeliuojant didelę sistemą, kai sugeneruojama labai daug programinio kodo reikalinga atsižvelgti į sugeneruoto kodo optimizavimą. Pašalinant nenaudojamus kodo fragmentus, o naudojamus suoptimizuoti tiek dydžio tik greitaveikos prasmėmis.

[26] Flow chart – nuoseklus funkcionalumo scenarijus (veiklos diagrama).

3. PROJEKTINĖ DALIS

3.1. Bendra sistema

3.1.1. Architektūros tikslai ir apribojimai

Kuriamos sistemos architektūros tikslai:

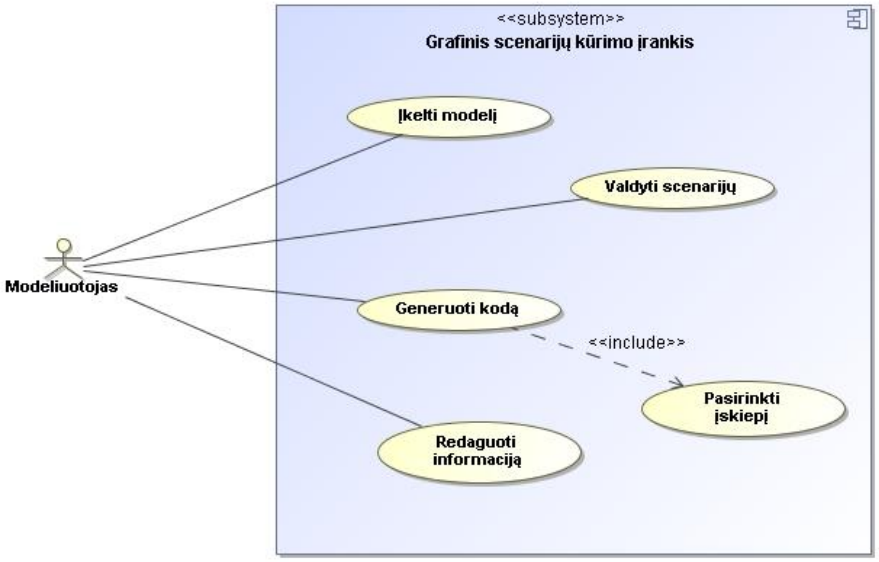
- Sistema bus galima naudotis visais kompiuteriais turinčiais interneto naršyklę ir prieigą prie interneto.
- Duomenimis paremta architektūra. Paduodame UML diagramas ir pagal tai modeliuojame scenarijus grafiniame redaktoriuje.
- Nepriklausoma taikymo sritis. Kiekviena taikymo sritis aprašoma kaip įskiepis kodo generavimo posistemėje.

Kuriamos sistemos apribojimai:

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

3.1.2. Panaudojimo atvejai

Projektavimo įrankio pagrindiniai panaudos atvejai pateikti paveiksle „Pav. 6“, o jų aprašymai lentelėse „Lentelė Nr. 3“ – „Lentelė Nr. 7“.



Pav. 6. Panaudos atvejų vaizdas

Lentelė Nr. 3. 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. 4. 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. 5. 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. 6. 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.

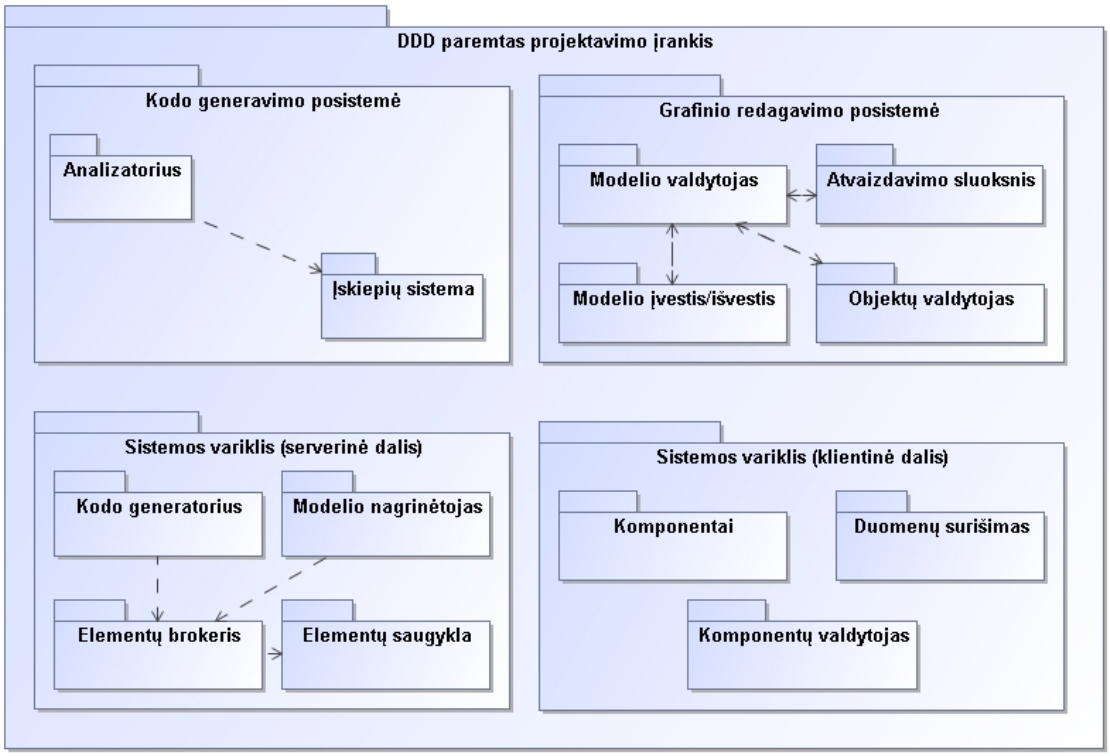
Lentelė Nr. 7. Panaudos atvejis „Redaguoti informaciją”

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.

3.1.3. Sistemos statinis vaizdas

Sistema suskirstyta į šiuos paketus (Pav. 7):

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



Pav. 7. Projektavimo įrankio paketų diagrama

3.1.3.1. 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.

3.1.3.2. 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.

3.1.3.3. 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.

3.1.3.4. 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.

3.1.4. 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. 8“, minimalūs reikalavimai vartotojo techninei įrangai pateikiami lentelėje „Lentelė Nr. 9“. Reikalavimai serverio programinei įrangai pateikiami lentelėje „Lentelė Nr. 10“, minimalūs reikalavimai serverio techninei įrangai pateikiami lentelėje „Lentelė Nr. 11“.

Lentelė Nr. 8. 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. 9. Minimalūs reikalavimai vartotojo techninei įrangai

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

Lentelė Nr. 10. 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. 11. 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.

3.2. Kodo generavimo posistemė

3.2.1. Funkciniai reikalavimai

Kodo generavimo posistemei suformuluoti šie funkciniai reikalavimai:

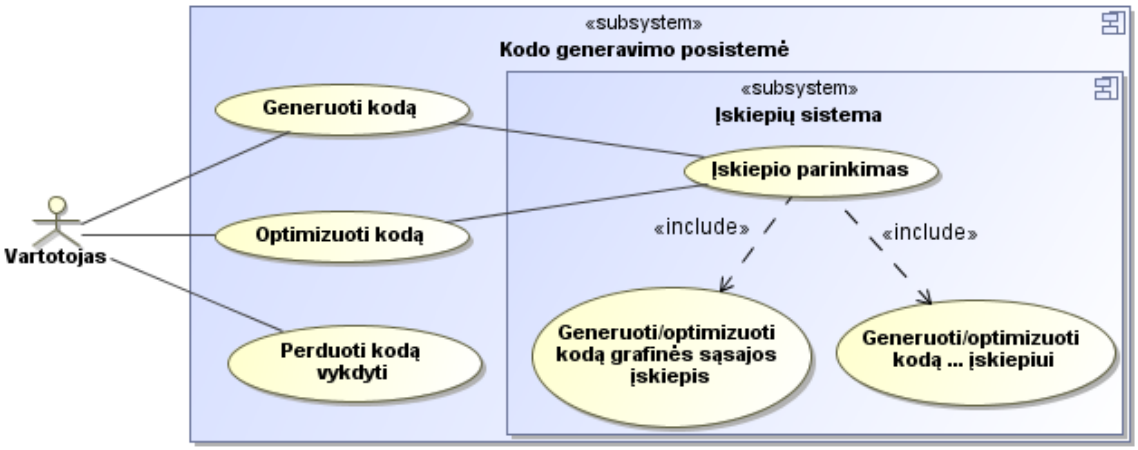
- Veikti žiniatinklio paslaugos principu.
- Galimybė pasirinkti vieną arba keletą taikymo sričių.
- Generuoti galutinį, realiai veikiantį kodą.

3.2.2. Nefunkciniai reikalavimai

- Aptikti sąlygos sakinius.
- Aptikti ciklo sakinius.
- Lengvai sukuriami įskiepiai.
- Greitas veikimas, nes gali būti naudojamas kaip realaus laiko kodo generatorius.
- Automatiškai aptikti ir parinkti reikalingą kodo generavimo įskiepį.
- Sugeneruoto JavaScript kodo optimizatorius.

3.2.3. Panaudos atvejai

Kodo generavimo posistemės panaudos atvejai pavaizduoti paveiksle „Pav. 8“, o jų aprašymai lentelėse „Lentelė Nr. 12“ – „Lentelė Nr. 16“.



Pav. 8. Kodo generavimo posistemės panaudos atvejai

Lentelė Nr. 12. Panaudos atvejis "Generuoti kodą"

Generuoti kodą	
Aprašas:	Generuoti programinį kodą iš scenarijaus skripto.
Vartotojas/Aktorius:	Grafinė scenarijų kūrimo posistemė.
Prieš-sąlyga:	Sumodeliuotas scenarijus.
Sužadinimo sąlyga:	Baigtas modeliuoti scenarijus.
Po-sąlyga:	Kviečiamas įskiepio parinkimas.

Lentelė Nr. 13. Panaudos atvejis "Optimizuoti kodą"

Optimizuoti kodą	
Aprašas:	Optimizuoti sugeneruotą programinį kodą.
Vartotojas/Aktorius:	Grafinė scenarijų kūrimo posistemė.
Prieš-sąlyga:	Sugeneruotas kodas.
Sužadinimo sąlyga:	Baigtas modeliuoti scenarijus.
Po-sąlyga:	Kviečiamas įskiepio parinkimas.

Lentelė Nr. 14. Panaudos atvejis "Perduoti kodą vykdyti"

Perduoti kodą vykdyti	
Aprašas:	Perduoti sugeneruotą kodą taikymo srities klientinei daliai.
Vartotojas/Aktorius:	Grafinė scenarijų kūrimo posistemė.
Prieš-sąlyga:	Sugeneruotas kodas.
Sužadinimo sąlyga:	Baigtas modeliuoti scenarijus.
Po-sąlyga:	Sugeneruotas kodas atitinka taikymo srities reikalavimus.

Lentelė Nr. 15. Panaudos atvejis "Įskiepio parinkimas"

Įskiepio parinkimas	
Aprašas:	Įskiepio parinkimas pagal nurodymą arba automatiškai.
Prieš-sąlyga:	Inicijuotas kodo generavimas ar optimizavimas.
Sužadinimo sąlyga:	Baigtas scenarijaus skripto analizavimas.
Po-sąlyga:	Parenkamas atitinkamas įskiepis.

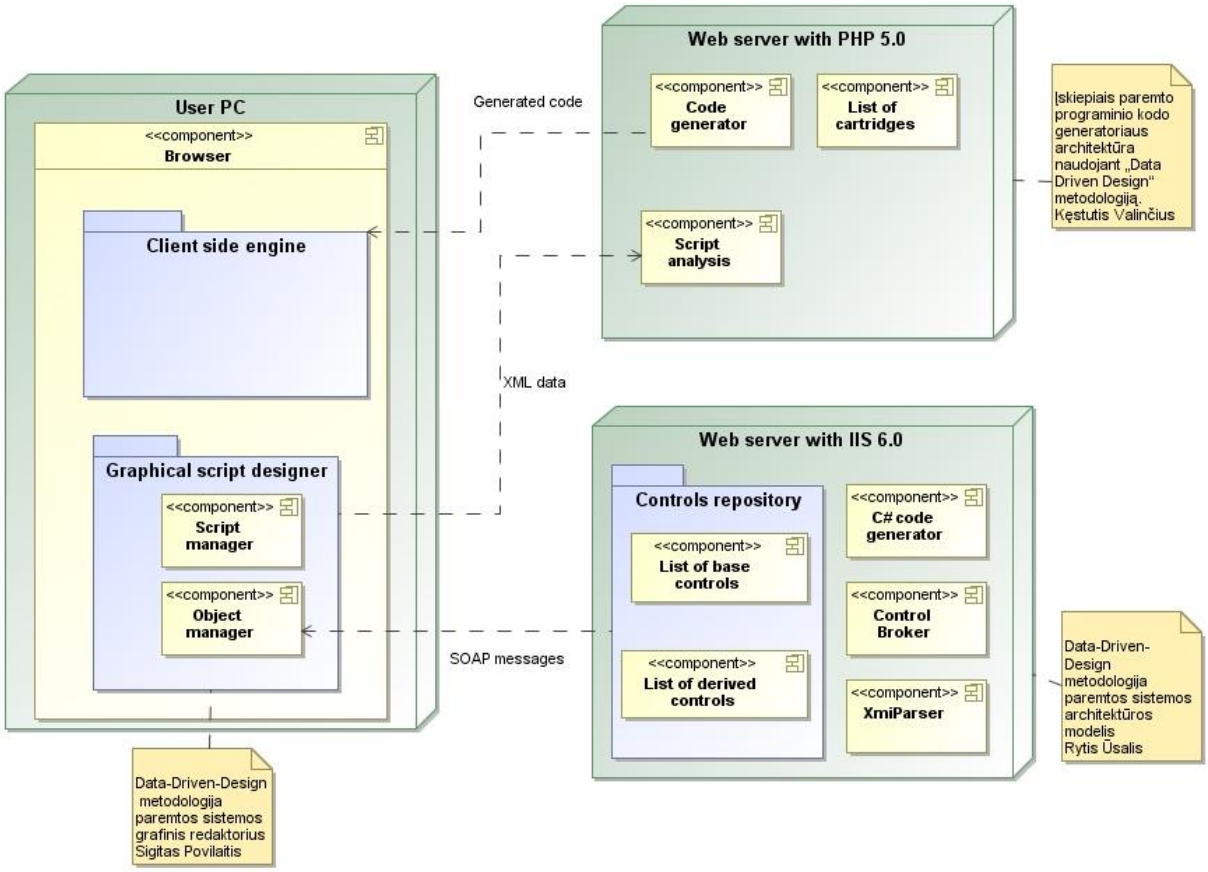
Lentelė Nr. 16. Panaudos atvejis "Generuoti/optimizuoti naudojant grafinės sąsajos įskiepį"

Generuoti/optimizuoti kodą naudojant grafinės sąsajos įskiepį	
Aprašas:	Generuojamas programinis kodas naudojant grafinės sąsajos įskiepį.
Prieš-sąlyga:	Parinktas įskiepis.
Sužadinimo sąlyga:	Parinktas grafinės sąsajos įskiepis.
Po-sąlyga:	Sugeneruotas/optimizuotas programinis kodas.

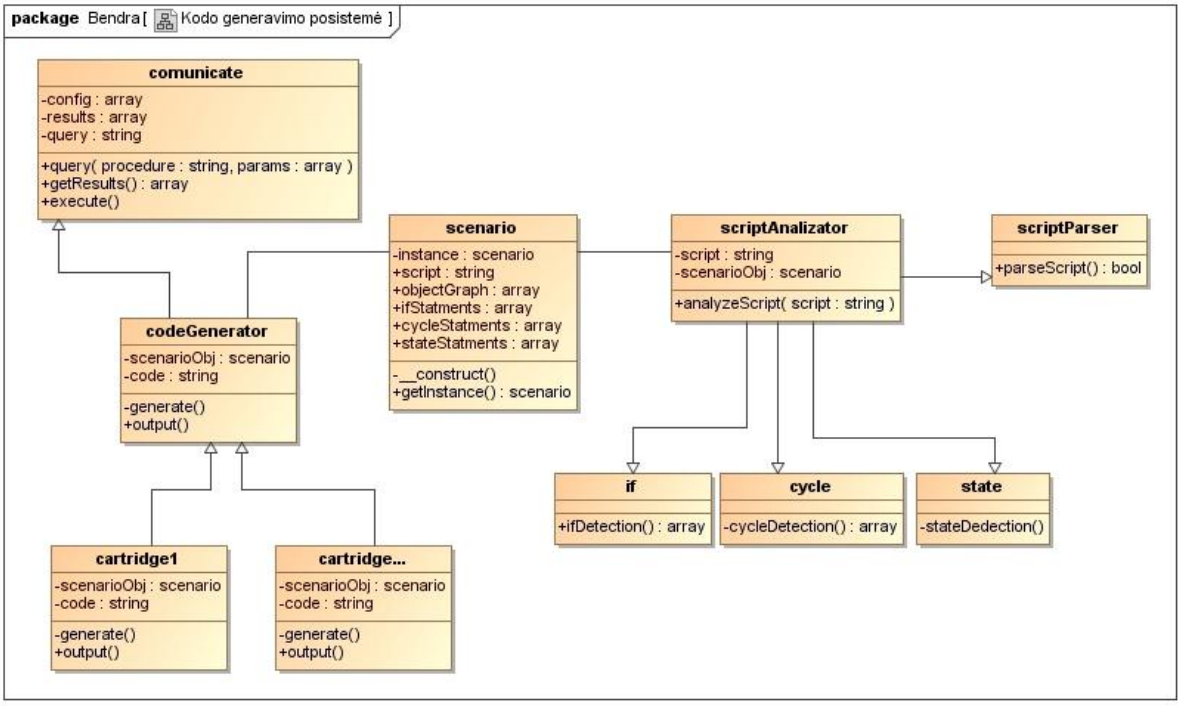
3.2.4. Architektūros specifikacija

3.2.4.1. Sistemos statinis vaizdas

Kodo generavimo posistemės statinis vaizdas, bendrame kontekste, pateiktas paveiksle „Pav. 9". Klasių diagrama paveiksle „Pav. 10".



