KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Justinas Strakšys

Sričiai orientuotos informacinės sistemos kūrimo metodikos tyrimas ir taikymas

Magistro darbas

Darbo vadovas
prof. E. Bareiša

Kaunas, 2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Justinas Strakšys

Sričiai orientuotos informacinės sistemos kūrimo metodikos tyrimas ir taikymas

Magistro darbas

Recenzentas
doc. T. Blažauskas
2009-01-

Vadovas
prof. E. Bareiša
2009-01-

Atliko
IFM-3/4 gr. stud.
Justinas Strakšys
2009-01-12

Kaunas, 2009

Turinys

1. Įvadas

Pastaruoju metu neretas informacinių sistemų projektuotojas ar paprastas programuotojas susimąsto, kaip būtų galima palengvinti ir pagreitinti sistemos kūrimo darbus. Nebereikia net įrodinėti, kad sistemos kūrimas turi prasidėti nuo sistemos suprojektavimo, suvokimo kam ji skirta, ką turės atlikti, kas ja naudosis ir kaip.

Turbūt nebeliko jau nei vieno programuotojo, kuris imasi programavimo darbų, tiksliai nežinodamas visų reikalavimų. Taip yra dėl to, kad šiuolaikinės informacinės sistemos tampa vis sudėtingesnės, joms keliami vis didesni reikalavimai, norima, kad jų kūrimo terminas būtų kiek įmanoma trumpesnis, o skiriami kaštai kuo mažesni.

Taigi atsiranda tokie klausimai: kokius metodus naudoti kūrimo metu, kokią programinę įrangą pasirinkti.

Šiam momentui yra sukurta daugybė įrankių, padedančių informacinių sistemų kūrėjams dar prieš pradedant programavimo darbus pamatyti bendrą sistemos vaizdą, įsitikinti jos funkcionalumo tinkamumu, priimtų sprendimų teisumu. Taip pat dauguma šių sistemų netgi pateikia pradinį programinį kodą, kas dar labiau palengvina darbą bei sumažina tikimybę atsirasti klaidoms. Tai ypač aktualu, kai kalbama apie didelio masto projektus, stambias informacines sistemas, kurių kūrimas trunka ne viena ir ne du mėnesius. Tuomet klaidų taisymas tampa sudėtinga procedūra, reikalaujančia daug žmogiškųjų išteklių. Sugaištama nemažai laiko bei padidėja programos realizacijos kaštai.

Esant galimybei rinktis iš daugybės įrankių, yra tikslinga surasti optimaliausią variantą t.y. tinkamiausią įrankį, galintį pilnai patenkinti keliamus reikalavimus sistemos kūrimui ir tuo pačiu turintį kuo mažiau perteklinių funkcijų.

Taigi nekyla abejonių, kad tokių įrankių paklausa yra didelė, tačiau ir pasiūla neatsilieka nuo jos. Taip pat atsiranda įrankių, suteikiančių galimybę vartotojui pačiam susikurti projektavimo ar programavimo įrankius, kurie atitiktų jo poreikius. Ši sritis sparčiai žengia į priekį, nes vis daugiau vartotojų pradeda naudotis tokiais įrankiais, nes jų atnešama nauda ženkliai viršija pastangas įsisavinti šiuos naujus produktus.

Kuo platesnė įrankio sprendžiamų uždavinių įvairovė, tuo siauresnės jo galimybės, sprendžiant kiekvieną atskirą uždavinį. Produktas pritaikytas konkrečiai probleminiai sričiai spręsti yra geresnis už skirtą spręsti visoms įmanomoms problemoms.

Tačiau ar visiems uždaviniams spręsti tinkamos vienodos priemonės? Ar bet kokio dydžio projektų kūrimas atsiperka, panaudojant naujausias metodologijas?

Šiai dienai turime daugybę metodikų ir technologijų, kaip reikėtų kurti programinę įrangą. Visos jos turi savo privalumų ir trūkumų, tačiau nei vienos iš jų negalima vadinti bloga. Savo laiku jos buvo naudingos bei paskatino naujų metodikų vystymąsi.

Tad iškyla problema renkantis sistemos kūrimo metodus. Dažniausiai yra naudojamos modeliais paremtos sistemų kūrimo metodikos. Jų yra įvairiausių: nuo daugybe modelių aprašančių kuriamą sistemą iki aprašomų modeliu, pritaikytu konkrečiai problemai.

Viena iš naujausių sistemų kūrimo metodikų – sričiai orientuotas modeliavimas (angl. domain specific modeling). Ji paremta sričiai orientuotos kalbos kūrimu ir jos panaudojimu sistemos kūrimui. Ši metodika leidžia sistemos kūrimui naudoti sąvokas, artimas probleminei sričiai, kas padidina abstrakcijos lygi projektavimo metu. Srities sąvokos aptariamos ir įvardinamos kartu su ekspertais ar paprastais darbuotojais, dirbančiais analizuojamoje srityje, ir nebūtinai išmanančiais sistemų kūrimo metodus. Ši metodika leidžia sistemos projektuotojui lengviau susikalbėti su užsakovais, nes modeliuose figūruojančios sąvokos yra užsakovui suprantamos ir aiškios.

Šiuo darbu siekiama ištirti sričiai orientuotos metodikos panaudojimą informacinės sistemos kūrimui. Pasirinkta probleminė sritis – duomenų, esančių duomenų bazėse, vieningas atvaizdavimas vartotojams. Problema atsiranda tuomet, kuomet įmonėje dirba skirtingų sričių specialistai: duomenų bazių specialistas, informacinių sistemų kūrėjas ir paprasti vartotojai, nesusiję su įmonės informacinėmis sistemomis (vadybininkai, direktorius, padalinių vadovai ir panašiai). Kuomet vartotojams prireikia konkrečių duomenų – duomenų bazių specialistas turi juos išgauti iš duomenų bazės, ir pateikti vartotojui priimtina forma. Tai užima daug laiko dėl prisijungimų prie duomenų bazių valdymo sistemų, užklausų formavimo, rezultatų apdorojimo.

Šiai problemai spręsti puikiai tinka sričiai orientuotos kalbos panaudojimas, kuomet duomenų bazių specialistas jam suprantama kalba (grafinio redaktoriaus pagalba) sukuria sistemą, kuri atlieka visą, daug laiko užimantį darbą ir gali būti duota naudotis įmonės darbuotojams.

2. Analizė

2.1. Analizės tikslas

Atliekant analizę, buvo siekiama išanalizuoti sričiai orientuotą kalbą bei jos panaudojimo galimybes sistemų kūrimui. Taip pat išanalizuoti kitas panašias metodologijas, naudojamas sistemų kūrimui ir jas palyginti su sričiai orientuotu modeliavimu.

2.2. Tyrimo sritis, objektas ir problema

Tyrimo sritis – informacinės sistemos projektavimas, panaudojant sričiai orientuotą kalbą Microsoft .Net platformoje.

Tyrimo objektas – sričiai orientuotos kalbos panaudojimas Microsoft .Net platformoje įgyvendinant praktinę užduotį.

Tyrimo problema – tobulėjant informacinių sistemų kūrimo technologijoms, vis daugiau dėmesio skiriama toms technologijoms, kurios leidžia supaprastinti ir pagreitinti kūrimo procesą. Yra sukurta keletas metodikų, leidžiančių ganėtinai išsamiai aprašyti kuriamos sistemos sprendžiamą sritį, tačiau kad jomis būtų galima pilnai pasinaudoti, reikalinga sukurti daug įvairių modelių, nes modeliavimo įrankiai yra pritaikyti visoms įmanomoms sritims. Sričiai orientuotų kalbų kūrimas leidžia orientuotis į konkrečią probleminę sritį, ko dėka mes turime kalbą, kuri ir aprašo mūsų probleminę sritį. Tačiau individualus šios metodikos panaudojimas dar yra ganėtinai naujas, nėra nusistovėjusių standartų jos aprašymui, neaiški jos panaudojimo gaunama nauda, todėl informacinių sistemų kūrėjai nesiryžta naudoti sričiai orientuotų kalbų savo projektuose. Tuo būdų jų vystymasis ir plitimas yra stabdomas.

2.3. Sričiai orientuota kalba (tyrimo objekto analize)

Sričiai orientuota kalba (toliau DSL) yra skirta (suprojektuota) išreikšti reikalavimus ir sprendimus tam tikroje verslo ar architektūrinėje srityje [14]. Su tokiomis kalbomis mes dažnai susiduriame projektuodami sistemas. Žinomiausi DSL pavyzdžiai būtų: struktūrizuota užklausų kalba (angl. Structured Query Language, SQL), hiperteksto žymėjimo kalba (angl.

Hyper text Markup Language, toliau HTML), grafiniai vartotojo sąsajos projektuotojai (angl. Graphical User Interface designer). Visos jos mums leidžia sutelkti dėmesį į išreiškimą to, kas mums reikalinga, terminais, tiesiogiai susijusiais su ta sritimi, paliekant platformai tinkamiausią viso to įgyvendinimą. Problemos, kurių sprendimas buvo sudėtingas darbas prieš šių kalbų ir jų įgyvendinimo atsiradimą, dabar yra nedaug laiko reikalaujantis procesas.

Tačiau ar galima visa tai pritaikyti ir konkretiems projektams įmonės viduje? Ar galima pasiekti tokių pačių rezultatų, susikuriant savo sričiai orientuotas grafines ar tekstines kalbas?

Terminas „sričiai orientuota kalba“ tampa vis populiaresnis pastaraisiais metais ir plačiau naudojamas programinės įrangos kūrime. Iš pavadinimo galima suprasti, kad sričiai orientuota kalba nėra įprasta programavimo kalba. Išsamų ir tikslų jos apibrėžimą pateikia kompanijos Microsoft darbuotojai, kuriantys ir tobulinantys DSL kalbai kurti skirtą įrankį „DSL Tools“ (kuris yra programinio paketo „Visual Studio 2008“ išplėtimo įrankio „Visual Studio 2008 Software Development Kit (SDK)“ dalis) [1]:

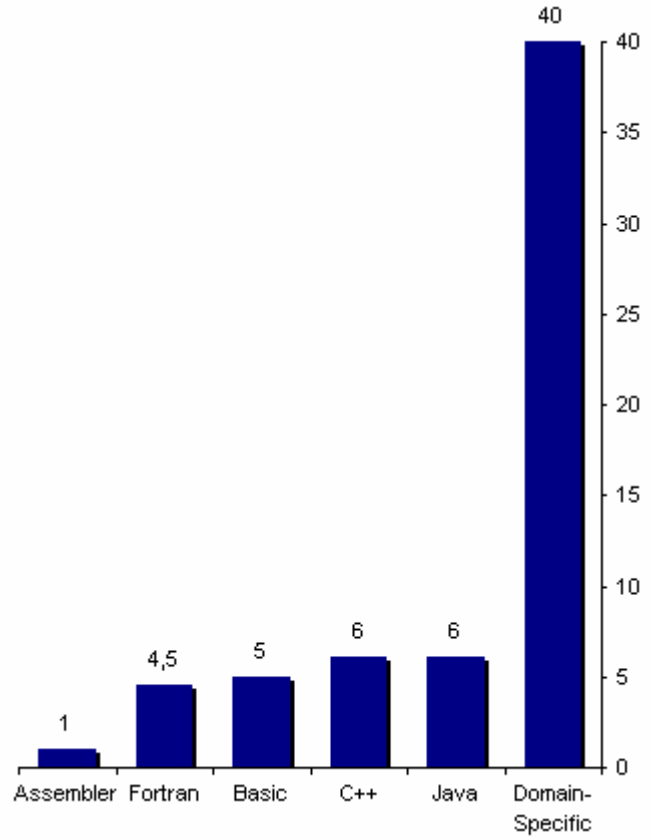
sričiai orientuota (specifinė sričiai) kalba – tai pagal užsakymą sukurta (programavimo) kalba, kuri orientuota į konkrečią nedidelę (nebūtinai) probleminę sritį, kurią ji apibrėžia (apibūdina) ir validuoja sąvokomis, artimomis tai sričiai.

Pati sąvoka nėra nauja – specialios paskirties programavimo kalbos bei įvairios modeliavimo ar specifikavimo kalbos visada egzistavo. Tačiau šis terminas pradėtas vis plačiau vartoti, nes atsirado poreikis kurti sistemas, o orientuotas vienai sričiai.

DSL priskiriama ketvirtosios kartos programavimo kalboms (angl. Fourth - Generation Programming Language, 4GL). Tai yra kalbos ar programavimo aplinka, sukurta tam tikrai sričiai realizuoti. Tokių kalbų pavyzdžiai yra SQL, MATLAB, CSS ir daugybė kitų. Šių kalbų pagalba, mes tam tikra, tik tai kalbai būdinga forma, aprašome užduotis ir gauname rezultatą, nesigilindami į viso to realizaciją programiniame lygmenyje. Tai mums sutaupo laiko ir supaprastina realizacijos procesą. Reikia paminėti, kad trečiosios kartos programavimo kalbomis (angl. Third - Generation Programming Language, 3GL) laikomos tokios bendrosios paskirties programavimo kalbos kaip C, C++, C#, Java ir panašiai, kurių tikslas padidinti kalbos panaudojamumą, kad jos taptų patrauklesnėmis vartotojui (pavyzdžiui pertvarkant galimų funkcijų kategorijas, kad jos taptų efektyvesnės, sutelkiant visą jų kodą klasėse).

Sričiai orientuota kalba yra kažkur tarp mažytės programavimo kalbos ir skriptų (angl. script) rašymo kalbos, ir dažnai vartojama panašiai kaip ir programavimo bibliotekos. Skirtumas tarp šių sąvokų yra ganėtinai mažas, maždaug kaip skirtumas tarp skriptų rašymo kalbų ir bendrosios paskirties kalbų.

Pradėjus naudoti DSL sistemų kūrime, smarkiai padidėjo šio proceso produktyvumas (įvairių šaltinių teigimu, produktyvumas padidėja nuo penkių iki dešimties kartų). Prieš tai įvykęs didžiausias produktyvumo šuolis buvo tuomet, kai buvo pereinama nuo asemblerio (angl. assembler) programavimo kalbos prie trečiosios kartos programavimo kalbų. Nuo to laiko atsiradusios technologijos ar programavimo kalbos, neturėjo ryškios produktyvumo didinimo tendencijos. Pavyzdžiui vidutinis produktyvumo padidėjimas, naudojant Java programavimo kalbą, yra tik 20% didesnis, negu naudojant BASIC programavimo kalbą[9][10].



1 pav. Skaičius naujų produkto savybių, įgyvendintų per duotą laiką[9]

Sričiai orientuotos kalbos privalumai:

- ji leidžia dirbti, naudojant probleminės srities terminus, sumažinat klaidų tikimybę, kurios atsiranda apibrėžiant probleminę sritį bendrojo naudojimo (angl. general purpose) kalbomis;
- darbas, naudojant probleminės srities terminus, leidžia sukurti modelius, kurie yra labiau suprantami žmonėms, kurie nėra susipažinę su įgyvendinimo technologijomis, įskaitant ir verslo žmones;

- DSL tuo pačiu pateikia ir sričiai orientuotą programavimo sąsają (angl. Application Programming Interface, API), kuri leidžia valdyti jos modelius, tokiu būdų padidindama kūrėjo produktyvumą;
- artefaktai, sugeneruoti DSL pagalba, nebūtinai turi būti technologiniai įgyvendinimo elementai; tinkamas modelis gali būti panaudotas sukurti skriptams (angl. script), dokumentacijai, medžiagų sąrašams ir panašiai.

Sričiai orientuotos kalbos trūkumai:

- kaštai, atsirandantys DSL kūrimui, realizavimui ir palaikymui;
- kartais sunku tiksliai apibrėžti ir pritaikyti probleminę sritį šiai metodikai;
- kartais sudėtinga suderinti probleminę sritį su bendrosios paskirties programavimo kalbos konstrukcijomis;
- reikalinga galingesnė aparatūrinė įranga, lyginant su ranka rašytu kodu;
- gali būt sunku ar net neįmanoma derinti (angl. debug).

2.4. DSL ir DSM

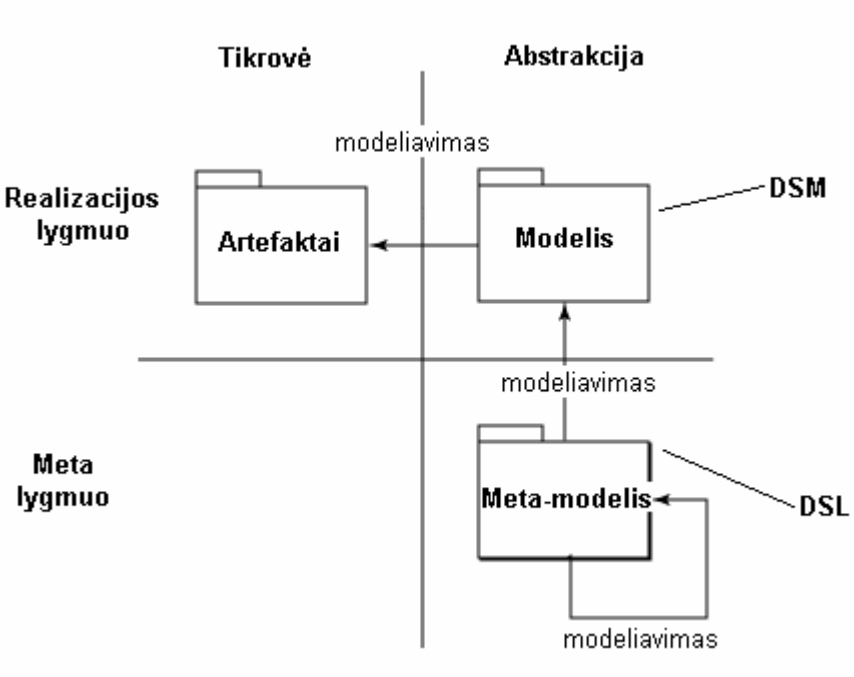
Dėl tos priežasties, kad DSL naudojimas vis didėja, o ir išvengti jų tampa vis sunkiau, atsiranda ir tokios sąvokos kaip:

- sričiai orientuotas modeliavimas (angl. Domain-Specific Modeling, toliau DSM) ir
- Sričiai orientuotas kūrimas (angl. Domain-Specific Development, DSD).

Bendrąja prasme šios abi sąvokos apibūdina tą patį: DSL‘ų kūrimo ir naudojimo būdus, dažniausiai vaizdinių DSL‘ų su sričiai orientuotais generatoriais (angl. generator), kurie sugeneruoja pilnai veikiantį kodą tiesiai iš sistemos modelio.

Pagrindinis DSM tikslas yra atlikti du darbus[13]:

- pakelti abstrakcijos lygį virš programavimo, apibrėžiant sprendimą projektavimo kalba, kuri tiesiogiai naudoja sąvokas bei taisykles iš konkrečios probleminės srities;
- sugeneruoti galutinį produktą pasirinkta programavimo kalba tiesiai iš šių aukšto lygio specifikacijų (šis automatizavimas tampa įmanomu dėl to, kad tiek kalba, tiek generatorius atitinka tą pačią probleminę sritį. Kitais žodžiais tariant, jie abu yra sričiai orientuoti).



2 pav. Sąvokų išsidėstymas DSM metodikoje

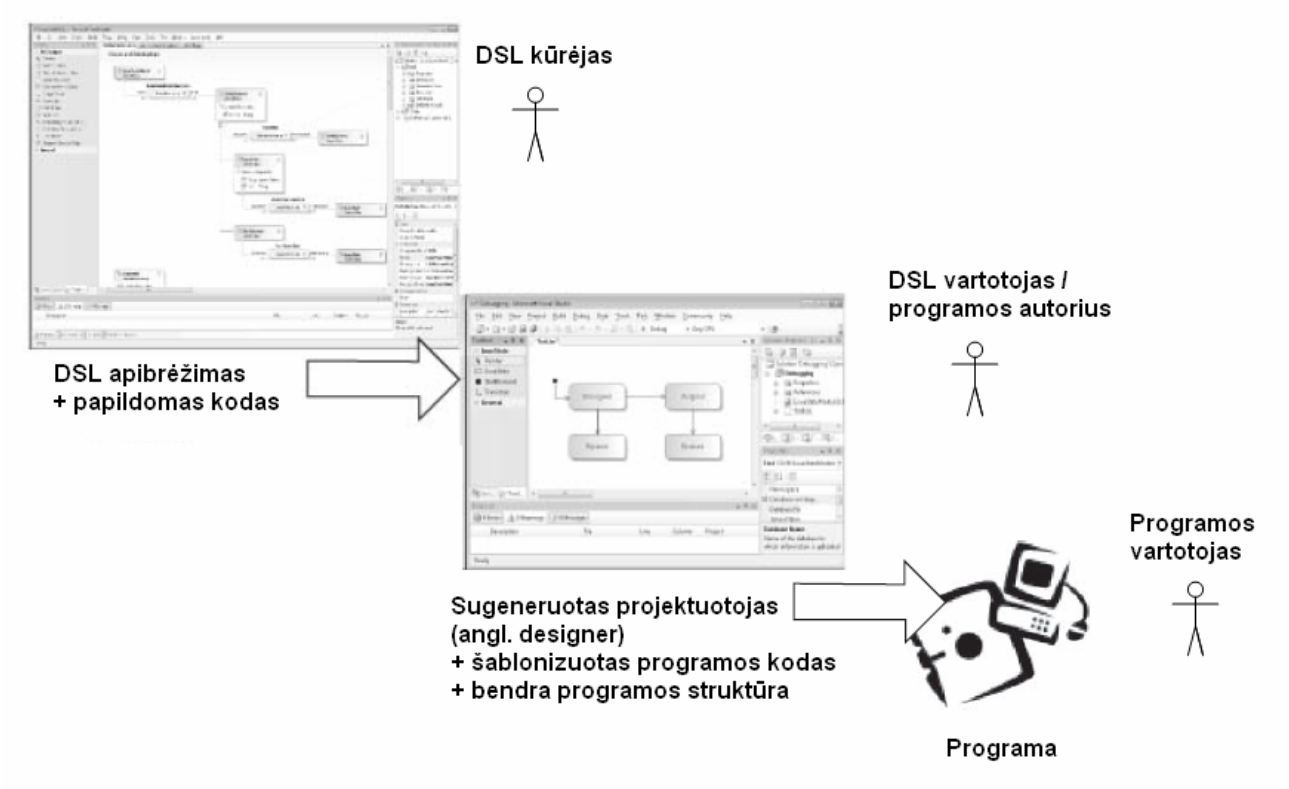
Pasiekti šiai metodikai keliamus tikslus, mums reikalinga DSL kalba ir ją atitinkantis kodo generatorius. Nors juos apibrėžti (angl. define) yra daug paprasčiau, nei bendrojo modeliavimo kalbą (angl. generic modeling language), tačiau tai užduotis, tinkanti ne kiekvienam kūrėjui. Šis procesas reikalauja gerų probleminės srities žinių, ir kodo, kuris jai parašytas. Tai būdinga situacijose, kuriose susiduriama su produktų šeimų kūrimu, testiniu sistemos kūrimu ar sistemos derinimu pagal specifinius vartotojų reikalavimus (darbų sekos, apmokėjimo sistemos, CRM (angl. Customer Relationship Management) sistemos ir panašiai).

