KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Sigitas Povilaitis

# DDD paremto projektavimo įrankio grafinio modelių redaktoriaus kūrimas ir tyrimas

Magistro darbas

Darbo vadovas

prof. Eduardas Bareiša

**KAUNAS, 2010**

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Sigitas Povilaitis

# DDD paremto projektavimo įrankio grafinio modelių redaktoriaus kūrimas ir tyrimas

Magistro darbas

Recenzentas

prof. Rimantas Butleris

2010-05-28

Vadovas

prof. Eduardas Bareiša

2010-05-28

Atliko

IFM-4/2 gr. studentas
Sigitas Povilaitis

2010-05-28

**KAUNAS, 2010**

# DDD paremto projektavimo įrankio grafinio modelių redaktoriaus kūrimas ir tyrimas

## SANTRAUKA

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.

Darbo tikslas buvo sukurti priemonę grafiškai modeliuoti programinės sistemos veiklą. Kuriant scenarijus grafiškai mažėja klaidų tikimybė, spartėja darbo našumas ir užtenka minimalių programavimo žinių. Šis tikslas buvo pasiektas naudojant tokius architektūrinius sprendimus, kaip modelio redagavimas statiniame ir dinaminiame vaizde, įrankio iškėlimas į internetą naudojant „Silverlight“ karkasą.O iš sistemoje sudaryto metamodelio programa generuoja tarpinį metakalbos skriptą, kuris transformuojamas į programinį kodą. Metakalba buvo sukurta atsižvelgiant į programavimo kalbų įvairovę, o jos saugojimui panaudota „XML“ kalba.

Eksperimentinėje magistro tezių dalyje patvirtinome hipotezę, kad grafinis sistemų modeliavimas yra daug efektingesnis už įprastą programavimą. Kai kuriais atvejais matomas 7 kartų pagreitėjimas. Taip pat grafinis modeliavimas sukelia mažiau klaidų, todėl – paspartina kūrimą.

# DDD methodology based design tool‘s graphical model editor development and research

## SUMMARY

Data Driven Design is widely used among computer systems. Main advantage of this methodology is to separate programmers and designers work. As a result this work can be made concurrently. In practice functionality of system engine is implemented as interface whereas system changes is expressed in form of script. This design adds more layer of abstraction, which lets system to be more flexible, easier to maintain and support. And these actions can be done at the same time.

Purpose of this work was to implement graphic tool to model software activity. When code snippets are generated by modeling system activity graphically certain problems can be evaded or minimized. These problems include error count, renewal speed and poor programming knowledge of average user. This intention is achieved by using several architectural features. Key features include model editing in static and dynamic view, implementing graphical tool in Microsoft Silverlight framework and deploying it on web server. To exchange metamodel among subsystems universal metalanguage was created. It incorporates differences and commonalities of multiple programming languages and is subset of XML.

Expermental studies had showed that graphical modeling of computer systems is much more effective than implementing it manually. In some cases seven times faster development has been viewed. Also it has been noted that suggested modeling process allows to minimize error count therefore speed of the process.

# TURINYS

# LENTELIŲ TURINYS

# PAVEIKSLŲ TURINYS

# PRATARMĖ

Informacinės technologijos – viena iš greičiausiai besivystančių mokslo šakų. Pastebėta, kad programinės įrangos sudėtingumo lygis proporcingas kitimo spartai. Tai galima pritaikyti ne tik globaliai, bet ir konkrečiam projektui. Kuo ilgesnis sistemos kūrimo procesas, tuo labiau keičiasi jos reikalavimai. O Programų reikalavimams keičiantis, dokumentacijos turinys pamažu sensta ir nebeatitinka realizuojamos sistemos. Tokiu atveju sistemos pakeitimai tampa daug sudėtingesni, nes perprasti programos veikimą galime tik iš programinio kodo.

Šis darbas buvo atliktas naudojantis duomenimis ir modeliais grįstomis metodologijomis. Programa buvo realizuota žiniatinklio technologijomis naudojant „Microsoft Silverlight“ karkaso galimybes.

Magistro tezių įvade išdėstysime duomenimis paremtą architektūrą, modeliais paremtą architektūrą ir palyginsime su pasiūlytu projektavimo modeliu. Analizės dalyje apžvelgsime MDA įrankius bei kitus modeliavimo įrankius. Taip pat supažindinsime su grafiniu modelių redaktoriumi ir probleminėmis sritimis. Darbo projektinėje dalyje pateiksime detalesnę posistemės architektūrą. Tiriamojoje dalyje apžvelgsime iškilusias problemas ir pateiksime jų sprendimo būdus. Tezių eksperimentinėje dalyje aprašomas eksperimentinis tyrimas, kuriame palyginsime programavimo tekstu ir projektavimo scenarijų įrankiu greičius. Paskutiniame skyriuje pateiksime svarbiausias tezių išvadas.

# 1. ĮVADAS

DDD[1] 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, o 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].

Toks programų kūrimas ypač paplitęs žaidimų kūrime. Sukūrus žaidimą pagal Data Driven Design metodologiją, jis gali būti užbaigtas skriptų pagalba. Taip pat galima atlikti testavimą, ar net keisti jo veikimą tiek, kad gims naujas žaidimas. Taigi naudodami skirtingus projektavimo sluoksnius ir „supaprastinus sudėtingus ar problematiškus etapus sumažinsime ir riziką“ [2]. Ši taisyklė galioja ir kitoms kompiuterinėms programoms.

Duomenimis paremtos architektūros idėja yra panaši į MDA[2]. Modeliais paremta architektūra yra naudojama sudaryti pilną programų sistemos modelį ir realizuoti programą pasirinktoje architektūroje. Tuo tarpu DDD skirta koreguoti jau sukurtą sistemą.

## 1.1. Modeliais paremta architektūra

Modeliais pagrįsta architektūra apibrėžia informacinių sistemų specifikacijos taikymą, kai sistemos veikimo specifikacija atskiriama nuo sistemos realizacijos specifikacijos, konkrečiai technologiniai platformai[3]. Ši programinės įrangos kūrimo architektūra sukurta OMG[3]. Pagrindinis MDA aspektas yra modelių svarba programinės įrangos kūrimo procese. Naudojant MDA, programinės įrangos kūrimo procesas yra pagrįstas sistemos modelių kūrimu. Modeliais paremta architektūra bando spręsti problemas sutinkamas tradiciniame programinės įrangos kūrimo procese. Modelio reikšmę galime aprašyti šiais teiginiais:

- Modelis tai supaprastintas realaus ar planuojamo kūrinio atitikmuo (modelyje nematome programinio kodo subtilybių).
- Modelis skiriasi nuo realaus objekto (nors galutinę programos versiją gauname iš modelio, tam reikalingos transformacijos).

[1]DDD (*angl. Data Driven Design*) – duomenimis paremta architektūra
[2] MDA (*angl. Model Driven Architecture*) – modeliais paremta architektūra

- Modelį galime naudoti kaip pavyzdį (grafinė modelio išvaizda yra panaši į galutinę programos versiją).

### 1.1.1. MDA modelių tipai ir transformacijos

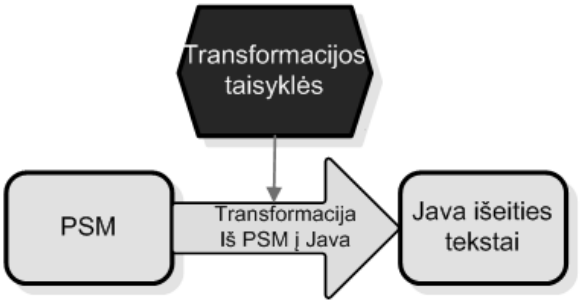
Modeliais paremta architektūra naudoja keletą modelių tipų atvaizduoti projektuojamą sistemą. Pradedant nuo bendriausio, reikalavimų surinkimo stadijoje sukurto, modelio. Baigiant modeliu, sukurtu žinant apie taikymo sritį, kuris bus transformuotas į galutinį programinį kodą.


1 pav. MDA modeliai ir transformacijos

Iliustracijoje (1 pav) matome MDA naudojamus modelių tipus:

- CIM[4] – nuo skaičiavimų nepriklausomas modelis. Jis sudaromas iš reikalavimų surinkimo stadijoje. Kai kurie MDA įrankiai šio modelio nepalaiko.
- PIM[5] – nuo platformos nepriklausomas modelis, kuriame nėra jokių žinių apie galutinę platformą.
- PSM[6] – nuo platformos priklausomas modelis, tai modelis kuris atvaizduoja sistemą su žiniomis apie platformą, kuriame ji bus realizuota.

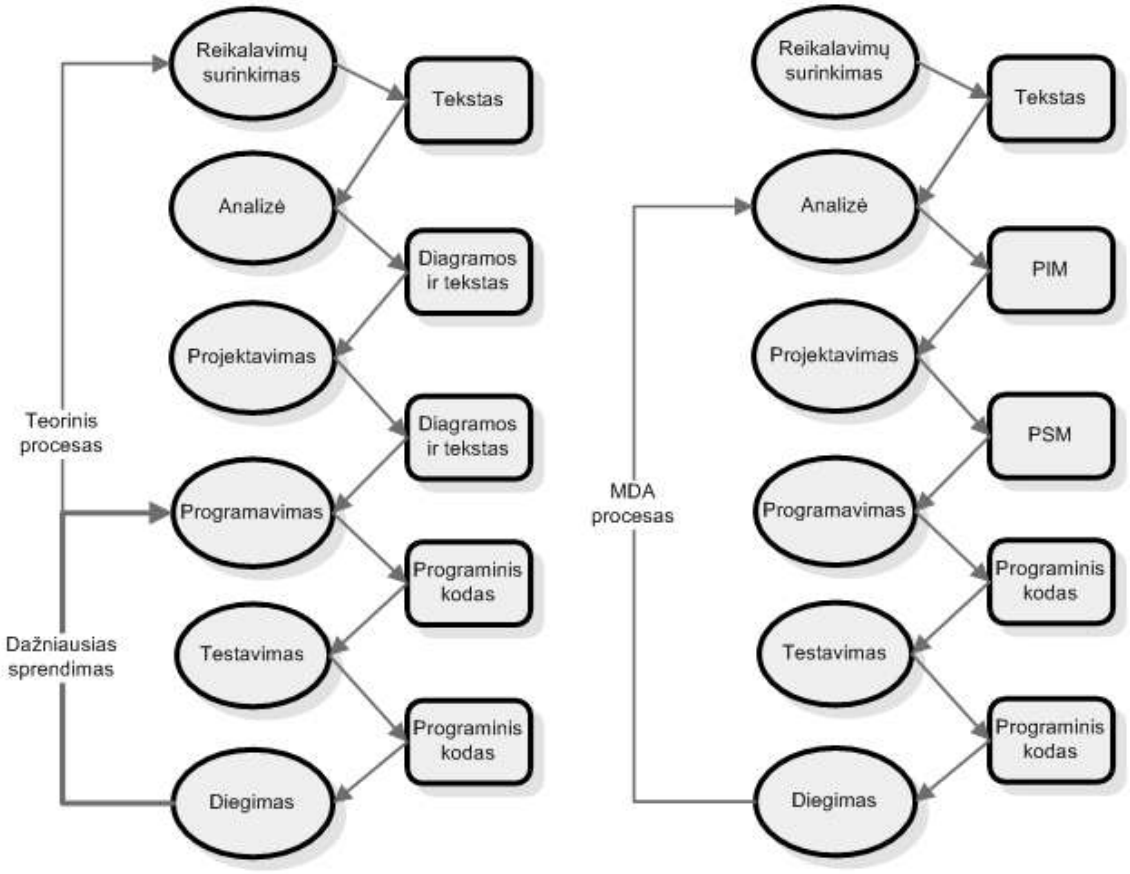
Šie modeliai MDA proceso metu transformuojami naudojant transformavimo kalbas. Transformacijos į programinį kodą aprašomos kiekvienai programavimo kalbai atskirai (pvz. 2 pav pavaizduoja transformaciją iš PSM modelio į Java kalbą).




[3] OMG (*angl. Object Management Group)* – tarptautinė programų kūrėjų grupė, ruošianti MDA standartus
[4] CIM (*angl. Computation Independent Model*) – nuo skaičiavimų nepriklausomas modelis
[5] PIM (*angl. Platform Independent Model*) – nuo platformos nepriklausomas modelis
[6] PSM (*angl. Platform Specific Model*) – nuo platformos priklausomas modelis

2 pav. Transformacija iš PSM modelio į Java programinį kodą.

### 1.1.2. MDA programų kūrimo procesas




3 pav. Įprasto (kairėje) ir modeliu paremtos architektūros (dešinėje) programų kūrimo proceso palyginimas

Palyginus tradicinį ir modeliais pagrįstą programų kūrimo procesą (3 pav) matome, kad MDA gali sugeneruoti programinį kodą sudarius PIM modelį. Kai kurie paketai naudoja CIM modelį ir darbas dar labiau paspartėja. Tuo tarpu tradicinis procesas reikalauja visus žingsnius atlikti žmonėms. Tai reikalauja daugiau kaštų, o kai jie taupomi ir procesas trumpinamas (pvz. naudoja XP[7] kūrimo procesą) sistemos dokumentacija pasensta ir tampa nepanaudojama.

[7] XP (*angl. eXtreme Programming*) – ekstremalus programavimas

## 1.2. Grafinis projektavimo įrankis

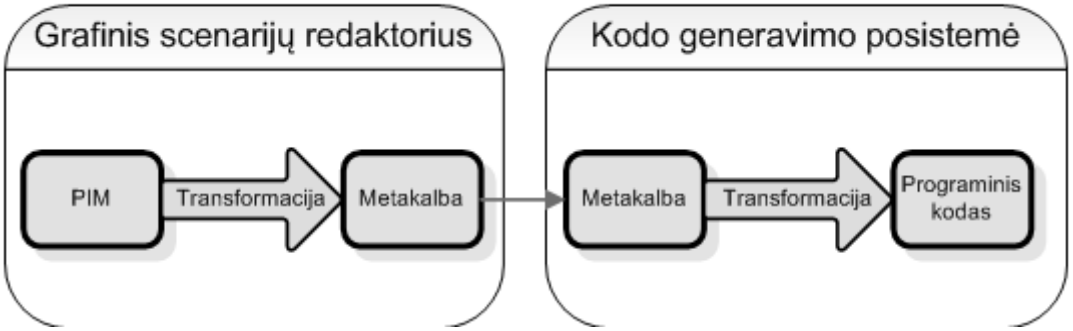
Šis įrankis apjungia duomenimis ir modeliais paremtas architektūras, leisdamas kurti naujus panaudos atvejus jau egzistuojančiai sistemai. Galutiniam programiniam kodui gauti reikalinga sistemos UML[8] specifikacija bei naujų galimybių modelis.

### 1.2.1. Modelių tipai ir transformacijos

Projektavimo įrankyje naudosime tik PIM modelį ir tarpinę metakalbą pagal kurią parašytas skriptas bus transformuotas į išeities tekstą.

- PIM – tai iš UML specifikacijos gautas modelis. Naudojant šį modelį bus atvaizduojami objektai bei jų funkcijos ir atributai.
- Metakalba – tai sukurtas formatas, kurio pagalba bus saugomas suprojektuoto panaudos atvejo metamodelis. Šis metamodelis bus transformuojamas į galutinį kodą.

Šiuos modelius ir transformacijas galima matyti (4 pav) iliustracijoje.



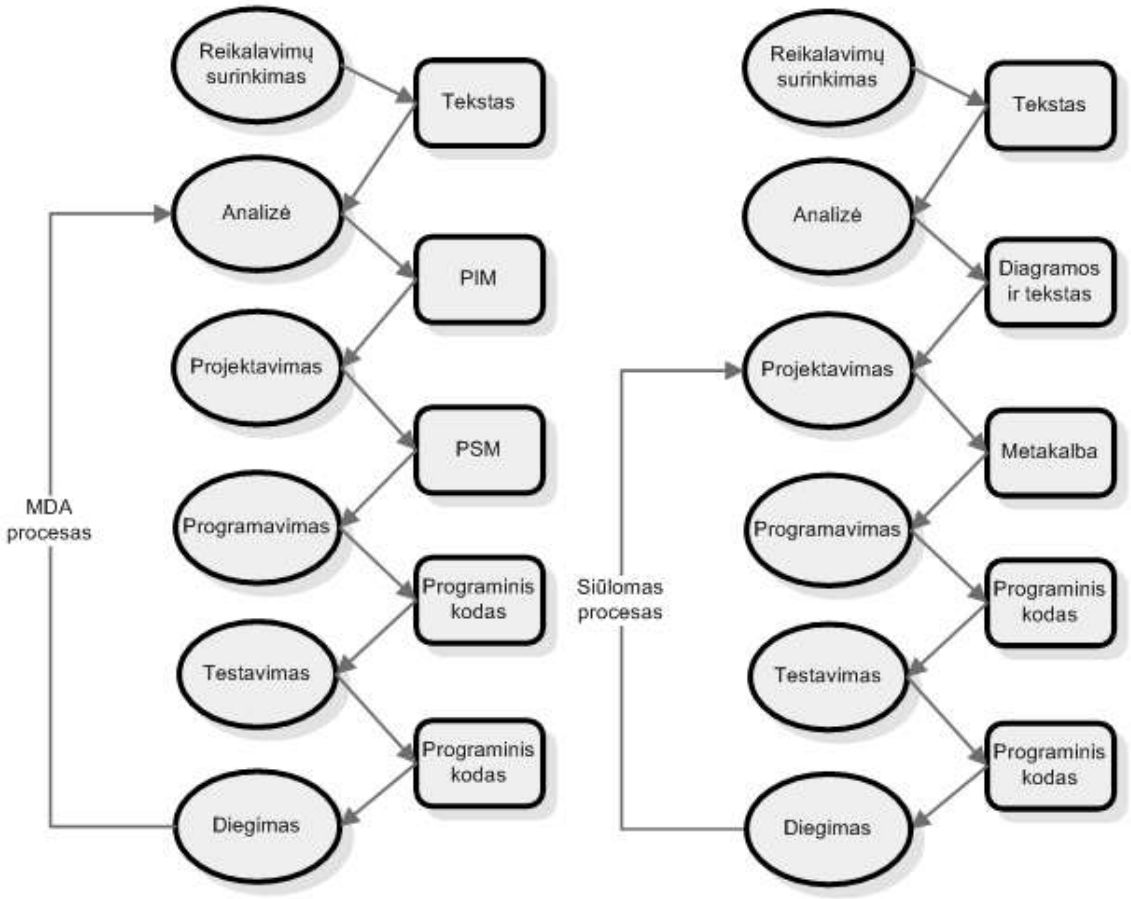

4 pav. Grafinio scenarijų redagavimo įrankio vidiniai modeliai ir transformacijos

- PIM transformacija į tarpinę kalbą bus atliekamos grafinio scenarijaus redaktoriaus posistemėje
- Metakalbos transformaciją į galutinį programinį kodą atliks kodo generavimo posistemė pagal iš anksto aprašytas keitimo taisykles. Šios taisyklės sudaromos konkrečiai taikymo sričiai (pvz. programavimo kalba, karkasas) ir vadinamos įskiepiais.