Pav. 9. Sistemos statinis vaizdas

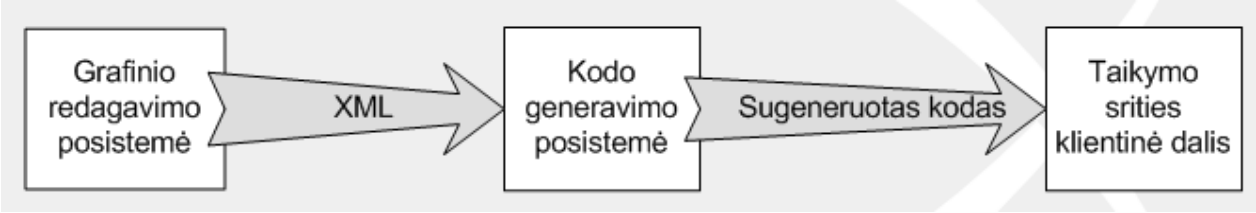


Pav. 10. Klasių diagrama

Posistemei realizuoti buvo pasirinkta PHP programavimo kalba. Šis sprendimas pasirinktas dėl mažų palaikymo kaštų bei pakankamos patirties norint įgyvendinti užsibrėžtiems tikslams.

3.2.4.2. Bendravimas su kitomis sistemos dalimis

Grafinėje scenarijų kūrimo posistemėje sumodeliuojamas scenarijus ir transformuojamas į skriptą (metakalbą). Šis skriptas perduodamas kodo generavimo posistemei generuoti galutinį, tam tikrai taikymo sričiai skirtą, programinį kodą [4]. Sugeneruotas kodas perduodamas taikymo srities klientinei daliai (priklauso nuo taikymo srities reikalavimų) (Pav. 11).



Pav. 11. Bendravimas su kitomis sistemomis

3.2.4.3. Scenarijaus metakalbos sintaksė

Išanalizavus bendrus posistemių bendravimo principus buvo nuspręsta scenarijaus metakalbą apibrėžti remiantis XML [5].

Scenarijaus metakalba – tam tikromis taisyklėmis apibrėžta žymių[27] XML kalba [6]. Žymuo scenarijuje apibrėžia atskirą komponentą, ar jo konfigūraciją. Visos galimos žymos ir jų konfigūracijos aprašytos lentelėje „Lentelė Nr. 17“.

Veiklos diagrama scenarijaus skripte aprašoma kaip grafas. Aprašoma viena viršūnė su nuoroda į kitą viršūnę. Pradžios viršūnė identifikuojama „Start“ požymiu, o pabaigos – „End“.

Scenarijaus skripto pavyzdys nurodomas paveiksle „Pav. 12“.

[27] Tag – Metakalbos žymuo.

Lentelė Nr. 17. Metakalbos taisyklės

Žymuo	Apibūdinimas
`<scenrio>`	Scenarijaus aprašymo žymė. Atributai naudojami bendriems nustatymams. Nenurodžius naudojami nutylėti. Galimi atributai: • „cartridge“ – naudojamas įskiepis. • „optimization“ – optimizacijos lygmuo (priklauso nuo tam tikro įskiepio).
`<item>`	Scenarijaus komponento aprašymo žymė. Galimi šie atributai: • „type“ – nurodo komponento tipą. Galimos šios reikšmės: o „object“ – komponentas yra naujo objekto kūrimas. o „function“ – komponentas yra metodo iškvietimas. • „class“ – jeigu komponentas yra naujo objekto kūrimas tada nurodoma kuriamo objekto klasės pavadinimas, priešingu atveju šis atributas nenaudojamas. • „name“ – kintamojo pavadinimas kuriam priskiriamas komponentas (PVZ.: kuriamo objekto kintamojo vardas)
`<param>`	Komponento parametrų aprašymo žymė. Galimas dvejopas naudojimas. • Visi parametrai aprašomi viena žyme su daug atributų. • Kiekvienas parametras aprašomas atskira žyme.
`<FlowChart>`	Veiklos diagramos aprašymo žymė. Galimas atributas „type“ – funkcionalumo paleidimo sąlyga. Galimos reikšmės: o „onClick“ – paspaudimo sąlyga o „...“
`<MyNodeData>`	Veiklos diagramos grafo viršūnių aprašymo žymė. Galimi šie atributai: • „Key“ – grafo viršūnės identifikacijos aprašymas. Galimos reikšmės: o „Start“ – pradžios identifikatorius o „End“ – pabaigos identifikatorius. o „Conditional“ – sąlygos identifikatorius. o „...“ – bet koks vartotojo nurodytas identifikatorius. • „Category“ – grafo viršūnės kategorijos aprašymas. • „Text“ – grafo viršūnės sąlygos ar funkcionalumo aprašymas.
`<MyLinkData>`	Veiklos diagramos grafo viršūnių sąryšio žymė. Galimi šie atributai: • „From“ – prieš tai buvusios būsenos identifikatorius. • „To“ – kitos būsenos identifikatorius. • „Text“ – nurodoma jeigu tai sąlygos būsenos sąryšiai. Galimos reikšmės: o „Yes“ – sąlygos būsenos tenkinimo sąryšis („then“). o „No“ – sąlygos būsenos neigimo sąryšis („else“).

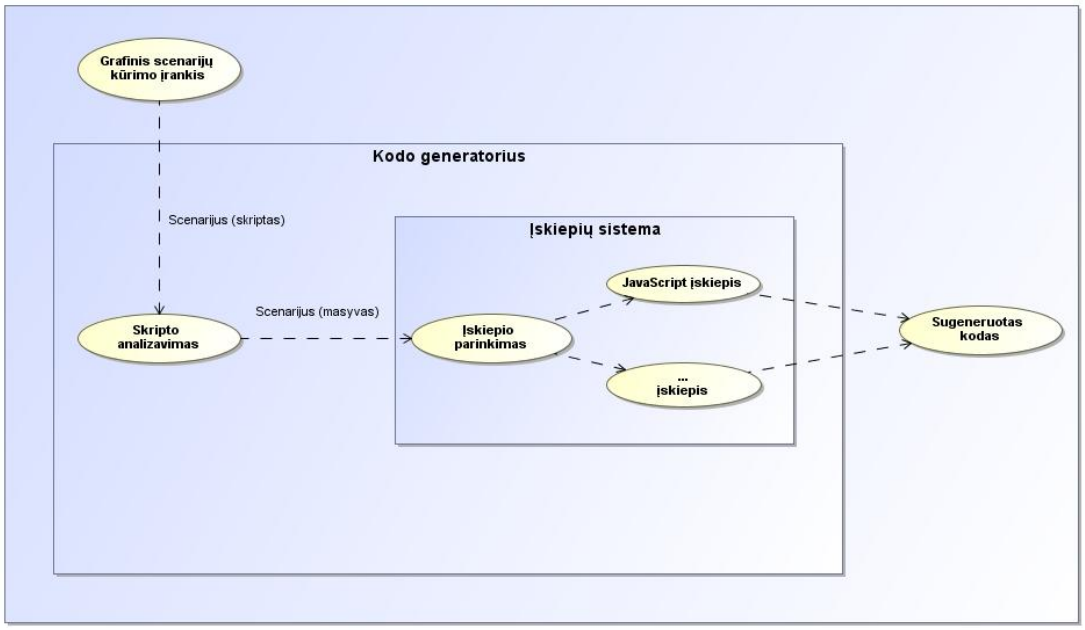
```
<?xml version="1.0" encoding="utf-8"?>
<scenario>
	<item type="object" name="search" class="Dialog">
		<param>DialogType.Modal</param>
		<param id="search" width="285" align="Align.Right" valign="VAlign.Bottom" />
	</item>
	<item type="function" name="search.AddControl">
		<param>
			<item type="object" class="Picture">
				<param  id="imgFind"  image="data/images/search_bw.png"  wheight="14"
left="2" top="2" onClick="function() {}" />
			</item>
		</param>
	</item>
	<item type="function" name="search.AddControl">
		<param>
			<item type="object" class="Label">
				<param id="lblQuery" text="Paieška:" width="50" left="2" top="3">
	<FlowChart event="onClick">
		<MyNodeData Key="Start" Category="Start" Text="Start" />
		<MyNodeData Key="key1" Category="Standard"  Text="alert('label onclick event')" />
		<MyNodeData Key="End" Category="End" Text="End" />
		<MyLinkData From="Start" To="key1" Text="Yes" />
		<MyLinkData From="key1" To="End" Text="Yes" />
	</FlowChart>
				</param>
			</item>
		</param>
	</item>
</scenario>
```

Pav. 12. Scenarijaus skripto pavyzdys

3.2.4.4. Įskiepių sistema

Tam kad kuriamas įrankis būtų pritaikomas architektūriniu požiūriu skirtingoms taikymo sritims naudojama įskiepių sistema. Atskiras įskiepis skirtas konkrečiai taikymo sričiai, tai gali būti konkreti programavimo kalba arba konkretus naudojamas karkasas ar jų visuma[3].

Priklausomai ko reikalauja taikymo sritis įskiepis gali atlikti kodo generavimą, optimizavimą, kompiliavimą, perdavimą klientinei daliai, papildomos informacijos surinkimą ir t.t. Įskiepių sistemos vaizdas pateiktas paveiksle „Pav. 13“.



Pav. 13. Įskiepių sistema

3.2.4.5. Kodo generavimas

Kodo generavimas atliekamas pagal iš anksto aprašytus fragmentų šablonus. Atpažįstami kodo fragmentai pateikiami lentelėje „Lentelė Nr. 18“. Eksperimento metu buvo pasirinkti šie fragmentai. Praktiškai jų pakanka aprašyti vidutinio sudėtingumo sistemas.

Lentelė Nr. 18. Atpažystami kodo fragmentai

Fragmentas	Aprašymas
Priskyrimas	Reikšmės priskyrimas kintamajam.
Funkcijos deklaravimas	Funkcijos/metodo deklaravimas ir realizacija.
Funkcijos kvietimas	Funkcijos iškvietimas.
Objekto kūrimas	Naujo objekto kūrimas
Sąlyga	Sąlygos sakinų ir jų poveikio veiksmų aptikimas.
Ciklas	Ciklo sakinių ir jų veiksmų aptikimas.

3.2.4.6. Kodo optimizavimas

Priklausomai nuo taikymo srities galima naudoti įvairius programinio kodo optimizavimo metodus [7]. Kodo optimizavimas vykdomas įskiepio ribose. Todėl optimizavimo technologijos gali būti naudojamos kiekvienam įskiepiui atskirai.

Eksperimentinėje sistemoje buvo panaudotas JavaScript supakavimo[28] technologiją, „Dean Edwards“ algoritmas. Ši technologija programinį kodą užkoduoja taip sumažinamas kodo apimtis. Visos interneto naršyklės geba šį kodą perskaityti ir interpretuoti. Veikimo greitis šio užkoduoto kodo yra panašus neužkoduotam. Panaudojus šią technologiją sumažinamas programinio kodo dydis neprarandant jo veikimo greičio. Toks kodas jau tampa žmogui neperskaitomas, tačiau jį galima iškoduoti ir lengvai perskaityti.

[28] Packer – supakavimas.

4. TYRIMO DALIS

4.1. Įvadas

Šiame skyriuje aprašysime įskiepiais paremto programinio kodo generatoriaus specifikacijos atitikimą ir kokybės analizę. Bus aptartos probleminės generatoriaus sritys ir pateikti programos tobulinimo pasiūlymai.

4.2. Kokybės analizė

4.2.1. Specifikacijos atitikimas

Sukurtas įskiepiais paremtas programinio kodo generatorius atlieka visus specifikacijoje aprašytus veiksmus. Tačiau iki galo neišspręsti sugeneruoto kodo optimizacijos klausimai. Gaunamas scenarijus (skriptas) iš grafinio scenarijų kūrimo posistemės laikomas teisingu, tačiau reikėtų papildomai jį patikrinti.

4.3. Iškilusios problemos

Tyrimo metu buvo tiriamas sukurtas projektavimo įrankis. Tyrimas vyko atliekant gyvenimiškus veiksmus, projektuojant įvairaus sudėtingumo sistemas. buvo susidurta su tam tikromis problemomis, bei pasiūlyti patobulinimai joms spręsti.

4.3.1. Kodo generavimas

4.3.1.1. Metakalbos sintaksė

Projektuojant bendravimo aspektus tarp grafinio modeliavimo įrankio ir kodo generavimo posistemės teko ilgai ieškoti universalaus sprendimo metakalbai. Tačiau tyrimo metu paaiškėjo kad pasirinktam sprendimui gali trūkti universalumo. Universalumo trūkumas gali pasireikšti projektuojant itin sudėtingus sprendimus, kada bus projektuojamos klasės ar sudėtingi hierarchiniai ryšiai [9].

4.3.1.2. Atpažystami kodo fragmentai

Scenarijaus analizės metu atpažystami įvairūs kodo fragmentai. Sudėtingų fragmentų atpažinimas tampa sudėtingas. Fragmentų kurie tarpusavyje susiję arba fragmentas fragmente (ciklo cikle sakiniai) aptikimas itin sudėtingas ir nebuvo numatytas projektuojant posistemę.

4.3.2. Testavimas

Testavimui buvo naudojamas vienetų testavimo[29] metodika. Buvo susidurta su tam skirtų įrankių problemomis, todėl buvo suprojektuotas ir realizuotas specifinis vienetų testavimo modulis kodo generavimo posistemei.

4.4. Siūlymai tobulinti programą

Atsižvelgiant į problemas, su kuriomis buvo susidurta tyrimo metu, buvo pasiūlyti patobulinimai.

4.4.1. Kodo generavimui

4.4.1.1. Metakalbai

Reikalinga suprojektuoti lankstesnes metakalbos taisykles. Esamos struktūros nepakaks jeigu bus naudojamos sudėtingesnės duomenų struktūros, modeliuojamos klasės ar jų paketai. Kuriant sudėtingą žiniatinklio aplikaciją klasių ar jų paketų modeliavimas būtinas.

4.4.1.2. Atpažystami kodo fragmentai

Dabartinis kodo generatorius geba atpažinti tik pagrindinius kodo fragmentus. Reikalinga išplėsti atpažįstamų kodo fragmentų kiekį. Reikėtų papildyti kad atpažintų duomenų struktūras, klases ir jų paketus.

4.4.1.3. Scenarijaus skripto analizavimas

Dabartinėje sistemoje įeinamas scenarijaus skriptas laikomas teisingu. Reikalinga tikrinti šio skripto korektiškumą. Šitaip bus išvengta nekorektiško modelio generavimo.

[29] Unit Test – vienetų testavimas.

4.5. Išvados

Realizavus ir ištyrus eksperimentinę sistemą rezultatas gavosi tokio kokio buvo tikėtasi ir užsibrėžti tikslai įgyvendinti. Siūlomi patobulinimai sistemą padarytų universalią ir patenkintų visus vartotojų poreikius. Tačiau siūlomų sprendimų įgyvendinimas gali privesti prie to kad kodo generavimo posistemę reikės perdaryti iš esmės.

5. EKSPERIMENTINĖ DALIS

5.1. Programavimo ir modeliavimo greičių eksperimentas

5.1.1. Užduotys

Atliekant tyrimą buvo sugalvotos trys skirtingo sunkumo užduotys, kurios leis nustatyti laiko skirtumo priklausomybę projektų sudėtingumui augant.

5.1.1.1. Pirmoji užduotis (mažas sudėtingumas)

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.

5.1.1.2. Antroji užduotis (vidutinis sudėtingumas)

Sumodeliuokite meniu komponentą, kuriame rodomi trys mygtukai. Nuspaudus pirmąjį mygtuką pereinama į „Pradinį“ langą. Nuspaudus antrąjį pereinama į „Paieška“ langą. Trečias mygtukas veikia priklausomai nuo prisijungimo būsenos. Jei vartotojas prisijungęs, jame rodomas tekstas „Išsiregistruoti“, nuspaudus jį vartotojas atsijungia nuo sistemos ir įjungiamas „Pradinis“ langas. Kitu atveju rodomas tekstas „Prisijungti“, o jį paspaudus atsidaro prisijungimo langas „Prisijungti“.

5.1.1.3. Trečioji užduotis (aukštas sudėtingumas)

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 „Vartotojo vardas“.*
- *Slaptažodžio žymės tekstą nurodyti „Slaptažodis“.*
- *Pateikimo mygtuko tekstą nurodyti „Prisijungti“. 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 „Slapta“.*

5.1.2. Nepatyrusio programuotojo eksperimento aprašymas

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, projektams sudėtingėjant.

5.1.2.1. Rezultatai

Išanalizavus surinktus duomenis buvo nustatyta, kad nepatyręs, neturintis žinių apie taikymo sritį, programuotojas mažo sudėtingumo užduotį programuojant įvykdė per 33,8 minutes, tuo tarpu modeliuojant jis užtruko 4,5 minutes. Taigi jis naują užduotį įgyvendino 7,51 kartus greičiau.

Vidutinio sudėtingumo užduotis programuojant buvo atlikta per 21 minutę. Modeliuojant grafiškai ši užduotis atlikta per 4,1 minutes. Pagreitėjimas šiuo atveju 5,12 karto.

Trečia užduotis, rašant programinį kodą, buvo įvykdyta per 37,4 minutes, o modeliuojant per 5,8 minutes. Pagreitėjimas – 6,45 karto.

Pagreitėjimo rezultatai atvaizduoti lentelėje „Lentelė Nr. 19“.

