Kauno technologijos universitetas

Informatikos fakultetas

Modeliais grindžiamos architektūros principų taikymas Laravel karkaso PHP kodui generuoti

Baigiamasis magistro krypties studijų projektas

Mantas Ražinskas

Projekto autorius

doc. Lina Čeponienė

Vadovė

Kaunas, 2021

Kauno technologijos universitetas

Informatikos fakultetas

Modeliais grindžiamos architektūros principų taikymas Laravel karkaso PHP kodui generuoti

Baigiamasis magistro projektas

Informacinių sistemų inžinerija (6211BX009)

Mantas Ražinskas
Projekto autorius

doc. Lina Čeponienė
Vadovė

lekt. Lina Bisikirskienė
Recenzentė

Kaunas, 2021

Kauno technologijos universitetas

Informatikos fakultetas

Mantas Ražinskas

Modeliais grindžiamos architektūros principų taikymas Laravel karkaso PHP kodui generuoti

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Mantas Ražinskas

Patvirtinta elektroniniu būdu

Ražinskas, Mantas. Modeliais grindžiamos architektūros principų taikymas Laravel karkaso PHP kodui generuoti. Magistro projektas / vadovė doc. dr. Lina Čeponienė; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informacijos sistemos, Informatikos mokslai.

Reikšminiai žodžiai: MDA, UML, kodo generavimas, PHP, Laravel.

Kaunas, 2021. 75 p.

Santrauka

Šiuolaikinis poreikis kurti informacines sistemas kokybiškai ir kuo mažesniais laiko kaštais sparčiai keičia jų kūrimo metodikas, didesnis dėmesys skiriamas informacinių sistemų kūrimo patogumui ir jų priežiūrai. Organizacijos, norėdamos sumažinti didelius technologinius migracijos kaštus kuriant programinę įrangą, siekia taikyti modeliais grindžiamos architektūros principus, tačiau nėra sukurtų pakankamai išsamių metodikų ir jas apimančių algoritmų, padedančių sugeneruoti PHP programinį kodą interneto informacinėms sistemoms. Šiuo metu interneto svetainėms kurti naudojamai vienai populiariausių programavimo kalbų, PHP, ir jos Laravel karkasui nėra sukurtos metodikos, skirtos pritaikyti MDA modelių transformacijas ir kodo generavimą. Dabartinė pasiūla nepatenkina sparčiai augančio poreikio.

Šio darbo tikslas – palengvinti interneto informacinių sistemų, realizuojamų PHP programavimo kalba, kūrimą, pasiūlant MDA grindžiamą metodiką, apimančią modelių transformacijas ir kodo generavimą. Tikslui pasiekti formaliai apibrėžta ir praktiškai išbandyta metodika, apimanti PIM ir PSM modelių struktūros specifikacijas, paruoštą PHP programavimo kalbos Laravel karkasui pritaikytą profilį ir apibrėžtus algoritmų aprašus skirtus modelių transformacijoms ir PHP Laravel kodo generavimui. Metodikai įgyvendinti panaudoti Eclipse ATL ir Eclipse Acceleo transformacijų įrankiai, kuriems parengtos transformacijų konfigūracijos ir transformacijų algoritmai.

Siekiant įvertinti sukurtą MDA grindžiamą metodiką, apimančią modelių transformacijas ir PHP Laravel kodo generavimą, atliktas eksperimentas, kurio metu siekta nustatyti, ar modelių transformacijos ir kodo generavimas PHP Laravel karkasui palengvintų informacinių sistemų, realizuojamų šiuo karkasu, kūrimą. Eksperimento rezultatai parodė, kad tyrimo objekto naudotojai sutinka, jog modelių transformacijos ir kodo generavimas PHP Laravel karkasui palengvintų informacinių sistemų, realizuojamų šiuo karkasu ir programavimo kalba, kūrimą. Taip pat, atlikus palyginimo eksperimentus, nustatyta, kad iš modelio sugeneruotas kodas padengia daugiau veikiančios sistemos funkcionalumo nei programinis kodas, kurtas su Artisan komandinės eilutės sąsaja.

Ražinskas, Mantas. Applying Model Driven Architecture Principles for Laravel Framework PHP Code Generation. Master's Final Degree Project / supervisor doc. dr. Lina Čeponienė; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Information Systems, Computing.

Keywords: MDA, UML, code generation, PHP, Laravel.

Kaunas, 2021. 75 p.

Summary

The need to create information systems qualitatively within the least possible duration rapidly changes the creating methodologies. More attention is paid to the convenience of creation and maintenance. In order to decrease the technological migration costs while creating the software, organizations aim to apply the principles of model-driven architecture. However, there are no comprehensive methodologies and their algorithms that would help generating PHP code for the internet information systems. Nowadays, PHP is one of the most popular programming languages which is used to create websites, however, there is no methodology that would help applying MDA model transformations and code generation for PHP and its Laravel framework. At the moment, the market cannot fulfil the needs of the users.

The aim of this work is to make the creation of internet information systems, that are created with PHP, easier by offering MDA-based methodology that includes model transformations and code generation. To achieve the intended goal, a methodology for model transformations and PHP Laravel programming code generation has been created in theory and applied in practice. In order to implement the methodology, ATL and Acceleo transformation tools have been used with the created transformation configurations and transformation algorithms.

An experiment has been carried out in order to evaluate the created MDA-based methodology of model transformations and PHP Laravel code generation. The aim of the experiment was to evaluate whether model transformations and code generation for PHP Laravel framework would ease the creation of information systems. The results showed that the users agree that model transformations and code generation for PHP Laravel framework make the creation of information systems less complicated. Also, the comparative experiments showed that the code generated from the model covers more functionality of the existing system than the programming code created with the Artisan command line interface.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

OMG (angl. Object Management Group) – tarptautinis technologinių standartų konsorciumas.

UML (angl. Unified Modeling Language) – vieninga modeliavimo ir specifikacijų kūrimo kalba.

CIM (angl. Computation independent model) – nuo skaičiavimų nepriklausomas modelis.

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

PSM (angl. Platform Specific Model) – specifinis platformos modelis.

MDA (angl. Model Driven Architecture) – modeliais grindžiama architektūra.

MBE (angl. Model Based Engineering) – pagrindimas modeliais.

MDE (angl. Model Driven Engineering) – modeliais grindžiama inžinerija.

MDD (angl. Model Driven Development) – modeliais grindžiamas kūrimas.

QVT (angl. Query / View / Transformation) – transformacijos kalba.

MOF (angl. Meta Object Facility) – objektų valdymo grupės sukurtas standartas.

XMI (angl. XML Metadata Interchange) – metaduomenų mainų standartas.

M2M (angl. Model to Model) – transformaciją iš modelio į modelį.

M2T (angl. Model to Text) – transformacija iš modelio į programinį kodą.

IS (angl. Information System) – informacinė sistema.

MVC (angl. Model – View – Controller) – architektūra suteikianti galimybę kurti modulines, struktūrizuotas programas.

ATL – įrankis skirtas transformuoti modelius iš nuo platformos nepriklausomo modelio į platformai specifinį modelį.

Acceleo – įrankis skirtas atlikti transformacijas iš platformai specifinio modelio į programinį kodą.

LaravelStats – automatinio kodo palyginimo įrankis.

PHPLoc – automatinio kodo palyginimo įrankis.

Įvadas

Šiuolaikinis poreikis kurti informacines sistemas kokybiškai ir kuo mažesniais laiko kaštais sparčiai keičia jų kūrimo metodikas, didesnis dėmesys yra skiriamas informacinių sistemų (IS) kūrimo patogumui ir jų priežiūrai. Vienas iš galimų variantų norint kurti sistemas efektyviau yra naudotis kompiuterinių pramonės standartų konsorciumo (OMG, angl. Object Management Group) sukurtu vieningos modeliavimo kalbos standartu (UML, angl. Unified Modeling Language) [1], suteikiančiu inžinerinius įrankius nuo paprastų iki sudėtingų sistemų modeliavimui ir šio standarto modeliais grindžiamos architektūros (MDA, angl. Model Driven Architechture) [2] principu, grindžiamu modeliais ir transformacijomis tarp jų. Modeliai gali būti skirtingų tipų, pavyzdžiui: nuo realizavimo platformos nepriklausomas modelis (PIM, angl. Platform Independent Model) neatsižvelgiant į realizacijos technologijas apibrėžia abstrakčias sistemos funkcijas, o konkrečiai realizavimo platformai pritaikytas modelis (PSM, angl. Platform Specific Model) apibrėžia sistemos funkcijų įgyvendinimą konkrečiai platformai. Apibrėžus PSM modelį galima sugeneruoti programinį kodą pasirinktai programavimo kalbai [3] (C#, Java, C++ ar kitai, kuriai ši galimybė yra realizuota), kuris atitinka PSM modelyje apibrėžtus elementus. Modelių transformacijos suteikia galimybę turėti sistemos modelį ir programinį kodą, taip sumažinant galimų klaidų riziką ir padarant IS patogiai modifikuojamą.

Modeliais grindžiamos architektūros principai taikomi įvairių IS kūrimui, įskaitant ir internetines informacines sistemas. Kuriant internetines IS dažnai naudojama PHP programavimo kalba, kuri tarp interneto svetainėms kurti skirtų programavimo kalbų yra viena populiariausių ir tinkamiausių programavimo kalbų [4]. Siekiant palengvinti kūrimą vienas iš variantų yra naudoti PHP programavimo kalbos karkasus. Karkasai palengvina sistemų kūrimo procesą, turi paruoštą struktūrą, supaprastinto darbo su duomenų baze galimybes, juos lengvai galima praplėsti naudojant papildomas bibliotekas ir jie grįsti MVC architektūros principu [4].

MVC architektūra suteikia galimybę kurti modulines, struktūrizuotas programas. Ši architektūra dalina programas į skirtingas dalis ir suteikia galimybę jas analizuoti ir realizuoti atskirai. MVC architektūra, atskirdama modelius nuo vaizdų, sumažina programų struktūros sudėtingumą, padaro programinį kodą lengviau pasikartotinai naudojamą [4].

Darbo problematika ir aktualumas

Organizacijos patiria didelius technologinius migracijos kaštus, kurdamos programinę įrangą skirtą naudoti skirtingose realizacijos platformose (informacinės sistemos, kompiuterinės programos, programėlės išmaniesiems ir kt.). Šiuos kaštus padėtų sumažinti MDA principų taikymas programinės įrangos kūrimo procese, tačiau nėra sukurtų pakankamai išsamių metodikų ir jas apimančių algoritmų, padedančių sugeneruoti PHP programinį kodą internetinėms sistemoms. Šiuo metu vienai iš populiariausių internetinėms svetainėms kurti naudojamai PHP programavimo kalbai ir jos Laravel karkasui nėra sukurtos metodikos, skirtos jai pritaikyti MDA modelių transformacijas ir kodo generavimą, ir tai lemia augančio poreikio pasiūlos trūkumą [5].

Darbo tikslas ir uždaviniai

Tikslas – palengvinti internetinių informacinių sistemų, realizuojamų PHP programavimo kalba, kūrimą, pasiūlant MDA architektūra grindžiamą metodiką, apimančią modelių transformacijas ir kodo generavimą.

Norint pasiekti pagrindinį darbo tikslą, iškelti šie uždaviniai:

1. išanalizuoti UML modeliavimo kalbą, jos galimybes ir modeliais grindžiamą architektūrą;
2. išanalizuoti egzistuojančias modelių transformacijas MDA kontekste, jų įrankius ir egzistuojančius kodo generavimo sprendimus;
3. išanalizuoti UML sąsajas su PHP kalba ir šios kalbos karkasais bei išanalizuoti UML transformacijų į PHP Laravel programinį kodą esamus sprendimus;
4. sukurti metodiką, apimančią algoritmus modelių transformacijoms ir kodo generavimui;
5. realizuoti metodiką su pasiūlytais algoritmais ir ją ištestuoti paruošiant modelius, iš kurių turi būti sugeneruotas PHP Laravel programinis kodas;
6. eksperimentiškai ištirti sukurtą metodikos realizaciją, panaudojant ją informacinei sistemai kurti;
7. apibendrinti tyrimo rezultatus.

Darbo rezultatai ir jų svarba

MDA principais grindžiama ir pasirinktam PHP kalbos karkasui pritaikyta metodika, apimanti PIM ir PSM struktūros specifikacijas, PHP Laravel karkasui pritaikytą UML profilį PSM ir algoritmus transformacijoms iš PIM į PSM ir iš PSM į programinį kodą.

Darbo struktūra

Magistro baigiamasis darbas sudarytas iš septynių skyrių, literatūros sąrašo ir priedų. Įvade pristatoma darbo problematika, jos aktualumas, nurodomas darbo tikslas ir apibrėžiami iškelti uždaviniai. Probleminės srities analizės dalyje pateikiama MDA kontekste atliekamų modelių transformacijų ir kodo generavimo analizė, tyrimo objekto naudotojų ir esamų problemos sprendimų metodų analizė, taip pat pateikiamas siekiamo sprendimo apibrėžimas. Sprendimo reikalavimų specifikacijos ir projekto formalaus aprašo dalyje pateikiama metodikos reikalavimų specifikaciją, šios metodikos dalykinės srities modelis ir transformacijos iš modelio į modelį ir iš modelio į programinį kodą metodikos formalus aprašas. Eksperimentinės realizacijos projekto aprašo dalyje pateikiamas detalus transformacijos iš modelio į modelį ir iš modelio į PHP Laravel programinį kodą eksperimentinės realizacijos projektas ir jo dedamosios dalys. Sprendimo realizacijos ir testavimo dalyje pateikiamas metodikos realizavimo sprendimas, rankinio testavimo scenarijai, duomenys ir rezultatai. Eksperimentinio sprendimo tyrimo dalyje aprašomas keturių dalių eksperimentas ir jo rezultatai. Pabaigoje pateikiamos darbo išvados, išaiškėjusios atlikus magistro baigiamąjį darbą.

1. Probleminės srities analizė

1.1. Analizės tikslas

Šiame darbe atliekamos analizės tikslas – išsiaiškinti apie MDA architektūros kontekste atliekamas transformacijas iš modelio į modelį ir iš modelio į programos kodą. Tai bus pasiekta išanalizuojant:

- UML modeliavimo kalbą, jos galimybes ir modeliais grindžiamą architektūrą;
- egzistuojančias modelių transformacijas MDA kontekste, jų įrankius ir egzistuojančius kodo generavimo iš PSM sprendimus;
- UML sąsajas su PHP kalba ir šios kalbos karkasais bei išanalizuoti UML transformacijų iš PIM į PSM ir iš PSM į PHP Laravel programinį kodą esamus sprendimus;
- tyrimui aktualių UML diagramų metamodelių pagrindinius elementus, MOF (angl. Meta Object Facility) paskirtį ir UML metamodelio išplėtimo profiliais galimybes.

1.2. Tyrimo objektas, sritis ir problema

Tyrimo sritis – modeliais grindžiama architektūra (MDA), šia architektūra grindžiamas sistemų kūrimas (MDD) ir kūrimo metu naudojami metodai, įrankiai ir technologijos.

Tyrimo objektas – MDA architektūros kontekste atliekamos transformacijos iš modelio į modelį ir iš modelio į programos kodą.

Tyrimo problema – nors organizacijos, norėdamos sumažinti didelius technologinius migracijos kaštus šiais laikais siekia taikyti MDA principus kurdamos programinę įrangą, tačiau nėra sukurtų pakankamai išsamių metodikų ir jas apimančių algoritmų, padedančių sugeneruoti PHP programinį kodą internetinėms sistemoms. Šiuo metu vienai iš populiariausių internetinių svetainių kūrimui PHP programavimo kalbai ir jos Laravel karkasui nėra sukurtos metodikos, norint jai pritaikyti MDA modelių transformacijas ir kodo generavimą, kas lemia augančio poreikio pasiūlos trūkumą [5].

1.3. MDA kontekste atliekamų modelių transformacijų ir kodo generavimo analizė

1.3.1. Vieninga modeliavimo kalba (UML)

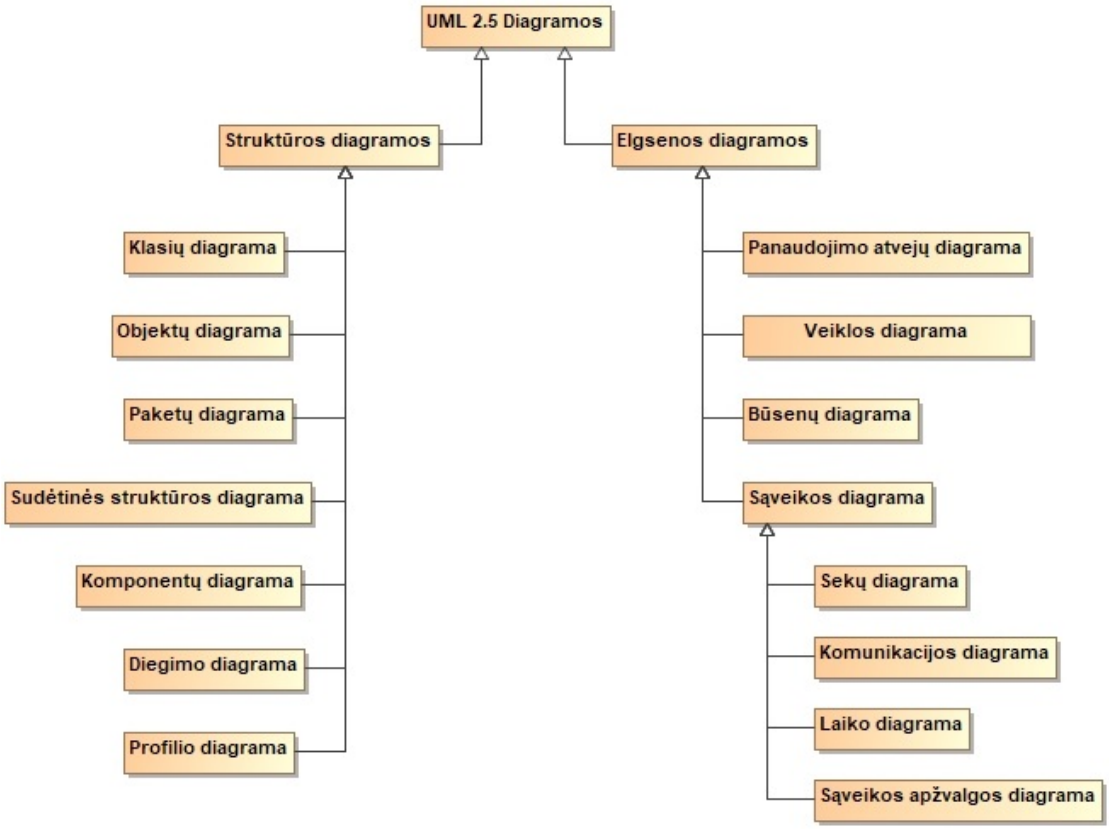
UML [1] modeliavimo kalbos paskirtis yra suteikti sistemų architektams, sistemų inžinieriams ir sistemų kūrėjams įrankius, skirtus analizei, projektavimui ir sistemų įgyvendinimui, taip pat įrankius modeliuoti verslo ar kitus procesus. Vienas iš pagrindinių UML tikslų yra patobulinti technologijų industriją įgalinant vizualinį modeliavimą, suteikiant tam reikalingus įrankius. Norint sudaryti galimybę keistis modelių informaciją tarp įrankių, būtinas susitarimas dėl sintaksės ir semantikos. UML atitinka toliau nurodytus reikalavimus [1]:

- UML turi oficialų apibrėžimą, pagrįstą MOF pagrindu sukurtu metamodeliu, nurodančiu abstrakčią UML sintaksę. Abstrakčioji sintaksė nusako UML modeliavimo sąvokų rinkinį, jų požymius ir ryšius, taip pat šių sąvokų derinimo taisyklės, siekiant sudaryti dalinius ar pilnus UML modelius.
- UML turi išsamius kiekvienos UML modeliavimo koncepcijos semantikos paaiškinimus. Semantika apibrėžiama nuo realizacijos technologijos nepriklausančia struktūra.
- UML suteikia žmonėms suprantamų žymėjimo elementų, apibūdinančių individualų UML modelį, specifikacijos sąvokas bei taisykles kaip jas sujungti į daugybę skirtingų diagramų tipų, atitinkančių skirtingus modeliuojamų sistemų aspektus.

Dėl UML galimybių spręsti plataus pobūdžio problemas, UML turi įvairių diagramų tipų [1]. UML diagrama yra dalinis grafinis kuriamos, įgyvendinamos ar jau egzistuojančios sistemos modelis (vaizdas). UML diagramoje yra grafiniai elementai (simboliai) – UML mazgai, sujungti su briaunomis (dar vadinamais keliais ar srautais), kurie žymi elementus UML modeliuojamame sistemos modelyje. UML specifikacija nusako du pagrindinius UML diagramų tipus:

- Struktūros diagramos:
  Struktūros diagramos parodo sistemos ir jos dalių statinę struktūrą skirtingais abstrakcijos ir įgyvendinimo lygiais ir kaip jos yra susijusios viena su kita. Struktūros diagramos elementai atspindi prasmingus sistemos aspektus bei gali apimti abstrakčius įgyvendinimo aspektus.
- Elgsenos diagramos:
  Elgsenos diagramos parodo objektų dinaminę elgseną sistemoje, kurią galima apibūdinti kaip sistemos pokyčius bėgant laikui.

1 pav. pateikiamos visos UML 2.5 diagramos ir jų hierarchija.