Patyrę kūrėjai, kurie dirba tokiose srityse, žino kokios sąvokos figūruoja konkrečiose srityse, bei yra susipažinę su taisyklėmis, kurios jas apibrėžia, ir kaip kodas ar konfigūracijos rinkmenos turi būti aprašyti geriausiai. Kartais tai yra vienas ekspertas, kartais šis darbas padalinamas mažoms grupėms žmonių, su skirtingais įgūdžiais. Ar tai būtų vienas žmogus, ar keli, žinios, kurias jie turi, leidžia jiems kompetentingai apibrėžti automatizavimo mechanizmą, kuris leis likusiems kūrėjams dirbti žymiai produktyviau.

Visų pirma reikėtų išskirti ir apibrėžti DSM procese veikiančių vartotojų vaidmenis. Jie yra trys:

- DSL kūrėjas;
- DSL vartotojas / programinės įrangos (informacinės sistemos) autorius;
- Vartotojas (užsakovas).

Kuomet vartotojas iškelia problemą (ar norą papildyti sistemą nauju funkcionalumu) ir sistemos autorius tai konkretizuoja, DSL kūrėjas sukuria modelį, kuris apibrėžia iškilusį klausimą ir jam spręsti reikalingus elementus. Tuomet, DSL vartotojas, naudodamasis šia naujai sukurta kalba, apibrėžiančia jam aktualią sritį, sudaro modelį ir jo, bei šablonų pagalba sugeneruoja jam reikalingą programinį kodą, kurį gali įterpti į jau veikiančią savo sistemą. Reikia paminėti, kad tas pats asmuo gali įeiti į kelias grupes (tarkime DSL kūrime gali dalyvauti ir DSL vartotojas ar programos vartotojas, nes jie dažniausiai yra geriau susipažinę su problemine sritimi nei DSL kūrėjas).



3 pav. DSL kūrimo ir naudojimo vartotojai [1]

Naudojantis tuo pačiu modeliu gali būti sugeneruotos bet kokio tipo bylos, ne tik programinis kodas. Tarkime vartotojas norėtų gauti šio modelio santrauką HTML formatu, kad galėtų ją patalpinti žiniatinklyje. Tuomet šablonų pagalba tai galima nesudėtingai atlikti ir vartotojui ją pateikti. Tokio būdo privalumas yra tas, kad šie duomenys identiškai atitiktų sistemą, nes generuojami iš to paties modelio.

DSL panaudojimas ypatingas dar ir tuo, kad kuriant pačią kalbą įvedami apribojimai modeliui. Tai pasiekiama ryšių tarp elementų pagalba – nustatant jų daugialypiškumą (kiek ir kokiu elementų kitas elementas gali turėti). Tai padeda apsisaugoti nuo programavimo klaidų, kuomet neapgalvotai galima bandyti įgyvendinti negalimus perėjimus tarp sistemos elementų ar būsenų, kuomet kuriant modelį mes tiesiog negalime šių veiksmų atlikti. Taip sumažinama klaidų tikimybė.

Sričiai orientuoto modeliavimo privalumai[1][8]:

- modeliai, išreikšti DSL pagalba, gali būti patikrinami dar probleminės srities abstrakcijos lygmenyje, kas reiškia, jog modelio supratimo ir išreiškimo klaidos gali būti ištaisomos žemesniuose kūrimo gyvavimo ciklo etapuose;
- modeliai gali būti panaudojami imituoti sprendimą tiesiogiai, kas leidžia iškart pamatyti modelio tinkamumą;
- modeliai gali būti panaudojami derinti įgyvendinimą, susidedantį iš kelių skirtingo tipo technologijų, kas gali sumažinti reikalingus įgūdžius ir pastangas, sprendimą įgyvendinant būtent tomis technologijomis;
- modeliai taip pat gali būti panaudoti, kad sukurtų kitus modelius, ir formuotų kitas sistemas, tinklus, ir produktus, galbūt kombinacijose su kitomis technologijomis, tokiomis kaip automatiniai vedliai (angl. wizards);
- kuomet svarbios verslo žinios yra surenkamos į modelį, sprendimo perkėlimas iš vienos technologijos į kitą, ar kitą tos pačios technologijos versiją, tampa daug paprastesnis ir greitesnis. Dažniausiai tai reikalauja tik kuklių generatoriaus ar interpretatoriaus pakeitimų;
- kadangi stengiamasi kodą generuoti tiesiai iš modelio, kodas tampa savaime dokumentuojamas (įterpiant komentarus į generatorius ar atskirai generuojant dokumentaciją);
- naudojant šias metodikas sistemų kūrimui, galima ženkliai padidinti kūrimo produktyvumą.

Sričiai orientuoto modeliavimo trūkumai:

- atsiranda papildomų išlaidų, skirtų DSL kūrimui, įgyvendinimui ir palaikymui;
- kartais sudėtinga aprašyti probleminę sritį taip, kad būtu galima išgauti bendrosios paskirties programavimo kalbų struktūrą;
- kūrimui reikalinga galingesnė aparatūrinė įranga, lyginant su įprastinėmis sistemų kūrimo technologijomis (ar ranka rašomu kodu);
- kartais sukurti tinkamą DSL, kuri atitiktų visus poreikius ir lūkesčius, gali būti sunku, ypač tai darant pirmą kartą ar neturint pakankamai probleminės srities žinių.

2.5. Automatinis kodo generavimas

Kadangi DSM glaudžiai susijusi su kodo generavimu, panagrinėkime kodo generavimą.

Programinės įrangos kūrimui didelės įtakos turi du teiginiai[5]:

- Programuotojų laikas yra labai vertingas.
- Programuotojai nemėgsta pasikartojančių bei nuobodžių užduočių.

Taigi iškyla problema, kaip sumažinti šių teiginių daromą įtaką programinės įrangos kūrimui.

Didelė dalis pasikartojančio kodo rašymo yra perduodama kodo generavimui, kas leidžia programuotojams daugiau laiko skirti įvairiems programavimo iššūkiams, kas jiems teikia malonumą.

Kodo generavimas ne tik pašalina nuobodų (monotonišką) darbą, bet ir atneša naudos programinės įrangos gyvavimo ciklui šiose keturiose kategorijose:

Produktyvumas.

Naudojant kodo generavimą, galima pasiekti tokį produktyvumo lygmenį, kurio niekaip neįmanoma pasiekti rašant kodą ranka. Kodo generatoriai gali sukurti šimtus klasių per kelias sekundes, o jei reikia pakeisti kodą – tai taip pat atliekama labai greitai visoje sistemoje.

Kokybė.

Sugeneruotas kodas turi vieningą kokybę visame kodo branduolyje. Kuomet randate klaidą, ją galima ištaisyti (visame kode) vieninteliu generavimo ciklu (generatoriaus paleidimu). Kodo generavimas taip pat palaiko blokų (angl. unit) testavimą, kas yra pagrindas kokybės palaikymui programinės įrangos sistemose.

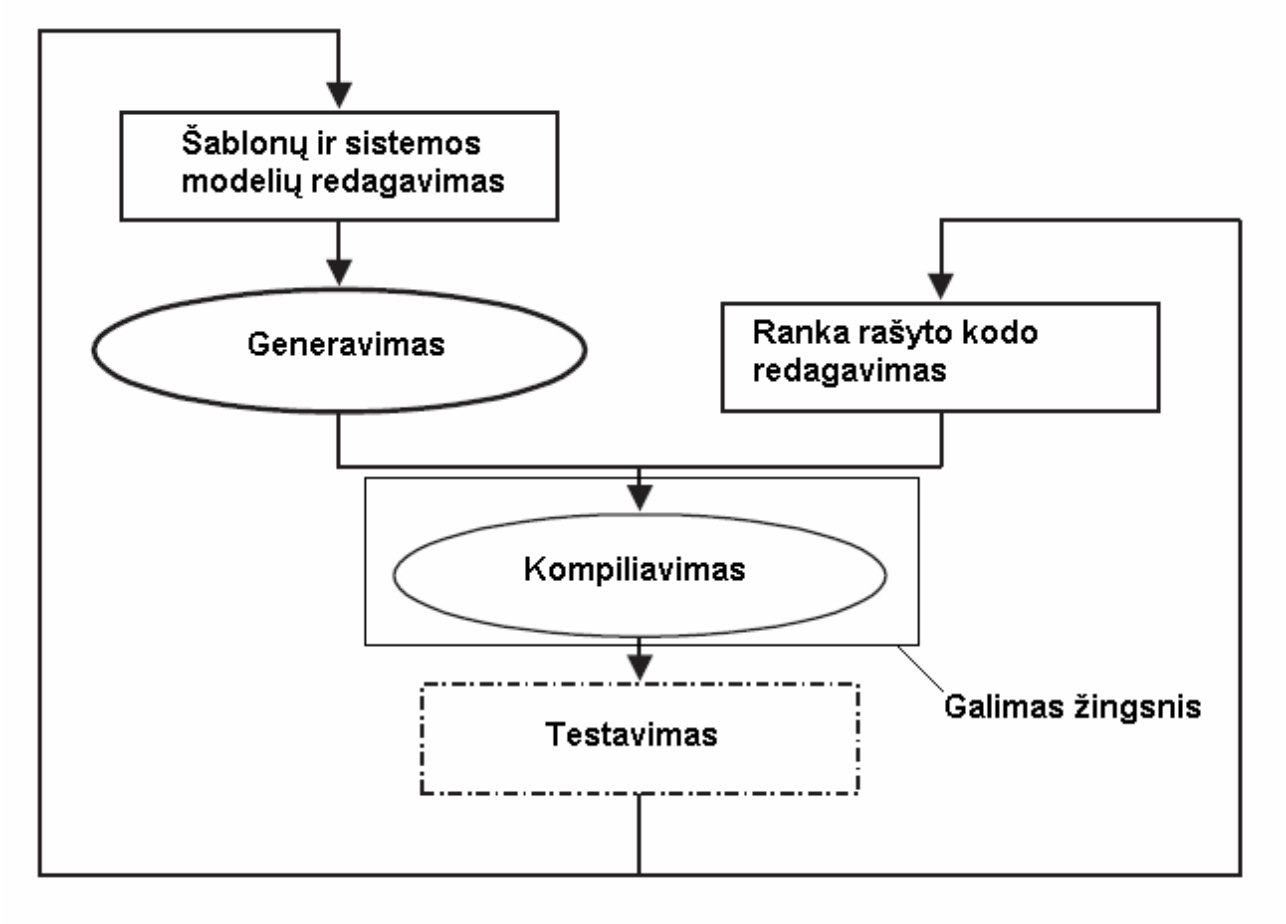
Vientisumas (neprieštaringumas, nuoseklumas, logiškumas).

Programavimo sąsajose (API), sukurtose panaudojant kodo generavimo technologijas, matomas vientisumas aprašuose bei kintamųjų pavadinimuose. Tai palengvina sistemų redagavimą rankinių būdu vėlesniuose etapuose, jei tai reikalinga, ir viršutinį sąsajos funkcionalumo kūrimo sluoksnį padaro vienodai paprastą.

Abstrakcija.

Kai kurie kodo generatoriai, kaip pradinius duomenis, naudoja abstraktų planuojamos sistemos modelį. Šis abstraktus modelis perteikia aukšto lygio planuojamos sistemos verslo taisykles. Tokiu būdu taisyklės yra matomos tiek analitikui, tiek projektuotojui (priešprieša tam būtų tai, kad ranka rašytame kode šios taisyklės yra paslėptos sistemos realizacijoje). Beje, abstraktus modelis daro sistemą lanksčia, nes leidžia taikyti

šablonus, naudojamus generatoriaus, kitoms programavimo kalboms, technologijoms ar projektavimo aplinkoms.



4 pav. Kodo generavimo procesas naudojant DSL[5]

Jei įsigilintume į sąvoką „kodo generavimas“, tai galima pastebėti, jog tai yra programos, kuri kuria programas, kūrimas[5]. Turėdami tokias sudėtingas ir galingas projektavimo aplinkas kaip Java 2 Enterprise Edition (J2EE) ar Microsoft .NET, mes tiesiog privalome panaudoti savo meistriškumą kurdami programas, kurios mums padėtų taikomųjų uždavinių kūrime. Tuo labiau, kad projektavimo aplinkos tampa vis sudėtingesni. O kuo jie sudėtingesni – tuo kodo generavimas atrodo patrauklesnis.

Vien programinės įrangos kūrimui sugaištamas laikas kelia daugybę problemų. Todėl renkantis kodo generavimą, šią problemą galima sumažinti. Tokios kompanijos kaip Nokia, Matsushita (Panasonic), Lucent ir Eads viešai pareiškė, kad DSM ženkliai pagerino jų programinės įrangos kūrimą. Lucent (dabar Alcatel – Lucent) pareiškė, kad jie pagerino savo produktyvumą nuo trijų iki keturių kartų, priklausomai nuo kuriamo produkto, kuomet jie pradėjo naudoti DSL. Nokia savo ruožtu pareiškė, kad taikomųjų programų kūrimą mobiliesiems telefonams pagerino dešimčia kartų, naudodami DSM įrankius. Tai leidžia šiai kompanijai pristatyti daugybę naujų įrenginių modelių kasmet.[3] Jungtinių valstijų oro

pajėgos atliko įdomų tyrimą, kurio metu palygino DSM su komponentais grįstu kūrimu. Jo metu prieita prie išvadų, jog DSL bei generatorių naudojimas ne tik tris kartus paspartina sistemų kūrimą, bet ir sumažina klaidų skaičių 50%.[4] Tai ypač aktualu toms pramonės šakoms, kurių kuriami produktai – kritinės sistemos.

Galima išskirti **generuoto kodo naudą**:

- dideli kiekiai ranka rašyto kodo dažnai yra labai skirtingi, nes kūrimo eigoje kūrėjai randa naujų ar geresnių metodų pasiekti tikslui. Kodo generavimas šablonų pagalba padeda to išvengti. Visas kodas yra suvienodinamas atlikus generavimą po šablonų atnaujinimo ar klaidų ištaisymo;
- sugeneruotas kodas yra logiškas, visi vardai yra sukurti pagal tas pačias taisykles, todėl toliau vystant projektus neiškyla nesusipratimų ir kodas yra lengvai suprantamas;
- įvykus pakitimui duomenyse, kodo generavimo architektūra įgalina kodo keitimą įvykdyti vienoje vietoje ir pergeneruoti reikiamus dokumentus, kas pakeičia duomenis visose reikiamose vietose. Jei kodas rašytas ranka, toks pakeitimas gali užimti labai daug laiko, nes net vieno kintamojo pasikeitimas mus įpareigoja pertikrinti visą kodą ir pakeisti jį rankomis;
- kodo generavimas leidžia daugiau laiko skirti sistemos dizaino tobulinimui ir prototipų testavimui, siekiant išvengti šių dalykų perdarymo (kas gali daryti įtaką ir architektūros pasikeitimui), nes sutaupoma laiko sistemos pritaikymui konkrečioje platformoje (šablonų pagalba). Pasikeitus technologijai – tiesiog pakeičiami (perrašomi) šablonai, o ne visas kodas, ką reiktų atlikti, jei sistema kuriama kodą rašant ranka;
- naudojant kodo generavimą, verslo taisyklės yra aiškiau matomos: kodo generatoriai naudoja abstrakčius apibrėžimo failus, pagal kuriuos ugdo generavimą; šie failai yra žymiai mažesni, nei gaunami kodo failai, tad juose daug lengviau įžvelgti apribojimus bei taisykles;
- kodo generatorius, naudojamas projekte – tai architektūrinių sprendimų realizacija, sukurta kūrimo metu. Tai suteikia kelis pranašumus:
  - programuotojai yra padrąsinami dirbti su architektūra;
  - jei atsiranda sunkumų priversti generatorių atlikti tai, ko iš jo norima, tai parodo, jog naujoji savybė nedera prie esamos architektūros;

- gerai aprašytas ir palaikytas generatorius ir toliau teikia nuoseklią kodo struktūrą bei siekiamą tikslą, net jei kuris nors komandos narys palieka projektą;

- kodo generavimas leidžia projektuotojams susikoncentruoti į sudėtingas ir rimtas problemas, taupo laiką ir taip leidžia projektą užbaigti greičiau;

1 lentelė. Generuoto kodo palyginimas su ranka rašytu

Generuotas kodas	Ranka rašytas kodas
Kodo kokybė yra vienoda visoms esybėms (angl. entity)	Kiekviena nauja klasė yra nežymiai geresnės kokybes nei prieš tai buvusi. Yra daugybe kopijuoto ir įterpto kodo. Kodo kokybė yra nevienoda visoms esybėms.
Kuomet reikia didelių pakeitimų, šablonai yra atnaujinami nauju kodu ir kodas yra sugeneruojamas iš naujo.	Kodo papildymai atliekami einant per visus objektus vienas paskui kitą.
Klaidos kode yra taisomos redaguojant šablonus ir pergeneruojant kodą. Visos buvusios (ištaisytos) klaidos yra sutvarkomos automatiškai.	Klaidos šalinamos viena po kitos. Jei klaida yra ne vienoje klasėje – jos visos turi būti ištaisytos rankiniu būdu.
Kadangi schemos ir biznio logika saugoma meta modelyje, generatoriai ir šablonai gali būti pritaikomi kitoms kalboms ar projektavimo aplinkoms.	Suderinamumo sluoksnis yra reikalingas, norint kodą pritaikyti kitai kalbai ar projektavimo aplinkai.

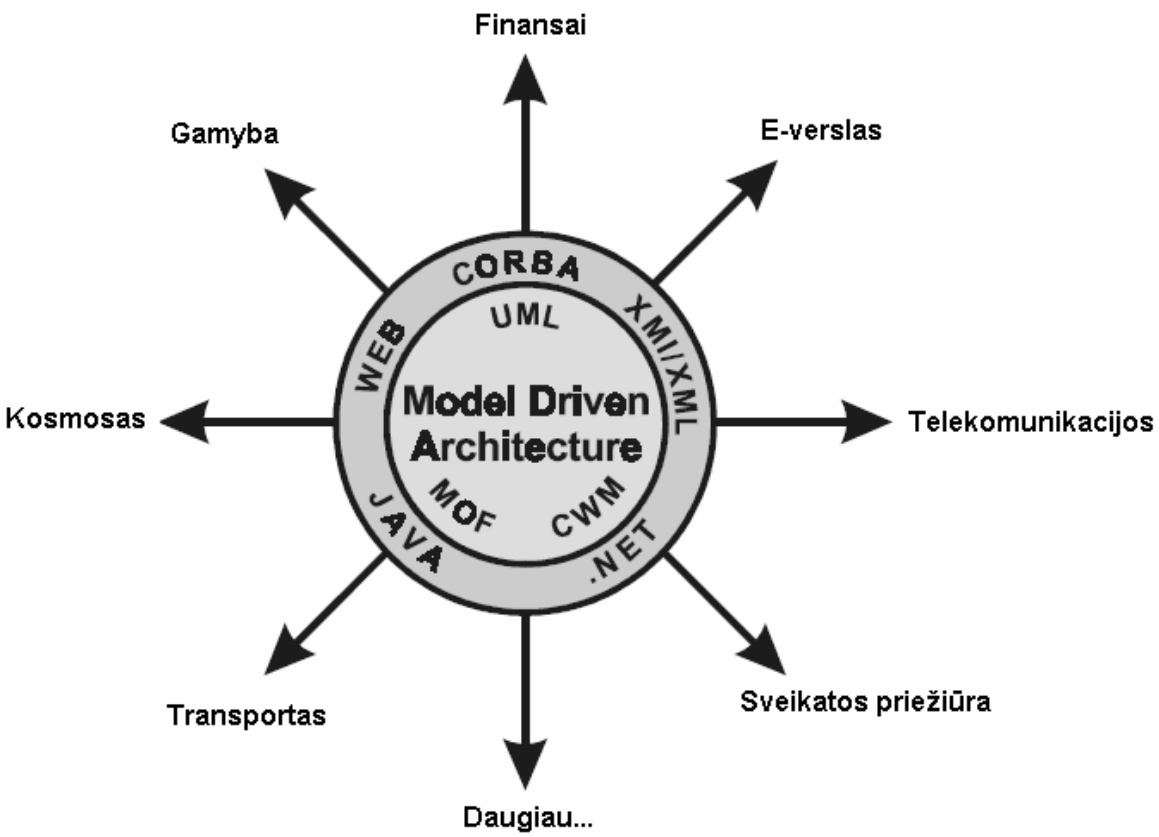
2.6. Panašių metodikų analizė

2.6.1. Modeliais grindžiama architektūra

Daugiausia panašumų su sričiai orientuotu modeliavimu galima rasti modeliais grįstos architektūros (angl. Model-Driven Architecture, toliau MDA) metode. Object Management Group (toliau OMG) pirmininkas ir vadovaujantis pareigūnas (angl. Chief Executive Officer, CEO) Ričardas Soleys (angl. Richard Soley) vienoje iš savo paskaitų tvirtino, jog MDA ir DSL (DSM) yra vienas ir tas pats, kadangi MDA proceso rezultate gaunamos DSL[6].

Modeliais grindžiama architektūra – tai siekis atskirti sistemos specifikavimo lygį nuo jos įgyvendinimo konkrečioje technologinėje platformoje, ko pasekoje gaunama architektūra, kuri yra nepriklausoma nei nuo konkrečios kalbos, nei nuo platformos ar gamintojo.

MDA taikomas pilnai vystyti sistemos modeliavimo, diegimo, integravimo ir taikomųjų programų valdymo gyvavimo ciklus, panaudojant tokius nemokamus standartus kaip UML, XML (angl. eXtensible Markup Language), XMI (angl. XML Metadata Interchange) ir CORBA (angl. Common Object Request Broker Architecture).



5 pav. MDA struktūra ir panaudojimas [11].

Modeliais grindžiamos architektūros sistemos modeliai yra labiau grindžiami funkcionalumu, o ne programavimo kalba, platforma ar technologija, kas reiškia, jog gerai sukurta ir MDA paremta sistema gali būti keičiama, išplečiama bėgant laikui, neardant sistemos branduolio. MDA priartina integravimo supaprastinimą, kūrimo laiko sutrumpinimą bei įmones resursų tausojimą, leisdama vystyti daugiau sprendimų nereikalaudama papildomo laiko ar personalo.

MDA terminologijoje žodis modelis apibūdina sistemos bei jos veikimo aplinkos aprašymą. Dažniausiai tai būna diagramų bei tekstinio aprašymo kombinacija. Modeliais grindžiamas projektavimas – tai toks projektavimas, kuomet modelis susieja sistemos projektavimą, kūrimą ir palaikymą. Iš to natūraliai seka, kad modeliais grindžiama

architektūra – tai tokia architektūra, kuomet sistemos architektūra yra gaunama iš sistemos modelio.