Lentelė Nr. 19. Nepatyrusio programuotojo eksperimento rezultatai

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

5.1.2.2. Išvados

Atlikus eksperimentą galime pastebėti, kad didžiausias laiko skirtumas modeliuojant ir programuojant matomas vykdant lengviausią užduotį, vėliau skirtumas sumažėjo, bet vėl 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 didėja kylant uždavinio sudėtingumui.

5.1.3. Patyrusio programuotojo eksperimento aprašymas

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.

5.1.3.1. Rezultatai

Patyrusio programuotojo rezultatai pateikiami lentelėje „Lentelė Nr. 20“.

Lentelė Nr. 20. Patyrusio programuotojo eksperimento rezultatai

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

5.1.3.2. Išvados

Rezultatai parodo, kad patyręs programuotojas, realizuodamas nesudėtingą užduotį, ją įgyvendino greičiau nei modeliuojant, tačiau kai užduočių sudėtingumas išaugo modeliavimo pagreitėjimas sparčiai didėjo. Todėl galime pasakyti, kad sudėtingų sistemų modeliavimas yra efektyvesnis už programavimą.

5.2. Kodo optimizavimo eksperimentas

5.2.1. Aprašymas

Kodo generatoriuje atliekamo kodo optimizavimo eksperimentui vykdyti pasirinkome JavaScript grafinės vartotojo sąsajos karkaso įskiepį. Grafiniu scenarijų kūrimo įrankio pagalba buvo sumodeliuotas paieškos komponentas. Gautas sugeneruotas programinis kodas (neoptimizuotas ir optimizuotas) ir palygintas su kitais JavaScript programinio kodo optimizavimo algoritmais.

Eksperimente papildomai panaudoti šie optimizavimo algoritmai:

- „YUI Compressor 2.4.2“ (http://www.refresh-sf.com/yui/).
- „iWeb Tools Online“ (http://www.iwebtoolsonline.com/javascript-optimizer).
- „xtreeme“ (http://www.xtreeme.com/javascript-optimizer/).
- „Javascript Optimizer“ (http://www.javascriptoptimizer.co.uk/).

- „Dean Edwards packer“ (integruotas į posistemę).

Eksperimente fiksuojamos programinio kodo dydžio, vykdymo laiko ir procesoriaus aprovimo vykdymo metu metrikos.

Eksperimento metu naudojamo interneto naršyklės „Google Chrome 5.0“, „Mozilla Firefox 3.6“, „Internet explorer 8.0“.

5.2.2. Rezultatai

Rezultatai naudojant „Chrome“ interneto naršyklę pateikti lentelėje „Lentelė Nr. 21“.

Rezultatai naudojant „Firefox“ interneto naršyklę pateikti lentelėje „Lentelė Nr. 22“.

Rezultatai naudojant „IE 8“ interneto naršyklę pateikti lentelėje „Lentelė Nr. 23“.

Lentelė Nr. 21. Kodo optimalumo eksperimento rezultatai naudojant „Chrome“

Algoritmas	Programinio kodo dydis (KB)*	Vykdymo laikas (ms)**	Procesoriaus apkrovimas (%)***
Neoptimizuotas	92,96 (0%)	6570 (0%)	58,0 (0%)
„YUI Compressor 2.4.2“	82,71 (12,0%)	6555 (0,2%)	59,0 (-1,7%)
„iWeb Tools Online“	83,00 (11,4%)	6610 (-0,6%)	58,0 (0,0%)
„xtreeme“	86,62 (6,6%)	6645 (-1,1%)	59,0 (-1,7%)
„Javascript Optimizer“	82.32 (10,7%)	6635 (-1,0%)	58,0 (0,0%)
„Dean Edwards packer“	33,94 (63,5%)	6860 (-4,4%)	59,5 (2,5%)

Lentelė Nr. 22. Kodo optimalumo eksperimento rezultatai naudojant „Firefox“

Algoritmas	Programinio kodo dydis (KB)*	Vykdymo laikas (ms)**	Procesoriaus apkrovimas (%)***
Neoptimizuotas	92,96 (0%)	21000 (0%)	59,0 (0%)
„YUI Compressor 2.4.2“	82,71 (12,0%)	21300 (-1,4%)	60,0 (-1,7%)
„iWeb Tools Online“	83,00 (11,4%)	21700 (-3,3%)	59,0 (0,0%)
„xtreeme“	86,62 (6,6%)	21550 (-2,6%)	59,0 (0,0%)
„Javascript Optimizer“	82.32 (10,7%)	21600 (-2,8%)	59,0 (0,0%)
„Dean Edwards packer“	33,94 (63,5%)	21750 (-3,5%)	61,0 (3,4%)

Lentelė Nr. 23. Kodo optimalumo eksperimento rezultatai naudojant „IE 8“

Algoritmas	Programinio kodo dydis (KB)*	Vykdymo laikas (ms)**	Procesoriaus apkrovimas (%)***
Neoptimizuotas	92,96 (0%)	6570 (0%)	60,0 (0%)
„YUI Compressor 2.4.2“	82,71 (12,0%)	6555 (0,2%)	61,0 (-1,7%)
„iWeb Tools Online“	83,00 (11,4%)	6610 (-0,6%)	61,0 (-1,7%)
„xtreeme“	86,62 (6,6%)	6645 (-1,1%)	61,0 (-1,7%)
„Javascript Optimizer“	82.32 (10,7%)	6635 (-1,0%)	60,0 (0,0%)
„Dean Edwards packer“	33,94 (63,5%)	6860 (-4,4%)	62,0 (3,3%)

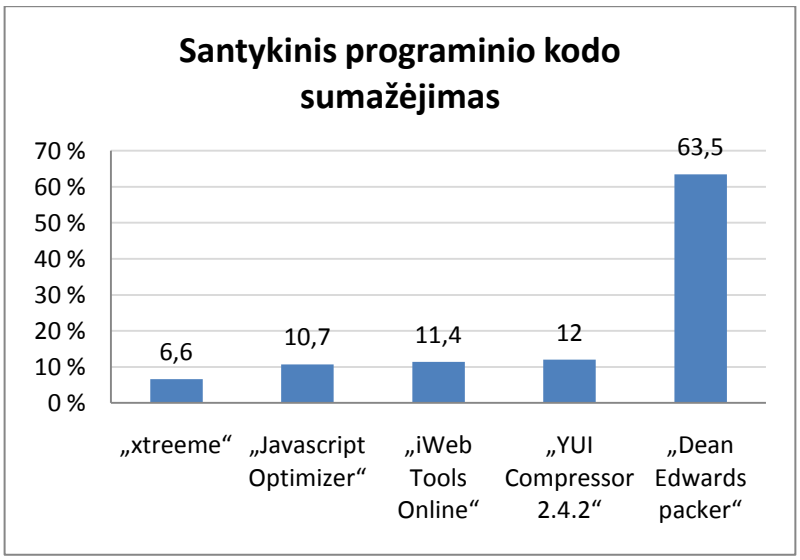
* – nurodomas santykinis programinio kodo sumažėjimas (procentais).

** – nurodomas santykinis vykdymo laiko sumažėjimas / padidėjimas (procentais).

*** – nurodomas santykinis procesoriaus apkrovimo padidėjimas / sumažėjimas (procentais).

Eksperimentas buvo pakartotas naudojant „Mozilla Firefox 3.6“ interneto naršyklę. Vykdymo laikas apie 3 kartus padidėjo, tačiau optimizuoto ir neoptimizuoto programinio kodo vykdymo laikas beveik identiškas.

Naudojant „Internet explorer 8.0“ vykdymo laikas padidėjo 4,5 karto lyginant su „Google Chrome“. Optimizuoto ir neoptimizuoto kodo vykdymo laikai taip pat beveik identiški.



Pav. 15. Santykinis programinio kodo sumažėjimas optimizuojant

5.2.3. Išvados

Grafinės vartotojo sąsajos įskiepio sugeneruoto kodo optimizacijos eksperimentas parodė, kad Dean Edwards supakavimo algoritmas sumažinio programinio kodo dydį net 63% ir aplenkė kitus algoritmus apie 50% (Pav. 15). Iš to seka kad supakavimo technologijos pakanka JavaScript kodui optimizuoti.

Eksperimentas parodė kad supakavimo technologija apdorotas programinis kodas gali veikti apie 4% lėčiau, tačiau į šį rodiklį galime nekreipti dėmesio. Nes sumažintas programinio kodo kiekis sąlygoja ženklų sistemos bandrą pagreitėjimą (pagreitinamas programinio kodo atsiuntimas į kliento kompiuterį). Naudojant kitą interneto naršyklę vykdymo greitis beveik identiškas neoptimizuoto kodo vykdymui.

Procesoriaus apkrovimas naudojant bet kokius optimizacijos algoritmus ar interneto naršykles išlieka labai panašus, todėl ši metrika jokios įtakos renkantis optimizacijos algoritmą neturi.

6. IŠVADOS

Šiame darbe buvo nagrinėjamas modelio transformavimas į programinį kodą, kuris atsakingas už kodo generavimą, nepriklausomą nuo architektūros, programavimo kalbos ar taikymo srities. Analizės metu nuspręsta naudoti įskiepiais paremtą architektūrą posistemei projektuoti. Taip pat nustatytos taisyklės papildomam abstrakcijos lygmeniui (metakalbai).

Įgyvendinus lanksčią architektūrą kodo generavimo posistemėje, pasiektas projektavimo įrankio lankstus pritaikomumas įvairioms taikymo sritims, ar net kelioms vienu metu.

Atlikus eksperimentus nustatėme, kad modeliuojant grafiškai, pasiektas iki septynių kartų greitesnis programų sistemos sukūrimo procesas, o modeliuojant sudėtingas sistemas galima pasiekti ir aukštesnių rezultatų.

Kito eksperimento metu nustatyta, kad generuojant žiniatinklio sistemų JavaScript programinį kodą, jo papildomai optimizuoti nebūtina, jeigu naudojama supakavimo technologija. Gauti rezultatai parodė, kad galima sumažinti sugeneruoto kodo apimtį iki 60%, neprarandant nei funkcionalumo nei greitaveikos.

7. LITERATŪRA

[1]. **A. Kleppe, J. Warmer, and W. Bast.** *MDA Explained: The Model Driven Architecture: Practice and Promise.* Boston: : Addison Wesley Professional, 2003. 192.

[2]. **Šarūnas Packevičius, Vilma Eidukynaitė, Andrej Ušaniov.** MDA CASE ĮRANKIŲ ANALIZĖ. [Tinkle] 2006 m. 01 20 d. [Cituota: 2010 m. 05 10 d.] https://mms.mruni.lt/DownloadFile.aspx?FileID=77.

[3]. **Lúcia Abrunhosa Fernandes, Beatriz Helena Neto, Vladimir Fagundes, Geraldo Zimbrão, Jano Moreira de Souza, Rodrigo Salvador.** IEEE Xplore. *Model-driven Architecture Approach for Data Warehouse.* [Tinkle] 2010 m. 03 13 d. [Cituota: 2010 m. 05 02 d.] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5442607.

[4]. *A Metalanguage Based on a Theory of Specification.* **Karl B. Zerangue, Joseph E. Urban.** 04 : s.n., 2004. A metalanguage based on a theory of specification - Computer Software and Applications Conference, 1992. COMPSAC '92. Proceedings., Sixteenth Annual I.

[5]. **Elisa Bertino, Barbara Catania.** Integrating XML and databases. *Internet Computing, IEEE.* [Tinkle] 2001 m. 08 02 d. [Cituota: 2010 m. 05 10 d.] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=939454.

[6]. *Building Digital Government by XML.* **Salminen, Airi.** Hawaii : s.n., 2005.

[7]. **Chengwan He, Fei He, Keqing He, Wenjie Tu.** IEEE Xplore. *Constructing Platform Independent Models of Web Application.* [Tinkle] 2005 m. 10 20 d. [Cituota: 2009 m. 02 10 d.] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1551133.

[8]. *Techniques for improving web application performance in bandwidth constrained scenarios.* **Ajay V Mahajan, Aravind Ajad Yarra.** TENCON : s.n., 2003.

[9]. *XML Rule Based Source Code Generator for UML CASE Tool.* **Dong Hyuk Park, So0 Dong Kim.** Seoul : s.n., 2004.

[10]. **Wilson, Kyle.** Data-Driven Design. *GameArchitect.net.* [Tinkle] 2002 m. 05 29 d. [Cituota: 2009 m. 02 03 d.] http://www.gamearchitect.net/Articles/DataDrivenDesign.html.

[11]. **Valentina Dagienė, Gintautas Grigas, Tatjana Jevsikova.** Enciklopedinis kompiuterijos žodynas. *Anglų–lietuvių kalbų kompiuterijos žodynėlis.* [Tinkle] 2009 m. 01 01 d. [Cituota: 2010 m. 05 20 d.] http://www.likit.lt/en-lt/angl.html.

8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

9. PRIEDAI

9.1. 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].

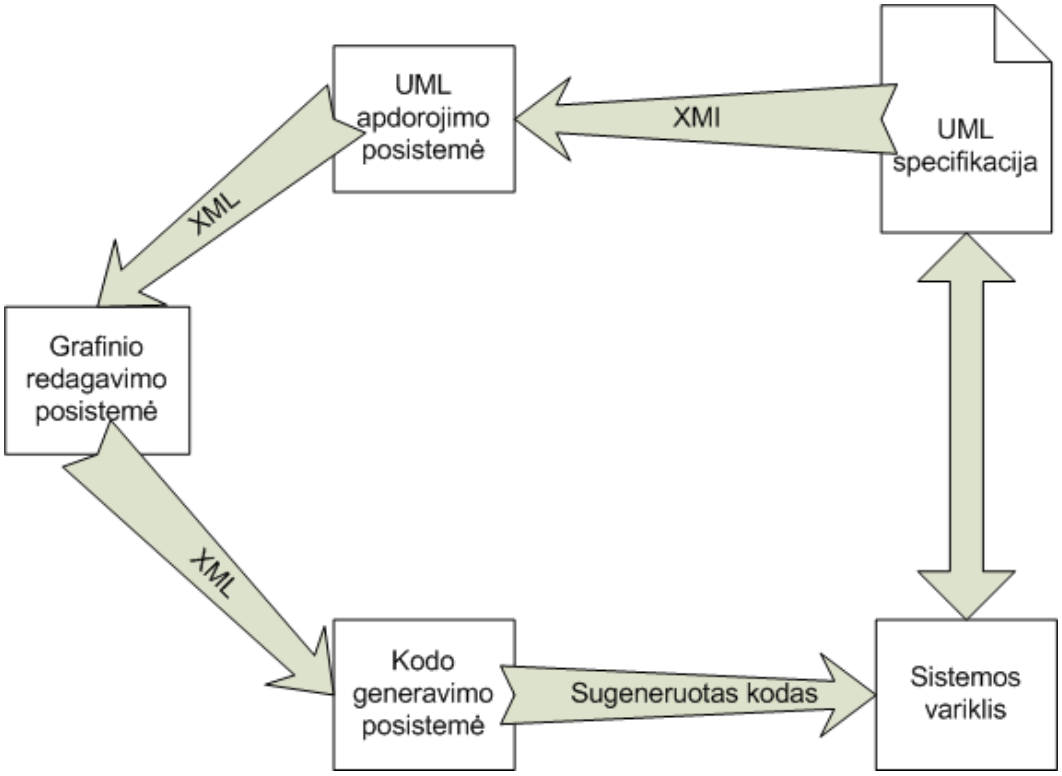
Tačiau ši programinė įranga labiau pritaikyta ne naudoti jau veikiančios sistemos galimybes, o kurti naujiems algoritmams.

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.



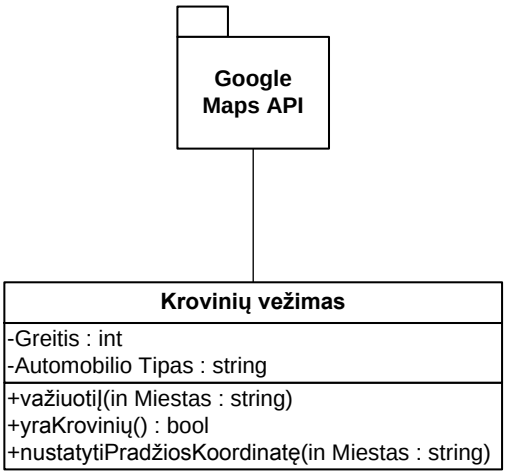
Pav. 16. Įrankio architektūros diagrama

4. Įrankio veikimo pavyzdys

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

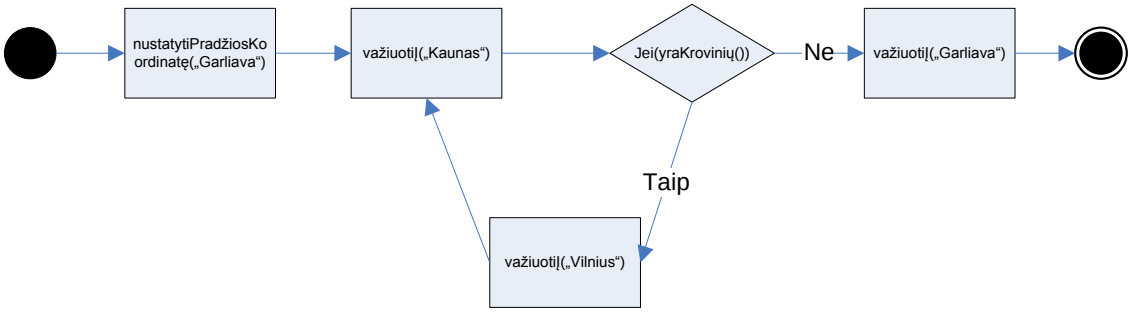
Vartotojui reikia sudaryti krovinio vežimo scenarijaus iš Kauno į Vilnių demonstraciją. Įmonės būstinė yra Garliavoje. Vairuotojas važiuoja į Kauną, kur tikrinama ar yra pervežimo užsakymų. Jeigu užsakymų yra, krovinys vežamas į Vilnių ir grįžtama atgal. Ši procedūra kartojama tol, kol kroviniai baigiasi.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

Sistemos UML specifikacijos pavyzdys:



Pav. 17. Pervežimų sistemos UML klasių diagrama

Grafinio redaktoriaus sumodeliuotas vaizdas:



Pav. 18. 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" />
		<false id="B2" />
	</function>
</Busena>

<Busena id="B4">
	<function name="vaziuotiI">
		<param value="Vilnius" />
	</function>
	<next id="B2" />
</Busena>

<Busena id="B5">
	<function name="vaziuotiI">
		<param value="Garliava" />
	</function>
	<next id="end" />
</Busena>
```

Pav. 19. Grafinio redaktoriaus XML išvestis

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");}

B1();
```

Pav. 20. Sugeneruotas programinis kodas

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>

9.2. Detali architektūra

Data Driven Design metodologija paremto grafinio scenarijų kūrimo įrankio architektūros specifikacija.

