KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA


Paulius Paškevičius

# Hipertekstinės grafinės vartotojo sąsajos kūrimas aukšto abstrakcijos lygmens deklaratyvia sintakse

Magistro darbas

Darbo vadovas:
prof. Eduardas Bareiša

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA


Paulius Paškevičius

# Hipertekstinės grafinės vartotojo sąsajos kūrimas aukšto abstrakcijos lygmens deklaratyvia sintakse

Magistro darbas

Recenzentas:
prof. dr. Rimantas Butleris
2010-05-28

Vadovas:
prof. Eduardas Bareiša
2010-05-31

Atliko:
IFM – 4/2 gr. stud.
Paulius Paškevičius
2010-05-31

Kaunas, 2010

# Hypertext graphical user interface definition using high abstraction level declarative syntax

Paulius Paškevičius

## SUMMARY

In master theses hypertext graphical user interface definition using high abstraction level declarative syntax is analyzed and architecture model is suggested.

Suggested architecture defines graphical user interface using high abstraction level elements. Basic element set of more than 20 elements is defined, declarative XML[1] notation is suggested and graphical user interface library for HTML[2] is developed with JavaScript to ensure much faster and easier standard-based graphical user interface development.

Provided method has no public analogues yet and is suggested for complex graphic user interfaces. Experimental studies proved that declarative syntax and high abstraction level can reduce programming language code from 3.1 times on trivial GUI[3] elements to 204.1 times on complex GUI solutions. Studies have showed that when the number of HTML elements composing graphical user interface grows, even better effectiveness can be achieved.

Developed architecture is integrated in software development model where graphical user interface and logic are semi-automated using UML model. Traditional text editor is changed by data driven design tool, use-cases are developed using graphical editor, data infrastructure is build from the model and solution is delivered in several programming technologies.

[1] XML (angl. extensible markup language)
[2] HTML (angl. hypertext markup language)
[3] GUI (angl. graphic user interface)

# Anotacija

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

Tezėse nagrinėjamas hipertekstinės grafinės vartotojo sąsajos aprašymas aukšto abstrakcijos lygmens elementais, juos apibrėžiant deklaratyvia sintakse.

Siūloma architektūra aprašo hipertekstinę grafinę vartotojo sąsają aukšto abstrakcijos lygmens elementais. Apibrėžiamas 20-ties esminių grafinių elementų rinkinys, deklaratyvi XML notacija ir suprojektuojama HTML grafinės vartotojo sąsajos biblioteka, veikianti JavaScript pagrindu bei užtikrinanti ženkliai greitesnį ir paprastesnį grafinės vartotojo sąsajos kūrimą.

Pateiktas metodas neturi viešų analogų ir yra skirtas sudėtingiems grafinės vartotojo sąsajos sprendimams. Eksperimentiniais tyrimais parodoma, kad deklaratyvi notacija ir aukštas abstrakcijos lygmuo gali sumažinti programinį kodą nuo 3,1 karto trivialiems GUI elementams iki 204,1 karto sudėtingiems GUI sprendimams. Eksperimentai patvirtina, kad didėjant grafinę vartotojo sąsają sudarančių HTML elementų kiekiui, galima tikėtis dar geresnių efektyvumo rodiklių.

Sukurta architektūra yra integruota į programinės įrangos projektavimo modelį, kuriame vartotojo sąsajos ir logikos kūrimas iš dalies automatizuotas naudojant UML modelį. Tradicinis tekstinis redaktorius pakeistas duomenimis paremtu projektavimo įrankiu, panaudos atvejai vystomi scenarijais su grafiniu redaktoriumi, duomenų infrastruktūra generuojama iš modelio, o realizacija pateikiama keliomis programavimo technologijomis.

**Turinys**

## Lentelės



## Paveikslai

# 1. ĮVADAS

## 1.1. Tiriamoji problema

Šiame darbe nagrinėjamas hipertekstinės grafinės vartotojo sąsajos aprašymas aukšto abstrakcijos lygmens elementais, juos apibrėžiant deklaratyvia sintakse.

## 1.2. Temos aktualumas

Kompiuterių mokslas deklaratyvų programavimą apibrėžia kaip programavimo paradigmą, aprašančią logiką, neatskleidžiant kaip yra atliekami veiksmai. Deklaratyvios programavimo kalbos nurodo, ką turi atlikti programa, vietoj to kaip tai atliekama – priešingai nei imperatyvios kalbos, kurios reikalauja aprašyti konkretų įgyvendinimo algoritmą.

Abstrakcija yra kitas mechanizmas, leidžiantis sumažinti detalumą bei paslėpti tam tikrus faktorius. Kuo aukštesnis abstrakcijos lygmuo, tuo mažiau kreipiama dėmesio, apie tai iš ko susideda ar kaip veikia naudojamas objektas.

Deklaratyvus aprašymo būdas ir aukštas abstrakcijos lygmuo gali ženkliai supaprastinti programavimą. Remiantis sistemos architektūrą atspindinčia topologija ir naudojant abstrakciją, sprendimą galima konstruoti iš stambių aukšto abstrakcijos lygmens elementų, o naudojantis deklaratyvia sintakse pagal kontekstą iš dalies automatizuoti funkcionalumą. Abiejų metodų naudojimas didina programos kodo skaitomumą, pašalina perteklines instrukcijas bei suskaido kūrimo procesą į dalis, kurias gali įgyvendinti skirtingų sričių profesionalai.

Grafinė vartotojo sąsaja turi potencialo išnaudoti daugelį šių metodų privalumų. Turint sąsają atspindinčią topologiją, sprendimo grafinę vartotojo sąsają galima konstruoti iš stambių aukšto abstrakcijos lygmens elementų. Tai aktualu sudėtingoms sistemoms, kurių grafinė vartotojo sąsaja susideda iš daugelio atominių elementų, susietų silpnais semantiniais ryšiais.

Tokia yra hipertekstinė grafinė vartotojo sąsaja. HTML yra populiariausia hipertekstinės grafinės vartotojo sąsajos aprašymo kalba. Ši kalba yra labai universali. Iš jos 91 (HTML 4.01) atominio elemento galima sukonstruoti bet kokį aukšto abstrakcijos lygmens elementą, tačiau sprendimas pasižymi dideliu semantiškai neapibrėžtų ryšių tarp atskirų elementų gausa ir silpnu programavimo modeliu.

# 2. HIPERTEKSTINĖS GRAFINĖS VARTOTOJO SĄSAJOS BIBLIOTEKOS ANALIZĖ

## 2.1. Problema

HTML universalumas yra didžiausias šios kalbos privalumas, bet kartu, deja, ir trūkumas. Norint sukonstruoti sudėtingą grafinės vartotojo sąsajos elementą reikia apjungti šimtus ar tūkstančius atominių elementų. Didelių sprendimų grafinė vartotojo sąsaja paprastai susideda iš dešimčių sudėtingų elementų, kuriuos sudaro dešimtys tūkstančių atominių elementų.

Kuo funkcionalesnė grafinė vartotojo sąsaja, tuo daugiau atominių elementų ją sudaro. Problema yra atominių elementų valdymas – kuo didesnis kiekis, tuo sudėtingesnis valdymas. Dažnai tai lemia nesuvaldomą grafinę vartotojo sąsają, klaidingą funkcionalumą ar neatitikimą svarbiems standartams. Didelį sprendimą yra sudėtinga vystyti, o vienas neatsargus patobulinimas dažnai gali baigtis visišku funkcionalumo sutrikdymu.

## 2.2. Hipotezė

Perėjus nuo hipertekstinės grafinės vartoto sąsajos HTML dokumentuose kūrimo iš atominių elementų prie jų kūrimo iš stambių aukšto abstrakcijos lygmens elementų, pavyktų pasiekti didesnį sprendimų vystymo efektyvumą.

Jeigu HTML 4.01 kalba leistų aprašyti aukšto abstrakcijos lygmens elementus, tai:

- paspartintų grafinės vartotojo sąsajos kūrimą, nes operuojant stambiais aukšto abstrakcijos lygmens elementais atsiranda mažiau ryšių tarp elementų
- pagerintų grafinės vartoto sąsajos architektūrą, nes stambesnius aukšto abstrakcijos lygmens elementus lengviau valdyti.
- užtikrintų mažesnę klaidų tikimybę bei atitikimą W3C[4] standartams, nes aukšto abstrakcijos lygmens elementų funkcionalumas yra lengviau patikrinamas.

## 2.3. Darbo tikslai ir uždaviniai

Pagrindinis šio darbo tikslas yra sukurti architektūrą HTML grafinės vartotojo sąsajos aprašymui aukšto abstrakcijos lygmens elementais, juos apibrėžiant deklaratyvia sintakse.

Architektūrinis modelis turėtų gerai integruotis į egzistuojantį HTML hipertekstinės grafinės vartotojo sąsajos modelį ir nekelti papildomų reikalavimų esamiems HTML vartotojams.

[4] W3C (angl. world wide web consortium)

Siekiant šio tikslo, buvo sprendžiami šie uždaviniai:

1) atiliekama HTML architektūros išplėtimo galimybių analizė
2) pasirenkamos architektūros įgyvendinimo technologijas
3) apibrėžiama naujų grafinės vartotojo sąsajos elementų aibė
4) pasiūlyta naujų grafinės vartotojo sąsajos elementų deklaratyvią notacija
5) suprojektuota naujos HTML grafinės vartotojo sąsajos biblioteka
6) eksperimentiškai ištirtas pasiūlytos architektūros efektyvumas

### 2.4. Darbo metodai ir priemonės

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

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

### 2.5. Panašių sprendimų apžvalga

Pasaulyje egzistuoja keletas panašių sprendimų, kurie naudoja aukšto abstrakcijos lygmens deklaratyvią sintaksę sprendimo grafinės vartotojo sąsajos struktūros aprašymui.

Šiandieną nėra analogiško sprendimo hipertekstinės (pvz. HTML) grafinės vartotojo sąsajos kūrimui iš vartotojo apibrėžtų (aukšto abstrakcijos lygmens) elementų.

Apžvalgoje yra nagrinėjamos trys technologijos, kurių architektūra labiausiai panaši į siūlomą architektūrą HTML grafinei vartotojo sąsajai:

- Adobe Flex
- Microsoft WPF
- Microsoft Silverlight

Apžvalgos tikslas yra išryškinti stipriąsias ir silpnąsias produktų puses ir atsirinkti veiksmingas strategijas ir išbandytus metodus savojo sprendimo architektūrai pagerinti.

### 2.5.1. Adobe Flash/Flex sprendimas

Adobe naudoja deklaratyvią sintaksę Flash/Flex platformos sprendimams vystyti. Adobe Flex programinis karkasas [5] leidžia aprašyti grafinę vartotojo sąsają iš vartotojo apibrėžtų elementų, juos komponuoti tarpusavyje ir automatizuoti paprastus veiksmus su šiais elementais.

Adobe Flex technologijos branduolys yra grafinė biblioteka, kuri įgyvendina elementų funkcionalumą. Architektūra buvo pasiūlyta 2004 m. Šiandieną architektūra pasižymi:

- dideliu lankstumu (lengvai konfigūruojama ir praplečiama)
- efektyvumu (pagreitina sprendimų vystymą, mažina klaidų tikimybę)

Adobe Flex programinio karkaso sprendimus be duomenų sudaro dviejų tipų resursai:

- grafinės vartotojo sąsajos aprašymo (*.mxml) failai
- funkcionalumo aprašymo (*.as) failai

MXML deklaratyvia XML sintakse apibrėžiami grafinės vartoto sąsajos elementai, jų parametrai, unikalūs identifikatoriai, kuriamos jų hierarchijos. MXML failai turi apibrėžtą standartinių elementų aibę iš kurių konstruojama grafinė vartotojo sąsaja ir funkcionalumas. Tai pagrindiniai grafinės vartotojo sąsajos elementai:

- mygtukas (Button)
- teksto laukelis (Text)
- komentarinis laukelis (Label)
- iškrentantis sąrašas (Combobox)
- požymio pasirinkimas (Checkbox)
- grupavimo elementas (Panel)
- ir daugelis kitų dažai naudojamų elementų

Vienas didžiausių Adobe Flex privalumų lankstus deklaratyviai sintakse paremtas grafinės vartotojo sąsajos kūrimo modelis, leidžiantis kurti savo atominius elementus.

Tuo tarpu AS failuose ActionScript imperatyvia sintakse per unikalų identifikatorių pasiekiami grafinės vartotojo sąsajos elementą bei įgyvendinama išplėstinė veiksmų logika. Kai sprendimas kompiliuojamas į tarpinį dvejetainį kodą, visi MXML deklaratyvūs aprašai verčiami į atitinkamas imperatyvias konstrukcijas, kurios įrašomos į AS failus ActionScript kalba.

[5] programinis karkasas (angl. framework)

Apibendrinant galima teigti, kad Adobe Flex sprendimas yra labai pavykęs deklaratyvios sintaksės naudojimo grafinei vartotojo sąsajai kurti pavyzdys. Jis palengvina ir pagreitina sprendimų vystymą ir nuosekliai laikosi interneto standartų (XML, ActionScript, CSS[6]).

### 2.5.2. Microsoft WPF7 sprendimas

Antrasis ne mažiau populiarus sprendimas yra Microsoft technologija, skirta darbastalio grafinei vartotojo sąsajai kurti.

Microsoft WPF dokumentai taip pat yra sudaryti iš dviejų dalių, kurių vienoje deklaratyvia XML sintakse aprašoma grafinės vartotojo sąsajos struktūra, o kituose pasirinkta .NET kalba užrašomas funkcionalumas.

Ši technologija labai panaši į Adobe Flex sprendimą, nes turi panašų elementų rinkinį ir XML deklaratyvią sintaksę visuomet transformuoja į .imperatyvią sintaksę. Windows darbastalio programų grafinė vartotojo sąsaja verčiama į .NET kalbos kodą, o vėliau į dvejetainį kodą[8] .

Nors Microsoft WPF yra panaši į Adobe Flex technologiją, svarbu pastebėti esminį skirtumą, kad deklaratyvios XML kalbos sakiniai ne visuomet transformuojami į dvejetainį kodą.

Microsoft WPF privalumas – didelis grafinės vartotojo sąsajos elementų rinkinio palaikymas bei programuotojo ir dizainerio darbo atskyrimas o trūkumas – naujų nestandartinių sprendimų kūrimas, ignoruojant atvirus interneto standartus ir rekomendacijas.

### 2.5.3. Microsoft Silverlight sprendimas

Trečiasis sprendimas yra Microsoft Silverlight technologija, kuri skirta interneto programų grafinei vartotojo sąsajai kurti.

Microsoft Silverlight yra WPF poaibis, kurio dokumentai sudaryti iš tų pačių dviejų dalių, kai vienoje deklaratyvia XML sintakse aprašoma grafinės vartotojo sąsajos struktūra, o kituose funkcionalumas. Tačiau nuo Adobe Flex ir Microsoft WPF sprendimų ši technologija skiriasi tuo, kad ji netransformuoja XML deklaratyvios sintaksės į .NET kalbos sakinius.

Silverlight žiniatinklio programų grafinė vartotojo sąsaja paliekama XML formatu bei integruojama į HTML dokumentus, o vėliau interpretuojama ir vykdoma.

Silverlight privalumas – deklaratyvaus aprašo interpretavimas ir vykdymas realiu laiku.

---

[6] CSS (angl. Cascading Style Sheets)
[7] WPF (angl. windows presentation foundation)
[8] dvejetainis kodas (angl. byte code)

### 2.5.4. Įvertinimas

Apžvelgus visus deklaratyvią sintaksę grafinei vartoto sąsajai kurti naudojančius sprendimus, buvo palyginti šių paketų privalumai ir trūkimai.

Šio palyginimo tikslas – išskirti svarbiausius nagrinėtų technologijų aspektus ir pagal juos įvertinti įrankių galimybes bei kokybę.

Apibendrinti kiekvieno sprendimo apžvalgos rezultatai pagal bendrus kriterijus pareikiami apačioje esančiose lentelėse.

Pirmoji lentelė įvertina sprendimuose palaikomas/nepalaikomas galimybes.

1 lentelė. Panašių sprendimų apžvalga. Galimybių palaikymas

| Galimybės | Adobe Flash/Flex | Microsoft WPF | Microsoft Silverlight |
|---|---|---|---|
| Grafinei vartotojo sąsajai kurti **naudojama deklaratyvi sintaksė** | ✓ | ✓ | ✓ |
| Grafinei vartotojo sąsajai kurti **naudojamas esminių elementų rinkinys** | ✓ | ✓ | ✓ |
| Grafinei vartotojo sąsajai kurti **naudojamas sudėtingų elementų rinkinys** | ✓ | ✓ | |
| Grafinei vartotojo sąsajai kurti **naudojami atviri standartai** | ✓ | | |
| | 4/4 | 3/4 | 2/4 |

Antroji lentelė įvertina sprendimuose palaikomų galimybių kokybę.