Modeliais grindžiamoje architektūroje naudojami trys požiūriai:

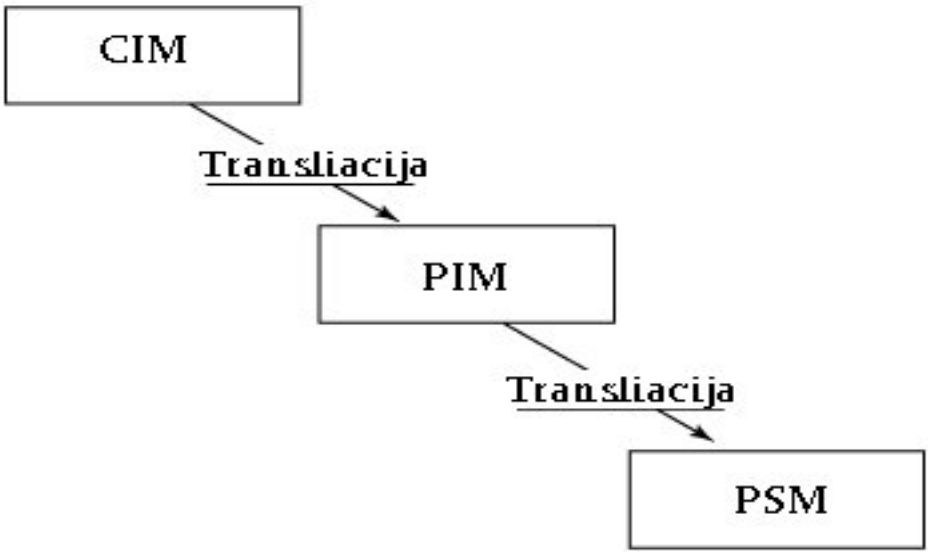
- nuo skaičiavimų nepriklausantis požiūris (angl. Computation Independent Viewpoint, CIV) – tai pirmasis MDA požiūris, kurio vaidmuo yra atskirti sistemos fundamentalią logiką ir nuo platformos priklausančią sistemos detalizaciją. CIV akcentuojamas į sistemos keliamus reikalavimus bei jos veikimo aplinką. Šioje dalyje sistemos struktūra bei įgyvendinimas yra paslėpti arba dar nėra apibrėžti;
- nuo platformos nepriklausomas požiūris (angl. Platform Independent Viewpoint, PIV) – tai antrasis požiūris, kuris akcentuojamas į sistemos veikimą neatsižvelgdamas į nuo platformos priklausančias savybes. Jame gali būti panaudotos bendrosios paskirties, nuo platformos nepriklausančios modeliavimo kalbos, tokios kaip UML;
- nuo platformos priklausantis požiūris (angl. Platform Specific Viewpoint, PSV) – tai trečiasis ir paskutinysis požiūris, kuris akcentuojamas į sistemos įgyvendinimo detales konkrečioje platformoje.

Kiekvienas iš šių požiūrių, žinoma, turi ir savo modelį. Tai yra:

- nuo skaičiavimų nepriklausantis modelis (angl. Computation Independent Model, CIM), kuris parodo sistemos veiklos modelį ir dažniausiai sudaromas veiklos analitiko;
- nuo platformos nepriklausantis modelis (angl. Platform Independent Model, PIM) – tai sistemos funkcionalumo modelis, kurį sudaro sistemos projektuotojas;
- nuo platformos priklausantis modelis (angl. Platform Specific Model, PSM), kuriame modeliuojama PIM modelyje aprašytos sistemos įgyvendinimas vienoje ar keliose skirtingose platformose.

Modelio transformacijos

Tikroji modeliais grindžiamos architektūros vertė slypi tame, kad CIM modelis gali būti nesunkiai sutransliuojamas į PIM modelį, panaudojant nesudėtingą žymėjimą (angl. mapping). Taip pat PIM modelis gali būti transliuojamas į PSM, o PSM – į kodą. Čia pagrindiniai elementai yra žymėjimas ir MDA įrankis ar įrankiai, kurie ir atlieka transliaciją.



6 pav. MDA transformacijos

Paprastai yra naudojami du lygiai: nuo platformos nepriklausantis modelis ir nuo platformos priklausantis modelis. Jie atvaizduojami standartinėmis unifikuotos modeliavimo kalbos priemonėmis ir nepadidina abstrakcijos lygio. Labai svarbu paminėti, kad nuo šiame kontekste termino „platforma“ negalima sutapatinti su .NET, J2EE ar panašiomis platformomis, kurių abstrakcijos lygmuo yra bent vienu žingsniu aukščiau už šioje metodologijoje vartojamą „platformos“ prasmę[7].

Modeliais grindžiamoje architektūroje, kiekvienoje stadijoje reikia redaguoti modelius, juos vis papildyti naujomis detalėmis, atlikti tiesioginę ir atvirkštinę inžineriją ir taip, galų gale, gauti realų kodą iš galutinio modelio. Šia metodologija OMG siekia sukurti galimybę tą patį modelį (PIM) naudoti su skirtingose platformose, standartizuoti visas transformacijas ir modelių formatus, kad modelius būtų galima naudoti su skirtingų gamintojų įrankiais. Tai ir yra pagrindinis skirtumas tarp MDA ir DSM.

Kiti MDA ir DSM skirtumai:

- naudojant modeliais grindžiamą architektūrą galima sugeneruoti iki 70% programinio kodo, tuo tarpu sričiai orientuotas modeliavimas leidžia sugeneruoti iki 100% programinio kodo [6];
- MDA reikalinga tiesioginė ir atvirkštinė inžinerija, kad modelis ir kodas atitiktų vienas kitą, o DSM viskas generuojama iš modelio, naudojant transformavimo šablonus, o visi keitimai atliekami juose;
- MDA kūrime galima naudoti tik konkrečius elementus (diagramas ir jų elementus), apibrėžtus standartais, kuomet DSM mums patiems leidžia susikurti reikiamus elementus.

2.6.2. Unifikuota modeliavimo kalba

UML (angl. Unified Modeling Language) – tai specifikavimo kalba, naudojama programinės įrangos projektavimo srityje. Ji gali būti apibrėžta kaip bendrojo naudojimo kalba, kur grafinių žymėjimų pagalba galima sukurti abstrakčius modelius. Tuomet šie abstraktūs modeliai gali būti naudojami sistemoje. Ši sistema vadinama UML modeliu. Object Management Group (toliau OMG) yra atsakinga už UML apibrėžimą ir jie tai atlieka per UML metamodelį.

UML paprastai naudojama atvaizduoti ir projektuoti informacinėms sistemoms. Kadangi informacinės sistemos tampa vis sudėtingesnės, tai ir jų kūrėjams šių sistemų kūrimas tampa vis didesniu iššūkiu, sukuriant jas per ganėtinai trumpą laiką. Dėl šių priežasčių jos dažnai turi klaidų ir šių klaidų pašalinimas užtrunka nevieną savaitę. Tačiau tai yra laikas, kuris iššvaistomas veltui, ko būtų galima išvengti pasirinkus technologijas, kurių pagalba klaidų kiekis gali būti ženkliai sumažintas dar iki pradedant programavimo darbus.

Reikia pabrėžti, kad UML neapsiriboja tik paprastu sistemų modeliavimu. Ji gali būti panaudota sistemų projektavimui, verslo procesų ir organizacijos struktūros atvaizdavimui. Speciali kalba, pavadinta SML (angl. Systems Modeling Language) buvo sukurta valdyti sistemoms, kurios yra apibrėžtos UML 2.0 kalba. UML yra svarbi dėl daugybės priežasčių, bet visu svarbiausia dėl to, kad ji paskatino pažangesnių, modeliais grindžiamų technologijų, tarp kuriu yra ir MDD (angl. Model Driven Development) bei MDA, atsiradimą bei vystymąsi.

Kadangi UML kalboje pabrėžiama grafinio žymėjimo svarba, tai ji labai tinkama atvaizduoti elgsenoms, klasėms ar asociacijoms. Tokios kalbos kaip UML leidžia sistemų kūrėjams koncentruotis ties kuriamų sistemų struktūra ir projektavimu. Taip pat reikia pastebėti, kad UML modeliai gali būti transformuoti į įvairiausias kitas išraiškas, ir dažnai neįdedant į tai daug pastangų.

Kaip ir paminėta aukščiau, UML buvo skirta tapti vieninga kalba, leidžiančia informacinių technologijų profesionalams modeliuoti informacines sistemas. Viso to pradininkai buvo Džimis Rumbautas (angl. Jim Rumbaugh), Aivaras Jakobsonas (angl. Ivar Jacobson) ir Greidis Butčas (angl. Grady Booch), kurie turėjo savo konkuruojančius metodus (OMT, OOSE ir Booch). Galiausiai jie suvienijo savo jėgas ir sukūrė atvirą standartą. Viena iš priežasčių, kodėl UML tapo standartine modeliavimo kalba yra ta, kad ji nepriklauso nuo jokios programavimo kalbos (pavyzdžiui IBM Rational modeliavimo įrankiai plačiai naudojami tiek su J2EE, tiek su .NET platformomis). Taip pat UML visuma – tai ne

metodika, o kalba. Tai yra svarbu todėl, kad kalba, o ne metodologija, gali lengvai įsiterpti į bet kokį kompanijos siekį valdyti verslą be kažkokių pasikeitimų.

Kalbant apie UML, labai svarbu atskirti sistemos diagramas ir UML modelį. Sistemos diagrama – tai grafinių simbolių visuma, skirta išreikšti sistemos modeliui. Modelis tuo tarpu yra dokumentas, skirtas apibrėžti, susieti diagramas ir modelio elementus. Yra trys modelių tipai:

- objektinis modelis (angl. object model),
- funkcinis modelis (angl. functional model), ir
- dinaminis modelis (angl. dynamic model).

Kiekvienas iš šių modelių atlieką konkretų vaidmenį apibūdinant sistemą.

Objektinis modelis yra atsakingas už sistemos struktūros ir jos pagrindo parodymą ir tai atlieka panaudojant ryšius, atributus bei operacijas. Klasių diagramos tiksliausiai iliustruoja šio tipo modelį.

Kaip galima spręsti iš pavadinimo, funkcinis modelis parodo sistemos funkcionalumą, žiūrint į sistemą iš vartotojo pozicijos. Aiškiausias to pavyzdys – panaudojimo atvejų diagramos.

Dinaminis sistemos modelis atsakingas už vidinės sistemos elgsenos atvaizdavimą. Tas parodoma per būsenų, sekų bei veiklos diagramas.

Taip pat reikia paminėti, kad modeliais gali būti keičiamasi tarp įvairių UML įrankių naudojantis XMI bylų formatu.

Įvertinus visa tai, galima pastebėti, jog unifikuota modeliavimo kalba labiausiai tinkama sistemos modeliavimo etape, kuomet reikia aprašyti sistemos veikimą, jos reikalavimus. Kodo generavimas UML pagalba yra ganėtinai prastas: dažniausiai klasių modelis konvertuojamas į klases C ar Java programavimo kalboje. Modeliavimo procesas su tokiu rezultatu nėra labai naudingas, todėl reikalinga pasitelkti papildomas priemones, kad būtu gaunami geresni rezultatai: modeliais grindžiamos architektūros naudojimas (kaip minėta, ji paremta UML diagramomis) arba UML profilius (angl. UML profiles), ko naudojimas taip pat turi savų neigiamų savybių.

Kita priežastis, kodėl efektyvaus kodo generavimas naudojant UML nėra pasiekiamas, tai dažnai daroma prielaida, kad ši modeliavimo kalba yra per didelė ir neaiški. Iš to seka, kad generuojamas kodas yra prastos kokybės (generuojamos tik klasės) ir daroma prielaida, kad naudojant UML negalima išreikšti kažkokių abstraktesnių ar specializuotų elementų[2].

Nuo unifikuotos modeliavimo kalbos atsiradimo labai daug kas pasikeitė: atsirado daugybė naujų standartų ir technologijų, kompiuterinių sistemų panaudojimo galimybės

išsiplėtė (programos veikia serveriuose, stacionariuose kompiuteriuose, delniniuose kompiuteriuose, mobiliuosiuose telefonuose), atsirado internetiniai servisai (angl. web service) ir panašiai. Viso to atvaizduoti modeliuose ir pilnai aprašyti UML pagalba tampa neįmanoma. Žinoma, vėlgi galima pasitelkti papildomas technologijas, tačiau tuomet visas modeliavimo procesas tampa per daug sudėtingas, tad tenka ieškoti paprastesnių sprendimo metodų.

Viską apibendrinus galima teigti, kad UML pagrindinė paskirtis – sistemų modelių kūrimas, kurių pagalba projektuotojai gali bendrauti tarpusavyje ar su klientais. Pagrindinės diagramos yra aiškios ir suprantamos visiems projektuotojams, kas ir toliau skatina šią modeliavimo kalbą naudoti. Tačiau norint modeliavimo pagalba sukurti veikiantį produktą, geriau bandyti naudotis kitomis modeliavimo metodikomis.

2.7. Analizės išvados

Apžvelgus plačiausiai naudojamas sistemų kūrimo metodikas, paremtas modeliavimo principais galima teigti, kad visos jos turi bendrą tikslą – sugeneruoti kaip įmanoma daugiau galutinio produkto iš sukurtų modelių.

Visos jos palengvina sistemų projektuotojų darbą, tačiau jų panaudojimo naudingumas skiriasi priklausomai nuo to, kokios sistemos bus kuriamos, kaip ir kam bus naudojamos. Tad labai svarbu prieš pradedant kurti sistemą, tinkamai pasirinkti sistemos kūrimo metodiką.

Sričiai orientuota kalba – tai gana naujas požiūris į sistemų kūrimą (nors jos naudojimo pavyzdžių galima rasti ir jau kurį laiką naudojamuose produktuose), todėl daugelis programinės įrangos gamintojų (Microsoft, MetaCase) bando šią metodiką diegti į savo produktus ir taip skatina modeliais pagrįsto sistemų kūrimo vystymąsi. Net nesudėtingoms sistemoms kurti, ar atskiriems jų komponentams rekomenduojama naudoti šiuos sprendimus. Būtent šiems veiksmams sričiai orientuota kalba yra ypač tinkama.

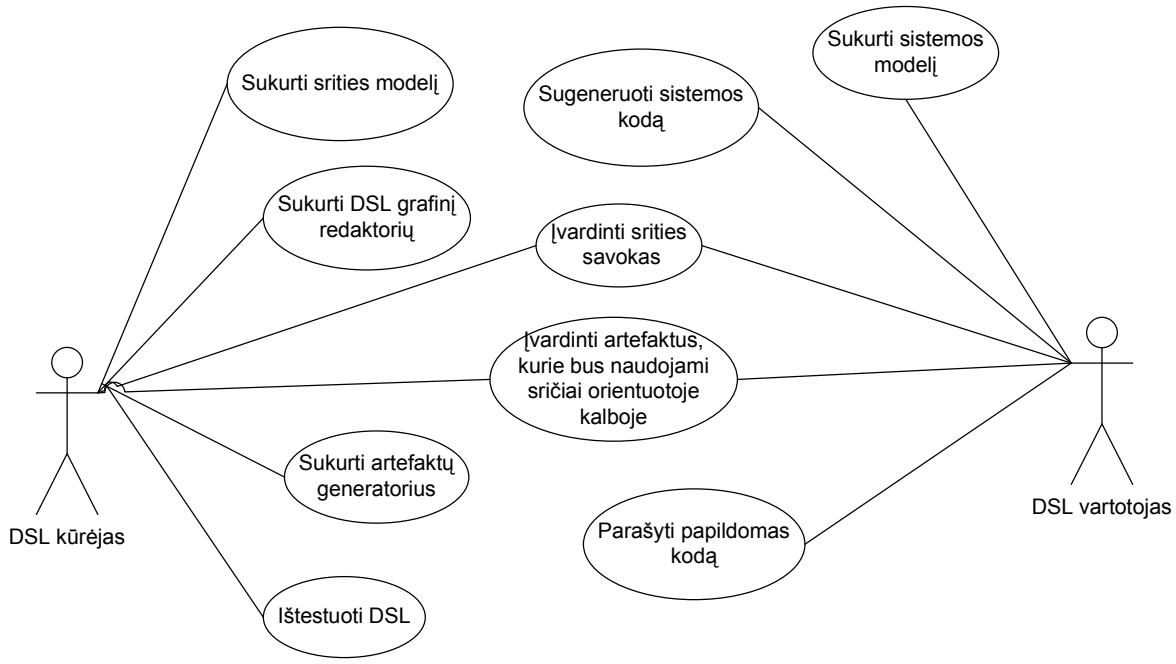
3. Projektiniai sprendimai

3.0. Projekto tikslas

Projekto tikslas yra sukurti duomenų atvaizdavimo ir jų kitimo stebėjimo sistemą. Sistema skirta naudoti bet kuriems įmonės darbuotojams, kuriems reikalinga stebėti ir gauti duomenis. Pagrindinis sistemos tikslas – atvaizduoti duomenis paprasta ir lengvai suprantama forma, kurie saugomi duomenų bazėje ir realiu laiku yra į ją įrašomi. Sistema turi palengvinti ir supaprastinti duomenų išgavimo iš duomenų bazės ir pateikimo galutiniam vartotojui procesą taip pat turėti savybę nesudėtingai plėsti jos funkcionalumą.

Kadangi pasirinkta projektą realizuoti panaudojant sričiai orientuotą kalbą, todėl kuriant ją reikia vadovautis DSM metodikos pateikiama DSL kūrimo eiga:

- srities sąvokų įvardinimas;
- artefaktu, kurie bus naudojami sričiai orientuotoje kalboje, įvardinimas;
- srities modelio sudarymas;
- sričiai orientuotuos kalbos grafinio redaktoriaus sukūrimas;
- artefaktu generatorių sukūrimas;
- tikrinimų ir apribojimų realizavimas;
- sričiai orientuotos kalbos testavimas ir realizavimas.



7 pav. Sistemos kūrimo etapų pasiskirstymas tarp DSL aktorių

3.1. Sričiai orientuotos kalbos kūrimas

3.1.1. Srities sąvokų įvardinimas

Norint pradėti kurti sričiai orientuotą kalba, pirmiausiai reikia įvardinti sąvokas, kurios bus naudojamos sričiai orientuotoje kalboje ir priklauso probleminei sričiai. Pagrindinės sąvokos yra šios:

forma: langas, kuriame bus atvaizduojami duomenys.

meniu: meniu pagalba nuspręsta realizuoti sistemos funkcionalumą.

atvaizdavimas: nurodoma, kokiame elemente bus atvaizduojami duomenys.

parametras: nurodo, kokius parametrus paduosime duomenų užklausimui.

tipas: kokius veiksmus atliks sistema meniu punkto paspaudimo metu.

periodinė: nurodo, ar veiksmai atliekami periodiškai (realaus laiko duomenų atvaizdavimo gavimui).

periodas: kokiu dažniu reikalinga atnaujinti duomenis.

3.1.2. Artefaktų, kurie bus naudojami sričiai orientuotoje kalboje, įvardinimas

Artefaktai – tai informacija, kuri gali būti sukurta, keičiama ar naudojama proceso eigoje. Artefaktai gali būti įeities ar išeities duomenys darbuotojui vykdant veiklą.

Kuriant sistemą naudojantis sričiai orientuota kalba, planuojama sukurti ir panaudoti šiuos artefaktus:

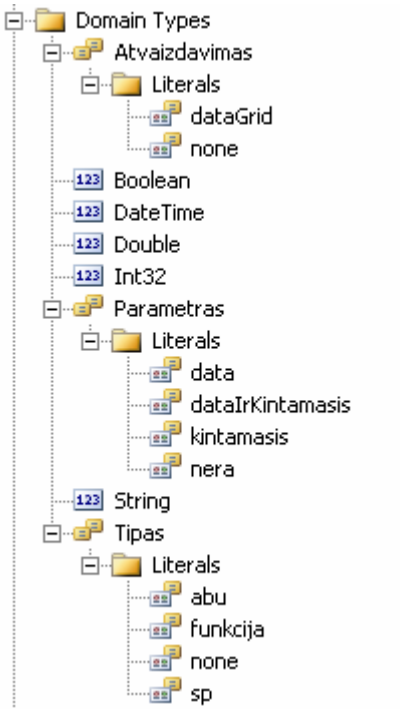
- srities modelį;
- sričiai orientuotos kalbos dizaineris;
- sistemos modelis;
- transformacijos šablonai;
- programinis kodas.

Kadangi viena svarbiausių transformacijų, kurias leidžia atlikti įrankis „DSL Tools“ – tai sukūrimas (kitų) artefaktų, kaip paprastų tekstinių failų, kurie gali būti: programinis kodas, duomenų bazių „scriptai“, dokumentacija ir kiti su problemine sritimi susiję dokumentai, todėl pagrindinis tikslas – veikiančios sistemos gavimas iš sistemos modelio, pasinaudojant visais sukurtais artefaktais.

3.1.3. Srities modelio sudarymas

Prieš pradedant kurti srities modelį, reikia susikurti galimus naudoti srities atributų tipus. Sukūrus projektą, automatiškai yra sugeneruojami standartiniai tipai, naudojami įprastinėse programavimo kalbose. Kuriant sričiai orientuotą kalbą, mums reikia susikurti naujų tipų, kurie apibrėžtų tam tikras srities savybes. Tam mes susikuriame sekančius atributų tipus (visi naudojami tipai pateikti 7 paveiksle):

- atvaizdavimas;
- parametras;
- tipas.



8 pav. Srities atributų tipai

Tipas „atvaizdavimas“ gali įgyti reikšmes „dataGrid“ ir „none“. Šis tipas skirtas pasirinkti kokiame elemente bus atvaizduojamas rezultatas: DataGrid komponente, arba neatvaizduojamas išviso.

Tipas „parametras“ naudojamas nustatyti, kokie parametrai bus paduodami numatytam objektui. Šie parametrai gali būti: data, kintamasis, data ir kintamasis arba nei vieno.