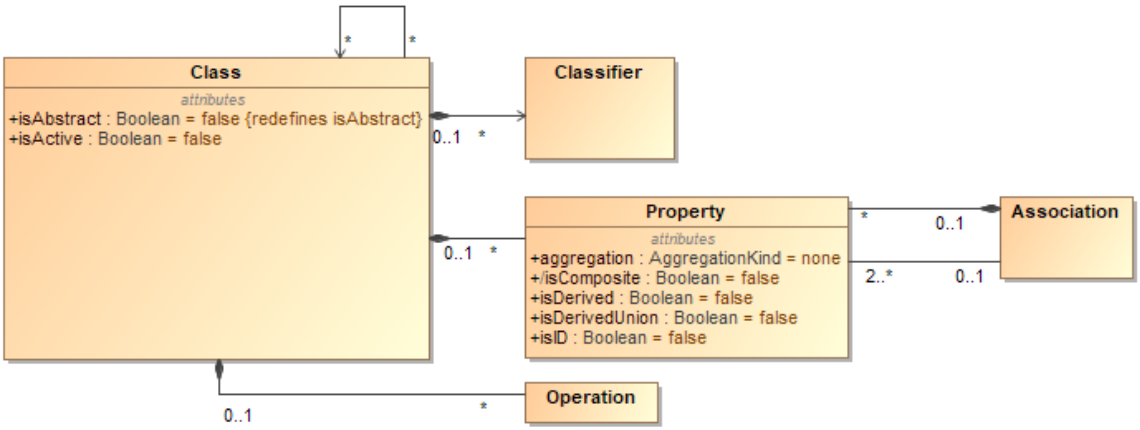


1 pav. UML 2.5 diagramos [1]

Siekiant kuriamuose modeliuose aprašyti sistemos struktūrą ir elgseną, šiame darbe pasirinkta detaliau analizuoti vieną iš struktūrinio tipo diagramų (klasių) ir keletą elgsenos tipo diagramų (veiklos, sekų, būsenų).

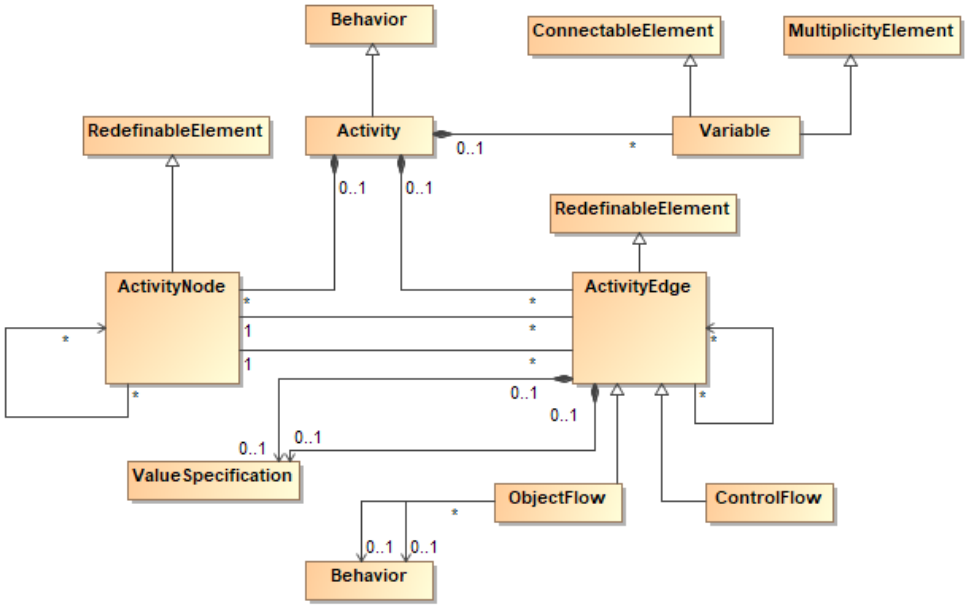
Klasių diagrama – tai diagrama, vaizduojanti suprojektuotos sistemos, posistemio ar komponento struktūrą kaip susijusias klases ir sąsajas su jų ypatybėmis, apribojimais ir ryšiais: asociacijomis, apibendrinimais, priklausomybėmis ir kt.

2 pav. pateikiamas klasių diagramos metamodelio pagrindinis fragmentas.



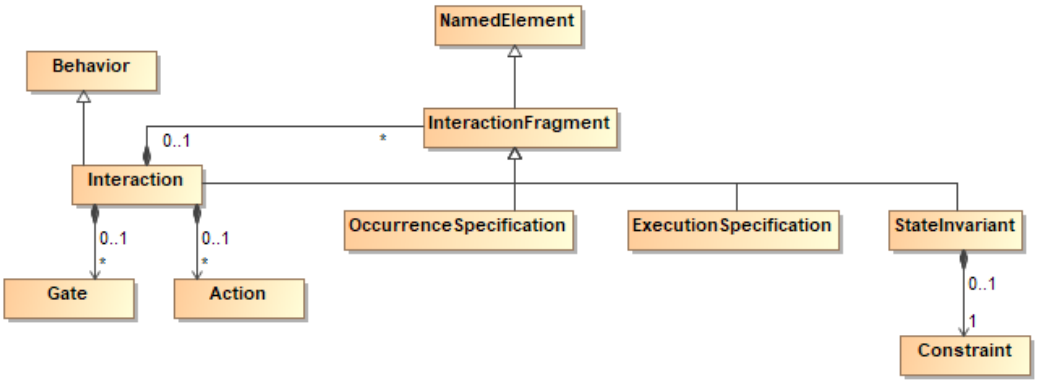
2 pav. Klasių diagramos metamodelio fragmentas [1]

Veiklos diagrama – diagrama, kuri vaizduoja seką ir sąlygas koordinuojant žemesnio lygmens elgsenas. Tai dažniausiai vadinama valdymo srauto ir objektų srauto modeliais. 3 pav. pateikiamas veiklos diagramos metamodelio fragmentas.



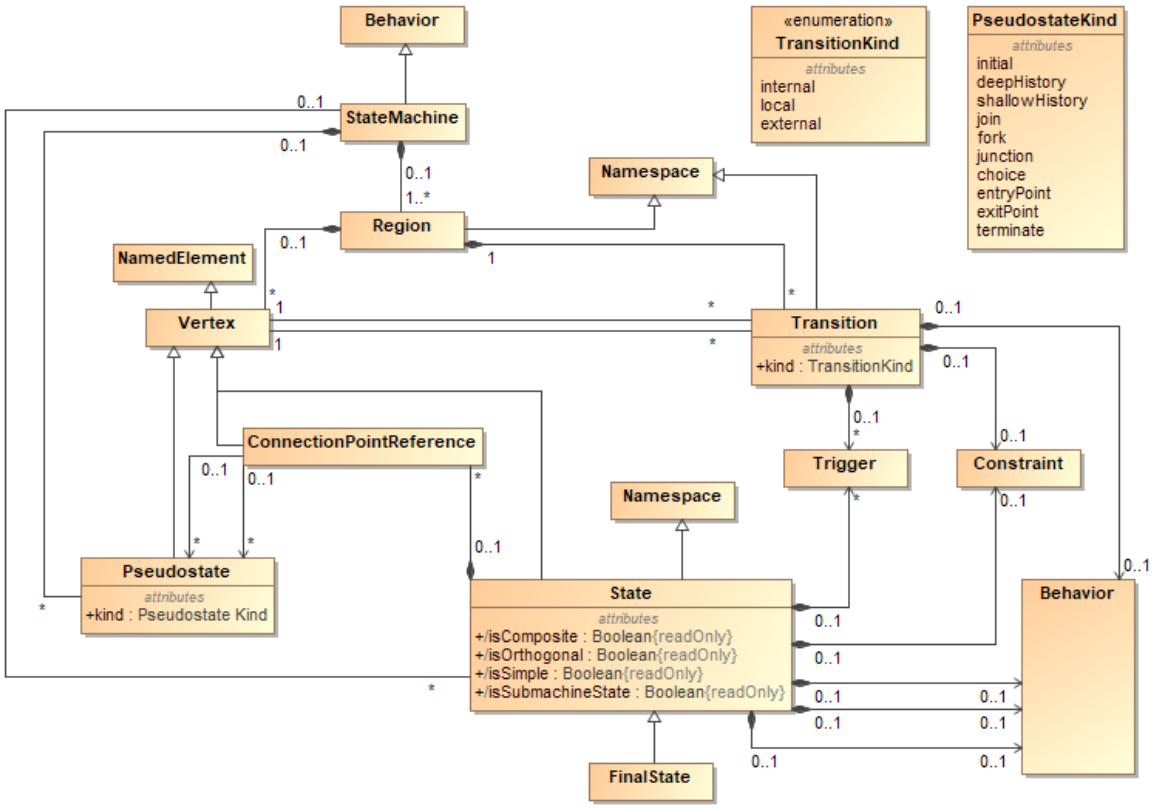
3 pav. Veiklos diagramos metamodelio fragmentas [1]

Sekų diagrama – tai dažniausiai pasitaikantis sąveikos diagramos tipas, kuriame pagrindinis dėmesys skiriamas žinučių keitimuisi tarp linijų (objektų). 4 pav. pateikiamas sekų diagramos metamodelio fragmentas.



4 pav. Sekų diagramos metamodelio fragmentas [1]

Būsenų diagrama – diagrama naudojama modeliuoti atskiras elgsenas per baigtinius būsenų perėjimus. 5 pav. pateikiamas būsenų diagramos metamodelio fragmentas.



5 pav. Būsenų diagramos metamodelio fragmentas [1]

Atlikus UML analizę, išsiaiškinta, kad UML modeliavimo kalba suteikia sistemų architektams, sistemų inžinieriams ir sistemų kūrėjams įrankius, skirtus analizei, projektavimui ir sistemų įgyvendinimui. UML paskirtis yra patobulinti technologijų industriją įgalinant vizualinį modeliavimą ir suteikiant tam reikalingus įrankius. UML turi struktūros ir elgsenos diagramų tipus, kiekviena iš diagramų turi savo metamodelį. Tyrime išanalizavus struktūrinio tipo diagramą ir elgsenos tipo diagramas, jų metamodelius ir išsiaiškinus pagrindinius šių diagramų metaklasių struktūros fragmentus, buvo pasirinkta kuriamus modelius sudaryti iš klasių ir sekų diagramų. Klasių diagrama yra tinkama apibrėžti sistemos struktūrai, o sekų diagrama – sistemos elgsenai.

1.3.2. Modeliavimo įrankiai

Informacinių sistemų modeliavimas yra viena pagrindinių veiklų, kuriant sistemas MDA principu, o norint modeliuoti sistemas, svarbu naudoti įrankius palaikančius UML standartą. Rinkoje yra įvairių įrankių suteikiančių galimybę modeliuoti, pavyzdžiui: Bouml, Enterprise Architect, IMB Rational Rhapsody, MagicDraw, Papyrus, Visual Paradigm, Star UML, Umple ir kt [6]. Vieni plačiausiai naudojamų modeliavimo įrankių išskiriami MagicDraw ir Papyrus [7]. MagicDraw yra vienas pagrindinių komercinių modeliavimo įrankių, o Papyrus yra atvirojo kodo Eclipse modeliavimo įrankis. Nors šių įrankių funkcionalumas turi nemažai skirtumų, tačiau abu turi grafinį redaktorių, kuris palengviną modelių sudarymą.

Elena Planas ir kt. autorių 2020 m. tyrime [7], buvo išanalizuotas ir palygintas MagicDraw ir Papyrus įrankių panaudojamumas. Autoriai įvertino daugiau nei 12 valandų trukmės vaizdo įrašus naudojant šiuos įrankius iš trijų aspektų: modeliavimo proceso, modeliavimui reikalingų pastangų ir iškilusių kliūčių modeliavimo procese. Autoriai taip pat atliko spartos tyrimą MagicDraw ir Papyrus įrankiams (paspaudimų kiekio ir laiko atžvilgiu), ši tyrimo dalis buvo atlikta pateikiant užduotis ekspertines žinias turintiems asmenims. Autorių eksperimentai reikšmingo skirtumo tarp šių įrankių neparodė, rezultatai buvo tolygūs.

Atsižvelgus į autorių tyrimo rezultatus, nors ir MagicDraw ir Papyrus įrankiai yra vieni plačiausiai naudojamų rinkoje ir iš panaudojamumo perspektyvos jie ypatingai didelio skirtumo neturi, tačiau šiame tyrime buvo pasirinkta modeliavimui ir pavyzdžių sudarymui naudoti MagicDraw įrankį.

1.3.3. UML metamodelis ir jo išplėtimo galimybės

UML, MOF vaidina svarbų vaidmenį modeliais grindžiamoje architektūroje teikdamos kalbas, kuriomis kuriami ir transformuojami modeliai.

Abstrakti UML sintaksė nurodoma naudojant UML modelį, vadinamą UML metamodeliu. Šis metamodelis naudoja apriboto UML pogrupio atskaitos tašką, kuris yra apibrėžta MOF specifikacijoje ir naudojamas kuriant metamodelius [1].

MOF

MOF [8] – OMG konsorciumo sukurtas standartas. Tai yra detali iliustracija, rodanti ryšius tarp objektų. UML MOF koncepcija naudojama siekiant atitikti MDA principo reikalavimus [9]. MOF apibrėžia, kaip modeliai yra pasiekiami ir kokie mainai tarp modelių gali vykti. Šiam tikslui yra naudojama OMG kompiuterinės pramonės konsorciumo patvirtinta XML metaduomenų apsikeitimo sąsaja (*angl. XML Meta-data Interchange*) [10].

XMI

UML architektūra suteikia standartą, vadinamą XMI, kuris aprašo metamodelio nuorodas į XML standartus. Pagrindinis XMI tikslas – aprašyti, kaip XML žymomis turi būti atvaizduojamas nuoseklus MOF modelis XML formatu [9].

XMI palengvina metaduomenų mainus tarp UML modeliavimo įrankių ir MOF paremtų metaduomenų saugyklų paskirstytoje nevienalytėje aplinkoje. Kadangi XMI yra pagrįstas XML, tiek metaduomenys (žymos), tiek jų aprašomi egzemplioriai (elemento turinys) gali būti supakuoti tame

pačiame dokumente. Tai leidžia programoms nuosekliai reprezentuoti objektus ir jų asociacijas per jų metaduomenis [10].

Metamodelio hierarchija

Metamodelio hierarchija prasideda MOF aukščiausiu abstrakcijos sluoksniu, kiekviename iš sluoksnių yra elementų sluoksnis, kuris priklauso nuo aukščiau esančių elementų [11].

1 lentelė. Modelių hierarchija [11]

M3 (Meta metamodelio sluoksnis) **MOF**	Apibrėžia kalbą metamodeliui nurodyti Pavyzdys: MOF
M2 (Metamodelis) **UML**	Apibrėžia kalbą modeliams nurodyti Pavyzdys: UML
M1 (Modelis) **Naudotojo modeliai**	Apibrėžia kalbą, apibūdinančią srities semantiką Pavyzdys: probleminės srities modelis
M0 (Realizacija) **Veikla**	Sudaro modelio panaudojimą, kuris buvo apibrėžtas modelyje

UML Profiliai

Profilio diagrama yra struktūros diagrama, apibūdinanti lengvą UML išplėtimo mechanizmą, apibrėždama pasirinktinius stereotipus (*angl. Stereotypes*), žymas (*angl. Tags*) ir apribojimus (*angl. Constraints*). Profiliai leidžia pritaikyti UML metamodelį įvairioms platformoms arba veiklos sritims, tokioms kaip verslo procesų modeliavimas, į paslaugas orientuotos architektūros, medicinos sričiai skirtos programos ir kt [1].

Priešingai negu MOF, profilių mechanizmas nėra pirmos rūšies išplėtimo mechanizmas. Jis neleidžia modifikuoti esamų metamodelių ar sukurti naują metamodelį, kaip tokią galimybę galintis suteikti MOF. Profilis leidžia pritaikyti arba praplėsti esamą metamodelį struktūromis, būdingomis konkrečiai sričiai, platformai ar metodui. Profiliuose neįmanoma panaikinti jokių metamodeliui taikomų apribojimų, tačiau galima pridėti naujų, specifiniam profiliui taikytinų apribojimų [1].

Metamodelio modifikacijos, kurios vėliau yra pritaikomos paketui, yra apibrėžtos profilyje. Stereotipai yra specifinės metaklasės, žymos yra standartiniai metaatributai, o profiliai yra konkrečių rūšių paketai. Profiliai gali būti dinamiškai pritaikyti arba pašalinti iš modelio. Juos taip pat galima dinamiškai apjungti, kad keli profiliai tuo pačiu metu būtų taikomi tam pačiam modeliui.

Pagrindiniai UML profilio diagramos elementai [1]:

- Profilis – profilio paketas, praplečiantis pamatinį metamodelį (pvz. UML) leidžiant pritaikyti arba modifikuoti metamodelį papildiniais, kurie yra būdingi konkrečiai sričiai, platformai ar programinės įrangos kūrimo metodui. Kitaip tariant, profilis yra plėtinys skirtas išplėsti UML standartą [11];

- Stereotipas – profilio klasė apibrėžianti, kaip esamą meta-klasę galima išplėsti kaip profilio dalį. Tai leidžia naudoti platformai ar sričiai specifinę terminologiją ar notaciją vietoje naudojamų išplėstajai metaklasei. Stereotipas negali būti naudojamas vienas pats, visada turi būti naudojamas su viena iš jo išplėstų metaklasių, taip pat stereotipo negalima išplėsti kitu stereotipu [1];

- Žyma – vienas iš UML išplėtimo mechanizmų, leidžiantis prie modelių pridėti papildomą informaciją (kurios negalima išreikšti UML). Žymos reikšmė yra raktinių žodžių ir reikšmių pora, kuri gali būti prijungta prie bet kokio modelio elemento [1];
- Metaklasė – profilio klasė kurią galima išplėsti per vieną ar daugiau stereotipų [1];
- Plėtinys (*angl. Extension*) – asociacijos ryšys, naudojamas parodyti, kad metaklasės savybės yra praplečiamos per stereotipą. Šis ryšys suteikia galimybę lanksčiai pridėti stereotipus prie klasių ir esant poreikiui vėliau juos pašalinti [1];
- Profilio taikymas (*angl. Profile Application*) – nukreipiamasis ryšys, naudojamas parodyti kokie profiliai buvo pritaikyti paketui [1].

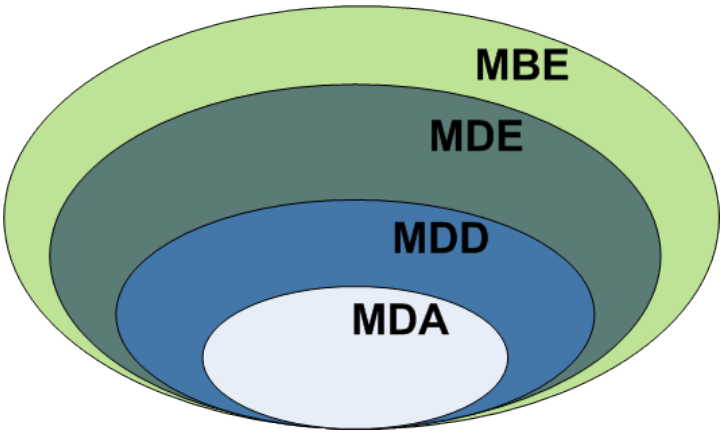
Atlikus metamodelių ir stereotipų analizę, išsiaiškinta, kad abstrakti UML sintaksė nurodoma naudojant UML modelį, vadinamą metamodeliu. Taip pat išsiaiškinta, kad MOF apibrėžia, kaip modeliai yra pasiekiami ir kokie mainai tarp modelių gali vykti, kad šiam tikslui yra naudojama OMG XML meta duomenų apsikeitimo sąsaja. Siekiant išsiaiškinti metamodelio hierarchiją, rasta, kad ji prasideda MOF aukščiausiu abstrakcijos sluoksniu, kiekviename iš sluoksniu yra elementų sluoksnis kuris priklauso nuo aukščiau esančių elementų. Išanalizavus UML profilius nustatyta, kad kuriamai sistemai galima pritaikyti profilių išplėtimo mechanizmą ir sukurti profilį, kuris apimtų specifinius elementus PSM modelyje.

1.3.4. MDA

Prieš nagrinėjant MDA, svarbu išsiaiškinti modeliais grindžiamus principus. Modeliais grindžiama inžinerija (*angl. Model-based Engineering*) yra abstrakcija, kurią sudaro:

- modeliais grindžiama inžinerija (*angl. Model-driven Engineering*);
- modeliais grindžiamas kūrimas (*angl. Model-driven Development*);
- modeliais grindžiama architektūra (*angl. Model-driven Architecture*).

Visi šie principai yra vienas kitą paveldintys, 6 pav. diagramoje parodomas jų ryšys.



6 pav. Ryšys tarp MBE, MDE, MDD, MDA [12]

Pagrindimas modeliais (MBE) yra paprastesnė modeliais grindžiamos inžinerijos (MDE) versija. MBE procese modeliai turi didelę įtaką sistemų kūrimui, tačiau jie nebūtinai yra esminiai artefaktai sistemų kūrimo procese. Pavyzdžiui: sistemų analitikas gali sukurti sistemos modelį, tačiau tuomet šis modelis yra perduodamas programuotojui, kad jis jį realizuotu rankiniu būdu rašydamas visą

programinį kodą (jokio automatinio kodo generavimo iš modelių). Tokiame procese, modeliai yra naudojami labiau tik bendram vaizdui susidaryti. MBE yra MDE abstrakcija [12].

MDE yra MDD abstrakcija, tai nusako MDE esantis žodis inžinerija (abstraktesnis terminas, nei MDD – kūrimo procesas). MDE yra aukščiau grynojo sistemų kūrimo proceso ir apima kitas modeliais grindžiamas užduotis pilname sistemų inžinerijos procese (pvz. modelio evoliuciją į sistemą, arba sistemos transformaciją į modelį) [12].

MDD yra kūrimo paradigma, naudojanti modelius kaip pagrindinį artefaktą kūrimo procese. MDD procese, sistemų realizacija yra pusiau automatiškai sugeneruojama iš modelių [12].

MDA yra specifinė MDD vizija, pasiūlyta OMG kompiuterinių pramonės standartų konsorciumo. Modeliais grindžiama architektūra remiasi OMG sukurtais standartais. Modeliais grindžiama architektūra yra MDD dalis, kur modeliavimo ir transformacijos kalbos yra standartizuotos OMG kompiuterinių pramonės standartų konsorciumo [12]. Šiame tyrime toliau nagrinėjama modeliais grindžiama architektūra.

Modeliais grindžiama architektūra buvo pasiūlyta OMG kompiuterinių pramonės standartų konsorciumo 2001 metais. Šiuo metodu, buvo siekiama kurti programas naudojant modelius vietoj programinio kodo, kur modelis būtų apibrėžiamas kaip supaprastintas sistemos vaizduojančios brėžinius ir tekstus vaizdas. Prieš atsirandant MDA, modeliai buvo laikomi sistemų ar jų dalių abstrakcijomis, kurios buvo naudojamos planuojant ir aiškinant sistemų principus kitiems asmenims. Sistemų modelių kūrimas ir jų įgyvendinimas buvo dvi atskiros dalys, trūko nuoseklumo tarp įvairių programinės įrangos kūrimo aspektų. Tai, kūrė kliūtis siekiant automatizuoti užduotis susijusias su programinės įrangos kūrimu.