[8] UML (*angl. Unified Modeling Language*) – Vieninga modeliavimo kalba

### 1.2.2. Programų kūrimo procesas

Siūlomas procesas (5 pav) šiek tiek skiriasi nuo MDA programų kūrimo proceso. Naudojant grafinį scenarijų redagavimo įrankį PSM modelis keičiamas metakalba. Tai universali programavimo kalba atvaizduota XML[9] standartu ir sudaryta taip, kad būtų lengvai transformuojama į konkrečią kalbą. Proceso taikymas pradedamas projektavimo stadijoje, kai sumodeliuojama UML specifikacija.




5 pav. MDA (kairėje) ir siūlomas (dešinėje) programų kūrimo procesas

## 1.3. Apibendrinimas

Nors modeliais ir duomenimis paremtos architektūros yra panašios, pagrindinis skirtumas yra jų taikymas. Modeliu paremtos architektūros taikymas remiasi transformaciją į pilną sprendimą, o duomenimis paremtą metodologiją taikome, kai egzistuojanti sistema pritaikoma naujam funkcionalumui įgyvendinti. Kuriamas scenarijų įrankis palaikys programų kūrimo procesą nuo projektavimo dalies ir turės tik PIM modelį ir metakalbą.

[9] XML (*angl. eXtensible Markup Language*) – praplečiama žymėjimo kalba

# 2. GRAFINIO PROJEKTAVIMO ĮRANKIO ANALIZĖ

Šiame skyriuje apžvelgsime grafinio įrankio problemas, iškelsime hipotezes, taip pat ištirsime esamus sprendimus rinkoje ir juos palyginsime. Taip pat bus apžvelgtos grafinio redagavimo posistemės problemos ir palyginti egzistuojantys produktai.

## 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“ karkasą.

#### 2.1.4.2. Kodo generatorius

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

#### 2.1.4.3. Duomenų sluoksnio modelis

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.

#### 2.1.4.4. Hipertekstinės grafinės vartotojo sąsajos biblioteka

Eksperimentiniai tyrimai ir HTML[10] grafinės vartotojo sąsajos bibliotekos realizacija atlikta deklaratyvia XML ir imperatyvia JavaScript 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 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[11] projektus, realizuotus Java programavimo kalba iš UML modelių. Generuojami žiniatinklio projektai pritaikyti Hibernate[12], EJB[13], Struts[14], Spring[15] ir WebServices[16] technologijoms.

Principinė „AndroMDA“ veikimo schema pavaizduota 6 pav.

[10] HTML (*angl. Hyper text Markup Language*) – tai kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete
[11] J2EE (*angl. Java 2Enterprise Edition*) – standartinė daugialyčių programų kūrimo Java kalba platforma
[12] Hibernate – Java objektų saugykla
[13] EJB (*angl. Enterprise JavaBeans*) – modulinė verslo sprendimų kūrimo architektūra
[14] Struts – atviro kodo karkasas, skirtas kurti internetinėms aplikacijoms
[15] Spring – atviro kodo karkasas, skirtas kurti .NET ir Java aplikacijoms
[16] WebServices – žiniatinklio paslauga


6 pav. „AndroMDA“ veikimas

Programų sistemos generavimas naudojant „AndroMDA“ veiksmų seka:

- UML įrankiu („MagicDraw“, „Poseidon UML“, „Rational Rose“) sukuriamas sistemos nuo Platformos Nepriklausomas Modelis ir eksportuojamas į XMI[17] dokumentus.
- Iškviečiama „AndroMDA“ programa kuri atlieka reikalingas transformacijas.

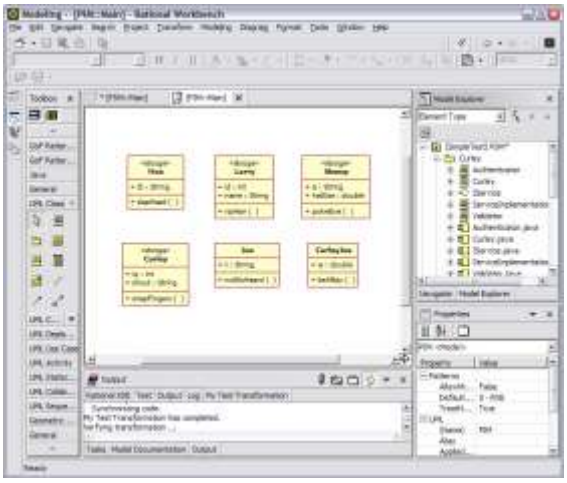
Transformacijos aprašomos įskiepių[18] sistema. Todėl vartotojas turi galimybę kurti savo transformacijos modelius. Šie modeliai aprašomi šabloniniu principu Java programavimo kalba.

Darbas su įrankiu vyksta terminalo[19] 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“ (7 pav) tai yra modeliavimo įrankio papildinys, leidžiantis atlikti kodo generavimą iš UML modelių. Bendrą paketą sudaro modeliavimo įrankis, MDA transformacijų įrankis ir kodo generavimo įrankis.

[17] XMI (*angl. XML Metadata Interchange*) – XML formatas skirtas aprašyti UML diagramas
[18] Įskiepis (*angl. Cartrige*) – lengvai pakeičiama sistemos dalis, transformacijoms aprašyti
[19] Terminalas (*angl. Console*) – vartotojo sąsaja, skirta tekstinėm komandoms įvesti



7 pav. „Rational XDE MDA Toolkit“ vaizdas

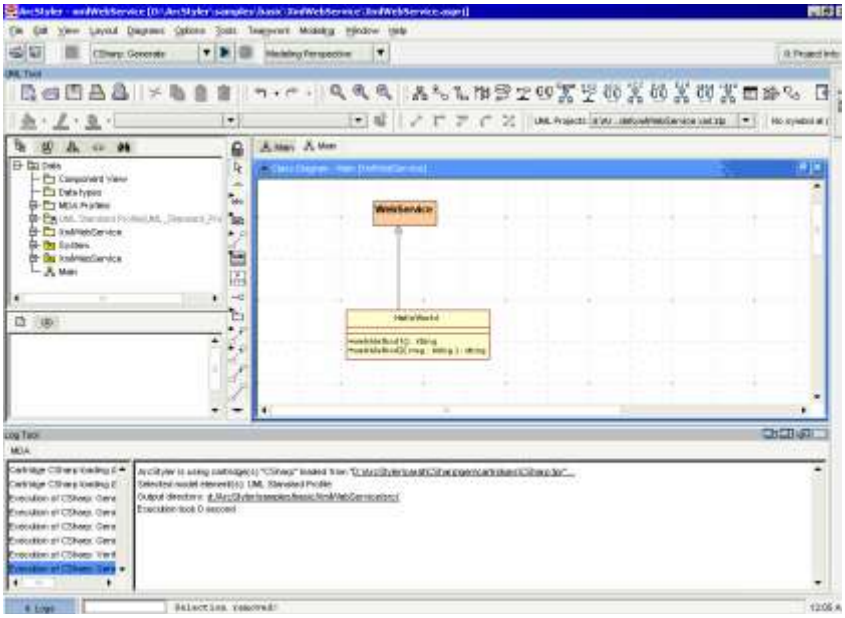
Įsidiegus „Rational XDE MDA Toolkit“ įrankį pastebėta kad nėra paruoštų transformacijų bibliotekų tarp modelių, jas reikia pasiruošti patiems. Šis įrankis nėra iki galo išbaigtas.

#### 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.

„Arcstyler“(8 pav) 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.



8 pav. „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, J2EE, CORBA, .NET platformose.
- Žinomiausių modeliavimo įrankių palaikymas („Rational Rose“, „Poseidon UML“, „MagicDraw“).
- Aspektais paremto programavimo palaikymas.

#### 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, vadinami šablonais. „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.

### 2.1.6. MDA įrankių įvertinimas

Įvertinsime aukščiau aprašytus modeliavimo įrankius pagal šiuos kriterijus:

- PIM palaikymas. Ar yra galimybė sudaryti PIM modelį.
- PSM palaikymas. Ar yra galimybė sudaryti PSM modelį.
- Ar gali generuoti skirtingus PSM. Ar įrankis leidžia generuoti įvairius PSM iš to pačio PIM modelio.
- Modelių integracija. Galima dirbti su keliais modeliais iš kurių generuojama viena programa.
- Programinės įrangos vystymas. Įrankis palaiko programinės įrangos vystymą.
- Bendravimas su kitais įrankiais. Įrankis gali bendrauti su kitais įrankiais, importuoti eksportuoti modelius.

- Galimybė kurti savo transformacijas. Įrankis leidžia kurti savo transformacijas, t.y. nepasitenkinama tik paruoštomis transformacijomis.
- Korektiškumas. Įrankis teikia priemonę patikrinti modelių korektiškumą, ar jie atitinka nustatytas taisykles.
- Išraiškingumas. Įrankis pateikia pakankamai išraiškingas PIM, PSM notacijų priemones, kuriomis galima pilnai išreikšti sistemos modelį.
- Šablonai ir apibendrinimai. Įrankis leidžia kurti modelio elementų šablonus.
- Kodo pertvarkymas. 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.

UML įrankių įvertinimo rezultatai pateikiami lentelėje (1 lentelė).

1 lentelė. UML įrankių įvertinimo rezultatai[4]

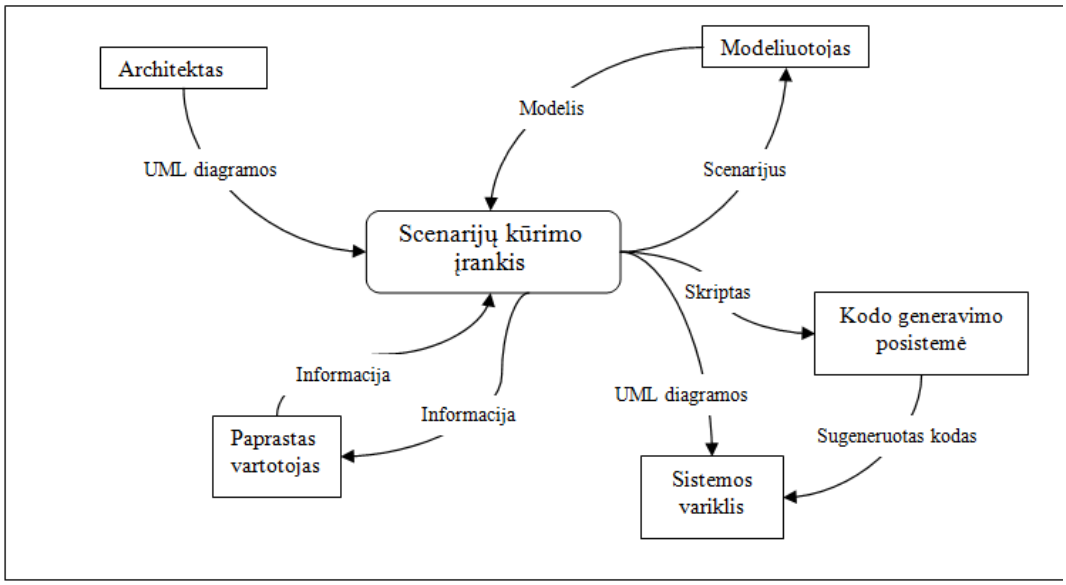
| Kriterijus | Andro MDA | Rational XDE MDA Toolkit | ArcStyler | Open MDX | OptimalJ |
|---|---|---|---|---|---|
| PIM palaikymas | ✓ | ✓ | ✓ | ✓ | ✓ |
| PSM palaikymas | ✓ | ✓ | ✓ | ✓ | ✓ |
| Ar gali generuoti skirtingus PSM | ✓ | ✓ | ✓ | ✓ | × |
| Modelių integracija | × | ✓ | ✓ | × | ✓ |
| Programinės 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“.

### 2.1.7. 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 (9 pav), o aprašymas lentelėje (2 lentelė).

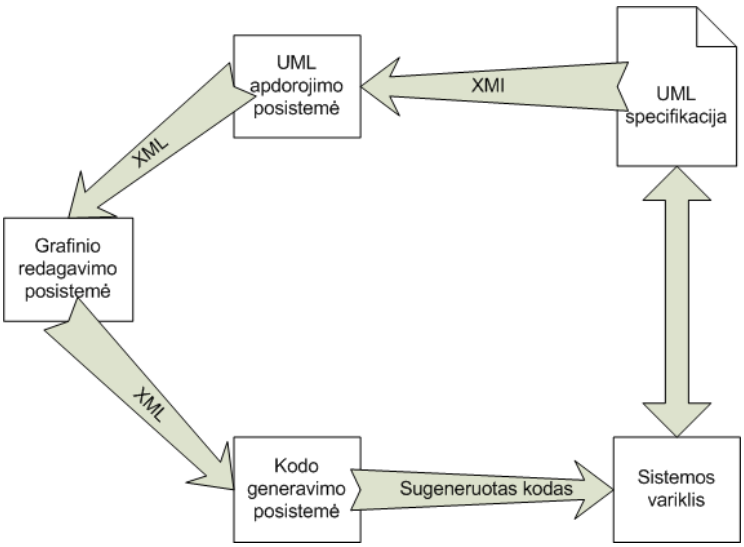



9 pav. Bendros sistemos veiklos sfera

2 lentelė. Veiklos įvykių sąrašas

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

### 2.1.8. Veiklos pasidalinimas




10 pav. Įrankio architektūros diagrama

#### 2.1.8.1. Data Driven Design metodologija paremto projektavimo įrankio grafinio modelių redaktoriaus kūrimas ir tyrimas

Grafinis scenarijų kūrimo įrankis skirtas grafiškai kurti scenarijus, modeliuoti veiklos diagramas. Posistemė naudojasi duomenų sluoksnio posistemėje patalpintu UML modeliu. O sudarytas modelis transformuojamas į tarpinį abstrakcijos lygmenį (skriptą) ir perduodamas įskiepiais paremtai kodo generavimo posistemei.

#### 2.1.8.2. Data Driven Design 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.

#### 2.1.8.3. Data Driven Design 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.

#### 2.1.8.4. Hipertekstinės grafinės vartotojo sąsajos kūrimas aukšto abstrakcijos lygmens deklaratyvia sintakse

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

## 2.2. Grafinis scenarijų redaktorius

### 2.2.1. Posistemės paskirtis

Grafinė sistemos redagavimo posistemė skirta keisti sistemos funkcionalumą, modeliuojant jį scenarijais. Ji leidžia vartotojui kurti objektus ir jais manipuliuoti įvedant naują funkcionalumą. Ši įrankio dalis taip pat atsakinga už modelio transformavimą į metakalbą.

Šis įrankis suskirstytas į dvi dalis:

- Objektų kūrimo įrankis.
- Scenarijų modeliavimo įrankis.

Duomenys į objektų kūrimo įrankį atkeliaus iš sistemos specifikacijos apdorojimo dalies, o pačius objektus naudos scenarijų įrankis. Po grafinio redagavimo duomenys bus siunčiami kodo generavimo posistemei XML formatu.

### 2.2.2. Iškeltos problemos posistemei

- Esant sudėtingoms sistemų architektūroms, programuotojams reikia daug laiko jas perprasti.
- Programų kūrėjams reikia išmokti dirbti su vis daugiau programavimo kalbų.
- Rašant programinį kodą daroma daug sintaksinių klaidų.

### 2.2.3. Kūrimo pagrindas

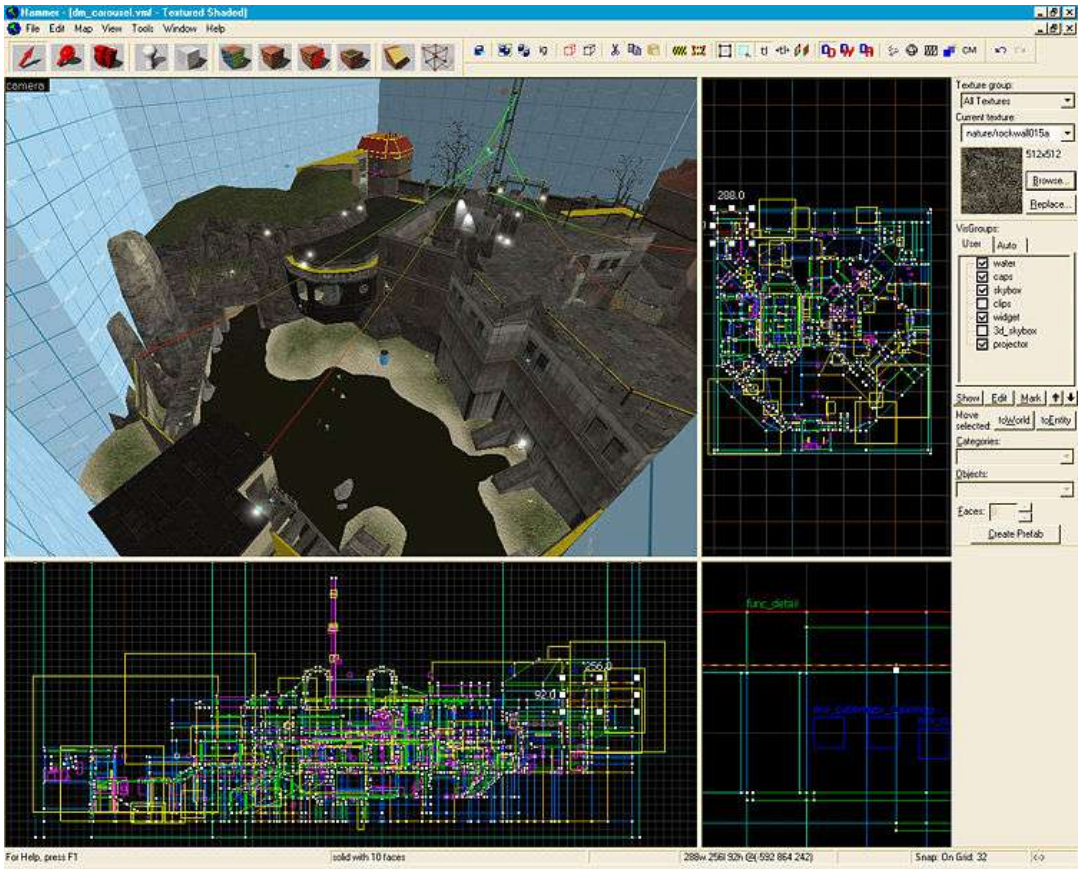
Iškeltų problemų sprendimas grafinio modelių redaktoriaus lygmenyje.

- Architektūros struktūrizavimas ir grafinis scenarijų modeliavimas yra paprastesnis ir aiškesnis.
- Grafinis redaktorius generuoja scenarijaus modelį, kuris gali būti transformuotas į konkrečią kalbą.

- Generuojant programinį kodą grafiškai, išlieka tik loginės klaidos.

### 2.2.4. Egzistuojantys sprendimai

#### 2.2.4.1. „Valve hammer editor 3.5“




11 pav. „Valve hammer“ žemėlapių redaktorius

Šis produktas skirtas kurti populiaraus žaidimo „Counter Strike“ žemėlapius. Šiuo įrankiu modeliuojama aplinka, parenkami žaidimo veikėjai, ginklai. Šio redaktoriaus sukurtuose virtualiuose pasauliuose „vartotojas ne tik lankankosi, tačiau jis taip pat gali sąveikauti su kitais objektais lygyje“[6].