Dokumento paskirtis - pateikti kuriamos sistemos architektūrinį aprašymą. Jame aprašomi architektūros tikslai ir apribojimai, panaudojimo atvejai, sistemos statinis vaizdas bei dinaminė jos elgsena, sistemą sudarančios komponentės.

KAUNO TECHNOLOGIJOS UNIVERSITETAS
programų inžinerijos katedra

Grafinis scenarijų kūrimas naudojant Data-Driven-Design metodologiją

Architektūros specifikacija

Vadovas:
prof. Eduardas Bareiša

Užsakovas:
doc. dr. Tomas Blažauskas

Autoriai:
IFM-4/2 gr. stud.
Sigitas Povilaitis
Kęstutis Valinčius
Rytis Ūsalis
Paulius Paškevičius

KAUNAS, 2009

1. Įvadas

1.1. Dokumento paskirtis

Dokumento paskirtis - pateikti kuriamos sistemos architektūrinį aprašymą. Jame aprašomi architektūros tikslai ir apribojimai, panaudojimo atvejai, sistemos statinis vaizdas bei dinaminė jos elgsena, sistemą sudarančios komponentės.

1.2. Apibrėžimai ir sutrumpinimai

PĮ – programinė įranga
UML – standartizuota modeliavimo kalba (Unified Modeling Language)
DDD – duomenimis paremta architektūra (Data Driven Design)
XML – išplėsta žymėjimo kalba (Extensible Markup Language)

1.3. Apžvalga

Dokumentą sudaro šie skyriai:

Architektūros pateikimas: šiame skyriuje aprašoma kokia programinės įrangos (PĮ) architektūros pateikimo forma bus naudojama.

- Architektūros tikslai ir apribojimai – pateikiami kuriamos sistemos įgyvendinimo siekiai bei reikalavimai jai, turintys esminius architektūrai būdingus bruožus.
- Panaudojimo atvejų vaizdas – pateikiami sistemos panaudos atvejai.
- Sistemos statinis vaizdas – Nurodomos sistemą sudarančios komponentės (posistemės). Aprašoma posistemėse sąveikaujančios klasės.
- Sistemos dinaminis vaizdas – aprašoma kaip kinta sistemos būsenos, kaip sąveikauja objektai.
- Išdėstymo vaizdas – pateikiamas kuriamos sistemos fizinis išdėstymas techninėje įrangoje.
- Duomenų vaizdas – aprašomi sistemos naudojami duomenų tipai bei struktūros.
- Kokybė - aprašomi kriterijai, kuriais bus sprendžiama apie sistemos kokybę.

2. Architektūros pateikimas

Aprašymas, kaip pateikiama architektūra. Nurodoma, kokie yra reikalingi vaizdai (views) ir kiekvienam vaizdui nurodoma, kokie modeliavimo elementai jį sudaro.

Architektūros specifikacija pateikiama naudojant Rational Unified Process šabloną. Architektūros specifikacija bus sudaryta iš panaudojimo atvejų modelio, sistemos statinio vaizdo, sistemos dinaminio vaizdo, išdėstymo vaizdo ir duomenų vaizdo. Šie dalys dar bus skaidomos į smulkesnes.

Pagrindinių dalių paaiškinimai:

- Panaudojimo atvejų vaizdas - pateikiami esminiai panaudojimo atvejai
- Sistemos statinis vaizdas - pateikiamas sistemos išskaidymas į posistemes

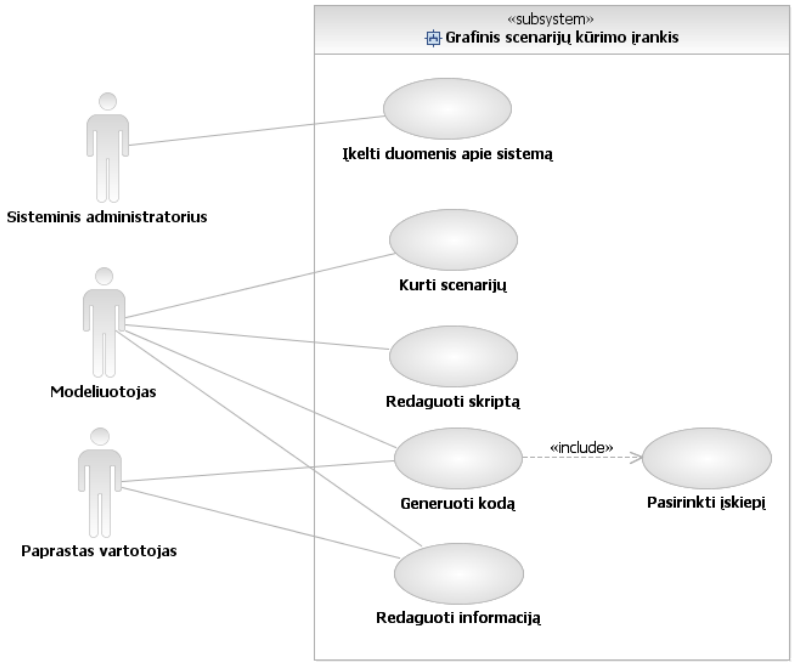
atvaizduojamos jų klasių diagramos
- Sistemos dinaminis vaizdas - Pateikiamos sąveikos, būsenų ir veiklos diagramos
- Išdėstymo vaizdas - Aprašoma techninės įrangos, kurioje sistema bus išdėstyta ir veiks, konfigūracija bei sistemos komponentai
- Duomenų vaizdas - Pateikiamas duomenų bazės modelis

3. 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.

4. Panaudojimo atvejų vaizdas



Pav. 21. Panaudos atvejų vaizdas

Lentelė Nr. 24. Panaudos atvejai

Panaudos atvejis	**Įkelti duomenis apie sistemą.**
Aprašas:	Tai pradinių duomenų įkėlimas į įrankį naudojant sistemos UML specifikaciją.
Vartotojas/Aktorius:	Sisteminis administratorius.
Prieš-sąlyga:	Sistemos specifikacija yra parengta.
Sužadinimo sąlyga:	Įrankio diegimas į sistemą.
Po-sąlyga:	Įrankiu galima pradėti naudotis.
Panaudos atvejis	**Kurti scenarijų.**
Aprašas:	Tai veiksmų, kuriuos atliks sistema, modeliavimas naudojant grafinę sąsają.
Vartotojas/Aktorius:	Modeliuotojas.
Prieš-sąlyga:	Nėra.
Sužadinimo sąlyga:	Kilo poreikis sistemoje įgyvendinti naują funkcionalumą.
Po-sąlyga:	Sistema gali naudotis scenarijuje sumodeliuota veikla.
Panaudos atvejis	**Redaguoti skriptą.**
Aprašas:	Galimybė įvesti scenarijaus pakeitimus, kuriuos yra sunkiau (neįmanoma) atlikti per grafinį įrankį.
Vartotojas/Aktorius:	Modeliuotojas.
Prieš-sąlyga:	Scenarijus yra sukurtas. Skriptas yra sugeneruotas.
Sužadinimo sąlyga:	Sistemoje reikalingas naujas funkcionalumas.
Po-sąlyga:	Įvesti skripto pakeitimai tenkina modeliuotojo lūkesčius.
Panaudos atvejis	**Generuoti kodą.**
Aprašas:	Generuoja skriptą, pagal sumodeliuotą scenarijų.
Vartotojas/Aktorius:	Modeliuotojas, vartotojas.
Prieš-sąlyga:	Scenarijus yra sukurtas.
Sužadinimo sąlyga:	Baigtas modeliuoti scenarijus.
Po-sąlyga:	Sugeneruotas kodas atitinka scenarijaus veiksmus.

Panaudos atvejis	**Pasirinkti įskiepį.**
Aprašas:	Parenką įskiepį atitinkantį taikymo srities architektūrą.
Vartotojas/Aktorius:	Modeliuotojas, vartotojas.
Ryšys su kitais PA:	Generuoti kodą.
Prieš-sąlyga:	Scenarijus yra sukurtas.
Sužadinimo sąlyga:	Baigtas modeliuoti scenarijus.
Po-sąlyga:	Sugeneruotas kodas atitinka taikymo srities architektūrą.
Panaudos atvejis	**Redaguoti informaciją.**
Aprašas:	Galimybė pakeisti sistemos scenarijaus modelį, neleidžiant pakeisti sistemos loginės veiklos.
Vartotojas/Aktorius:	Modeliuotojas, vartotojas.
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. Sistemos statinis vaizdas

5.1. Apžvalga

Sistema suskirstyta į šiuos paketus:

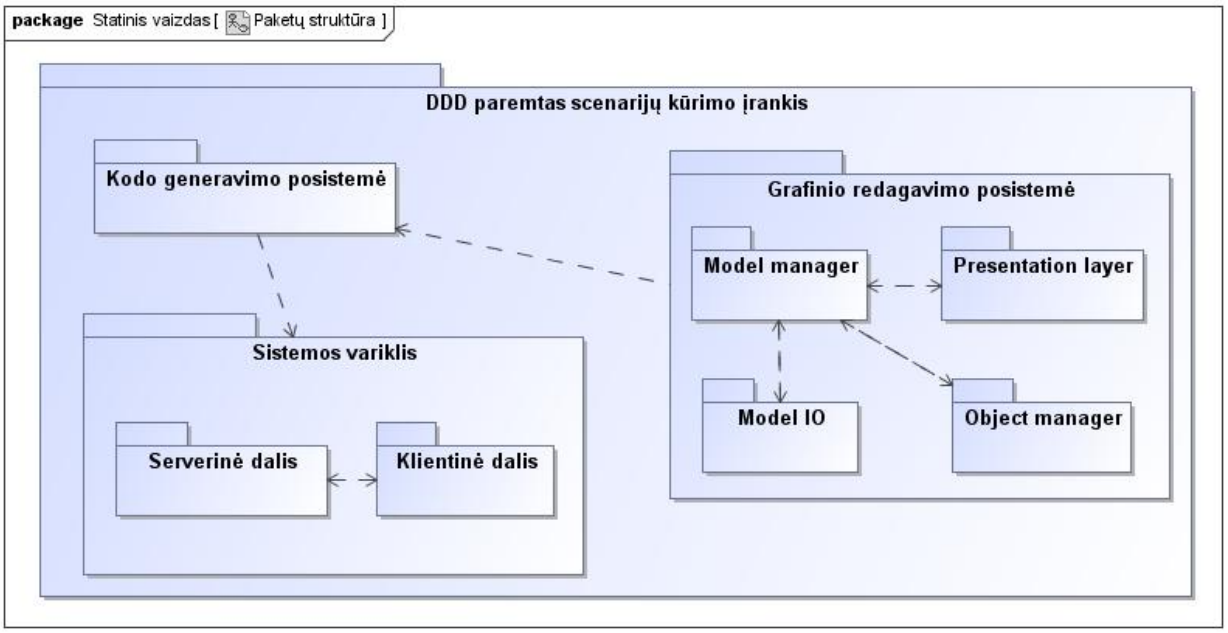
- Grafinio redagavimo posistemė
- Kodo generavimo posistemė
- Sistemos variklis

Grafinio redagavimo posistemė skirstoma į smulkesnius paketus

- Model manager
- Model IO
- Presentation layer
- Object manager

Sistemos variklis skirstomas į smulkesnius paketus:

- Serverinė dalis
- Klientinė dalis

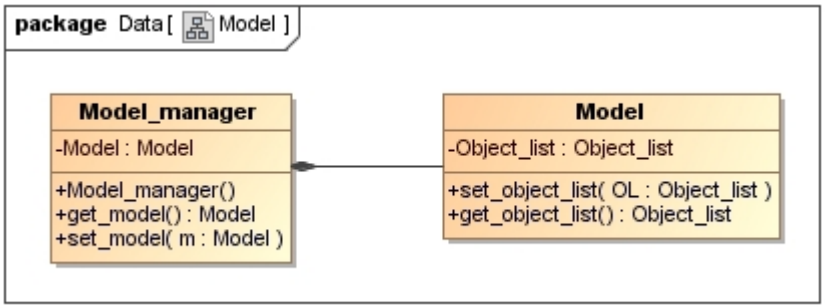


Pav. 22. Sistemos paketų diagrama

5.2. Paketų detalizavimas

5.2.1. Grafinio redagavimo posistemė.

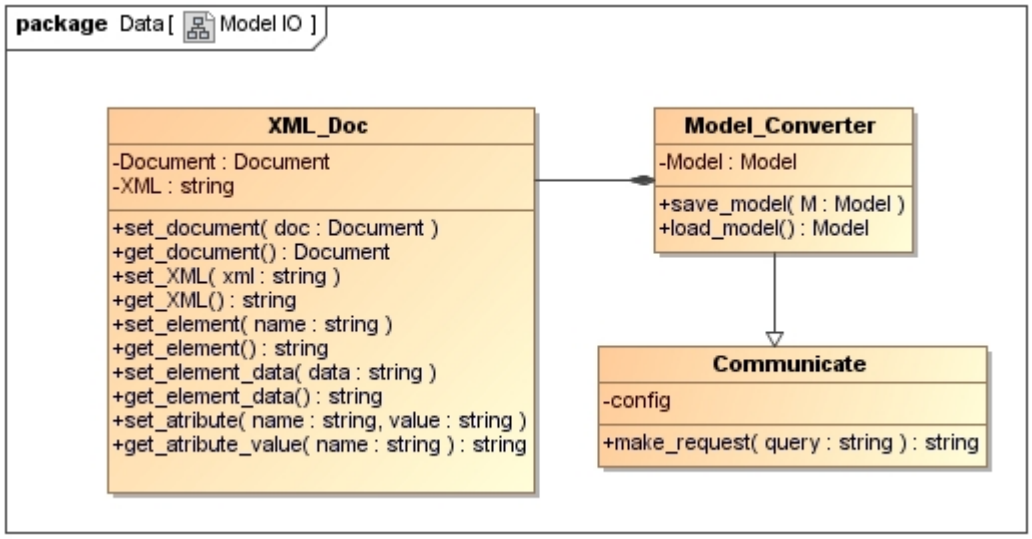
5.2.1.1. „Model manager“ paketas



Pav. 23. Model manager paketo klasių diagrama

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

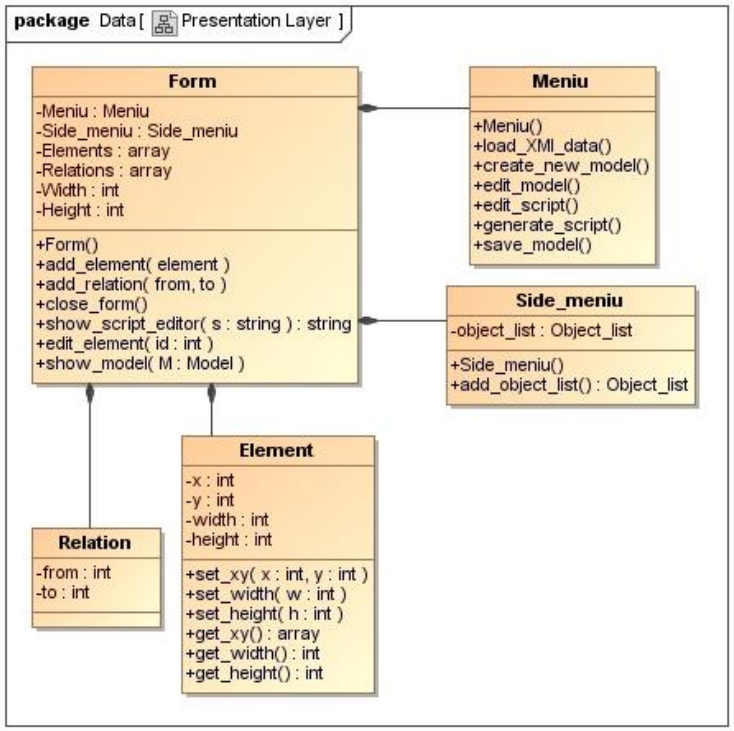
5.2.1.2. „Model IO“ paketas



Pav. 24. Model IO paketo klasių diagrama

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

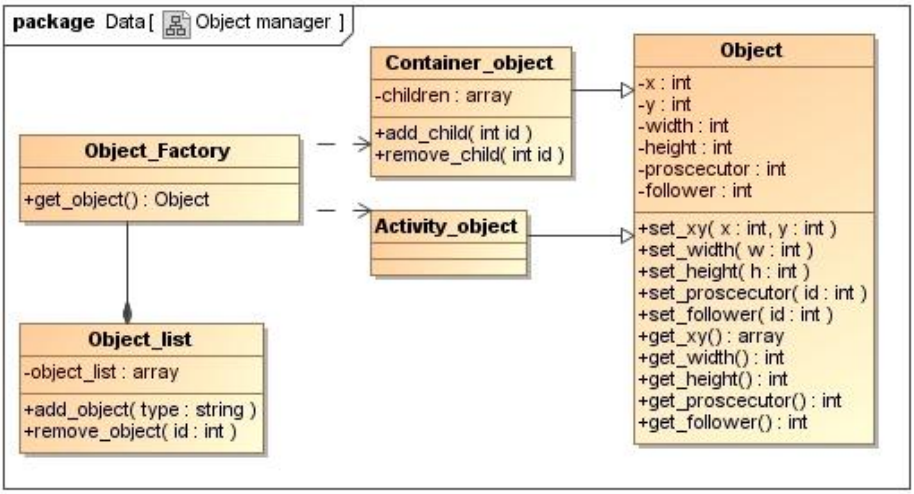
5.2.1.3. „Presentation layer“ paketas



Pav. 25. Presentation layer paketo klasių diagrama

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

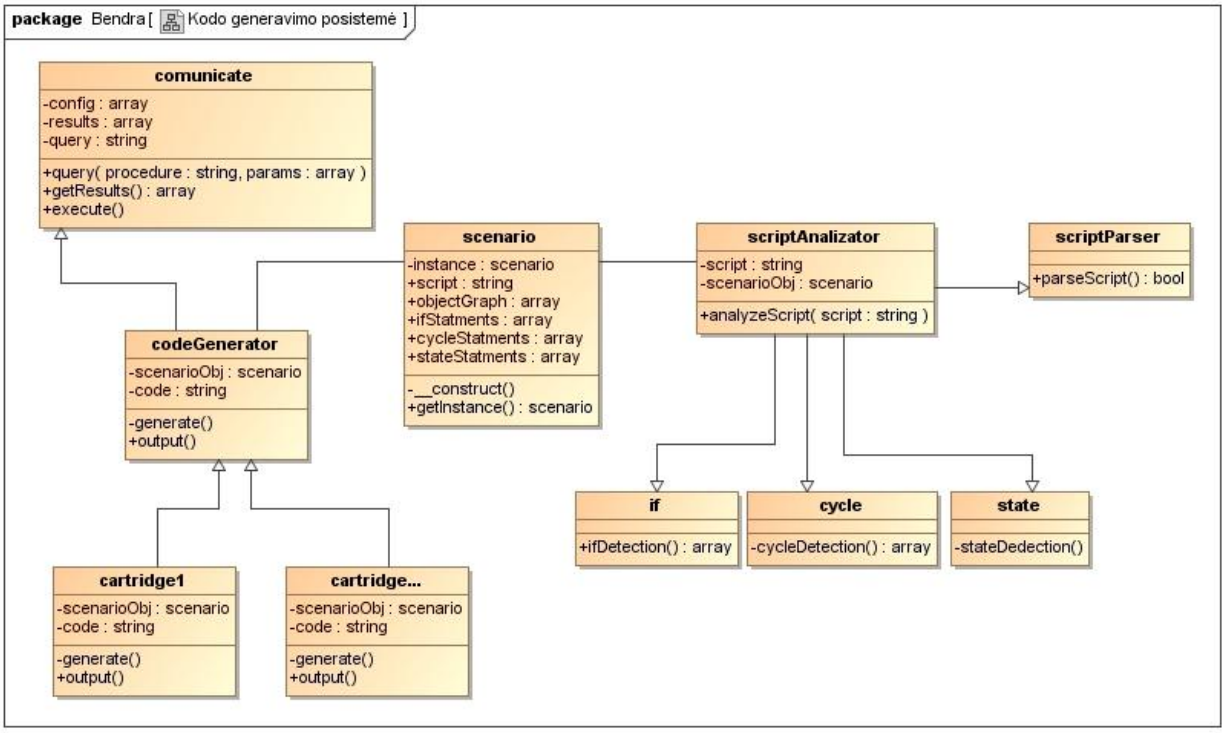
5.2.1.4. „Object manager“ paketas



Pav. 26. Object manager paketo klasių diagrama

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

5.2.1.5. Kodo generavimo posistemė

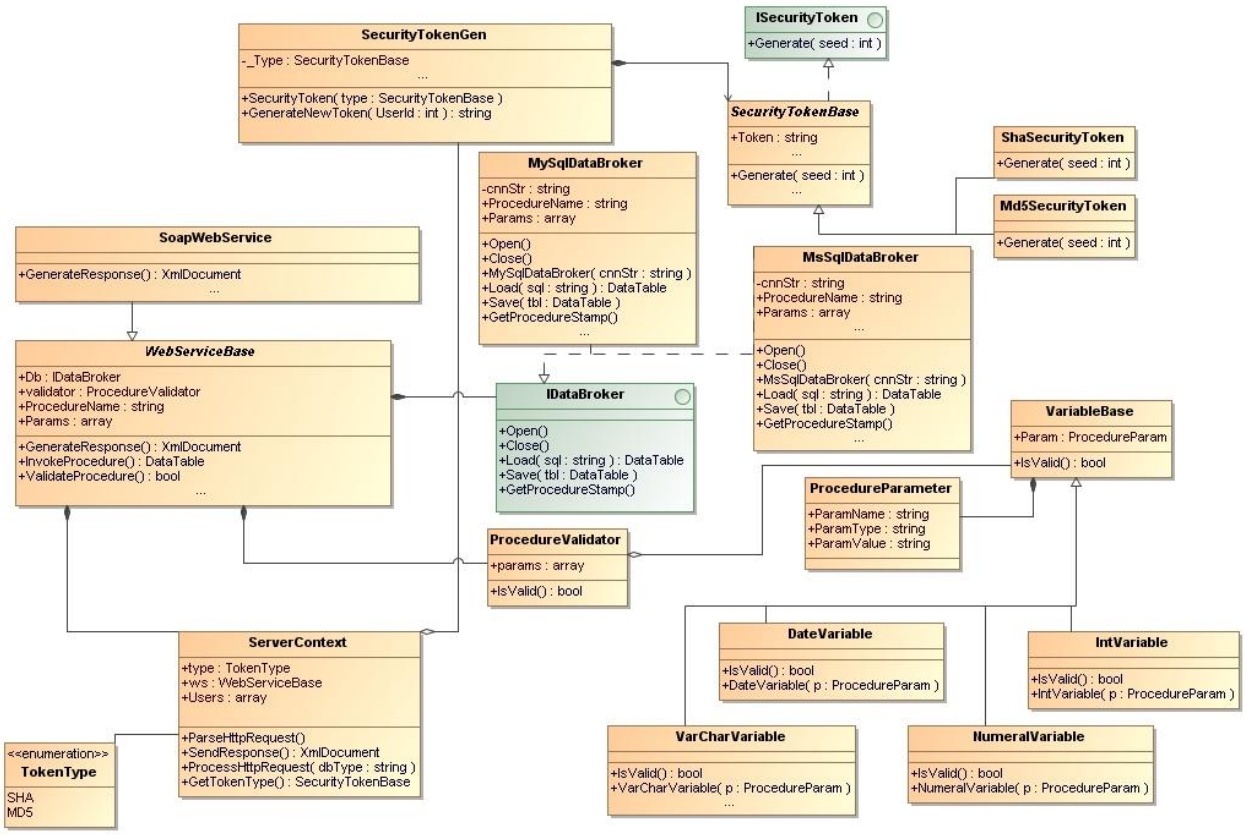


Pav. 27. Kodo generavimo posistemės klasių diagrama

Kodo generavimo posistemė – generuoja programinį kodą. Gautą skriptą iš grafinio redagavimo posistemės transformuoti į programinį kodą pagal pasirinktą taikymo sritį (įskiepį) .

5.2.2. Sistemos variklis

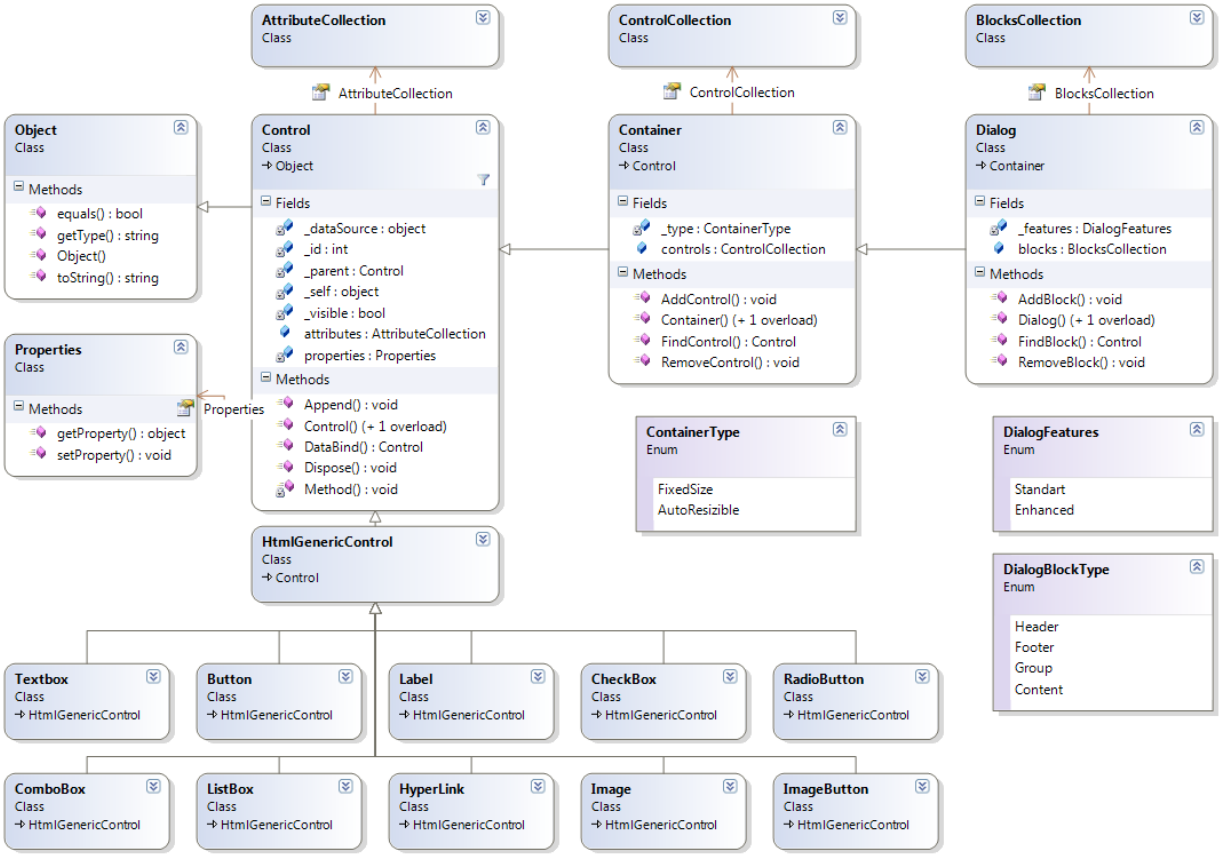
5.2.2.1. Serverinė dalis



Pav. 28. Sistemos variklio serverinės dalies klasių diagrama

Paketo paskirtis - vartotojo autentifikacijos užtikrinimas bei duomenų mainų sąsajų vartotojui užtikrinimas

5.2.2.2. Klientinė dalis



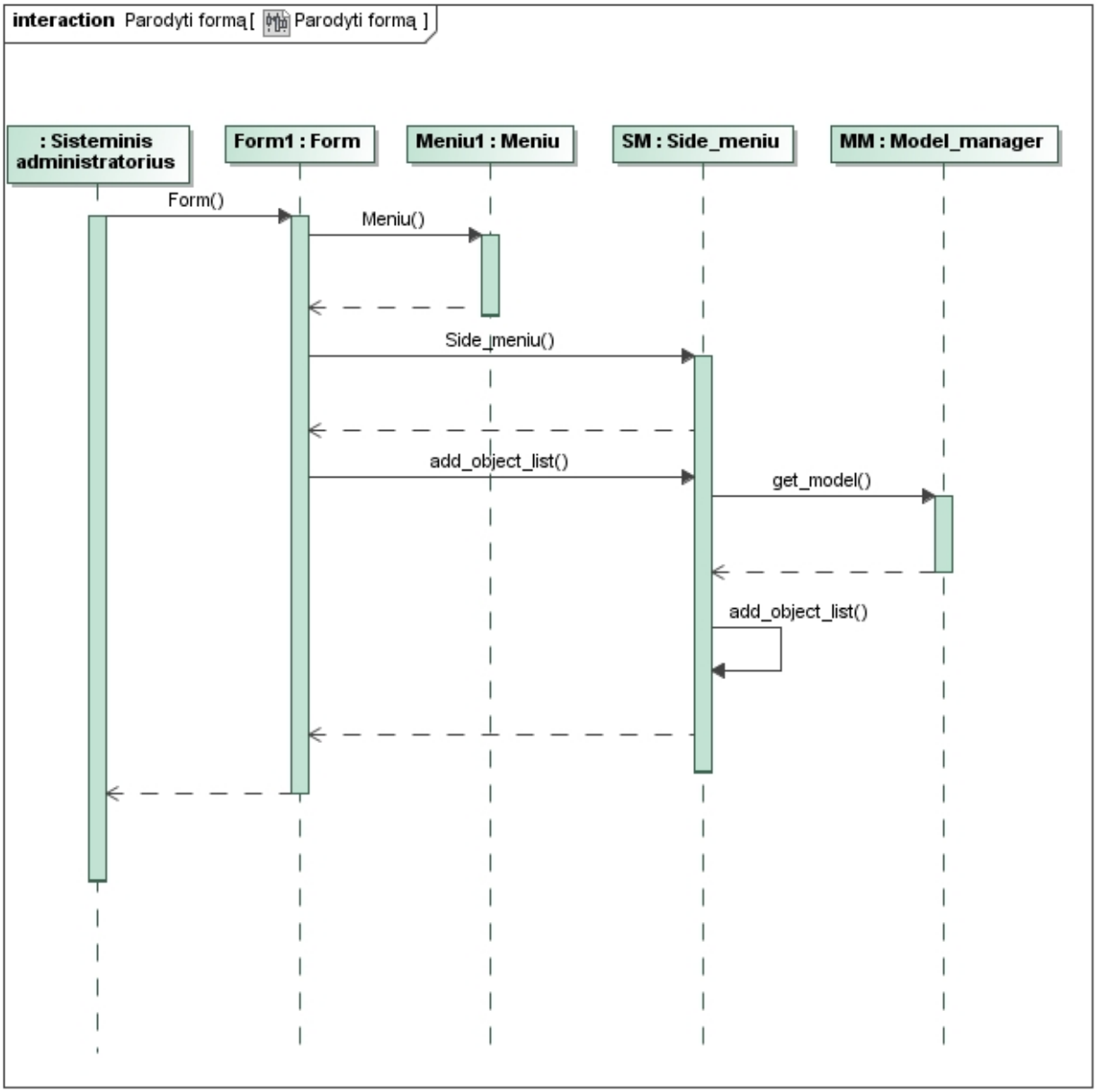
Pav. 29. Sistemos variklio klientinės dalies klasių diagrama

Ši sistemos branduolio dalis skirta prezentacijos sluoksniui (presentation layer) generuoti. Ji aprašo esminius žiniatinklio GUI komponentus. Jos pagalba galima projektuoti sudėtingus GUI komponentus (dialogus, vedlius, meniu ir kt.).