Ši problema paskatino MDA požiūrį, kuris pagrindinį dėmesį skyrė veiklų susijusių su projekto gyvavimo ciklu supaprastinimui ir standartizavimui, kad kūrimo procesas galėtų būti tam tikru mastu automatizuotas. Modeliais grindžiama architektūra prisideda prie platformų nepriklausomumo programinės įrangos kūrimo procese. [13]

OMG kompiuterinių pramonės standartų konsorciumas palaiko ir plėtoja modeliais grindžiamą architektūrą, kuri remiasi platformų nepriklausomumo principu. MDA leidžia kurti sistemas modeliais grindžiamu principu ir pagrindinis MDA principas yra tas, kad kūrėjo pagrindinis prioritetas – rūpintis modeliu, o ne programiniu kodu. Modelis yra išsamus bendrinis sisteminis vaizdas, kurį turėtų būti galima vėliau pritaikyti bet kuriai pasirinktai programavimo kalbai. Pats MDA procesas priklauso nuo modelio kūrimo, jei modelis buvo sumodeliuotas teisingai, jį galima be klaidų transformuoti į norimą platformą [9].

MDA transformacijos

Modeliais grindžiamą architektūrą sudaro modelių transformacijos iš CIM į PIM, iš PIM į PSM ir iš PSM į programinį kodą. (žr. 7 pav.). Šių transformacijų modeliai yra CIM, PIM ir PSM.

CIM

Tai verslo modeliai – faktinių žmonių, vietų, daiktų ir įstatymų modeliai. Šių modelių egzemplioriai yra tikri realiai egzistuojantys dalykai, o ne tų dalykų vaizdas informacinėje sistemoje. MDA modeliuose tai istoriškai buvo vadinama nuo skaičiavimo nepriklausomais modeliais (CIM) [14].

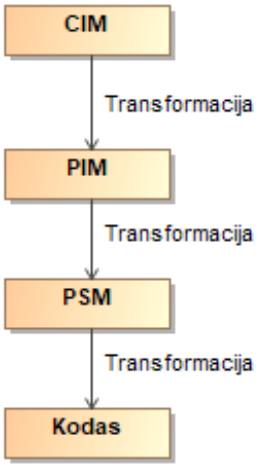
PIM

Tai loginiai sistemos modeliai apibrėžiantys abstraktų sistemos funkcionalumą ir nurodantys kaip šis funkcionalumas siejasi vienas su kitu. Loginių modelių sudarymas gali puikiai padėti organizacijų bendruomenėms siekiant savų tikslų [14].

PSM

Tai modeliai kurie apibrėžia, kaip sistema skirta specifinei platformai turėtų įgyvendinti savo funkcionalumą. Realizavimo platformai pritaikyti modeliai yra susieti su tam tikra įdiegimo aplinka ar technologija [14].

Kadangi šis tyrimas yra orientuotas į transformacijų pritaikymą konkrečiam karkasui, CIM modelis nėra aktualus. Transformacijos realizuojamos iš PIM į PSM ir iš PSM į programinį kodą.



7 pav. MDA transformacijos [15]

Toliau apibrėžiami tyrimui aktualūs transformacijų etapai:

Pirmiausia, suprojektuojamas nuo platformos nepriklausantis modelis (PIM). PIM modelis turi aukštą abstrakcijos lygmenį, jis atskiria logika nuo technologinio įgyvendinimo. Šis modelis suteikia projektuojamai sistemai struktūrą, kuri tiks bet kuriai realizacijos platformai.

Toliau, gaunamas konkrečiai platformai pritaikytas modelis (PSM). Šis modelis yra gaunamas atliekant transformaciją iš PIM į PSM. PSM modelis apibrėžia sistemos funkcionalumo įgyvendinimą konkrečiai platformai.

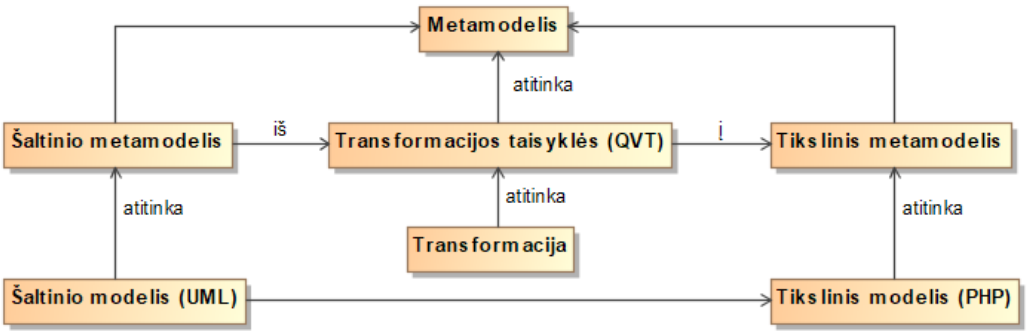
Paskutiniame žingsnyje, atliekant transformaciją iš PSM į kodą, sugeneruojamas specifinės platformos programinis kodas. [9]

Modelių transformacijas sudaro apribojimai ir taisyklės, kurios transformuoja elementus, apibrėžtus transformacijos šaltinio metamodelyje, į kitus reikiamo metamodelio elementus. Standartinė OMG konsorciumo parengta transformacijos kalba MOF QVT (angl. Query / View / Transformation) buvo sukurta apibrėžti metamodelį, leidžiantį įvertinti modelio transformaciją. Trys pagrindiniai aspektai su kiekvieno paaiškinimu pateikiami 2 lentelėje.

2 lentelė. QVT specifikacijos aspektai [16]

Pavadinimas	**Aspektas**
Užklausa *(angl. Query)*	Užklausa yra išraiška, kuri yra vertinama per modelį. Užklausos rezultatas yra vienas ar daugiau apibrėžtų šaltinio modelio tipų arba užklausos kalbos apibrėžimas.
Rodinys *(angl. View)*	Rodinys yra modelis, kuris buvo išgautas iš kito modelio.
Transformacija *(angl. Transformation)*	Modelio transformacija yra automatinis tikslinio modelio generavimas iš šaltinio modelio, atsižvelgiant į transformacijos apibrėžimą.

QVT ryšių kalboje, transformacijos tarp modelių yra nurodomos kaip ryšių visuma. Transformacijos yra pritaikomos šaltinio modeliui, kuris atitinka šaltinio metamodelį generuojant tikslinį modelį kuris atitiks metamodelį [15]. (žr. 8 pav.)



8 pav. Transformacijos iš modelio į programinį kodą MDA principu procesas [15]

Transformacijas iš PIM į PSM ir iš PSM į programinį kodą galima atlikti su įvairiais šį funkcionalumą palaikančiais įrankiais. Įrankiai skirti modelių transformacijai iš PIM į PSM, kurie palaiko QVT notaciją yra Merlin, JQVT, JTL, PTL, BOTL ir kt. Rinkoje esantys įrankiai atitinkantys MOFM2T standartą ir skirti transformacijai iš PSM į programinį kodą yra: Acceleo, MOFScript, ModelAnt ir kt [17]. Eclipse ATL ir Eclipse Acceleo įrankiai pasirinkti kaip toliau tyrime aktualūs įrankiai, dėl galimybės šiuos įrankius naudoti vienoje platformoje – Eclipse. Taip pat šie įrankiai yra atvirojo kodo sprendimai ir Acceleo įrankis palaiko kodo generavimą į PHP programinį kodą.

ATLAS Transfomation Language

Transformacija iš PIM į PSM (M2M transformacija) galima atlikti naudojant Eclipse ATL įrankį ir transformacijos taisyklėmis grįstą algoritmą kuris turėtų būti apibrėžtas ATL transformacijų kalba, kuri atitinka MOF QVT standartą. ATL transformacijų kalba leidžia apibrėžti 3 rūšių transformacijos dalis: ATL transformacijų modulius, ATL užklausas ir ATL bibliotekas. Be viso to, šias rūšis papildyti ir sudaryti gali ATL pagalbinės funkcijos, atributai, atitikties ir iškviečiamosios taisyklės. Pats ATL įrankis suteikia galimybę transformuoti norimą kiekį modelių iš norimo kiekio modelių apibrėžiant transformacijos taisyklėmis grįstą algoritmą [18].

Acceleo

Transformacija iš PSM į programinį kodą (M2T transformacija) galima atlikti naudojant Eclipse Acceleo įrankį ir šablonu grįsta Acceleo kalbą. Įrankis yra atvirojo kodo generatorius atitinkantis OMG MOF Model to Text Language standartą ir leidžiantis naudoti modelius, siekiant sugeneruoti

programinį kodą [19]. Pats įrankis yra grįstas Java programavimo kalba ir turi galimybe veikti, kaip atskiras kodo generatorius.

Atlikus pagrindinių modeliais grindžiamų principų, MDA, modeliais grindžiamos architektūros transformacijų modelių ir etapų analizę, buvo išsiaiškinta, kad pagrindiniai modeliais grindžiami principai yra pagrindimas modeliais kurį sudaro modeliais grindžiama inžinerija, modeliais grindžiamas kūrimas ir modeliais grindžiama architektūra. Nustatyta, kad MDA leidžia kurti sistemas modeliais grindžiamu principu ir pagrindinis MDA principas yra tas, kad kūrėjas pagrinde rūpinasi modeliu, o ne programiniu kodu. Rasta, kad modeliais grindžiama architektūrą sudaro modelių transformacijos iš CIM į PIM, iš PIM į PSM ir iš PSM į programinį kodą. Taip pat išsiaiškinta, kad M2M transformacijas galima atlikti naudojant ATL transformacijų įrankį, o MT2 transformacijas galima atlikti naudojant Acceleo transformacijų įrankį. Išanalizuota informacija yra aktuali tyrimui, siekiant pritaikyti pagrindinius modeliais grindžiamos architektūros principus kuriant sprendimo realizaciją.

1.3.5. PHP programavimo kalba ir jos karkasai

PHP programavimo kalba yra laikoma viena iš plačiausiai naudojamų programavimo kalbų internetinių sistemų kūrime, nes ji suteikia didelį lankstumą, yra lengvai naudojama ir lengvai perprantama. Ji turi intuityvias savybes, atvirą kodą, tarp-platforminį suderinamumą ir SQL palaikymą. Taip pat, ši programavimo kalba turi sąsaja su UML įrankiais, suteikiančiais galimybę sugeneruoti PHP kalbos programinį kodą iš UML modelio. Pagal Ozkaya, M. autoriaus atliktą tyrimą [6], išskirti UML įrankiai kurie palaiko kodo generavimo galimybes PHP programavimo kalbai: ArgoUML, Astah, Bouml, Enterprise Architect, SoftwareIdeas, Umbrello, Visual Paradigm, UMLStudio, Umple. Su šiais įrankiais galima generuoti šios kalbos programinį kodą.

Atliktoje programavimo kalbų (PHP, Perl, VB.NET, Ruby, AspectJ, Haskell, C++, Scala, Java, Scheme) lyginamojoje analizėje kurioje vertinami pagrindiniai programų kūrimo veiksniai Ruby ir PHP išrinktos kaip lyderės internetinių sistemų kūrime [4].

PHP yra serverinės dalies programavimo kalba, naudojama kurti dinamiškas ir interaktyvias internetines sistemas. Kuriant sistemas su PHP programavimo kalba, verslo logika yra sumaišoma su duomenų bazių užklausomis, dėl šio sumaišymo sistemų priežiūra ir palaikymas tampa sudėtingas. Sprendžiant šią sudėtingumo problemą, yra sukurti įvairūs PHP karkasai.

PHP karkasai padeda kūrėjams greičiau ir paprasčiau kurti internetines sistemas, suteikdami pagrindinių modelių struktūrą, paprastą API pasiekiamumą, bibliotekas ir papildinius. Taip pat karkasai padeda kūrėjams tapti produktyvesniems, sumažinant pasikartojantį kodą vykdomame projekte [20].

Šie karkasai yra grindžiami MVC architektūros principu. Šis principas yra efektyvus ir patikrintas būdas kuriant modulines, struktūrizuotas sistemas. MVC architektūra yra dalinama į atskiras dalis, kurios gali būti atskirai analizuojamos ir įgyvendinamos [21]:

- vaizdai (angl. Views) – atvaizdavimui skirta dalis;
- valdikliai (angl. Controllers) – dalis skirta naudotojo užklausų apdorojimui ir veiksmų siuntimui į vaizdų ir modeliavimo dalis;
- modeliai (angl. Models) – skirti sistemos duomenims ir su jais susijusiomis operacijomis.

Atskirdama modelius ir vaizdus MVC architektūra padeda sumažinti architektūrinio projektavimo sudėtingumą, padidiną programinio kodo lankstumą bei pakartotinį panaudojimą [21]. Pastaruoju metu rinkoje yra didelis PHP programavimo kalbos karkasų pasirinkimas, tokių kaip: Symfony, CodeIgniter, Laravel, CakePHP. Nors šie karkasai turi daugybę privalumų, sistemų kūrėjai turi patys nuspręsti, kuris karkasas yra jiems tinkamiausias tam tikrų sistemų kūrime [4]. Khaoula ir kt. autoriai [20] tyrime išanalizavo ir palygino rinkoje esančius populiariausius karkasus. 3 lentelėje pateikiamas 3 populiariausių naudojamų karkasų palyginimo pagal atliktą tyrimą galutinis apibendrinimas pagal keturias pagrindines kategorijas:

- Pastovumas – karkasų branda, stabilumo rodikliai, istorija, žinomos problemos;
- Sprendimo sėkmė – atsižvelgiama į sprendimo palaikymą, dokumentacijas, papildoma pagalbą naudotojams;
- Pritaikomumas – atsižvelgiama į techninį pritaikomumą, atitikimą PHP kalbos versijai, vienetų testavimo galimybės, integracijos su kitomis platformomis, kodo generavimo galimybės ir kt.;
- Strategija – atsižvelgiama į karkaso licencijos taisykles, teisių turėtojus, vystomo planus, strateginę nepriklausomybę.

Rezultatai buvo išreikšti procentine forma nuo autorių tyrime gautų balų.

3 lentelė. PHP karkasų palyginimas [20]

Lyginimo kriterijai	**Laravel**	**Symfony**	**CodeIgniter**
Pastovumas	83,33%	83,33%	76,66%
Sprendimo sėkmė	73,68%	73,68%	63,15%
Pritaikomumas	100%	100%	100%
Strategija	70%	70%	60%

Khaoula ir kt. autorių tyrime [20] ir Laaziria ir kt. autorių tyrime [4] ištirta, kad Laravel ir Symfony karkasai atitinkastandartus ir reikalavimus pagal autorių nurodytus lyginimo kriterijus. Jie yra paklausūs ir pasižymi aktualiomis savybėmis. Lyginant CodeIgniter karkasą su šiais – pagal autorių tyrimą jis ne taip atitinka standartus ir pasižymi prastesniais rezultatais pagal palyginimo kriterijus.

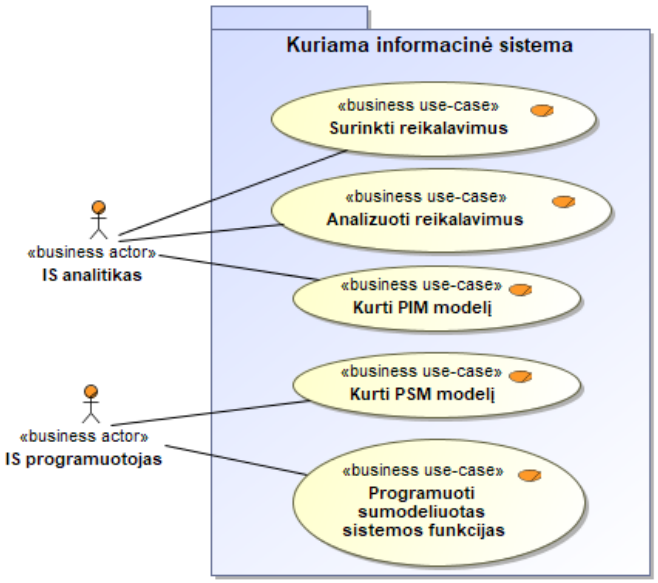
Kitame Laaziria ir kt. autorių tyrime [21] teigiama, kad Laravel, ir Symfony karkasai yra lygiaverčiai. Symfony karkasas statistiškai yra labiau naudojamas didesnio dydžio projektuose, bet yra kompleksiškesnis. Laravel karkasas yra naudojamas nuo mažos iki didelės apimties sistemų kūrimui ir yra lengviau suprantamas pradedantiesiems.

Šiame tyrime pasirinkta naudoti PHP Laravel karkasą. Pagal atliktus tyrimus [4] [20] [21] šis karkasas yra tinkamas sparčiam nuo mažos iki didelės apimties sistemų kūrimui, net turint sąlyginai mažai patirties. Pradedantiesiems yra paprasčiau pradėti dirbti su Laravel, nei su Symfony, kadangi Symfony reikalauja patirties kuriant kompleksines sistemas su šiuo karkasu. CodeIgniter pasirinkimas nebuvo svarstomas, kadangi nevisiškai atitinka standartus, pasižymi prastesniais rezultatais pagal palyginimo kriterijus ir pagrinde yra tinkamas mažos apimties trumpiems projektams. Laravel karkasas padeda sistemų kūrėjams supaprastinti sistemų kūrimo procesą, suteikdamas galimybę rašyti švarų ir pernaudojamą kodą, taip pat supaprastinti autentifikacijos, sistemos maršrutų sudarymo, sesijų valdymo veiksnius. Šis karkasas remiasi MVC architektūros principu, suteikdamas galimybę logiškai skirstyti kuriamos sistemos logiką. Taip pat šis karkasas turi

savo Eloquent ORM įrankį, skirta optimizuoti sudėtingas užklausas į duomenų bazę, suteikdamas galimybę jas aprašyti įprasta PHP sintakse [22].

1.4. Tyrimo objekto naudotojų analizė

Šiame darbe atliekamo tyrimo objekto naudotojai yra asmenys, dalyvaujantys informacinių sistemų kūrimo procese. (9 pav.) pateikiami suinteresuoti asmenys ir jų pagrindinės abstrakčios funkcijos kuriant informacines sistemas remiantis modeliais grindžiamos architektūros principu. Išskiriami dveji suinteresuoti asmenys: IS analitikas ir IS programuotojas. IS analitikas yra atsakingas už reikalavimų surinkimą, jų analizę, bei abstrakčių nuo platformos nepriklausančių modelių kūrimą. Programuotojo atsakomybė yra tolimesnis IS projektavimas ir sistemos realizavimas.



9 pav. Tyrimo objekto naudotojai

Informacinės sistemos kūrimui naudojant PHP programavimo kalbą ir karkasą, MDA architektūrai sukurta ir pritaikyta metodika apimanti transformacijas tarp modelių ir į programinį kodą bei įrankis realizuojantis sukurtą metodiką padėtų suinteresuotiems asmenims kūrimo procese.

IS analitikas galėtų suprojektavęs nuo realizavimo platformos nepriklausomą modelį jį transformuoti į realizavimo platformai pritaikytą modelį. Tuomet, IS programuotojas apibrėžęs sistemos funkcijų įgyvendinimą konkrečiai platformai galėtų sugeneruoti PHP programinį kodą su paruošta struktūra.

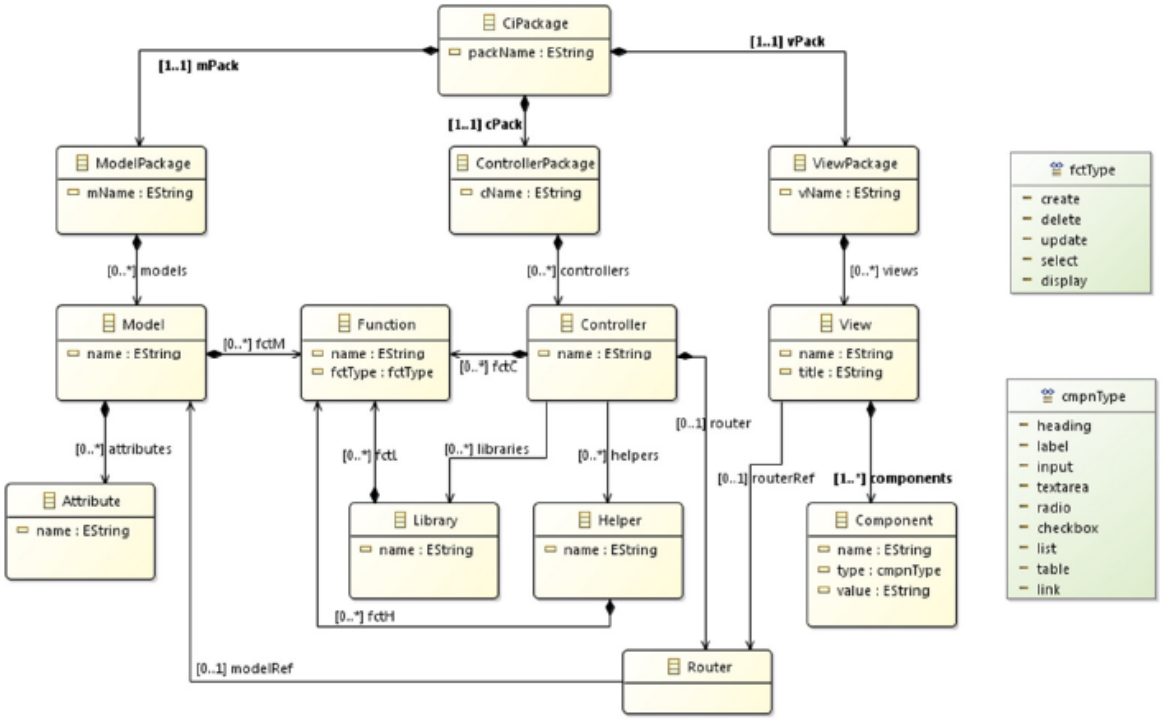
Atsižvelgiant į tai, kad organizacijos, kurios kuria informacines sistemas, siekia taikyti MDA principus, tačiau nėra sukurtų pakankamai išsamių metodikų ir jas apimančių algoritmų, padedančių sugeneruoti PHP programinį kodą informacinėms sistemoms, galima teigti, kad metodikos ir įrankio sukūrimas, padėtų organizacijų suinteresuotiems asmenims taikyti MDA principus, kuriant informacines sistemas, kurios bus realizuojamos konkrečiam PHP programavimo kalbos karkasui.