2 lentelė. Panašių sprendimų apžvalga. Galimybių palaikymo kokybė

| Galimybės | Adobe Flash/Flex | Microsoft WPF | Microsoft Silverlight |
|---|---|---|---|
| **Grafinės vartotojo sąsajos standartinių elementų** gausa, aktualumas ir funkcionalumas | 10 | 9 | 7 |
| **Grafinės vartotojo sąsajos architektūros** lankstumas ir efektyvumas | 10 | 10 | 8 |
| **Grafinės vartotojo sąsajos kūrimo dokumentacijos** gausa ir kokybė (aktualumas, išbaigtumas, tvarka) | 10 | 10 | 8 |
| **Grafinės vartotojo sąsajos kūrimo pavyzdžių** gausa ir kokybė (aktualumas, išbaigtumas, tvarka) | 10 | 10 | 9 |
| **Grafinės vartotojo sąsajos kūrimo bendruomenės** aktyvumas ir kūrybiškumas | 10 | 9 | 9 |
| | 10,00 | 9,60 | 8,20 |

Iš visų trijų apžvelgtų sprendimų geriausia kokybe pasižymėjo Adobe Flex (10) ir Microsoft WPF (9,6). Abu įrankiai turi gerai išvystytą labai lanksčią architektūrą (įvertinta 10/10) ir labai kokybiškai parašytą dokumentaciją. Tačiau Microsoft WPF (9,6) sprendimas atsilieka standartinių (nemokamų) grafinės vartotojo sąsajos elementų gausa ir bendruomenės aktyvumu ir kūrybiškumu (tikėtina priežastis – sprendimas yra uždaro kodo).

Adobe Flex sprendimas turi labai gerą standartinių (nemokamų) grafinės vartotojo sąsajos elementų rinkinį, kurį vertėtų paanalizuoti, prieš apibrėžiant hipertekstinės grafinės vartotojo sąsajos elementų aibę.

Microsoft Silverlight, nors ir pasižymi galimybių stoka bei žemesne jų įgyvendinimo kokybe (8,2) pasižymi svarbia savybe, deklaratyvų elementų aprašą interpretuoti ir vykdyti realiu laiku. Ši savybė yra labai aktuali hipertekstinės grafinės kūrime ir į ją bus atsižvelgta projektuojant architektūrą.

## 2.6. Projektas

### 2.6.1. Įvadas

Projektinė dalis yra padalinta į dvi dalis, kurios pirmoji apima grafinės vartotojo sąsajos architektūrinio modelio kūrimą, o antroji modelio integravimą į programinės įrangos projektavimo procesą.

Pirmojoje dalyje, kuri yra įvardinama kaip projektinis darbas, pristatomas hipertekstinės grafinės vartotojo sąsajos modelis, leidžiantis kurti sprendimus iš stambių aukšto abstrakcijos lygmens elementų, aprašomų deklaratyvia sintakse.

Antrojoje dalyje, kuri yra įvardinama kaip grupinis projektinis darbas, hipertekstinės grafinės vartotojo sąsajos modelis yra integruojamas į programinės įrangos projektavimo procesą. Pristatomas modelis, leidžiantis grafinės vartotojo sąsajos kūrimą optimizuoti integruotu grafiniu redaktoriumi, pasirinktos programavimo kalbos automatiniu išeities kodo generavimu bei automatiniu duomenų sluoksnio generavimu.

Šis skyrius supažindina su abiem projektais. Pagrindinis dėmesys skiriamas projektiniam darbui, nes jo architektūra grupiniame projektiniame darbe integruojama į programinės įrangos projektavimo procesą.

### 2.6.2. Projektinis darbas

Atlikus panašių sprendimų apžvalgą įsitikinta, kad pasaulyje dar nėra analogiško modelio HTML grafinei vartotojo sąsajai kurti iš aukšto abstrakcijos lygmens elementų, naudojant deklaratyvią sintaksę.

Norint išspręsti tezėse suformuluotas problemas – lėtą ir sudėtingą didelių HTML sprendimų vystymą, dažną funkcionalumo sutrikdymą ar nuoseklų standartų nesilaikymą, buvo siekiama pašalinti problemos priežastį – didelį kiekį silpnų semantinių ryšių tarp elementų.

Projektinis darbas susideda iš šešių nuoseklių etapų, kuriuose reikėjo:

1) atilikti HTML architektūros išplėtimo galimybių analizę
2) pasirinkti architektūros įgyvendinimo technologijas
3) apibrėžti naujų grafinės vartotojo sąsajos elementų aibę
4) pasiūlyti naujų grafinės vartotojo sąsajos elementų deklaratyvią notaciją
5) suprojektuoti naujos HTML grafinės vartotojo sąsajos biblioteką
6) eksperimentiškai ištirti siūlomos architektūros efektyvumą

Pirmajame etape teko gilintis į HTML funkcionalumo branduolį, už kurį atsakingas DOM ir kuris pasiekiamas JavaScript funkcijomis. Šiame etape taip pat buvo nagrinėjamos JavaScript kalbos galimybės sudėtingai architektūrai aprašyti. Atlikus analizę paaiškėjo kad:

- JavaScript kalba nėra OOP[9], tačiau šį funkcionalumą iš dalies galima atkurti
- JavaScript turi visas būtinas priemones XML elementams iš HTML analizuoti
- JavaScript yra pakankamai palaikoma visose moderniose naršyklėse

Šie trys faktai užtikrino, kad siūlomą architektūrą galima įgyvendinti pasitelkus tik HTML ir Javascript technologijas. Tai labai svarbu, kadangi nekuriama naujų standartų, vartotojui nereikia įdiegti papildomų įskiepių ir svarbiausia, architektūros realizacija veikia identiškai tradiciniam HTML sprendimui.

Antruoju etapu teko pasirinkti architektūros įgyvendinimo technologiją. Pirmiausia reikia parinkti deklaratyvią kalbą grafinės vartotojo sąsajos aprašymui. Žinant, kad HTML standartas

[9] OOP (angl. object oriented programming)

yra XML deklaratyvios kalbos poaibis, sprendimas naudot XML yra palankiausias. Jis užtikrina dviejų svarbių magistro tezių tiksluose ir uždaviniuose suformuluotų teiginių įgyvendinimą:

- architektūra turi būti paremta atvirais standartais (pvz. W3C)
  (XML yra W3C standartas. Jo laikymasis užtikrina greitą modelio adaptavimą)

- architektūra turi lengvai integruotis į HTML kalbą
  (HTML yra XML poaibis. Abi kalbos yra giminingos ir gerai integruojasi)

Kitas uždavinys buvo pasirinkti kalbą, kuri galėtų skaityti deklaratyvia sintakse aprašytą grafinę vartotojo sąsają. Remiantis XML ir HTML giminingumu, XML analizei pasirinkta ta pati technologija,tiksluose kuri analizuoja HTML elementus. JavaScript kalba turi programavimo sąsają DOM[10] gebančią manipuliuoti XML manipuliuoti. Toks pasirinkimas užtikrino dar vieno tezių tiksluose ir uždaviniuose suformuluoto teiginio įgyvendinimą:

- architektūra turi nekelti papildomų reikalavimų kitiems HTML vartotojams
  (vartotojams nereikia įdiegti papildomų įskiepių)

Abi pasirinktos XML ir JavaScript technologijos taip pat gerai dera tarpusavyje ir yra minimalus technologijų rinkinys užtikrinantis pilnavertį siūlomos architektūros įgyvendinimą.

Trečiuoju etapu buvo apibrėžta grafinės vartotojo sąsajos elementų aibė. Atlikus panašių sprendimų analizę buvo suformuotas pagrindinių elementų rinkinys vartotojo sąsajai kurti.

Grafinės vartotojo sąsajos elementai buvo padalinti į dvi grupes: grupuojančius ir atominius elementus. Atominiai elementai yra baigtiniai ir savyje negali talpinti kitų elementų, tuo tarpu grupuojančių elementų paskirtis yra apjungti atskirus atominius elementus. Atominių elementų aibę sudaro pagrindiniai grafinės vartotojo sąsajos elementai: mygtukas, tekstinis laukas, požymio pasirinkimas, paveikslėlis, komentarinis laukas ir pan. Iš atominių elementų kuriami aukštesnio abstrakcijos lygmens elementai kaip žemėlapis, galerija ir kt. Grupuojančių elementų pavyzdžiai yra dialogas, kontekstinis meniu, vedlys ir t.t.

HTML grafinės vartotojo sąsajos elementų rinkinį sudaro apie bazinių 20 elementų. Elementus galima komponuoti tarpusavyje ir sudarinėti naujus atominius elementus.

[10] DOM (angl. document object model)

Ketvirtuoju etapu buvo suformuota deklaratyvi grafinės vartotojo sąsajos elementų notacija XML kalba. Pasinaudojant panašių sprendimų analize, ypač Adobe Flex Framework buvo pasirinktos taisyklės grafinės vartotojo sąsajai aprašyti.

Apačioje pateikiamas deklaratyvus HTML grafinės vartotojo sąsajos elemento aprašas.

```
<Dialog id="search" width="265" height="150" align="left" valign=top" type="modal">
  </Image id="imgFind" image="./search_bw.png" wheight="14" top="2" onclick="{search.focus()}">
  </Label id="lblQuery" text="Paieška:" width="50" top="3" onclick="{search.focus()}">
  </Button id="btnHelp" text="?:" wheight="17" left="1" align="right" onclick="{search.help()}">
  </Button id="btnFind" text="»" wheight="17" left="1" align="right">
  </TextBox id="txtQuery" width="150" height="13" maxlength="25" stateful="true" align="right">
</Dialog>
```

1 pav. Grafinės vartotojo sąsajos elemento prototipo deklaratyvi notacija

Viršuje aprašomas teksto paieškos dialogas, turintis penkis elementus: paveikslėlį, antraštę, du mygtukus bei tekstinį laukelį. Apraše XML elementai atitinka HTML grafinės vartotojo sąsajos elementus, o XML atributai nurodo elementų savybes (`width`, `height`, `align`, `valign`, `top`, `left`, `text` ir kt.) bei užregistruoja įvykius (`onclick`) apdorojančius metodus. Aprašas pasižymi glaustumu ir aiškia struktūra bei ateityje gali būti sudaromas grafiniu redaktoriumi.

Deklaratyvus apibrėžimas vykdymo metu yra transformuojamas į imperatyvias JavaScript instrukcijas, kurias tuoj pat įvykdo HTML grafinės vartotojo sąsajos biblioteka:

```
search = new Dialog(DialogType.Modal, {id:"search",width:265,height:150,align:Align.Left, valign:VAlign.Top});
search.AddControl(new Image({id:"imgFind", image:"./search_bw.png", wheight:14, top:2, onClick:function(){
  search.focus()
}}));
search.AddControl(new Label({id:"lblQuery",text:"Paieška:",width:50,top:3,onClick:function(){search.focus()}}))
search.AddControl(new Button({id:"btnHelp",wheight:17,left:1,text:"?", align:Align.Right, onClick:function(){
  search.help()
}}));
search.AddControl(new Button({id:"btnFind", wheight:17, left:1, text:"»", align:Align.Right}));
search.AddControl(new TextBox({id:"txtQuery",width:150,height:13,maxlength:25,stateful:true,align:Align.Right})

search.focus = function() {
  // CUSOM CODE //
}
search.help = function() {
  // CUSOM CODE //
}
search.AppendTo("content");
```

2 pav. Grafinės vartotojo sąsajos elemento prototipo imperatyvi notacija

Tolimesnis HTML grafinės vartotojo sąsajos elemento funkcionalumas plečiamas naudojantis supaprastintu HTML grafinės vartotojo sąsajos programavimo modeliu. Šio funkcionalumo užrašymas ateityje taip pat gali būti automatizuotas naudojant grafinį redaktorių.

Grafinės vartotojo sąsajos biblioteka sukuria ir ekrane atvaizduoja prototipinį elementą.


3 pav. Grafinės vartotojo sąsajos elemento prototipo vizualus vaizdas.

Penktuoju etapu buvo suprojektuota HTML grafinės vartotojo sąsajos biblioteka, kurios pagrindinis uždavinys atvaizduoti grafinę vartotojo sąsają. Tai JavaScript biblioteka vykdymo metu komponuojanti HTML 4.1 standarto elementus, suprojektuota objektiškai orientuotu programavimo stiliumi bei naudojanti standartinę DOM sąsają.

HTML grafinės vartotojo sąsajos biblioteka palengvina ir pagreitina sprendimų vystymą. Naudojantis biblioteka galima kurti sudėtingas HTML sistemas.

Suprojektuota architektūra tinkamai išsprendžia magistro tezėse suformuluotą problemą – silpnus semantinius ryšius tarp standartinių HTML 4.1 kalbos elementų, dėl kurių didelių HTML sprendimų grafinės vartotojo sąsajos vystymas tampa lėtas ir sudėtingai įgyvendinamas. XML ir JavaScript technologijos gerai integruojasi į standartinius HTML sprendimus, nesukuria naujų standartų, o JavaScript grafinės vartotojo sąsajos biblioteka užtikrina stiprius semantinius ryšius tarp elementų bei pateikia programavimo modelį jiems valdyti.

### 2.6.3. Grupinis projektinis darbas

Grupinis projektinis darbas – tai papildomos pastangos skirtos hipertekstinės grafinės vartotojo sąsajos struktūros ir funkcionalumo kūrimo automatizavimui jį integruojant į programinės įrangos projektavimo procesą.

Grupinis projektinis darbas yra atliekamas keturių žmonių komandoje, kuriuos jungia bendras tikslas – naujo programinės įrangos projektavimo modelio pasiūlymas pagal duomenimis paremtą architektūrą[11]. Pagrindinis grupinio projekto uždavinys jungiantis jį su individualiu projektiniu darbu yra HTML grafinės vartotojo sąsajos kūrimo automatizavimas grafiniu redaktoriumi bei vartotojo sąsajos logiko kūrimas, remiantis jos modeliu.

[11] duomenimis paremta architektūra(angl. data-driven-design)

# 3. PROJEKTAVIMO IR DIEGIMO ĮRANKIO ANALIZĖ

## 3.1. Problema

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

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

## 3.2. Hipotezė

Perėjus nuo tradicinio sprendimo kūrimo tekstiniu redaktoriumi (programavimo) prie duomenimis paremto projektavimo, kai panaudos atvejai kuriami scenarijais grafiniame redaktoriuje, o duomenų infrastruktūra generuojama remiantis modeliu, pavyktų pasiekti didesnį sprendimo vystymo efektyvumą.

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

## 3.3. Darbo tikslai ir uždaviniai

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

Siūlomas architektūrinis modelis turėtų:

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

### 3.4. Darbo metodai ir priemonės

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

**Grafinio modelių redaktorius**

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

**Kodo generatorius**

Posistemė realizuota PHP[12] programavimo kalba ir talpinama dedikuotame serveryje. Kūrimo aplinka NuSpharePhpED 5.9 profesional.

**Duomenų sluoksnio modelis**

Posistemė įgyvendinta Microsoft Visual Studio 2008 programų kūrimo aplinkoje, panaudojant .NET Framework 3.5 SP1. Posistemė įdiegta į Windows 7 operacinę sistemą su MS SQL[13] Server 2008 ir Internet Information Service 7.0.

**Hipertekstinės grafinės vartotojo sąsajos biblioteka**

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

[12] PHP (angl. hypertext preprocessor)
[13] SQL (angl. structured query language)

### 3.5. Panašių sprendimų apžvalga

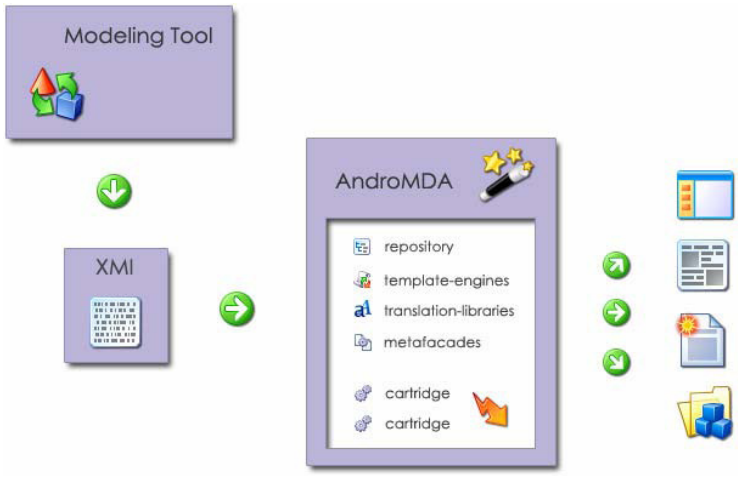
Šiuo metu analogiškų projektavimo įrankių pasaulyje nėra. Tačiau galima apžvelgti panašius įrankius, kurie palaiko modeliu paremtos architektūros[14] principus. Populiariausi ir daugiausiai naudojami yra šie:

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

#### 3.5.1. AndroMDA

AndroMDA yra atviro kodo programinė įranga. Šiuo įrankiu galima generuoti J2EE[15] projektus, realizuotus Java programavimo kalba iš UML[16] modelių. Generuojami žiniatinklio projektai pritaikyti Hibernate [17], EJB[18], Struts[19], Spring[20] ir WebServices[21] technologijoms.

Principinė AndroMDA veikimo schema pavaizduota paveikslėlyje (4 pav. AndroMD veikimas).