6. Sistemos dinaminis vaizdas

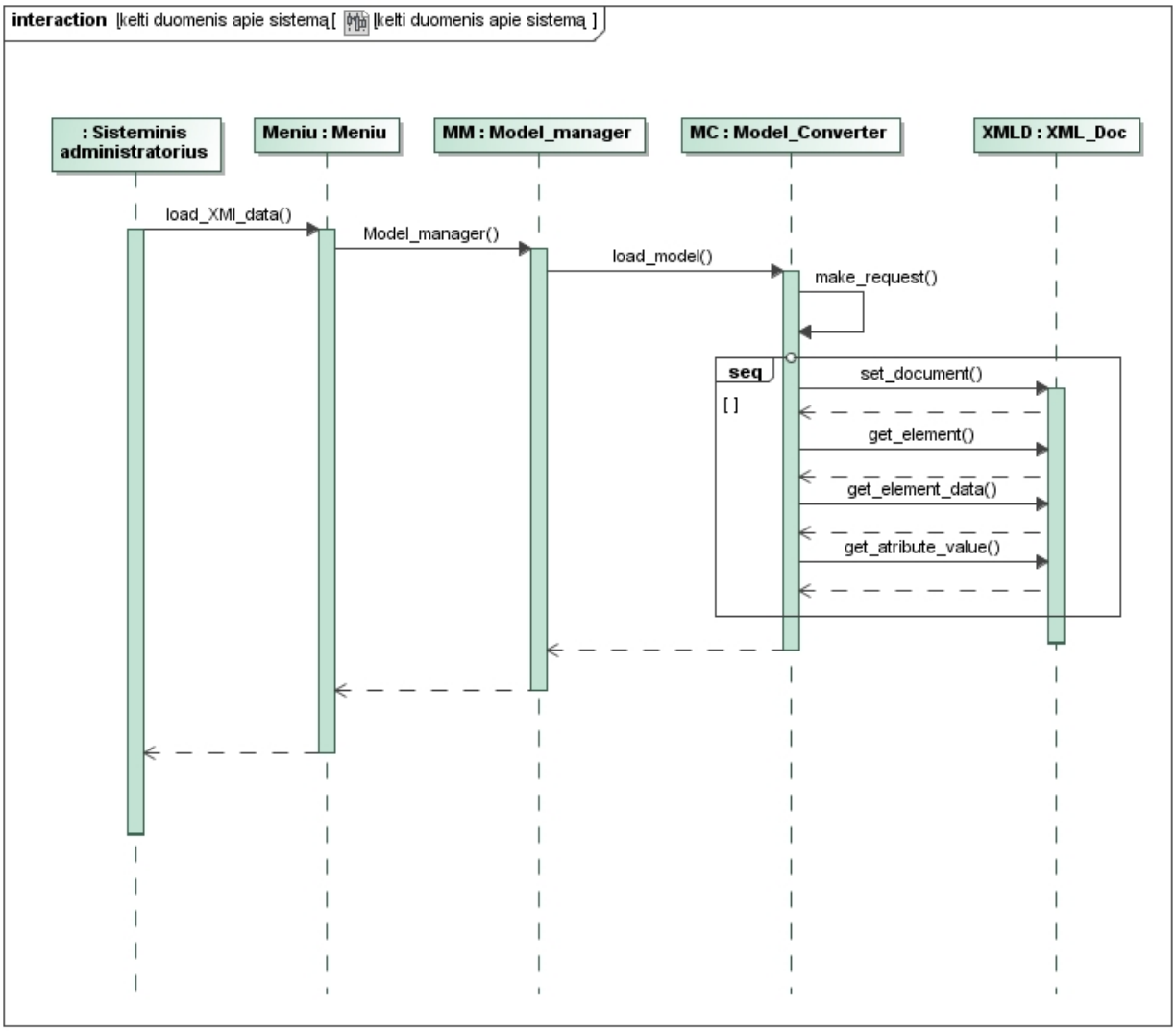
6.1.Sekų diagramos

6.1.1. Parodyti formą



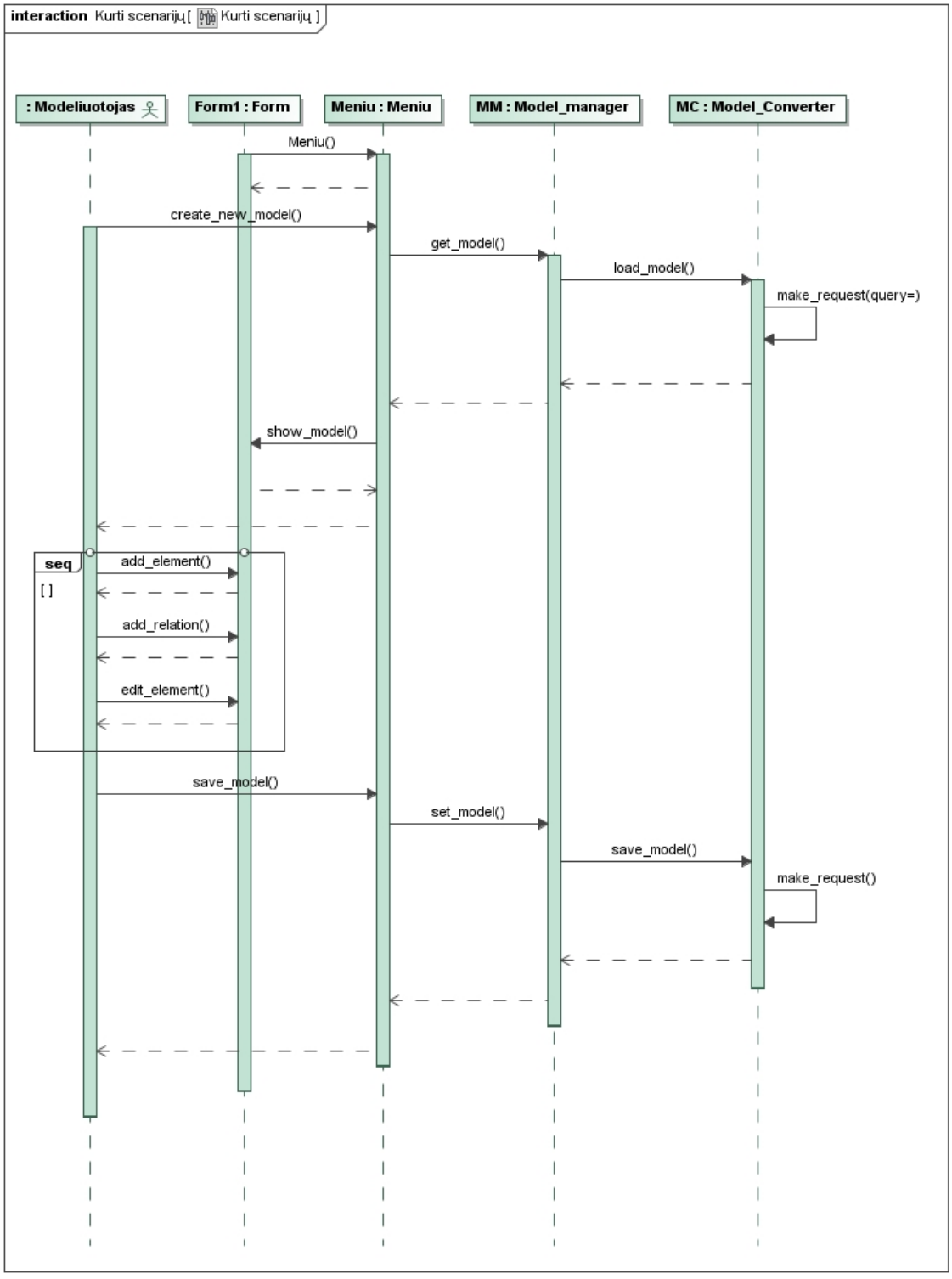
Pav. 30. Sekų diagrama „parodyti formą“

6.1.2. Įkelti duomenis apie sistemą



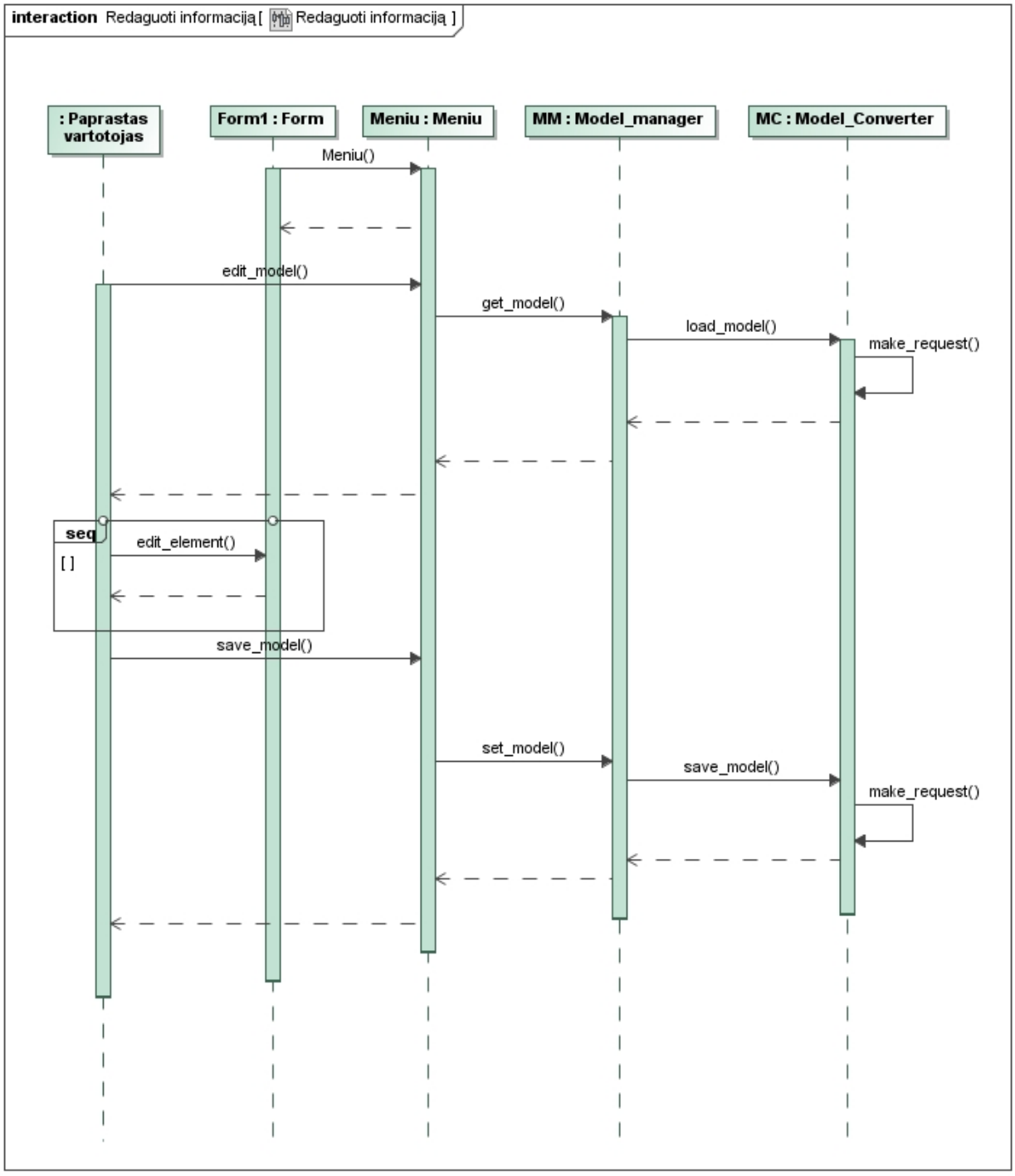
Pav. 31. Sekų diagrama „įkelti duomenis apie sistemą“

6.1.3. Kurti scenarijų



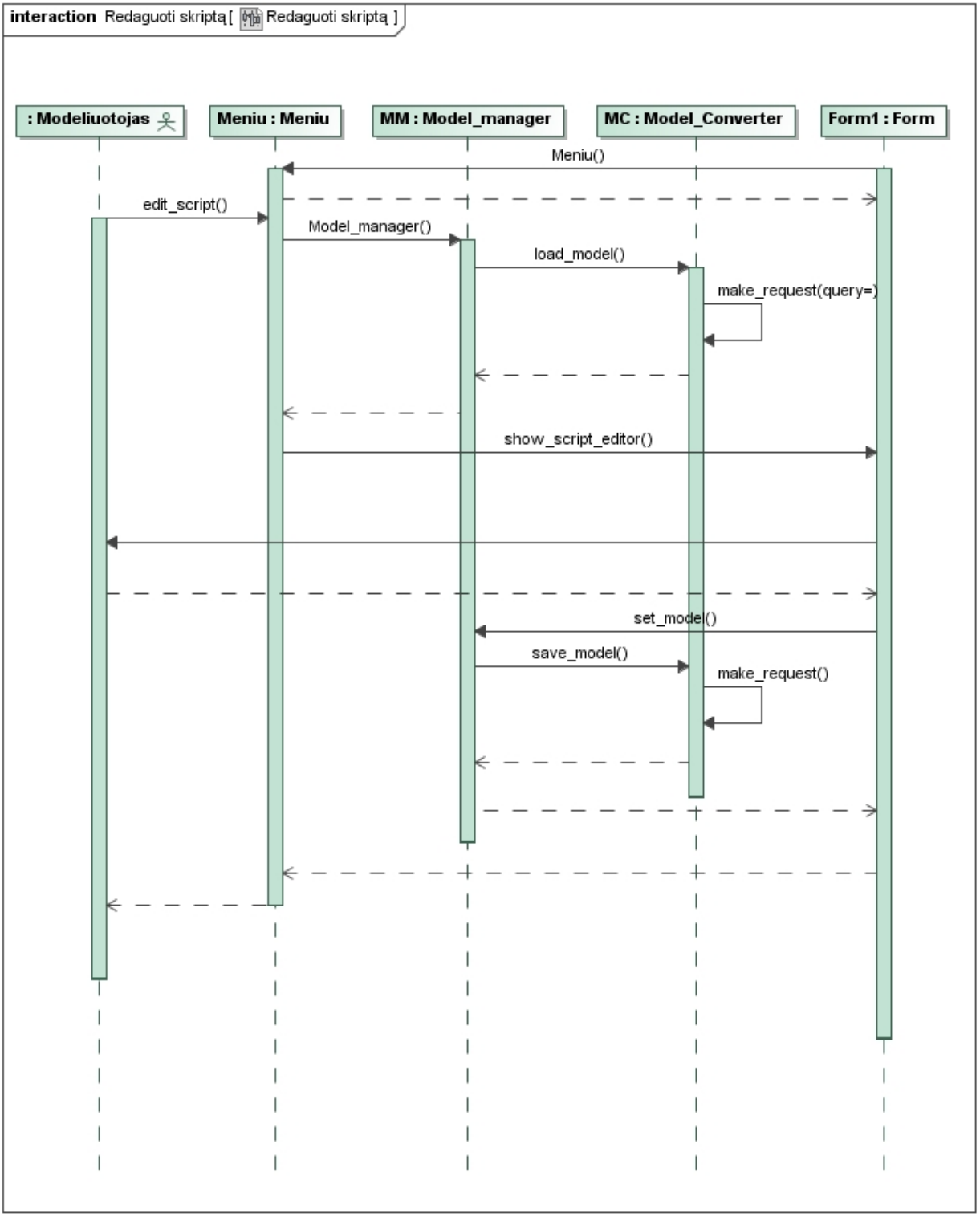
Pav. 32. Sekų diagrama „kurti scenarijų“

6.1.4. Redaguoti informaciją



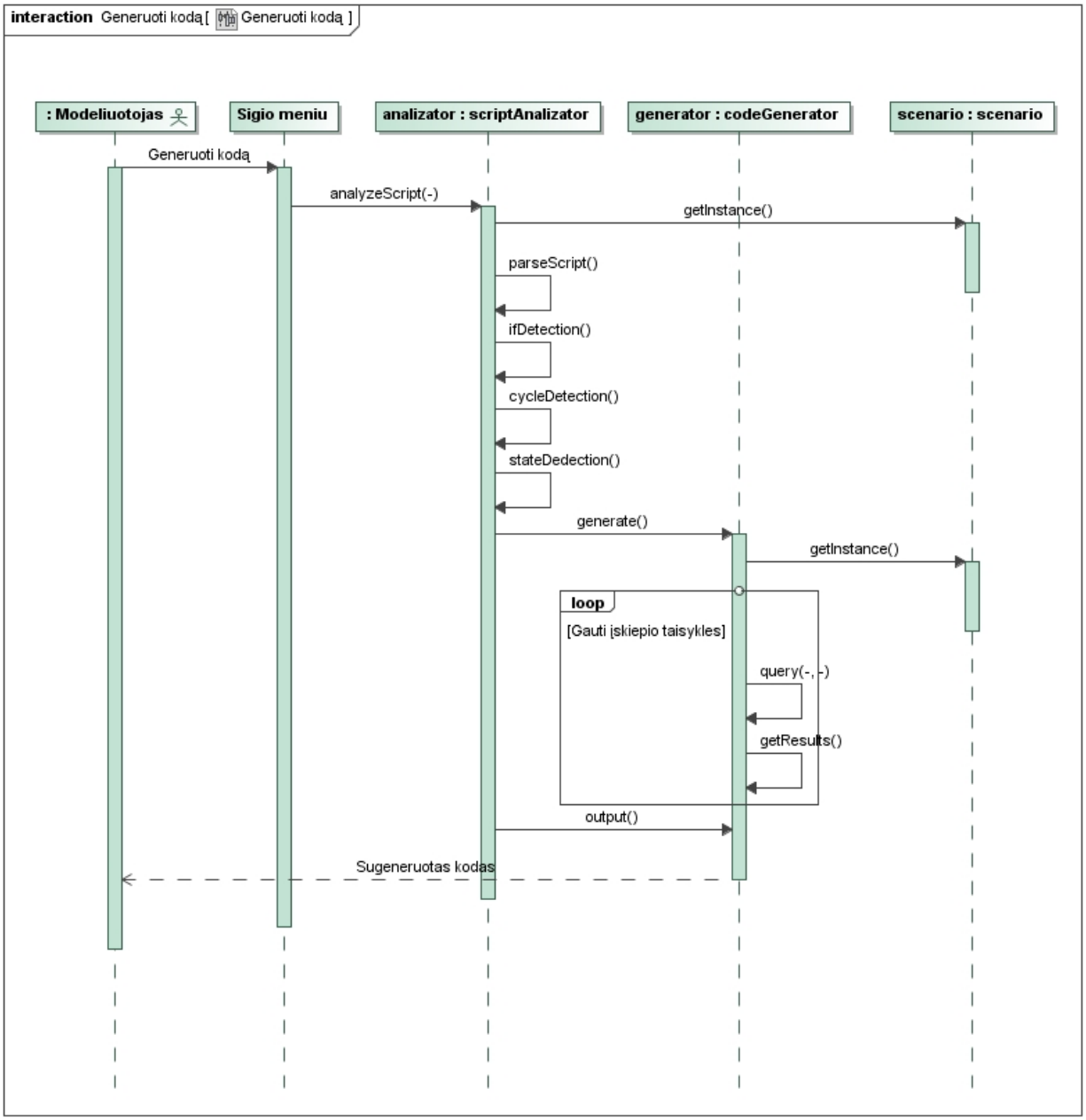
Pav. 33. Sekų diagrama „redaguoti informaciją“