1.5. Esamų problemos sprendimo metodų analizė

Per keletą pastarųjų metų buvo atlikta nemažai tyrimų, siekiant iš modelio sugeneruoti programinį kodą. Šiame tyrime, pasirinkta analizuoti tyrimus, kuriuose pateikiamas kodo generavimas iš modelio į karkasams skirtą MVC struktūros programinį kodą.

1.5.1. Metodika PHP programinio kodo generavimui CodeIgniter karkasui

Arrhioui ir kt. autorių tyrime [2], kuris buvo atliktas 2019 metais, autoriai nagrinėjo modeliais grindžiamos architektūros požiūrį ir modeliavo bei generavo PHP CodeIgniter karkasu grįstą sistemą. Tyrimo autoriai pasiūlė metodiką, kuri leidžia modeliuoti internetines sistemas grįstas PHP CodeIgniter karkasu. Autoriai kurdami metodiką, taikė modeliais grindžiamos architektūros sistemų kūrimo požiūrį. Atliko sukurto modelio modelių transformacijas iš PIM į PSM naudodami UML klasių diagramą kaip šaltinio modelį, kad sugeneruotų XML failą su esminiais CodeIgniter karkaso komponentais. Modelių transformacijų naudojimas suteikė privalumų pagerinant sistemų kūrimo proceso kokybe ir tuo pačiu sumažinant sąnaudų ir laiko kaštus. Šis požiūris pademonstravo efektyvumą įgalindamas galimybę sukurti pilną CodeIgniter MVC architektūros karkasą apibrėžiantį metamodelį ir pagal šį metamodelį sugeneruoti programinį kodą. Autorių sukurtas CodeIgniter karkasui skirtas metamodelis pateikiamas 10 pav. modelis išskirstytas į modelių, vaizdų ir valdiklių paketus. Kiekvienas paketas turi konkrečias meta klases pagal MVC architektūrą.



10 pav. CodeIgniter metamodelis [2]

- CiPackage: išreiškia paketo konceptą kuris apima visus modelio elementus – modelį, valdiklį ir vaizdą. Jie atitinkamai yra sugrupuojami į ModelPackage, ControllerPackage ir ViewPackage;
- Helper ir Library yra funkcijų rinkiniai, kurie gali būti iškviečiami iš valdiklio, vaizdo arba modelio;
- View: sudaro komponentus kurie yra apibrėžiami Component;
- Controller: tarpininkas tarp modelio ir vaizdo. Jo turinys – sistemos funkcijos;
- Component: sudaro grafiniai elementai, tokie kaip įvestis, tekstai ir antraštės.

View / Controller dalys yra atsakingos už vaizdų struktūros ir turinio aprašymą, o navigacijos srautas užtikrinamas naudojant specifines valdiklio funkcijas, prijungtas prie nurodytų dalių iš modelio dalies. Ši dalis surenka visas sistemos verslo logikos dalis.

Kai, autoriai baigė modeliuoti sistemą, kodo generavimo procedūrą vykdė pagal iš modelio į tekstą transformaciją (M2T). Norėdami sugeneruoti CodeIgniter programinį kodą, autoriai naudojo Acceleo programinę įrangą, kuri transformacijas įgyvendina pagal šablono metodą. Transformacija buvo vykdoma pagal CodeIgniter metamodelio šabloną.

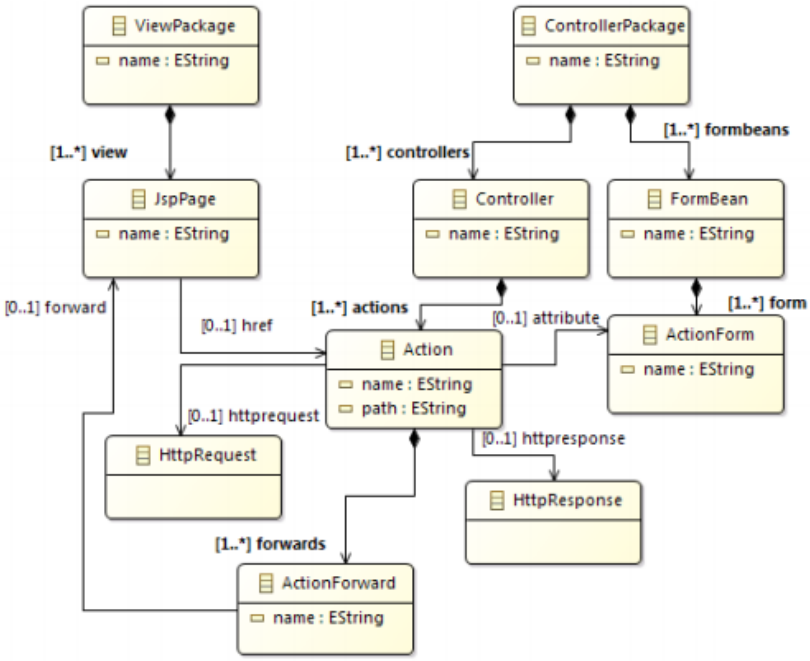
Arrhioui ir kt. autorių tyrimo [2] metodas atlieka kodo generavimą, bet metodas yra pritaikytas PHP kalbos CodeIgniter karkasui ir tai yra atliekama su viena iš struktūrinių diagramų – klasių diagrama, elgsenos diagramos nėra naudojamos. Šiame darbe aprašomas sprendimas, kuris suteikia galimybę generuoti programinį kodą ne tik iš struktūrinių, tačiau ir iš elgsenos diagramų ir yra pritaikytas PHP kalbos Laravel karkasui.

1.5.2. Metodika Java programinio kodo generavimui Spring MVC karkasui

Šiame Srai ir kt. autorių darbe [5] buvo pritaikyti modeliais grindžiamos architektūros principai kuriant Spring MVC internetinę sistemą naudojantis UML klasių diagrama.

Pirmiausia buvo apibrėžtas ir sukurtas metamodelis kuris atitiko Spring MVC šabloną.

Tikslinį metamodelį sudarė dvi esminės dalys: pirmoji dalis nurodo vaizdų paketą, šį paketą sudaro daugybė JSP puslapių. Antroji dalis sudaro valdiklių paketą, paketas susideda iš daugybės valdiklių, kiekviename valdiklyje yra vienas ar daugiau veiksmų ir kiekvienoje formoje yra viena ar daugiau veiksmų formų. Žemiau pateiktas 11 pav. pavaizduoja Spring MVC metamodelį.



11 pav. Spring MVC meta modelis [5]

Autoriai apibrėžė transformacijos taisykles ir iš modelio į modelį ir iš modelio į programinį kodą. Pagal šias taisykles, galima sukurti XML failą, kuriame yra visi veiksmai, formos ir JSP puslapiai, kurie gali būti panaudoti generuojant sistemos programinį kodą. Autoriai sukūrė transformacijos algoritmą naudodamiesi ATL transformacijų kalba. Ši kalba yra Eclipse M2M dalis.

Toliau, buvo sukurta M2T transformacija, norint sugeneruoti Spring MVC programinį kodą. M2T transformacijai buvo apibrėžtas Spring MVC šablonas programiniam kodui naudojant OMG

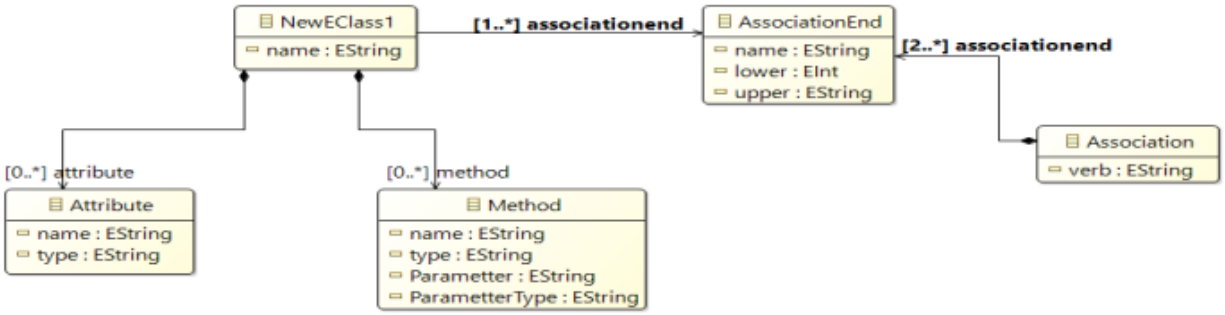
standartą Acceleo. Įvykdžius sukurtą šabloną, gaunamas trijų dalių programinis kodas: su vaizdais, modeliais ir valdikliais.

Nors Srai ir kt. autorių darbe [5] metodas atlieka kodo generavimą, tačiau tai yra atliekama tik su klasių diagrama, elgsenos diagramos nebuvo naudotos, taip pat metodas pritaikytas Java kalbos Spring MVC karkasui.

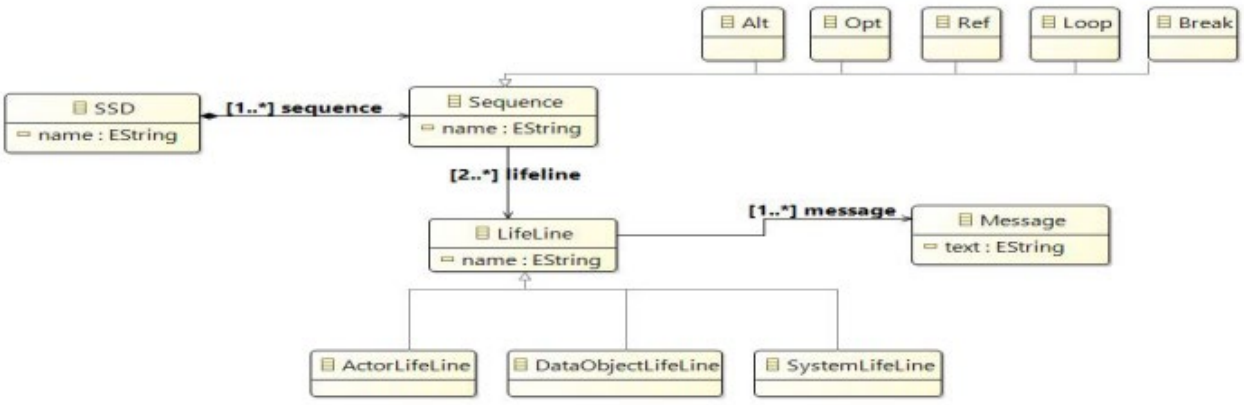
1.5.3. MoDAr-WA įrankis Java MVC programų programinio kodo generavimui

Essebaa ir kt. autoriai šiame darbe [23], pristatė įrankį palaikanti modeliais grindžiamą architektūrą (MoDAr-WA), šis įrankis įgyvendina autorių siūlomą metodiką, kurios tikslas – automatizuoti transformacijas iš aukščiausio MDA lygmens (CIM) iki žemiausio (programinis kodas) ir užtikrinti transformacijų atsekamumą. Šis autorių darbas yra jų buvusio darbo tęsinys automatizuojant transformacijas iš CIM į PIM. Šiame darbe autoriai sukūrė metamodelių rinkinius klasių ir sekų diagramoms, QVT ir Acceleo transformacijas, taip pat įrankius sukurti Eclipse papildiniui MoDAr-WA.

Visų pirma, buvo naudojamos QVT taisyklės transformacijoms tarp modelių (iš CIM į PIM, iš PIM į PSM). PIM lygmenyje, autoriai apibrėžė pagrindinius struktūrinius ir elgsenos aspektus klasių ir sekų diagramų metamodeliuose. Struktūrinis aspektas apibrėžė sąsajas tarp elementų kurie atvaizduoja sistemą. Tai buvo apibrėžta naudojantis klasių diagrama. 12 pav. pateiktas autorių požiūrio klasių diagramos metamodelis. Elgsenos aspektas apibūdino veiksmų srautą tarp sistemos elementų. 13 pav. pateiktas autorių požiūrio sekų diagramos metamodelis.



12 pav. MoDAr-WA įrankio klasių diagramos metamodelis [23]



13 pav. MoDAr-WA įrankio sekų diagramos metamodelis [23]

Toliau programinis kodas iš PSM buvo sugeneruotas su Acceleo pagalba ir galiausiai sukurtas įrankis buvo pritaikytas muzikos parduotuvės sistemos kūrime.

Pateiktame Essebaa ir kt. autorių darbe [23] buvo pristatytas metodas atliekantis kodo generavimą, nepaisant to, tai yra atliekama su klasių diagrama ir viena iš elgsenos diagramų – sekų diagrama, taip pat metodas ir realizuotas įrankis pritaikytas Java programavimo kalbai, MVC architektūra grįstai programai.

1.5.4. Metodika Java MVC sistemų programinio kodo generavimui

Šiame Paolone ir kt. autorių darbe [24] buvo pasiūlas MDA grįstas metodas, kuris turėtų supaprastinti sistemų kūrimą modelių lygmenyje. Naudojant autorių sukurtą įrankį, galima sugeneruoti MVC struktūros programinį kodą internetinėms sistemoms, kurios yra kuriamos naudojant JAVA programavimo kalbą. Autorių siūlomame sistemų kūrimo procese, analitikai ir dizaineriai turėtų būti atsakingi už CIM sukūrimą, o PIM ir PSM kodą sugeneruotų jų įrankis – *xGenerator*. CIM lygmenyje turėtų būti apibrėžiamos veiklos procesų sekų diagramos, aktoriai, objektai ir jų sąryšiai. Toliau, pagal metamodelius ir transformacijos taisykles vykdoma transformaciją į PIM ir iš PIM į PSM. Paolone ir kt. autorių teigimu, jų sukurtas įrankis, turėtų panaikinti atotrūkį tarp modelio ir programinio kodo, leidžiant atnaujinti modelį neprarandant programinio kodo. Jų metodikoje, pagrindinės transformacijos tarp MDA lygių yra akcentuojamos į panaudojimo atvejų ir klasių metamodelius. Metamodeliai ir transformacijų algoritmai yra pačiame įrankyje, taigi analitikams ir dizaineriams nereikia rūpintis, kaip vyksta kodo generavimas.

Paolone ir kt. autorių tyrimo [24] metu sukurtas įrankis atlieka kodo generavimą, įrankyje yra realizuotos transformacijos apimančios visus lygmenis, pradedant transformacija iš CIM į PIM ir baigiant transformacija iš PSM į programinį kodą, bet metodas yra pritaikytas JAVA programavimo kalbos MVC architektūros sistemoms.

1.5.5. Apibendrinimas

Išanalizuoti tyrimai siekia taikyti MDA principus sistemų kūrimo procese. Visuose tyrimuose autorių pateikiami sprendimai apima formaliai aprašytą generavimo algoritmą ir šių aprašytų sprendimų realizaciją autorių pasirinktuose įrankiuose.

Autorių atlikti tyrimai teoriškai apibrėžia ir praktiškai realizuoja kodo generavimą iš klasių diagramos, du iš keturių tyrimų teoriškai apibrėžia ir praktiškai realizuoja kodo generavimą iš sekų diagramos ir nei vienas iš tyrimų teoriškai neapibrėžia ir praktiškai nerealizuoja transformacijų iš veiklos ir būsenų diagramų. Visi tyrimai skirti transformuoti modelius ir sugeneruoti programinį kodą MVC architektūrą grįstiems karkasams, trys tyrimai skirti Java programavimo kalbos karkasams, o vienas PHP programavimo kalbos karkasui. Dviejuose tyrimuose konkretizuoti įrankiai naudoti transformacijai iš PIM į PSM, viename tyrime naudota QVT transformacijų kalba, kitame – Eclipse ATL. Trijuose iš keturių tyrimų konkretizuotos transformacijos naudotos aplinkos, tai yra Eclipse aplinka, šie tyrimai transformacijai iš PSM į programinį kodą naudojo Eclipse Acceleo įrankį. Visuose keturiuose tyrimuose realizuotos transformacijos tiek iš PIM į PSM, tiek iš PSM į programinį kodą.

4 lentelė. Esamų problemos sprendimo metodų palyginimas

Lyginimo kriterijai	**A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications**	**Applying MDA approach for Spring MVC Framework**	**MoDAr-WA: Tool Support to Automate an MDA Approach for MVC Web Application**	**Automatic Code Generation of MVC Web Applications**
Sprendimas apima formaliai aprašytą generavimo algoritmą	+	+	+	+-
Generavimo algoritmas realizuotas įrankyje	+	+	+	+
Kodo generavimas iš klasių diagramos	+	+	+	+
Kodo generavimas iš veiklos diagramos	-	-	-	-
Kodo generavimas iš būsenų diagramos	-	-	-	-
Kodo generavimas iš sekų diagramos	-	-	+	+
Kodo generavimas architektūros principui	MVC	MVC	MVC	MVC
Kodo generavimas programavimo kalbai	PHP	Java	Java	Java
Kodo generavimas programavimo kalbos karkasui	CodeIgniter	Spring MVC	J2EE MVC	J2EE MVC
Transformacijų kalba	Neapibrėžta	ATL	QVT	Neapibrėžta
Naudoti įrankiai	Eclipse / Acceleo	Eclipse / Acceleo	Eclipse / Acceleo	Neapibrėžta
Realizuota transformacija iš PIM į PSM	+	+	+	+
Realizuota transformacija iš PSM į programinį kodą	+	+	+	+

Šiame tyrime taip pat taikomi MDA principai, analizuojamos ir formaliai apibrėžiamos modelių transformacijos ir kodo generavimas, tačiau tai atliekama PHP Laravel karkaso kontekste, kas nėra

išanalizuota ir realizuota kituose autorių tyrimuose. Taip pat šiame tyrime analizuojamas kodo generavimas ne tik iš struktūrinio tipo diagramų, tačiau ir iš elgsenos – sekų diagramos, kituose išanalizuotuose autorių tyrimuose dažniau programinis kodas generuojamas iš struktūrinio tipo diagramų, nei iš elgsenos tipo diagramų.

1.6. Siekiamo sprendimo apibrėžimas

Siekiamas sprendimas – MDA principais grindžiama ir Laravel PHP kalbos karkasui pritaikyta metodika. Ši metodika apima PIM ir PSM struktūros specifikacijas ir išsamius reikalavimus sudarant PIM modelio elgsenos diagramą. Metodikoje apibrėžiamas ir sudaromas PHP Laravel karkasui pritaikytas UML profilis skirtas PSM. Taip pat metodika detaliai apibrėžia algoritmus transformacijoms iš PIM į PSM ir iš PSM į programinį kodą.

1.7. Analizės išvados

Probleminės srities analizėje buvo išanalizuotos MDA architektūros kontekste atliekamos transformacijos iš modelio į modelį ir iš modelio į programos kodą:

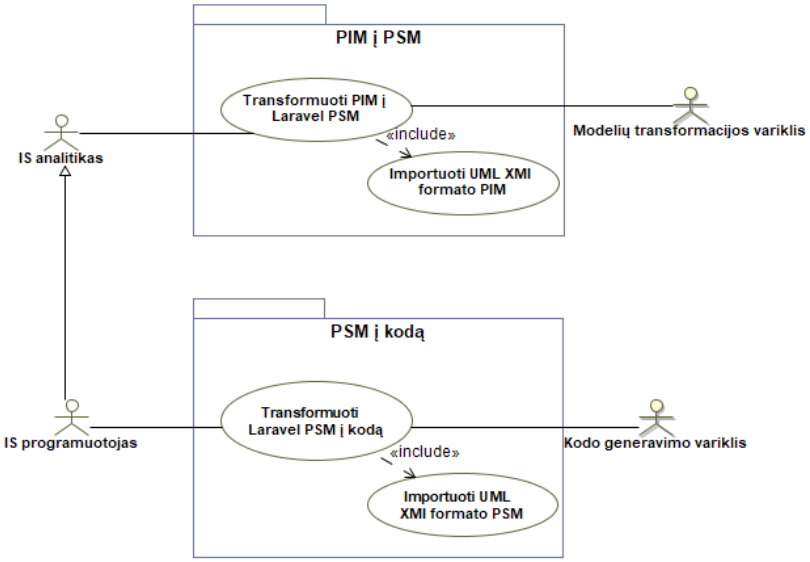
1. Išanalizavus UML kalbą ir jos modelių transformacijas MDA nustatyta, kad UML paskirtis yra patobulinti technologijų industriją įgalinant vizualinį modeliavimą suteikiant tam reikalingus įrankius, o UML modeliais grindžiamas sistemų kūrimas suteikia galimybę naudoti skirtingas realizavimo platformas ir dalinai automatizuoja kūrimo procesą.
2. Atlikus egzistuojančių modelių transformacijų MDA kontekste, jų įrankių ir egzistuojančių kodo generavimo iš PSM sprendimų analizę, nustatyta, kad egzistuoja pritaikytų sprendimų PHP kalbos kitiems karkasams (pvz. CodeIgniter), Java kalbai ir jos karkasams (pvz. Spring MVC), didžioji dalis šių sprendimų dažniau generuoja kodą iš UML struktūrinio tipo diagramų, rečiau – iš UML elgsenos diagramų.
3. Išanalizavus tris populiariausius PHP kalbos karkasus, pasirinkta naudoti Laravel karkasą dėl naudojamos klasikinės MVC architektūros, tinkamumo kuriant skirtingos apimties sistemas ir techninio karkaso stabilumo. Toliau analizuojant UML sąsajas su PHP programavimo kalba ir Laravel karkasu MDA architektūros kontekste, nerasta egzistuojančių MDA transformacijų ir kodo generavimo sprendimų PHP kalbos Laravel karkasui, tačiau nustatyta, kad PHP kodo generavimui galima sėkmingai pritaikyti MDA transformacijų ir kodo generavimo principus.
4. Tyrime pasirinkus plačiau analizuoti klasių, sekų, būsenų ir veiklos diagramas išsiaiškinta, kokie yra šių diagramų metamodelių pagrindiniai metaklasių struktūros fragmentai, kad MOF yra naudojamas kaip metamodelio metamodelis ir apibrėžia kaip modeliai yra pasiekiami ir kokie mainai tarp modelių gali vykti. Profiliai leidžia pritaikyti arba praplėsti esamą metamodelį su struktūromis būdingomis konkrečiai sričiai, platformai ar metodui, taigi norint realizuoti transformacijas PHP kalbos Laravel karkasui, reikia sukurti profilį, kuris apimtų specifinius šio karkaso elementus PSM modelyje.