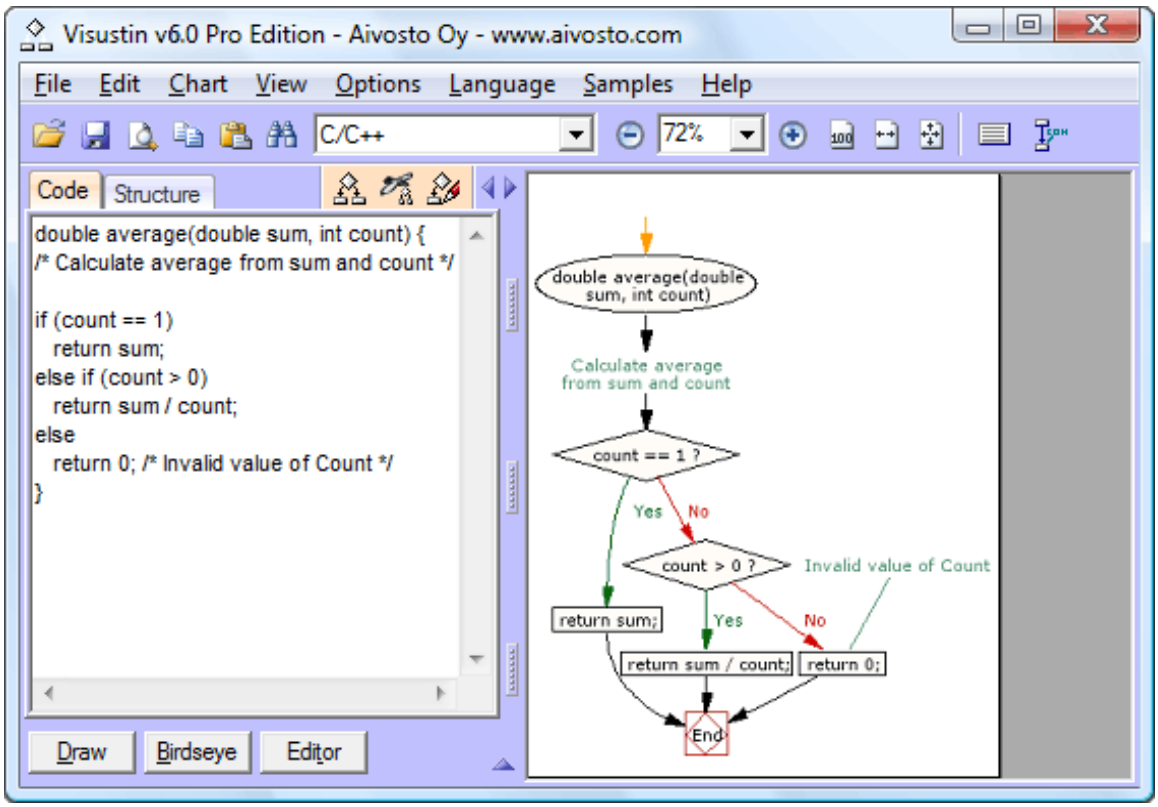
Šis įrankis žymiai supaprastina dizainerių darbą ir leidžia lygiagretinti žaidimo kūrimą ir žemėlapių šiam žaidimui kūrimą.

Šiuo produktu žaidimų kūrėjai išsprendė kelias problemas:

- Lygiui sukurti galėjo ir dizaineriai (nebereikėjo programavimo žinių kurti paprastus žemėlapius).
- Žemėlapius kurti galėjo žaidėjai susipažinę tik su šiuo įrankiu. Jiems nebūtina žinoti apie žaidimo vidinį variklį.

- Taip pat nauji lygiai galėjo būti kuriami po žaidimo išleidimo. Tam žaidime įdiegta sinchronizavimo sistema, atsiunčianti neturimus žemėlapius.

#### 2.2.4.2. „Visustin v5 Flow chart generator“




12 pav. „Visustin“ įrankio vaizdas

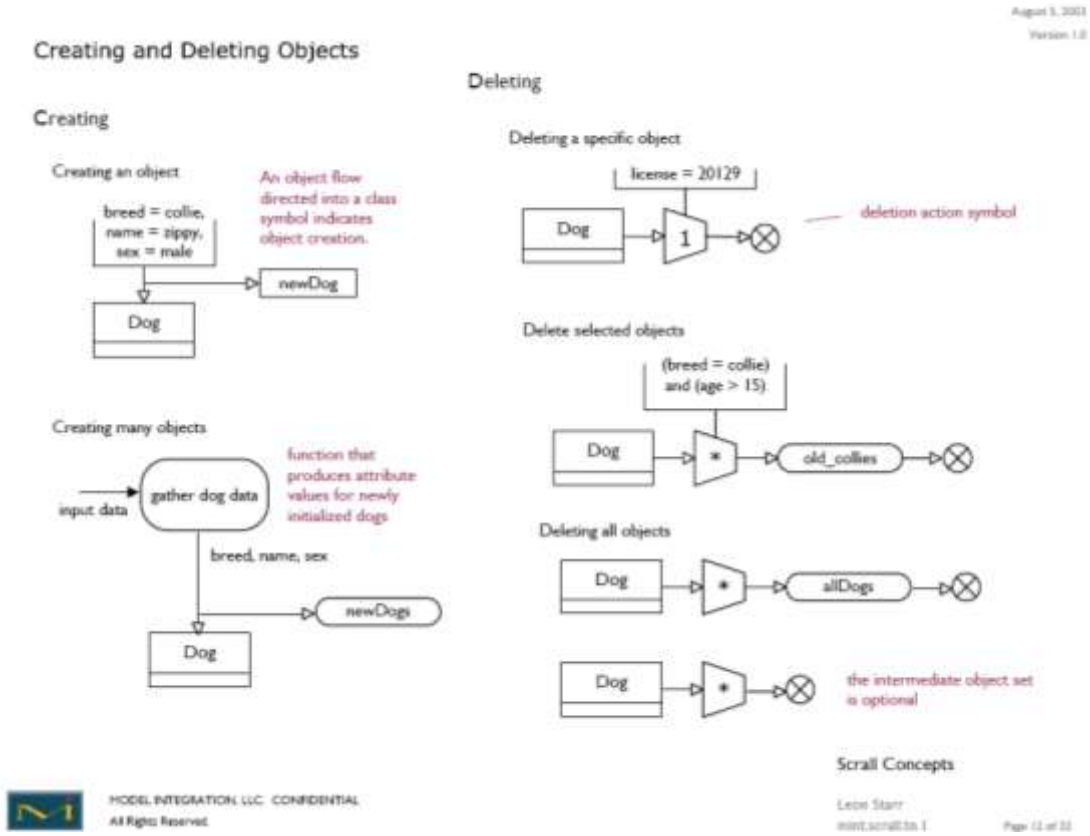
Šis produktas gali atvaizduoti programinį kodą į diagramas, panašias į UML veiklos diagramas. Taip pat galima atlikti atvirkščią veiksmą.

Šiuo įrankiu išspręstos šios problemos:

- Automatizuojamas programos įgyvendinimo etapas. Užtenka algoritmui suprojektuoti veiklos diagramą ir ji bus realizuota.
- Kelių programavimo kalbų palaikymas.
- Ši programa leidžia suprasti programinį kodą nesigilinant į programinės kalbos ypatybes ar sintaksę.
- Atsiradus programos pakeitimams galima sugeneruoti naują veiklos diagramą.
- 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“[7].

### 2.2.4.3. „SCRALL“ atvaizdavimo kalba




13 pav. Scrall kalbos anotacijos pavyzdys

Ši kalba dar neturi programinės priemonės automatizuoti kūrimą. Šios kalbos tikslas aprašyti programos veiklą objektų ir klasių lygmenyje, taigi tai labiau abstrakti kalba nei „Visustin“.

Šio projekto privalumai:

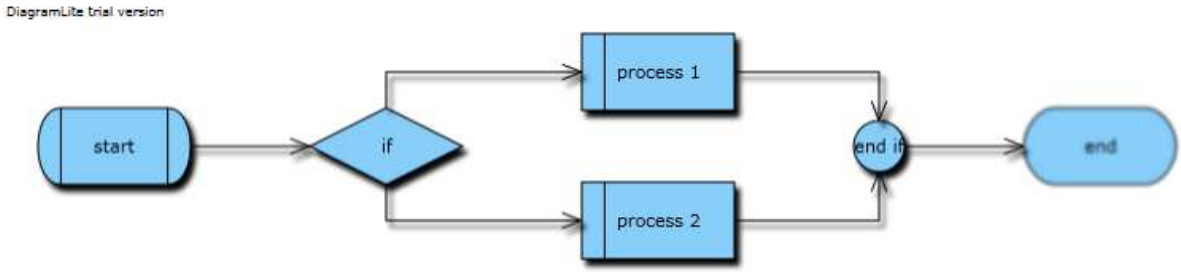
- Šia kalba atvaizduojami klasės, objektai ir tarpusavio ryšys.
- Projektavimas tampa detalesnis nei klasikinis UML.
- Leidžia įterpti programos kodą į UML.

Trūkumai:

Ši kalba nėra pritaikyta UML, tačiau „bus bandymų pritaikyti kai kuriuos Scrall simbolius ir notaciją, kad kuo labiau priartinti prie UML notacijos standarto“[8].

### 2.2.5. Egzistuojantys scenarijų modeliavimo sprendimai

#### 2.2.5.1. „Mindfusion diagramlite“




14 pav. „Mindfusion diagramlite“ veiklos diagramos pavyzdys

Tai mokama komponentų biblioteka pritaikyta įvairioms platformoms. Ji praplečia „Silverlight“ vizualizacijos galimybes. Biblioteka yra lengvai praplečiama ir pritaikoma pagal poreikius. Šio paketo galimybės:

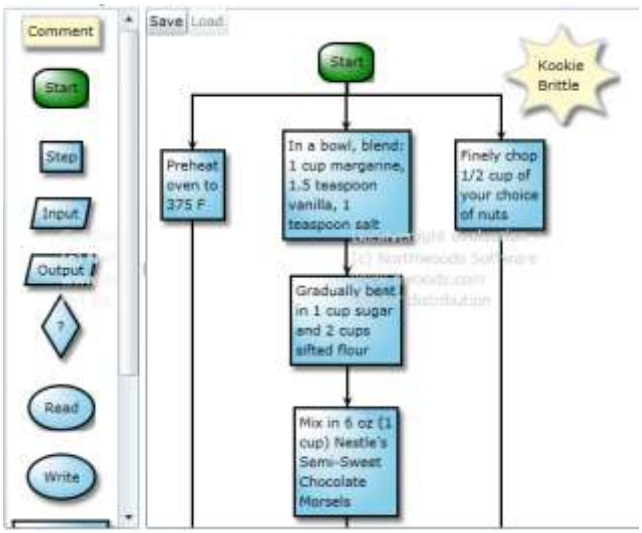
Veiklos diagramų palaikymas

- Objektų hierarchija.
- Grafų ir tinklų grafikai.
- Organizacijų schemos.

Trūkumai:

- Nėra įgyvendintas diagramų saugojimas.
- Biblioteka mokama.

#### 2.2.5.2. „Northwoods Software Goxam“




15 pav. „Northwoods Software Goxam“ veiklos diagramos pavyzdys

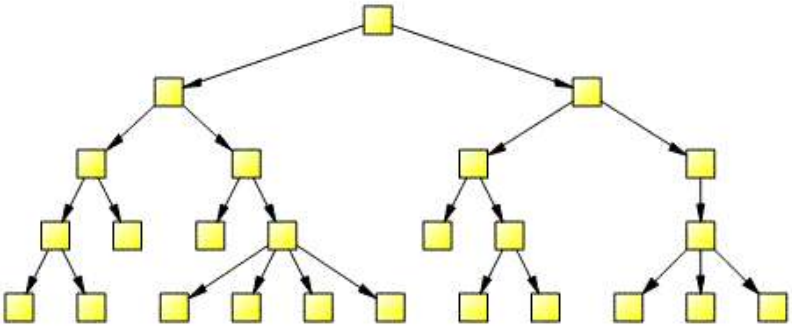
Tai „Microsoft Silverlight“ ir „Windows Presentation Foundation“ aplikacijų kūrimo biblioteka. Ji yra specialiai sukurta, kad pilnai išnaudotų šių platformų galimybes. Šios bibliotekos galimybės:

- Mazgų ir ryšių šablonai.
- Duomenų susiejimas.
- Animacija.
- Eksportas į XML.
- Įgyvendintas mąstelis.
- Z – indeksas.
- 180 pradinių figūrų.

Trūkumai:

- Biblioteka mokama.

#### 2.2.5.3. „Lassalle AddFlow for Silverlight“



16 pav. „Lassalle AddFlow for Silverlight“ diagramos pavyzdys

Tai yra analogiška biblioteka, kuri turi įgyvendintus hierarchinio, medžių, simetrinio išdėstymo algoritmus. Šios bibliotekos privalumai:

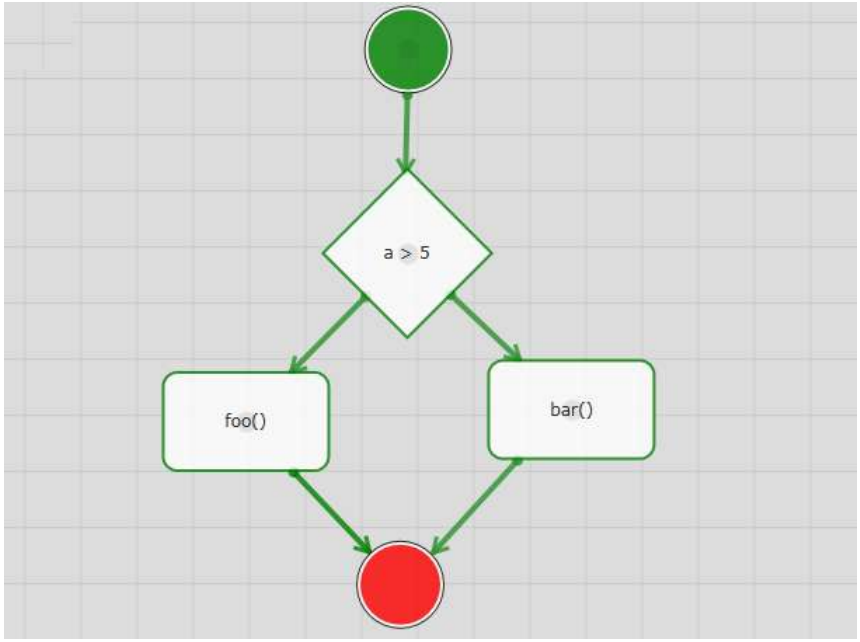
- Maža apimtis.
- Lengvas programavimas.
- Spartus veikimas.
- Automatinis išdėstymas.

Trūkumai:

- Mažas vizualumas (nėra pradinių figūrų).
- Nėra įgyvendintas diagramų saugojimas.

- Biblioteka mokama.

#### 2.2.5.4. „Sharedesigner“




17 pav. „Sharedesigner“ veiklos diagramos pavyzdys

Tai yra atviro kodo biblioteka, leidžianti įgyvendinti dauguma mokamų paketų funkcijų. Tačiau ši biblioteka yra pradinėje versijoje. Jos privalumai:

- Diagramos saugojimas XML formatu.
- Mazgų kopijavimas lygiavimas.
- Kontekstinis meniu.
- Įgyvendintas mąstelis.
- Biblioteka atviro kodo.

Trūkumas:

- Nėra dokumentacijos.

### 2.2.6. Posistemės tikslai

Iš ankstesnių pastraipų galime išskirti šiuos posistemės tikslus:

- Leisti vartotojui kurti metamodelį grafiškai.
    - Scenarijams naudoti lengvai suprantamą anotaciją.
    - Padidinti panaudojamumo lygį objektus atvaizduojant hierarchiškai.

- Leisti vartotojui įgyvendinti pagrindines logines operacijas (sąlygos sakiniai, ciklai).
- Transformuoti sukurtą modelį į metakalbą, iš kurios bus generuojamas programinis kodas.

### 2.2.7. Redaktoriaus problemos

#### 2.2.7.1. Objektų atvaizdavimas

Kuriant sistemą reikės nuspręsti kaip tinkamai atvaizduoti objektus ir ryšius tarp jų. Sumodeliuotas grafikas turi būti lengvai skaitomas ir neperkrautas. Taip pat reikia kuo mažiau prisirišti prie taikomosios srities ir nenaudoti atitinkamų ikonų. Kad išvengti šių problemų reikia:

- Naudoti tekstą, kur jis aprašo labiausiai[8].
- Simbolio forma turi apibūdinti veiklą[8].
- Naudoti grafiką, kur dėmesys turėtų būti skiriamas srauto duomenų ir kontrolė. Taip diagramai suteikiamas tvarkingas, glaustumas ir naudingumas[8].

#### 2.2.7.2. Objektų sujungimas

Programa turės surasti patogų būdą atvaizduoti ryšius tarp objektų. Jungiant objektus ryšiai negali susilieti. Taip pat sudėtinguose scenarijuose reikės neperkrauti diagramų. Patogiausia būtų naudoti modelio dekompoziciją. Tačiau reikia sugalvoti efektyvų būdą atskirti persipynusius scenarijus. Sprendimai gali būti:

- Atskirti smulkesnes modelio dalis pagal eilės tvarką. Pavyzdžiui pirmas trečdalis priklauso vienam fragmentui, o likusi dalis – kitam.
- Atskirti smulkesnes ir bendresnes modelio dalis pagal mąstelio dydį. Pavyzdžiui „planas, aukščiausiam lygį, matomas kaip viena esybė, gali būti įgyvendinta kaip kelių žemėlapių sujungimas žemesniame“[9] . Taip mąstelis turės atitinkamą abstrakcijos lygį.

#### 2.2.7.3. Palaikomumas

Sukurta sistema turi būti prieinama ir kuriama „Internet Explorer“ ir „Firefox“ naršyklėms. Taip bus užtikrinta, kad dirbti bus galima be papildomų programų įdiegimo. Šiam tikslui galime naudoti tris technologijas: „JavaScript“, „Adobe Flash“ ir „Microsoft Silverlight“. Tačiau „greitaveika, kurią suteikia flash technologija vektorių atvaizdavimui,

negalima naudojant Javascript“ (10). Panašias išvadas galima pasakyti palyginus „Silverlight“ su „JavaScript“.

#### 2.2.7.4. Saugumo problemos

Jeigu bus nutarta programos rezultatus automatiškai įvesti į sistemą, ar drausti viešai prieiti prie programos tai reikės užtikrinti vartotojų autentifikaciją.