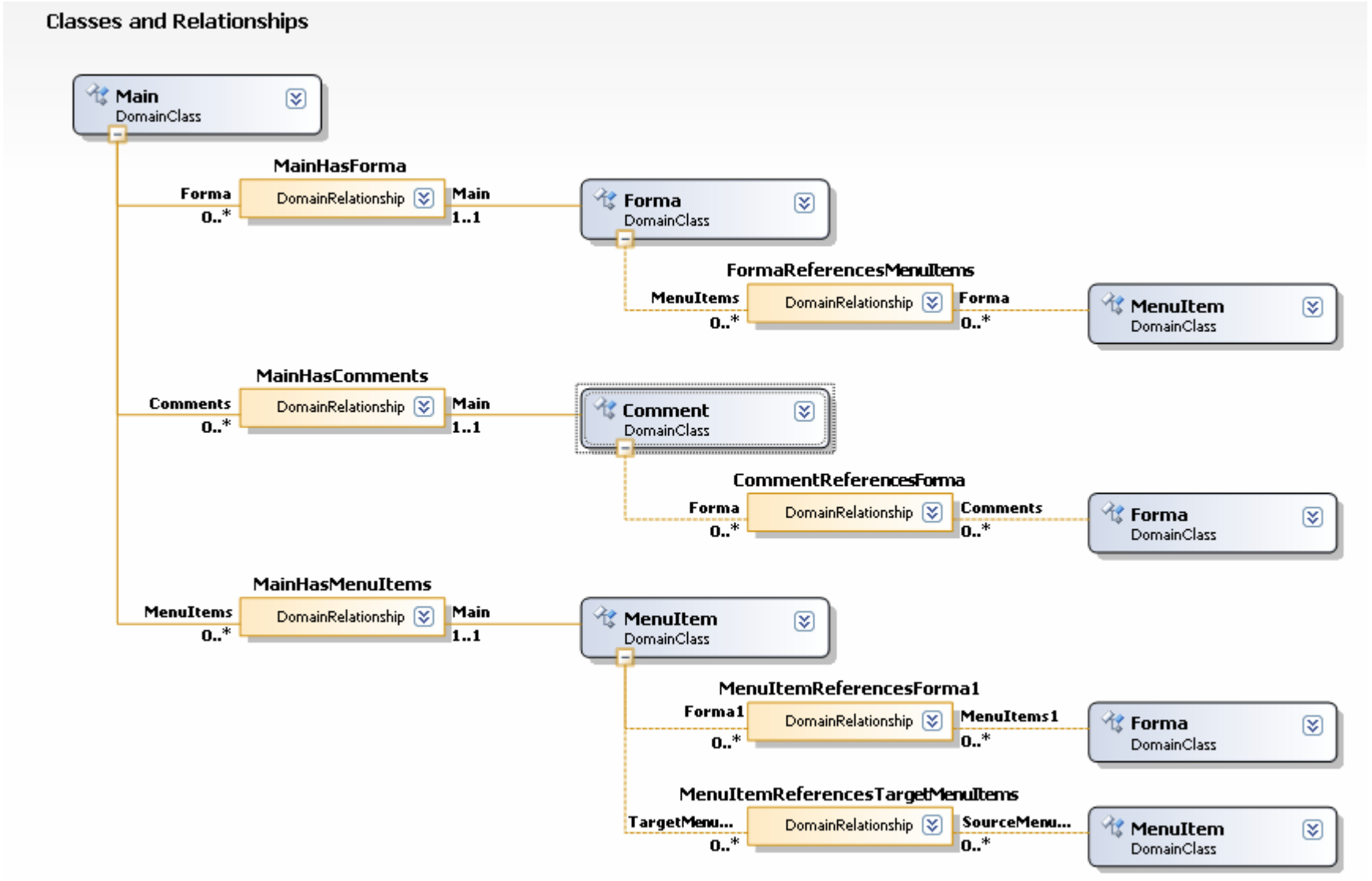
Tipas „tipas“ naudojamas nustatyti kokio tipo veiksmai bus atliekami. Tai gali būti: paprasta funkcija, išsaugota procedūra (angl. Stored Procedure, SP), ir paprasta procedūra ir išsaugota procedūra, arba nei viena iš jų.

Apsirašius reikiamus atributų tipus, sudaromas srities modelis. Bendra srities modelio diagrama pateikta paveiksle. Joje atvaizduoti visi srities elementai bei jų tarpusavio sąsajos. Šią diagramą galima pavadinti projektuojamos sričiai orientuotos kalbos metamodeliu. Joje figūruojantys elementai yra:

- srities klasė (angl. Domain Class);
- srities ryšys (ang. Domain Relationship).

Savo ruožtu srities ryšiai skirstomi į:

- įterpimo ryšius (angl. Embedding Relationship) ir
- nuorodos ryšius (angl. Reference Relationship).



9 pav. Bendra srities modelio diagrama

Srities modelyje naudojamų klasių pilnas aprašymas pateikiamas 2 lentelėje.

2 lentelė : Modelyje naudojamos srities klasės

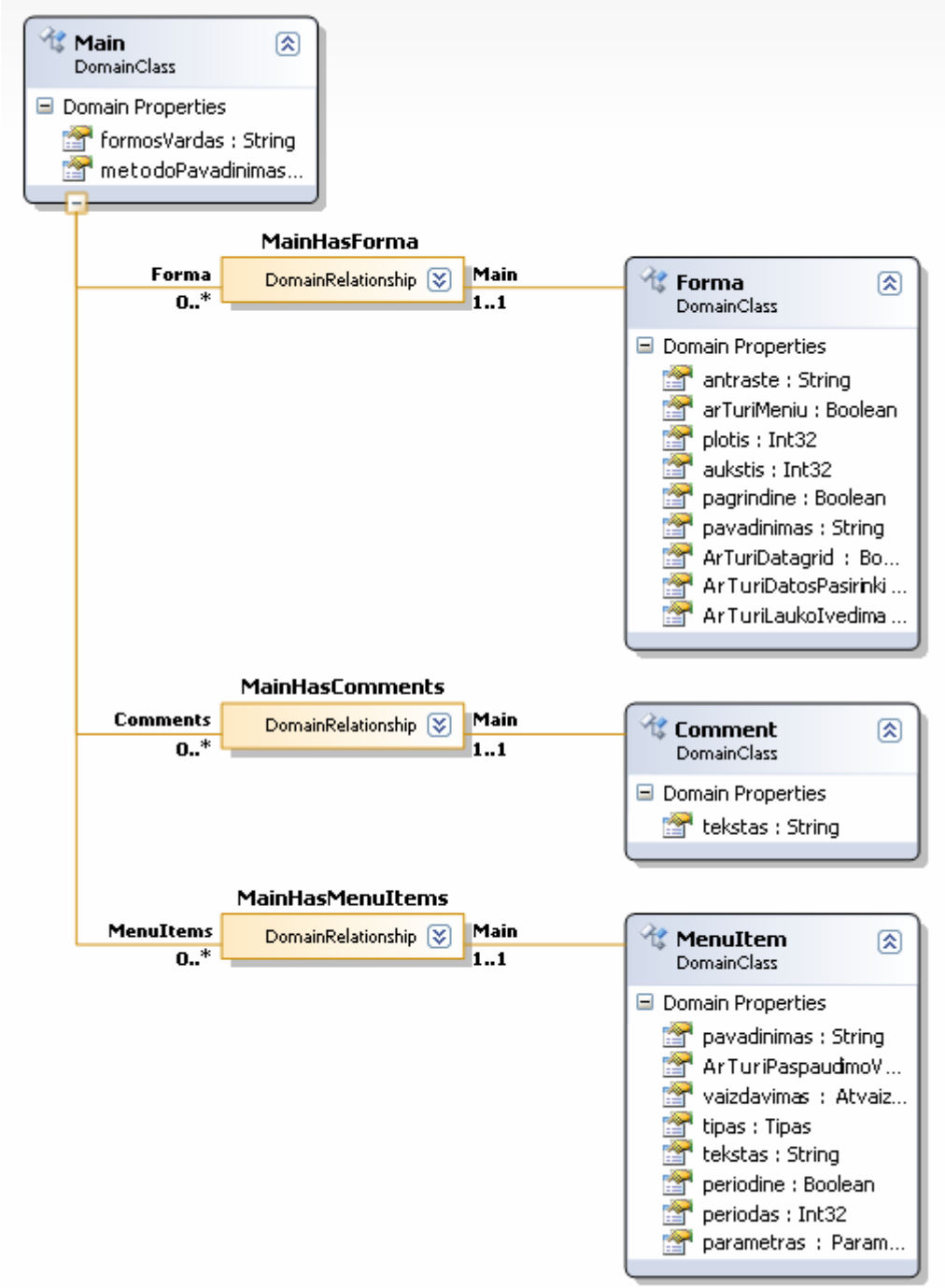
Srities klasės pavadinimas	Srities atributai	Aprašymas
Main	formosVardas metodoPavadinimas	Pagrindinė DSL klasė, kuria aprašomas sistemos paleidimas.
Forma	antraste arTuriMeniu plotis aukstis pagrindine pavadinimas ArTuriDatagrid ArTuriDatosPasirinkima Ar turiLaukoIvedima	Naudojama aprašyti kuriamai formai, suteikti reikiamus parametrus jos sukūrimui.
Comment	tekstas	DSL kūrimo grafiniame redaktoriuje leidžia sukurti komentarą formai (atsispindi tik grafiniame redaktoriuje).
MenuItem	pavadinimas ArTuriPaspaudimoVeiksma vaizdavimas tipas tekstas periodine periodas parametras	Aprašomi meniu punktai, jų paspaudimo metu atliekami veiksmai.

Pagrindinis modelio elementas yra srities klasė „Main“ (10 paveikslas). Tai tarsi pagrindas, ant kurio stovi visas modelis ir visa būsima kalba. Ji turi dvi savybes: „formosVardas“ ir „metodoPavadinimas“.

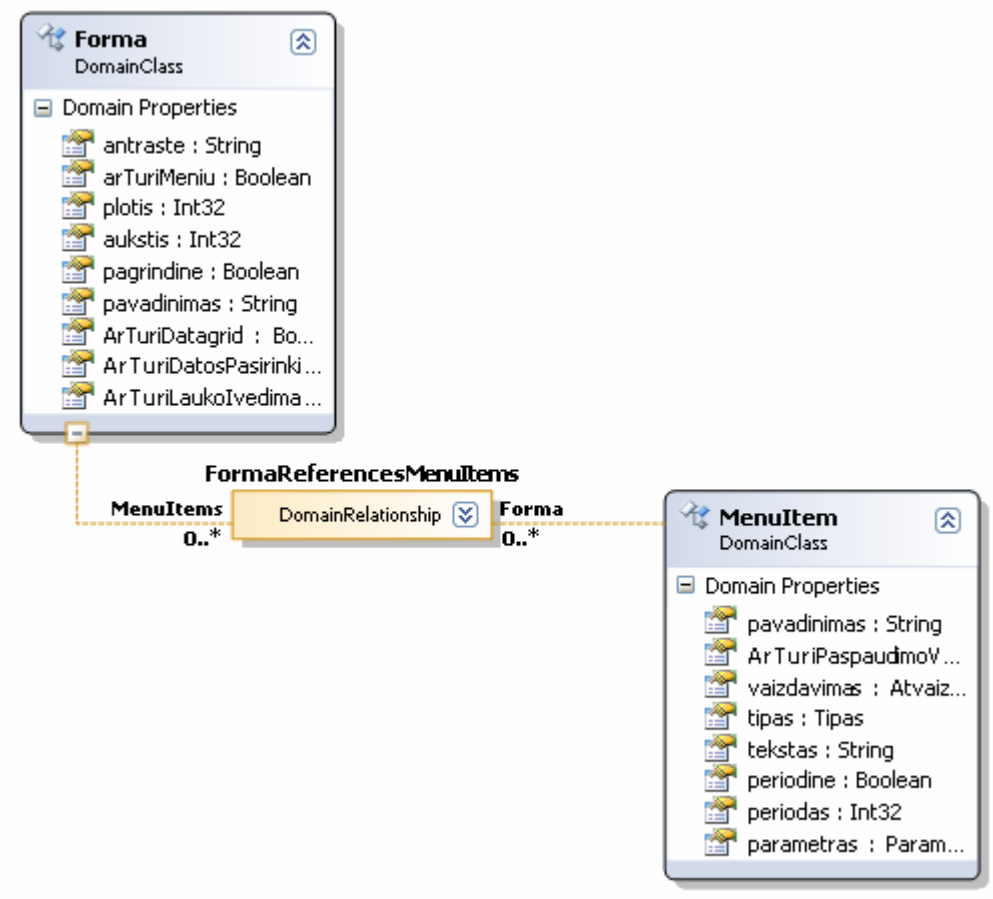
Savybė „formosVardas“ nurodo pagrindinės klasės pavadinimą.

Savybė „metodoPavadinimas“ nurodo metodo, kuris iškviečia pagrindinę formą pavadinimą.

Srities klasė „Main“, susieta su kitomis trimis srities klasėmis „Forma“, „Comment“ ir „MenuItem“ įterpimo ryšiais, kas nurodo jų priklausomybę. Tai leidžia šias klases talpinti (kurti) pirmojoje. Ryšiai yra apriboti. „Main“ klasė gali turėti neribojamą kiekį visų kitų klasių, tačiau visos jos gali priklausyti tik vienai „Main“ klasei.



10 pav. Srities klasės „Main“ sąryšis su kitomis klasėmis



11 pav. Srities klasės „Forma“ sąryšis su kitomis klasėmis

Srities klasė „Forma“ naudojama aprašyti standartinį vartotojo sąsajos elementą „Form“. Kuriant šį elementą atsižvelgiama į galimus nurodyti parametrus:

„antraste“: nustatoma „Form“ elemento antraštė;

„arTuriMeniu“: nustato, ar forma turės meniu juostą;

„plotis“: formos plotos;

„aukštis“: formos aukštis;

„pagrindine“: nustatoma, ar forma yra pagrindinė, t.y. ji paleidžiama pati pirmoji; tokia forma gali būti tik viena;

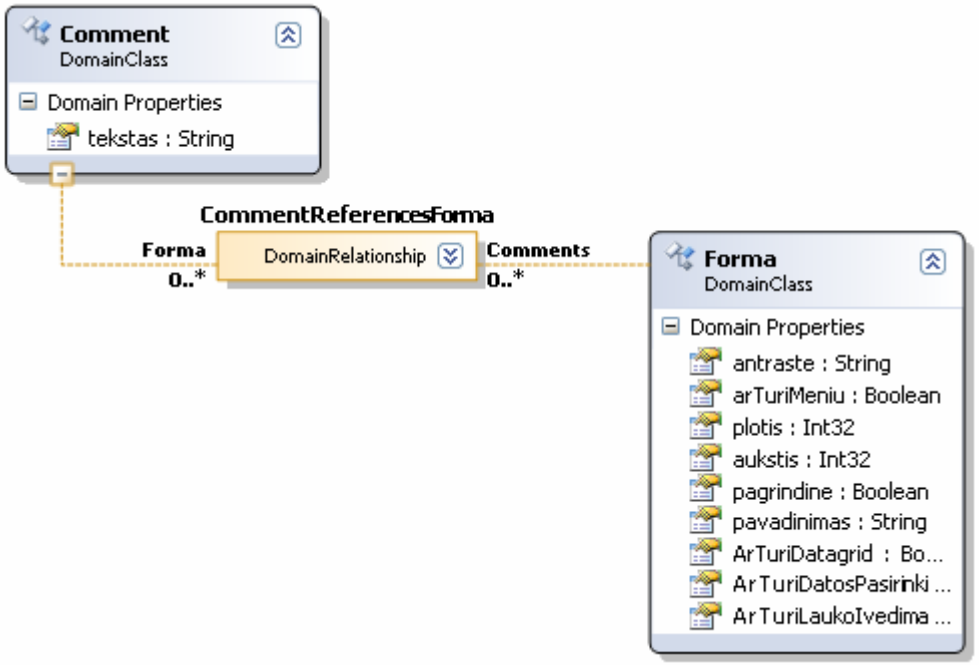
„pavadinimas“: formos klasės pavadinimas;

„ArTuriDatagrid“: nurodo, ar forma turi „DataGridView“ tipo elementą, kuris naudojamas duomenų atvaizdavimui;

„ArTuriDatosPasirinkima“: nurodo, ar forma turi „DateTimePicker“ tipo elementą, kuris leidžia pasirinkti datą ir pagal ją atvaizduoti duomenis;

„ArTuriLaukoIvedima“: nurodo, ar forma turi „TextBox“ tipo elementą, kuris leidžia duomenų užklausimui naudoti papildomus parametrus.

Srities klasė „Forma“ turi nuorodos ryšį su srities klase „MenuItem“. Tai leidžia sukurti nuorodą iš „Form“ klasės į klasę „MenuItem“. Ryšys nėra apribojamas visiškai ir tai leidžia formai turėti neribojamą skaičių meniu elementų.



12 pav. Srities klasės „Comment“ sąryšis su kitomis klasėmis

Srities klasė „Comment“ naudojama DSL kūrimo grafiniame redaktoriuje įterpti komentarą formoms. Tai įgyvendinta tam, kad būtų galima aiškiau suprasti ką kokia forma atlieka. Ši klasė turi tik viena srities savybę – „tekstas“. Ji saugo tekstą, kuris ir atvaizduojamas redaktoriuje.

Komentaro klasė susieta su formų klase nuorodos ryšiu, kas leidžia grafiškai atvaizduoti, kuriai formai komentaras skirtas. Jokių apribojimų šiam ryšiui nėra, todėl šiuos elementus galima susieti bet kokiu skaičiumi ryšių.

Srities klasė „MenuItem“ (pavaizduota 13 paveiksle) aprašo meniu elementus. Pagal šias klases ir jų sąryšius su kitomis srities klasėmis sugeneruojama kuriamos sistemos meniu struktūra ir to atliekami veiksmai. „MeniuItem“ klasė nuorodos ryšiais susieta su klasėmis „Forma“ ir savimi pačia. Tai leidžia kurti nuorodas iš meniu į formą ar kitą meniu elementą. Tokių ryšių gali būti neribojamas kiekis.

Meniu kūrimui gali būti nustatyti tokie parametrai:

„pavadinimas“: meniu punkto pavadinimas;

„ArTuriPaspaudimoVeiksma“: nurodo, ar paspaudus ant šio meniu punkto bus vykdomas nurodytas veiksmas;

„vaizdavimas“: nurodo, kur bus atvaizduojami rezultatai, jei bus kas nors vykdoma;

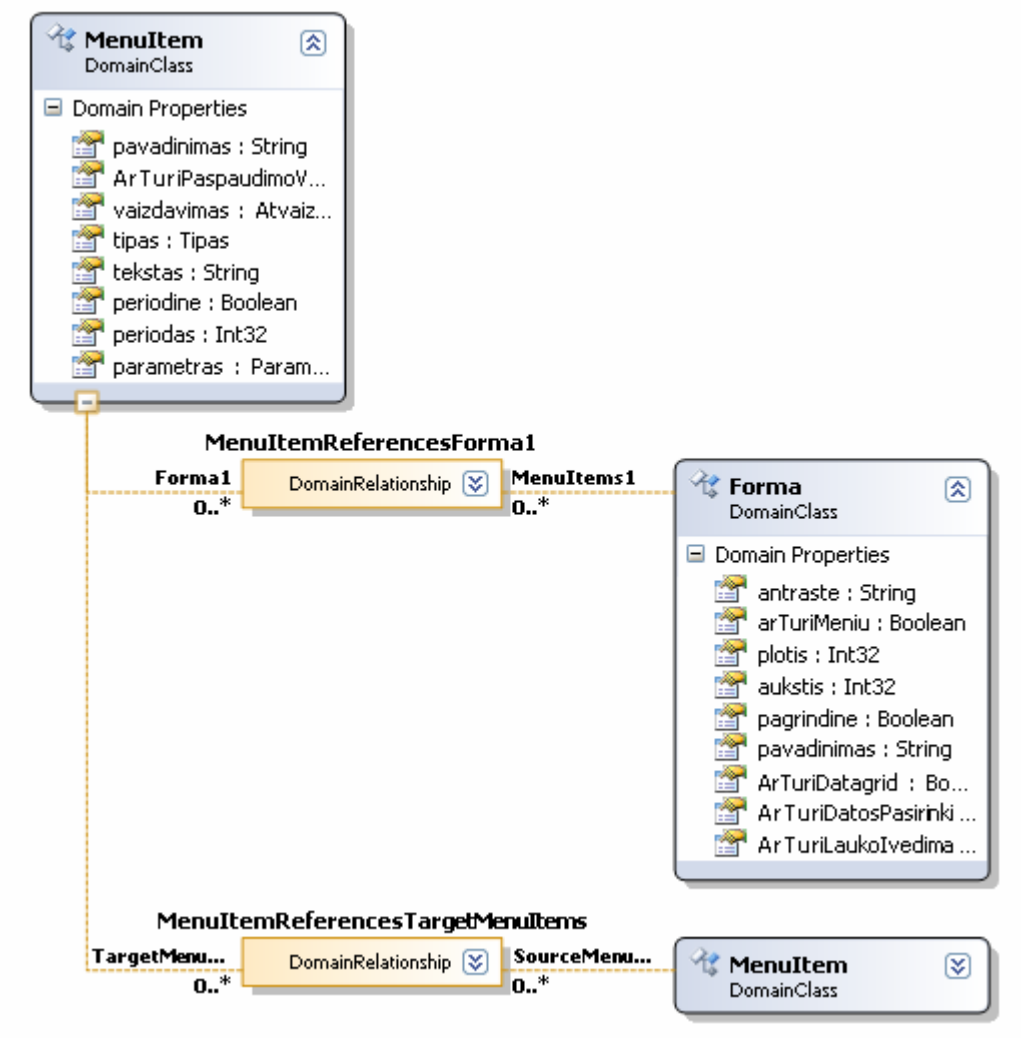
„tipas“: nurodo, kokio tipo veiksmai bus vykdomi;

„tekstas“: šis parametras nurodo užklausą, kuri bus vykdoma meniu pasirinkimo metu;

„periodine“: nurodo, ar veiksmai, atliekami šio meniu punkto bus atliekami periodiškai;

„periodas“: nurodo, kaip dažnai duomenys turi būti atnaujinami;

„parametras“: nurodo, ar bus ir kokie parametrai paduodami užklausai.