2. Sprendimo reikalavimų specifikacija ir projektas, formalus aprašas

Šiame sprendimo reikalavimų specifikacijos ir projekto formaliame apraše apibrėžiama metodika, pateikiami algoritmų aprašai, aprašomi aktualūs klasių ir sekų PIM ir PSM metamodelių fragmentai. Taip pat pateikiami pavyzdiniai modelių ir kodo fragmentai.

2.1. Reikalavimų specifikacija

Detaliai apibrėžiant siūlomą metodiką, 14 pav. pateikiama UML panaudojimo atvejų diagrama, kurioje pateikiami galimi IS analitiko ir IS programuotojo veiksmai transformuojant modelius ir transformacijų variklių sąsajos su šiais veiksmais.



14 pav. Generavimo procesą apibrėžianti panaudojimo atveju diagrama

Pateiktame pavyzdyje, IS analitikas ir IS programuotojas gali importuoti modelius į transformacijų variklius ir atlikti transformaciją iš PIM į PSM, detalesnė šios transformacijos veiklos diagrama pateikiama 16 pav. IS programuotojas taip pat gali atlikti transformaciją iš PSM į programinį kodą, tai detalizuojama 18 pav.

Pritaikant metodiką svarbu, kad, naudojantis pasirinktu UML CASE įrankiu, turėtų būti sumodeliuotas PIM. Sudarant PIM, būtina tinkamai sumodeliuoti sekų diagramas.

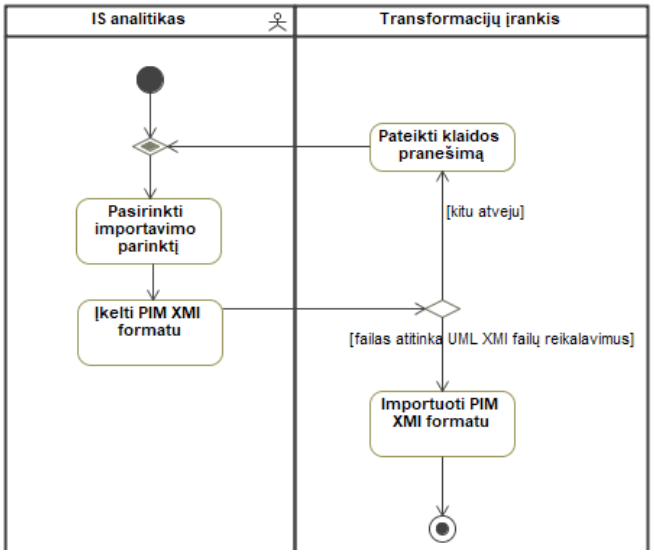
Sekų diagramų reikalavimai:

- sekų diagramose turi būti naudojamos klasės tik iš klasių diagramos;
- žinučių operacijos tarp gyvavimo linijų turi būti naudojamos tik tos, kurios yra nurodytos klasėse;
- sekų diagrama turi turėti aktoriaus, klasės elementus ir *Message* žinučių tipus;
- jei sekų diagrama skirta aprašyti naujo lango atidarymą, tuomet sekų diagramą turi sudaryti šie elementai (nurodoma žinučių siuntimo eilės tvarka): Actor, boundary, control, boundary. Sudarytos sekų diagramos pavyzdys pateikiamas 32 pav.;
- jei sekų diagrama skirta aprašyti naujo lango, skirto redagavimui atidarymą, tuomet sekų diagramą turi sudaryti šie elementai (nurodoma žinučių siuntimo eilės tvarka): Actor,

boundary, control, boundary ir žinutės tarp gyvavimo linijų turi turėti kintamąjį. Sudarytos sekų diagramos pavyzdys pateikiamas 34 pav.;

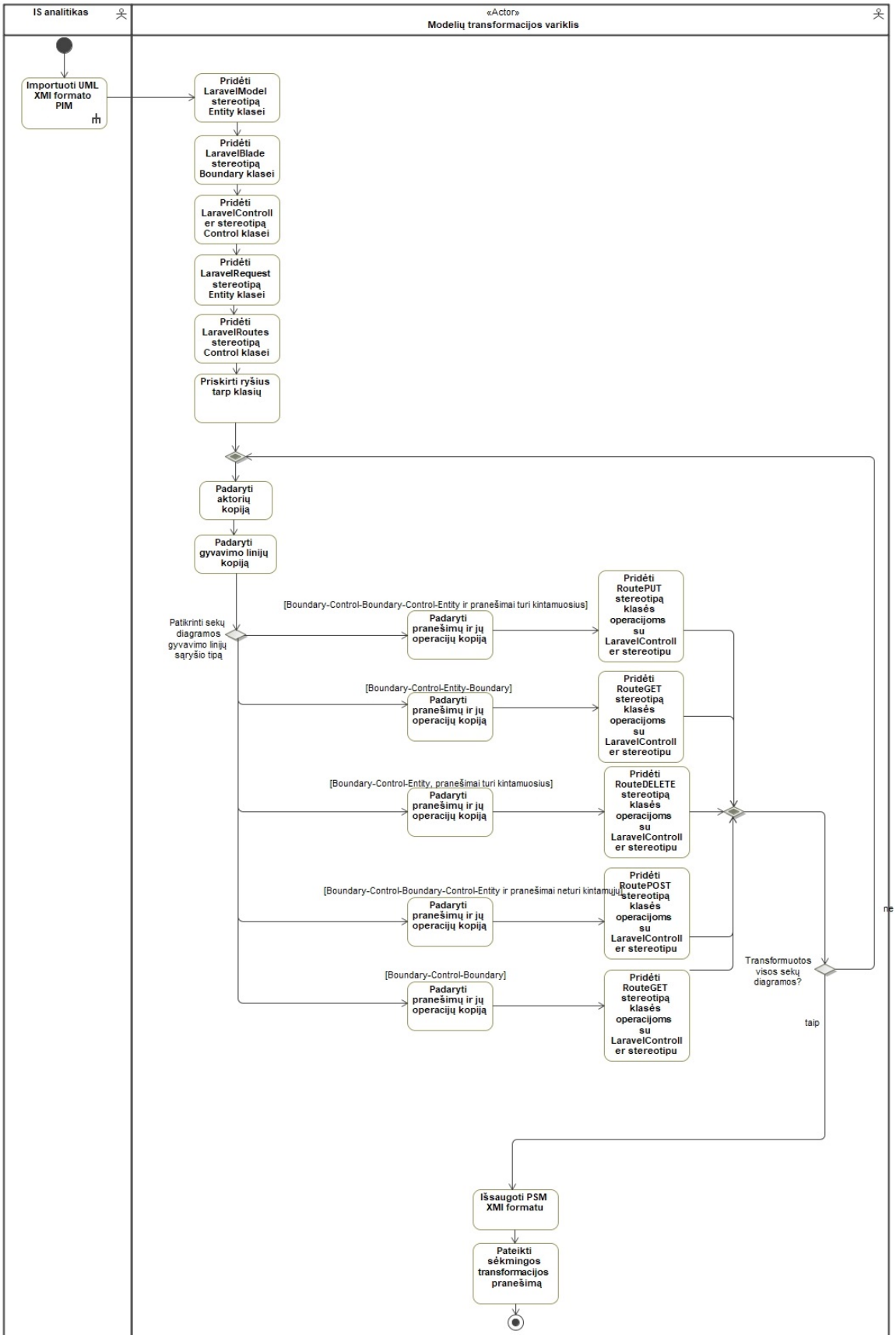
- jei sekų diagrama skirta atvaizduoti veiksmą, skirtą specifinio elemento šalinimui, tuomet sekų diagramą turi sudaryti šie elementai (nurodoma žinučių siuntimo eilės tvarka): Actor, boundary, control, entity ir žinutės tarp gyvavimo linijų turi turėti kintamąjį. Sudarytos sekų diagramos pavyzdys pateikiamas 36 pav.;
- jei sekų diagrama skirta atvaizduoti veiksmą, skirtą elemento sukūrimui, tuomet sekų diagramą turi sudaryti šie elementai (nurodoma žinučių siuntimo eilės tvarka): Actor, boundary, control, boundary, control, entity. Sudarytos sekų diagramos pavyzdys pateikiamas 38 pav.;
- jei sekų diagrama skirta atvaizduoti veiksmą, skirtą specifinio elemento informacijos atnaujinimui, tuomet sekų diagramą turi sudaryti šie elementai (nurodoma žinučių siuntimo eilės tvarka): Actor, boundary, control, boundary, control, entity ir žinutės tarp gyvavimo linijų turi turėti kintamąjį. Sudarytos sekų diagramos pavyzdys pateikiamas 40 pav.

Sumodeliavus PIM, modelis XMI formatu išeksportuojamas į pasirinktą M2M transformacijų įrankį. Toliau, PIM transformuojamas į PSM naudojant modelių transformacijos variklį, pritaikant transformacijos taisyklėmis grįstą algoritmą ir Laravel PSM profilį. Po transformacijos iš PIM į PSM pasirinktas transformacijų įrankis sugeneruota PSM XMI failą. Sugeneruotą PSM failą esant poreikiui galima modifikuoti naudojantis pasirinktu modeliavimo įrankiu, arba transformuoti šį modelį tiesiai į programinį kodą. 15 pav., 16 pav. pateikiamos veiklos diagramos detalizuojančios transformacijos iš PIM į PSM procesą. Pirmiausia IS analitikas turėtų importuoti XMI formato PIM, jei modelis atitiks UML XMI failų reikalavimus modelių transformacijos variklis pradės transformaciją.



15 pav. PIM importavimo į transformacijų įrankį veiklos diagrama

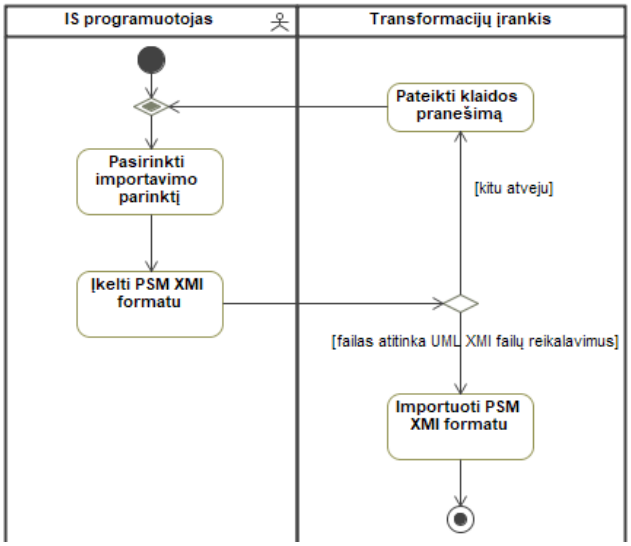
Transformacijos metu, pirmiausia bus priskiriamas LaravelModel ir LaravelRequest stereotipai klasėms turinčioms Entity stereotipą, tuomet bus priskiriamas LaravelBlade stereotipas klasėms turinčioms Boundary stereotipą, tuomet LaravelController ir LaravelRoutes stereotipai bus priskiriamas klasėms turinčioms Control stereotipą. Priskiriant naujus stereotipus klasėms, taip pat yra perkopijuojami buvę šių klasių ryšiai.



16 pav. Transformacijos iš PIM į PSM veiklos diagrama

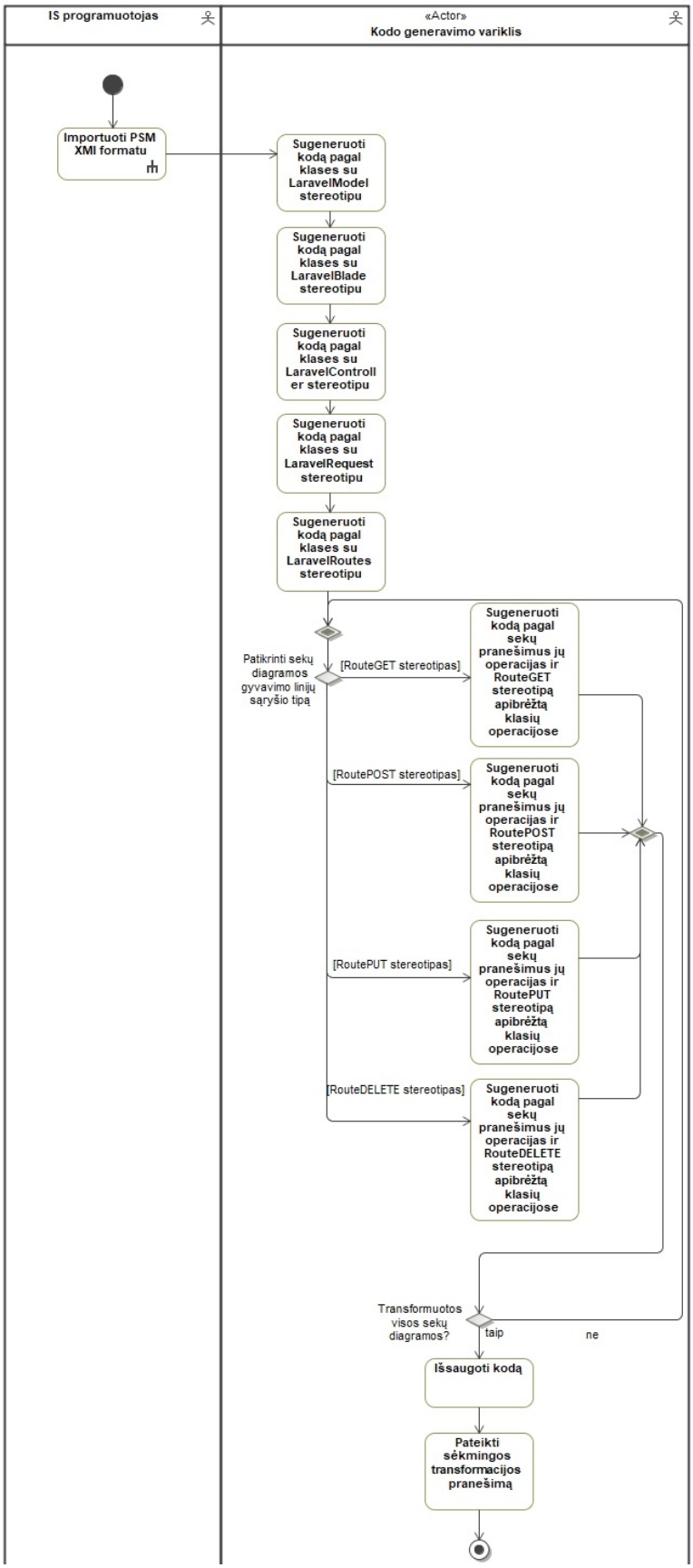
Po naujų stereotipų priskyrimo ir naujų klasių sukūrimo perkopijuojami sekų diagramų aktoriai ir gyvavimo linijos. Po perkopijavimo, priklausomai nuo pranešimų siuntimo eigos, pagal taisykles padaromos pranešimų ir juose esančių operacijų bei kintamųjų kopijos. Pagal pranešimų siuntimo eiliškumo taisykles taip pat priskiriami stereotipai klasių operacijoms (RoutePUT, RouteGET, RoutePOST, RouteDELETE). Atlikus visus veiksmus, modelių transformacijos variklis išsaugos sugeneruotą PSM XMI formatu ir pateiks sėkmingos transformacijos pranešimą.

Transformacija iš PSM į programinį kodą atliekama naudojant kodo generavimo variklį, pritaikant transformacijos algoritmą, grįstą transformacijos taisyklėmis. Po šios transformacijos gaunami PHP Laravel programinio kodo fragmentai, kuriuos programuotojas gali pabaigti programuoti, norėdamas pilnai realizuoti sumodeliuotą sistemą. 17 pav., 18 pav. pateikiamos veiklos diagramos detalizuojančios transformacijos iš PSM į programinį kodą procesą. Pirmiausia IS programuotojas turėtų importuoti XMI formato PSM, jei modelis atitiks UML XMI failų reikalavimus kodo generavimo variklis pradės transformaciją.



17 pav. PSM importavimo į transformacijų įrankį veiklos diagrama

Transformacijos metu, pirmiausia kodas bus generuojamas pagal klases turinčias LaravelModel stereotipą, toliau pagal klases turinčias LaravelBlade stereotipą, tuomet pagal klases turinčias LaravelController, LaravelRequest ir LaravelRoutes stereotipus. Sugeneravus kodą pagal klasių stereotipus, priklausomai nuo LaravelController stereotipą turinčios klasės operacijų stereotipų, toliau kodas generuojamas pagal šios klasės operacijų stereotipo tipą, sekų diagramoje esančius pranešimus ir juose esančias operacijas su kintamaisiais.



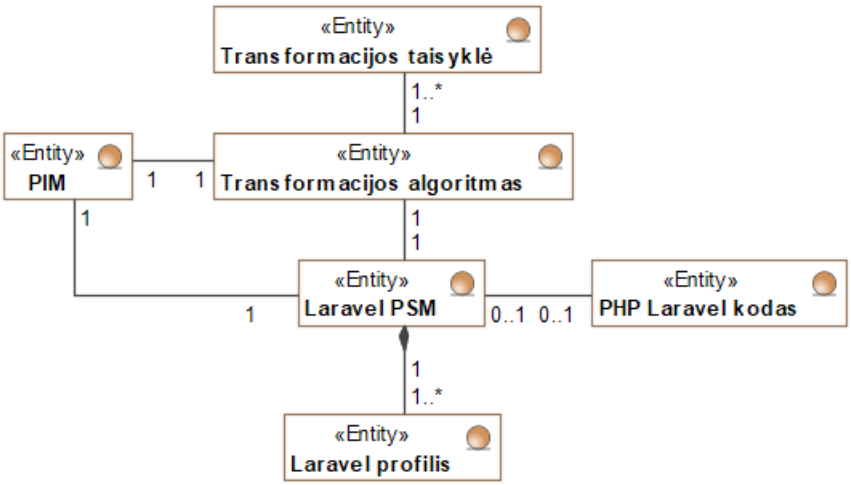
18 pav. Transformacijos iš PSM į programinį kodą veiklos diagrama

Po generavimo kodo fragmentai yra išsaugomi ir pateikiamas sėkmingos transformacijos pranešimas. Šiuos kodo fragmentus naudotojas gali sėkmingai įkelti į Laravel projektą ir pabaigti programuoti pagal numatytą funkcionalumą.

2.2. Dalykinės srities modelis

19 pav. pateiktas siūlomos metodikos dalykinės srities modelis kurį sudaro:

1. Nuo platformos nepriklausomas modelis aprašytas UML kalba, kurį sudaro klasių ir sekų diagrama.
2. Laravel karkasui specifinis modelis aprašytas UML kalba ir turintis Laravel specifinius stereotipus.
3. Programinis kodas, kuris po transformacijos yra gaunamas iš PSM.
4. Transformacijos algoritmai, kurie yra naudojami transformacijai į PSM, transformacijai į programinį kodą ir yra sudaryti pagal transformacijos taisykles, kurios sudarytos pagal UML metamodelį.

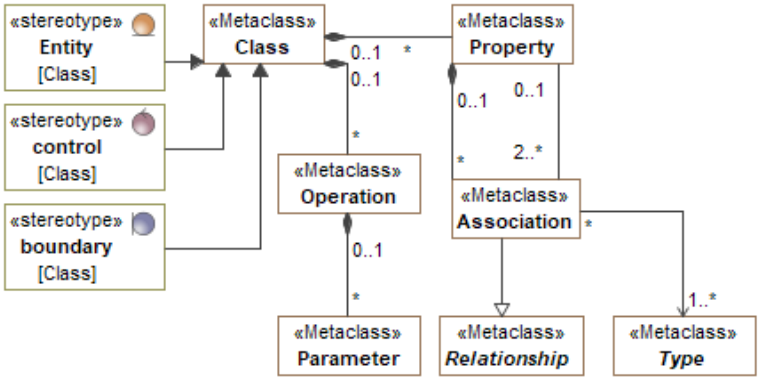


19 pav. Siūlomos metodikos dalykinės srities modelis

20 pav. pateikiamas transformacijai iš PIM į PSM aktualus PIM klasių UML metamodelio fragmentas. Šį metamodelio fragmentą sudaro [1]:

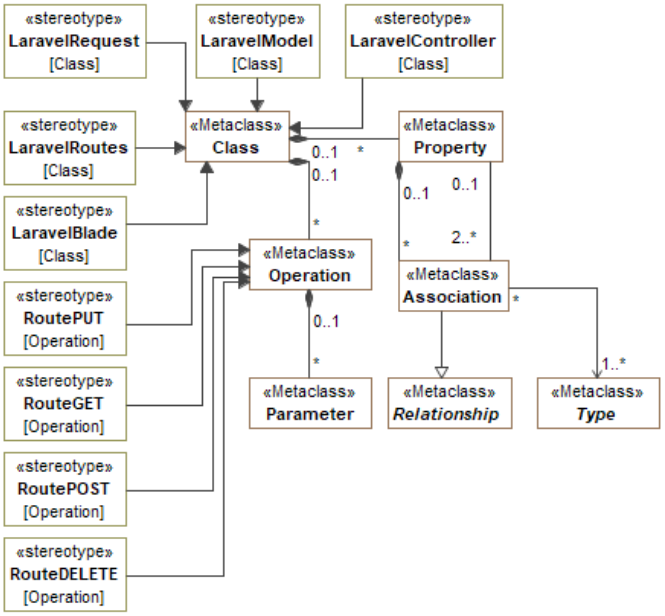
- Class metaklasė – skirta klasifikuoti objektus nurodant jų ypatybes, apibūdinančias šių objektų struktūrą ir elgseną.
- Property metaklasė – struktūrinė savybė, kurios paskirtis yra apibūdinti klasifikatorių ir jo dalių atributus.
- Operation metaklasė – klasės elgsenos ypatybė, kurios paskirtis nurodyti Interface, DataType ar Class operacijas su vardu, tipu, parametrais ir apribojimais.
- Parameter metaklasė – Operation dalis, jo paskirtis perduoti informaciją į ar iš Operation kvietimo. Jei Operation turi grįžtamąjį Parameter, tuomet Operation tipas yra toks pat kaip šio Parameter, kitu atveju Operation tipo neturės.
- Association metaklasė – nurodo semantinį ryšį, kuris gali atsirasti tarp esamų egzempliorių. Ji turi bent du memberEnds atstovaujančius Property, kiekvienas iš jų turi tam tikrą ryšio pabaigos tipą.
- Relationship metaklasė – elementas, kurio paskirtis nurodyti tam tikrą ryšį su kitais elementais.