4 pav. AndroMD veikimas

[14] modeliu paremta architektūra (angl. model driven engineering)
[15] J2EE (angl. java 2 platform, enterprise edition)
[16] UML (angl. unified modeling language)
[17] Hibernate
[18] EJB (angl. enterprise java beans)
[19] Struts
[20] Spring
[21] Žiniatinklio paslaugos (angl web services)

Programų sistemos generavimas naudojant AndroMDA veiksmų seka:

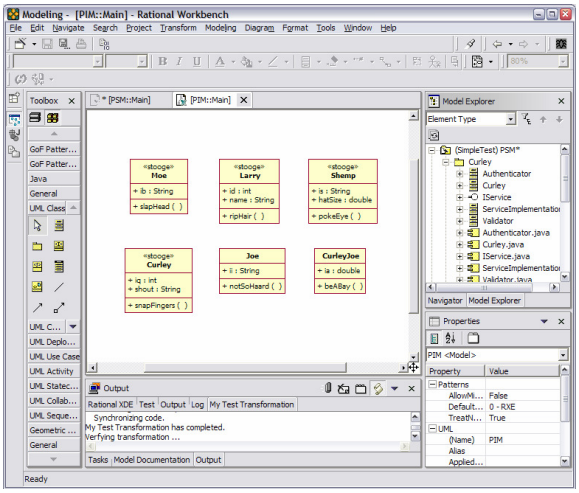
- UML įrankiu (MagicDraw, Poseidon UML, Rational Rose) sukuriamas sistemos nuo platformos nepriklausomas modelis[22] ir eksportuojamas į XMI[23] dokumentus.
- Iškviečiama AndroMDA programa kuri atlieka reikalingas transformacijas.

Transformacijos aprašomos įskiepių[24] sistema. Todėl vartotojas turi galimybę kurti savo transformacijos modelius. Šie modeliai aprašomi šabloniniu principu Java programavimo kalba.

Darbas su įrankiu vyksta terminalo[25] pagalba, todėl valdymas labai nepatogus, tačiau kaip kūrėjai teigia itin lankstus.

### 3.5.2. Rational XDE MDA Toolkit

Rational XDE MDA Toolkit yra „Rational Rose XDE“ papildinys. Tai yra tiesiog modeliavimo įrankio papildinys leidžiantis atlikti kodo generavimą iš UML modelių. Bendrą paketą sudaro modeliavimo įrankis, MDA transformacijų įrankis ir kodo generavimo įrankis (5 pav. Rational XDE MDA Toolkit vaizdas).




5 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.

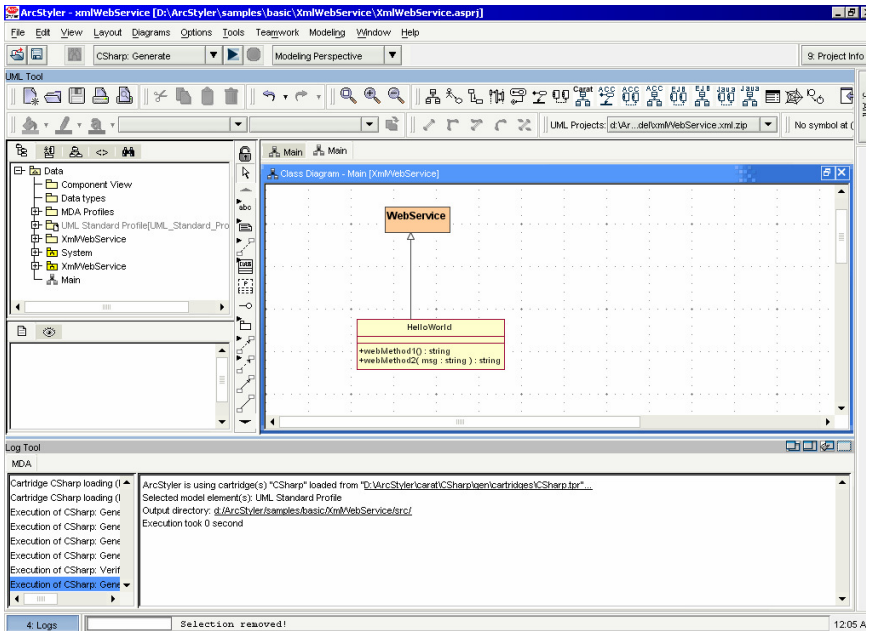
[22] nuo platformos nepriklausantis modelis (angl. platform independent modelį)
[23] XMI (angl. xml metadata interchange)
[24] įskiepis (angl. cartridge)
[25] terminalas (angl. console)

### 3.5.3. Arcstyler 4.0

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

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

- Sumažinti kūrimo proceso laiką. Kūrimo proceso laikas sumažėja dėl naudojamo vizualaus modeliavimo bei kodo generavimo.
- Aukštas kokybės laipsnis – aiškios architektūros reikalavimas, proceso dokumentavimas, iš anksto nustatytas testavimas, atitikimo specifikacijai patikra.
- 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 atgalinės inžinerijos pritaikymas Java programoms.




6 pav. Arcstyler 4.0 vaizdas

### 3.5.4. OpenMDX

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

Pagrindiniai privalumai:

- Atviras kodas.
- Galimybė kurti plečiamas organizacijų sistemas.
- MDA principų taikymas .
- J2SE[26], J2EE, CORBA[27], .NET platformose.
- Modeliavimo įrankių palaikymas (Rational Rose, Poseidon UML, MagicDraw).
- Aspektais paremto programavimo palaikymas.

### 3.5.5. Optimal J

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

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

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

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

[26] J2SE (angl. java 2 standard edition)
[27] CORBA (angl. common object request broker architecture)

Optimal J centre yra projektavimo ir realizavimo karkasai, vadinami šablonais[28]. Optimal J naudoja šablonus visos vykdymo programos kodo generavimui. Šablonai įtraukia geriausius kodo pavyzdžius į J2EE specifikacijas.

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

### 3.5.6. Įvertinimas

Įvertinsime aukščiau aprašytus modeliavimo įrankius pagal šiuos kriterijus:

- PIM palaikymas. Ar yra galimybė sudaryti PIM modelį.
- PSM[29] palaikymas. Ar yra galimybė sudaryti PSM modelį.
- Ar gali generuoti skirtingus PSM. Ar įrankis leidžia generuoti įvairius PSM iš to pačio PIM modelio.
- Modelių integracija. Galima dirbti su keliais modeliais iš kurių generuojama viena programa.
- Programinės įrangos vystymas. Įrankis palaiko programinės įrangos vystymą.
- Bendravimas su kitais įrankiais. Įrankis gali bendrauti su kitais įrankiais, importuoti eksportuoti modelius.
- Galimybė kurti savo transformacijas. Įrankis leidžia kurti savo transformacijas, t.y. nepasitenkinama tik paruoštomis transformacijomis.
- Korektiškumas. Įrankis teikia priemonę patikrinti modelių korektiškumą, ar jie atitinka nustatytas taisykles.
- Išraiškingumas. Įrankis pateikia pakankamai išraiškingas PIM, PSM notacijų priemones, kuriomis galima pilnai išreikšti sistemos modelį.
- Šablonai ir apibendrinimai. Įrankis leidžia kurti modelio elementų šablonus.
- Keitimo[30] palaikymas. Pakeitimai PIM perduodami iš kart į PSM ir atvirkščiai.
- Ryšiai tarp modelių. PIM, PSM ir kodo modeliai išlaiko ryšius.
- Programinės įrangos gyvavimo ciklo palaikymas. Įrankis leidžia palaikyti programinės įrangos kūrimo ciklą (analizė, projektavimas, testavimas, diegimas, palaikymas).

---

[28] šablonas (angl. pattern)
[2929] PSM (angl. platform specific model)
[30] keitimas (angl. refactoring)

- 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.

3 lentelė. UML įrankių įvertinimo rezultatai [5]

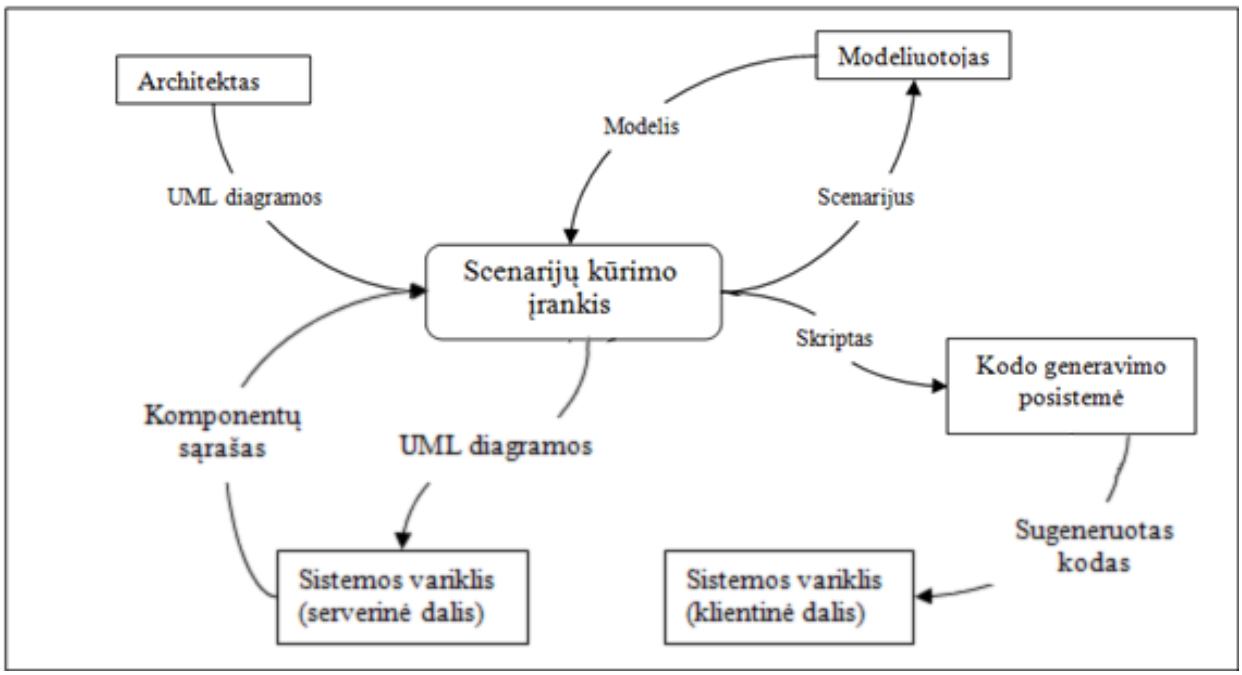
| **Kriterijus** | **Andro MDA** | **Rational XDE MDA Toolkit** | **ArcStyler** | **Open MDX** | **OptimalJ** |
|---|---|---|---|---|---|
| PIM palaikymas | ✓ | ✓ | ✓ | ✓ | ✓ |
| PSM palaikymas | ✓ | ✓ | ✓ | ✓ | ✓ |
| Ar gali generuoti skirtingus PSM | ✓ | ✓ | ✓ | ✓ | × |
| Modelių integracija | × |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

### 3.6. Veiklos sfera ir veiklos pasidalinimas

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 (*7 pav. Bendros sistemos veiklos sfera*), o aprašymas lentelėje.

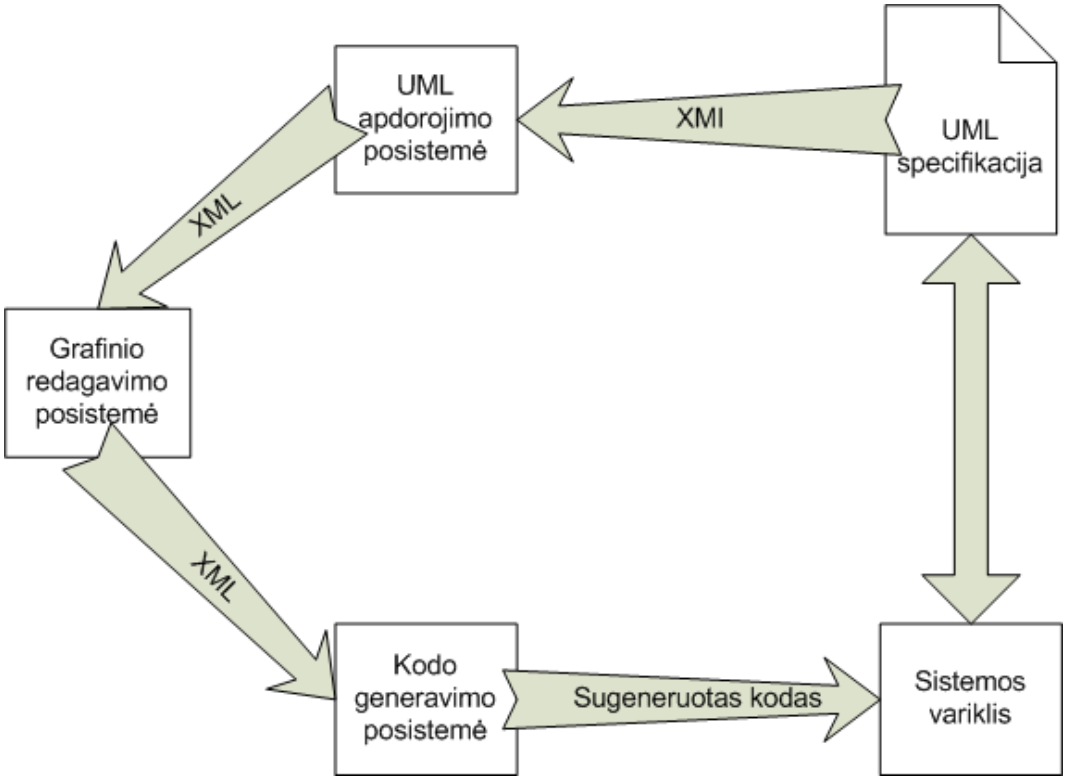



7 pav. Bendros sistemos veiklos sfera

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

Taip atrodo supaprastinta įrankio architektūros schema.




8. pav. Įrankio architektūros diagrama

### 3.6.1. Grafinio redagavimo posistemė

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

### 3.6.2. Kodo generavimo posistemė

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

### 3.6.3. UML apdorojimo posistemė

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

### 3.6.4. Sistemos variklis

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

# 4. HIPERTEKSTINĖS GRAFINĖS VARTOTOJO SĄSAJOS BIBLIOTEKOS PROJEKTAS

Norint patvirtinti arba paneigti magistro tezėse suformuluotą hipotezę, kad perėjus nuo HTML grafinės vartoto sąsajos kūrimo iš atominių elementų prie jos kūrimo iš stambių aukšto abstrakcijos lygmens elementų pavyktų pasiekti didesnį sprendimų vystymo efektyvumą, buvo atliekamas projektinis darbas.

Projektinis darbas susideda iš penkių nuoseklių etapų, kuriuose reikėjo:

1) atilikti HTML architektūros išplėtimo galimybių analizę
2) pasirinkti architektūros įgyvendinimo technologijas
3) apibrėžti naujų grafinės vartotojo sąsajos elementų aibę
4) pasiūlyti naujų grafinės vartotojo sąsajos elementų deklaratyvią notaciją
5) suprojektuoti naujos HTML grafinės vartotojo sąsajos biblioteką
6) eksperimentiškai ištirti siūlomos architektūros efektyvumą

Projektinis darbas sprendžia pagrindinę magistro tezėse suformuluotą problemą – silpnus semantinius ryšius tarp standartinių HTML 4.1 kalbos elementų, dėl kurių didelių HTML sprendimų grafinės vartotojo sąsajos vystymas tampa lėtas ir sudėtingai įgyvendinamas.

## 4.1. Įgyvendinimo technologija

Pirmuoju etapu teko pasirinkti architektūros įgyvendinimo technologiją. Pirmasis uždavinys yra nutarti kokią deklaratyvios sintaksės kalbą naudoti grafinei vartotojo sąsajai aprašyti. Kadangi HTML standartas yra platesnio XML standarto poaibis, o XML yra deklaratyvi kalba, sprendimas pasirinkti XML buvo palankiausias. Jis užtikrino, kad architektūra turi bus paremta W3C standartais bei šią architektūrą bus paprasta architektūra lengvai integruoti į HTML žymenų kalbą

Antras svarbus uždavinys buvo pasirinkti deklaratyvios kalbos analizei tinkamą kalbą, kuri nuskaitytų deklaratyvia kalba aprašytos grafinės vartotojo sąsajos struktūrą ir ją transformuotų į HTML kalbą. Remiantis XML ir HTML kalbų giminingumu, XML analizei buvo pasirinkta ta pati technologija, kuri operuoja HTML elementais – tai JavaScript kalba. Ši scenarijų kalba turi integruotą programavimo sąsają DOM skirtą XML elementams interpretuoti ir jais manipuliuoti. JavaScript kalbos pasirinkimas užtikrino , kad architektūra nekels papildomų reikalavimų HTML vartotojams.

### 4.2. Grafinės vartotojo sąsajos elementų aibė

Antruoju etapu buvo apibrėžta grafinės vartotojo sąsajos elementų aibė.Grafinės vartotojo sąsajos elementai buvo padalinti į dvi grupes: atominius elementus ir konteinerinius elementus. Atominiai elementai yra baigtiniai ir negali savyje talpinti kitų elementų, tuo tarpu konteinerinių elementų paskirtis yra grupuoti atskirus atominius elementus. Atominių elementų aibę sudaro pagrindiniai grafinės vartotojo sąsajos elementai: tekstinis laukas, mygtukas, požymio pasirinkimas, paveikslėlis, komentarinis laukas ir pan. Iš atominių elementų galima formuoti tokius aukštesnio abstrakcijos lygmens elementus kaip žemėlapis ar kt. Konteineriniai elementai yra dialogas, kontekstinis meniu ar vedlys.

HTML grafinės vartotojo sąsajos elementų rinkinį sudaro apie bazinių 20 elementų. Elementus galima komponuoti tarpusavyje ir taip sudarinėti vis naujus atominius elementus.