- Slaptažodžių naudojimas. „Šis būdas nėra efektyvus, kai naudojamas vienas, nes „vartotojas nebandys arba neįsimins pakankamai sudėtingo slaptažodžio kad išvengtų žodyno-atakos“[10].
- Naudoti slaptažodžių sudėtingumo matuoklį. „Daug kompiuterių vartotojų neturi pakankamai žinių, kad galėtų pasirinkti stiprų slaptažodį, todėl jie turi naudoti matavimo priemonę patikrinti savo slaptažodžius“ (11). Toks matuoklis gali būti nesunkiai įdiegtas į sistemą.
- Žiniatinklio aplinka. Kadangi programos veikimas bus nukeltas į interneto puslapį, tai yra slaptažodžių nulaužimas taps žymiai lėtesniu procesu. „O jei svetainės kūrėjai yra apsukrūs, svetainė užblokuos paskyrą, jei buvo per daug neteisingų bandymų iš eilės“[12].
- Naudoti kelis autentifikacijos būdus vienu metu. „Mes galime padidinti patikimumo ir saugumo lygį autentifikavimo metu, sujungiant keletą būdų į vieną modelį“[13].

Neatsižvelgus į šiuos punktus sistema gali tapti pažeidžiama.

# 3. PROJEKTINĖ DALIS

## 3.1. Bendra sistema

### 3.1.1. Architektūros tikslai ir apribojimai

Kuriamos sistemos architektūros tikslai:

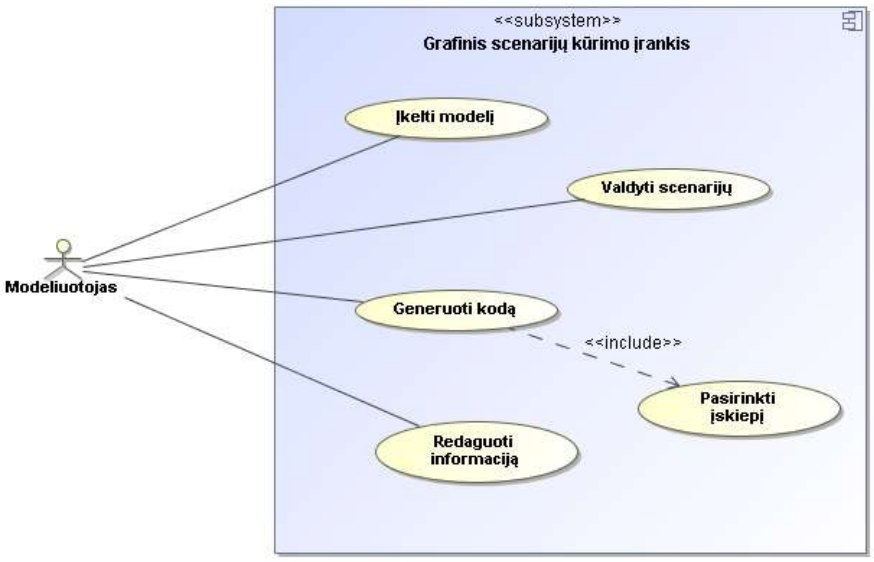
- Sistema bus galima naudotis visais kompiuteriais turinčiais interneto naršyklę ir prieigą prie interneto.
- Duomenimis paremta architektūra. Paduodame UML diagramas ir pagal tai modeliuojame scenarijus grafiniame redaktoriuje.
- Nepriklausoma taikymo sritis. Kiekviena taikymo sritis aprašoma kaip įskiepis 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 18 pav., o jų aprašymai lentelėse (3 lentelė – 7 lentelė).




18 pav. Grafinio scenarijų įrankio panaudos atvejų vaizdas

3 lentelė. Panaudos atvejo „Įkelti modelį“ informacija

| **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. |

4 lentelė. Panaudos atvejo “ Valdyti scenarijų ” informacija

| **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. |

5 lentelė. Panaudos atvejo „Generuoti kodą” informacija

| **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. |

6 lentelė. Panaudos atvejo „Pasirinkti įskiepį” informacija

| **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. |

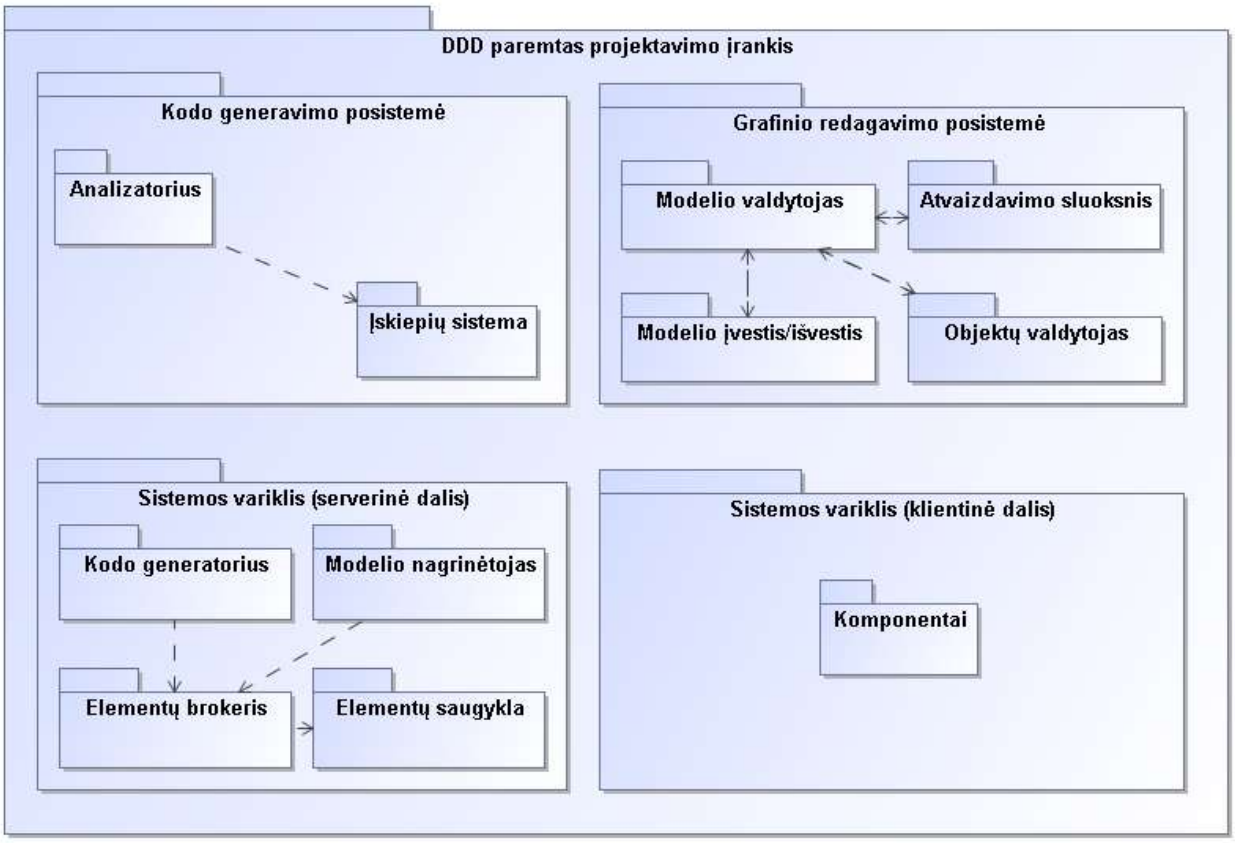
7 lentelė. Panaudos atvejis „Redaguoti informaciją”

| **Redaguoti informaciją** | |
|---|---|
| Aprašas: | Galimybė pakeisti sistemos scenarijaus modelį, neleidžiant pakeisti sistemos loginės veiklos. |
| Vartotojas/Aktorius: | Modeliuotojas. |
| Prieš-sąlyga: | Yra bent vienas scenarijus.<br>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 (19 pav):

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




19 pav. 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[14].

#### 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:

- Operacinė sistema
- Interneto naršyklė (su įjungtu „JavaScript“ rėžimu)
- Įdiegtas „Microsoft Silverlight“ karkasas

Minimalūs reikalavimai vartotojo techninei įrangai:

- 1Ghz procesorius
- Prijungtas prie interneto kompiuteris

Reikalavimai serverio programinei įrangai:

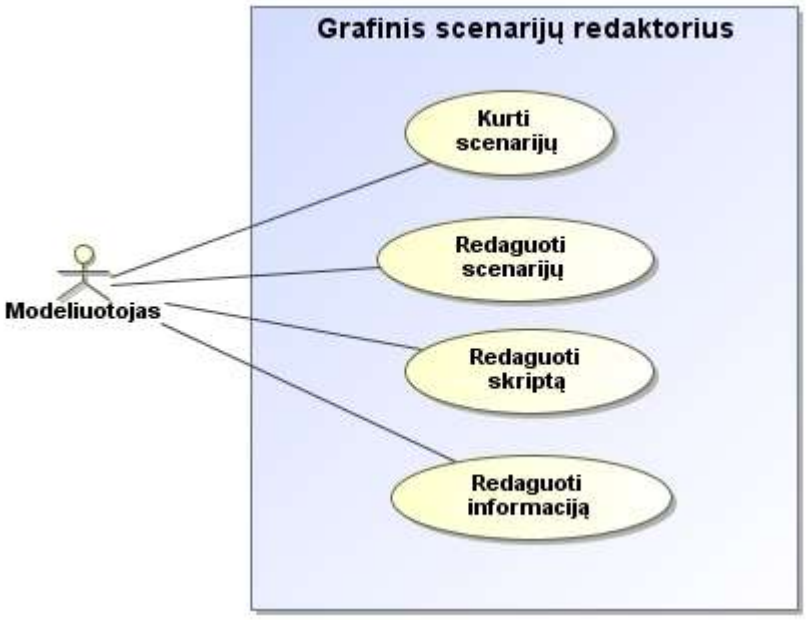
- Apache 1.3 ir naujesnis.
- PHP 5.0 ir naujesnis.
- Micorosft Internet Information Service 7.0 (ir aukštesnės versijos)
- Microsoft Windows Server 2003 (ir aukštesnės versijos)

Minimalūs reikalavimai serverio techninei įrangai:

- 1Ghz procesorius
- 128 MB operatyviosios atminties
- 100MB disko vietos

## 3.2. Grafinis scenarijų redaktorius

### 3.2.1. Grafinio scenarijų redaktoriaus posistemės panaudos atvejų diagrama




20 pav. Grafinio scenarijų redaktoriaus posistemės panaudos atvejų diagrama

### 3.2.2. Funkciniai reikalavimai

Grafiniame scenarijaus redaktoriuje suprojektuoti panaudos atvejai pateikiami (8 lentelė, 9 lentelė, 10 lentelė, 11 lentelė, 12 lentelė) lentelėse. Čia detalizuojami panaudos atvejų vartotojai, aprašymai ir vykdymo sąlygos.

8 lentelė. Panaudojimo atvejo „Įkelti duomenis apie sistemą“ išsami informacija

| PANAUDOJIMO ATVEJIS: Įkelti duomenis apie sistemą. | |
|---|---|
| Vartotojas/Aktorius | Modeliuotojas. |
| Aprašas | Tai pradinių duomenų įkėlimas į įrankį, naudojant sistemos UML specifikaciją. |
| Prieš sąlyga | Sistemos specifikacija yra parengta. |
| Sužadinimo sąlyga | Įrankio diegimas į sistemą. |
| Po-sąlyga: | Įrankiu galima pradėti naudotis. |

9 lentelė. Panaudojimo atvejo „Kurti scenarijų“ išsami informacija

| PANAUDOJIMO ATVEJIS: Kurti scenarijų. | |
|---|---|
| Vartotojas/Aktorius | Modeliuotojas. |
| Aprašas | Tai veiksmų, kuriuos atliks sistema, modeliavimas naudojantis grafine sąsaja. |
| Prieš sąlyga | Nėra. |
| Sužadinimo sąlyga | Sistemoje reikalingas naujas funkcionalumas. |
| Po-sąlyga: | Sukurtas modelis galės būti transformuotas į tarpinę kalbą. |

10 lentelė. Panaudojimo atvejo „Redaguoti scenarijų“ išsami informacija

| PANAUDOJIMO ATVEJIS: Redaguoti scenarijų. | |
|---|---|
| Vartotojas/Aktorius | Modeliuotojas. |
| Aprašas | Jau esamo scenarijaus redagavimas norint įgyvendinti naują ar pakeisti jau esamą funkcionalumą. |
| Prieš sąlygos | Yra bent vienas scenarijus. |
| Sužadinimo sąlyga | Sistemos veiklos taisyklės pasikeitė ir reikalingas atnaujinimas. |
| Po-sąlyga: | Naujas funkcionalumas atitinka modeliuotojo poreikius. |

11 lentelė. Panaudojimo atvejo „Redaguoti informaciją“ išsami informacija

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

12 lentelė. Panaudojimo atvejo „Redaguoti skriptą“ išsami informacija

| PANAUDOJIMO ATVEJIS: Redaguoti skriptą. | |
|---|---|
| Vartotojas/Aktorius | Modeliuotojas |
| Aprašas | Galimybė įvesti scenarijaus pakeitimus, kuriuos yra sunkiau (neįmanoma) atlikti per grafinį įrankį. |
| Prieš sąlyga | Yra bent vienas scenarijus. |
| | Skriptas yra sugeneruotas. |
| Sužadinimo sąlyga | Sistemoje reikalingas naujas funkcionalumas. |
| Po-sąlyga: | Įvesti skripto pakeitimai tenkina modeliuotojo lūkesčius. |

### 3.2.3. Nefunkciniai reikalavimai

Nusako sistemos savybes, kuriomis ji turi pasižymėti. Tai kokybinės funkciniuose reikalavimuose numatytų funkcijų vykdymo charakteristikos

#### 3.2.3.1. Reikalavimai sistemos išvaizdai

Bendri reikalavimai vartotojo sąsajai:

- lengvai skaitoma sąsaja.
- paprastas (nesudėtingas) panaudojimas.
- prieinamumas, kad vartotojas galėtų dirbti nuotoliniu būdu.
- atitinkantis kitus vartotojo naudojamus produktus (pavyzdžiui, „Microsoft Visio“, „IBM Rational Rose“, „MagicDraw“).
- neįkyri sąsaja (pavyzdžiui, naudojanti švelnias akiai spalvas, nereikalaujanti pastoviai ką nors kelis kartus patvirtinti).
- išvaizda, imituojanti realią aplinką.
- sąveikaujanti (aktyviai “bendraujanti”) sąsaja.

#### 3.2.3.2. Reikalavimai panaudojamumui

Panaudojimo paprastumas (lengvumas), kuris gali būti vertinamas konkrečiais kriterijais.

- paprastas naudotis (įprasti žymėjimai ar pan.);
- nacionalinės kalbos panaudojimas;
- veiklos našumo prieaugis dėl sistemos diegimo.

#### 3.2.3.3. Reikalavimai sistemos priežiūrai

Sistemą planuojama kurti su galimybe naudoti ne tik patogiau.lt sistemoje, bet ir kitose. Šios sistemos gali skirtis veikimu, architektūra ar net operacine platforma. Taigi reikia naudoti įvairius būdus produkto pernašamumui padidinti, taip taupant lėšas ir laiką.

#### 3.2.3.4. Reikalavimai saugumui

Jeigu bus nutarta programos rezultatus automatiškai įvesti į sistemą ar drausti viešai prieiti prie programos, tai reikės užtikrinti vartotojų autentifikaciją. Kitu atveju apsaugos priemonės bus įgyvendintos serverio pusėje.

- Slaptažodžių naudojimas. Šis būdas nėra efektyvus, kai naudojamas vienas, nes slaptažodžius nulaužiančios programos dabar gali rasti slaptažodžius, kuriuos galima tikėtis atsiminti.
- Naudoti slaptažodžių sudėtingumo matuoklį. Daug kompiuterių vartotojų neturi pakankamai žinių, kad galėtų pasirinkti stiprų slaptažodį, todėl jie turi naudoti matavimo priemonę patikrinti savo slaptažodžius. Toks matuoklis gali būti nesunkiai įdiegtas į sistemą.
- Žiniatinklio aplinka. Kadangi programos veikimas bus nukeltas į interneto puslapį, tai yra slaptažodžių nulaužimas taps žymiai lėtesniu procesu. O jei svetainės kūrėjai yra apsukrūs, svetainė užblokuos paskyrą, jei buvo per daug neteisingų bandymų iš eilės.
- Naudoti kelis autentifikacijos būdus vienu metu. Mes galime padidinti patikimumo ir saugumo lygį autentifikavime, sujungiant keletą būdų į vieną sistemą.

#### 3.2.3.5. Kultūriniai-politiniai reikalavimai

Apribojimai, susiję su politiniais bei kultūriniais sistemos veikimo terpės ypatumais. Jie ypač aktualūs platinant sistemą įvairiose šalyse.

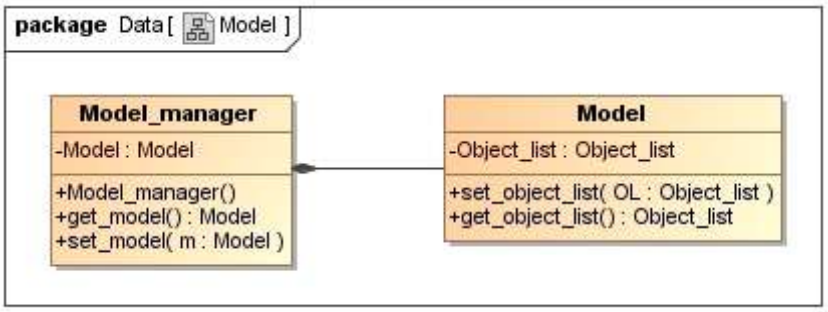
- politinis korektiškumas - sistemoje negalima naudoti ką nors įžeidžiančių terminų ar iliustracijų;
- kalbos universalumas (galimybė naudoti įvairių šalių rašmenis).

### 3.2.4. Architektūros specifikacija

#### 3.2.4.1. Sistemos statinis vaizdas

Šiame skyrelyje pateikiamos paketų klasių diagramos ir trumpi paketų aprašymai. Būsenų, veiklos, sekų diagramas ir klasių aprašymus galima rasti prisegtuose prieduose.

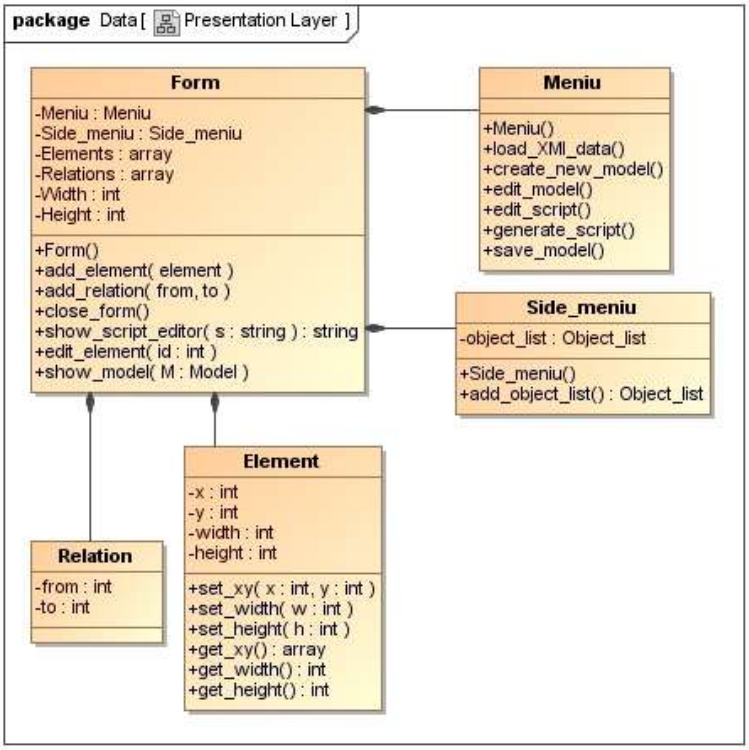
**„Model manager“ paketas**




21 pav. Model manager paketo klasių diagrama

„Model manager“ (21 pav) paketas atsakingas už modelio sąveiką su kitomis posistemės dalimis. Tai yra pagrindinis Grafinės redagavimo posistemės paketas.