- Type metaklasė – nurodo Association ryšio tipą.
- Entity, Control, Boundary stereotipai – Entity yra stereotipas reprezentuojantis duomenis. Boundary yra stereotipas reprezentuojantis sąveikas su aktoriais ir Control. Control yra objektai tarpininkaujantys tarp Entity ir Boundary. Control atsakingas už operacijų vykdymą einančių iš Boundary komunikuojant su Entity ir Boundary objektais.



20 pav. PIM klasių metamodelio fragmentas

21 pav. pateikiamas transformacijai iš PIM į PSM ir iš PSM į programinį kodą aktualus PSM klasių UML metamodelio fragmentas. Šiame metamodelio fragmente nebėra Entity, Control, Boundary stereotipų, juos pakeičia Laravel specifiniai stereotipai: LaravelModel, LaravelController, LaravelBlade, LaravelRequest, LaravelRoutes, RoutePUT, RouteGET, RoutePOST, RouteDELETE.

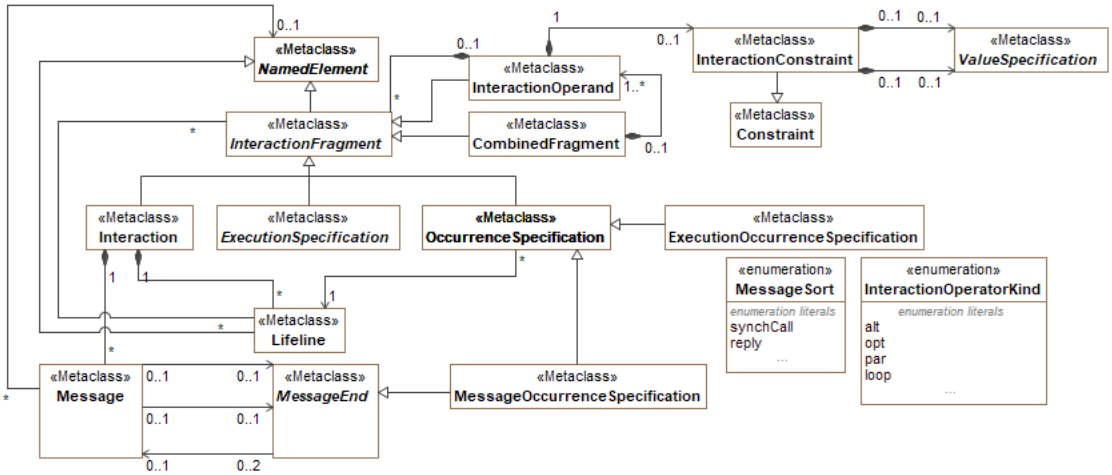


21 pav. PSM klasių metamodelio fragmentas

22 pav. pateikiamas transformacijai iš PIM į PSM aktualus sekų diagramos UML metamodelio fragmentas. Šį metamodelio fragmentą sudaro [1]:

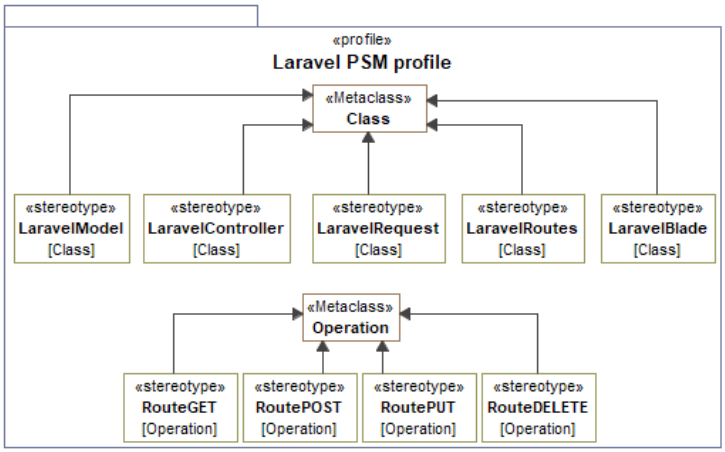
- NamedElement metaklasė – modelio elementas, kuris gali turėti pavadinimą.
- InteractionFragment metaklasė – abstrakčiausia bet kokio veiksmo vieneto metaklasė.

- InteractionOperand metaklasė – CombinedFragment dalis, nurodanti operando išraišką pateiktą antrojoje CombinedFragment dalyje.
- CombinedFragment metaklasė – metaklasė apibrėžianti InteractionFragments išraišką kuria naudotojas gali aprašyta norimą logiką.
- InteractionConstraint metaklasė – loginė išraiška, sauganti operandą CombinedFragment.
- Constraint metaklasė – teiginys, nurodantis apribojimą, kurį turi atitikti bet kuri modelio realizacija, kurioje šis teiginys yra apibrėžtas.
- ValueSpecification metaklasė – metaklasė tiesiogiai apibrėžianti reikšmių rinkinius.
- Interaction metaklasė – metaklasė apibrėžianti abstrakčią sintaksę, semantiką ir notacijas InteractionFragment, OccurrenceSpecification, ExecutionSpecification, StateInvariant metaklasėms.
- ExecutionSpecification metaklasė – metaklasė apibrėžianti veiksmų sekos semantika <pradžia, pabaiga>.
- OccurrenceSpecification metaklasė – pagrindinis veiksmų semantinis vienetas. OccurrenceSpecification nurodytos įvykių sekos yra veiksmų reikšmės.
- ExecutionOccurrenceSpecification metaklasė – metaklasė nurodanti momentus, kuriais prasideda veiksmai ar elgsenos.
- Lifeline metaklasė – metaklasė apibrėžianti atskirus veiksmų dalyvius.
- Message metaklasė – apibrėžia abstrakčią sintaksę, semantiką ir notaciją Message, MessageEnd, MessageOccurrenceSpecification, MessageSort, MessageKind, DestructionOccurrenceSpecification, Gate metaklasėms.
- MessageEnd metaklasė – metaklasė apibrėžianti specifinę Messages semantiką, atitinkančią Message naudojamas sąvokas.
- MessageOccurrenceSpecification metaklasė – metaklasė reprezentuojanti siuntimo arba gavimo įvykį susietą žinute tarp Lifelines.
- InteractionOperatorKind enumeratorius – enumeratorius nurodantis įvairius CombinedFragment operatorius. Šis enumeratorius apibrėžia CombinedFragment operatoriaus tipą.
- MessageSort enumeratorius – enumeratorius tipas, kuris nurodo ryšio veiksmo tipą kuris buvo panaudotas generuojant Message.



22 pav. Sekų diagramos metamodelio fragmentas

PSM sekų UML metamodelio fragmentas nėra pateikiamas, kadangi tolimesnių pokyčių sekų diagramoje nėra, sekos išlieka sudarytos tą pačią struktūrą pagal 22 pav. pateiktą transformacijai iš PIM į PSM aktualaus sekų diagramos UML metamodelio fragmentą. 23 pav. pateikiamas transformacijai iš PIM į PSM aktualus UML profilis Laravel karkasui. Šį profilį sudaro stereotipai išplečiantis Class ir Operation metaklases.



23 pav. PSM Laravel profilis

Pagrindiniai išplečiantys metaklasę stereotipai ir jų apibūdinimai pateikiami 5 lentelėje.

5 lentelė. Metaklasę išplečiantys specifiniai Laravel stereotipai [22]

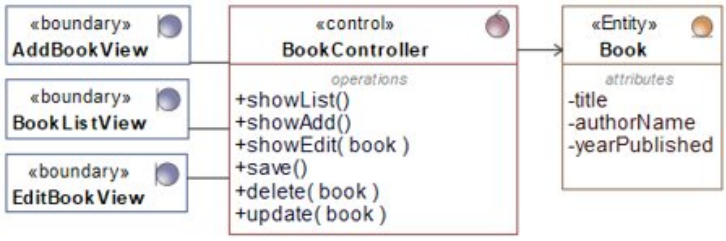
Stereotipas	**Aprašymas**
LaravelModel	Stereotipas apibrėžiantis Laravel karkaso Model elementų tipą. Laravel karkase, modeliai atitinka duomenų bazės lentelės rodinį ir yra sąsaja darbui su duomenų bazės lentele. Modeliai leidžia su Eloquent ORM užklausomis pasiekti ir valdyti duomenis iš duomenų bazės.
LaravelController	Stereotipas apibrėžiantis Laravel karkaso Controller elementų tipą. Laravel karkase, valdikliai skirti sutelkti užklausų apdorojimo ir tvarkymo logiką vienoje klasėje. Pagal MVC architektūrą, karkase valdiklio klasė veikia kaip srautinis elementas apjungiantis modelio ir rodinio elementų tipus.
LaravelBlade	Stereotipas apibrėžiantis Laravel karkaso Blade elemetų tipą. Laravel karkase, Blade yra paprastas, tačiau galingas rodinių šablonų valdymo variklis. Blade paskirtis yra sudaryti atvaizdavimo rodinius, kuriuos galima paveldėti, ar iš jų sudaryti atskiras sekcijas.
LaravelRequest	Stereotipas apibrėžiantis Laravel karkaso Request elementų tipą. Laravel karkase, užklausų valdymo elementas Request yra skirtas apdoroti užklausas ateinančias iš atvaizdavimo rodinių formų, suteikiant galimybę aprašyti specifinę logiką ar reikalavimus skirtus būtent tam tikros užklausos apdorojimui prieš šiai užklausai pasiekiant valdiklį.
LaravelRoutes	Stereotipas apibrėžiantis Laravel karkaso Routes elementų tipą. Laravel karkase, Routes elementas yra maršruto parinkimo sistema, apibrėžianti ir susiejanti HTTP užklausas su jiems priskirtais metodais pagal užklausos metodą ir nurodytą kelią su esančiais kintamaisiais.
RoutePUT	Stereotipas apibrėžiantis Laravel karkaso Routes elementų tipo faile esančių metodų tipą. Šis metodų tipas nurodo, kad tai yra modifikavimo metodo tipas.
RouteGET	Stereotipas apibrėžiantis Laravel karkaso Routes elementų tipo faile esančių metodų tipą. Šis metodų tipas nurodo, kad tai yra rodinio atvaizdavimo metodo tipas.
RoutePOST	Stereotipas apibrėžiantis Laravel karkaso Routes elementų tipo faile esančių metodų tipą. Šis metodų tipas nurodo, kad tai yra naujo elemento pridėjimo metodo tipas.
RouteDELETE	Stereotipas apibrėžiantis Laravel karkaso Routes elementų tipo faile esančių metodų tipą. Šis metodų tipas nurodo, kad tai yra elemento šalinimo metodo tipas.

Lentelėje pateikti devyni specifiniai Laravel stereotipai išpildo siūlomos metodikos dalykinės srities modelio Laravel profilio dalį, apibrėždami pagrindines Laravel karkaso bendrines dalis. Taigi, metodikos dalykinės srities modelis apibrėžiamas turint PIM ir PSM aktualius klasių metamodelio fragmentus, PIM aktualius sekų metamodelio fragmentus, pagal kuriuos sudaromi PIM ir PSM. Taip pat, Laravel profilį su specifiniais Laravel stereotipais, pačias transformacijos taisykles, transformacijos algoritmą ir sugeneruojamą programinį kodą.

2.3. Transformacijos iš modelio į modelį ir iš modelio į PHP Laravel programinį kodą metodikos formalus aprašas

2.3.1. Transformacijos iš PIM į PSM metodikos formalus aprašas

Transformacijos iš PIM į PSM metodikos dalies formalus aprašas susideda iš ATL kodo fragmentų ir nedidelės apimties modelio transformacijos pavyzdžio prie kiekvieno algoritmo žingsnio. Pavyzdinio nuo platformos nepriklausomas modelio fragmentai sudaryti iš klasių ir sekų diagramų, kuriuos būtų galima transformuoti iš PIM į PSM pateikiami 24 pav., 32 pav., 34 pav., 36 pav., 38 pav., 40 pav.



24 pav. PIM – knygų parduotuvės klasių diagramos fragmentas

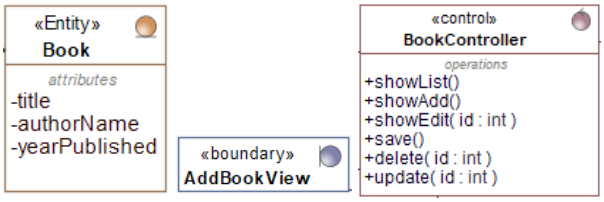
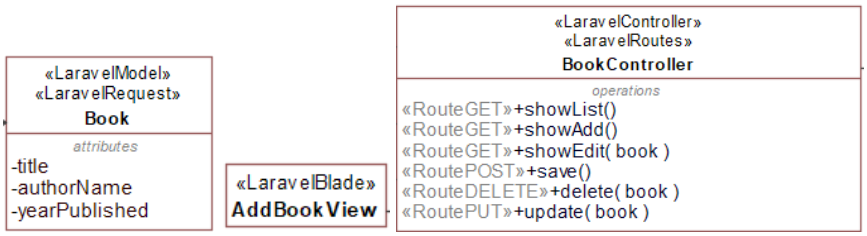
Transformacija iš PIM į PSM atliekama naudojant Eclipse ATL įrankį, toliau pateikiami algoritmų fragmentų pavyzdžiai 6, 7 lentelėse yra sudaryti ATL transformacijų kalba, kuri atitinka MOF QVT standartą. Transformuojant PIM į PSM, PIM XMI formatu turėtų būti importuotas į įrankį ir pasirenkama transformavimo parinktis. Tuomet ATL modelių transformacijos variklis pagal transformacijos taisyklėmis grįstą algoritmą transformuotų modelį į PSM, modelis būtų sugeneruotas ir išsaugotas XMI formatu. Transformacijos iš PIM į PSM pilnas algoritmo fragmentas pateikiamas 32 lentelėje. Toliau detalizuojami šio algoritmo pagrindiniai fragmentai. Vienas iš jų yra LaravelModel, LaravelController, LaravelRequest, LaravelRoutes, LaravelBlade stereotipų pritaikymas modelio klasėms. Algoritmo fragmentas klasių stereotipų pritaikymui pateikiamas 6 lentelėje.

6 lentelė. Algoritmo fragmentas skirtas pritaikyti klasių stereotipus

Algoritmo dalies aprašas	**Klasių stereotipų pritaikymas** Pritaikytų klasių stereotipų radimas ir naujų atitinkamų stereotipų pritaikymas atitinkamoms klasėms.

6 lentelė. Algoritmo fragmentas skirtas pritaikyti klasių stereotipus

Algoritmo fragmentas	`if(s.hasStereotype('boundary')) {`
	`  for(as in s.getApplicableStereotypes()){`
	`  if(as.name = 'LaravelBlade' and t.hasStereotype('LaravelBlade') = false){`
	`  t.applyStereotype(as);`
	`  for (a in as.getAllAttributes()) {`
	`  if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {`
	`  t.setValue(as, a.name, s.getValue(as, a.name));`
	`  }`
	`  }`
	`  }`
	`  }`
	`}`
	`else if(s.hasStereotype('control')) {`
	`  for(as in s.getApplicableStereotypes()){`
	`  if(as.name = 'LaravelController' and t.hasStereotype('LaravelController') = false){`
	`  t.applyStereotype(as);`
	`  for (a in as.getAllAttributes()) {`
	`  if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {`
	`  t.setValue(as, a.name, s.getValue(as, a.name));`
	`  }`
	`  }`
	`  }`
	`  if(as.name = 'LaravelRoutes' and t.hasStereotype('LaravelRoutes') = false){`
	`  t.applyStereotype(as);`
	`  for (a in as.getAllAttributes()) {`
	`  if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {`
	`  t.setValue(as, a.name, s.getValue(as, a.name));`
	`  }`
	`  }`
	`  }`
	`  }`

PIM pavyzdys	
PSM pavyzdys	

Šiame algoritmo fragmente patikrinama, ar įvesties modelio klasės turi boundary arba control stereotipus, tada patikrinama, ar yra galimybė pritaikyti LaravelBlade, LaravelController ar LaravelRoutes stereotipus. Jei tokia galimybė yra ir išvesties modelis dar neturi pritaikyto vieno iš stereotipų, tuomet pritaikomi reikiami stereotipai ir priskiriamos šių stereotipų reikšmės išvesties klasei. Šiame algoritmo fragmente taip pat naudojamas atskiras pagalbinis metodas patikrinti, ar elementas turi tam tikrą stereotipą. Šis pagalbinis metodas pateikiamas 25 pav.

```
helper context UML2!Element def: hasStereotype(stereotype : String) : Boolean =
    self.getAppliedStereotypes() -> collect(st | st.name) -> includes(stereotype);
```

25 pav. Pagalbinis stereotipų tikrinimo metodas

Šis pagalbinis metodas priima string tipo stereotipo pavadinimo reikšmę, tuomet iš visų elementui pritaikytų stereotipų sąrašo patikrina ar egzistuoja reikiamas stereotipas ir gražina true arba false reikšmę, priklausomai ar elementas kuriam iškviečiamas metodas turi tam tikrą stereotipą, ar ne.

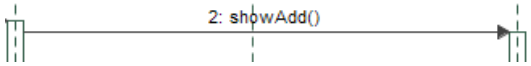
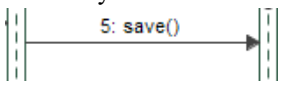
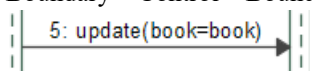
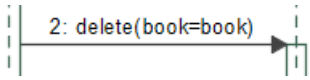
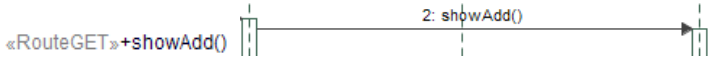
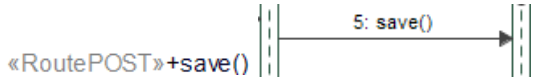
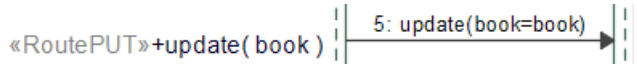
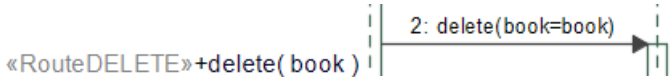
Kita algoritmo dalis yra stereotipų pritaikymas klasių operacijoms pagal sekų diagramas. Algoritmo fragmentai operacijų stereotipų pagal sekų diagramą pritaikymui pateikiamas 7 lentelėje.

7 lentelė. Algoritmo fragmentas skirtas operacijų stereotipų pritaikymui pagal sekų diagramas