Grafinės vartotojo elementai pagal jų paskirtį suskirstyti į šias grupes:

- duomenų formos elementai
- struktūriniai elementai
- įvairialypės terpės elementai
- talpinimo elementai
- specializuoti elementai

Kiekvienos grupės elementai turi ne tik panašią paskirtį, bet ir programavimo sąsają.

5 lentelė. Elementai pagal grupes

| Grupė | Pavadinimas | Komponentas |
|---|---|---|
| Duomenų formos komponentas | Textbox | Duomenų įvedimo laukas |
| | Combobox | Sąrašas |
| | Option | Pasirinkimas |
| | Checkbox | Požymis |
| | Button | Mygtukas |
| | FileSelect | Failo pasirinkimas |
| Struktūrinis komponentas | Label | Komentarinis laukas |
| | Link | Nuoroda |
| Įvairialypės terpės komponentas | Image | Paveikslėlis |
| | Svg | Vektorinė grafika |
| | Embed | Įskiepis |
| | Map | Žemėlapis |
| Talpinimo komponentas | Container | Konteineris |
| | Table | Lentelė |
| | WindowBlock | Blokas |
| Specializuotas komponentas | Window | Langas |
| | Menu | Meniu |
| | Wizard | Vedlys |

### 4.3. Deklaratyvi grafinės vartotojo sąsajos elementų notacija

Trečiuoju etapu buvo suformuota deklaratyvi grafinės vartotojo sąsajos elementų notacija XML kalba. Pasinaudojant analize buvo parinktos taisyklės grafinės vartotojo sąsajai aprašyti.

Apačioje pateikiamas deklaratyvus HTML grafinės vartotojo sąsajos elemento aprašas.

```
<Dialog id="search" width="265" height="150" align="left" valign=top" type="modal">
  </Image id="imgFind" image="./search_bw.png" wheight="14" top="2" onclick="{search.focus()}">
  </Label id="lblQuery" text="Paieška:" width="50" top="3" onclick="{search.focus()}">
  </Button id="btnHelp" text="?:" wheight="17" left="1" align="right" onclick="{search.help()}">
  </Button id="btnFind" text="»" wheight="17" left="1" align="right">
  </TextBox id="txtQuery" width="150" height="13" maxlength="25" stateful="true" align="right">
</Dialog>
```

9. pav. Pavyzdinio grafinės vartotojo sąsajos elemento deklaratyvi notacija

Viršuje aprašomas teksto paieškos dialogas, turintis penkis elementus: paveikslėlį, antraštę, du mygtukus bei tekstinį laukelį. Apraše XML elementai atitinka HTML grafinės vartotojo sąsajos elementus, o XML atributai nurodo elementų savybes (`width`, `height`, `align`, `valign`, `top`, `left`, `text` ir `kt.`) bei užregistruoja įvykius (`onclick`) apdorojančius metodus. Aprašas pasižymi glaustumu ir aiškia struktūra bei ateityje gali būti sudaromas grafiniu redaktoriumi.

Deklaratyvus apibrėžimas vykdymo metu yra transformuojamas į imperatyvias JavaScript instrukcijas, kurias tuoj pat įvykdo HTML grafinės vartotojo sąsajos biblioteka.

```
search = new Dialog(DialogType.Modal, {id:"search",width:265,height:150,align:Align.Left, valign:VAlign.Top});
search.AddControl(new Image({id:"imgFind", image:"./search_bw.png", wheight:14, top:2, onClick:function(){
  search.focus()
}}));
search.AddControl(new Label({id:"lblQuery",text:"Paieška:",width:50,top:3,onClick:function(){search.focus()}}))
search.AddControl(new Button({id:"btnHelp",wheight:17,left:1,text:"?", align:Align.Right, onClick:function(){
  search.help()
}}));
search.AddControl(new Button({id:"btnFind", wheight:17, left:1, text:"»", align:Align.Right}));
search.AddControl(new TextBox({id:"txtQuery",width:150,height:13,maxlength:25,stateful:true,align:Align.Right})

search.focus = function() {
  // CUSOM CODE //
}
search.help = function() {
  // CUSOM CODE //
}
search.AppendTo("content");
```

10 pav. Pavyzdinio grafinės vartotojo sąsajos elemento imperatyvi notacija

Tolimesnis HTML grafinės vartotojo sąsajos elemento funkcionalumas plečiamas naudojantis praplėstu HTML grafinės vartotojo sąsajos programavimo modeliu. Šio funkcionalumo užrašymas ateityje taip pat gali būti automatizuotas naudojant grafinį redaktorių.

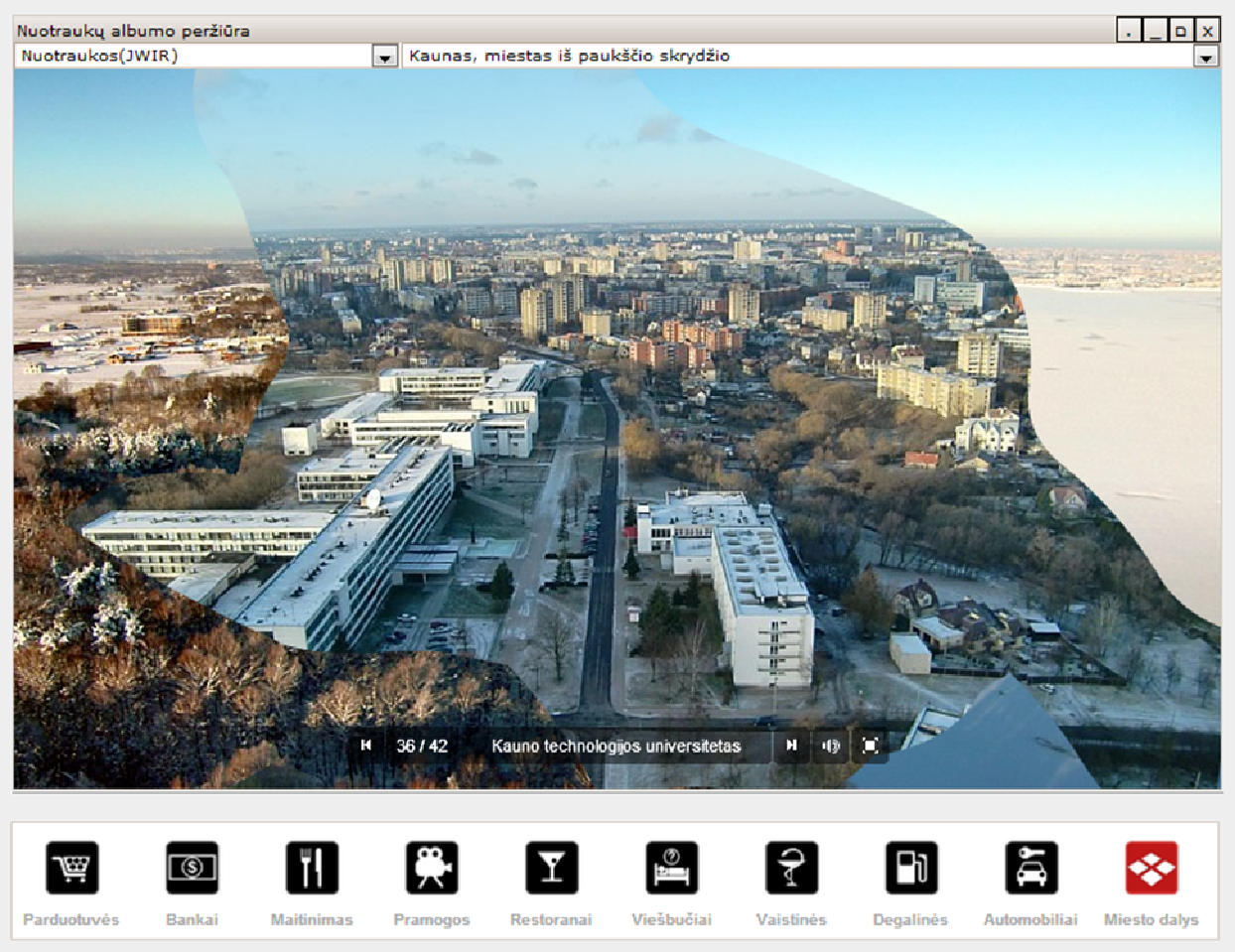
Grafinės vartotojo sąsajos biblioteka sukuria ir ekrane atvaizduoja naująjį elementą.


11 pav. Pavyzdinio grafinės vartotojo sąsajos elemento vizualus vaizdas.

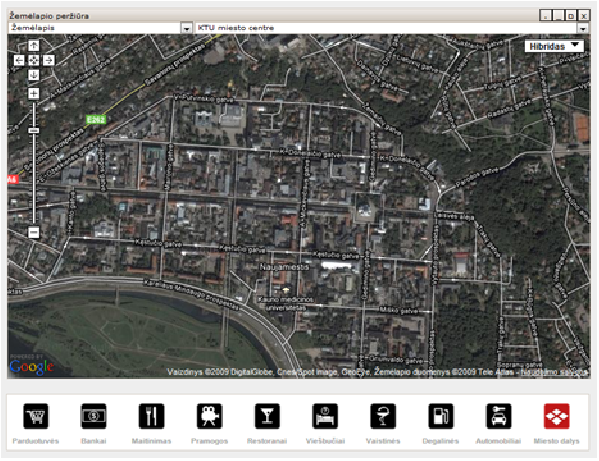
### 4.4. HTML grafinės vartotojo sąsajos biblioteka

Ketvirtuoju etapu buvo suprojektuota HTML grafinės vartotojo sąsajos biblioteka, kurios pagrindinis uždavinys generuoti grafinę vartotojo sąsają. Biblioteka vykdymo metu komponuoja HTML elementus. Biblioteka realizuota objektiškai orientuotu programavimo stiliumi JavaScript kalba bei naudoja standartinę DOM sąsają.

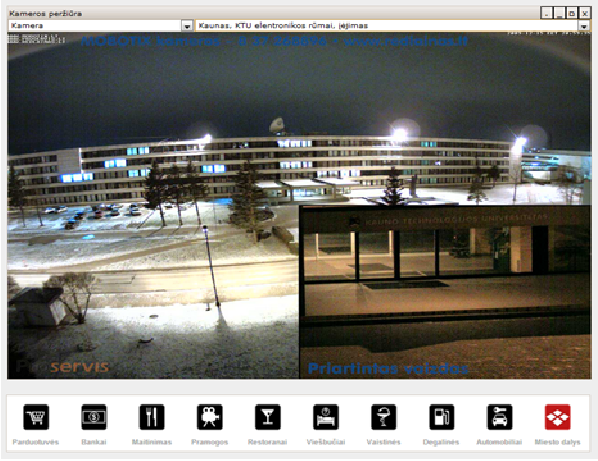
HTML grafinės vartotojo sąsajos biblioteka palengvina sudėtingų sprendimų vystymą.




12 pav. HTML grafinės vartotojo sąsajos biblioteka realizuotas albumas



13 pav. HTML grafinės vartotojo sąsajos biblioteka realizuotas žemėlapis



14 pav. HTML grafinės vartotojo sąsajos biblioteka realizuota tiesioginė vaizdo transliacija

### 4.4.1. Funkciniai reikalavimai

HTML grafinės vartotojo sąsajos biblioteka yra grafinės vartotojo sąsajos funkcijų rinkinys. Ji pagreitina ir palengvina didelių HTML sprendimų įgyvendinimą bei palaikymą.

HTML grafinės vartotojo sąsajos biblioteka atlieka šias pagrindines funkcijas:

- Grafinės vartotojo sąsajos elementų kūrimas
  *Grafinė vartotoj sąsaja susideda iš elementų, kurie aprašomi deklaratyviomis aukšto abstrakcijos lygmens konstrukcijomis, naudojant XML sintaksę.*
- Grafinės vartotojo sąsajos elementų ryšys su duomenimis
  *Elementai turi vieningą sąsają su duomenų šaltiniais:*
  *interneto paslaugomis[31] ir duomenų failais (palaiko XML, CSV, JSON formatus).*
- Grafinės vartotojo sąsajos elementų sąveika
  *Elementai turi vieningą objektiškai orientuotą programavimo sąsają,*
  *pagalba sumažinamas sprendimo sudėtingumas bei didinama vystymo sparta.*
- Grafinės vartotojo sąsajos elementų vidinių būsenų valdymas
  *Elementai turi vidines būsenas, kurios kinta laike ir lemia elemento veikimą.*
- Grafinės vartotojo sąsajos komponentų išvaizdos stilių įgyvendinimas
  *Elementai turi stilių rinkinį, kurio savybes galima keisti.*
  *Palaiko CSS2/CSS3 standartus.*

Biblioteka suteikia architektūrinį apvalką[32] HTML grafinei vartotoj sąsajai kurti, o jos pagrindu gali būti kuriami įvairūs grafinės vartotojo sąsajos elementai:

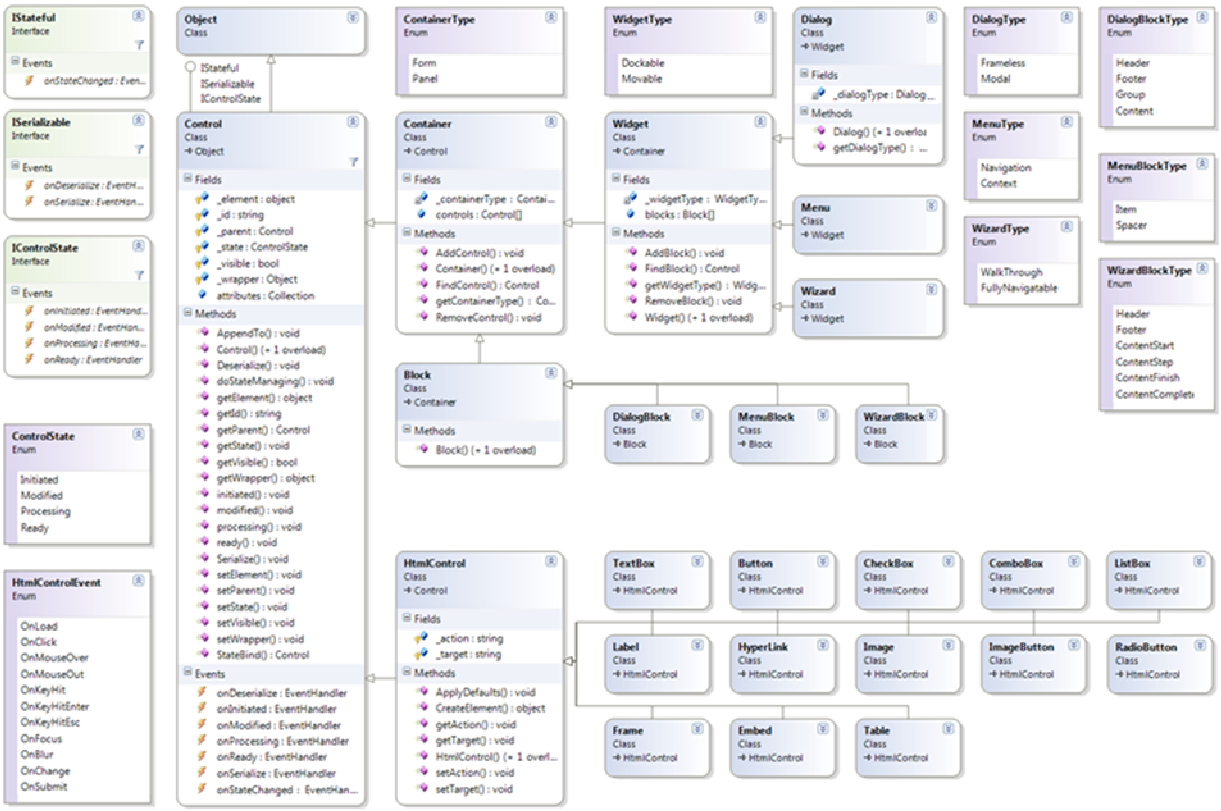
- sudėtingi dialogų langai
- sudėtingi vedliai[33]
- sudėtingi meniu
- sudėtingi įvairialypės terpės[34] elementai

Biblioteka yra geras pagrindas modeliu paremtai architektūrai [35] bei komandomis paremtai architektūrai[36] įgyvendinti. Galimas grafinės vartotojo sąsajos ir jos generavimas iš modelio (pvz. UML, SCRALL), konfigūracijų, scenarijų bei kitų išplėtojimų įgyvendinimas.

[32] architektūrinis apvalkalas (angl. wrapper)
[33] vedlys (angl. wizard)
[34] Įvairialypė terpė (angl. multimedia)

### 4.4.2. Elementų hierarchija

HTML grafinės vartotojo sąsajos elementų rinkinį sudaro apie 20 elementų. Lanksti architektūra leidžia komponuoti elementus tarpusavyje (agregacija), plėsti elementų funkcionalumą bei kurti elementų hierarchijas (paveldėjimas).




15 pav. Grafinių elementų hierarchija

HTML grafinės vartotojo sąsajos bibliotekos bazinis elementas yra `Control` klasė. Ši abstrakti klasė palaiko `IStateful`, `ISerializable` ir `IControlState` sąsajas bei kyla iš abstrakčios `Object` klasės. Iš `Control` klasės kyla dvi abstrakčios klasės: talpinantiems elementams `Container` ir atominiams `HtmlControl` elementams.