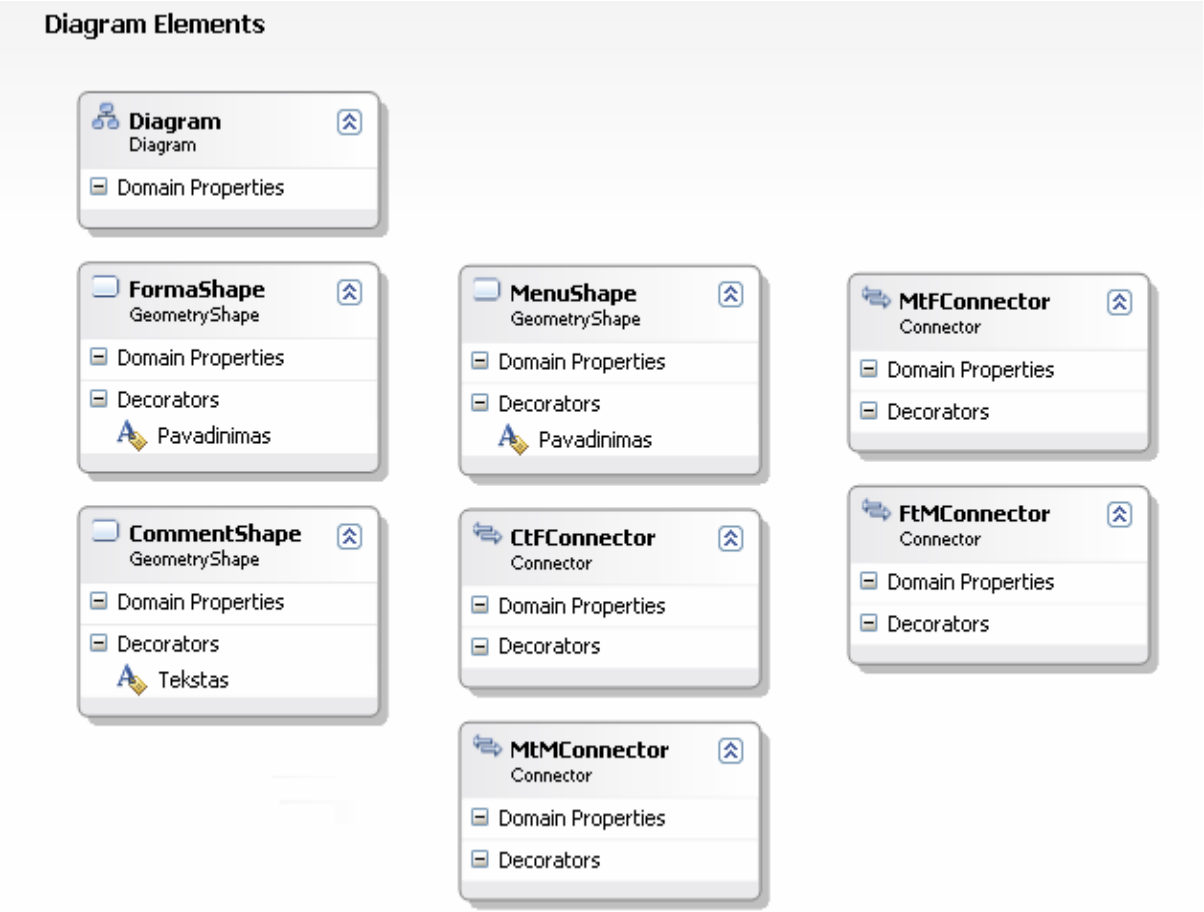
13 pav. Srities klasės „MenuItem“ sąryšis su kitomis klasėmis

3.1.4. Sričiai orientuotuos kalbos grafinio redaktoriaus sukūrimas

Kad galėtume kurti sričiai orientuotą kalbą naudodamiesi grafiniu redaktoriumi, reikalinga sukurti jo elementus:

- diagramos elementus ir
- įrankių juostos elementus.

Sukurti diagramos elementai pavaizduoti paveiksle ir aprašyti lentelėje.

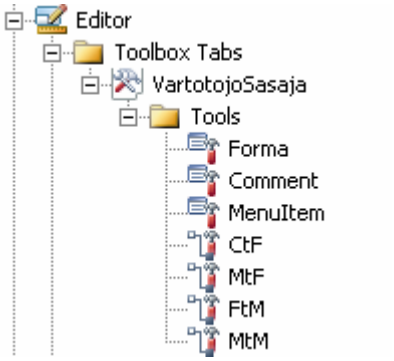


14 pav. : Diagramos elementai

3 lentelė. Diagramos elementai

Pavadinimas	Tipas	Aprašomas elementas
Diagram	diagrama	„Main“
FormaShape	geometrinė forma	„Forma“
CommentShape	geometrinė forma	„Comment“
MenuShape	geometrinė forma	„MenuItem“
CtFConnector	jungiamasis elementas	„CommentReferencesForma“
MtMConnector	jungiamasis elementas	„MenuItemReferencesTargetMenuItems“
MtFConnector	jungiamasis elementas	„MenuItemReferencesForma1“
FtMConnector	jungiamasis elementas	„FormaReferencesMenuItems“

Kad galima būtų atvaizduoti sukurtus diagramos elementus grafiniame dizaineryje, reikia sukurti įrankių juostos elementus. Jie pavaizduoti paveiksle ir aprašyti lentelėje.

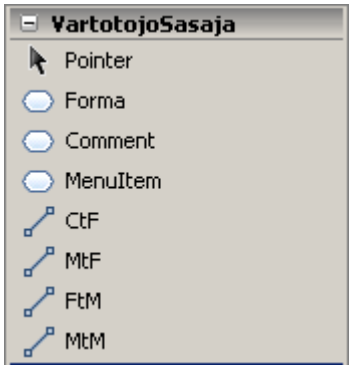


15 pav. Įrankių juostos elementai

4 lentelė. Įrankių juostos elementai

Pavadinimas	Aprašomas Diagramos elementas	Aprašymas
Forma	FormaShape	sukuria FormShape elementą
Comment	CommentShape	sukuria CommentShape elementą
MenuItem	MenuShape	sukuria MenuShape elementą
CtF	CtFConnector	sukuria CtFConnector elementą
MtF	MtFConnector	sukuria MtFConnector elementą
FtM	FtMConnector	sukuria FtMConnector elementą
MtM	MtMConnector	sukuria MtMConnector elementą

Aprašyta grafinio redaktoriaus struktūra sukuria paveiksle 16 pavaizduotą įrankių juostą, kurios pagalba galime kurti sistemos modeli naudodami sukurtą DSL.



16pav. Sugeneruota įrankių juosta

3.1.5. Artefaktu generatorių sukūrimas

Dauguma planuojamų naudoti artefaktų yra gaunami juos apsirašius įrankio pagalba: grafinio dizainerio pagalba. Kiekvienas jų yra naudojamas kitų artefaktų gavimui ar tiesiog yra susiejami tarpusavyje. To pavyzdys galėtu būti sričiai orientuotos kalbos modelio ir sričiai orientuotos kalbos dizainerio sąryšis, kuomet neturint modelio, negalima kurti ir dizainerio, nes modelio elementai naudojami aprašant atitinkamus dizainerio elementus.

„DSL Tools“ paketas atlieka didžiąją dalį artefaktų generavimo proceso, kas supaprastina DSL kūrimą, tačiau palieka galimybę keisti standartinius generavimo šablonus.

Kuriamos sričiai orientuotos kalbos artefaktas – programinis kodas, kuris turi būti sugeneruojamas automatiškai. Dėl šios priežasties pagrindiniu ir labai svarbiu darbu tampa transformavimo šablonų sukūrimas.

Kuriamoje sričiai orientuotoje kalboje, transformacijos šablonai naudojami programiniam kodui generuoti. Kadangi „DSL Tools“ yra „Visual Studio 2008“ programinio paketo dalis, tai ji palaiko visas šio paketo teikiamas savybes. Kodo generavimui svarbiausia savybė yra ta, kad galima naudoti dalines (angl. partial) klases, kuomet ta pati klasė gali būti aprašoma keliuose skirtinguose failuose. Tokiu būdu, transformacijos šablonų pagalba ir dalinių klasių panaudojimu, galima atlikti 100% veikiančio programinio kodo generavimą.

Transformacijos šablonų rašymas labai nedaug skiriasi nuo programavimo. „DSL Tools“ pakete jų rašymui naudojama C# kalba, tuo pačiu galima naudoti ir visas C# kalboje naudojamas funkcijas.

Visi šablonai turi standartinę antraštę, kurioje aprašomos apdorojimo instrukcijos:

```
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransfor
mation" #>
<#@ output extension=".cs" #>
<#@ VartotojoSasaja processor="VartotojoSasajaDirectiveProcessor"
requires="fileName='../Test.UIdsl'" #>
```

Pirmoji direktyva nurodo bendrąsias šablono apdorojimo taisykles.

Antroji direktyva nurodo, kokio tipo failas bus sugeneruotas iš šablono. Apribojimų tam jokių nėra. Pavaizduotu atveju bus generuojamas C# kalbos kodas.

Trečioji direktyva nurodo, koks modelis bus apdorojamas. Šablonas gali apdoroti tik vieną modelį, tačiau galima keliais modeliais aprašyti projektuojamą sistemą ir skirtingų šablonų pagalba, nurodžius skirtingus modelius jų apdorojimui, sugeneruoti bendrą programinį kodą.

Transformavimo šablonuose, apdorojimo kodas nuo tiesiogiai išvedamo kodo atskiriamas simboliais„<#” ir „#>“. Kodas, aprašytas tokiame bloke interpretuojamas kaip šablono darbinis kodas, ir išeities tekste yra nematomas. Šiuose blokuose dažniausiai aprašomi įvairūs ciklai, loginės sąlygos, vykdomi veiksmai, pagal kuriuos atliekamas generavimas.

Jei norima išvesti modelio elemento kažkokią reikšmę, tam naudojama tokia simbolių struktūra:

```
<#= ElementoPavadinimas.AtributoPavadinimas#>
```

Vietoje, kurioje bus šis įrašas, gausime nurodyto elemento atributo modelyje įvestą reikšme.

Kuriamoje sričiai orientuotoje kalboje yra naudojami trys transformavimo šablonai, kurie realizuoja:

- grafinės sąsajos kūrimą;
- kuriamos sistemos funkcionalumą;
- papildomus veiksmus, atliekamus su sistema (sukurto modelio paleidimą).

Pilni naudotų šablonų aprašai pateikti prieduose.

3.1.6. Apribojimų ir validavimo realizavimas

Žinant reikalavimus sistemai, reikalinga įvesti apribojimus. Kadangi sričiai orientuota kalba kuriama grafinio dizainerio pagalba, tai įgalina mus paprasčiau apriboti duomenis.

Dažniausiai vartojamus apribojimus galima suskirstyti į dvi kategorijas[1]:

- apribojimai, kurių vartotojai negali pažeisti (kietieji apribojimai);
- apribojimai, kuriuos vartotojas tam tikru metu gali pažeisti (minkštieji apribojimai).

„DSL Tools“ naudojami abiejų kategorijų apribojimai.

Pirmoji apribojimų kategorija realizuojama aprašant srities klasių atributus. Jiems priskyrus tam tikrus tipus – įrankis automatiškai vartotojui neleidžia įvesti reikšmių, kurios netenkina tipų reikalavimų. Tarkime: jei atributui „aukštis“ nurodėme, kad jo tipas yra „integer“ (sveikas skaičius), tuomet priskirdami jam reikšmė, kuri neatitinka šio tipo keliamų reikalavimų - gausime klaidą ir negalėsime tokios reikšmės įvesti.

Taip pat, jei naudojame specifines reikšmes (mums reikalinga, kad atributas įgauti tik konkrečias reikšmes), susikuriame naują tipą, jam priskiriame reikiamas reikšmes, ir tokiu būdu mūsų atributas galės įgauti tiktai jas.

Dalis kietųjų apribojimų realizuojami ir realizuojant sričiai orientuotos kalbos modelį. Jame šie apribojamai išreiškiami ryšių pagalba. Kiekviena srities klasė su kita, jei reikia, yra susieta tam tikru ryšiu, kuriam nurodomas kardinalumas (kardinalumas – tai tokia ryšio charakteristika, kuri rodo maksimalų kiekį vieno objekto elementų, surištų su vienu kito objekto elementu). Tokiu būdu vartotojas, realizuodamas DSL, negali susieti arba netinkamai

naudoti elementų (panaudojant skirtingų tipų ryšius, galima įvairiai manipuliuoti srities elementais), naudoti netinkamą jų kiekį.

Visus pirmosios kategorijos (kietuosius) apribojimus galima pamatyti bendroje sričiai orientuotos kalbos diagramoje (9 paveikslas) bei atskiruose srities klasių aprašuose (klasių savybių tipus, 9 – 13 paveikslai).

Antroji apribojimų kategorija „DSL Tools“ pakete realizuojama panaudojant valiavimo funkcijas. Jos leidžia DSL kūrėjui pačiam aprašyti specifinius reikalavimus modeliui, modelio elementams ar tam tikriems atributams ir juos patikrinti tam tikru momentu. Įrankis įgalina validavimą atlikti šiais momentais:

- įkrovimo metu;
- pasirenkant iš meniu;
- atidarymo metu;
- išsaugojimo metu;

Realizuojant projektą pasirinkta, kad validavimas bus atliekamas projekto išsaugojimo metu bei iš meniu (modelyje, dešiniojo pelės klavišo paspaudimo metu atsirandančio meniu). Šio pasirinkimo realizavimas pavaizduotas paveiksle.

Notes	
Uses Custom	False
Uses Load	False
Uses Menu	**True**
Uses Open	False
Uses Save	**True**

17 pav. Validavimo vykdymo momentai

Norint kažkuriam srities elementui (klasei) įgyvendinti validavimo metodą, reikalinga susikurti naują failą, kuriame turi būti dalinė šio elemento klasė, kurioje aprašomos elemento savybės, kurios bus tikrinamos (validavimo klasės ir metodai pateikiami prieduose). Visi validavimo metodai (kartu su klasėmis) turi vienodą struktūrą:

5 lentelė. Validavimo metodo struktūra

`[ValidationState(ValidationState.Enabled)]` `public partial class Forma`	Klasės antraštė

`[ValidationMethod(ValidationCategories.Menu |`
`ValidationCategories.Save)]`
`private void ValidateName(`
`ValidationContext context)`
	Metodo antraštė

`if (string.IsNullOrEmpty(this.pavadinimas))`
`{`
`  context.LogError("Forma privalo tureti pavadinima", "FN1001", this);`
`}`
	Tikrinimas

Pirmoji dalis – tai klasės antraštė. Joje aprašoma kokiam objektui (srities klasei) bus taikomos taisyklės. Validavimas realizuojamas panaudojant dalines klases, kurios suteikia galimybę keliuose failuose apsirašyti tą patį objektą (klasę). Prieš klasės pavadinimą einantis tekstas „[ValidationState(ValidationState.Enabled)]“ nurodo, kad žemiau bus aprašytas objekto validavimas ir jis (validavimas) įjungiamas.

Toliau seka validavimo metodai su savo antraštėmis. Šios antraštės nurodo, kad tai yra validavimo metodas ir kokiu metu bus jis atliekamas. Taip pat metodui būtina paduoti (tai atliekama automatiškai) validavimo aplinką (sukurtą sistemos modelį) „ValidationContext context“. Pagal šiuos aprašus įrankis automatiškai nustato ką ir kada reikia validuoti.

Susikūrus metodus, reikalinga aprašyti tikrinimo sąlygas. Joms aprašyti naudojama įprastinė C# programavimo kalba, su visomis jai prieinamomis funkcijomis, klasėmis ir savybėmis. Tai ženkliai pagerina ir supaprastina modelio apribojimų realizavimą, kuomet mums nereikia kurtis savo metodų ar rašyti daug teksto, kad realizuotume standartines, plačiai naudojamas funkcijas (pavyzdžiui: „string.Contains“, „string.IndexOf“ ir panašiai). Kuomet aprašomos negalimos modelio būsenos, reikalinga išvesti pranešimus apie šias situacijas. Tam naudojamos funkcijos:

- LogError – klaidos išvedimas;
- LogMessage – žinutės išvedimas;
- LogWarning – perspėjimo išvedimas.

Tai standartiniai „Visual Studio 2008“ paketo informacijos pateikimo būdai. Jos susideda iš pranešimo aprašymo, klaidos kodo ir objekto (kas leidžia du kartus spustelėjus ant klaidos patekti į klaidą iššaukusią vietą). Paskutinis parametras svarbus, kuomet panašių elementų ar vietų, galinčių iššaukti pranešimą yra daug, ir mes greitai galime atrasti nekorektišką vietą.

3.1.7. Sričiai orientuotos kalbos testavimas ir realizavimas

Sukūrus sričiai orientuotą kalbą, kaip ir su kiekvienu nauju produktu, reikalinga atlikti testavimą. Kad būtu pilnai įsitikinta DSL korektiškumu, reikalinga ištestuoti šias dalis:

- validavimo apribojimus;
- generavimo šablonus;

- sugeneruotą kodą;

Naudojant „Visual Studio 2008“ programinį paketą su „DSL Tools“ įskiepiu sričiai orientuotos kalbos kūrimui, testavimas atliekamas eksperimentinėje aplinkoje (sukompiliavus ir pradėjus vykdyti (angl. run) DSL projektą, paleidžiama pavyzdinė „Visual Studio 2008” programa), kurioje galima atlikti sistemos modeliavimą ir validavimą, kurti transformacijos šablonus ir kitus su sistemos kūrimu, panaudojant DSL, susijusius veiksmus.

Validavimo apribojimų paskirtis – informuoti (klaidomis ar įspėjimais) apie negalimas modelio objektų kombinacijas, savybes ar ryšius. Jei modelis praeina visas validavimo funkcijas, tuomet jam parašyti bet kokie generatoriai turėtu korektiškai atlikti visus generavimus.

Pats paprasčiausias būdas atlikti šių elementų testavimą – tai kiekvienai validavimo situacijai rankiniu būdu sukurti atskirą modelį (arba bendrą, apimantį kelias situacijas) ir patikrinti jų veikimą.

Kitas būdas yra automatizuotas testavimas, panaudojant *unit* testavimą (angl. unit test), kuomet modeliai yra aprašomi tekstine kalba ir validavimas atliekamas kreipiantis į validavimo metodus per programinę sąsają (API). Šie testai rezultatus gali išvesti į tekstinius failus ir juos palyginti su reikiamais rezultatais.

Generavimo šablonų testavimas gali būti atliekamas tais pačiais būdais, kaip ir validavimo apribojimų testavimas: rankiniu arba automatizuotais testais.

Pirmuoju būdu atliekant testavimą reikalinga sukurti keletą modelių rankomis, „Visual Studio 2008“ pagalba paleisti šablonus ir patikrinti gautus rezultatus: ar jie atitinka norėtus ir turėtus gauti rezultatus. Jei matomi neatitikimai, pirmiausia reikia įsitikinti modelių teisingumu (koreguoti modelių validavimo apribojimus) ir tik po to taisyti šablonus, ko pasekoje modeliai tampa teisingi, o ne šablonai pritaikomi išimtims.

Atliekant automatizuotus generavimo šablonų testavimus, kurie galimi pasinaudojant „Visual Studio 2008“ programavimo aplinka arba komandinės eilutės sąsaja tekstinių šablonų varikliui. Testai turėtų rezultatus patikrinti su reikiamais rezultatais. Atliekant testus tokiu būdu nereiktų pamiršti atkreipti dėmesio į vietas, kuriose naudojami elementai, galintys įgauti skirtingas reikšmes kiekvieno paleidimo metu (pavyzdžiui: dabartinė data, atsitiktiniai skaičiai), ir jas išvalyti. Automatinius testus naudoti reiktų tik teisingiems modeliams. Rekomenduojama šiuos testus atlikti su modeliais, kurie buvo aprašyti validavimo apribojimų testavimui. Jei pastebima, kad jų naudojimas negražina laukiamų rezultatų – pirmiausia reiktų patikrinti modelių validavimą, nes tai gali reikšti, jog

buvo praleistas svarbus validavimo apribojimas. Rekomenduojama sukurti papildomą validavimo apribojimą, o ne koreguoti šabloną ir pritaikyti jį rastai išimčiai.

Kadangi testavimas atliekamas eksperimentinėje aplinkoje, tai mums leidžia testuoti ir sugeneruotą kodą DSL kūrimo metu.

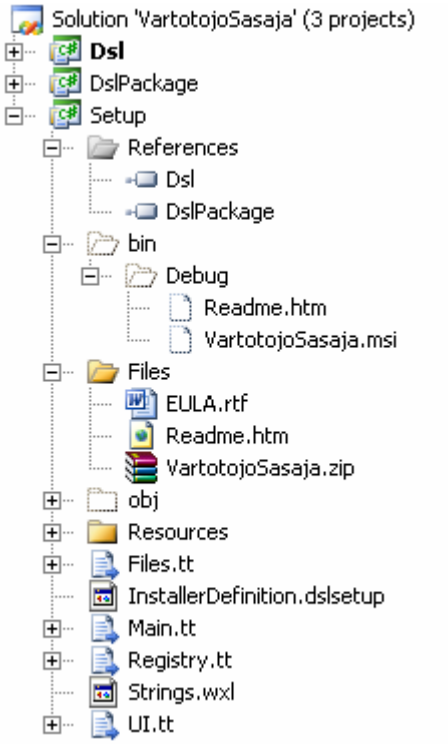
Kadangi kodas generuojamas naudojant transformacijos šablonus, tai jo testavimas papildomai atliks ir šablonų testavimą. Jei generuojamas kodas nepraeis testų – pirmiausia klaidų reiktų ieškoti transformacijos šablonuose.

Jei generuotas kodas yra „Visual Studio 2008" programinio paketo palaikoma programavimo kalba, tuomet jo sintaksė ir struktūra yra ištestuojama pačio programinio paketo, kuris standartinėmis savo priemonėmis informuoja mus apie kode esančias klaidas ar neatitikimus, negalimas situacijas. Taip pat jei projektas privestas iki stadijos, kuomet galima jį kompiliuoti ir bandyti įvykdyti, tai taip pat gali būti patikrinta programinio paketo pagalba: atlikus kompiliavimą bei įvykdymą. Šių veiksmų neįvykdymas mus informuotų apie sugeneruotame kode esančias klaidas.

Toliau sugeneruoto kodo testavimas gali būti atliekamas aukščiau aprašytais metodais.

Pilnai atlikus sričiai orientuotos kalbos testavimą, jos panaudojimo metu sugeneruotai artefaktai nereikalauja papildomo testavimo. Tai žinoma galima atlikti, bet tam nėra pagrindo, nes visos įmanomos situacijos DSL kūrimo metu jau yra ištestuotos, arba apribojimų ir validavimų pagalba, jos yra surandamos ir vartotojas apie jas yra informuojamas. Tokiu būdu, visus generuojamus artefaktus galime laikyti visiškai ištestuotais.

Kuomet sričiai orientuota kalba yra ištestuota, galima ją paruošti naudojimui. „Visual Studio 2008" paketas leidžia sukurti sričiai orientuotos kalbos sprendime (angl. solution) naują projektą (sričiai orientuotos kalbos sąranka (angl. setup), 18 paveikslas), kuris sukuria DSL diegimo failą.



18 pav. DSL sąrankos projekto sudėtis

Projekte standartiškai sukuriamos reikiamos nuorodos į kuriamą kalbą, sukuriamas licenzijos bei „Perskaityk mane“ (angl. read me) failai, kuriuos galima redaguoti ir juose pateikti reikiamą informaciją.

Jei norima, kad sričiai orientuota kalba turėtų ir mūsų aprašytus šablonus, testinį modelį bei kitus kurtus elementus, jai reikia priskirti projekto šabloną, kuris gaunamas „Visual Studio 2008“ pagalba iš meniu punkto pasirinkus „File -> Export Template“. Tokiu būdu sričiai orientuota kalbą bus galima naudoti kaip paprastą projektą ir ji savyje jau turės veikiantį modelį, šablonus ir kitus sukurtus reikalingus darbui elementus.