Algoritmo dalies aprašas	**Sekų diagramos apdorojimas** Susijusių klasių sekų diagramose sąryšių radimas ir sekų diagramų elementų, jų žinučių ir pritaikytų stereotipų išsaugojimas, bei reikiamų stereotipų operacijoms pritaikymas.
Algoritmo fragmentas	

```
thisModule.counter <- 1;

for(i in s.message){
if(s.message->size() >= thisModule.counter){
thisModule.outputMessage <- thisModule.outputMessage.append(s.message->at(thisModule.counter).name);
thisModule.fragmentOut <- s.fragment -> select(frg | frg.__xmiID__ = s.message->at(thisModule.counter).receiveEvent.__xmiID__) -> collect(frg | frg.covered);
for(fragment in thisModule.fragmentOut){
    for(lif in fragment){
        thisModule.lifelineOut <- s.lifeline -> select(ll |ll.__xmiID__ = lif.__xmiID__) -> collect(ll | ll.represents.__xmiID__);
        for(lrepresents in thisModule.lifelineOut){
            thisModule.ownedAttr <- s.ownedAttribute -> select(oa | oa.__xmiID__ = lrepresents) ->collect(oa | oa.type.__xmiID__);
            for(owned in thisModule.ownedAttr){
                thisModule.sclass <- UML2!"uml::Class".allInstances() -> select(cl | cl.__xmiID__ = owned) -> collect(cl | cl);
                thisModule.rsclass <- UML2!"uml::Class".allInstances() -> select(cl | cl.__xmiID__ = owned) -> collect(cl | cl.name);
                for(class in thisModule.sclass){
                    thisModule.outputClass <- thisModule.outputClass.append(class);
                }
                for(rclass in thisModule.rsclass){
                    thisModule.output <- thisModule.output.append(rclass);
                }
            }
        }
    }
}



thisModule.counter <- thisModule.counter + 1;
}else{
    thisModule.counter <- 1;
}

thisModule.stereotypedValues<-thisModule.stereotypedValues.append(thisModule.output);
thisModule.stereotypedClasses<-thisModule.stereotypedClasses.append(thisModule.outputClass);
thisModule.savedMessages<-thisModule.savedMessages.append(thisModule.outputMessage);

thisModule.output <- Sequence{};
thisModule.outputClass <- Sequence{};
thisModule.outputMessage <- Sequence{};



for(singleClassArray in thisModule.stereotypedClasses){
    for(singleClassNameArray in thisModule.stereotypedValues){
        if(singleClassArray.size() = 3){
            if(singleClassNameArray.size() = 3){
                if(singleClassArray.at(1).hasStereotype('LaravelBlade') and singleClassArray.at(2).hasStereotype('LaravelController')
                        and singleClassArray.at(3).hasStereotype('LaravelBlade')){
                    for(outOp in thisModule.savedMessages){
                        if(outOp.size() = 3){
                            if(opas.owner.name = singleClassNameArray.at(2) and outOp->exists(p | p.startsWith(opas.name))){
                                for(ops in opas.getApplicableStereotypes()){
                                    if(ops.name = 'RouteGET' and top.hasStereotype('RouteGET') = false and top.hasStereotype('RouteDELETE') = false
                                            and top.hasStereotype('RoutePOST') = false
                                            and top.hasStereotype('RoutePUT') = false and opas.owner.hasStereotype('control')){
                                        top.applyStereotype(ops);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```

7 lentelė. Algoritmo fragmentas skirtas operacijų stereotipų pritaikymui pagal sekų diagramas

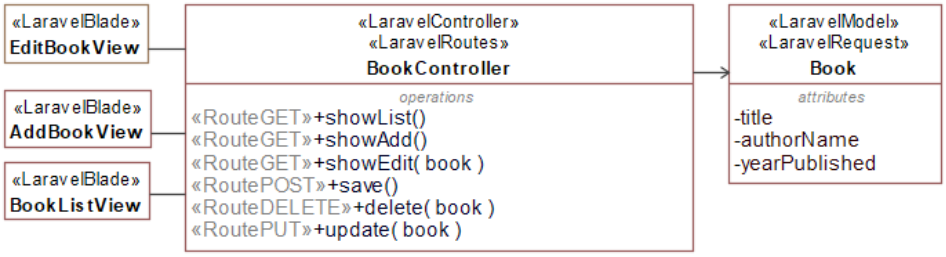
PIM pavyzdys	1. Pranešimai, jų operacijos ir kintamieji, jei pranešimų tarp gyvavimo linijų eiliškumas nenutrūkstamai Boundary - Control – Boundary:  2. Pranešimai, jų operacijos ir kintamieji, jei pranešimų tarp gyvavimo linijų eiliškumas nenutrūkstamai Boundary – Control – Boundary – Control – Entity ir pranešimo operacija neturi kintamojo:  3. Pranešimai, jų operacijos ir kintamieji jei pranešimų tarp gyvavimo linijų eiliškumas nenutrūkstamai Boundary – Control – Boundary – Control – Entity ir pranešimo operacija turi kintamąjį:  4. Pranešimai, jų operacijos ir kintamieji, jei pranešimų tarp gyvavimo linijų eiliškumas nenutrūkstamai Boundary - Control – Entity, pranešimo operacija turi kintamąjį: 
PSM pavyzdys	1. Sekų diagrama – pranešimai, jų operacijos ir (jei turi) kintamieji. Klasių diagrama – klasės turinčios LaravelController stereotipą nurodytai operacijai priskiriamas RouteGET stereotipas:  2. Sekų diagrama – pranešimai, jų operacijos. Klasių diagrama – klasės turinčios LaravelController stereotipą nurodytai operacijai priskiriamas RoutePOST stereotipas:  3. Sekų diagrama – pranešimai, jų operacijos ir kintamieji. Klasių diagrama – klasės turinčios LaravelController stereotipą nurodytai operacijai priskiriamas RoutePUT stereotipas:  4. Sekų diagrama – Messages, jų operacijos ir kintamieji. Klasių diagrama – klasės turinčios LaravelController stereotipą nurodytai operacijai priskiriamas RouteDELETE stereotipas: 

Šis algoritmo fragmentas pritaiko stereotipus klasių diagramos operacijoms pagal naudotojo sudarytas sekų diagramas. Keliaudamas per sekų diagramas, algoritmas atranda sąryšį tarp sekų diagramų ir naudojamų elementų tipą, ar tai boundary, control, entity stereotipus turintys elementai ir kokia eiga jie siejasi žinutėmis. Kiekvienos sekų diagramos žinučių sekos tipas, sekų diagramos dydis ir patys elementai eigos tvarka yra išsaugomi į kintamuosius ir tolimesnėje algoritmo dalyje, tikrinami ši žinučių eiga. Pagal elementų tipą ir žinučių sekos tipą pritaikomas specifinių Laravel stereotipų priskyrimas klasių operacijoms.

2.3.2. Transformacijos iš PSM į Laravel PHP programinį kodą metodikos formalus aprašas

Po transformacijos iš modelio į modelį, galima atlikti transformaciją į PHP Laravel programinį kodą naudojant Eclipse Acceleo įrankį ir transformacijos taisyklėmis grįstą algoritmą. Transformacijos iš PSM į programinį kodą metodikos dalies formalus aprašas susideda iš pilnų ir skaidytų esminių Acceleo kodo fragmentų ir nedidelės apimties modelio transformacijos pavyzdžio prie kiekvieno algoritmo žingsnio. Acceleo įrankis naudoja šablonu grįstą Acceleo kalbą, kuri atitinka OMG MOF

Model to Text Language standartą. Pavyzdinio nuo platformos priklausomo modelio fragmentai sudaryti iš klasių ir sekų diagramų, kurie yra transformuojami iš PSM į programinį kodą pateikiami 26 pav., 33 pav., 35 pav., 37 pav., 39 pav., 41 pav.



26 pav. PSM – knygų parduotuvės klasių diagramos fragmentas

Acceleo programinio kodo generavimo variklis pagal transformacijos taisyklėmis grįstą algoritmą transformuoja modelį į PHP Laravel programinį kodą. Algoritmo fragmentai su pavyzdžiais, kurie buvo sukurti su Acceleo transformacijų kalba, pateikiami 8, 9, 10, 11, 12 lentelėse. Pilnas transformacijos iš PSM į programinį kodą algoritmas pateikiamas 33 lentelėje.

8 lentelė. Algoritmo fragmentas skirtas Laravel Controller programinio kodo generavimui

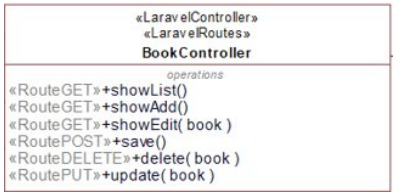
Algoritmo dalies aprašas	**Laravel Controller kodo generavimas** Atitinkamo Laravel Controller stereotipo, šį stereotipą turinčių klasių ir jų operacijų radimas ir Laravel Controller specifinės struktūros pritaikymas programiniam kodui.

Algoritmo fragmentas	[template public generateControllerClassDeclarations(c:Class) ? (c.hasStereotype('LaravelController'))]
	[file ('app/Http/Controllers/' + c.name + '.php', false, 'UTF-8')]
	<?php
	
	namespace App\Http\Controllers;
	
	[for (ass:Association | c.getAssociations())]
	[for (rel:Class | ass.relatedElement)]
	[if (rel.hasStereotype('LaravelModel'))]
	[if (rel.name <> c.name and c.name.startsWith(rel.name))]
	use App\Models\[rel.name/];
	[/if]
	[/if]
	[if (rel.hasStereotype('LaravelRequest'))]
	[if (rel.name <> c.name and c.name.startsWith(rel.name))]
	use App\Http\Requests\[rel.name/]Request;
	[/if]
	[/if]
	[/for]
	[/for]
	//[protected (c.name)]
	//Nepergeneruojama sasaju su kitais modeliais apibrezimo dalis
	//[/protected]
	use Illuminate\Http\Request;
	
	class [c.name/] extends Controller
	{
	
	[for (op:Operation | c.getOperations())]
	[if (op.hasStereotype('RouteGET'))]
	[for (ass:Association | c.getAssociations())]
	[for (rel:Class | ass.relatedElement)]
	[if (rel.hasStereotype('LaravelModel'))]
	[if (rel.name <> c.name and c.name.startsWith(rel.name))]
	[op.visibility/] function [op.name/]([if (op.getOwnedMembers()->notEmpty())][rel.name/] ['$'+ op.getOwnedMembers().name/][/if]) {
	//[protected (op.name)]
	// RouteGET metodo kodo nepergeneruojama dalis
	//[/protected]
	}
	
	[/if]
	[/if]
	[/for]
	[/for]
	[/if]
	[if (op.hasStereotype('RoutePOST'))]
	[for (ass:Association | c.getAssociations())]
	[for (rel:Class | ass.relatedElement)]
	[if (rel.hasStereotype('LaravelModel'))]
	[if (rel.name <> c.name and c.name.startsWith(rel.name))]

8 lentelė. Algoritmo fragmentas skirtas Laravel Controller programinio kodo generavimui

<table>
<tr><td>Algoritmo fragmentas</td><td>

```
[op.visibility/] function [op.name/]([rel.name/]Request $request) {
    //[protected (op.name)]
    $[rel.name.toLower()/] = [rel.name/]::create($request->all());
    //[/protected]
   }
[/if]
            [/if]
        [/for]
[/for]
    [/if]
    [if (op.hasStereotype('RoutePUT'))]
[for (ass:Association | c.getAssociations()) ]
        [for (rel:Class | ass.relatedElement) ]
            [if (rel.hasStereotype('LaravelModel'))]
                [if (rel.name <> c.name and c.name.startsWith(rel.name))]
[op.visibility/] function [op.name/]([rel.name/]Request $request, [rel.name/] $[rel.name.toLower()/]) {
    //[protected (op.name)]
    $[rel.name.toLower()/]->update($request->all());
    //[/protected]
   }

[/if]
            [/if]
        [/for]
[/for]
    [/if]
    [if (op.hasStereotype('RouteDELETE'))]
[for (ass:Association | c.getAssociations()) ]
        [for (rel:Class | ass.relatedElement) ]
            [if (rel.hasStereotype('LaravelModel'))]
                [if (rel.name <> c.name and c.name.startsWith(rel.name))]
[op.visibility/] function [op.name/]([if (op.getOwnedMembers()->notEmpty())][rel.name/] ['$'+ op.getOwnedMembers().name/][/if]) {
    //[protected (op.name)]
    $[rel.name.toLower()/]->delete();
    //[/protected]
   }
[/if]
            [/if]
        [/for]
[/for]
    [/if]
    [/for]
}
//[protected (c.name)]
//Erdve kitam kodui
//[/protected]
[/file]
[/template]
```

</td></tr>
<tr><td>PSM pavyzdys</td><td>Klasė turinti LaravelController stereotipą, jos operacijos, operacijų kintamieji ir (jei egzistuoja) ryšiai su kitomis klasėmis:
 
</td></tr>
<tr><td>Pavyzdinis programinis kodas</td><td>KlasėController.php failas:

```
<?php
use App\Klasė;
namespace App\Http\Controllers;
use App\Http\Requests\KlasėRequest;
use App\KlasėTurintiRyšį;
use Illuminate\Http\Request;

class KlasėController extends Controller
{
   public function metodas()
   {
       //
   }
}
```

</td></tr>
</table>

Pateiktame Laravel Controller programinio kodo generavimo algoritmo fragmente iš modelio randamos klasės turinčios Laravel Controller stereotipą, šių klasių operacijos su Laravel specifiniais stereotipais ir sąryšiai tarp klasių su kitomis klasėmis turinčiomis Laravel Model ir Laravel Request stereotipus. Algoritmas pagal klasės pavadinimą sugeneruoja klasės failą ir prideda jį į tipinį Laravel struktūros aplanką skirtą saugoti valdiklių klases. Sąryšiai tarp klasių sugeneruojami kaip importavimo dalys programiniame kode, pats sugeneruotas kodas įgauna Laravel Controller specifinę

struktūrą. Taip pat į sugeneruojamą klasę sugeneruojami šiai klasei priklausantys metodai, pagal jų atitinkamus stereotipus. Operacijos turėjusios RoutePUT, RoutePOST ir RouteDELETE stereotipus tampa pilnai veikiančiais iš vidinės sistemos dalies atnaujinimo ir šalinimo metodais. Kituose metoduose paliekama nepergeneruojama programinio kodo erdvė, skirta programuotojams pabaigti programuoti sugeneruotus metodus. Toliau, 9 lentelėje pateikiamas algoritmo fragmentas skirtas Laravel Model programinio kodo generavimui.

9 lentelė. Algoritmo fragmentas skirtas Laravel Model programinio kodo generavimui

<table>
<tr><td>Algoritmo dalies aprašas</td><td>Laravel Model kodo generavimas
Atitinkamo Laravel Model stereotipo, šį stereotipą turinčių klasių ir jų atributų radimas ir Laravel Model specifinės struktūros pritaikymas programiniam kodui.</td></tr>
<tr><td>Algoritmo fragmentas</td><td>

```
[template public generateModelClassDeclarations(c:Class) ? (c.hasStereotype('LaravelModel'))]
[file ('app/Models/' + c.name + '.php', false, 'UTF-8')]
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
//[protected (c.name)]
//Erdve importams
//[/protected]
class [c.name/] extends Model
{
    use HasFactory;

   protected $fillable = ['['/]
    [for (p: Property | c.attribute) separator(',')]
    [if (p.name <> '')]
    '[p.name/]'
    [/if]
    [/for]
    [']'/];

    [for (ass:Association | c.getAssociations()) ]
        [for (rel:Class | ass.relatedElement) ]
            [if (rel.hasStereotype('LaravelModel'))]
                [if (rel.name <> c.name)]
    public function [rel.name.toLower()/]() {
    //[protected (rel.name)]
    //Erdve belongsTo, hasMany.. rysiams
    return $this->([rel.name/]);
    //[/protected]
    }
                [/if]
            [/if]
        [/for]
[/for]
//[protected (c.name)]
//Erdve kitam kodui
//[/protected]
}
[/file]
[/template]
```

</td></tr>
<tr><td>PSM pavyzdys</td><td>Klasė turinti LaravelModel stereotipą jos atributai ir (jei egzistuoja) ryšiai su kitomis klasėmis:
 
</td></tr>
<tr><td>Pavyzdinis programinis kodas</td><td>Klasė.php failas:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;

class Klasė extends Model
{
   protected $fillable = [
        'atributas'
   ];
}
```

</td></tr>
</table>

Šiame algoritmo fragmente pateikiamas atitinkamų modelio klasių turinčių Laravel Model stereotipą ir jų atributų radimas ir pritaikymas Laravel Model specifinei struktūrai. Algoritmas pagal Laravel Model stereotipą turinčios klasės pavadinimą sugeneruoja klasės failą ir prideda jį į tipinį Laravel struktūros aplanką skirtą saugoti modelių klases. Algoritmo fragmentas sugeneruoja Laravel Model tipo klasės turinį su visais atributais kurie buvo apibrėžti klasėje turinčioje Laravel Model stereotipą. Algoritmo fragmentas skirtas Laravel Blade programinio kodo generavimui pateikiamas 10 lentelėje.

10 lentelė. Algoritmo fragmentas skirtas Laravel Blade programinio kodo generavimui

Algoritmo dalies aprašas	**Laravel Blade kodo generavimas** Atitinkamo Laravel Blade stereotipo, šį stereotipą turinčių klasių (kurios yra susietos per Laravel Controller Laravel Model klases) ir jų atributų radimas ir Laravel Blade šablone atributų pateikimas programiniam kodui.

Algoritmo fragmentas	`[template public generateBladeClassDeclarations(c:Class) ? (c.hasStereotype('LaravelBlade'))]`
	`[for (ass:Association | c.getAssociations()) ]`
	`[for (rel:Class | ass.relatedElement) ]`
	`[if (rel.hasStereotype('LaravelController'))]`
	`[for (ass1:Association | rel.getAssociations()) ]`
	`[for (rel1:Class | ass1.relatedElement) ]`
	`[if (rel1.hasStereotype('LaravelModel'))]`
	`[if (rel.name.startsWith(rel1.name))]`
	`[file ('resources/views/'+ rel1.name.toLower()+ '/' + c.name.toLower() + '.blade.php', false, 'UTF-8')]`
	`//[protected (c.name)]`
	`<form method="GET/PUT/POST" action="{{ route('[rel1.name.toLower()/].method') }}">`
	
	`<div class="form">`
	`[for (p: Property | rel1.attribute)]`
	`[if (p.name <> '')]<p>[p.name/]:{{ $[rel1.name.toLower()/]->[p.name/] }}</p>[/if]`
	`[/for]`
	`</form>`
	`</div>`
	`</div>`
	`//[/protected]`
	`[/file]`
	`[/if]`
	`[/if]`
	`[/for]`
	`[/for]`
	`[/if]`
	`[/for]`
	`[/for]`
	`[/template]`

PSM pavyzdys	Klasė turinti LaravelBlade stereotipą ir susietos per LaravelController LaravelModel klasės atributai: «LaravelBlade» **AddBookView**

Pavyzdinis programinis kodas	Klasė.blade.php failas:
	`<form method="GET/PUT/POST" action="{{ route('klasė.metodas') }}">`
	
	`<div class="form">`
	`<p>Atributas: {{ $klase->atributas }}</p>`
	`</div>`

Algoritmo fragmente realizuotas atitinkamų Laravel Blade stereotipą turinčių klasių radimas, sąsajų per Laravel Controller stereotipą turinčias klases su klasėmis turinčiomis Laravel Model stereotipą radimas. Aplankų ir failų skirtų Blades failams sugeneravimas, bei šių failų pagal klases užpildymas su atributais iš Laravel Model tipo klasių, su kuriomis Laravel Blade stereotipą turinčios klasės turi sąsają. 11 lentelėje pateikiamas algoritmo fragmentas, kuris skirtas Laravel Request programinio kodo generavimui.

11 lentelė. Algoritmo fragmentas skirtas Laravel Request programinio kodo generavimui

<table>
<tr><td>Algoritmo dalies aprašas</td><td>Laravel Request kodo generavimas
Atitinkamo Laravel Request stereotipo, šį stereotipą turinčių klasių ir jų atributų radimas ir Laravel specifinės struktūros pritaikymas programiniam kodui.</td></tr>
<tr><td>Algoritmo fragmentas</td><td>

```
[template public generateRequestDeclarations(c:Class) ? (c.hasStereotype('LaravelRequest'))]
[file ('app/Http/Requests/' + c.name + 'Request.php', false, 'UTF-8')]
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class [c.name/]Request extends FormRequest
{
   public function authorize()
   {
       return true;
   }
   public function rules()
   {
       return ['['/]
    [for (p: Property | c.attribute) separator(',')]
    [if (p.name <> '')]
    '[p.name/]' => 'required'
    [/if]
    [/for]
    [']'/];
   }
   public function messages()
   {
       return ['['/]
    [for (p: Property | c.attribute) separator(',')]
    [if (p.name <> '')]
    '[p.name/].required' => 'This field is required'
    [/if]
    [/for]
    [']'/];
   }
}

[/file]
[/template]
```

</td></tr>
<tr><td>PSM pavyzdys</td><td>Klasė turinti LaravelRequest stereotipą:
 
</td></tr>
<tr><td>Pavyzdinis programinis kodas</td><td>KlasėRequest.php failas:

```
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
class KlasėRequest extends FormRequest
{
   public function authorize()
   {
       return true;
   }
   public function rules()
   {
       return [
           'atributas' => 'required',
       ];
   }
   public function messages()
   {
       return [
           'atributas.required' => '',       ];
   }
}
```

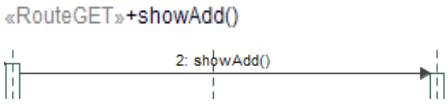
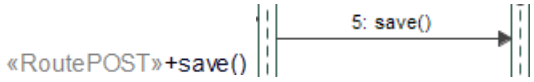
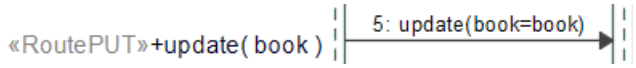
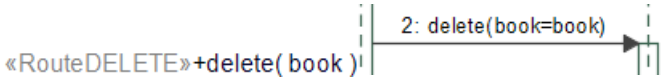
</td></tr>
</table>

Pateiktame algoritmo fragmente iš modelio randamos atitinkamą Laravel Request stereotipą turinčios klasės ir šių klasių atributai. Sugeneruojamas klasės failas su specifine Laravel Request struktūra ir nurodoma failo lokacija pagal Laravel failų struktūrą. Sugeneruotoje Laravel Request tipo klasėje sugeneruojami rules ir messages metodai, kuriuose pateikiami atributai iš modelio klasės. Algoritmo fragmentas skirtas Laravel Routes programinio kodo generavimui pateikiamas 12 lentelėje.

12 lentelė. Algoritmo fragmentas skirtas Laravel Routes programinio kodo generavimui

Algoritmo dalies aprašas	**Laravel Routes kodo generavimas** Atitinkamų Laravel Routes maršrutų radimas klasėms turinčios LaravelController stereotipą, šių klasių operacijų turinčių RouteGET, RoutePOST, RoutePUT, RouteDELETE stereotipus radimas ir Laravel specifinės struktūros maršrutų pritaikymas programiniam kodui.

Algoritmo fragmentas	`[template public generateRouteClassDeclarations(c:Class) ? (c.hasStereotype('LaravelRoutes'))]`
	`[file ('routes/web.php', true, 'UTF-8')]`
	`[for (ass:Association | c.getAssociations()) ]`
	`[for (rel:Class | ass.relatedElement) ]`
	`[if (rel.hasStereotype('LaravelModel'))]`
	`[if (rel.name <> c.name and c.name.startsWith(rel.name))]`
	
	`[for (op:Operation | c.getOperations()) ]`
	`[if (op.hasStereotype('RouteGET'))]`
	`Route::get('[rel.name.toLower()/]s[if (op.getOwnedMembers()->notEmpty())]/['{'+ op.getOwnedMembers().name + '}'/][/if]','[c.name/]@[op.name/]');`
	`[/if]`
	`[if (op.hasStereotype('RoutePOST'))]`
	`Route::post('[rel.name.toLower()/]s','[c.name/]@[op.name/]');`
	`[/if]`
	`[if (op.hasStereotype('RoutePUT'))]`
	`Route::put('[rel.name.toLower()/]s[if (op.getOwnedMembers()->notEmpty())]/['{'+ op.getOwnedMembers().name + '}'/][/if]','[c.name/]@[op.name/]');`
	`[/if]`
	`[if (op.hasStereotype('RouteDELETE'))]`
	`Route::get('[rel.name.toLower()/]s[if (op.getOwnedMembers()->notEmpty())]/['{'+ op.getOwnedMembers().name + '}'/][/if]','[c.name/]@[op.name/]');`
	`[/if]`
	`[/for]`
	`[/if]`
	`[/if]`
	`[/for]`
	`[/for]`
	`[/file]`
	`[/template]`

PSM pavyzdys	1. Klasė turinti LaravelController stereotipą, šios klasės operacija turi priskirtą RouteGET stereotipą, sekų diagramoje esantys pranešimai, jų operacijos ir (jei egzistuoja) kintamieji:  2. Klasė turinti LaravelController stereotipą, šios klasės operacijos turinčios RoutePOST stereotipą, sekų diagramoje esantys pranešimai, jų operacijos:  3. Klasė turinti LaravelController stereotipą, šios klasės operacijos turinčios RoutePUT stereotipą, sekų diagramoje esantys pranešimai, jų operacijos ir kintamieji:  4. Klasė turinti LaravelController stereotipą, šios klasės operacijos turinčios RouteDELETE stereotipą, sekų diagramoje esantys pranešimai, jų operacijos ir kintamieji: 
Pavyzdinis programinis kodas	1.web.php failo papildymas: Route::get('/klasė/metodas, 'KlasėController@metodas); arba: Route::get('/klasė/metodas/{kintamasis}', 'KlasėController@metodas'); 2. web.php failo papildymas: `Route::post('/klasė/metodas, 'KlasėController@metodas);` 3. web.php failo papildymas: `Route::put('/klasė/metodas/{kintamasis}', 'KlasėController@metodas);` 4. web.php failo papildymas: `Route::get('/klasė/metodas/{kintamasis}, 'KlasėController@metodas);`

Šiame algoritmo fragmente randamos klasės turinčios Laravel Routes stereotipą, jų operacijos ir operacijų atributai. Rastų klasių operacijoms vykdomas patikrinimas, kokį stereotipą jos turi. Pagal turimo stereotipo tipą papildomas failas saugantis Laravel maršrutus su Laravel specifiniais maršrutais iš klasių operacijų tipo.