Iš abstrakčios `Widget` klasės kyla pagrindinės talpinančios klasės: `Dialog`, `Menu`, `Widget`. Tuo tarpu iš abstrakčios `HtmlCaontrol` klasės kyla pagrindinės atominės klasės: `Textbox`, `Button`, `Checkbox`, `Button`, `Checkbox`, `Button`, `Checkbox`, `Combobox`, `Listbox`, `Label`, `Hyperlink`, `Image`, `ImageButton`, `Radiobutton`, `Frame`, `Embed`, `Table`. Be šių klasių tarp atominių elementų dar yra `Svg`, `Map` ir `FileSelect`.

[35] modeliu paremta architektūra (angl. modelį driven architecture)
[36] komandomis paremta architektūra (angl. command driven architecture)

### 4.4.3. Elementų sąsaja

HTML grafinės vartotojo sąsajos bibliotekos elementai tarpusavyje bendrauja per savybes[37], metodus[38] ir įvykius[39].

#### 4.4.3.1. Savybės

Savybės atspindi grafinių elementų požymius. Jas galima skaityti arba keisti bet kuriuo programos veikimo momentu. Apačioje pateiktos savybės pagal elementų grupes.

6 lentelė. Elementų savybės

| Pavadinimas | Paskirtis | Tipas |
|---|---|---|
| `Align` | horizontalus lygiavimas | `Control.Align` |
| `Valign` | vertikalus lygiavimas | `Control.VAlign` |
| `Width` | plotis (skaičiais/procentais) | `integer/string` |
| `Height` | aukštis (skaičiais/procentais) | `integer/string` |
| `Left` | atstumas nuo dešinės (skaičiais/procentais) | `integer/string` |
| `Top` | atstumas nuo viršaus (skaičiais/procentais) | `integer/string` |
| `Visible` | matomumo požymis | `boolean` |
| `ControlState` | komponento būsena | `Control.State` |
| `Parent` | tėvo objektas | `Container` |
| `Instance` | paties objektas | `object` |
| `Element` | paties HTML elementas | `HTMLElement` |
| `Wrapper` | apvalkalo HTML elementas | `HTMLElement` |
| `DataSource` | duomenų šaltinis | `object` |
| `DataMapping` | duomenų susiejimas | `object` |
| `Move` | judinimo požymis | `boolean` |
| `Resize` | dydžio keitimo požymis | `boolean` |
| `WindowState` | dialogo būsena | `Window.State` |

- Standartiniai elementai (duomenų formos, struktūriniai, įvairialypės terpės elementai)
- Talpinimo elementai
- Specializuoti elementai (talpinimo elementų poaibis)

Kiekviena savybė realizuota dviem metodais. Iškvietus savybės metodą be parametrų gražinama savybės reikšmė, o iškvietus metodą su parametru, nustatoma nauja savybės reikšmė:

- `<elementas>.<savybė>()` – gražinama savybės reikšmė
- `<elementas>.<savybė>(<reikšmė>)` – nustatoma savybės reikšmė

Reikšmių tipai aprašyti aukščiau pateiktoje savybių lentelėje. Tipas naudojamas savybei nustatyti sutampa su reikšmės tipu grąžinamu nuskaitant savybę.

[37] savybė (angl. property)
[38] metodas (angl. method)
[39] įvykis (angl. event)

#### 4.4.3.2. Metodai

Metodais galima atlikti veiksmus su elementu. Juos galima kviesti, bet kuriuo programos veikimo momentu, po elemento sukūrimo. Apačioje pateikti metodai pagal elementų grupes.

7 lentelė. Elementų metodai

| Pavadinimas | Paskirtis | Tipas |
|---|---|---|
| `AppendTo` | prikabinti prie tėvinio HTML elemento | `Container/HTMLElement` |
| `Remove` | pašalinti iš dokumento | `Control.VAlign` |
| `FindParent` | surasti tėvinį elementą | `Container/null` |
| `DataBind` | susieti su duomenų šaltiniu(url, map, f) | `string,object,function` |
| `StateBind` | susieti su įvykiu(state, elem, event) | `Control.State,` |
| `AddControl` | prijungti komponentą | `HtmlControl` |
| `RemoveControl` | pašalinti komponentą | `string` |
| `FindControl` | surasti komponentą | `string` |

- Standartiniai elementai (duomenų formos, struktūriniai, įvairialypės terpės elementai)

- Talpinimo elementai

Kiekvienas elementas turi papildomus metodus, padedančius lengviau atlikti dažnai pasikartojančias operacijas. Šių metodų aprašymas yra dokumentacijoje. Štai keletas pavyzdžių: Combobox – `Fill` bei `Clear` (sąrašui užpildyti ir išvalyti), Textbox – `SelectAll` (tekstui pažymėti), Container – `AddControl`, `RemoveControl` ir `RemoveControl`, Window – `Minimize`, `Maximize` ir `Restore` (lango dydžiui valdyti).

#### 4.4.3.3. Įvykiai

Įvykiai yra iškviečiami įvykus konkrečiam elemento veiksmui (pvz. įdėjus ar pašalinus iš konteinerio, susiejus su duomenų šaltiniu ir kt.). Elementai turi standartinį įvykių rinkinį. Įvykius sukelia pati biblioteka arba vartotojo veiksmai, todėl jie vykdomi automatiškai. Apačioje pateikti įvykiai pagal elementų grupes:

8 lentelė. Elementų įvykiai

| Pavadinimas | Paskirtis |
|---|---|
| `OnPreInit` | komponentas inicializuojamas |
| `OnInitiated` | komponentas inicializuotas |
| `OnModified` | komponentas pakeistas po sukūrimo |
| `OnProcessing` | komponentas apdoroja duomenis |
| `OnFinished` | komponentas baigė apdoroti duomenis |
| `OnStateBinding` | komponento būsena susiejama su HMTL įvykiu |
| `OnDataBindBefore` | komponentas yra susiejamas su duomenų šaltiniu (prieš) |
| `OnDataBindAfter` | komponentas yra susiejamas su duomenų šaltiniu (po) |
| `OnAdd` | komponentas įdedamas į konteinerį |
| `OnAttach` | komponentas įdedamas į dokumentą |
| `OnMove` | komponentas yra velkamas |
| `OnMoveStart` | komponentas yra pradedamas vilkti |
| `OnMoveStop` | komponentas yra baigiamas vilkti |
| `OnResize` | komponentas yra didinamas/mažinamas |
| `OnResizeStart` | komponentas yra pradedamas didinti/mažinti |
| `OnResizeStop` | komponentas yra baigiamas didinti/mažinti |

- Standartiniai elementai (duomenų formos, struktūriniai, įvairialypės terpės elementai)
- Talpinimo komponentai
- Specializuoti komponentai (talpinimo elementų poaibis)

### 4.4.4. Vidinės būsenos

Grafinės vartotojo sąsajos bibliotekos žiniatinkliui elementai turi vidines būsenas kurios keičiasi visą programos vykdymo laikotarpį, gali būti išsaugotos ir atkurtos.

Elementų būsenos yra keičiamos nustatant savybes.

### 4.4.5. Duomenys

Grafinės vartotojo sąsajos bibliotekos žiniatinkliui elementai turi bendrą sąsajos su duomenimis modelį, kuriam priklauso šie metodai ir įvykiai:

9 lentelė. Elementų duomenys

| Pavadinimas | Paskirtis |
|---|---|
| DataBind | elementą susieja su duomenų šaltiniu (`url, map, callback`) |
| OnDataBindBefore | įvykis prieš elemento susiejimo su duomenų šaltiniu |
| OnDataBindAfter | įvykis prieš elemento susiejimo su duomenų šaltiniu |

- Metodas
- Įvykis

Bibliotekos elementai turi sąsają su duomenų šiais duomenų šaltiniais ir formatais:

- `interneto paslaugomis (SOAP)`
- `duomenų failais (XML, CSV, JSON)`

### 4.4.6. Stilių šablonai

Grafinės vartotojo sąsajos bibliotekos žiniatinkliui elementai palaiko CSS stilius. Stilių apibrėžimas yra atskirtas nuo elemento logikos aprašymo.

Apačioje pateikti elementų stiliai pagal elementų grupes.

10 lentelė. Elementų stilių šablonai

| Komponentas | Komentaras | Stilius |
|---|---|---|
| Textbox | Duomenų įvedimo laukas | `textboxAPI` |
| Combobox | Sąrašas | `comboboxAPI` |
| Option | Pasirinkimas | `optionAPI` |
| Checkbox | Požymis | `checkboxAPI` |
| Button | Mygtukas | `buttonAPI` |
| FileSelect | Failo pasirinkimas | `fileSelectAPI` |
| Label | Komentarinis laukas | `labelAPI` |
| Link | Nuoroda | `lintAPI` |
| Picture | Paveikslėlis | `imageAPI` |
| SVGImage | Vektorinė grafika | `svgImageAPI` |
| Embed | Įskiepis | `objAPI` |
| Map | Žemėlapis | `mapAPI` |
| Container | Konteineris | `containerAPI` |
| Table | Lentelė | `tableAPI` |
| MenuBlock<br>WindowBlock<br>WizardBlock | Blokas | `containerAPI, menuBlockAPI`<br>`containerAPI, windowBlockAPI`<br>`containerAPI, wizardBlockAPI` |
| Window | Langas | `containerAPI, windowAPI` |
| Menu | Meniu | `containerAPI, windowAPI` |
| Wizard | Vedlys | `containerAPI, windowAPI` |

Stiliai yra laisvai keičiami, palaiko hierarchijas, juos nesudėtinga keisti programai veikiant bei pritaikyti atskirai sprendimo realizacijai.

### 4.4.7. Sprendimo aplinka

Grafinės vartotojo sąsajos bibliotekos žiniatinkliui veikimas organizuojamas Javascript sakiniais, kurie generuojami iš deklaratyvių XML aprašų ir įrašomi *.HTML arba *.JS failuose.

Bazinė HTML dokumento struktūra, į kurią integruojamas sprendimas pateikta apačioje.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <title>Core: Presentation Layer</title>
  <script type="text/javascript" src="./com/helpers.js"></script>
  <script type="text/javascript" src="./com/core.js"></script>
  <script type="text/javascript" src="./com/core-ext.js"></script>
  <script type="text/javascript" src="./com/core-data.js"></script>
  <style type="text/css">
    html, body{ margin: 0px; padding: 0px; width: 100%; height:100%; overflow: hidden }
  </style>
</head>