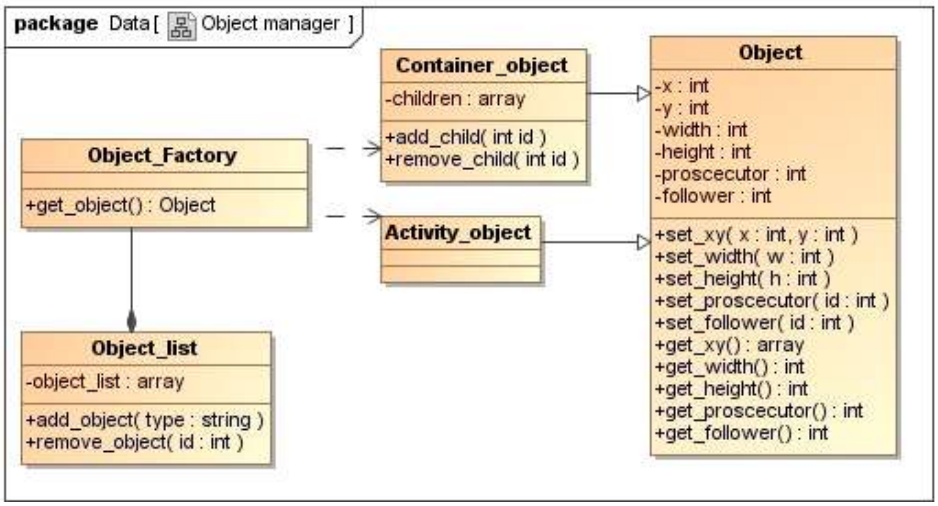
**„Presentation layer“ paketas**




22 pav. Presentation layer paketo klasių diagrama

„Presentation layer“ (22 pav) paketas atsakingas už grafinėje sąsajoje atvaizduojamus objektus ir ryšius. Taip pat šis paketas inicijuoja paketų veiklą.

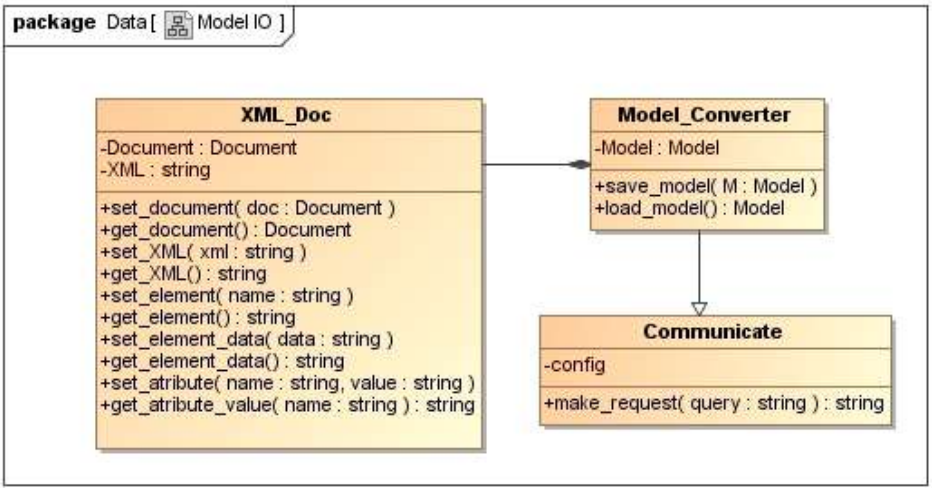
**„Object manager“ paketas**




23 pav. Object manager paketo klasių diagrama

„Object manager“ (23 pav) atsakingas už objektų kūrimą modelyje. Šis paketas paremtas „factory method“ šablonu, kuris leidžia praplėsti objekto tipus naudojamus įrankyje.

**„Model IO“ paketas**

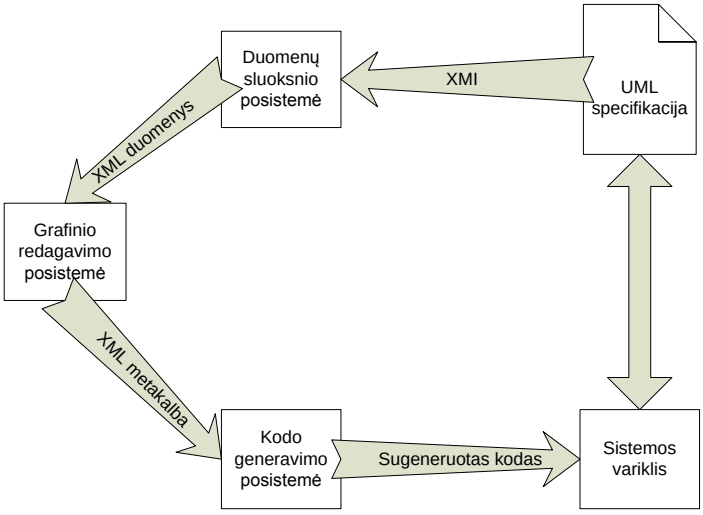



24 pav. Model IO paketo klasių diagrama

„Model IO“ (24 pav) - atsakingas už pradinės sistemos specifikacijos konvertavimą į posistemės objektus. Taip pat šis paketas atsakingas už sukurto scenarijaus saugojimą duomenų bazėje.

### 3.2.4.2. Bendravimas su kitomis sistemos dalimis

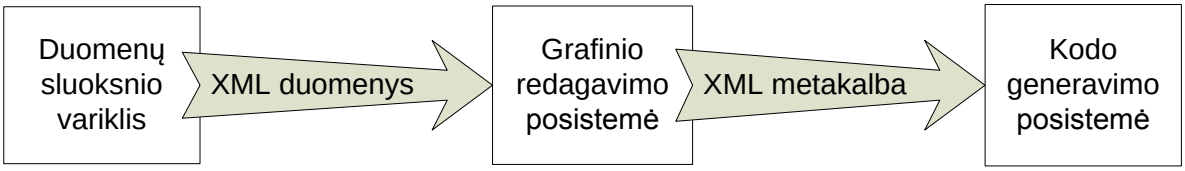
Grafinio scenarijų redagavimo posistemė yra smarkiai surišta su kitomis sistemos dalimis. Visą duomenų mainų ciklą galima matyti (25 pav) iliustracijoje.




25 pav. Scenarijų generavimo įrankio duomenų mainų ciklas

Kaip matome grafinis redaktorius naudoja duomenų sluoksnio posistemėje apdorotus XML duomenis iš UML specifikacijos. Tai išrinktos sistemos variklio klasės, kurias galima panaudoti objektų kūrimui. Klasių struktūra taip pat perduodama, ji įtraukia kintamuosius, funkcijas ir įvykius.

Sumodeliavus naujas sistemos veiklas, modelis yra transformuojamas į tarpinį skriptą pagal metakalbos taisykles (aprašytas 3.2.4.3 skyriuje). Šis skriptas persiunčiamas kodo generavimo posistemei, kur bus sugeneruotas galutinis programinis kodas.




26 pav. Grafinio redaktoriaus posistemės bendravimas su kitomis sistemos dalimis

### 3.2.4.3. Metakalbos sintaksė

Išanalizavus bendrus posistemių bendravimo principus buvo nuspręsta scenarijaus metakalbą apibrėžti remiantis XML formatu.

Scenarijaus metakalba – tam tikromis taisyklėmis apibrėžta žymių XML kalba. Žymuo scenarijuje apibrėžia atskirą komponentą, ar jo konfigūraciją. Žymių sintaksė ir detalus atributų aprašymas pateiktas (13 lentelė) lentelėje.

Sukurtų objektų hierarchija skripte atvaizduojama kaip hierarchinės XML žymos. Tai nurodo, kad galutiniame kode vidinės žymos objektas bus transformuotas kaip tėvinio objekto vaikas.

Veiklos diagrama scenarijaus skripte aprašoma kaip grafas. Viena viršūnė veda į kitą viršūnę. Pradžios viršūnė identifikuojama „Start“ požymiu, o pabaigos – „End“.

#### 3.2.4.4. Tarpinės metakalbos aprašymas

13 lentelė. Tarpinės metakalbos aprašymas

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

Pagal šią sintaksę sugeneruoto skripto pavyzdys matomas (27 pav.) iliustracijoje.

```
<?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>
```

27 pav. Tarpinės metakalbos skripto pavyzdys.

# 4. TYRIMO DALIS

Šiame skyriuje aprašysime grafinio scenarijų redaktoriaus specifikacijos atitikimą ir kokybės analizę. Bus aptartos probleminės redaktoriaus sritys ir pateikti programos tobulinimo pasiūlymai.

## 4.1. Kokybės analizė

### 4.1.1. Specifikacijos atitikimas

Sukurtas grafinis scenarijų redaktorius atlieka visus specifikacijoje aprašytus veiksmus, tačiau liko neįgyvendintos vartotojų grupių veiksmų apribojimai. Esamoje versijoje visi programos aktoriai gali atlikti visus veiksmus.

## 4.2. Iškilusios problemos

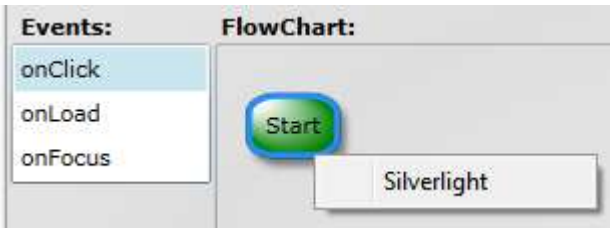
### 4.2.1. Bendros problemos

#### 4.2.1.1. Elementų paieška

Norint surasti konkretų objektą ar veiklos elementą modelyje yra sudėtinga. O jeigu modelį kūrė ne tas pats vartotojas tai jam reikės peržiūrėti visus objektus, kad surastų reikiamą.

#### 4.2.1.2. Kontekstinis meniu

„Silverlight 2.0“ platformoje, su kuria buvo kurtas scenarijų redaktorius nepalaikė kontekstinio meniu iškvietimo dešiniu pelės mygtuko paspaudimu. Bandant paspausti dešinį pelės mygtuką ant tokių aplikacijų parodomas naršyklės kontekstinis meniu (28 pav). Klaviatūros naudojimas šiam veiksmui atlikti yra neįprastas ir nepatogus.




28 pav. „Silverlight 2.0“ kontekstinio meniu problema

#### 4.2.1.3. Interneto trukdžiai

Dabartinėje programos versijoje nėra įvertinta interneto atsijungimo galimybė. Vartotojai nežinos, kad jų darbas ne tik nebus nusiųstas į tarnybinę stotį, bet ir nebus išsaugotas. Tokiu atveju vartotojas taps nepatenkintas ir gali nustoti naudotis produktu.

### 4.2.2. Objektų kūrimo langas

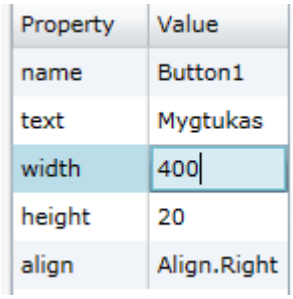
#### 4.2.2.1. Objektų kūrimas

Šioje programos versijoje nauji objektai kuriami šoninio meniu mygtukų paspaudimais. Sukurti objektai įkeliami į bendrą langą arba į sufokusuotą objektą. Šiuo atžvilgiu vartotojui reikia iš pradžių sukurti objektą, o antru žingsniu jį perkelti į norimą vietą. Naudojant vieną žingsnį šį veiksmą atlikti būtų patogiau.


29 pav. Objektų kūrimo meniu dabartinėje versijoje

#### 4.2.2.2. Objektų redagavimas

Objektų redagavimas taip pat nėra pakankamai patogus. Įkėlus naują objektą, jo atvaizdavimą galima keisti tik keičiant reikšmes objekto parametrų skydelyje. Ši problema atsiranda tik tada, kai naudojami objektai, kurie bus atvaizduojami suprojektuotoje sistemoje. Jeigu objektai nebus vaizduojami jų išvaizdos redagavimas nėra naudojamas.




30 pav. Objekto parametrų keitimas dabartinėje versijoje

#### 4.2.2.3. Objektų išvaizda

Šiuo metu objektų išvaizda yra standartizuota. Yra dvi objektų rūšys, objektai, kuriuose galima įkelti kitus objektus, ir tie, į kuriuos negalima. Visi vienos objektų klasės objektai atrodo taip pat. Juos atskirti galima tik iš užrašyto pavadinimo, todėl vartotojams yra sudėtinga orientuotis tarp jų.


31 pav. Objektų išvaizda dabartinėje versijoje

## 4.3. Siūlymai tobulinti programą

Šiame skyrelyje apžvelgsime iškeltų problemų sprendimo būdus ir papildomus pasiūlymus kaip pagerinti sukurtą redaktorių. Pradžioje apžvelgiami tobulinimai pritaikomi visam scenarijų redaktoriui, o toliau – tobulinimai pritaikomi objektų kūrimui.

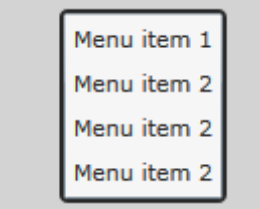
### 4.3.1. Siūlymai visai programai

#### 4.3.1.1. Elementų paieška

Naują funkciją galima įgyvendinti įkomponavus paieškos laukelį objektų ir scenarijų kūrimo languose. Ši funkcija galėtų ieškoti tarp objektų pavadinimų, jų atributų ir funkcijų. Rezultatai būtų pateikiami naujame lange, kur pasirinkus rezultatą surastas elementas bus parodomas.

#### 4.3.1.2. Kontekstinis meniu

Kontekstinio meniu problemos sprendimui siūloma grafinio redaktoriaus posistemę perrašyti „Microsoft Silverlight 4“ platformai. Šioje karkaso versijoje jau yra palaikomas dešinio pelės mygtuko paspaudimai. Pavyzdinį komponentą matome (32 pav) iliustracijoje.




32 pav. Siūlomas kontekstinio meniu pavyzdys

#### 4.3.1.3. Apsauga nuo interneto sutrikimų

Ši apsauga turėtų būti įgyvendinta kaip iššokantis pranešimo langas, kuris informuotų apie interneto sutrikimą. Kai internetas dingsta diagramos būtų saugomos į lokalų failą, o vartotojui prisijungus failas persiunčiamas į serverį ir išsaugojamas.

#### 4.3.1.4. Vartotojų teisės

Vartotojų teisės gali būti įgyvendintos tiek pačiame redaktoriuje, tiek svetainėje, kur bus patalpintas redaktorius. Tačiau apribojimus įgyvendinus svetainėje, reikėtų sugeneruoti scenarijų redaktorių kiekvienam aktoriui.

#### 4.3.1.5. Spartieji klavišai

Tyrimai rodo, kad sparčiųjų klavišų[20] naudojimas patyrusiems vartotojams yra daug greitesnis nei meniu punktų ar piktogramų naudojimas[16]. Todėl dažnai naudojamų veiksmų susiejimas su sparčiaisiais klavišais yra didelis privalumas.

### 4.3.2. Siūlymai objektų kūrimo langui

#### 4.3.2.1. Objektų kūrimas

Objektų kūrimas bus atliekamas pernešimo būdų. Iš šoninio meniu lango pernešant norimą objektą į modelį. Jo pozicija iškarto bus teisinga, ir modelis atsinaujins.

#### 4.3.2.2. Objektų redagavimas

Redaguoti objektus galima atlikti ne tik keičiant parametrus, bet ir keičiant jų atributus pelės pagalba. Tam reikalingas priskyrimas prie pločio, aukščio ir kitų parametrų. Šie priskyrimai turi būti aprašyti konfigūraciniuose failuose arba sistemos UML modelyje.

#### 4.3.2.3. Objektų išvaizda

Pagerinti objektų išvaizdą galima naudojant paveiksliuką kiekvienam objektui. Tačiau šiuos priskyrimus taip pat reikės aprašyti konfigūraciniuose failuose. Jei priskyrimams naudosime skirtingas spalvas šios problemos išvengsime.


33 pav. Objektų vaizdavimo pavyzdžiai. Teksto įvedimo laukas, mygtukas, akutė, žymimasis langelis

[20] Spatieji klavišai (*angl. shortcut key*) – Klavišai arba klavišų deriniai, tam tikriems programos valdymo veiksmams sukelti.[15]

# 5. PROGRAMAVIMO IR MODELIAVIMO GREIČIŲ IR KLAIDŲ SKAIČIAUS PALYGINIMAS

## 5.1. Užduotys

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

### 5.1.1. 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 „Results“ langą.*

### 5.1.2. Antroji užduotis (vidutinis sudėtingumas)

*Sumodeliuokite meniu komponentą, kuriame rodomi trys mygtukai. Nuspaudus pirmąjį mygtuką pereinama į „Home“ langą. Nuspaudus antrąjį pereinama į „Search“ langą. Trečias mygtukas veikia priklausomai nuo prisijungimo būsenos. Jei vartotojas prisijungęs, jame rodomas tekstas„Logout“, nuspaudus jį vartotojas atsijungia nuo serverio ir įjungiamas „Home“ langas. Kitu atveju rodomas tekstas “Login“, o jį paspaudus atsidaro prisijungimo langas „Login“.*

### 5.1.3. Trečioji užduotis (aukštas sudėtingumas)


34 pav. Trečios užduoties pavyzdys

*Sumodeliuokite prisijungimo langą, kuris atrodytų panašiai į 34 pav. „Dialog“ komponento viduje yra logotipas, klaidos pranešimo laukas, vartotojo vardo žymė, vartotojo vardo įvedimo laukas, slaptažodžio žymė, slaptažodžio įvedimo laukas ir pateikimo mygtukas.*

- *Logotipo paveikslui nurodyti „logo.jpg“ failą.*

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

## 5.2. Nepatyrusio programuotojo eksperimento aprašymas

Sukurto projektavimo įrankio įvertinimui buvo atliktas praktinis bandymas. Jo metu komponentams buvo sukurti grafiniai modeliai ir įgyvendinamas 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, projektų sudėtingumui augant. Taip pat buvo skaičiuojamos padarytos klaidos. Klaida buvo įskaitoma, kai įvykdavo klaidingas kompiliavimas (sintaksės klaidos) arba programa klaidingai veikė (loginės klaidos).

### 5.2.1. 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 kartus.

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

Pagreitėjimo rezultatai atvaizduoti (14 lentelė) lentelėje.