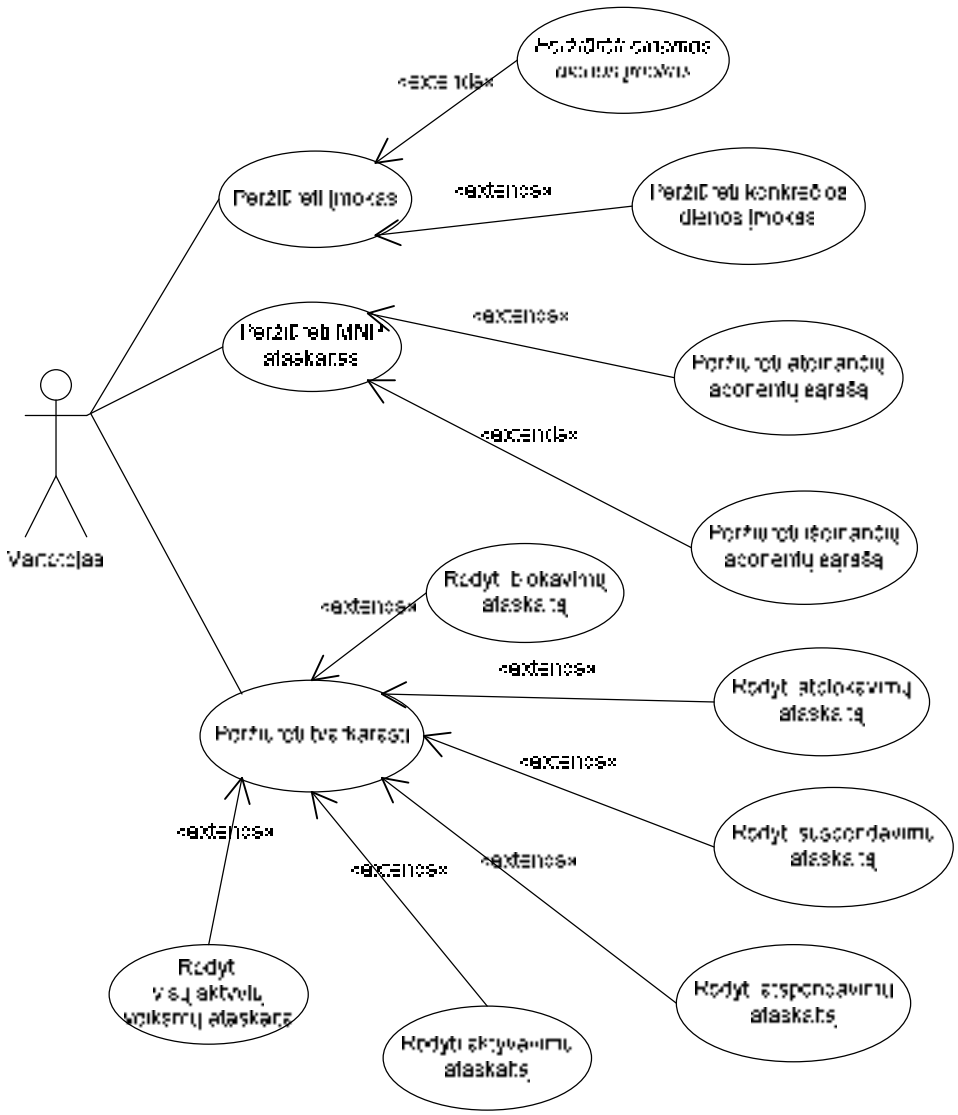
Kuomet visa tai atlikta, DSL perkėlimas į kitus kompiuterius yra nesudėtingas ir užtrunkantis nedaug laiko. Reikalinga tik turėti „Visual Studio 2008“ paketą.

Instaliacinis failas perkeliamas į reikiamą kompiuterį ir paleidžiamas. Paleidus instaliacinį failą, pasileidžia sričiai orientuotos kalbos diegimo vedlys. Instaliavus DSL į naująjį kompiuterį ir paleidus „Visual Studio 2008“, galime pasirinkti naują projektą, kuris bus kuriamas naudojant naujajai sričiai orientuotą kalbą.

3.2. Sistemos kūrimas naudojant sukurtą sričiai orientuotą klabą

3.2.1. Sistemos panaudojimo atvejai

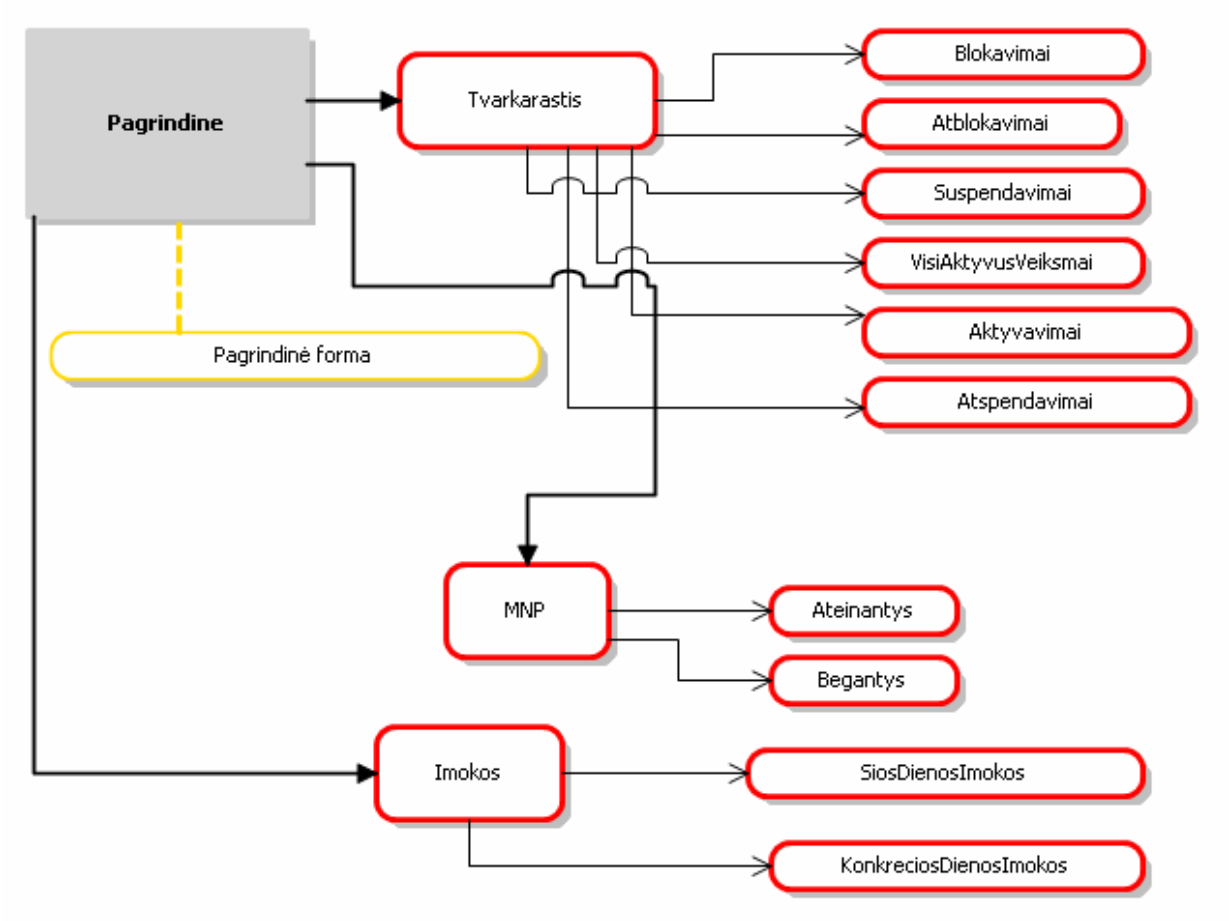
Pagrindinis sistemos vartotojas – vadybininkas. Sistemos pagalba jis gali operatyviai gauti reikiamą informaciją duotuoju momentu, bei realiu laiku stebėti informacijos kitimą ar vykstančius procesus. Sistemos panaudos atvejų diagrama pateikiama 19 paveiksle.



19 pav. Sistemos panaudos atvejų diagrama

3.2.2. Sistemos modelis

Turint sistemos panaudojimo atvejų diagramą, galima sukurti sistemos modelį, kurio pagalba ir bus realizuojama sistema. Šis modelis sukuriamas panaudojant ankščiau sukurtos sričiai orientuotos kalbos dizainerio pagalba. Sistemos modelis pavaizduotas 20 paveiksle. Kiekvienam modelio elementui priskiriamos reikiamos reikšmės, kurios nurodo reikiamą sistemos veikimą.



20 pav. Sistemos modelis

Turint Sistemos modelį, kuris tenkina visas reikiamas sąlygas (modelio validavimas gražina teigiamus rezultatus), atlikus automatinį generavimą gauname veikiančią sistemą, kuri atlieka mūsų apsibrėžtas funkcijas.

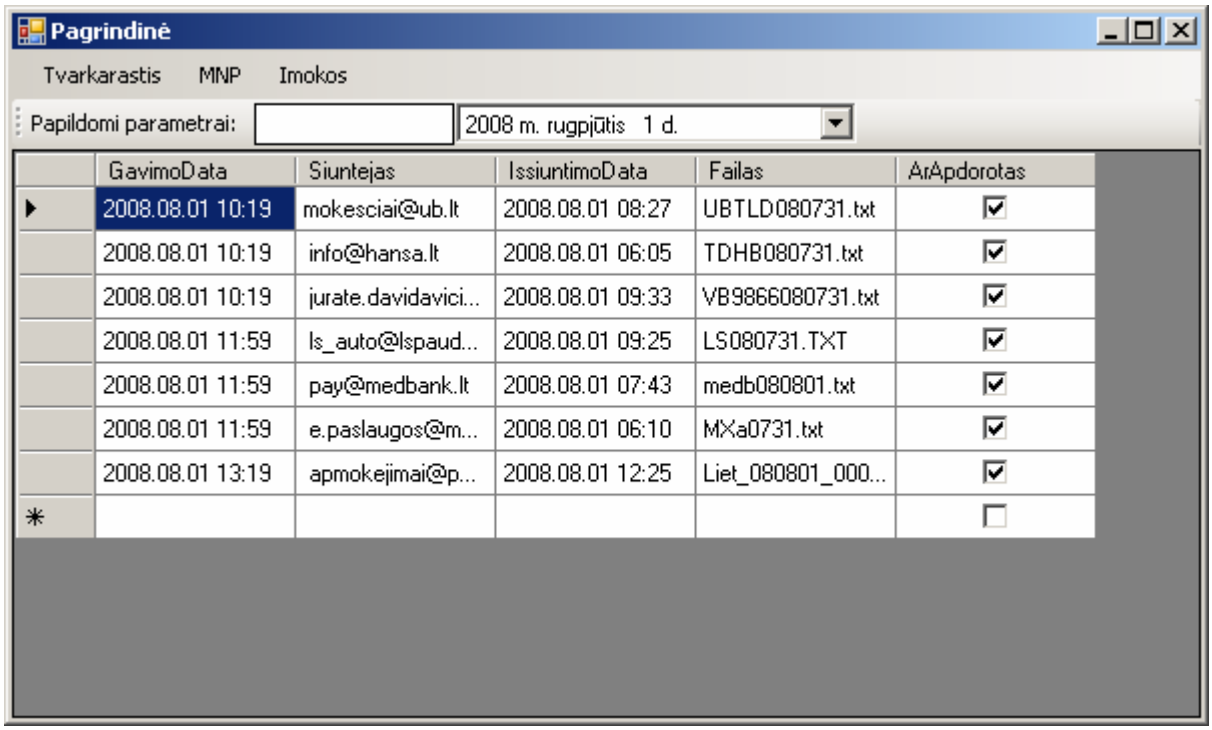
3.2.3. Sistemos generavimo rezultatai

Atlikus generavimą, sugeneruota:

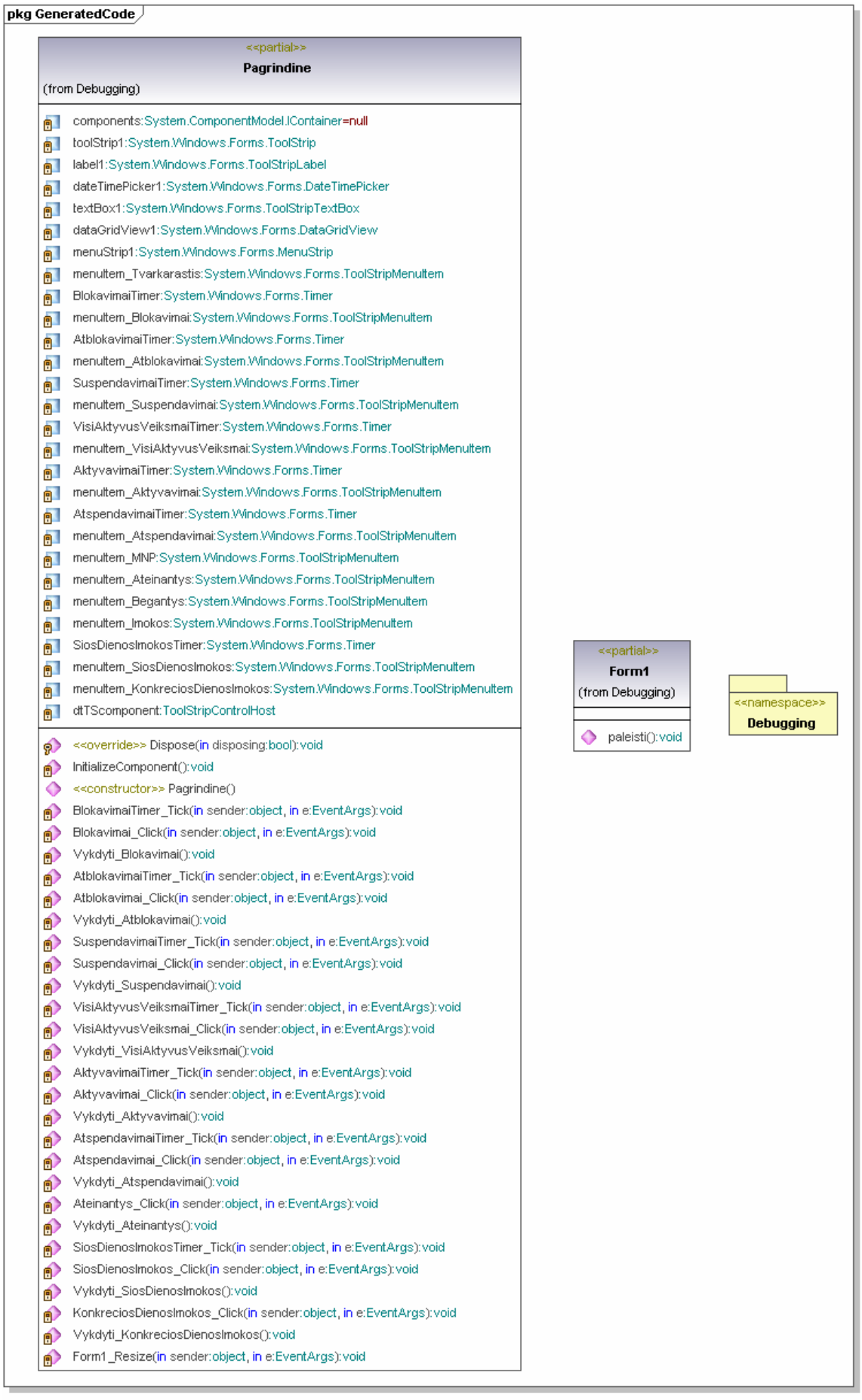
- 575 kodo eilutės;

- 3 C# programavimo kalbos išeities failai (jie priklauso nuo transformavimo šablonų kiekio);
- 2 klasės;
- 30 metodų.

Gautas sistemos vartotojo sąsajos vaizdas pavaizduotas 21 paveiksle, o sugeneruoto kodo gauta klasių diagrama pavaizduota 22 paveiksle.



21 pav. Sukurtos sistemos vartotojo sąsajos vaizdas



22 pav. Sugeneruoto sistemos kodo klasių diagrama

4. Eksperimentinis tyrimas

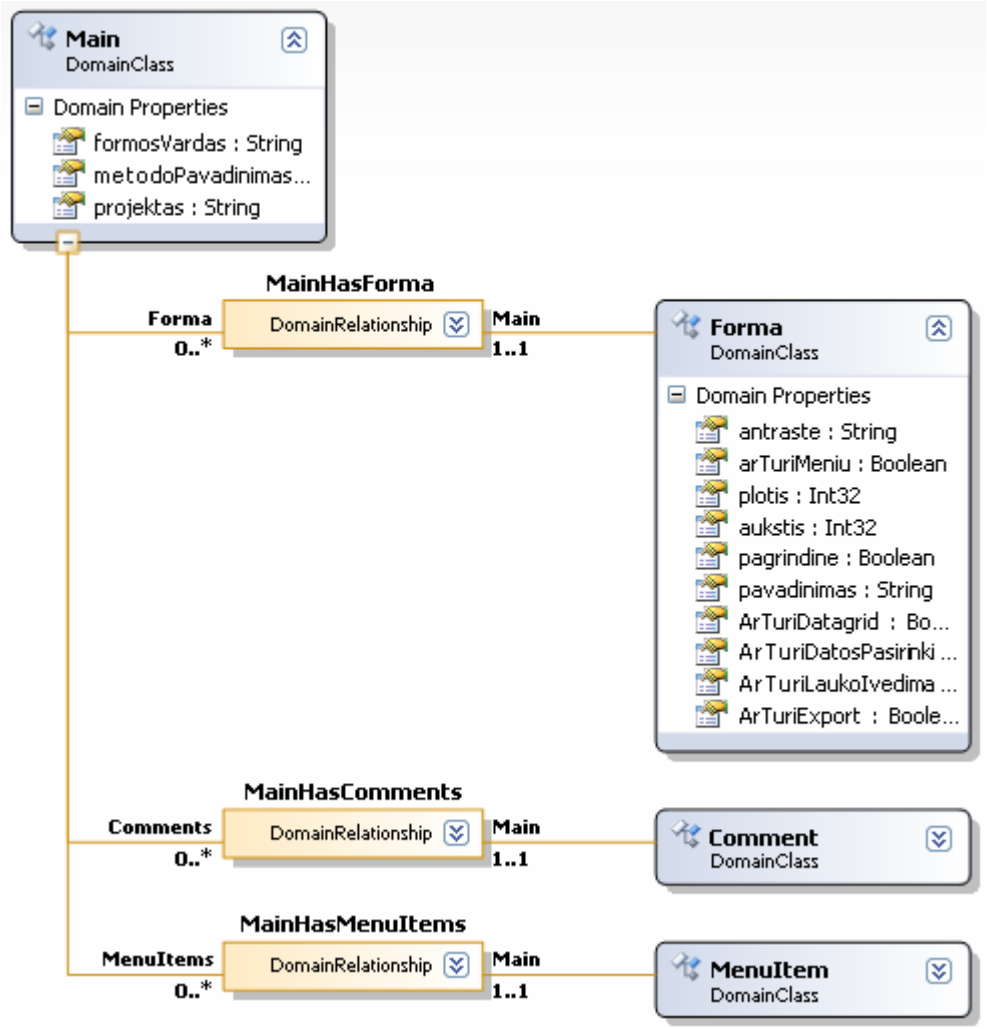
4.1. Sričiai orientuotos kalbos modelio modifikavimas

4.1.1. Atlikti pakeitimai

Pirmas variantas

Atlikti srities modelio pakeitimai:

- srities klasei „Main“ pridėta savybę „projektas“, kuri yra tekstinio tipo ir nurodo generuojamo programinio kodo vardų sritį (angl. namespace);
- srities klasei „Forma“ pridėti savybę „ArTuriExport“, kuri yra loginio tipo ir nurodo, ar sugeneruota forma turi mygtuką, kurio pagalba galima eksportuoti joje vaizduojamus duomenis į tekstinį failą.

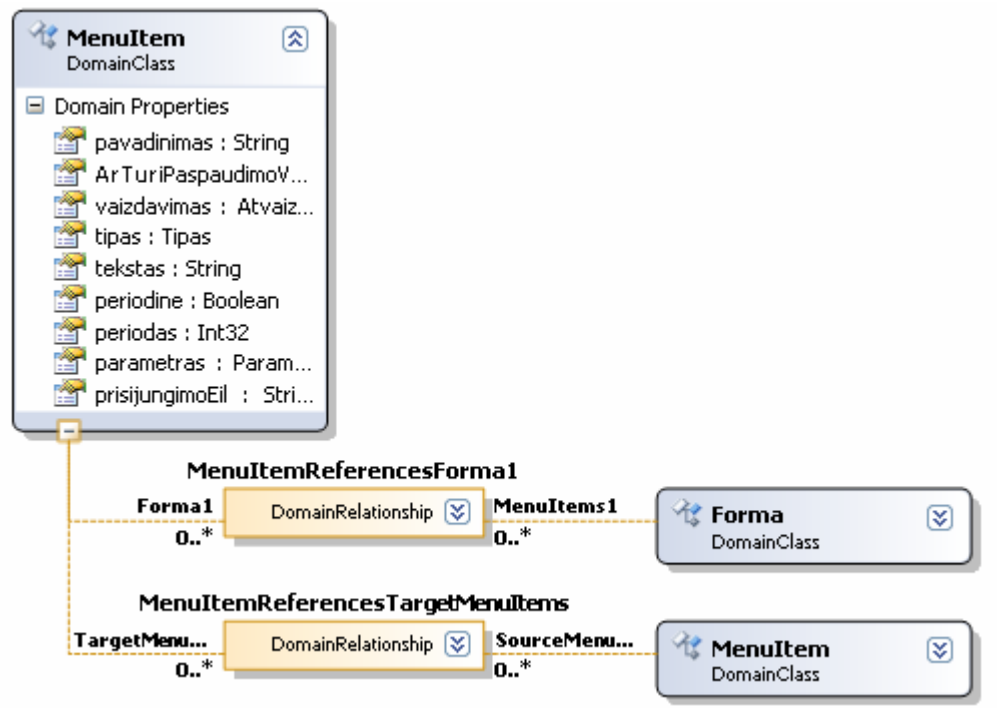


23 pav. pirmas modifikuotas srities modelio fragmentas

Antras variantas

Atlikti srities modelio pakeitimai:

- srities klasei „MenuItem“ pridėta savybė „prisijungimoEil“, kuri saugo prisijungimo prie duomenų bazės duomenis.