6.1.5. Redaguoti skriptą



Pav. 34. Sekų diagrama „redaguoti skriptą“

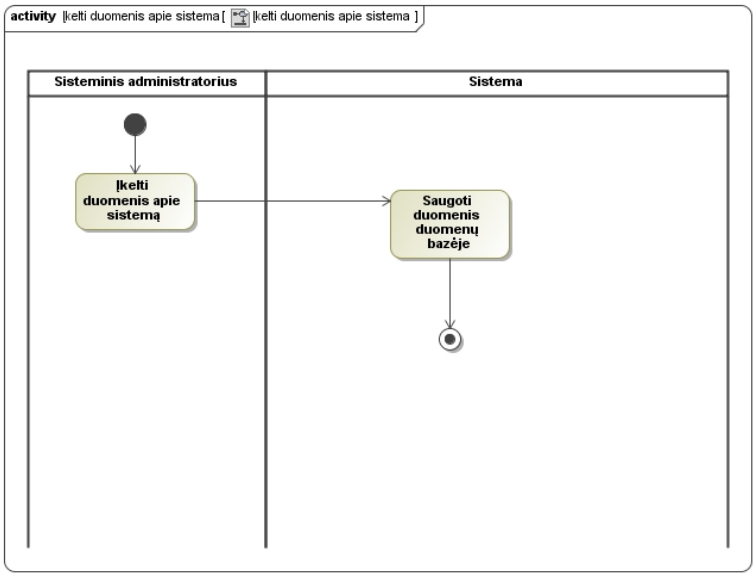
6.1.6. Generuoti kodą



Pav. 35. Sekų diagrama „generuoti kodą“

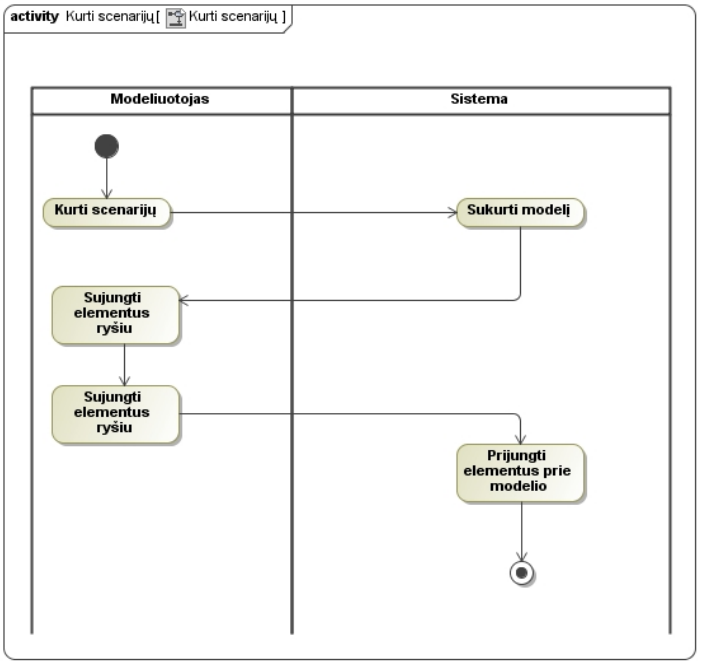
6.2. Veiklos diagramos

6.2.1. Įkelti duomenis apie sistemą



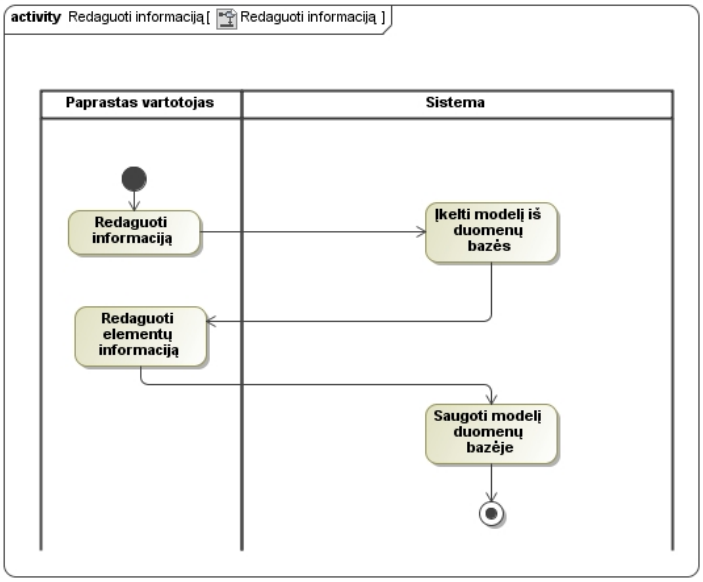
Pav. 36. Veiklos diagrama „įkelti duomenis apie sistemą“

6.2.2. Kurti scenarijų



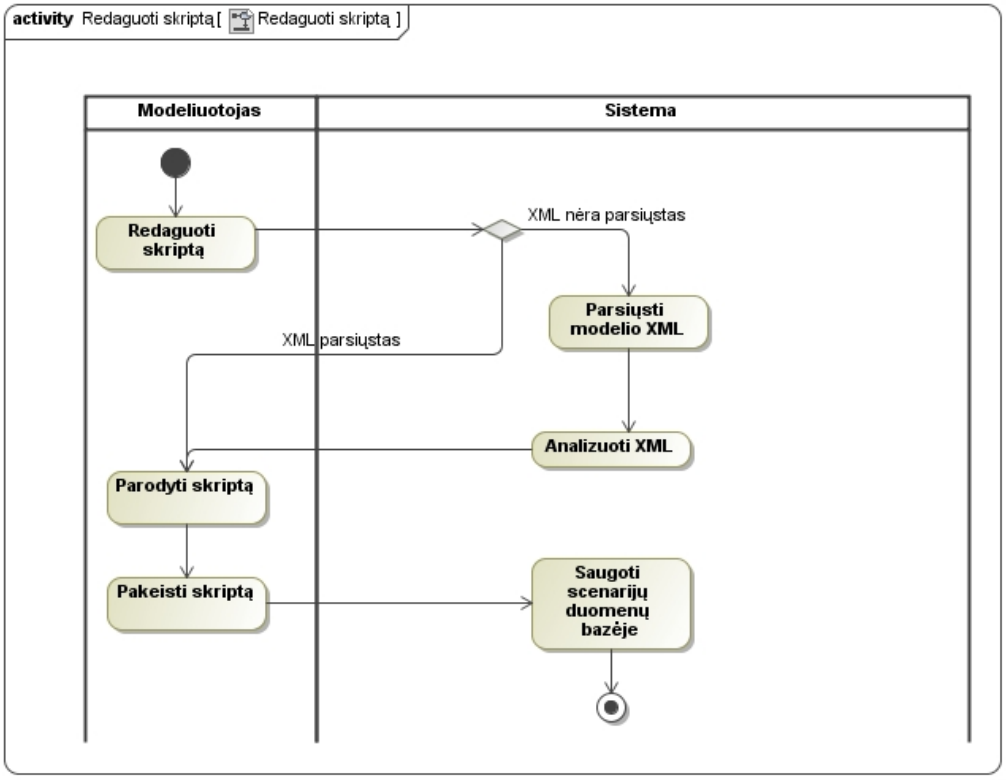
Pav. 37. Veiklos diagrama „kurti scenarijų“

6.2.3. Redaguoti informaciją



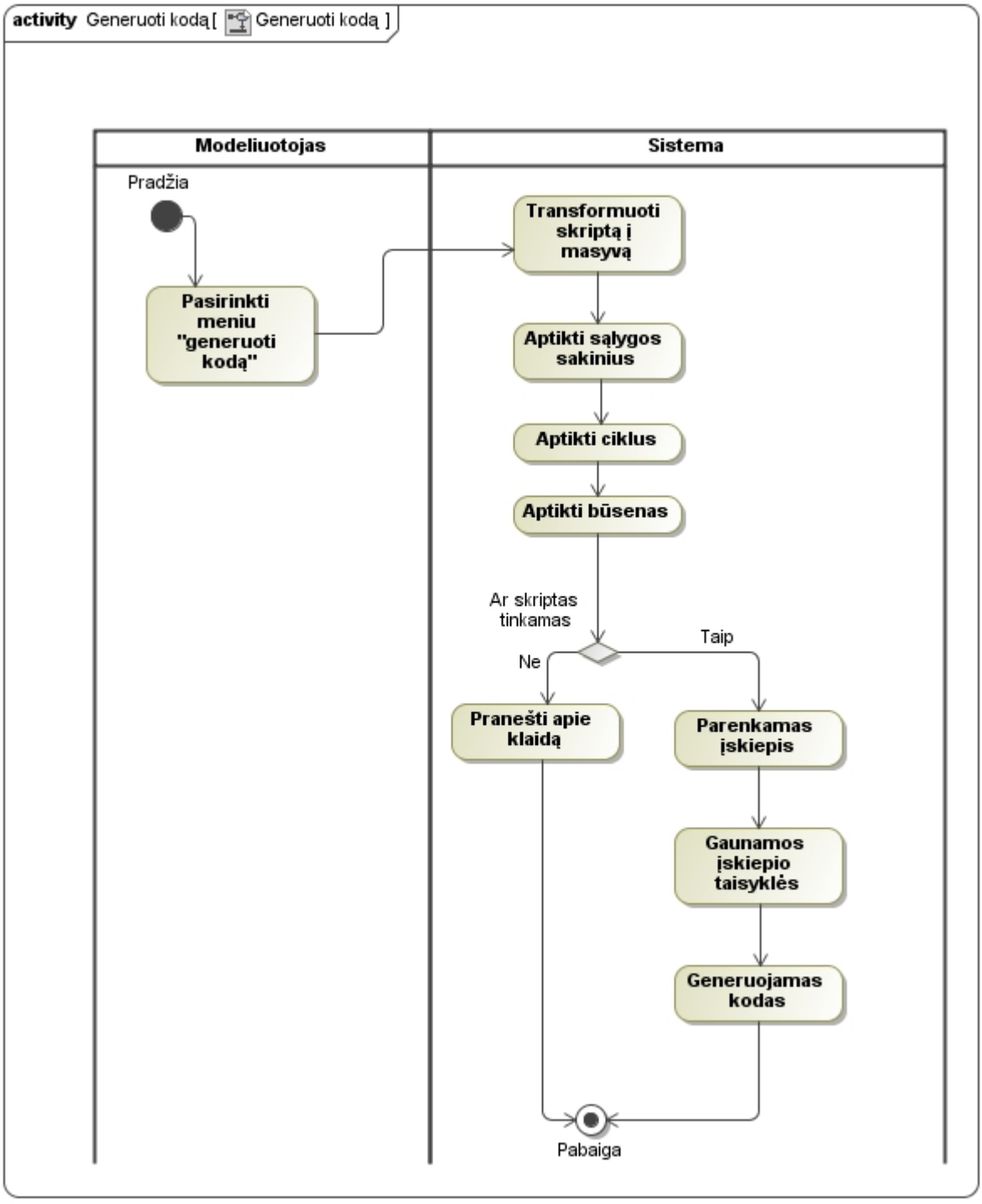
Pav. 38. Veiklos diagrama „redaguoti informaciją“

6.2.4. Redaguoti skriptą



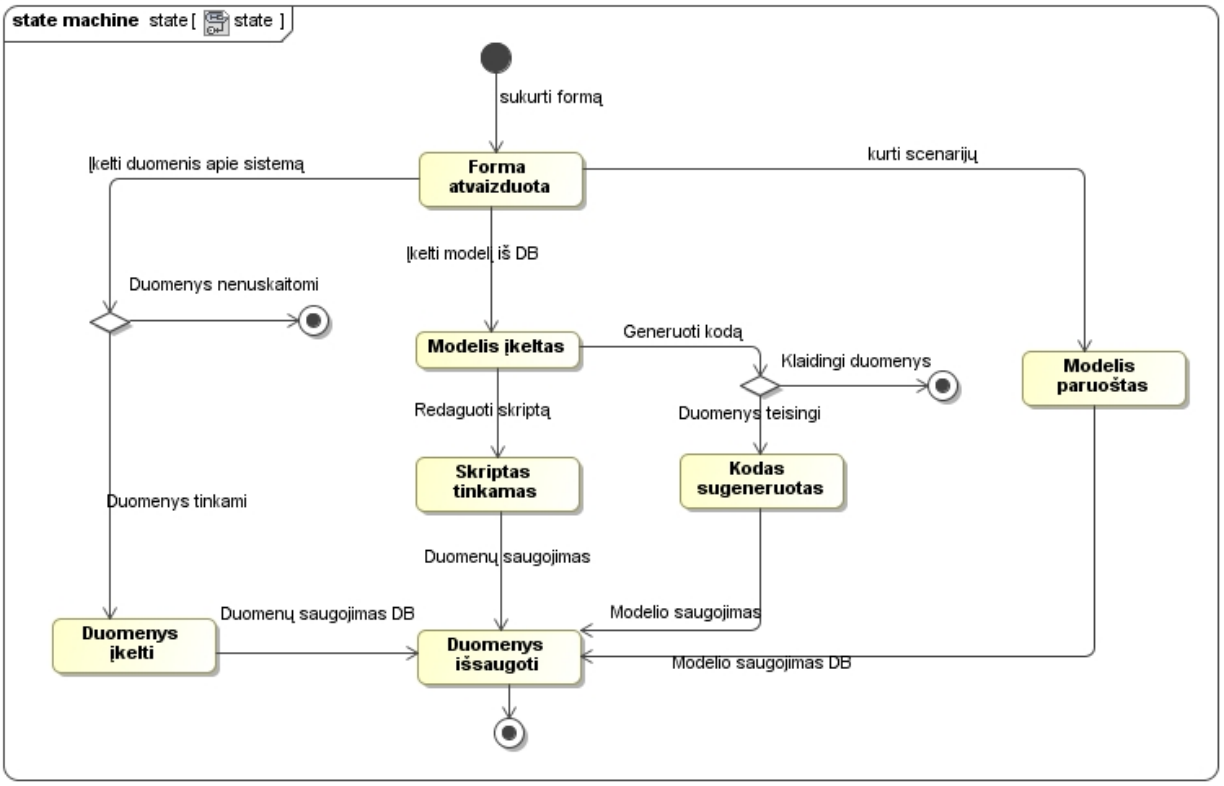
Pav. 39. Veiklos diagrama „redaguoti skriptą“

6.2.5. Generuoti kodą



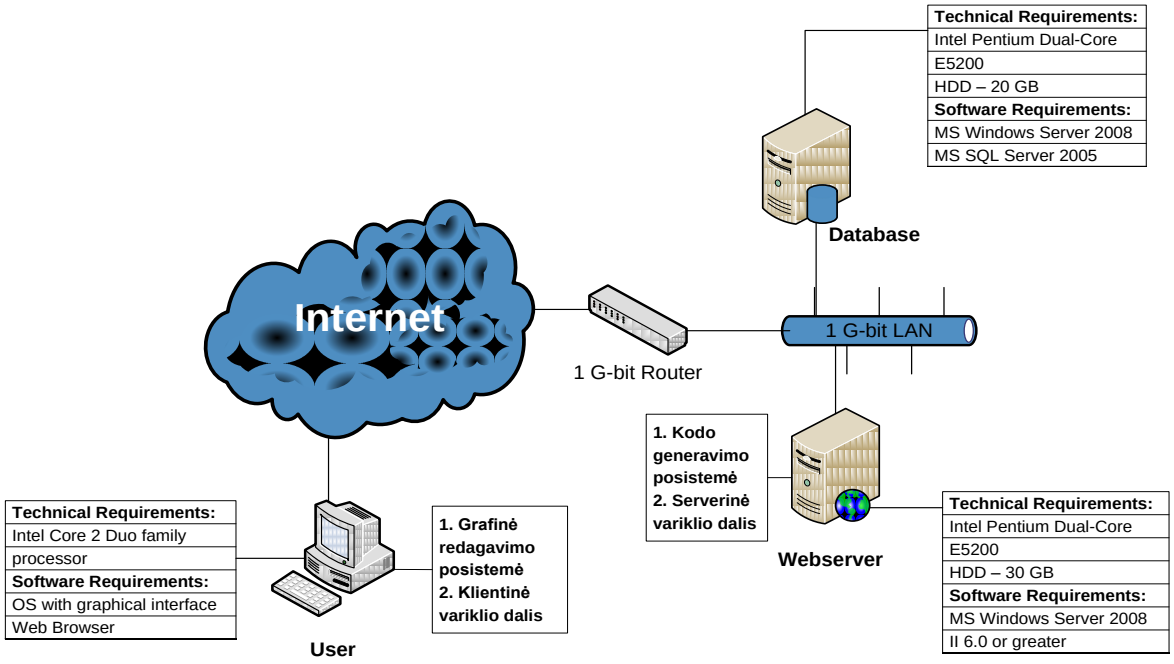
Pav. 40. Veiklos diagrama „generuoti kodą“

6.3.Būsenų diagrama



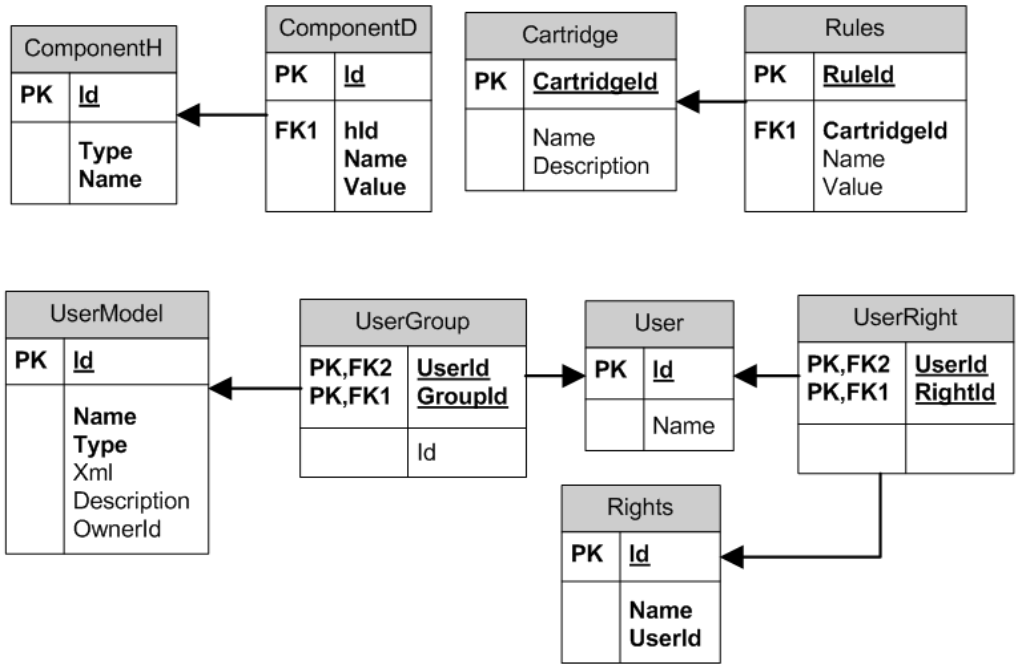
Pav. 41. Būsenų diagrama

7. Išdėstymo vaizdas



Pav. 42. Išdėstymo vaizdas

8. Duomenų vaizdas



Pav. 43. Duomenų vaizdas

9. Kokybė

Lentelė Nr. 25. Kokybės kriterijai

Kriterijus	Aprašymas
Patikimumas (Reliability)	Pasirinkta architektūra mažina sistemos patikimumą, kadangi ji buvo projektuojama pagal PĮ kūrimo šablonus, o tai didina sistemos komponentų skaičių, kas gali sukelti patikimumo problemas dėl sąsajų.
Išplečiamumas (Extensibility)	Pasirinkta architektūra didina sistemos išplečiamumą, kadangi projektuojant panaudoti standartiniai PĮ šablonai leisiantys programinį kodą suskirstyti į individualius vienetus.
Palaikomumas (Maintainence)	Kadangi sistema lengvai plečiama, todėl jos ir palaikomumas lengvas.
Našumas (Performance)	Sudėtingesnė architektūra lėtina sistemos našumą

10.Nuorodos

[1] Factory method pattern [žiūrėta 2009 06 19]. Prieiga internete: http://en.wikipedia.org/wiki/ Factory_method_pattern

[2] Abstract Factory Desig [žiūrėta 2009 06 19]. Prieiga internete: http://www.dofactory.com/Patterns/ PatternAbstract.aspx

[3] Singleton pattern [žiūrėta 2009 06 19]. Prieiga internete: http://en.wikipedia.org/wiki/Singleton_pattern

[4] The Strategy Design Pattern [žiūrėta 2009 06 19]. Prieiga internete: http://www.exciton.cs.rice.edu/ JavaResources/DesignPatterns/StrategyPattern.htm

9.3. Vartotojo vadovas

Data Driven Design metodologija paremtos sistemos vartotojo vadovas.