14 lentelė. Nepatyrusio programuotojo pagreitėjimas modeliuojant grafiškai

| **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** |

Įvertinus klaidų statistiką buvo sudaryta apibendrinamoji lentelė(15 lentelė).

15 lentelė. Nepatyrusio programuotojo klaidų skaičiaus sumažėjimas

| **Realizavimo tipas** | **Sudėtingumas** | | |
|---|---|---|---|
| | Žemas | Vidutinis | Aukštas |
| Modeliavimo laikas (min.) | 5 | 9 | 15 |
| Programavimas laikas (min.) | 56 | 83 | 206 |
| **Pagerėjimas (%)** | **1120** | **922,22** | **1373,33** |

### 5.2.2. Išvados

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

## 5.3. 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.

### 5.3.1. Rezultatai

Patyrusio programuotojo rezultatai pateikiami lentelėje (16 lentelė).

16 lentelė. Patyrusio programuotojo pagreitėjimas modeliuojant grafiškai

| **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** |

Patyrusio programuotojo klaidų tyrimo rezultatai pateikti lentelėje().

| **Realizavimo tipas** | **Sudėtingumas** | | |
|---|---|---|---|
| | Žemas | Vidutinis | Aukštas |
| Modeliavimo klaidos | 4 | 6 | 8 |
| Programavimo klaidos | 10 | 16 | 37 |
| **Pagerėjimas(%)** | **250** | **266,6667** | **462,5** |

### 5.3.2. Išvados

Rezultatai parodo, kad patyręs programuotojas, realizuodamas nesudėtingą užduotį, ją įgyvendino greičiau nei modeliuojant, tačiau kai užduočių sudėtingumas kilo modeliavimo pagreitėjimas sparčiai didėjo. Todėl galime pasakyti, kad sudėtingų sistemų modeliavimas yra efektyvesnis už programavimą. Kaip ir nepatyrusio programuotojo atveju matome panašumų tarp šių eksperimentų. Todėl galime daryti išvadą, kad greičiau grafiškai modeliuoti sistemas galima dėl to, kad padaroma mažiau klaidų, tačiau ne tik dėl to.

# 6. IŠVADOS

Magistro tezių darbo metu buvo ištirtos DDD ir MDA metodologijos ir pabrėžtas pagrindinis skirtumas tarp jų:

- DDD metodologija skirta realizuoti pakeitimus apibrėžtos sistemos tobulinimui
- Modeliais paremta architektūra yra naudojama sudaryti pilną programų sistemos modelį ir realizuoti programą pasirinktoje architektūroje.

Analizės dalyje apžvelgėme egzistuojančius įrankius paremtus šiomis metodologijomis, iš kurių patogiausias efektyviausias yra „Arcstyler 4,0“. Taip pat buvo aptartos modeliavimo priemonės, iš kurių išsirinktas „Northwoods Software Goxam“ produktas.

Projektinėje dalyje pateikiama suprojektuota grafinė modelių redagavimo posistemė. Ši posistemė realizuota „Silverlight“ technologija, o su kitomis sistemos dalimis bendrauja žiniatinklio paslaugomis.

Tiriamojoje dalyje aprašytos po realizacijos iškilusios problemos. Šioje dalyje atskleistos naujos „Silverlight 4“ galimybės ir pasiūlyta svarbus paieškos patobulinimas

Eksperimentinėje dalyje pateikti rezultatai patvirtina analizės dalyje suformuluotą hipotezę, kad programų sistemų modeliavimas grafiškai yra efektingesnis nei įprastas programavimas. Nepatyrusių programuotojų realizacijos pagreitėjimas, kartais viršija net 7 kartus. Tuo tarpu patyrusių programuotojų pagreitėjimas pastoviai kyla, didėjant sudėtingumui. Taip pat iš eksperimento išaiškėjo, kad klaidų skaičius atvirkščiai proporcingas realizavimo greičiui. Taigi viena iš pagreitėjimo iš priežasčių yra klaidų skaičiaus mažėjimas.

# LITERATŪRA

1. **K. Wilson**. Data-Driven Design [Žiūrėta 2010 05 20], prieiga internete < *http://www.gamearchitect.net/Articles/DataDrivenDesign.html*>
2. **Danc**. Managing game design risk: Part II - Data Driven Development [Žiūrėta 2010 05 20], prieiga internete <*http://lostgarden.com/2006/04/managing-game-design-risk-part-ii-data.html*>
3. **M. Asadi, M. Ravakhah, R. Ramsin**. An MDA-based System Development Lifecycle [Žiūrėta 2010 05 20], prieiga internete <*http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4530584*>
4. **Š. Packevičius, V. Eidukynaitė, A. Ušaniov**. MDA case įrankių analizė [Žiūrėta 2010 05 20], prieiga internete <*https://mms.mruni.lt/DownloadFile.aspx?FileID=77*>
5. **K. Wang**. Study of Secure Complicated Information System Architecture Model [Žiūrėta 2010 05 20], prieiga internete <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4125889&isnumber=4125782>
6. **R. Andreoli**. Interactive 3D Environments by using videogame engines [Žiūrėta 2010 05 20], prieiga internete <*http://www.unibas.it/utenti/erra/Papers/iv05.pdf*>
7. **S. J. Mellor**. Executable UML [Žiūrėta 2010 05 20], prieiga internete <*http://www.techonline.com/article/pdf/showPDF.jhtml?id=1931036231*>
8. **L. Starr**. SCRALL - Starr's Consise Relational Action Language [Žiūrėta 2010 05 20], prieiga internete <*http://www.modelint.com/downloads/mint.scrall.tn.1.pdf*>
9. **A. Voisard, H. Schweppe**. Abstraction and decomposition in interoperable GIS [Žiūrėta 2010 05 20], prieiga internete <*http://www.informaworld.com/smpp/content~content=a713811419~db=all*>
10. **B. Schneier**. Secrets and Lies: Digital Security in a Networked World (Hardcover) [Žiūrėta 2010 05 20], prieiga internete <*http://www.amazon.co.uk/Secrets-Lies-Digital-Security-Networked/dp/0471253111*>
11. **O. Salem**. Intelligent System to Measure the Strength of Authentication [Žiūrėta 2010 05 20], prieiga internete <*http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4530020*>

12. **B. Schneier**. Customers, Passwords, and Web Sites [Žiūrėta 2010 05 20], prieiga internete <*http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01324609*>
13. **C. Mallow**. Authentication Methods and Techniques [Žiūrėta 2010 05 20], prieiga internete <*http://www.giac.org/resources/whitepaper/access/2.pdf*>
14. **S. J. Metsker**. Design Patterns in C# [Žiūrėta 2010 05 20]
15. **V. Dagienė, G. Grigas, T. Jevsikova**. Enciklopedinis kompiuterijos žodynas [Žiūrėta 2010 05 20], prieiga internete <*http://www.likit.lt/en-lt/angl.html*>
16. **D. M. Lane, H. Albert Napier, S. Camille Peres, A. Sándor**. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts [Žiūrėta 2010 05 20], prieiga internete <*http://www.ruf.rice.edu/~lane/papers/hidden_costs.pdf*>

# TERMINŲ IR SANTRUMPŲ ŽODYNAS

(*angl. Java 2Enterprise Edition*) – standartinė daugialyčių programų kūrimo Java kalba platforma, 20

CIM (*angl. Computation Independent Model*) – nuo skaičiavimų nepriklausomas modelis, 14

DDD (*angl. Data Driven Design*) – duomenimis paremta architektūra, 13

Hibernate – Java objektų saugykla, 20

HTML (*angl. Hyper text Markup Language*) – tai kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete, 20

Įskiepis (*angl. Cartrige*) – lengvai pakeičiama sistemos dalis, transformacijoms aprašyti, 21

MDA (*angl. Model Driven Architecture*) – modeliais paremta architektūra, 13

OMG (*angl. Object Management Group)* – tarptautinė programų kūrėjų grupė, ruošianti MDA standartus, 14

PIM (*angl. Platform Independent Model*) – nuo platformos nepriklausomas modelis, 14

PSM (*angl. Platform Specific Model*) – nuo platformos priklausomas modelis, 15

Spatieji klavišai (*angl. shortcut key*) – Klavišai arba klavišų deriniai, tam tikriems programos valdymo veiksmams sukelti.[13], 55

Spring – atviro kodo karkasas, skirtas kurti .NET ir Java aplikacijoms, 20

Struts – atviro kodo karkasas, skirtas kurti internetinėms aplikacijoms, 20

Terminalas (*angl. Console*) – vartotojo sąsaja, skirta tekstinėm komandoms įvesti, 21

UML (*angl. Unified Modeling Language*) – Vieninga modeliavimo kalba, 16

Visustin v5 Flow chart generator, 81

WebServices – žiniatinklio paslauga, 20

XMI (*angl. XML Metadata Interchange*) – XML formatas skirtas aprašyti UML diagramas, 21

XML (*angl. eXtensible Markup Language*) – praplečiama žymėjimo kalba, 17

XP (*angl. eXtreme Programming*) – ekstremalus programavimas, 15

# 7. PRIEDAI

## 7.1. Grafinio scenarijų redaktoriaus detali architektūra

### 7.1.1. 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). Aprašomos 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.

### 7.1.2. 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

### 7.1.3. Architektūros tikslai ir apribojimai

Kuriamos sistemos architektūros tikslai:

- Sistema bus galima naudotis visais kompiuteriais turinčiais interneto naršyklę ir

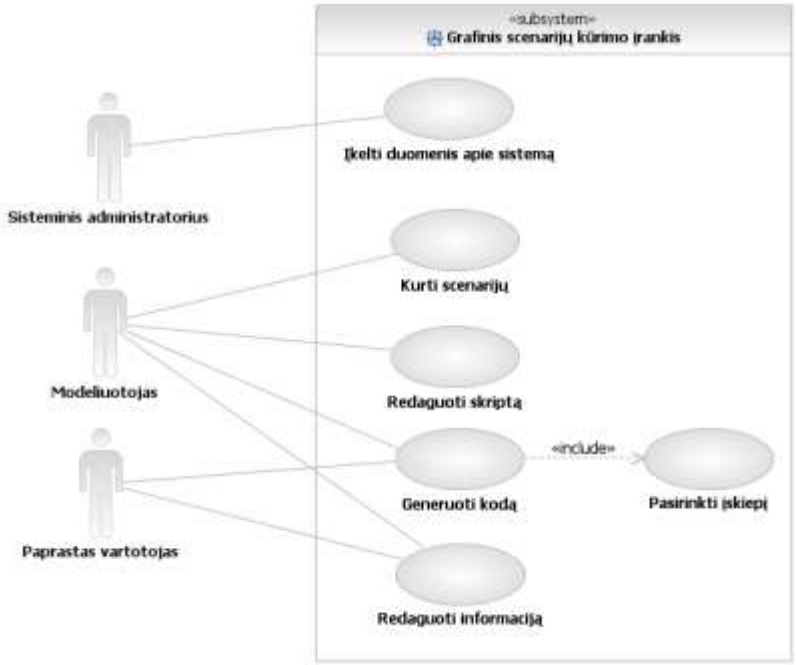
prieigą prie interneto.

- 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.

### 7.1.4. Panaudojimo atvejų vaizdas




35 pav. Panaudos atvejų vaizdas

17 lentelė. Panaudojimo atvejis „Įkelti duomenis apie sistemą“

| **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. |

18 lentelė. Panaudojimo atvejis „Kurti scenarijų“

| **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. |

19 lentelė. Panaudojimo atvejis „Redaguoti skriptą“

| **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.<br>Skriptas yra sugeneruotas. |
| Sužadinimo sąlyga: | Sistemoje reikalingas naujas funkcionalumas. |
| Po-sąlyga: | Įvesti skripto pakeitimai tenkina |

modeliuotojo lūkesčius.

20 lentelė. Panaudojimo atvejis „Generuoti kodą“

| **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. |

21 lentelė. Panaudojimo atvejis „Pasirinkti įskiepį“

| **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. |

22 lentelė. Panaudojimo atvejis „Redaguoti informaciją“

| **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.<br>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. |

### 7.1.5. Sistemos statinis vaizdas

#### 7.1.5.1. Apžvalga

Sistema suskirstyta į šiuos paketus:

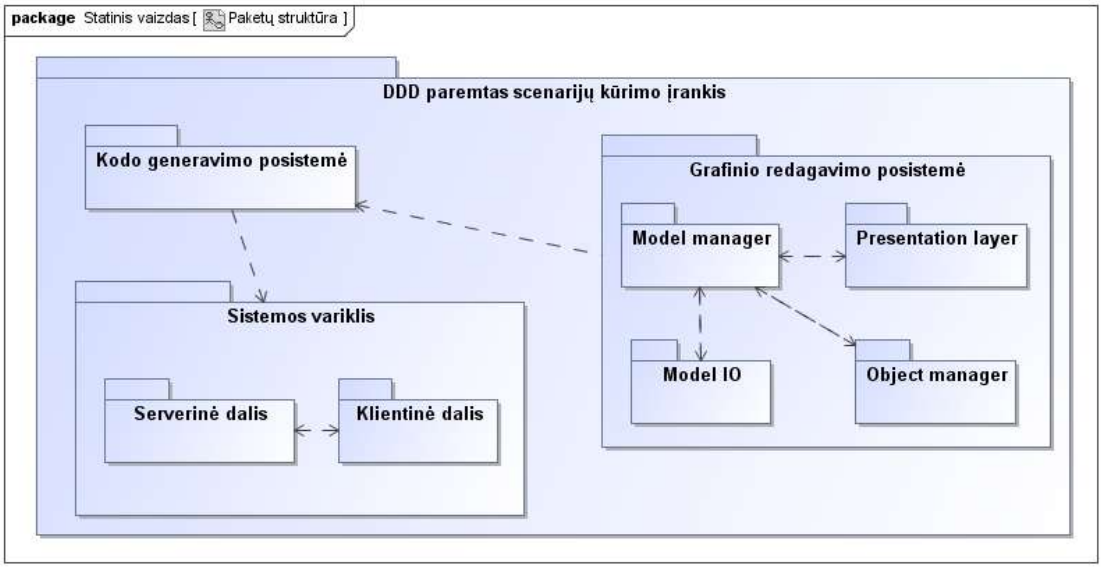
- Grafinio redagavimo posistemė
- Kodo generavimo posistemė
- Sistemos variklis

Grafinio redagavimo posistemė skirstoma į smulkesnius paketus

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

Sistemos variklis skirstomas į smulkesnius paketus:

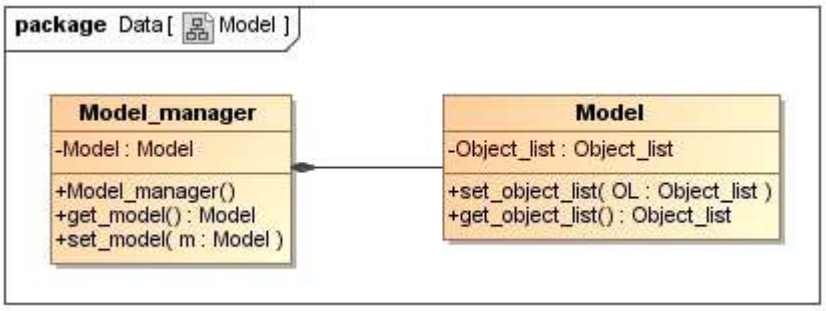
- Serverinė dalis
- Klientinė dalis




36 pav. Sistemos paketų diagrama

### 7.1.5.2. Paketų detalizavimas

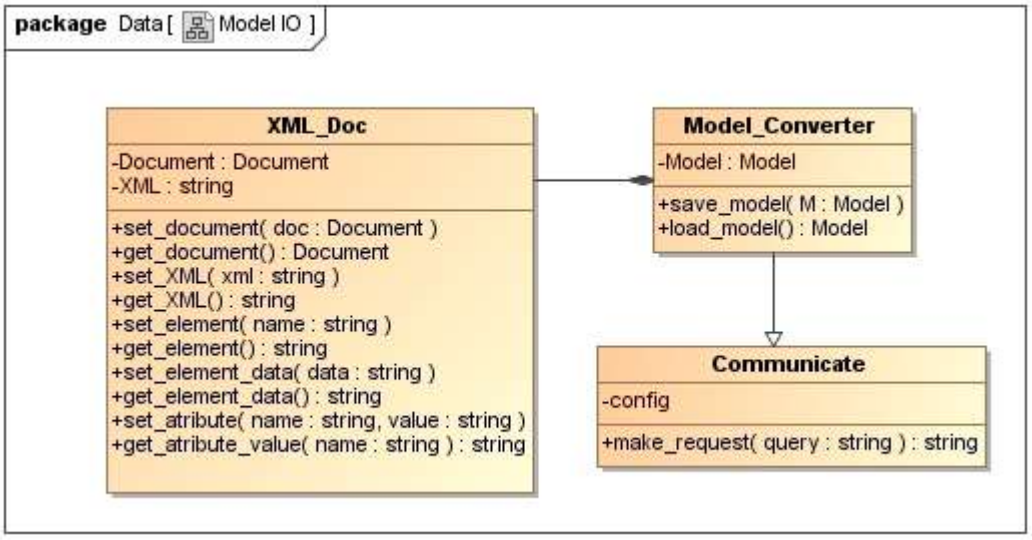
**„Model manager“ paketas**




37 pav. Model manager paketo klasių diagrama

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

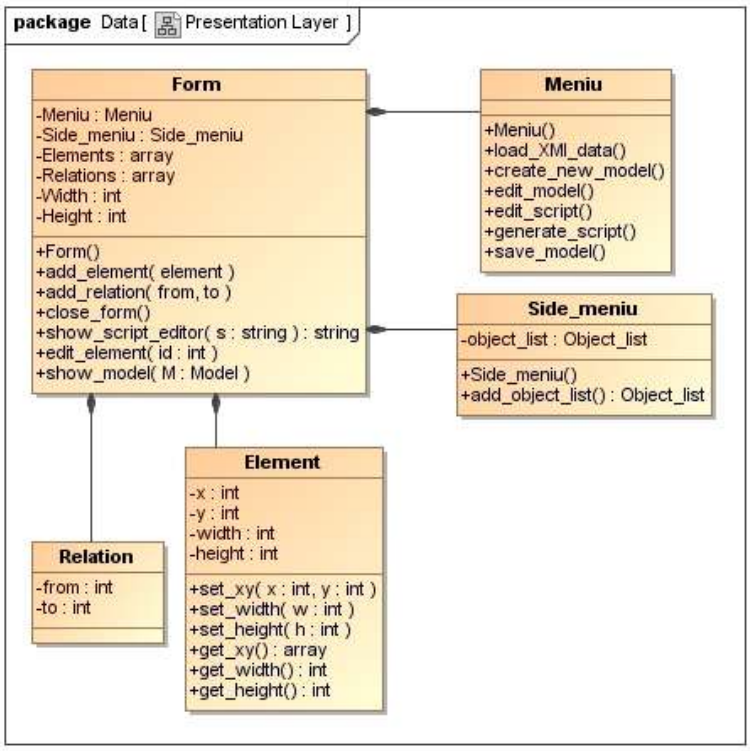
**„Model IO“ paketas**




38 pav. 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.