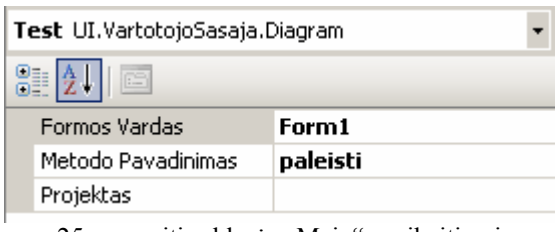


24 pav. antras modifikuotas srities modelio fragmentas

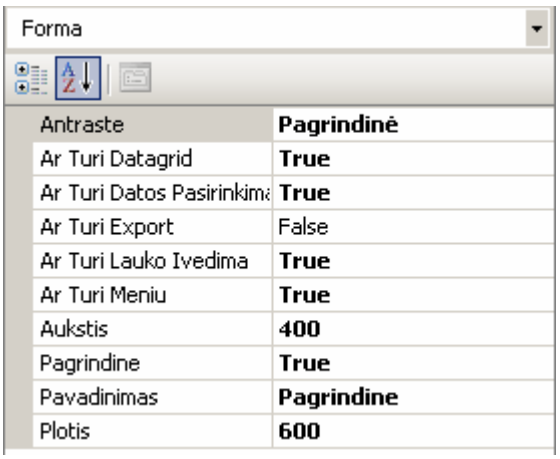
4.1.2. Gauti rezultatai

Atlikus projekto transformavimą ir paleidus eksperimentinę Visual Studio 2008 aplinką, DSL kūrimo aplinkoje (sistemos modelio kūrimo lange), pasirinkę atitinkamas srities klases, matome atliktų pakeitimų rezultatus (grafiškai pavaizduoti paveiksluose):

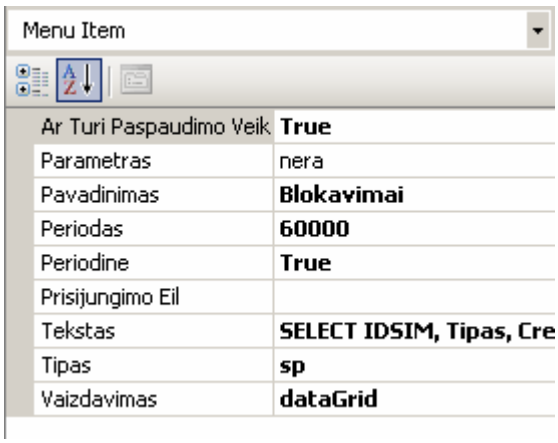
- pasirinkus diagramos elementą (paspaudę ant diagramos balto ploto), matome naujai sukurtą savybę „Projektas“, kuriai galime priskirti norimą reikšmę (pagal pirmą variantą);
- pasirinkus diagramos elementą „Forma“, jai galime parinkti savybės „ArTuriExport“ (pagal nutylėjimą jai priskiriama reikšmė „false“) reikšmes iš galimo rėžio (šiuo atveju „true“ arba „false“) (pagal pirmą variantą);
- pasirinkus diagramos elementą „MenuItem“, jai galime priskirti savybės „PrisijungimoEil“ reikšmę, nurodančia prisijungimo prie duomenų bazės duomenis (pagal antrą variantą).



25 pav. srities klasės „Main“ pasikeitimai



26 pav. Srities klasės „Forma“ pasikeitimai



27 pav. Srities klasės „MenuItem“ pasikeitimai

Reikia pastebėti, kad srities modelio modifikacijos, kuomet pridedamos papildomos srities klasių savybės ar sukuriamos naujos srities klasės, visiškai nepažeidžia turimos DSL: atsiradęs naujas funkcionalumas tiesiog nebūna realizuotas (savybėms priskyrus norimas reikšmes jos yra ignoruojamos).

Kuomet modelis modifikuojamas pašalinant tam tikras savybes, jos automatiškai pašalinamos iš sistemos modelio, tačiau transformavimo šablonuose lieka, ko pasekoje DSL generuojami artefaktai yra nekorektiški ir būtina juos modifikuoti.

4.2. Transformavimo šablonų modifikavimas

4.2.1. Pirmas variantas

Pagal 4.1. skyriuje aprašytą pirmąjį srities modelio modifikavimą, reikalinga modifikuoti ir transformavimo šablonus, kad jie atitiktų pakeitimus ir galėtume gauti norimą naują DSL funkcionalumą.

Pirmajam pakeitimui, visuose transformacijos šablonuose reikalinga įrašyti tekstą:

```
namespace <#= Main.projektas#>
```

Priskyrus šiai savybei reikiamą reikšmę, visos klasės turi vieningą vardo sritį, kuri gali kisti, kuriant skirtingus projektus. Pakeitimas leidžia išvengti klaidų ir kodo taisymo, kuomet naujo projekto pavadinimas nesutampa su standartiniu.

Rezultatas atrodo taip, kuomet „projekto“ savybei priskiriama reikšmė „DSL“:

```
namespace DSL
```

Šiuo atveju viena transformacijos šablono eilutė atitinka vieną generuojamo kodo eilutę.

Antrajam pakeitimui realizuoti reikalinga parašyti 73 transformacijos šablonų eilutes, iš kurių 35 yra C# kalbos kodo eilutės, o 38 transformacijos šablonų loginės eilutės. Jos aprašo srities klasės „Forma“ funkcionalumą. Atlikus generavimą pagal pakeitimą, sugeneruota 35 programinio kodo eilutės (jų skaičius gali kisti priklausomai nuo sistemos modelio) vienai klasei, kuomet savybių „ArTuriDatosPasirinkima“, „ArTuriExport “ ir „ArTuriLaukoIvedima“ reikšmės yra „true“. Jei į sistemos modelį įtraukiame dar viena „Forma“ tipo elementą (su tais pačiais parametrais), gauname 70 automatiškai sugeneruoto kodo eilučių.

Galima pastebėti, kad sugeneruotų kodo eilučių skaičius tiesiogiai priklauso nuo pakeitime aprašomų elementų skaičiaus. Tai galima užrašyti formule:

$$x \approx y \times z \text{ , kur}$$

x – sugeneruojamų programinio kodo eilučių skaičius;

y – sistemos modeliu aprašomų elementų skaičius;

z – transformacijos šablone elementą aprašančių programinio kodo eilučių skaičius.

4.2.2. Antras variantas

Sukurtoje sričiai orientuotoje kalboje pagrindinis duomenų šaltinis yra duomenų bazės. Tačiau atlikta realizacija leidžia jungtis tik prie vienos duomenų bazės, naudojant standartinę prisijungimo eilutę (angl. connection string).

Reikalinga DSL pakeisti taip, kad prisijungimo eilutę būtų galima įvesti pačiam vartotojui.

Tai galima atlikti dviem būdais:

- rankomis keisti sugeneruotą kodą arba
- modifikuoti srities modelį ir transformacijos šablonus.

Pirmasis variantas yra nepatogus tuo, kad mums rankiniu būru reikia eiti per visą kodą, ir jame keisti visas prisijungimo eilutes pagal poreikį. Taip pat yra tikimybė, kad neteisingai ar netinkamoje vietoje atliksime pakeitimus.

Antrajam variantui atlikti, reikalinga atlikti srities modelio pakeitimą, kuris aprašytas 4.1.1. skyriaus antrame variante. Tai atlikus, reikia modifikuoti ir transformavimo šabloną (klases.tt):

```
<#
              if (menu1.prisijungimoEil != "")
              {
#>
string pri = @"<#= menu1.prisijungimoEil#>";
<#
              }
              else
              {
#>
string pri = @"data source=JUSTO-NB\SQLEXPRESS;Database=TeledemaDB;Uid=justas;Pwd=justas1";
<#
              }
#>
```

Šis kodas patikrina, ar yra įrašyta prisijungimo eilutė. Jei taip – ji naudojama jungtis prie duomenų bazės. Jei ne – naudojama standartinė prisijungimo eilutė.

Toks tekstas įrašomas trijose transformavimo šablono vietose, ko pasakoje visas anksčiau sugeneruotas kodas yra modifikuojamas ir atnaujinamas (pagal 3.3.2. skyriaus 18 paveiksle pateiktą sistemos modelį, sugeneruojama (pakeičiamos) 10 kodo eilučių, iš viso tame faile yra 341 kodo eilutė). Mes išvengiame būtinybės tikrinti visą kodą ir ieškoti jame vietų, kur naudojamas prisijungimas prie duomenų bazės. Taip sutaupoma laiko ir išvengiama klaidų. Taip pat, atlikus šį pakeitimą, mes bet kada galime keisti prisijungimo prie duomenų bazės duomenis ir tai nereikalaus rankinio kodo redagavimo.

5. Išvados

1. Atlikus sričiai orientuotos metodikos analizę, pagrįstas sprendimas sistemos kūrimui naudoti sričiai orientuotą modeliavimo metodiką.
2. Sukurta veikianti sistema, panaudojant sričiai orientuotą modeliavimo metodiką.
3. Sukūrus sričiai orientuotą kalbą, DSL vartotojas gali sukurti veikiančią informacinę sistemą su minimaliomis programavimo žiniomis.
4. DSL naudojimas informacinės sistemos kūrime leidžia ženkliai padidinti pradinį projekto brandumo lygį (angl. project maturity level).
5. Atlikus eksperimentinį tyrimą, įsitikinta sričiai orientuotos modeliavimo metodikos naudojimo privalumais, kuomet reikalinga plėsti sukurtos sistemos funkcionalumą arba taisant pastebėtas klaidas.
6. Eksperimentinio tyrimo metu pastebėta, kad kad galima įžvelgti priklausomybę tarp transformacijos šablonuose aprašyto programinio kodo ir sugeneruoto programinio kodo, kurią galima aprašyti formule:

$$x \approx y \times z \text{ , kur}$$

x – sugeneruojamų programinio kodo eilučių skaičius;

y – sistemos modeliu aprašomų elementų skaičius;

z – transformacijos šablone elementą aprašančių programinio kodo eilučių skaičius.

6. Santrumpų ir terminų žodynas

DSL (angl. domain specific language) - srities (probleminės srities) specifikavimo kalba. Tai programavimo kalba, sukurta konkrečiai užduočiai spręsti. Tai tarsi priešingybė bendrosios paskirties programavimo kalboms, tokioms kaip C, arba bendrosios paskirties modeliavimo kalboms kaip UML. DSL pavyzdžiai yra visiems gerai žinomos ir naudojamos kalbos, tokios kaip HTML ar SQL, kurios turi konkrečiai apibrėžtą taikymo sritį.

DSM (angl. domain specific modeling) - DSL'ų kūrimo ir naudojimo būdus, dažniausiai vaizdinių DSL'ų su sričiai orientuotais generatoriais (angl. generator), kurie sugeneruoja pilnai veikiantį kodą tiesiai iš sistemos modelio.

DSD (angl. domain specific development) – tas pats kaip ir DSM.

Artefaktai – tai informacija, kuri gali būti sukurta, keičiama ar naudojama proceso eigoje. Artefaktai gali būti įeities ar išeities duomenys darbuotojui vykdant veiklą.

MDA (angl. modelį driven architecture)- tai metodika, kurioje siekiama atskirti sistemos specifikavimo lygį nuo jos įgyvendinimo konkrečioje technologinėje platformoje, ko pasekoje gaunama architektūra, kuri yra nepriklausoma nei nuo konkrečios kalbos, nei nuo platformos ar gamintojo

UML (angl. unified modeling language) – tai specifikavimo kalba, naudojama programinės įrangos projektavimo srityje. Ji gali būti apibrėžta kaip bendrojo naudojimo kalba, kur grafinių žymėjimų pagalba galima sukurti abstrakčius modelius.

HTML (angl. hyper text markup language) – tai kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete. Kalbą standartizuoja W3 konsorciumas.

SQL (angl. structured query language) – populiariausia iš šiuo metu naudojamų kalbų, skirtų aprašyti duomenis ir manipuliuoti jais reliacinių duomenų bazių valdymo sistemose.

GUI (angl. graphical user interface) – grafinė vartotojo sąsaja, skirta sistemų bendravimui su vartotojais.

API (angl. application programming interface) – tai sąsaja, kurią suteikia kompiuterinė sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis.

.NET projektavimo aplinka – tai „Microsoft Windows“ operacinės sistemos komponentas, sukurtas 2002 metais. Jis suteikia kitoms programoms galimybę naudotis daugybe jau paruoštu įvairių bibliotekų (pvz., duomenų bazių komponentus, formų komponentus ir t.t.). Be to, šis komponentas tvarko programos kodą jos vykdymo metu, jei programa parašyta specialiai šiai aplinkai.

Java 2 Platform, Enterprise Edition (J2EE) – standartinė daugialyčių programų kūrimo Java programavimo kalba platforma, paremta moduliniais komponentais, vykdomais programų serveryje.

Asembleris (angl. assembler) – programa, transliuojanti asemblerio kalba rašytą programos tekstą į objektinį (mašininį) kodą.

Modelis - įvairiose disciplinose, tai realaus reiškinio, proceso, struktūros, sistemos abstrakcija. Modelyje paliekamos tik tos savybes, kurios svarbios sprendžiamai problemai. Modelis gali būti tiek fizinis, tiek virtualus. Modeliui kurti gali būti naudojami modeliavimo įrankiai.

Metamodelis – tai modeliavimo kalbos modelis, kuriame apibrėžtos esminės kalbos savybės. Metamodelis turi būti tikslus išreikštinių darinių (angl. artifacts) ir taisyklių apibrėžimas, kuris reikalingas semantinių arba dalykinės srities modelių kūrimui.

Modeliavimas – tai abstrakčių ar konceptualių modelių kūrimas.

Programavimo kalba – dirbtinė kalba, skirta programoms užrašyti.

Validacija (validavimas) - tai objekto patvirtinimas objektyviais įrodymais, kad specifinio ketinamo naudojimo ar taikymo reikalavimai yra įvykdyti.

7. Naudota literatūra

[1] Cook, S., et al. Domain-Specific Development with Visual Studio DSL Tools. 2007. ISBN 13: 978-0-321-39820-8, ISBN 10: 0-321-39820-3.

[2] Dalgarno, M.; ir Fowler, M. UML vs. Domain – Specific Languages [interaktyvus]. 2008 m. vasara. ISSN 1661-402X. [žiūrėta 2008-11-03]. Prieiga per internetą: <http://www.martinig.ch/PDF/mt200802.pdf>.

[3] Hammond, J. L. Boosting productivity with domain – specific modeling [interaktyvus]. 2008 m. sausis. [žiūrėta 2008-11-03]. Prieiga per internetą: <http://www.metacase.com/papers/mte-boosting.pdf>.

[4] Hammond, J. L. Domain – specific modeling significantly reduces development time [interaktyvus]. 2008 m. balandis. [žiūrėta 2008-11-03]. Prieiga per internetą: <http://www.metacase.com/papers/ece_april2008.pdf>.

[5] Herrington, J. Code Generation in Action. 2003. ISBN 1-930110-97-9.

[6] Iseger, M. OMG: "MDA and DSL are the same!" [interaktyvus]. 2006 m. lapkritis. [žiūrėta 2008-10-04]. Prieiga per internetą: <http://softwarechimps.blogspot.com/2006/11/omg-mda-and-dsl-are-same.html>.

[7] Kelly, S. Improving Developer Productivity With Domain-Specific Modeling Languages [interaktyvus]. 2005 m. liepa. [žiūrėta 2008-11-03]. Prieiga per internetą: <http://www.developerdotstar.com/mag/articles/domain_modeling_language.html>.

[8] Kelly, S. Moving from Writing Code to Generating It [interaktyvus]. 2007. [žiūrėta 2008-10-15]. Prieiga per internetą: <http://www.iasahome.org/c/portal/layout?p_l_id=PUB.1.329>.

[9] MetaCase Increase productivity by a factor of 10! [interaktyvus]. [žiūrėta 2008-11-03]. Prieiga per internetą: <http://www.metacase.com/increase.html>.

[10] Prieiga per internetą: <http://www.metacase.com/increase.html>. [žiūrėta 2008-11-03].

[11] Prieiga per internetą: <http://www.omg.org/mda/>. [žiūrėta 2008-11-03].

[12] Prieiga per internetą: <http://www.uml.org/>. [žiūrėta 2008-11-03].

[13] Tolvanen, J. P. Domain-specific Modeling: Welcome to the Next Generation of Software Modeling [interaktyvus]. 2005 m. spalio 28 d.. [žiūrėta 2008-10-18]. Prieiga per internetą: <http://www.devx.com/enterprise/Article/29619/0/page/1>.

[14] Wills, C. A.; ir Kelly, S. Agile Development with Domain Specific Languages. Scaling up Agile - is Domain-Specific Modeling the key? [žiūrėta 2008-10-14]. Prieiga per internetą: <http://www.dsmforum.org/events/ADDSL05/>.

Santrauka anglų kalba

Research of DSL for information system development

Summary

Domain Specific Modeling is a software engineering methodology for designing and developing information systems. It involves systematic use of a graphical domain specific language (DSL) to represent the various facets of a system. DSM languages tend to support higher-level abstractions than general purpose modeling languages, so they require less effort and fewer low-level details to specify a given system which is very important nowadays, when the functionality of information system must be maximized with minimum development time and cost.

This paper compares usage of DSL for information system development with other two, most often used methodologies for this purpose: MDA and usage of UML. It gives basic advantages and disadvantages of DSL and DSM usage, describes differences of DSL (DSM), UML and MDA.

This paper also describes usage of DSM methodology (creating DSL and using it for software development) for creating information system. It covers main steps of creation process's: describing the domain concepts, describing the artifacts that are planning for the DSL, building domain model, building the designer for DSL, building the artifact generator, implementing validations and constraints, testing and deploying the DSL.