<body>
  <div id="content" style="width: 100%; height: 100%"></div>
  <script type="text/javascript">
    // <instrukcijos grafinei vartotojo sąsajai kurti>
  </script>
</body>
</html>
```

16 pav. Grafinės vartotojo sąsajos integravimo aplinka

■ - reikalingų bibliotekų prijungimas

■ - instrukcijos grafinės vartotojo sąsajos bibliotekai žiniatinkliui

Žinant sprendimo kūrimo aplinką ir įsisavinus grafinės vartotojo sąsajos žiniatinkliui bibliotekos elementų galimybes, jų savybes ir metodus, bet kokį funkcionalumą galima papildomai plėtoti imperatyviomis (JavaScript) konstrukcijomis.

# 5. PROJEKTAVIMO IR DIEGIMO ĮRANKIO PROJEKTAS

## 5.1. Architektūros tikslai ir apribojimai

Kuriamos sistemos architektūros tikslai:

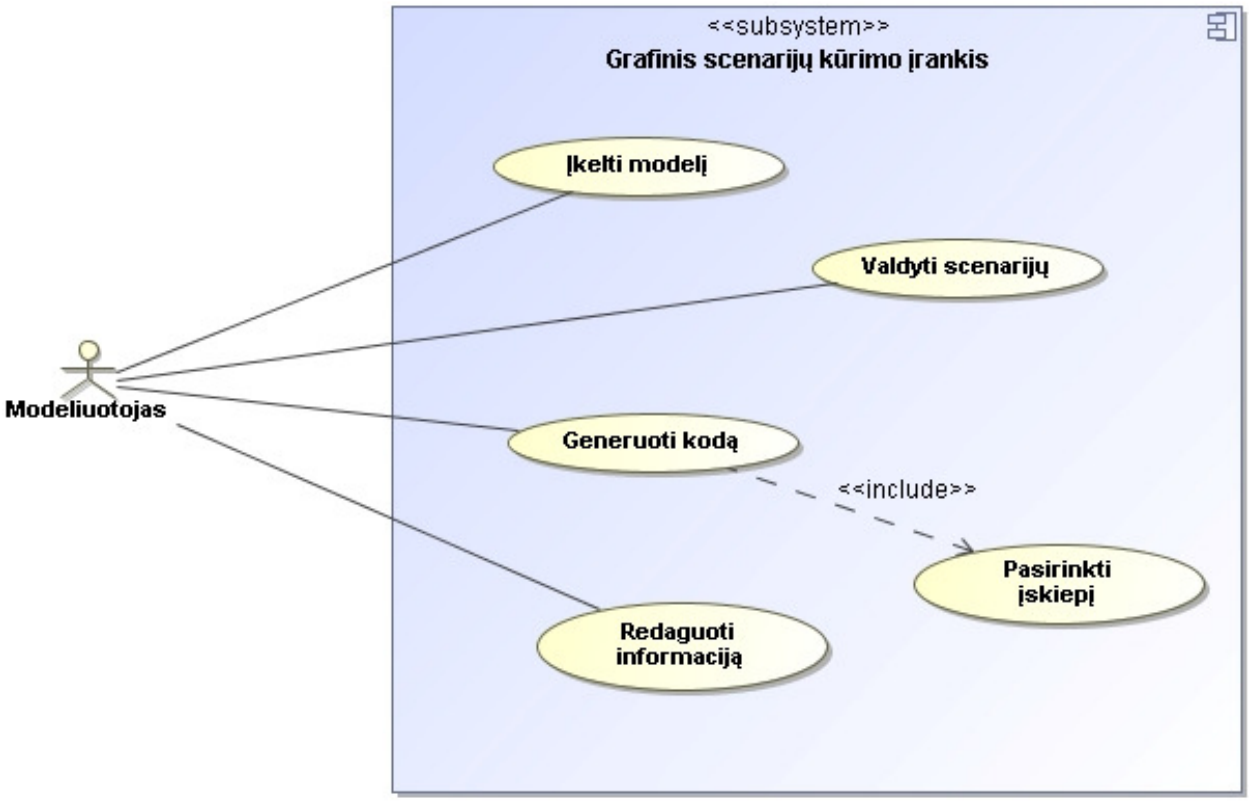
- Sistema galima naudotis visais kompiuteriais su interneto naršykle ir internetu (standartais paremta sistema)
- Duomenimis paremta architektūra.
  (įkeliame UML diagramas ir pagal tai kuriame scenarijus grafiniame redaktoriuje)
- Nepriklausoma taikymo sritis.
  (kiekviena taikymo sritis aprašoma kaip įskiepis kodo generavimo posistemėje).

Kuriamos sistemos apribojimai:

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

### 5.1.1. Panaudojimo atvejai

Projektavimo įrankio pagrindiniai panaudos atvejai pateikti paveiksle (17), o jų aprašymai lentelėse (**Error! Reference source not found. – Error! Reference source not found.**).




17 pav. Panaudos atvejų vaizdas

11 lentelė. Panaudos atvejis „Įkelti modelį“

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

12 lentelė. Panaudos atvejis “Valdyti scenarijų”

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

13 lentelė. Panaudos atvejis „Generuoti kodą”

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

14 lentelė. Panaudos atvejis „Pasirinkti įskiepį”

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

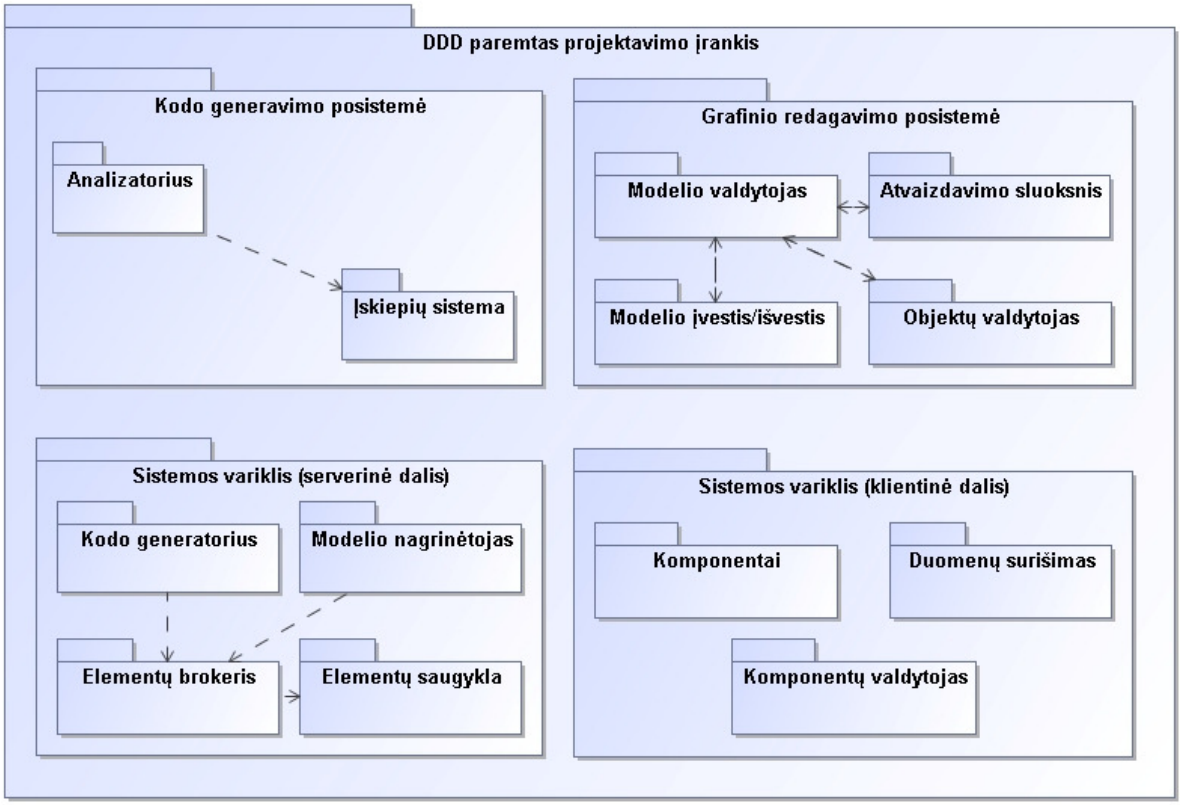
15 lentelė. Panaudos atvejis „Redaguoti informaciją”

| **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. |

### 5.1.2. Sistemos statinis vaizdas

Sistema suskirstyta į šiuos paketus (*18 pav. Projektavimo įrankio paketų diagrama*):

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




18 pav. Projektavimo įrankio paketų diagrama

#### 5.1.2.1. Grafinio redagavimo posistemė

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

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

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

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

#### 5.1.2.2. Kodo generavimo posistemė

„Analizatoriaus paketas“ atsakingas už scenarijaus perskaitymą ir parametrų surinkimą.

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

#### 5.1.2.3. Sistemos variklio serveri dalis

„Kodo generatoriaus“ paketas atsakingas už išeities teksto failų generavimą.

„Modelio nagrinėtojo“ paketas atsakingas už modelio teisingą nuskaitymą.

„Elementų brokerio“ paketas atsakingas už modelio transformaciją į komponentus.

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

#### 5.1.2.4. Sistemos variklio kliento dalis

„Komponentai“ paketas atsakingas už komponentų rinkinį it funkcionalumą.

„Duomenų surišimas“ paketas atsakingas už duomenų tiekimą komponentams.

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

### 5.1.3. Diegimo aplinka

Sistemą turi sudaryti dvi atskiros dalys bendraujančios sąsajomis – kliento pusėje[40] bei serverio pusėje[41]. Šios dalys veikia skirtingose aplinkose. Kliento dalis veikia vartotojo kompiuteryje interneto naršyklėje, čia vyksta informacijos atvaizdavimas bei įvedimas. Serverio dalis yra nutolusiame kompiuteryje (serveryje), čia saugomi sistemos duomenys, atliekami kodo generavimo veiksmai.

[40] kliento pusėje (angl. client-side)
[41] serverio pusėje (angl. server-side)

Reikalavimai vartotojo programinei įrangai pateikiami lentelėje „16 lentelė. Reikalavimai vartotojo programinei įrangai“, minimalūs reikalavimai vartotojo techninei įrangai pateikiami lentelėje „17 lentelė. Minimalūs reikalavimai vartotojo techninei įrangai“. Reikalavimai serverio programinei įrangai pateikiami lentelėje „18 lentelė. Reikalavimai serverio programinei įrangai“, minimalūs reikalavimai serverio techninei įrangai pateikiami lentelėje „19 lentelė. Minimalūs reikalavimai serverio techninei įrangai“.

16 lentelė. Reikalavimai vartotojo programinei įrangai

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

17 lentelė. Minimalūs reikalavimai vartotojo techninei įrangai

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

18 lentelė. Reikalavimai serverio programinei įrangai

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

19 lentelė. Minimalūs reikalavimai serverio techninei įrangai

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

# 6. TYRIMO DALIS

Suprojektavus ir realizavus sistemos architektūrą reikia įvertinti siūlomo sprendimo kokybę bei atlinkti galimų patobulinimų analizę. Šioje dalyje apžvelgiamos kokybės tobulinimo galimybės, siūlomi ir realizuojami patobulinimai.

Svarbiausi programinės sistemos tyrimo objektai yra:

- sistemos automatizavimas
- integravimas į programinės įrangos kūrimo ciklą
- pritaikymas skirtingoms žiniatinklio technologijoms

## 6.1. Įvadas

Realizavus grafinės vartojo sąsajos žiniatinkliui biblioteką pagreitėjo didelių HTML sprendimų vystymas ir palaikymas, o pasiūlytas architektūrinis modelis:

- paspartinto grafinės vartotojo sąsajos kūrimą, nes operuojant stambesniais aukšto abstrakcijos lygmens elementais atsiranda mažiau ryšių tarp elementų.
- pagerino grafinės vartoto sąsajos architektūrą, nes stambesnius aukšto abstrakcijos lygmens elementus lengviau valdyti.
- užtikrino mažesnę klaidų tikimybę bei atitikimą standartams, nes stambesnių aukšto abstrakcijos lygmens elementų funkcionalumas būtų ištestuotas viduje.

Taip pat buvo pasiekti analizės etape išsikelti tikslai:

- W3C standartais paremtą architektūrą (užtikrina greitą modelio adaptavimą)
- lengvą integraciją į HTML žymenų kalbą
  (XML ir HTML yra giminingos kalbos)
- papildomų reikalavimų nekėlimą HTML vartotojams
  (vartotojams nereikia įdiegti papildomų įskiepių)

Šiame skyriuje detaliau apžvelgiamos sukurto architektūrinio modelio patobulinimo galimybės. Plačiau analizuojamos integruoto grafinės vartotojo sąsajos kūrimo iš modelio, automatizavimo bei analogiškos grafinės vartotojo sąsajos generavimo skirtingoms technologijoms galimybė.

### 6.2. Automatizuotas grafinės vartotojo sąsajos kūrimas remiantis modeliu

Grafinės vartotojo sąsajos bibliotekos topologiją aprašius modeliavimo kalba, remiantis modeliu ją galima generuoti pagal duomenis.

Vietoj to, kad kiekvieną kartą aprašyti grafinę vartotojo sąsają skirtingiems panaudojimo atvejams, ji galėtų būti automatiškai sugeneruojama atsižvelgiant į duomenis.

Norint įgyvendinti šį patobulinimą, reikia susieti duomenis su grafinės vartotojo sąsajos elementais ryšiais – t.y. suteikti elementams semantinę prasmę.

Modelio atvaizdavimui parankiausia naudoti UML (Unified Modeling Language) standartą ir populiarų UML redaktorių kaip MagicDraw arba IBM Rational Rose. Tuomet pakaktų turėti klasių diagramą su papildomomis notacijomis ir būtų galima pasinaudoti vienu iš MDA[42] privalumų – programinio kodo generavimą, remiantis modeliu.

Siūlomas patobulinimas turi keletą užtikrintų privalumų:

- automatizuoja grafinės vartotojo sąsajos kūrimą
- gerina grafinės vartoto sąsajos architektūrą (didėja daugkartinis panaudojimas)
- užtikrino mažesnę klaidų tikimybę (nei rašant kodą konkrečiam pritaikymo atvejui)

Įgyvendinus siūlomą patobulinimą grafinės vartotojo sąsajos žiniatinklyje kūrimas taptų dar paprastesnis ir greitesnis.

[42] MDA (angl. model driven architecture)

### 6.3. Integruotas grafinės vartotojo sąsajos kūrimas redaktoriumi

Grafinės vartotojo sąsajos bibliotekos topologiją atvaizdavus modeliavimo kalba, grafinę vartotojo sąsają galima būtų kurti integruotu redaktoriumi.

Vietoj to, kad aprašyti grafinę vartotojo sąsają teksto redaktoriumi, ji būtų kuriama grafiniu redaktoriumi, kuris padėtų įgyvendinti panaudojimo atvejus ir dalį elementų logikos.

Norint įgyvendinti šį patobulinimą, reikia sukurti modelį, suteikti elementams semantinę prasmę bei sukurti grafinį redaktorių, kuris naudotų šį modelį ir jo pagalba pateiktų galimų elementų ir veiksmų su jais rinkinį dizaineriui.

Modelio atvaizdavimui parankiausia naudoti UML (Unified Modeling Language) standartą, klasių diagramų pavidalu su papildomomis notacijomis atvaizduotas modelis. Tada būtų galima pasinaudoti grafinės vartotojo sąsajos kūrimu su grafiniu redaktoriumi privalumais.

Siūlomas patobulinimas turi keletą aiškių privalumų:

- automatizuoja grafinės vartotojo sąsajos kūrimą
- integruoja grafinės vartotojo sąsajos kūrimą į programinės įrangos kūrimo ciklą
- užtikrino mažesnę klaidų tikimybę (nei rašant kodą konkrečiam pritaikymo atvejui)

Įgyvendinus siūlomą patobulinimą grafinės vartotojo sąsajos žiniatinklyje kūrimas dar lengviau integruotųsi į programinės įrangos kūrimo ciklą.

### 6.4. Analogiškos grafinės vartotojo sąsajos kūrimas skirtingomis technologijoms

Aprašius grafinės vartotojo sąsajos bibliotekos topologiją modeliavimo kalba, grafinę vartotojo sąsają galima atvaizduoti skirtingomis technologijomis.

Vietoj to, kad aprašyti grafinę vartotojo sąsają atskirai vienai ar kitai technologijai, ją galima kurti transformuoti į pasirinktą technologiją.

Norint įgyvendinti šį patobulinimą, reikia sukurti grafinės vartotojo sąsajos bibliotekas skirtingoms technologijomis bei naudoti kodo generatorius (transliatorius) universaliai iš modelio sukurtai grafinei vartotojo sąsajai į šias technologijas transformuoti.

Grafinės vartotojo sąsają galima atvaizduoti šiomis technologijomis:

- HTML/Javascript/CSS
- Flash/Flex
- Java Servlet

Siūlomas patobulinimas turi keletą privalumų:

- nuo aprašymo nepriklausoma panaudojimo technologija
- galimybė kurti analogišką grafinę vartotojo sąsają skirtingoms technologijomis

Įgyvendinus siūlomą patobulinimą analogiškos grafinės vartotojo sąsajos atvaizdavimas taptų įmanomas keliomis skirtingomis technologijomis.

# 7. HIPERTEKSTINĖS GRAFINĖS VARTOTOJO SĄSAJOS BIBLIOTEKOS EKSPERIMENTAS

Suprojektavus, realizavus ir ištyrus siūlomą architektūrą būtina įvertinti siūlomo efektyvumą. Šioje dalyje atliekamas magistratūros studijų metu sukurtos ir įdiegtos programinės įrangos bei jos patobulinimų eksperimentinis tyrimas.

## 7.1. Matuojamos metrikos

Kad atliktas eksperimentas atspindėtų realią situaciją, būtina išsirinkti aktualias eksperimento metrikas. Pasirenkant metrikas reikia atsižvelgti į sprendimo sudėtingumo matus, kurie lemia vystymo ir palaikymo spartą – jie atspinti situaciją, kurioje kiekvieną dieną atsiduria interneto sprendimų kūrėjai.

Pasiūlytai architektūros efektyvumui įvertinti bus naudojamos šios metrikos:

- kuriamų loginių elementų ir ryšių tarp šių elementų kiekis
- parašyto programinio kodo apimtis

Abi metrikos yra objektyvūs matai, įvertinantys kiek pastangų reikia programuotojui, kaip sparčiai įmanoma vystyti sprendimą ir kokia yra klaidų tikimybė naudojant šią architektūrą.

Abi metrikos yra nesudėtingai pamatuojamos ir gali būti lyginamos su analogiškomis tradicinės grafinės vartotojo sąsajos, paremtos HTML technologija kūrimo metrikomis.

## 7.2. Eksperimento objektai

Eksperimento patikimumui užtikrinti, būtina tinkamai pasirinkti eksperimente tiriamus panaudos atvejus.

Eksperimentas bus atliekamas su dvejais skirtingo sudėtingumo elementais:

- trivialus elementas (paieškos dialogas)
- sudėtingas elementas (žemėlapio dialogas)

Tikimasi, kad šie skirtingo sudėtingumo elementai išryškins siūlomos architektūros stipriąsias puses ir tinkamiausius panaudojimo atvejus.

### 7.2.1. Trivialus elementas

Pirmam siūlomos architektūros eksperimentui naudojamas trivialus paieškos dialogo, elementas, turintis penkis baigtinius elementus: paveikslą, antraštę, du mygtukus ir teksto lauką.

Dialogo elementai aprašomi aukšto abstrakcijos lygmens deklaratyvia XML sintakse.

```
<Dialog id="search" width="285" height="150" align="left" valign=top" type="modal">
  </Image id="imgFind" image="./search_bw.png" wheight="14" top="2" onclick="{search.focus()}">
  </Label id="lblQuery" text="Paieška:" width="50" top="3" onclick="{search.focus()}">
  </Button id="btnHelp" text="?:" wheight="17" left="1" align="right" onclick="{search.help()}">
  </Button id="btnFind" text="»" wheight="17" left="1" align="right">
  </TextBox id="txtQuery" width="150" height="13" maxlength="25" stateful="true" align="right">
</Dialog>
```

19 pav. Trivialaus grafinės vartotojo sąsajos elemento eksperimento deklaratyvi notacija.

Grafinę vartotojo sąsają sudaro 6 elementai: 1 konteineris ir 5 baigtiniai elementai. Maksimalus juos jungiančių loginių ryšių skaičius gali būti $n \cdot (n-1) = 6 \cdot 5 = 30$ ryšių. Taigi, programuojant šio komponento funkcionalumą gali tekti palaikyti iki 30 bendravimo kanalų, kurie lems elemento būseną ir elgseną.

Biblioteka transformuoja deklaratyvų XML aprašą į HTML standarto elementus.

```
<div id="search_18315564" width="285px" height="auto" style="border-width: 1px; position:
absolute; float: left; width: 285px; height: 17px; opacity: 1; z-index: 0; margin-left: 0px;
margin-right: 0px; left: 0px; right: auto; margin-top: 0px; bottom: 0px; top: auto; "
class="widgetAPI">
  <img height="14px" width="14px" id="search_18315564_imgFind" style="margin: 2px 0px 0px 2px;
position: relative; float: left; height: 14px; width: 14px; opacity: 1; z-index: 0; display:
block; left: 0px; right: auto; top: 0px; bottom: auto;" class="imageAPI"
src="./core/img/search_bw.png"/>
  <span id="search_18315564_lblQuery" width="50px" height="auto" style="margin: 3px 0px 0px;
position: relative; float: left; width: 50px; height: auto; opacity: 1; z-index: 0; display:
block; left: 0px; right: auto; top: 0px; bottom: auto;" class="labelAPI">Paieška:</span>
  <input type="button" height="17px" width="17px" id="search_18315564_btnClose" title="Užverti"
style="margin: 0px; position: relative; float: right; height: 17px; width: 17px; opacity: 1; z-
index: 0; display: block; right: 0px; left: auto; top: 0px; bottom: auto;" class="buttonAPI"
value="x"/>
  <input type="button" height="17px" width="17px" id="search_18315564_btnFind" title="Ieškoti"
style="margin: 0px; position: relative; float: right; height: 17px; width: 17px; opacity: 1; z-
index: 0; display: block; right: 0px; left: auto; top: 0px; bottom: auto;" class="buttonAPI"
value="»"/>
  <input type="text" height="13px" width="150px" id="search_18315564_txtQuery" title="Įveskite
paieškos frazę" style="position: relative; float: right; width: 150px; height: 13px; opacity: 1;
z-index: 0; display: block;" class="textboxAPI" maxlength="25"/>
</div>
```

20 pav. Trivialaus grafinės vartotojo sąsajos elemento eksperimento imperatyvi notacija.

Šiame sudėtingame elemente 6 XML elementai tiesiogiai atvaizduojami į 6 HTML elementus, todėl elementų kiekio ir loginių ryšių kiekio atžvilgiu abi realizacijos yra vienodos. Tuo tarpu pagal kodo apimtį, grafinės vartotojo sąsajos biblioteka žiniatinkliui šiuo atveju reikalauja 490 simbolių apibrėžimo, tuo tarpu standartinis HTML žymėjimas 1515 simbolių.

20 lentelė. Trivialaus grafinės vartotojo sąsajos elemento eksperimento rezultatai

| Metodas | Elementų kiekis | Loginių ryšių kiekis | Programos kodo apimtis |
|---|---|---|---|
| HTML 4.01 standartas | 6 | 30 | 1515 |
| Grafinės vartotojos sąsajos biblioteka žiniatinkliui | 6 | 30 | 490 |

### 7.2.2. Sudėtingas elementas

Antrajam siūlomos architektūros eksperimentui naudojamas sudėtingas žemėlapio dialogo, elementas, turintis tik vieną baigtinį elementą – išplėstą Google Maps žemėlapį.

Biblioteka sukuria dialogą iš šio aukšto abstrakcijos lygmens deklaratyvaus XML aprašo.

```
<Dialog id="mdialog" width="285" height="150" align="left" valign=top" type="modal">
  </Map id="embMedia" wheight="100%">
</Dialog>
```

21 pav. Sudėtingo grafinės vartotojo sąsajos elemento eksperimento deklaratyvi notacija.

Grafinę vartotojo sąsają sudaro 2 elementai: 1 konteineris ir 1 baigtinis elementas. Maksimalus juos jungiančių loginių ryšių skaičius gali būti $n \cdot (n-1) = 2 \cdot 1 = 2$ ryšiai.

Taigi, programuojant šio komponento funkcionalumą gali tekti palaikyti iki 2 bendravimo kanalų, kurie lems elemento būseną ir elgseną.

Dialogo elementai aprašomi aukšto abstrakcijos lygmens deklaratyvia XML sintakse.