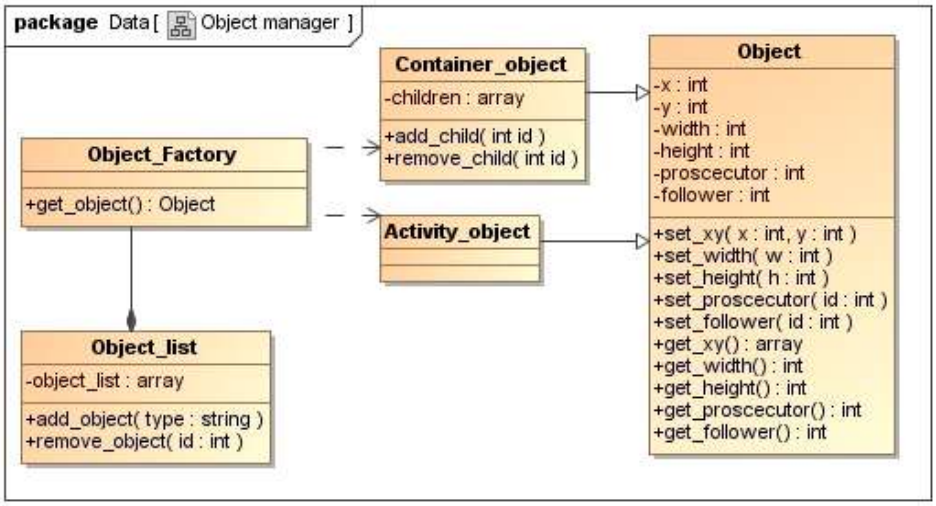
**„Presentation layer“ paketas**




39 pav. Presentation layer paketo klasių diagrama

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

### „Object manager“ paketas



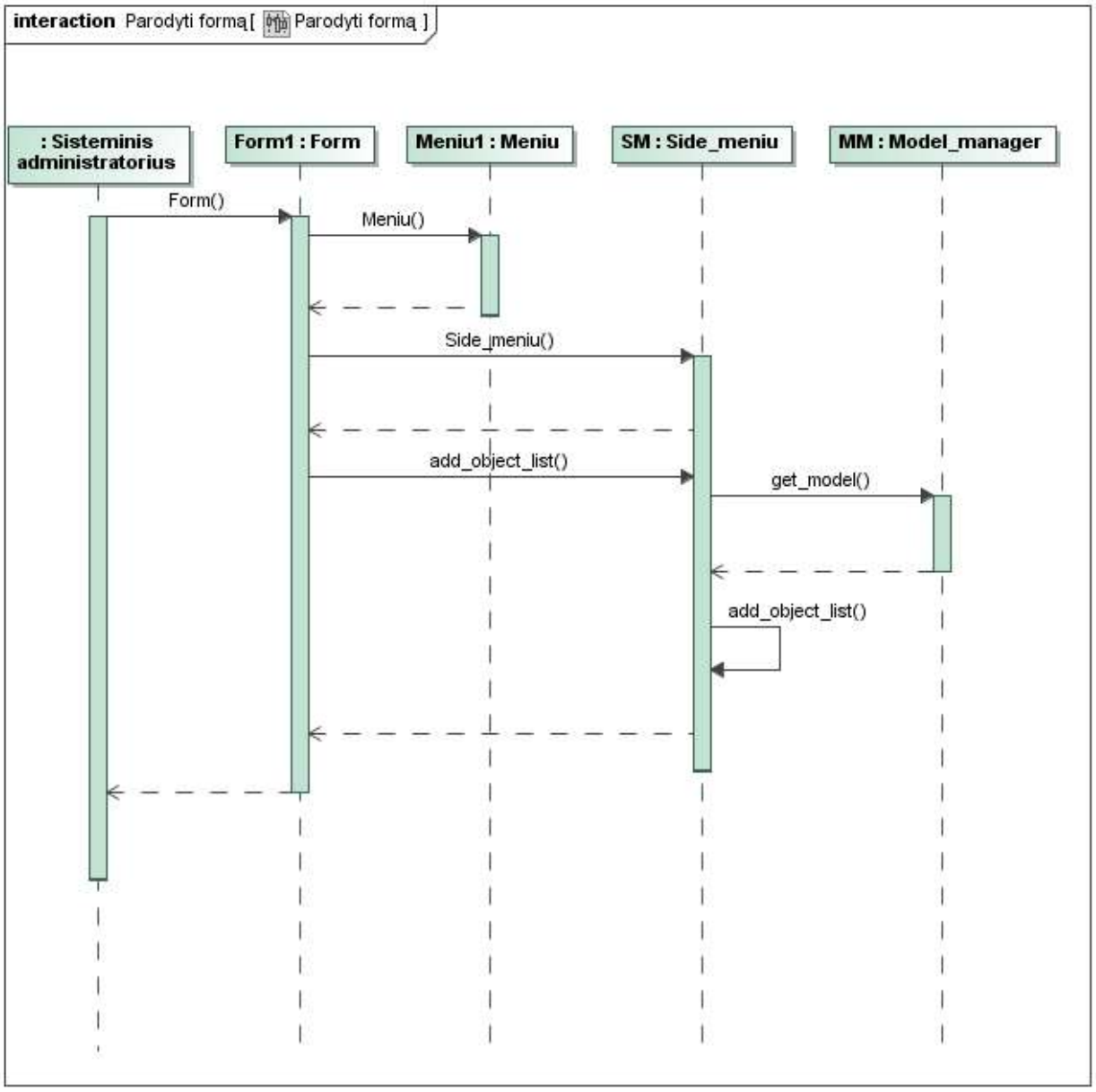

40 pav. Object manager paketo klasių diagrama

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

### 7.1.6. Sistemos dinaminis vaizdas

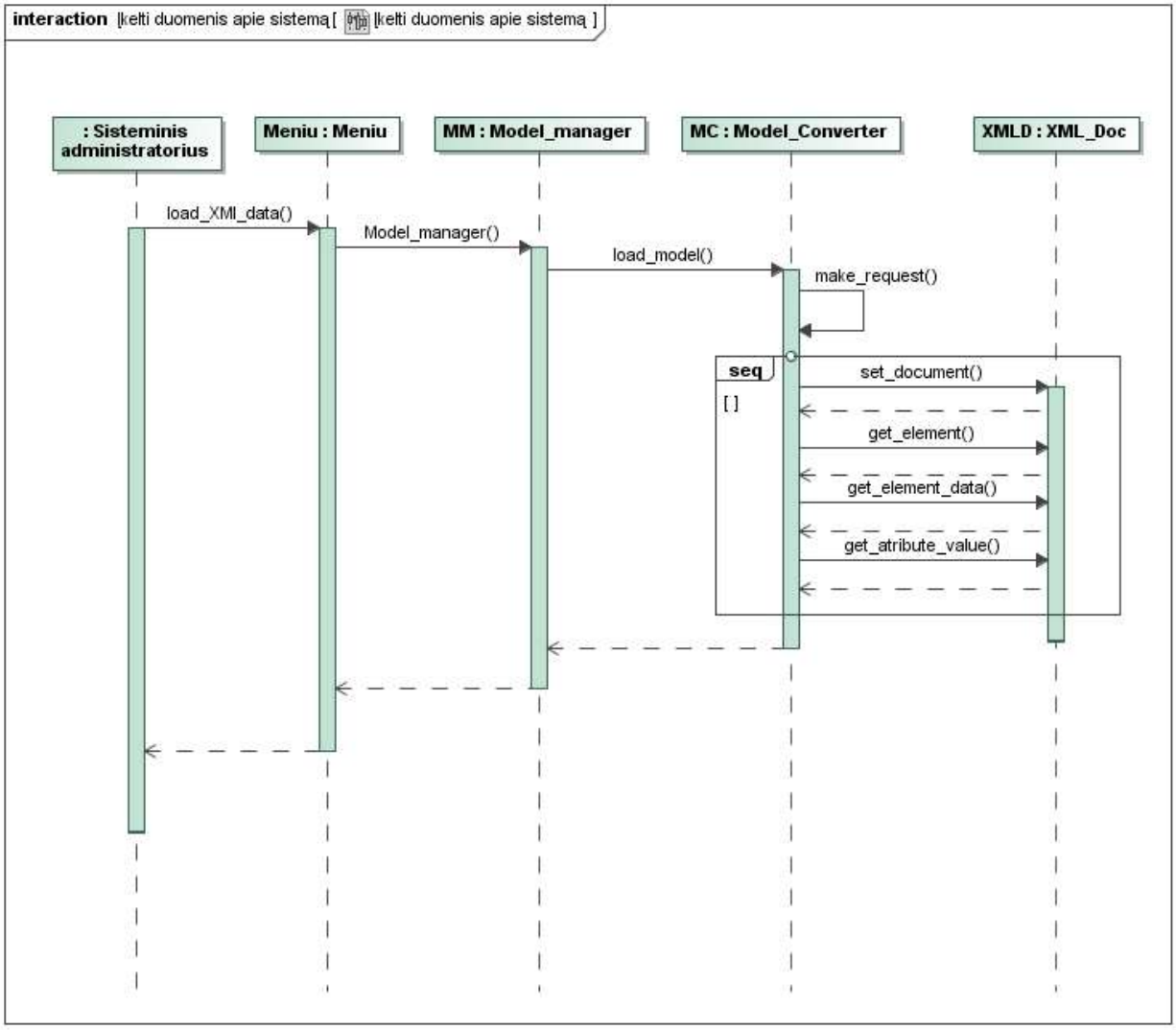
#### 7.1.6.1. Sekų diagramos

**Parodyti formą**



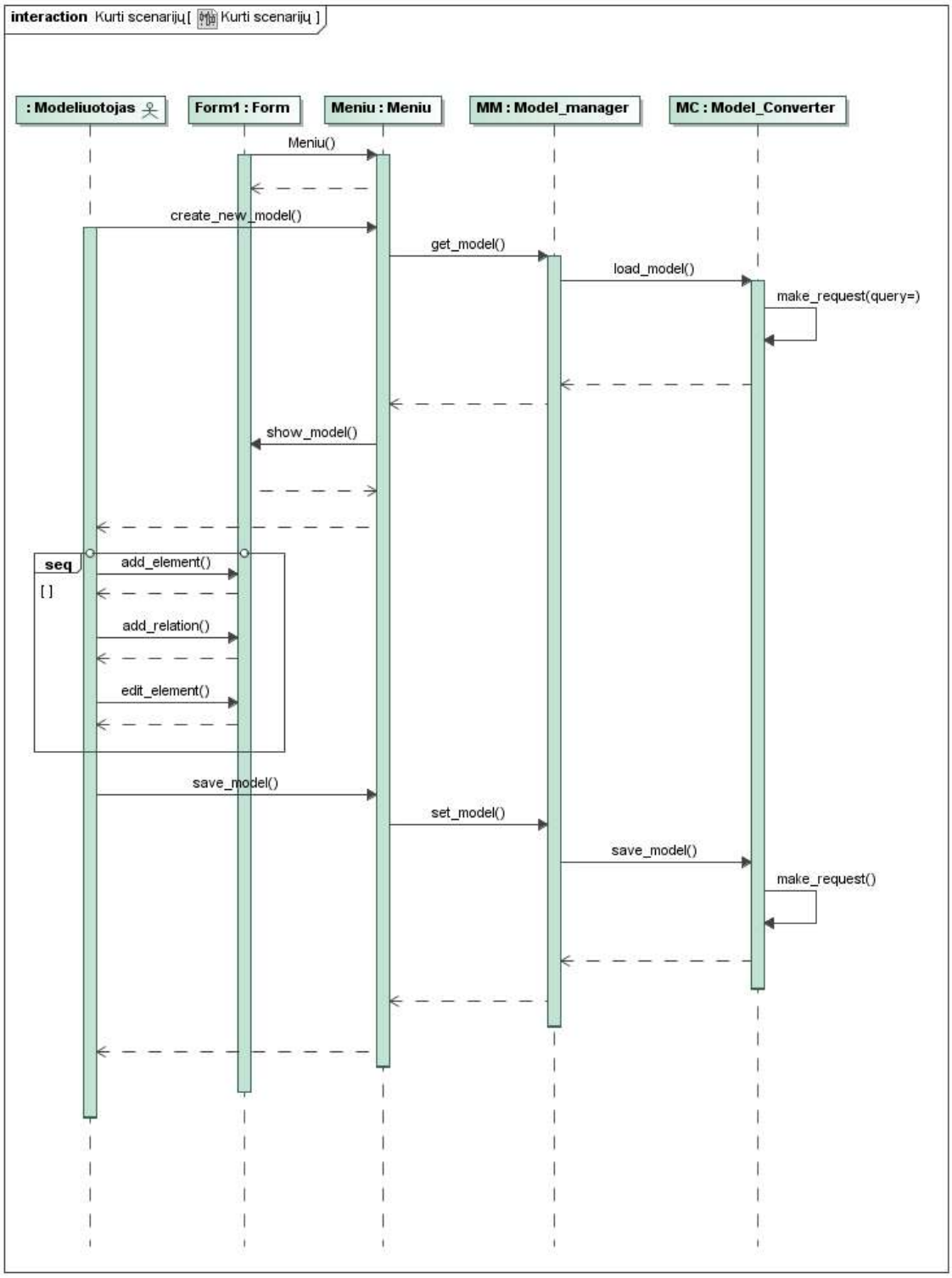

41 pav. Sekų diagrama „parodyti formą“

**Įkelti duomenis apie sistemą**



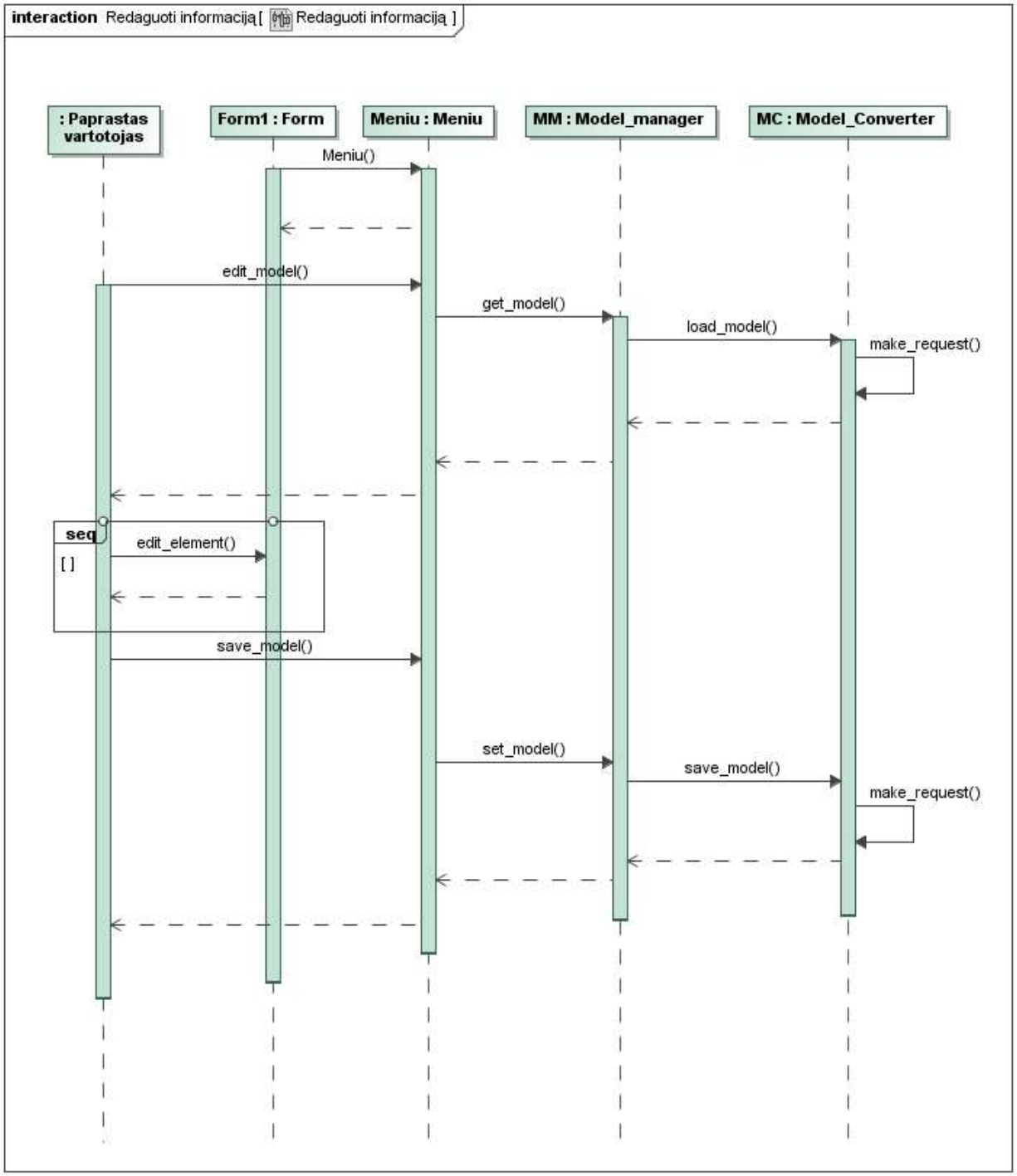

42 pav. Sekų diagrama „įkelti duomenis apie sistemą“

**Kurti scenarijų**



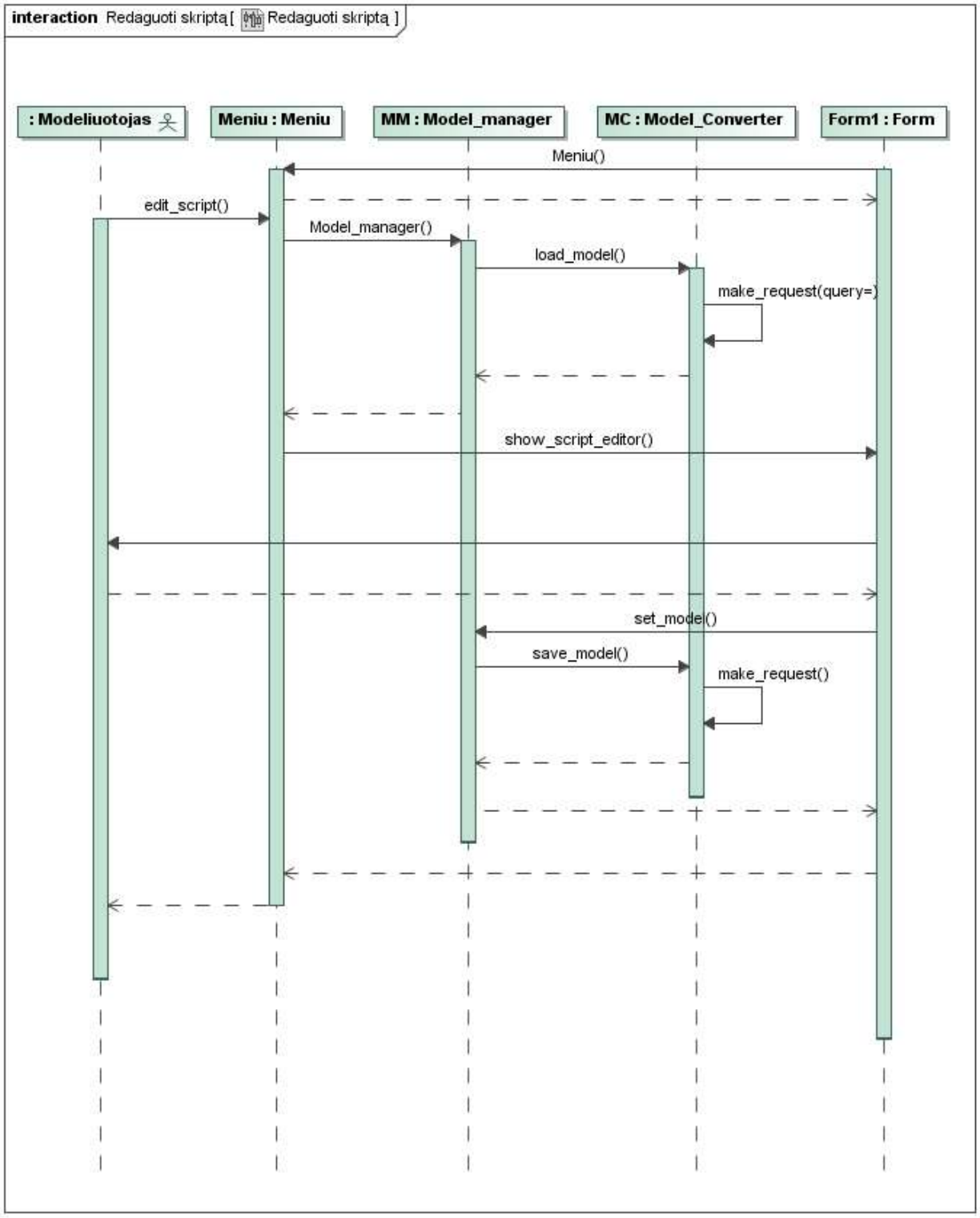

43 pav. Sekų diagrama „kurti scenarijų“

### Redaguoti informaciją




44 pav. Sekų diagrama „redaguoti informaciją“

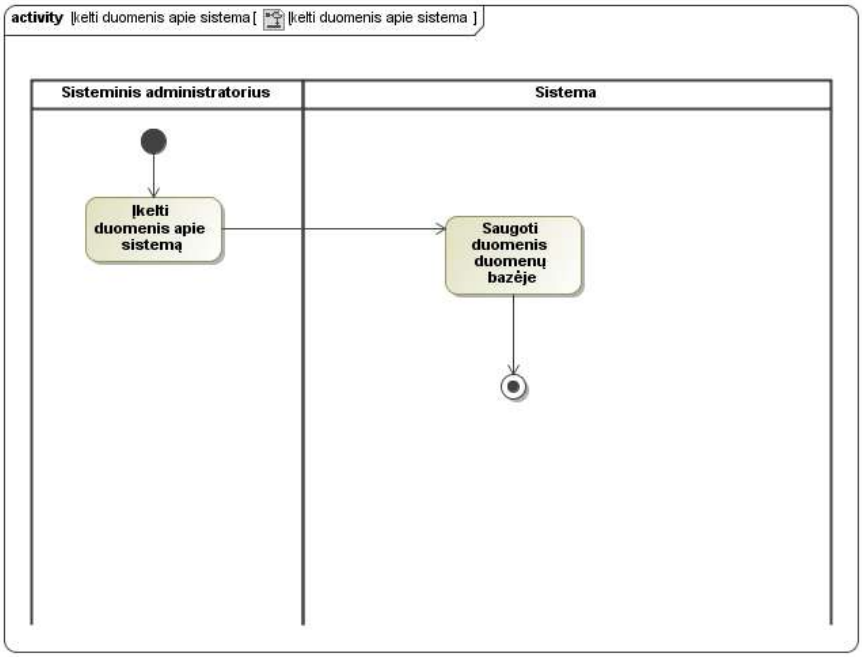
### Redaguoti skriptą




45 pav. Sekų diagrama „redaguoti skriptą“

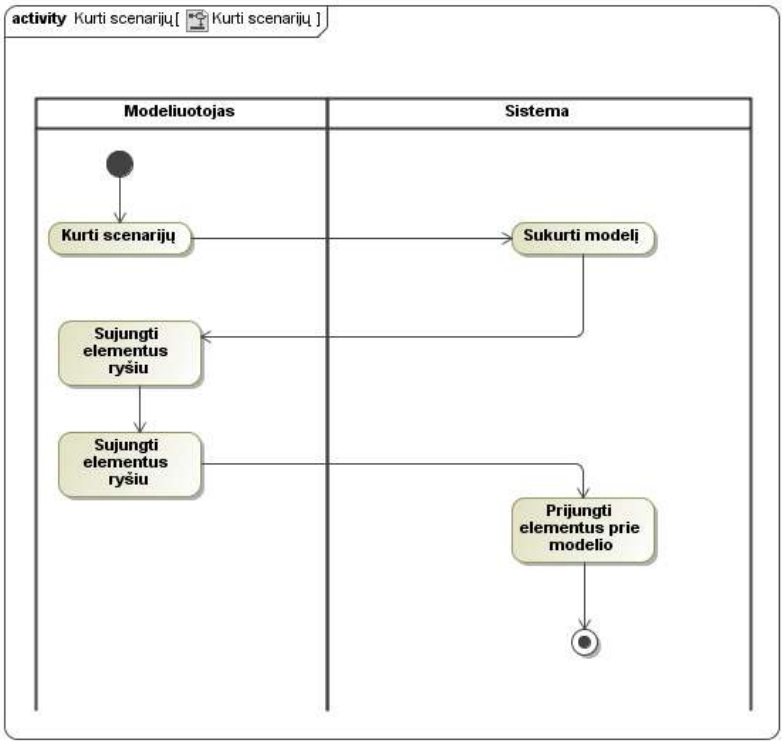
### 7.1.6.2. Veiklos diagramos

**Įkelti duomenis apie sistemą**



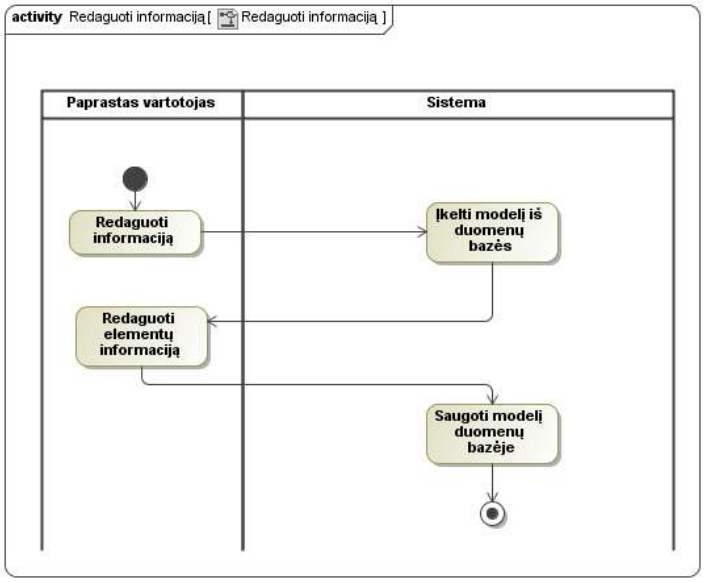

46 pav. Veiklos diagrama „įkelti duomenis apie sistemą“

**Kurti scenarijų**



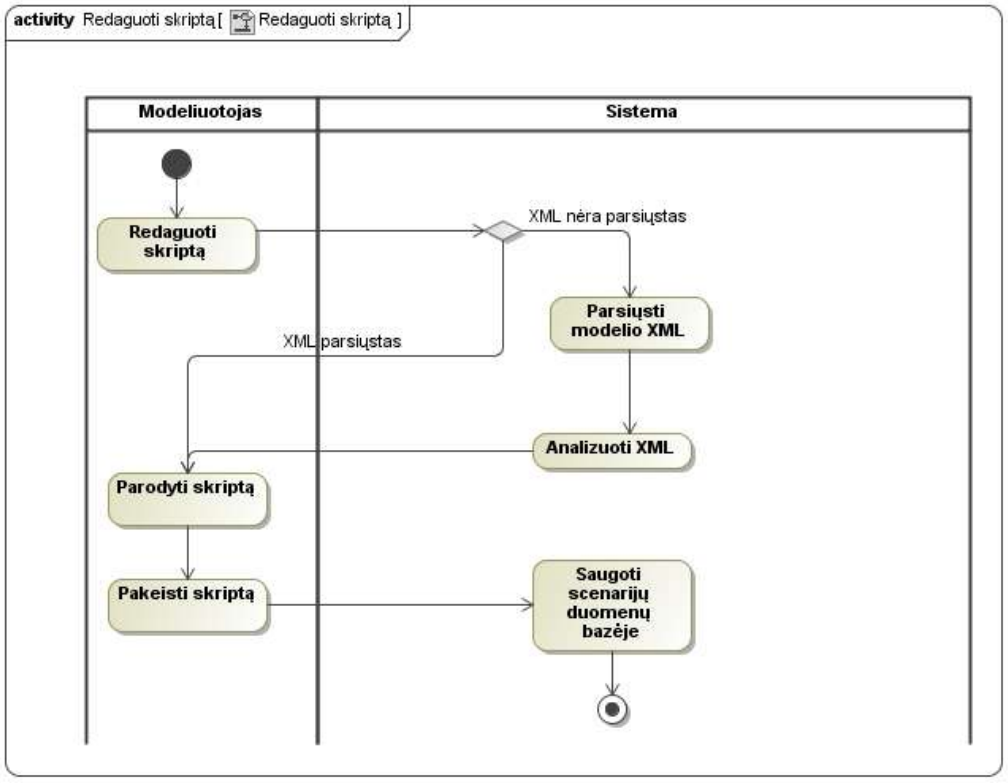

47 pav. Veiklos diagrama „kurti scenarijų“

## Redaguoti informaciją



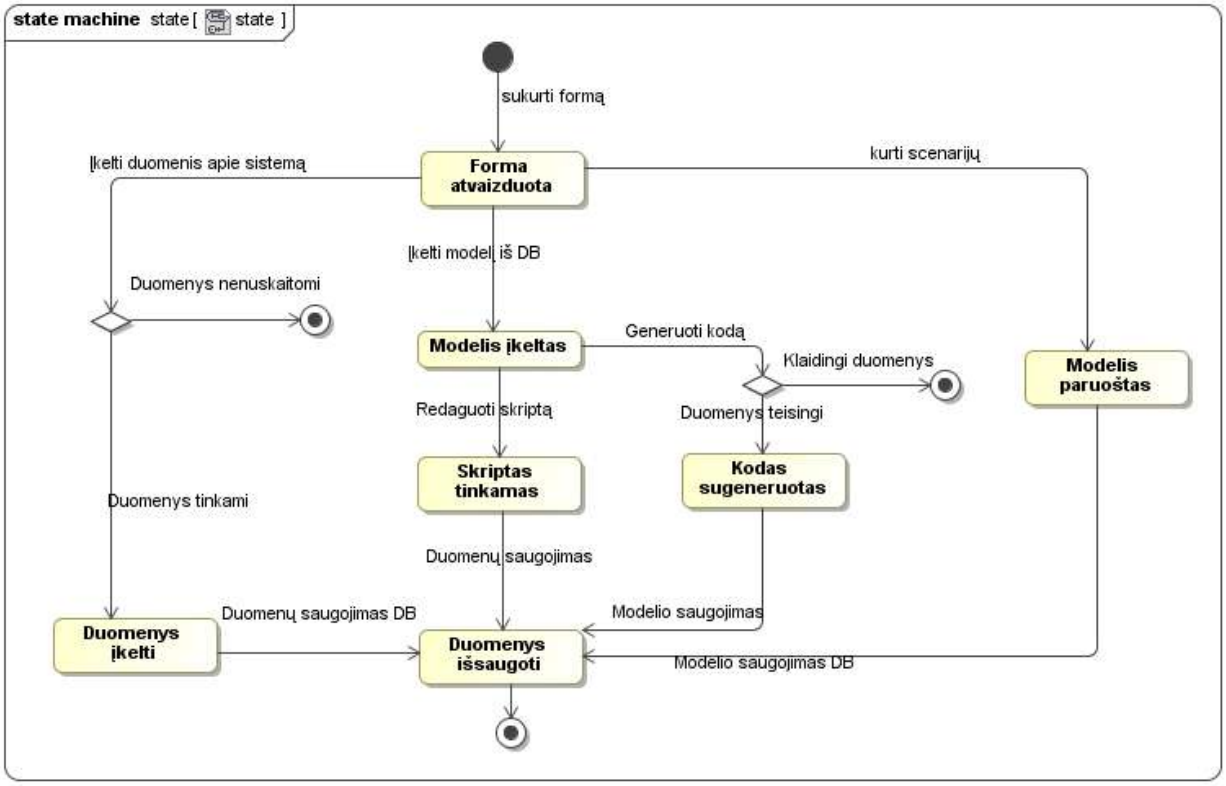

48 pav. Veiklos diagrama „redaguoti informaciją“