Priedai

Transformacijos šablonas „designer.tt“

```
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransfor
mation" #>
<#@ output extension=".cs" #>
<#@ VartotojoSasaja processor="VartotojoSasajaDirectiveProcessor"
requires="fileName='../Test.UIdsl'" #>
<#@ import namespace="System.Collections" #>

namespace Debugging
{
<#
           foreach (Forma forma in this.Main.Forma)
           {
#>
    partial class <#= forma.pavadinimas#>
    {

        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        private void InitializeComponent()
        {
                                #region Windows Form komponentai
                                //susikuriame visus objektus
                                this.components = new
System.ComponentModel.Container();
<#
                        if (forma.ArTuriDatagrid)
        {
#>
                                this.dataGridView1 = new
System.Windows.Forms.DataGridView();
<#
                        }
                        if ((forma.ArTuriDatosPasirinkima) ||
(forma.ArTuriLaukoIvedima))
        {
#>
                                this.toolStrip1 = new
System.Windows.Forms.ToolStrip();
                                this.label1 = new
System.Windows.Forms.ToolStripLabel();
<#
                                if (forma.ArTuriLaukoIvedima)
                                {
#>
                                this.textBox1 = new
System.Windows.Forms.ToolStripTextBox();
<#
                                }
```

```
#>
            this.toolStrip1.SuspendLayout();
<#
                        }
                        if (forma.ArTuriDatosPasirinkima)
                        {
#>
            this.dateTimePicker1 = new
System.Windows.Forms.DateTimePicker();
<#
                        }
                        if (forma.arTuriMeniu)
                        {
#>
                                    //meniu maksimaliai gali tureti 3 lygius
                                    this.menuStrip1 = new
System.Windows.Forms.MenuStrip();
<#
                                    foreach (MenuItem menu in forma.MenuItems)
                                    {
#>
<#
                                    if (menu.periodine)
                                    {
#>
                                    this.<#= menu.pavadinimas#>Timer = new
System.Windows.Forms.Timer(this.components);
<#
                                    }
#>
                                    this.menuItem_<#= menu.pavadinimas#> = new
System.Windows.Forms.ToolStripMenuItem();
<#
                                                foreach (MenuItem menu1 in
menu.TargetMenuItems)
                                                {
#>
<#
                                    if (menu1.periodine)
                                    {
#>
                                    this.<#= menu1.pavadinimas#>Timer = new
System.Windows.Forms.Timer(this.components);
<#
                                    }
#>
                                    this.menuItem_<#= menu1.pavadinimas#> = new
System.Windows.Forms.ToolStripMenuItem();
<#
                                                            foreach (MenuItem
menu2 in menu1.TargetMenuItems)
                                                            {
#>
<#
                                    if (menu2.periodine)
                                    {
#>
                                    this.<#= menu2.pavadinimas#>Timer = new
System.Windows.Forms.Timer(this.components);
<#
                                    }
#>
                                    this.menuItem_<#= menu2.pavadinimas#> = new
System.Windows.Forms.ToolStripMenuItem();
```

```
<#
                                                        }
                                                    }
                                    }
#>
                                    this.menuStrip1.SuspendLayout();
<#
                        }
#>
                                    this.SuspendLayout();
<#
                        if (forma.ArTuriDatagrid)
        {
#>

                                    //dataGridas

            ((System.ComponentModel.ISupportInitialize)(this.dataGridView1))
.BeginInit();

            this.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
             this.dataGridView1.Dock = System.Windows.Forms.DockStyle.Fill;
             this.dataGridView1.Name = "dataGridView1";
             this.dataGridView1.TabIndex = 1;
             this.Resize += new System.EventHandler(this.Form1_Resize);

<#
                        }

                        if ((forma.ArTuriDatosPasirinkima) ||
(forma.ArTuriLaukoIvedima))
                        {
#>
             //Jei pasirinktas dateTimePicker arba textBox
             this.toolStrip1.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
             this.label1
<#
                                    if (forma.ArTuriLaukoIvedima)
                                    {
#>
             ,this.textBox1<#
                                    }
#>});
             this.toolStrip1.Location = new System.Drawing.Point(0, 24);
             this.toolStrip1.Name = "toolStrip1";
             this.toolStrip1.Size = new System.Drawing.Size(600, 25);
             this.toolStrip1.Stretch = true;
             this.toolStrip1.Text = "toolStrip1";
             this.label1.Size = new System.Drawing.Size(113, 22);
             this.label1.Text = "Papildomi parametrai: ";
<#
                                    if (forma.ArTuriLaukoIvedima)
                                    {
#>
             this.textBox1.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle;
             this.textBox1.Size = new System.Drawing.Size(100, 25);
<#
                                    }
                        }
                        if (forma.ArTuriDatosPasirinkima)
                        {
```

```
#>
                                    this.dateTimePicker1.Location = new
System.Drawing.Point(36, 50);
            this.dateTimePicker1.Name = "dateTimePicker1";
            this.dateTimePicker1.Size = new System.Drawing.Size(200, 20);
<#
                        }
                        if (forma.arTuriMeniu)
                        {
                                    foreach (MenuItem menu in forma.MenuItems)
                                    {
#>

<#
                                    if (menu.periodine)
                                    {
#>
                                    this.<#= menu.pavadinimas#>Timer.Tick +=
new System.EventHandler(this.<#= menu.pavadinimas#>Timer_Tick);
<#
                                    }
                                                if (menu.Forma1.Count > 0 )
                                                {
#>
                                    this.menuItem_<#= menu.pavadinimas#>.Click
+= new System.EventHandler(this.<#= menu.pavadinimas#>_Click);
<#
                                                }
                                                foreach (MenuItem menu1 in
menu.TargetMenuItems)
                                                {
                                                            if (menu1.periodine)
                                                            {
#>
                                    this.<#= menu1.pavadinimas#>Timer.Tick +=
new System.EventHandler(this.<#= menu1.pavadinimas#>Timer_Tick);
<#
                                                            }
                                                            if
(menu1.Forma1.Count > 0 )
                                                            {
#>
                                    this.menuItem_<#= menu1.pavadinimas#>.Click
+= new System.EventHandler(this.<#= menu1.pavadinimas#>_Click);
<#
                                                            }
                                                            foreach (MenuItem
menu2 in menu1.TargetMenuItems)
                                                            {
                                                                        if
(menu2.periodine)
                                                                        {
#>
                                    this.<#= menu2.pavadinimas#>Timer.Tick +=
new System.EventHandler(this.<#= menu2.pavadinimas#>Timer_Tick);
<#
                                                                        }
                                                                        if
(menu2.Forma1.Count > 0 )
                                                                        {
#>
                                    this.menuItem_<#= menu2.pavadinimas#>.Click
+= new System.EventHandler(this.<#= menu2.pavadinimas#>_Click);
<#
```

```
                                                                    }
                                                                }
                                                            }
                                    }
#>
                                    this.menuStrip1.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
<#
                                    int c = 0;
                                    foreach (MenuItem menu in forma.MenuItems)
                                    {
#>
                                    <# if (c != 0) {#>,<#}#>this.menuItem_<#=
menu.pavadinimas#>
<#
                                            c++;
                                    }
#>
                                    });
            this.menuStrip1.Location = new System.Drawing.Point(0, 0);
            this.menuStrip1.Name = "menuStrip1";
            this.menuStrip1.Size = new System.Drawing.Size(292, 24);
            this.menuStrip1.Text = "menuStrip1";

<#
                                    foreach (MenuItem menu in forma.MenuItems)
                                    {
                                            foreach (MenuItem menu1 in
menu.TargetMenuItems)
                                            {
#>
                                    this.menuItem_<#=
menu.pavadinimas#>.DropDownItems.Add(this.menuItem_<#=
menu1.pavadinimas#>);
                                    this.menuItem_<#= menu1.pavadinimas#>.Name
= "menuItem_<#= menu1.pavadinimas#>";
            this.menuItem_<#= menu1.pavadinimas#>.Size = new
System.Drawing.Size(152, 22);
            this.menuItem_<#= menu1.pavadinimas#>.Text = "<#=
menu1.pavadinimas#>";
<#
                                    if (menu1.ArTuriPaspaudimoVeiksma)
                                    {
#>
            this.menuItem_<#= menu1.pavadinimas#>.Click += new
System.EventHandler(this.<#= menu1.pavadinimas#>_Click);
<#
                                    }
                                                    foreach (MenuItem
menu2 in menu1.TargetMenuItems)
                                                    {
#>
                                    this.menuItem_<#=
menu1.pavadinimas#>.DropDownItems.Add(this.menuItem_<#=
menu2.pavadinimas#>);
                                    this.menuItem_<#= menu2.pavadinimas#>.Name
= "menuItem_<#= menu2.pavadinimas#>";
            this.menuItem_<#= menu2.pavadinimas#>.Size = new
System.Drawing.Size(152, 22);
            this.menuItem_<#= menu2.pavadinimas#>.Text = "<#=
menu2.pavadinimas#>";
<#
                                    if (menu2.ArTuriPaspaudimoVeiksma)
                                    {
```

```
#>
            this.menuItem_<#= menu2.pavadinimas#>.Click += new
System.EventHandler(this.<#= menu2.pavadinimas#>_Click);
<#
                                    }
                                                        }
                                                }
#>
                                    this.menuItem_<#= menu.pavadinimas#>.Name =
"pirmasToolStripMenuItem";
            this.menuItem_<#= menu.pavadinimas#>.Size = new
System.Drawing.Size(50, 20);
            this.menuItem_<#= menu.pavadinimas#>.Text = "<#=
menu.pavadinimas#>";
<#
                                    if (menu.ArTuriPaspaudimoVeiksma)
                                    {
#>
            this.menuItem_<#= menu.pavadinimas#>.Click += new
System.EventHandler(this.<#= menu.pavadinimas#>_Click);
<#
                                    }
                                    }
                        }
#>
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(<#= forma.plotis#>,
<#= forma.aukstis#>);
<#
                        if (forma.ArTuriDatagrid)
        {
#>
                                    this.Controls.Add(this.dataGridView1);
<#
                        }
                        if (forma.ArTuriDatosPasirinkima)
                        {
#>
            this.Controls.Add(this.dateTimePicker1);
<#
                        }
                        if ((forma.ArTuriDatosPasirinkima) ||
(forma.ArTuriLaukoIvedima))
        {
#>
                                    this.Controls.Add(this.toolStrip1);
<#
                        }
                        if (forma.arTuriMeniu)
                        {
#>
                                    this.Controls.Add(this.menuStrip1);
            this.MainMenuStrip = this.menuStrip1;
<#
                        }
#>
            this.Name = "<#= forma.pavadinimas#>";
            this.Text = "<#= forma.antraste#>";
<#
                        if ((forma.ArTuriDatosPasirinkima) ||
(forma.ArTuriLaukoIvedima))
        {
#>
```

```
                                    this.toolStrip1.ResumeLayout(false);
            this.toolStrip1.PerformLayout();
<#
                        }
                        if (forma.arTuriMeniu)
                        {
#>
            this.menuStrip1.ResumeLayout(false);
            this.menuStrip1.PerformLayout();
<#
                        }
                        if (forma.ArTuriDatagrid)
                        {
#>

           ((System.ComponentModel.ISupportInitialize)(this.dataGridView1))
.EndInit();
<#
                        }
#>
            this.ResumeLayout(false);
            this.PerformLayout();
            #endregion
        }

<#
                        if (forma.ArTuriDatosPasirinkima)
                        {
#>
                        private System.Windows.Forms.ToolStrip toolStrip1;
                        private System.Windows.Forms.ToolStripLabel label1;
        private System.Windows.Forms.DateTimePicker dateTimePicker1;
<#
                        }
                        if (forma.ArTuriLaukoIvedima)
                        {
                                    if (!forma.ArTuriDatosPasirinkima)
                                    {
#>
                        private System.Windows.Forms.ToolStrip toolStrip1;
                        private System.Windows.Forms.ToolStripLabel label1;
<#
                                    }
#>
        private System.Windows.Forms.ToolStripTextBox textBox1;
<#
                        }
                        if (forma.ArTuriDatagrid)
                        {
#>
                        private System.Windows.Forms.DataGridView
dataGridView1;
<#
                        }

                        if (forma.arTuriMeniu)
                        {
#>
                        private System.Windows.Forms.MenuStrip menuStrip1;
<#
                                    foreach (MenuItem menu in forma.MenuItems)
                                    {
#>
<#
```

```
                                    if (menu.periodine)
                                    {
#>
                        private System.Windows.Forms.Timer <#=
menu.pavadinimas#>Timer;
<#
                                    }
#>
                        private System.Windows.Forms.ToolStripMenuItem
menuItem_<#= menu.pavadinimas#>;
<#
                                                foreach (MenuItem menu1 in
menu.TargetMenuItems)
                                                {
#>
<#
                                    if (menu1.periodine)
                                    {
#>
                        private System.Windows.Forms.Timer <#=
menu1.pavadinimas#>Timer;
<#
                                    }
#>
                        private System.Windows.Forms.ToolStripMenuItem
menuItem_<#= menu1.pavadinimas#>;
<#
                                                                foreach (MenuItem
menu2 in menu1.TargetMenuItems)
                                                                {
#>
<#
                                    if (menu2.periodine)
                                    {
#>
                        private System.Windows.Forms.Timer <#=
menu2.pavadinimas#>Timer;
<#
                                    }
#>
                        private System.Windows.Forms.ToolStripMenuItem
menuItem_<#= menu2.pavadinimas#>;
<#
                                                                }
                                                }
                                    }
                        }
#>
    }

<#
            }
#>
}
```

Transformacijos šablonas „klases.tt“

```
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransfor
mation" #>
<#@ output extension=".cs" #>
<#@ VartotojoSasaja processor="VartotojoSasajaDirectiveProcessor"
requires="fileName='../Test.UIdsl'" #>
<#@ import namespace="System.Collections" #>
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Data.Sql;
//using System.Threading;

namespace Debugging
{
<#
            foreach (Forma forma in this.Main.Forma)
            {
#>
            public partial class <#= forma.pavadinimas#> : Form
    {
<#
                                    if (forma.ArTuriDatosPasirinkima == true)
                                    {
#>
                        private ToolStripControlHost dtTScomponent;
<#
                                    }
#>
        public <#= forma.pavadinimas#>()
        {
            InitializeComponent();
<#
                                    if (forma.ArTuriDatosPasirinkima == true)
                                    {
#>
                                    dtTScomponent = new
ToolStripControlHost(dateTimePicker1);
            toolStrip1.Items.Add(dtTScomponent);
<#
                                    }
#>
        }


<#
                        if (forma.arTuriMeniu)
                        {
#>
                        //meniu maksimaliai gali tureti 3 lygius
                        //private System.Windows.Forms.MenuStrip menuStrip1;
<#
                                    foreach (MenuItem menu in forma.MenuItems)
                                    {
                                    if (menu.periodine)
                                    {
#>
                        private void <#= menu.pavadinimas#>Timer_Tick(object
sender, EventArgs e)
        {
            Vykdyti_<#= menu.pavadinimas#>();
        }
<#
                                    }
```

```
                                            if
(menu.ArTuriPaspaudimoVeiksma)
                                            {
#>
        private void <#= menu.pavadinimas#>_Click(object sender, EventArgs
e)
        {
                                //visus esamus taimerius disablinam:
                                System.Windows.Forms.Timer tm;
            foreach (Component c in this.components.Components)
            {
                if (c.GetType() == typeof(System.Windows.Forms.Timer))
                {
                    tm = (System.Windows.Forms.Timer)c;
                    tm.Enabled = false;
                }
            }
<#
                                if (menu.tipas == Tipas.sp)
                                {
#>
                                Vykdyti_<#= menu.pavadinimas#>();
<#
                                }
                                if (menu.tipas == Tipas.funkcija)
                                {
#>
                                Vykdyti_<#= menu.pavadinimas#>();
<#
                                }
                                if (menu.periodine)
                                {
#>
            this.<#= menu.pavadinimas#>Timer.Enabled = true;
            this.<#= menu.pavadinimas#>Timer.Interval = <#=
menu.periodas#>;
<#
                                }
#>
        }

        private void Vykdyti_<#= menu.pavadinimas#>()
        {
                                //string pri = @"data
source=192.168.1.253;Database=TeledemaDB;Uid=TeledemaUser;Pwd=dema";
                                string pri = @"data source=JUSTO-
NB\SQLEXPRESS;Database=TeledemaDB;Uid=justas;Pwd=justas1";
                                DataSet ds = new DataSet();
            SqlConnection conn = new SqlConnection(pri);
            SqlCommand cmd = new SqlCommand(String.Format("<#=
menu.tekstas#>"<#
                                            if (menu.parametras ==
Parametras.data)
                                            {
                                            #>, dateTimePicker1.Value<#
                                            }
                                            if (menu.parametras ==
Parametras.kintamasis)
                                            {
                                            #>, textBox1.Value<#
                                            }
                                            if (menu.parametras ==
Parametras.dataIrKintamasis)
                                            {
```

```
                                                #>, textBox1.Value,
dateTimePicker1.Value<#
                                                }
            #>), conn);
            conn.Open();
            SqlDataAdapter adapter = new SqlDataAdapter(cmd);
            adapter.Fill(ds);
            conn.Close();
            dataGridView1.DataSource = ds.Tables[0];
        }
<#
                                                }
                                                if (menu.Forma1.Count > 0 )
                                                {
#>
                        private void <#= menu.pavadinimas#>_Click(object
sender, EventArgs e)
        {
            //kuriam nauja forma
<#
                                    Forma f = menu.Forma1[0];
#>
            <#= f.pavadinimas#> form = new <#= f.pavadinimas#>();
            form.Show();
        }
<#
                                                }
                                                foreach (MenuItem menu1 in
menu.TargetMenuItems)
                                                {

                                    if (menu1.periodine)
                                    {
#>
                        private void <#= menu1.pavadinimas#>Timer_Tick(object
sender, EventArgs e)
        {
            Vykdyti_<#= menu1.pavadinimas#>();
        }
<#
                                    }

                                                            if
(menu1.ArTuriPaspaudimoVeiksma)
                                                            {
#>
                        private void <#= menu1.pavadinimas#>_Click(object
sender, EventArgs e)
        {
                                    //visus esamus taimerius disablinam:
                                    System.Windows.Forms.Timer tm;
            foreach (Component c in this.components.Components)
            {
                if (c.GetType() == typeof(System.Windows.Forms.Timer))
                {
                    tm = (System.Windows.Forms.Timer)c;
                    tm.Enabled = false;
                }
            }
<#
                                    if (menu1.tipas == Tipas.sp)
                                    {
#>
                                    Vykdyti_<#= menu1.pavadinimas#>();
```

```
<#
                                    }
                                    if (menu1.tipas == Tipas.funkcija)
                                    {
#>
                                    Vykdyti_<#= menu1.pavadinimas#>();
<#
                                    }
                                    if (menu1.periodine)
                                    {
#>
            this.<#= menu1.pavadinimas#>Timer.Enabled = true;
            this.<#= menu1.pavadinimas#>Timer.Interval = <#=
menu1.periodas#>;
<#
                                    }
#>
        }

        private void Vykdyti_<#= menu1.pavadinimas#>()
        {
                                    //string pri = @"data
source=192.168.1.253;Database=TeledemaDB;Uid=TeledemaUser;Pwd=dema";
                                    string pri = @"data source=JUSTO-
NB\SQLEXPRESS;Database=TeledemaDB;Uid=justas;Pwd=justas1";
                                    DataSet ds = new DataSet();
            SqlConnection conn = new SqlConnection(pri);
            SqlCommand cmd = new SqlCommand(String.Format("<#=
menu1.tekstas#>"<#
                                                if (menu1.parametras ==
Parametras.data)
                                                {
                                                #>, dateTimePicker1.Value<#
                                                }
                                                if (menu1.parametras ==
Parametras.kintamasis)
                                                {
                                                #>, textBox1.Value<#
                                                }
                                                if (menu1.parametras ==
Parametras.dataIrKintamasis)
                                                {
                                                #>, textBox1.Value,
dateTimePicker1.Value<#
                                                }
            #>), conn);
            conn.Open();
            SqlDataAdapter adapter = new SqlDataAdapter(cmd);
            adapter.Fill(ds);
            conn.Close();
            dataGridView1.DataSource = ds.Tables[0];
        }
<#
                                                        }
                                                        if
(menu1.Forma1.Count > 0 )
                                                        {
#>
                        private void <#= menu1.pavadinimas#>_Click(object
sender, EventArgs e)
        {
            //kuriam nauja forma
<#
                                    Forma f = menu1.Forma1[0];
```

```
#>
            <#= f.pavadinimas#> form = new <#= f.pavadinimas#>();
            form.Show();
        }
<#
                                                            }
                                                            foreach (MenuItem
menu2 in menu1.TargetMenuItems)
                                                            {

                    if (menu.periodine)
                                {
#>
                    private void <#= menu2.pavadinimas#>Timer_Tick(object
sender, EventArgs e)
        {
            Vykdyti_<#= menu2.pavadinimas#>();
        }
<#
                                }

                                                                    if
(menu2.ArTuriPaspaudimoVeiksma)
                                                                    {
#>
                    private void <#= menu2.pavadinimas#>_Click(object
sender, EventArgs e)
        {
                                //visus esamus taimerius disablinam:
                                System.Windows.Forms.Timer tm;
            foreach (Component c in this.components.Components)
            {
                if (c.GetType() == typeof(System.Windows.Forms.Timer))
                {
                    tm = (System.Windows.Forms.Timer)c;
                    tm.Enabled = false;
                }
            }
<#
                                if (menu2.tipas == Tipas.sp)
                                {
#>
                                Vykdyti_<#= menu2.pavadinimas#>();
<#
                                }
                                if (menu2.tipas == Tipas.funkcija)
                                {
#>
                                Vykdyti_<#= menu2.pavadinimas#>();
<#
                                }
                                if (menu2.periodine)
                                {
#>
            this.<#= menu2.pavadinimas#>Timer.Enabled = true;
            this.<#= menu2.pavadinimas#>Timer.Interval = <#=
menu2.periodas#>;
<#
                                }
#>
        }

        private void Vykdyti_<#= menu2.pavadinimas#>()
        {
```

```
                                    //string pri = @"data
source=192.168.1.253;Database=TeledemaDB;Uid=TeledemaUser;Pwd=dema";
                                    string pri = @"data source=JUSTO-
NB\SQLEXPRESS;Database=TeledemaDB;Uid=justas;Pwd=justas1";
                                    DataSet ds = new DataSet();
            SqlConnection conn = new SqlConnection(pri);
            SqlCommand cmd = new SqlCommand(String.Format("<#=
menu2.tekstas#>"<#
                                                if (menu2.parametras ==
Parametras.data)
                                                {
                                                #>, dateTimePicker1.Value<#
                                                }
                                                if (menu2.parametras ==
Parametras.kintamasis)
                                                {
                                                #>, textBox1.Value<#
                                                }
                                                if (menu2.parametras ==
Parametras.dataIrKintamasis)
                                                {
                                                #>, textBox1.Value,
dateTimePicker1.Value<#
                                                }
            #>), conn);
            conn.Open();
            SqlDataAdapter adapter = new SqlDataAdapter(cmd);
            adapter.Fill(ds);
            conn.Close();
            dataGridView1.DataSource = ds.Tables[0];
        }
<#
                                                                        }
                                                                        if
(menu2.Forma1.Count > 0 )
                                                                        {
#>
                        private void <#= menu2.pavadinimas#>_Click(object
sender, EventArgs e)
        {
            //kuriam nauja forma
<#
                                    Forma f = menu2.Forma1[0];
#>
            <#= f.pavadinimas#> form = new <#= f.pavadinimas#>();
                                    form.Show();
        }
<#
                                                                        }
                                                                }
                                                }
                                    }
                        }
#>
<#
                        if (forma.ArTuriDatagrid)
                        {
#>
                        private void Form1_Resize(object sender, EventArgs e)
        {
            dataGridView1.Size = this.Size;
        }
<#
                        }
```

```
#>

    }
<#
            }
#>
}
```

Transformacijos šablonas „pagalbine.tt“

```
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransfor
mation" #>
<#@ output extension=".cs" #>
<#@ VartotojoSasaja processor="VartotojoSasajaDirectiveProcessor"
requires="fileName='../Test.UIdsl'" #>
<#@ import namespace="System.Collections" #>
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Debugging
{
           public partial class <#= Main.formosVardas#>
           {
                     public void <#= Main.metodoPavadinimas#>()
                     {
<#
                                foreach (Forma forma in this.Main.Forma)
                                {
                                          if (forma.pagrindine == true)
                                          {
#>
                                <#= forma.pavadinimas#> form = new <#=
forma.pavadinimas#>();
            form.Show();
<#
                                          }
                                }
#>
                     }
           }
}
```

Validavimo klasių ir metodų failas „Validation.cs“

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.VisualStudio.Modeling.Validation;

namespace UI.VartotojoSasaja
{
    [ValidationState(ValidationState.Enabled)]
    public partial class Main
    {
        [ValidationMethod(ValidationCategories.Menu |
ValidationCategories.Save)]
        private void ValidatePagrindoElementus(ValidationContext context)
        {
```

```
            if (string.IsNullOrEmpty(this.formosVardas))
            {
                context.LogError("Pagrindine klase privalo tureti
pavadinima", "MN1001", this);
            }

            if (this.formosVardas.Contains(" "))
            {
                context.LogError("Pagrindines klases pavadinimas negali
tureti tarpu", "MN2001", this);
            }

            if (string.IsNullOrEmpty(this.metodoPavadinimas))
            {
                context.LogError("Pagrindine klase privalo tureti paleidimo
metoda", "MN3001", this);
            }

            if (this.metodoPavadinimas.Contains(" "))
            {
                context.LogError("Pagrindines klases metodas negali tureti
tarpu", "MN4001", this);
            }
        }

        [ValidationMethod(ValidationCategories.Menu |
ValidationCategories.Save)]
        private void ArYraPagrindineForma(ValidationContext context)
        {
            int sk = 0;
            foreach (Forma f in this.Forma)
            {
                if (f.pagrindine)
                    sk++;
            }
            if (sk != 1)
            {
                context.LogError(String.Format("Rasta {0} pagrindiniu formu
(gali buti tik 1)", sk), "M1001", this);
            }
        }
    }


    [ValidationState(ValidationState.Enabled)]
    public partial class Forma
    {
        [ValidationMethod(ValidationCategories.Menu |
ValidationCategories.Save)]
        private void ValidateName(ValidationContext context)
        {
            if (string.IsNullOrEmpty(this.pavadinimas))
            {
                context.LogError("Forma privalo tureti pavadinima",
"FN1001", this);
            }

            if (this.pavadinimas.Contains(" "))
            {
                context.LogError("Formos pavadinimas negali tureti tarpu",
"FN2001", this);
            }
        }
```

```
    }


    [ValidationState(ValidationState.Enabled)]
    public partial class MenuItem
    {
        [ValidationMethod(ValidationCategories.Menu |
ValidationCategories.Save)]
        private void ValidateMaxMenuLevel(ValidationContext context)
        {
            string kelias = this.pavadinimas;
            foreach (MenuItem meniu in this.TargetMenuItems)
            {
                if (meniu.TargetMenuItems != null)
                {
                    kelias += " -> " + meniu.pavadinimas;
                    //sk++;
                    foreach (MenuItem meniu1 in meniu.TargetMenuItems)
                    {
                        if (meniu1.TargetMenuItems != null)
                        {
                            kelias += " -> " + meniu1.pavadinimas;
                            foreach (MenuItem meniu2 in
meniu1.TargetMenuItems)
                            {
                                if (meniu2.TargetMenuItems != null)
                                {
                                    kelias += " -> " + meniu2.pavadinimas +
" ->";
                                    context.LogError(String.Format("Meniu
seka  {0} yra per ilga (gali buti 3 lygiu meniu).", kelias), "MMK001",
this);
                                }
                            }
                        }
                    }
                }
            }
        }

        [ValidationMethod(ValidationCategories.Menu |
ValidationCategories.Save)]
        private void ValidateName(ValidationContext context)
        {
            if (string.IsNullOrEmpty(this.pavadinimas))
            {
                context.LogError("Meniu privalo tureti pavadinima",
"FN1001", this);
            }

            if (this.pavadinimas.Contains(" "))
            {
                context.LogError("Meniu pavadinimas negali tureti tarpu",
"FN2001", this);
            }
        }

        [ValidationMethod(ValidationCategories.Menu |
ValidationCategories.Save)]
        private void ValidateEndItem(ValidationContext context)
        {
            if ((this.TargetMenuItems.Count == 0) && (this.tipas ==
Tipas.none))
            {
```

```
                context.LogWarning("Galinis meniu punktas neturi jokio
veiksmo", "ME001", this);
            }
        }
    }
}
```