Šiame dokumente aprašomos sistemos funkcijos bei detalus sistemos naudojimas. Vartotojo vadove suteikiama svarbi informacija apie sistemos veikimą ir naudojimąsi ja, kuri suteiks programuotojui galimybę greitai įsisavinti darbą su sistema

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Įskiepiais paremto programinio kodo generatoriaus architektūra naudojant data-driven-design metodologiją

Vartotojo dokumentacija

Vadovas:
Tomas Blažauskas
2010 01 04

Užsakovas:
Eduardas Bareiša
2010 01 04

Autorius:
IFM-4/2 gr. stud.
Kęstutis Valinčius
2010 01 04

KAUNAS, 2010

1. Apie vartotojo vadovą

Automatinis kodo generatorius yra vidinė bendros sistemos dalis. Prie jos prieiti ir ją modifikuoti gali tik sisteminis administratorius (programuotojas). Vartotojo vadovas yra skirtas tik sisteminiams administratoriams. Šiame dokumente aprašomos sistemos funkcijos bei detalus sistemos naudojimas. Vartotojo vadove suteikiama svarbi informacija apie sistemos veikimą ir naudojimąsi ja, kuri suteiks programuotojui galimybę greitai įsisavinti darbą su sistema.

Vartotojo vadovą sudaro tokie skyriai:

- Sistemos funkcinis aprašymas. Šiame skyriuje yra trumpai apžvelgiamos sistemos galimybės ir paskirtis.
- Detali sistemos atmintinė. Šiame skyriuje detaliai aprašomos visos sistemos funkcijos bei galimybės, taip pat jų naudojimas.
- Programuotojo dokumentacija. Šiame skyriuje detaliai aprašomos visos klasės bei jų atributai ir metodai.
- Instaliavimo vadovas. Čia pateikiama informacija apie sistemos diegimą. Tai skirta sistemos administratoriams. Pateikiama minimali sistemos techninės įrangos bei programinės įrangos konfigūracija.
- Administravimo vadovas. Šiame skyriuje pateikiama informacija apie sistemos administravimą, sistemos konfigūravimą. Administravimo vadove aprašomas sistemos bendravimas su kitomis sistemomis.

2. Sistemos funkcinis aprašymas

Šiame skyriuje trumpai apžvelgiamos sistemos galimybės bei jos paskirtis.

2.1. Trumpas sistemos aprašymas

Visa sistema – ‚Data-Driven-Design metodologija patemtas grafinis kodo generatorius“. „Įskiepiais paremtas programinio kodo generatorius“ yra šios sistemos posistemė.

Įskiepiais paremtas programinio kodo generatorius palengvina sistemos palaikomumą bei pritaikomumą kitai taikymo sričiai. Norint visą sistemą pritaikyti kitai taikymo sričiai tereikia parašyti sistemos specifikaciją UML kalba bei suprogramuoti programinio kodo generatoriaus įskiepį.

2.2. Sistemos paskirtis

Visa sistema kurta su tikslu – neprisirišti prie taikymo srities, sukurti universalų duomenimis paremtą programinio kodo generatorių. Scenarijai projektuojami pagal sistemos specifikaciją (UML), o kodas generuojamas pagal įskiepius.

Įskiepiais paremtas programinio kodo generatorius gali generuoti kodą nepriklausomai net nuo taikymo srities architektūros. Galimas net kodo generavimas keletui technologiniu požiūriu skirtingoms sistemoms vienu metu. Toks programinio kodo generatorius sumažina klaidų tikimybę sistemą tobulinant ir plečiant. Labai sumažinamos laiko sąnaudos, nes nereikia pilno sistemos perrašymo, tereikia tik suprogramuoti įskiepį.

2.3. Sistemos vartotojai

Sistema duomenis gauna „web service‘o“ pagalba, juos analizuoja ir sugeneruotą programinį kodą perduoda per „web service‘ą“. Viskas vyksta automatiškai, nereikalingas žmogaus įsikišimas. Vienintelis vartotojas yra kita bendros sistemos posistemė - grafinis scenarijų kūrimo įrankis.

Bendrą sistemą tobulinant reikalingas programuotojas naujam įskiepiui sukurti.

2.4. Sistemos funkcijos

Kuriama sistema yra skirta sumažinti žmogiškųjų išteklių poreikį, padidinti sistemos universalumą. Pagrindinės įskiepiais paremto programinio kodo generatoriaus funkcijos:

- Neribojamas įskiepių skaičius.
- Skripto analizavimas.
- Automatinis kodo generavimas.

3. Detali sistemos atmintinė

Šiame skyriuje detaliai aprašomos visos sistemos funkcijos bei galimybės.

3.1. Automatinis kodo generavimas

Tai pagrindinė šiuos posistemės funkcija – kodo generavimas iš skripto (XML).

3.1.1. Objekto sukūrimas

Paprasčiausias objekto sukūrimas.

Elemento aprašymo sintaksė

Lentelė Nr. 26. Objekto sukūrimo aprašymo sintaksė

Skriptas (XML)
`<item type="object" name="search" class="Dialog" />`
Sugeneruotas kodas
`var search = new Dialog();`

Elemento aprašymo pavyzdžiai

Lentelė Nr. 27. Objekto sukūrimo pavyzdys

Skriptas (XML)

`<item type="object" name="search" class="Dialog">`
`<param>DialogType.Modal</param>`
`<param id="search" width="285" align="Align.Right" valign="VAlign.Bottom" />`
`</item>`

Sugeneruotas kodas
`var search = new Dialog(DialogType.Modal, { id: "search", width: 285, align: Align.Right, valign: VAlign.Bottom });`

3.1.2. Funkcijos kvietimas

Paprasčiausios funkcijos kvietimas.

Elemento aprašymo sintaksė

Lentelė Nr. 28. Funkcijos kvietimo aprašymo sintaksė

Skriptas (XML)
`<item type="function" name="search.AddControl" />`
Sugeneruotas kodas
`Search.AddControl();`

Elemento aprašymo pavyzdžiai

Lentelė Nr. 29. Funkcijos kvietimo pavyzdys

Skriptas (XML)

`<item type="function" name="search.AddControl">`
`  <param>type1</param>`
`</item>`

Sugeneruotas kodas
`Search.AddControl(type1);`

3.1.3. Parametrai

Parametrai gali būti aprašyti dviems būdais. Kaip žymės ir kaip atributai XML skripte.

Elemento aprašymo sintaksė

Lentelė Nr. 30. Parametrų aprašymo sintaksė

Skriptas (XML)

`<param>DialogType.Modal</param>`
`<param>param1</param>`
`<param>param2</param>`

Sugeneruotas kodas
`DialogType.Modal, param1, param2`

Lentelė Nr. 31. Parametrų aprašymo pavyzdys

Skriptas (XML)

```
<param id="search" width="285" align="Align.Right" valign="VAlign.Bottom" />
```

Sugeneruotas kodas

```
{ id: "search", width: 285, align: Align.Right, valign: VAlign.Bottom }
```

3.1.4. Sąlygos sakinys

Kode atvaizduojamas kai „if () {} else {}“.

Elemento aprašymo sintaksė

Lentelė Nr. 32. Sąlygos sakinio aprašymo sintaksė

Skriptas (XML)

```
          <MyNodeData    Key="Start"    Category="Start"    Location="157    -42"    Text="Start"
Figure="RoundedRectangle" />
          <MyNodeData     Key="End"     Category="End"     Location="270     293"     Text="End"
Figure="RoundedRectangle" />
          <MyNodeData   Key="Conditional"   Category="Standard"   Location="158   49"   Text="Button
obj.text == 1" Figure="Diamond" />
          <MyNodeData  Key="Button  obj.Hide"  Category="Standard"  Location="272  153"  Text="Button
obj.Hide" />
          <MyNodeData Key="Button obj.Enable" Category="Standard" Location="121 220" Text="Button
obj.Enable" />
          <MyLinkData From="Start" To="Conditional" FromPort="0" ToPort="1" Text="Yes" />
          <MyLinkData  From="Conditional"  To="Button  obj.Hide"  FromPort="3"  ToPort="1"  Text="No"
/>
          <MyLinkData From="Button obj.Hide" To="End" FromPort="0" ToPort="1" Text="Yes" />
          <MyLinkData   From="Conditional"   To="Button   obj.Enable"   FromPort="0"   ToPort="1"
Text="Yes" />
          <MyLinkData From="Button obj.Enable" To="End" FromPort="0" ToPort="2" Text="Yes" />
```

Sugeneruotas kodas

```
if (Button obj.text == 1) {
  Button obj.Hide;
} else {
  Button obj.Enable;
}
```

3.2. Skripto analizavimas

Gaunamam XML skriptui atliekamas analizavimas. Analizavimas tai yra iš skripto išrinkimas skirtingų objektų, skirtingų funkcijų kvietimų. Taip pat atliekamas sistemos suprojektuoto funkcionalumo grafo sudarymas. Išrenkant sąlygos sakinius ciklus, pradžios bei pabaigos viršūnes.

Analizavimas reikalingas kaip pagalbinė priemonė projektuojant naują įskiepį. Taip pat suteikia galimybę ateityje vykdyti skripto optimizaciją.

3.3. Įskiepiai

Įskiepia programiškai atvaizduojami kaip klasės su vienoda struktūra.

Jų parinkimas nustatomas konfigūracijos faile.

4. Programuotojo dokumentacija

Šiame skyriuje aprašomos posistemę sudarančios klasės jų atributai bei metodai.

4.1. Karkasinės klasės („framework“ kataloge esančios klasės)

4.2. common.class

Bendrinė klasė skirta bendroms funkcijoms aprašyti.

Lentelė Nr. 33. “common” klasės aprašymas

Atributai	**Paaiškinimas**
$clientIp	Kliento IP adresas
$encoding	Naudojama koduotė
Metodai	**Paaiškinimas**
getIp()	Kliento IP nustatymas
formatDatetime(& $array, $keyName, $datetimeFormat = 'Y-m-d H:i')	Datos formavimas masyvo elementui. $array – masyvas $key – masyvo raktas $datetimeFormat – datos formatavimo stilius. Numatyta reikšmė – 'Y-m-d H:i'.

4.3.dump.inc

Diegėjui skirtas komponentas. Išveda testavimo rezultatus į ekraną.

Lentelė Nr. 34."dump" klasės aprašymas

Atributai	Paaiškinimas
$config	Nustatymų masyvas
Metodai	**Paaiškinimas**
dump($var = null)	Kintamojo išvedimas į ekraną. $var – kintamasis
execute($var = null)	Sugeneruoto kodo paleidimas. $var – kodas

4.4.xmlParser

XML kodo apdorojimo klasė.

Lentelė Nr. 35. "xmlParser" klasės aprašymas

Metodai	Paaiškinimas
xmlize($data)	XML kodo išskaidymas į masyvą $data – XML skriptas

4.5. Biznio logikos klasės („libraries“ kataloge esančios klasės)

4.6. scriptAnalizator.class

Skripto analizavimo klasė.

Lentelė Nr. 36. “scriptAnalizator” klasės aprašymas

Atributai	Paaiškinimas
$xmlParserObj	XML apdorojimo klasės objektas
$script	XML skriptas
$scriptRawArray	XML skriptas išskaidytas į masyvą.
$scriptObjectsArray	Išanalizuotų objektų masyvas
$scriptFlowArray	Išanalizuotų funkcionalumų masyvas
$functionsList	Skirtingų funkcijų kvietimų masyvas.
$objectsList	Skirtingų objektų kūrimo masyvas.
Metodai	**Paaiškinimas**
__construct()	Konstruktorius.
parseScript($script)	XML skripto apdorojimas.
translateScriptArray()	Masyvo papildomas apdorojimas
returnScriptObjectsArray()	Grąžina scenarijaus objektų masyvą.
returnScriptFlowArray()	Grąžina scenarijaus funkcionalumo masyvą.
getFunctionCallList($scriptArray)	Analizuoti funkcijų kvietimus. $scriptArray – scenarijus
getObjectsList($scriptArray)	Analizuoti objektų kūrimus. $scriptArray – scenarijus

4.7.scenario.class

Scenarijaus klasė. Tai pagrindinė logikos klasė.

Lentelė Nr. 37. "scenario" klasės aprašymas

Atributai	Paaiškinimas
$instance	Vienturtis objektas.
$scriptAnalizatorObj	Analizatoriaus objektas.
$codeGeneratorObj	Generatoriaus objektas.
$script	XML skriptas.
Metodai	**Paaiškinimas**
getInstance ()	Grąžina vienturtį.
__construct()	Konstruktorius.
loadScript($sourceFile)	XML failo skaitymas.
analyzeScript()	Skripto analizavimas.
generateScript()	Kodo generavimo iškvietimas.

4.8.codeGenerator.class

Parenka įskiepį ir iškviečia atitinkamą kodo generavimo klasę.

Lentelė Nr. 38. "codeGenerator" klasės aprašymas

Atributai	Paaiškinimas
$instance	Vienturtis objektas.
$cartridgeObj	Įskiepio objektas
Metodai	**Paaiškinimas**
getInstance ()	Grąžina vienturtį.
__construct()	Konstruktorius.
generateCode(($scriptObjectsArray)	Kodo generatoriaus iškvietimas. $scriptObjectsArray – skripto objektų masyvas.

4.9.cartridgeGmaps.class

Kodo generavimo klasė.

Lentelė Nr. 39. "cartridgeGmaps" klasės aprašymas

Atributai	Paaiškinimas
$templates	Kodo generavimo šablonai.
$code	Sugeneruotas kodas.
Metodai	**Paaiškinimas**
__construct()	Konstruktorius.
generateCode($scriptArray)	Kodo generavimas.
generateFlow($flowArray)	Funkcionalumo tvarkos nustatymas.
generateFlowCode(&$code, $key, $flowNodesArray, $flowLinksArray)	Funkcionalumo kodo generavimas.
generateFunction($functionName, $params = '')	Generuoti funkcijos kvietimo kodą.
generateObject($class, $params = '')	Generuoti objekto sukūrimo kodą.
generateAssign($name, $value = '')	Generuoti kintamojo priskyrimo kodą.
generateParams($params)	Generuoto parametrų kodą.
returnCode()	Grąžinti sugeneruotą kodą.

5. Instaliavimo vadovas

Sistema nėra reikli programinės bei techninės įrangos reikalavimams. Todėl kad nenaudojami jokie papildomi plėtiniai.

Sistemos įdiegimas:

- Nusiunčiami failai į priglobos serverį.
- Katalogui „data“ suteikiamos teisės „777“

Programinės įrangos reikalavimai:

- PHP 5

Techninės įrangos reikalavimai:

- Procesorius: 1GHz
- Operatyvioji atmintinė: 256MB
- Kietasis diskas: 5GB

6. Administravimo vadovas

Šis vadovas skirtas programuotojams tobulinantiems sistemą.

Sistemos konfigūracijos nustatymai saugomi šakniniame kataloge esančiame „config.inf.php“ faile. Kadangi sistema savo funkcijas atlieka automatiškai, be vartotojo įsikišimo ir dėl suprojektuotos lanksčios struktūros nustatymų nėra daug. Pagrindinis nustatymas – naudojamo įskiepio vardas.

Sistemos naudojamos klasės bei jų užkrovimas valdomas šakniniame kataloge esančiame „init.inc.php“ faile.

7. Apibrėžimai ir sutrumpinimai

Žemiau yra išvardinti apibrėžimai ir sutrumpinimai, panaudoti šiame dokumente:

UML	Unifikuota modeliavimo kalba, naudojama objektiškai orientuotame projektavime (angl. Unified Modeling Language)
XML	Išplečiama žymių (arba teksto išdėstymo ir struktūrizavimo) kalba (angl. eXtensible Markup Language)