## Redaguoti skriptą




49 pav. Veiklos diagrama „redaguoti skriptą“

**Būsenų diagrama**




50 pav. Būsenų diagrama

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

# 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.

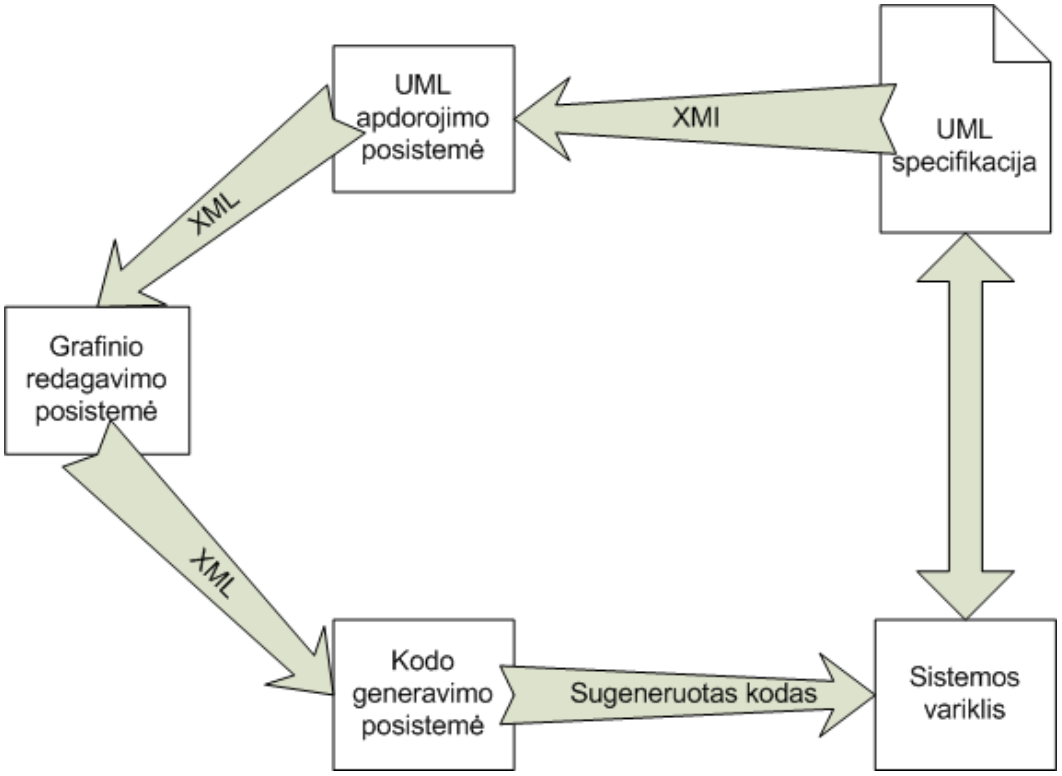
Šis produktas priartina projektavimą prie Executable UML. Executable UML leidžia iš anksto patikrinti programos kodą, sugeba išversti UML modelį tiesiai į efektyvų programinį kodą, ir leidžia atidėti įgyvendinimo sprendimus, iki paskutinės minutės[3].

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.

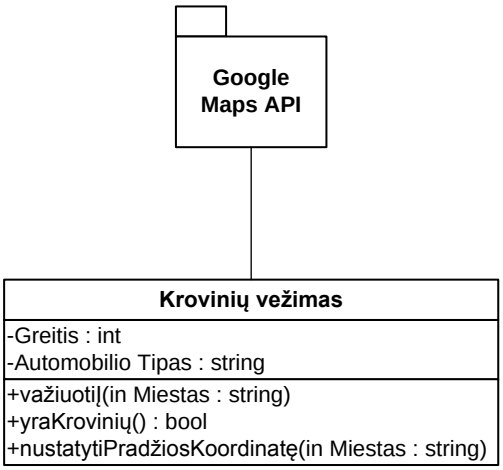



**Pav. 1. Įrankio architektūros diagrama**

# 4. Įrankio veikimo pavyzdys

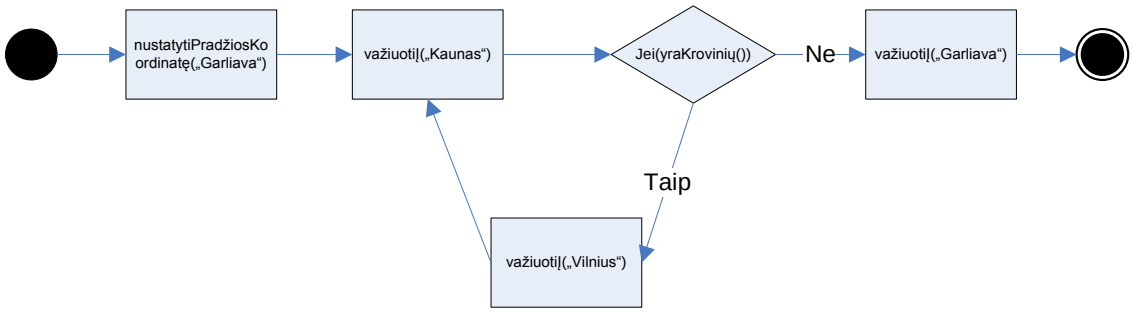
Pavyzdinė sistema realizuota su „Google maps“ įskiepiu. Trumpas scenarijaus aprašymas:
Vartotojui reikia sudaryti krovinio vežimo scenarijaus iš Kauno į Vilnių demonstraciją. Įmonės būstinė yra Garliavoje. Vairuotojas važiuoja į Kauną, kur tikrinama ar yra pervežimo užsakymų. Jeigu užsakymų yra, krovinys vežamas į Vilnių ir grįžtama atgal. Ši procedūra kartojama tol, kol kroviniai baigiasi. Tada vairuotojas grįžta į būstinę.

Sistemos UML specifikacijos pavyzdys:




**Pav. 2. Pervežimų sistemos UML klasių diagrama**

Grafinio redaktoriaus sumodeliuotas vaizdas:




**Pav. 3. 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. 4. 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. 5. 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>