```
<div id="mdialog_181142267" width="400px" height="auto" style="border-width: 1px; position:
absolute; width: 400px; height: 234px; opacity: 1; z-index: 0; margin-left: -200px; left: 50%;
right: auto; margin-top: -117px; top: 50%; bottom: auto;" class="widgetAPI ui-draggable ui-
resizable" aria-disabled="false">
  <div id="mdialog_181142267_media" width="100%" height="100%" style="overflow: auto; position:
relative; float: left; width: 100%; height: 200px; opacity: 1; z-index: 0;"
class="dialogBlockAPI">
    <div id="media_embMedia" width="100%" height="100%" style="margin: 0px; overflow: hidden;
position: relative; float: left; height: 100%; width: 100%; opacity: 1; z-index: 0; display:
block; background-color: rgb(229, 227, 223); left: 0px; right: auto; top: 0px; bottom: auto;">
      <div style="overflow: hidden; position: absolute; left: 0px; top: 0px; width: 100%;
height: 100%;">
...
...
...
      <div style="border-style: none none solid solid; border-color: -moz-use-text-color -moz-
use-text-color rgb(255, 0, 0) rgb(255, 0, 0); border-width: 0px 0px 2px 2px; width: 6px; height:
4px; line-height: 1px; font-size: 1px;"/>
      </div>
    </div>
  </div>
</div>
```

22 pav. Sudėtingo grafinės vartotojo sąsajos elemento eksperimento imperatyvi notacija

Šiame sudėtingame elemente 2 XML elementai tiesiogiai atvaizduojami į 115 HTML elementų, todėl įprastas HTML sprendimas sugeneruoja $n \cdot (n-1) = 115 \cdot 114 = 13110$ galimų loginių ryšių. Tuo tarpu pagal kodo apimtį, grafinės vartotojo sąsajos biblioteka žiniatinkliui šiuo konkrečiu atveju reikalauja 115 simbolių apibrėžimo ir 2 loginiai ryšiai, tuo tarpu standartinis HTML žymėjimas net 23508 simbolių ir net 13110 galimų loginių ryšių.

21 lentelė. Sudėtingo grafinės vartotojo sąsajos elemento eksperimento rezultatai

| Metodas | Elementų kiekis | Loginių ryšių kiekis | Programos kodo apimtis |
|---|---|---|---|
| HTML 4.01 standartas | 123 | 13110 | 23508 |
| Grafinės vartotojos sąsajos biblioteka žiniatinkliui | 2 | 2 | 115 |

# 8. PROJEKTAVIMO IR DIEGIMO ĮRANKIO EKSPERIMENTAS

## 8.1. Matuojamos metrikos

Kad atliktas eksperimentas atspindėtų realią situaciją, būtina išsirinkti aktualias eksperimento metrikas.

Pasiūlytai architektūros efektyvumui įvertinti bus naudojamos šios metrikos:

- elementų sukūrimą greitis

## 8.2. Eksperimento objektai

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

### 8.2.1. Pirmoji užduotis (mažas sudėtingumas)

Sumodeliuokite paieškos komponentą, kurį sudaro teksto įvedimo laukas ir paieškos vykdymo mygtukas. Nuspaudus jį, tekstas perduodamas serveriui ir rodomas „Results“ langas.

### 8.2.2. Antroji užduotis (vidutinis sudėtingumas)

Sumodeliuokite meniu komponentą, kuriame rodomi trys mygtukai. Nuspaudus pirmąjį 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 sistemos ir įjungiamas „Home“ langas. Kitu atveju rodomas tekstas “Login“, o jį paspaudus atsidaro prisijungimo langas „Login“.

### 8.2.3. Trečioji užduotis (aukštas sudėtingumas)


23 pav. Trečiojo eksperimento objekto grafinė vartotojo sąsaja

Sumodeliuokite prisijungimo langą, kuris atrodytų panašiai į „23 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 „Tokio vartotojo nėra“. Išvalomi abu įvedimo laukai. Jei vartotojas egzistuoja, bet netinka slaptažodis parodomas pranešimas „Blogas slaptažodis“. Ištrinamas slaptažodžio įvedimo laukas, o vartotojo lauke paliekamas. Įvedus teisingus prisijungimo duomenis įjungiamas langas „Secret“.

### 8.2.4. Aprašymas

Projektavimo įrankio įvertinimui buvo atliktas bandymas. Komponentams sukurti grafiniai modeliai ir rašomas programinis kodas. Tai padėjo įsitikinti, kiek kodo generavimas iš grafinio modelio greitesnis už kodo rašymą. Buvo parinktos 3 skirtingo sunkumo užduotys.

### 8.2.5. Rezultatai

Išanalizavus surinktus duomenis buvo nustatyta, kad nepatyręs, žinių apie taikymo sritį neturintis programuotojas mažo sudėtingumo užduotį programuojant įvykdė per 33 minutes, tuo tarpu modeliuojant jis užtruko vos 4 minutes. Jis naują užduotį įgyvendino 8,25 karto greičiau.

Vidutinio sudėtingumo užduotį programuotojas atliko per 21 minutę. Modeliuojant grafiškai šią užduotį įgyvendino per 4 minutes. Pagreitėjimas šiuo atveju 5,25 karto.

Trečia užduotis rašant programinį kodą buvo įvykdyta per 37 minutes, o modeliuojant per 5 minutes. Pagreitėjimas 7,4 karto.

### 8.2.6. Išvados

Atlikus eksperimentą pastebime, kad didžiausias laiko skirtumas modeliuojant ir programuojant matomas vykdant lengviausią užduotį, vėliau skirtumas mažėja, bet vėl išaugo vykdant sudėtingą užduotį. Tai rodo, kad nauji programuotojai įdeda daug pastangų kol įsigilina į karkaso galimybes ir jo dokumentaciją. Skirtumo sumažėjimas atliekant vidutinę ir padidėjimas vykdant sunkią užduotį rodo, kad įrankio efektingumas didėja kylant uždavinio sudėtingumui.

# 9. IŠVADOS IR REZULTATAI

Šiame darbe nagrinėjamas hipertekstinės grafinės vartotojo sąsajos aprašymas aukšto abstrakcijos lygmens elementais, juos apibrėžiant deklaratyvia sintakse.

1) Analizuojami galimi deklaratyvios kalbos panaudojimo hipertekstinei grafinei vartotojo sąsajai kurti sprendimai bei tiriamos jų savybės. Įvertinus HTML ir JavaScript praplėtimo galimybes, pasiūlomas architektūrinis modelis HTML grafinei varotojo sąsajai deklaratyvia sintakse kurti. Sprendimas atitinka W3C standartus, nekuria naujų standartų, lengvai integruojasi į HTML dokumento medelį bei nekelia papildomų reikalavimų esamiems HTML vartotojams.

2) Siekiant sumažinti ryšių tarp silpną semantinę prasmę turinčių HTML elementų kiekį, siūloma hipertekstinę grafinę vartotojo sąsają pradėti kurti iš aukšto abstrakcijos lygmens elementų. Šiam tikslui atrenkama bazinių elementų aibė, apibrėžiama deklaratyvi notacija, suprojektuojama HTML grafinės vartotojo sąsajos biblioteka.

3) Eksperimentiniais tyrimais parodoma, kad deklaratyvi notacija ir aukštas abstrakcijos lygmuo sumažina programinį kodą nuo 3,1 karto trivialiems GUI elementams iki 204,1 karto sudėtingiems GUI sprendimams. Eksperimentai patvirtina, kad didėjant grafinę vartotojo sąsają sudarančių HTML elementų kiekiui, galima tikėtis dar geresnių efektyvumo rodiklių.

4) Sukurta architektūra yra integruota į programinės įrangos projektavimo modelį, kuriame vartotojo sąsajos ir logikos kūrimas iš dalies automatizuotas naudojant UML modelį. Tradicinis tekstinis redaktorius pakeistas duomenimis paremtu projektavimo įrankiu, panaudos atvejai vystomi scenarijais su grafiniu redaktoriumi, duomenų infrastruktūra generuojama iš modelio, o realizacija pateikiama keliomis programavimo technologijomis.

## 10. LITERATŪRA

[1]. **Louridas, P.** *Declarative GUI Programming in Microsoft Windows* [Žiūrėta 2010 m. kovo 20 d.], s.l.: Software, IEEE, 2007 m., T. 24, psl. 16 - 19. ISSN 0740-7459.

[2]. **Leff, A ir Rayfield, J.T.** *WebRB: A Different Way to Write Web Applications.* [Žiūrėta 2010 m. vasario 17 d.] s.l.: IBM T.J. Watson Res. Center, Cambridge, MA , Internet Computing, IEEE, 2008 m., T. 12, psl. 52 - 61. ISSN:1089-7801

[3]. **Chengwan He, Fei He, Keqing He, Wenjie Tu.** IEEE Xplore. *Constructing Platform Independent Models of Web Application.* 2005 m. 10 20 d. [Žiūrėta 2009 m. balandžio 11 d.],prieiga internete *<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1551133>*

[4]. **Hudson, Roger.** The Evolving Web. *Web Usability.* [Žiūrėta 2009 m. vasario 1 d.], prieiga internete *<http://www.usability.com.au/resources/evolving-web.cfm>*

[5]. **Packevičius, Šarūnas, Eidukynaitė, Vilma ir Ušaniov, Andrej.** *MDA CASE ĮRANKIŲ ANALIZĖ.* [Žiūrėta 2010 m. gegužės 22 d.] s.l.: Kauno Technologijos Universitetas.

[6]. **Elisa Bertino, Barbara Catania.** Integrating XML and databases. *Internet Computing, IEEE.* 2001 m. 08 02 d. [Žiūrėta 2010 m. gegužės 10 d.], prieiga internete *<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=939454>*

[7]. **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 m. gegužės 20 d.], prieiga internete *<http://www.ruf.rice.edu/~lane/papers/hidden_costs.pdf>*

[8]. **Kovse, Jernej ir Härder, Theo.** *Object-Oriented Information Systems.* [Žiūrėta 2010 m. gegužės 20 d.] s.l. : Springer Berlin / Heidelberg, 2002. 978-3-540-44087-1.

[9]. **Roberto Andreoli, Rosario De Chiara, Ugo Erra, Vittorio Scarano.** Università degli Studi della Basilicata. *Interactive 3D Environments by using videogame engines.* [Žiūrėta 2008 m. lapkričio 25 d.], prieiga internete *<http://www.unibas.it/utenti/erra/Papers/iv05.pdf>*

[10]. **Mellor, Stephen J.** Executable UML. *TechOnline.* [Žiūrėta 2005 m. sausio 4 d.] , prieiga internete *<http://www.techonline.com/article/pdf/showPDF.jhtml?id=1931036231>*

[11].**V. Dagienė, G. Grigas, T. Jevsikova**. Enciklopedinis kompiuterijos žodynas [Žiūrėta 2010m.. lapkričios 25d.], prieiga internete *<http://www.likit.lt/en-lt/angl.html>*

# 11. TERMINŲ IR SUTRUMPINIMŲ ŽODYNAS

CORBA (angl. common object request broker architecture)

CSS (angl. Cascading Style Sheets)

DOM (angl. document object model)

EJB (angl. enterprise java beans)

GUI (angl. graphic user hipertext markup interface)

HTML (angl. hypertext tarkup language)

J2EE (angl. java 2 platform, enterprise edition)

J2SE (angl. java 2 standard edition)

OOP (angl. object oriented programming)

PHP (angl. hypertext preprocessor)

PSM (angl. platform specific model)

SQL (angl. structured query language)

UML (angl. unified modeling language)

W3C (angl. world wide web consortium)

WPF (angl. windows presentation foundation)

XMI (angl. xml metadata interchange)

XML (angl. extensible markup language)

# 12. PRIEDAI

## 12.1. Straipsnis „Automatinis kodo generavimas naudojant grafinį scenarijų kūrimą remiantis Data Driven Design šablonu“.

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

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

**Kęstutis Valinčius, Sigitas Povilaitis, Rytis Ūsalis ir Paulius Paškevičius**

*Kauno technologijos universitetas, Programų inžinerijos katedra*

## 1. Įvadas

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

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

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

## 2. Problemos sprendimas pasaulyje

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

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

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

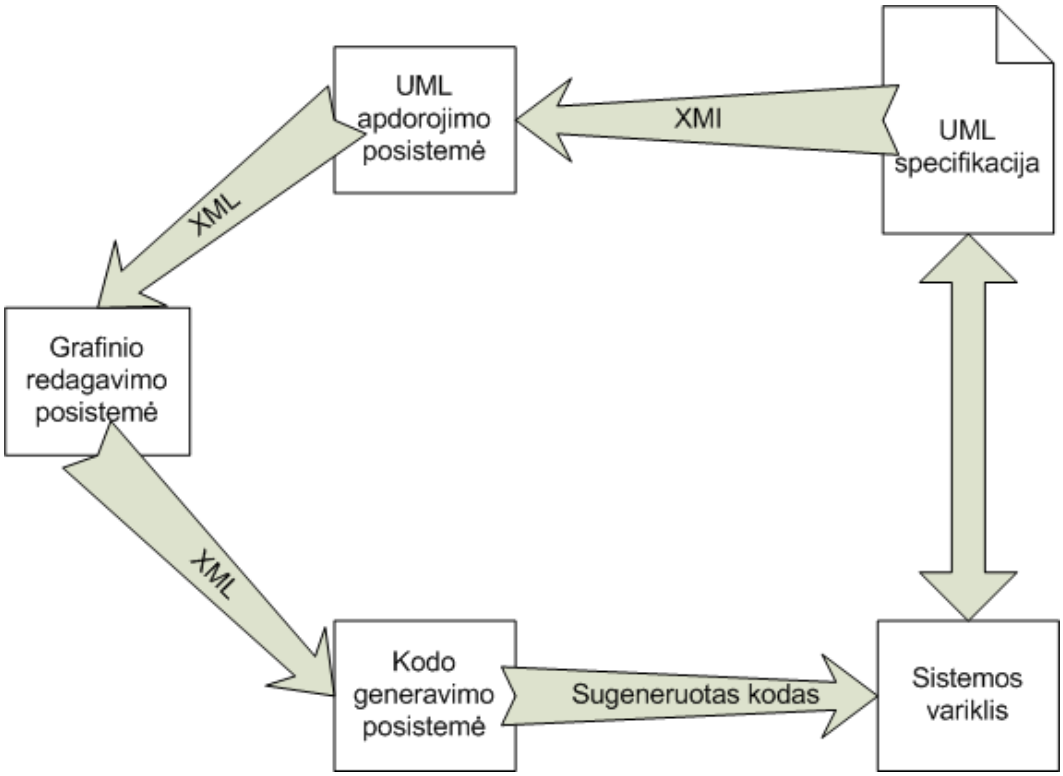
Tačiau ši programinė įranga labiau pritaikyta ne naudoti jau veikiančios sistemos galimybes, o kurti naujiems algoritmams.

## 3. Siūlomas sprendimas

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

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

įskiepis gali turėti taisykles būdingas specifinei architektūrai ar programavimo kalbai.



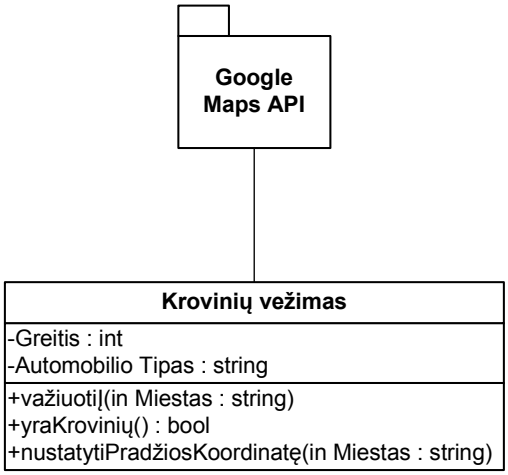

1 pav. Įrankio architektūros diagrama

## 4. Įrankio veikimo pavyzdys

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

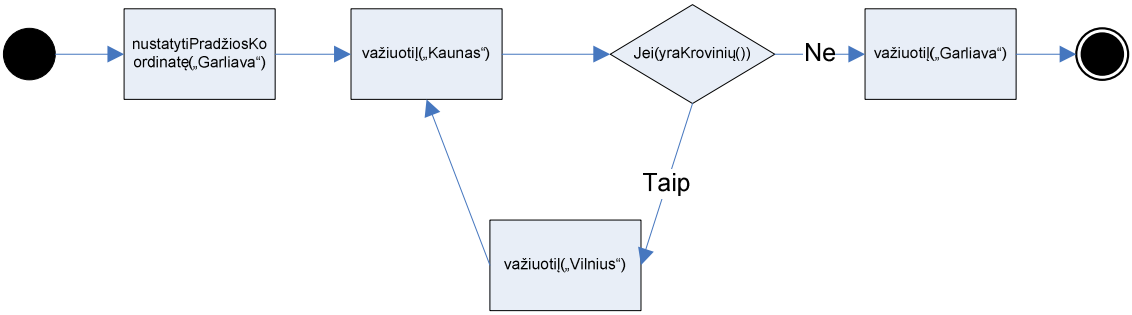
Vartotojui reikia sudaryti krovinio vežimo scenarijaus iš Kauno į Vilnių demonstraciją. Įmonės būstinė yra Garliavoje. Vairuotojas važiuoja į Kauną, kur tikrinama ar yra pervežimo užsakymų. Jeigu užsakymų yra, krovinys vežamas į Vilnių ir grįžtama atgal. Ši procedūra kartojama tol, kol kroviniai baigiasi.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

Sistemos UML specifikacijos pavyzdys:




1 pav. Pervežimų sistemos UML klasių diagrama

Grafinio redaktoriaus sumodeliuotas vaizdas:




3 pav. Krovinio vežimo grafinis modelis

Grafinio redaktoriaus XML išvestis:

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

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

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

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

4 pav. 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();
```

5 pav. 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>

# 1. HTML grafinės vartoto sąsajos bibliotekos detali architektūra

## 1. 1. Dokumento paskirtis

Šioje dalyje pateikiama grafinės vartotojo sąsajos bibliotekos detali architektūra.

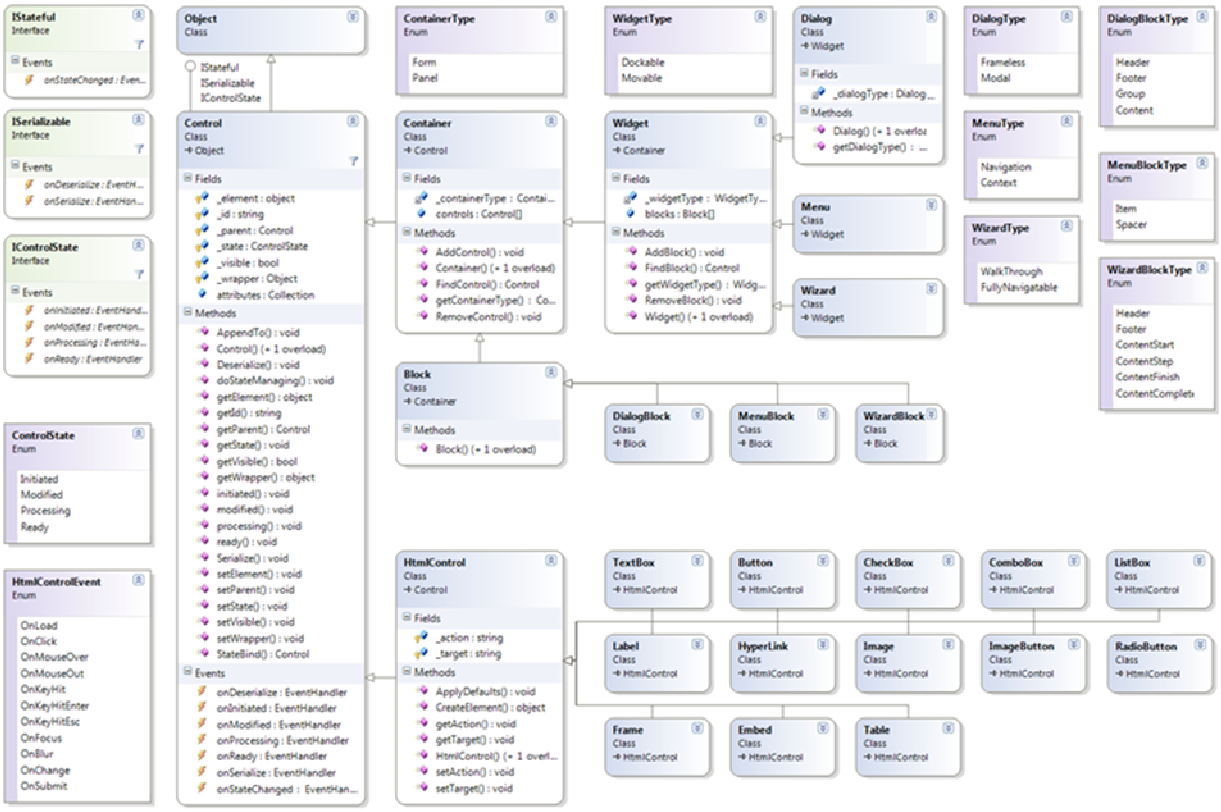
## 1.2. Naudojamų žymėjimų paaiškinimas

Architektūros specifikacijoje naudojami terminai ir žymėjimai.

| Žymėjimas | Reikšmė | Paaiškinimas |
|---|---|---|
|  | public class | viešai prieinama klasė |
|  | private class | uždara klasė |
|  | protected class | apsaugota klasė |
|  | public interface | viešai prieinama klasė |
|  | private interface | uždara klasė |
|  | protected interface | apsaugota klasė |
|  | public field | viešai prieinamas laukas |
|  | private field | uždaras laukas |
|  | protected field | apsaugotas laukas |
|  | public method | viešai prieinamas metodas |
|  | private method | uždaras metodas |
|  | protected method | apsaugotas metodas |
|  | public delegate | viešai prieinamas delegatas |
|  | private delegate | uždaras delegatas |
|  | protected delegate | apsaugotas delegatas |
|  | public property | viešai prieinama savybė |
|  | private property | uždara savybė |
|  | protected property | apsaugota savybė |
|  | public event | viešai prieinamas įvykis |
|  | private event | uždaras įvykis |
|  | protected event | apsaugotas įvykis |
|  | public enumeration | viešai prieinama sąrašas |
|  | private enumeration | uždaras sąrašas |
|  | protected enumeration | apsaugotas sąrašas |
|  | public  structure | viešai prieinamas struktūra |
|  | private structure | uždara struktūra |
|  | protected structure | apsaugota struktūra |
|  | namespace | vardų sritis |

### 1.2.1. Core.Presentation

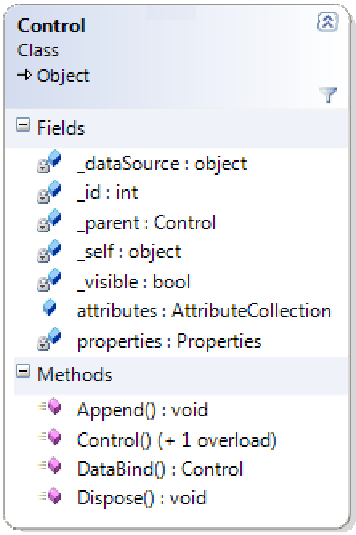
Ši sistemos branduolio dalis įgyvendina atvaizdavimo sluoksnį. Ji aprašo bazinius grafinės vartotojo sąsajos elementus, iš kurių komponuojami sudėtingi elementai.




Naudojantis branduolio atvaizdavimo sluoksniu galima greitai ir nesudėtingai kurti dialogus, vedlius, meniu bei kitus GUI komponentus. Šie komponentai palaiko temas (dizaino šablonus), kurių kūrimas ir redagavimas yra paprastas ir intuityviai suprantamas procesas.

Keli branduoliu sukurtų komponentų pavyzdžiai :

#### 1.2.1.1. Core.Presentation.Control




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Pamatinė klasė visiems komponentams.

*Atsakomybės*:
Klasė apibrėžia standartinį GUI komponentą.

*Apribojimai*:
Klasė naudojama tik kitų komponentų išvedimui.

*Sąsaja*: pasiekiami viešieji metodai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | attributes | Komponento atributų rinkinys. |
| | properties | Sąsaja su komponento privačiais laukais. |
| | Control | Sukuria komponentą. |
| | Append | Prijungia komponentą prie HTML elemento. |
| | DataBind | Susieja komponentą su duomenų šaltiniu. |
| | Dispose | Pašalima komponentą ir išvalo jo resursus. |
| | SetVisible | Nustato reikšmę, kuri lemia komponentas matomas ar nematomas. |

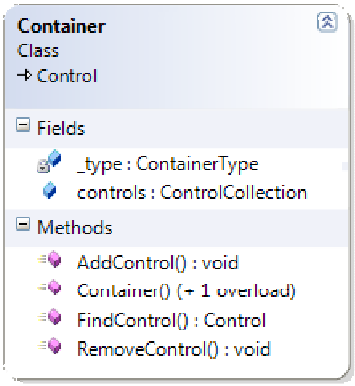
Sąsaja: viešieji sąsajos metodai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | getId | Gražina komponento unikalų identifikatorių. |
| | getDataSource | Gražina komponento duomenų šaltinį. |
| | getParent | Gražina komponento tėvą (HTML elementą). |
| | getSelf | Gražina komponento šakninį HTML elementą. |
| | getVisible | Gražina komponento matomumo požymį. |
| | setDataSource | Nustato komponento duomenų šaltinį. |
| | setParent | Nustato komponento tėvą (konteinerį). |
| | setSelf | Nustato komponento šakninį HTML elementą. |
| | setVisible | Nustato komponento matomumo požymį. |

Struktūra: vidiniai uždari laukai (pasiekiami tik per sąsajos metodus).

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | _id | Komponento unikalus identifikatorius. |
| | _dataSource | Komponento duomenų šaltinis. |
| | _parent | Komponento tėvas (HTML elementas). |
| | _self | Komponento šakninis HTML elementas . |
| | _visible | Gražina komponento matomumo požymį. |

#### 1.2.1.2. Core.Presentation.Container




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Pamatinė klasė sudėtiniams komponentams.
Praplečia klasę Control (komponentas).

*Atsakomybės*:
Klasė apibrėžia sudėtinį GUI komponentą.

*Apribojimai*:
Klasė naudojama tik kitų komponentų išvedimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*: pasiekiami viešieji metodai.

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | controls | Konteinerio komponentų rinkinys. |
| | Container | Sukuria kontenerio komponentą. |
| | AddControl | Įdeda komponentą į konteinerį. |
| | FindControl | Atlieka komponento paiešką koteinetyje. |
| | RemoveControl | Pašalima komponentą iš sąsajos ir iš komponentų saugyklos. |

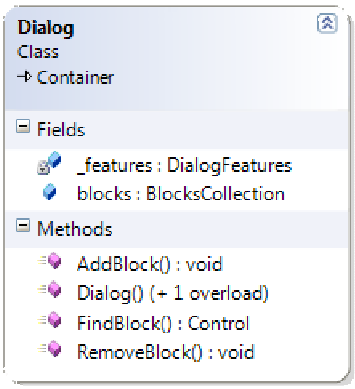
*Sąsaja*: viešieji sąsajos metodai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | getType | Gražina konteinerio tipą. |

*Struktūra*: vidiniai uždari laukai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | _type | Konteinerio tipas, kuris apibrėžia jo elgseną. |

### 1.2.1.3. Core.Presentation.Dialog




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė sudėtingiems komponentams: dialogams.
Praplečia klasę Container (sudėtinis komponentas).

*Atsakomybės*:
Klasė apibrėžia sudėtingą GUI komponentą.

*Apribojimai*:
Klasė turi būti naudojama kitų dialogų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*: pasiekiami viešieji metodai.

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | blocks | Komponento blokų rinkinys. |
| | Dialog | Sukuria dialogo komponentą. |
| | AddBlock | Įdeda bloką į dialogą. |
| | FindBlock | Atlieka bloko paiešką talpinančiajame. |
| | RemoveBlock | Pašalima bloką iš sąsajos ir komponentų saugyklos. |

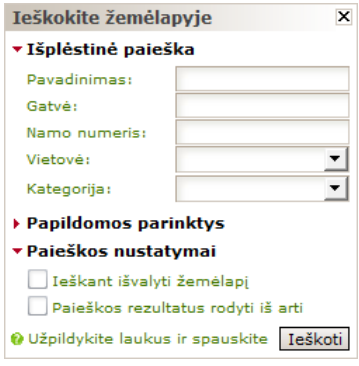
*Sąsaja*: viešieji sąsajos metodai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | getFeatures | Gražina dialogo savybes. |

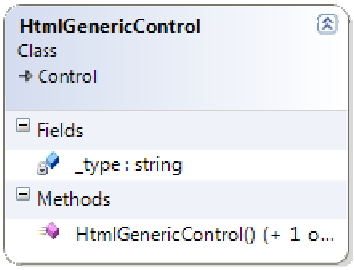
*Struktūra*: vidiniai uždari laukai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | _features | Dialogo savybės, kuris apibrėžia jo veikimą. |

*Pavyzdžiai*:
Paieškos ir informacijos dialogai (dizaino šablonas: tinklapis PATOGIAU.LT):




#### 1.2.1.4. Core.Presentation.HtmlGenericControl




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Pamatinė klasė standartiniams HTML komponentams.
Praplečia klasę Control (komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą.

*Apribojimai*:
Klasė naudojama tik kitų standartinių HTML komponentų išvedimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*: pasiekiami viešieji metodai.

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | HtmlGenericControl | Sukuria standartinį HTML komponentą. |

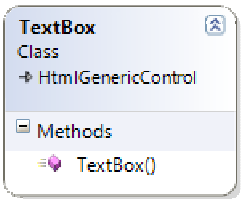
*Sąsaja*: viešieji sąsajos metodai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | getType | Gražina standartinio HTML komponento tipą. |

*Struktūra*: vidiniai uždari laukai.

| Matomumas | Pavadinimas | Aprašymas |
|---|---|---|
| | _type | Standartinio HTML komponento tipas, kuris apibrėžia jo elgseną. |

### 1.2.1.5. Core.Presentation. TextBox




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: teksto laukams.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą TextBox.

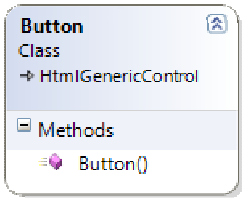
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML TextBox komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | Textbox | Sukuria standartinį HTML TextBox komponentą. |

### 1.2.1.6. Core.Presentation. Button




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams : mygtukams.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą Button.

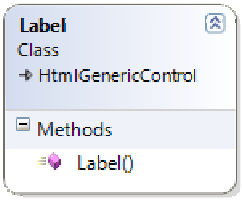
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML Button komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | Button | Sukuria standartinį HTML Button komponentą. |

#### 1.2.1.7. Core.Presentation. Label




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: teksto komentarams.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą Label.

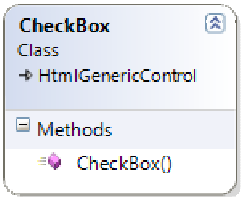
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML Label komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | Label | Sukuria standartinį HTML Label komponentą. |

#### 1.2.1.8. Core.Presentation. CheckBox




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: pasirinkimams.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą CheckBox.

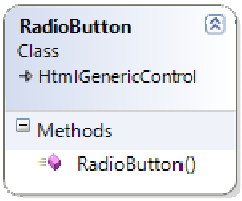
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML CheckBox komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | CheckBox | Sukuria standartinį HTML CheckBox komponentą. |

#### 1.2.1.9. Core.Presentation. RadioButton




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: grupės pasirinkimams.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą RadioButton.

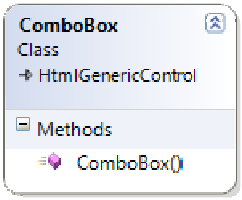
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML RadioButton komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | RadioButton | Sukuria standartinį HTML RadioButton komponentą. |

#### 1.2.1.10. Core.Presentation. ComboBox




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: iškrentantiems sąrašams.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą ComboBox.

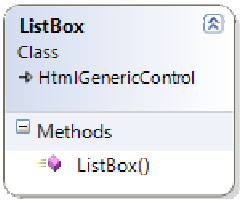
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML ComboBox komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | ComboBox | Sukuria standartinį HTML ComboBox komponentą. |

#### 1.2.1.11. Core.Presentation. ListBox




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: sąrašams.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą ListBox.

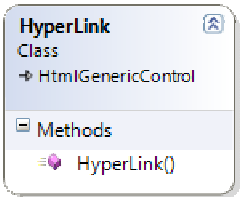
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML ListBox komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | ListBox | Sukuria standartinį HTML ListBox komponentą. |

#### 1.2.1.12. Core.Presentation. HyperLink




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: hipertekstinėms nuorodoms.
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą HyperLink.

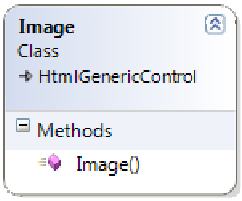
*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML HyperLink komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | HyperLink | Sukuria standartinį HTML HyperLink komponentą. |

### 1.2.1.13. Core.Presentation. Image




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: paveikslams
Praplečia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą Image.

*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML Image komponentų kūrimui.

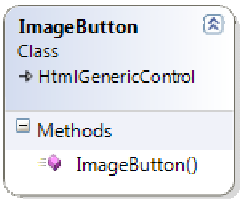
*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | Image | Sukuria standartinį HTML Image komponentą. |

*Pastabos*:
Tiesioginiai klasės atstovai turi grafinę vartotojo sąsają ir yra pilnai funkcionalūs.

### 1.2.1.14. Core.Presentation. ImageButton




*Klasifikacija*:
Komponento rūšis: klasė.

*Apibrėžimas*:
Klasė standartiniams HTML komponentams: nuorodoms-paveikslėliams.
Praplėčia klasę HtmlGenericControl (HTML standartinis komponentas).

*Atsakomybės*:
Klasė apibrėžia standartinį HTML GUI komponentą ImageButton.

*Apribojimai*:
Klasė turi būti naudojama kaip pagrindas kitų HTML ImageButton komponentų kūrimui.

*Skaičiavimai*:
Klasėje vykdomi trivialūs skaičiavimai.

*Sąsaja*:

| Matomumas | Metodai | Aprašymas |
|---|---|---|
| | ImageButton | Sukuria standartinį HTML ImageButton komponentą. |