Po įvykdytos transformacijos iš PSM į programinį kodą programuotojas gali išsikelti iš transformacijų aplinkos Laravel struktūros programinį kodą ir jį įkelti į Laravel projektą. Gautus PHP Laravel programinio kodo fragmentus programuotojas gali pabaigti programuoti, norėdamas pilnai realizuoti sumodeliuotą sistemą.

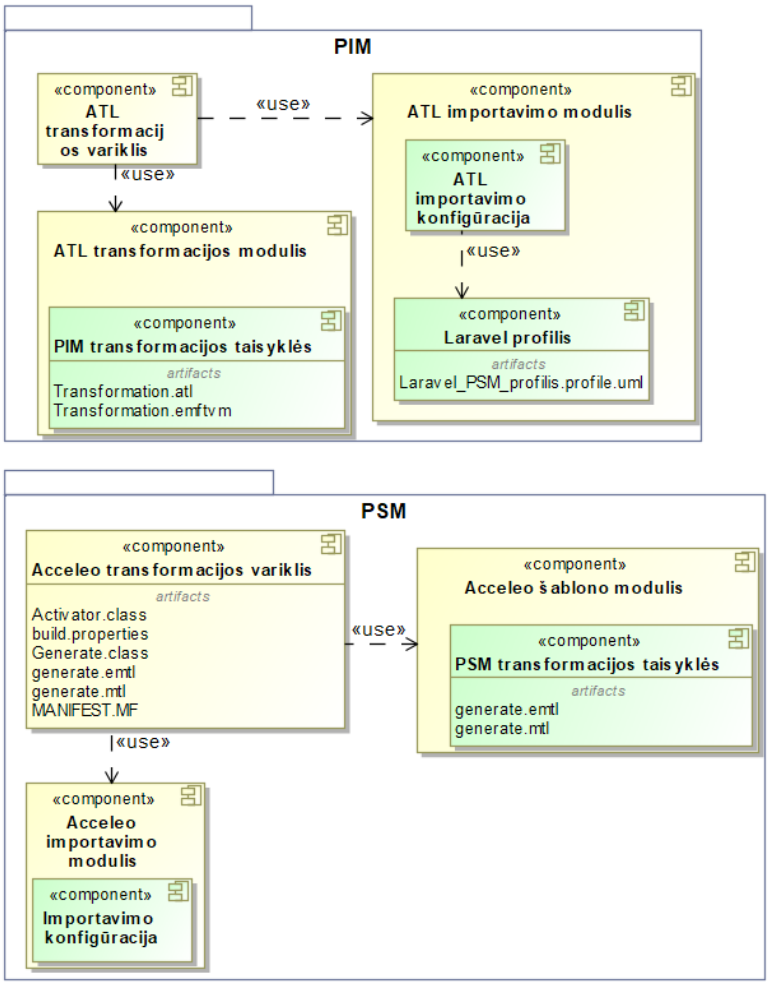
2.4. Reikalavimų apibendrinimas

Sprendimo specifikacijoje buvo apibrėžti iš PIM į PSM ir iš PSM į PHP Laravel programinį kodą transformacijoms aktualūs klasių ir sekų PIM ir PSM metamodelių fragmentai, formaliai apibrėžta metodika apimanti PIM ir PSM modelių struktūros specifikacijas, išnagrinėtos galimybės transformacijoms panaudoti ne tik struktūrinio tipo klasių diagramą, tačiau ir elgsenos tipo sekų diagramą. Metodikai taip pat buvo paruoštas PHP programavimo kalbos Laravel karkasui pritaikytas profilis, apibrėžti algoritmų aprašai ir pavyzdiniai modelių ir kodo fragmentai.

3. Modeliais grindžiamos PHP programinio kodo generavimo metodikos eksperimentinės realizacijos projektas

3.1. Detalus projektas

Detalus transformacijos iš modelio į modelį ir iš modelio į PHP Laravel programinį kodą eksperimentinės realizacijos projektas susideda iš kelių dalių, kurioms reikalinga Eclipse programinė įranga ir du šios programos įrankiai – Eclipse ATL ir Eclipse Acceleo. Detalaus projekto komponentų diagrama pateikiama 27 pav. Žalia spalva pažymėti komponentai yra realizuotos metodikos projekto indėlis į transformacijos procesą, naudojant šiuos įrankius. Geltona spalva pažymėti komponentai yra įrankių sudedamieji komponentai.

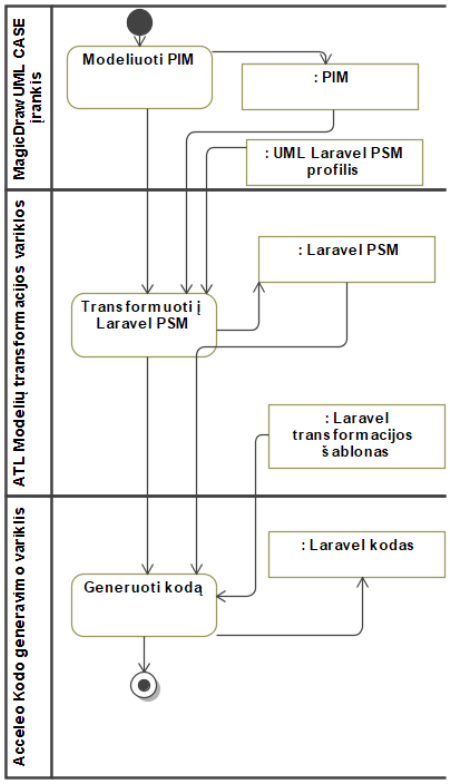


27 pav. Detalaus metodikos projekto komponentų diagrama

Pirmoji dalis, su kuria atliekama transformacija iš modelio į modelį, komponentų diagramoje pateikiama PIM pakete. Šiame pakete konkretizuojami Eclipse ATL įrankio esminiai komponentai. ATL įrankį iš esmės sudaro ATL transformacijų variklis, ATL importavimo modulis ir ATL transformacijų modulis. ATL importavimo modulis naudoja ATL importavimo konfigūraciją ir papildomus transformacijai aktualius profilius. Šios metodikos realizacijai, buvo paruošta ir atlikta importavimo konfigūracija, nurodant importavimo taisykles, parenkant atitinkamą UML metamodelį, transformacijos virtualią mašiną ir įkeliant bei importuojant transformacijai reikalingą Laravel profilį.

Antroji dalis, su kuria atliekama transformacija iš modelio į programinį kodą, pateikiama PSM pakete. Acceleo įrankio esminiai komponentai yra pats Acceleo transformacijos variklis, Acceleo importavimo modulis ir šablono modulis. Acceleo transformacijos variklis naudoja importavimo modulį ir šablono modulį. Šios metodikos realizacijai buvo paruošta importavimo konfigūracija, nurodanti transformacijos virtualią mašiną, aplinką ir reikalingą UML metamodelį. Taip pat paruoštos ir pritaikytos transformacijai specifinės PSM transformacijos taisyklės, skirtos PHP Laravel kodui generuoti.

Metodikoje, modeliavimo ir kodo generavimo procesas 28 pav. susideda iš trijų pagrindinių žingsnių: PIM modeliavimas, transformacija į PSM ir PHP Laravel kodo generavimas.



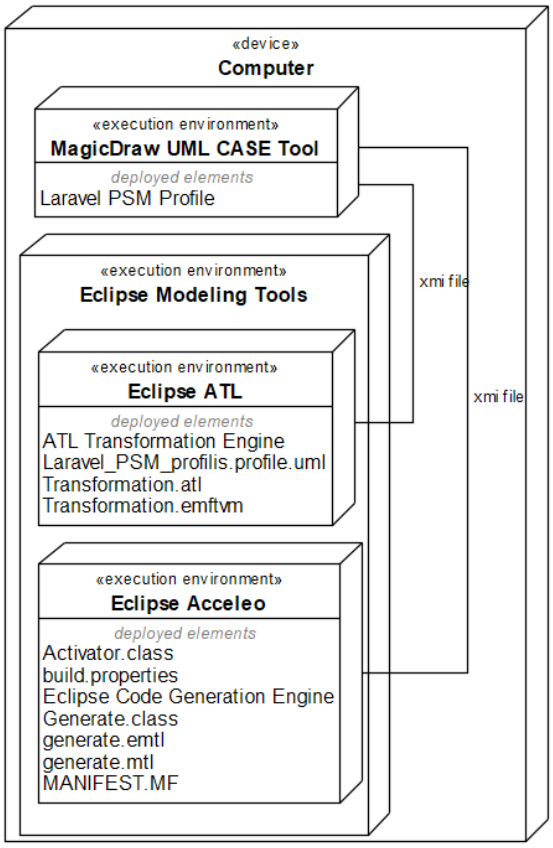
28 pav. Abstraktus modeliavimo ir kodo generavimo procesas

Pirmajame PIM modeliavimo žingsnyje, modelis sudaromas naudojant MagicDraw įrankį, modelyje sudaromos klasių diagrama ir sekų diagramos skirtos apibrėžti operacijų tipus. Po modelio sudarymo, pereinama į antrąjį žingsnį, tai transformacija į Laravel PSM, kurią atlieką ATL modelių transformacijos variklis pagal nurodytą transformacijos algoritmą. Transformacijai reikalingas sudarytas PIM modelis ir Laravel PSM profilis. Po transformacijos pereinama į trečią žingsnį – specifinio Laravel programinio kodo generavimą. Šiam žingsniui reikalingas iš ankstesnės transformacijos gautas Laravel platformai specifinis modelis. Transformacija atliekama su Acceleo kodo generavimo įrankiu, naudojant aprašytą Laravel transformacijų šabloną.

4. Sprendimo realizacija ir testavimas

4.1. Sprendimo realizacijos ir veikimo aprašas

Metodikos realizavimo sprendimas, siekiant sumodeliuoti ir sugeneruoti informacinės sistemos kodą pateikiamas 29 pav. Metodikos sprendimas yra diegiamas kompiuteryje, kuriame turi būti įdiegta MagicDraw ir Eclipse programinė įranga. MagicDraw ir Eclipse ATL įrankiuose svarbu įkelti Laravel specifinį profilį ir nurodyti įrankiams šio profilio lokaciją. Diegimo diagramoje pateikiami ryšiai su „xmi file“ pavadinimu nurodo galimybę asmenims, kuriantiems informacines sistemas, importuoti ar eksportuoti XMI formatu modelio failus iš vienos siūlomos platformos į kitą.



29 pav. Diegimo diagrama apibrėžianti modeliavimui ir generavimui siūlomus įrankius

Naudojant MagicDraw CASE įrankį, asmuo kuriantis informacines sistemas, turi sudaryti PIM. Baigus modeliavimą, modelis išsaugomas Eclipse XMI formatu siekiant atlikti transformaciją iš PIM į PSM (M2M transformacija). Po transformacijos iš modelio į modelį, jei yra poreikis atlikti pakeitimus, ar peržiūrėti PSM, galima PSM Eclipse XMI formatu importuoti atgal į MagicDraw CASE įrankį. Atlikus norimus pakeitimus PSM, šį modelį reikia išsaugoti Eclipse XMI formatu siekiant atlikti transformaciją iš PSM į programinį kodą (M2T transformacija).

Transformacija iš PIM į PSM (M2M transformacija) atliekama naudojant Eclipse ATL įrankį ir transformacijos taisyklėmis grįstą algoritmą kuris apibrėžtas ATL transformacijų kalba. ATL transformacijų kalba leidžia apibrėžti trijų rūšių transformacijos dalis: ATL transformacijų modulius, ATL užklausas ir ATL bibliotekas. Be viso to, šias rūšis papildyti ir sudaryti gali ATL pagalbinės funkcijos, atributai, atitikties ir iškviečiamosios taisyklės. Pats ATL įrankis suteikia galimybę

transformuoti norimą kiekį modelių iš norimo kiekio modelių apibrėžiant transformacijos taisyklėmis grįstą algoritmą [18].

Transformacija iš PSM į programinį kodą (M2T transformacija) atliekama naudojant Eclipse Acceleo įrankį ir šablonu grįsta Acceleo kalbą. Siekiant sugeneruoti programinį kodą iš PSM, XMI formato PSM turi būti importuojamas į Eclipse Acceleo įrankį ir pasirenkama transformavimo parinktis.

Norint naudoti modeliais grindžiamos PHP programinio kodo generavimo metodikos realizacija svarbu atsižvelgti į tam tikrus failus ATL ir Acceleo įrankiuose. Transformacijos atveju naudojant ATL įrankį, norint atlikti transformaciją svarbu, kad į projekto aplanką būtų įkeltas ir nurodytas Transformation.atl failas. Šiame faile pateikiamas visas transformacijos iš PIM į Laravel PSM algoritmas. Transformation.emftvm failas yra sukompiliuota Transformation.atl failo versija, skirta ATL EMF transformacijos virtualiai mašinai. Su šiais dviem failais esančiais projekte, nurodžius transformacijos konfigūraciją, galima atlikti transformaciją iš PIM į PSM. Transformacijos atveju naudojant Acceleo įrankį, svarbu užtikrinti, kad Acceleo projekte būtų Activator.class, buid.properties, Generate.class, MANIFEST.MF failai. Taip pat būtina įkelti pilną transformacijos algoritmą iš PSM į PHP Laravel programinį kodą saugantį generate.mtl failą ir jo generate.emtl sukompiliuotą failo versiją. Transformacijos paleidimas konfigūracijoje turi būti nurodomas kaip Java Application. Java Application ir Acceleo transformacijos aplinkos konfigūracijos paruošimą galima rasti oficialioje Acceleo dokumentacijoje, 1.6.2 skyriuje [25].

4.2. Testavimo modelis, duomenys, rezultatai

Sprendimo testavimui pasirinkta sudaryti rankinio testavimo scenarijus ir patikrinti metodikos sprendimo veikimą transformuojant mažesnės ir didesnės apimties modelį į programinį kodą. Sprendimas buvo testuojamas naudojant UML diagramas, sprendimo testavimui pasirinkta sudaryti klasių ir sekų diagramas. Sprendimo testavimui naudotas mažesnės apimties modelis ir didesnės apimties modelis. Mažesnės apimties modelio klasių diagramos fragmentą galima peržiūrėti 4 priede. Modelių klasių, operacijų, atributų, asociacijų ir sekų diagramų kiekiai detalizuojami 13 lentelėje.

13 lentelė. Testavimo modelių apimtys

Modelis	**Klasių skaičius**	**Operacijų skaičius**	**Atributų skaičius**	**Asociacijų skaičius**	**Sekų diagramų skaičius**
Mažesnės apimties modelis	12	12	10	12	12
Didesnės apimties modelis	71	73	116	73	73

Numatytas rankinio testavimo scenarijus modelio transformacijai iš PIM į PSM pateikiamas 14 lentelėje.

14 lentelė. Transformacijos iš PIM į PSM rankinio testavimo scenarijus

Rankinio testavimo scenarijus: Sėkmingai transformuoti modelį iš PIM į PSM	
Prieš-sąlyga	PIM išeksportuotas Eclipse UML XMI formatu
Po-sąlyga	PSM išeksportuotas Eclipse UML XMI formatu
Scenarijus	

14 lentelė. Transformacijos iš PIM į PSM rankinio testavimo scenarijus

Nr.	Naudotojo veiksmai	Laukiamas rezultatas iš ATL įrankio veiksmai	Gautas rezultatas iš ATL įrankio
1.	Naudotojas pasirenka transformacijos konfigūracijos parinktį.	ATL įrankis atidaro konfigūracijos langą.	ATL įrankis atidaro konfigūracijos langą.
2.	Naudotojas nurodo transformuojamo modelio lokaciją, UML metamodelį, Laravel profilį, išsaugo pakeitimus ir paleidžia transformaciją.	Įrankis pagal aprašytas transformacijos taisykles sėkmingai įvykdo transformaciją ir išsaugo PSM Eclipse UML XMI formatu.	Įrankis pagal aprašytas transformacijos taisykles sėkmingai įvykdo transformaciją ir išsaugo PSM Eclipse UML XMI formatu.

Transformacijos iš PIM į PSM testavimo metu pastebėtas trūkumas, kad ATL įrankio transformacijų variklis neaptinka modelio Entity stereotipo. Ši problema buvo išspręsta, nurodant transformacijos taisyklėse aptikti visus Boundary ir Control stereotipus, o klases, likusias be stereotipo, interpretuoti kaip Entity klasę ir pritaikyti jai reikalingus Laravel specifinius stereotipus. Šis sprendimas yra tinkamas, nes šioje metodikoje sudarant PIM klasių diagramą negali būti klasių be stereotipų ir turi būti naudojami tik Boundary, Control ir Entity stereotipai, pagal MVC architektūros principus. 15 lentelėje pateikiamas nesėkmingos ATL konfigūracijos scenarijus.

15 lentelė. Nesėkmingos ATL transformacijos konfigūracijos rankinio testavimo scenarijus

Rankinio testavimo scenarijus: Nesėkmingai sukonfigūruoti ATL transformacijos konfigūraciją			
Prieš-sąlyga	Atidaryta ATL transformacijos konfigūracijos aplinka		
Po-sąlyga	Nesėkminga transformacija iš PIM į PSM		
Scenarijus			
Nr.	**Naudotojo veiksmai**	**Laukiamas rezultatas iš ATL įrankio**	**Gautas rezultatas iš ATL įrankio**
1.	Naudotojas užpildo nevisus reikiamus konfigūracijos laukus (praleidžia profilį, metamodelį, transformuojamą modelį).	ATL įrankis neleidžia paleisti transformacijos.	ATL įrankis neleidžia paleisti transformacijos.
2.	Naudotojas užpildo visus reikiamus konfigūracijos laukus, tačiau pasirenka ne EMFTVM transformavimo virtualią mašiną.	ATL įrankis pateikia klaidos pranešimą apie neegzistuojantį sukompiliuotą transformacijos failą.	ATL įrankis pateikia klaidos pranešimą apie neegzistuojantį sukompiliuotą transformacijos failą.

Šis nesėkmingos transformacijos su ATL įrankiu scenarijus įvykdytas sėkmingai. Naudotojui nepilnai užpildžius reikiamus konfigūracijos laukus, ATL įrankis neleido atlikti transformacijos. Naudotojui užpildžius visus laukus, tačiau pasirinkus netinkamą transformacijos virtualią mašiną, įrankis pateikė klaidos pranešimą apie neegzistuojantį sukompiliuotą transformacijos failą. Toliau, 16 lentelėje pateikiamas rankinio testavimo scenarijus modelio transformacijai iš PSM į PHP Laravel programinį kodą.

16 lentelė. Transformacijos iš PSM į PHP Laravel programinį kodą rankinio testavimo scenarijus

Rankinio testavimo scenarijus: Sėkmingai transformuoti modelį iš PSM į PHP Laravel programinį kodą.	
Prieš-sąlyga	PSM išeksportuotas Eclipse UML XMI formatu
Po-sąlyga	Sugeneruotas PHP Laravel programinis kodas

16 lentelė. Transformacijos iš PSM į PHP Laravel programinį kodą rankinio testavimo scenarijus

Scenarijus		
Naudotojo veiksmai	**Laukiamas rezultatas iš Acceleo įrankio**	**Gautas rezultatas iš Acceleo įrankio**
Naudotojas pasirenka transformacijos konfigūracijos parinktį.	Acceleo įrankis atidaro konfigūracijos langą.	Acceleo įrankis atidaro konfigūracijos langą.
Naudotojas nurodo transformuojamo modelio lokaciją, UML metamodelį, programinio kodo išsaugojimo lokacija ir paleidžia transformaciją.	Įrankis pagal aprašytą transformacijos šabloną sėkmingai įvykdo transformaciją ir išsaugo PHP Laravel programinį kodą į naudotojo nurodytą aplanką.	Įrankis pagal aprašytą transformacijos šabloną sėkmingai įvykdo transformaciją ir išsaugo PHP Laravel programinį kodą į naudotojo nurodytą aplanką.

Transformacijos iš PSM į PHP Laravel programinį kodą eigoje trūkumų nebuvo pastebėta. 17 lentelėje pateikiamas nesėkmingos Acceleo įrankio konfigūracijos rankinio testavimo scenarijus.

17 lentelė. Nesėkmingos Acceleo transformacijos konfigūracijos rankinio testavimo scenarijus

Rankinio testavimo scenarijus: Nesėkmingai sukonfigūruoti Acceleo transformacijos konfigūraciją			
Prieš-sąlyga	Atidaryta Acceleo transformacijos konfigūracijos aplinka		
Po-sąlyga	Nesėkminga transformacija iš PSM į programinį kodą		
Scenarijus			
Nr.	**Naudotojo veiksmai**	**Laukiamas rezultatas iš Acceleo įrankio**	**Gautas rezultatas iš Acceleo įrankio**
1.	Naudotojas užpildo nevisus reikiamus konfigūracijos laukus (praleidžia pagrindinę klasę, modelį, generavimo lokaciją).	Acceleo įrankis neleidžia paleisti transformacijos.	Acceleo įrankis neleidžia paleisti transformacijos.
2.	Naudotojas užpildo visus reikiamus konfigūracijos laukus, tačiau pasirenka transformacijos paleidimą ne kaip Java Application, o Acceleo plug-in application.	Acceleo įrankis pateikia klaidos pranešimą apie negalimą transformaciją.	Acceleo įrankis pateikia klaidos pranešimą apie negalimą transformaciją.

Nesėkmingos Acceleo konfigūracijos scenarijus įvykdytas sėkmingai. Naudotojui nepilnai užpildžius reikiamus konfigūracijos laukus, Acceleo įrankis neleido atlikti transformacijos. Užpildžius visus laukus, tačiau pasirinkus netinkamą transformacijos paleidimo tipą, naudotojui Acceleo įrankis pateikė klaidos pranešimą apie negalimą transformacijos tipą.

Įvykdžius visus sėkmingo testavimo scenarijus, mažesnio modelio atveju, M2M transformacijos metu pritaikyti 5 Laravel specifiniai stereotipai skirti klasėms, jie pritaikyti 12 klasių, taip pat pritaikyti 4 specifiniai stereotipai skirti klasių operacijoms, jie pritaikyti 14 operacijų. M2T transformacijos metu, sėkmingai sukurta aplankų struktūra, sugeneruota 15 failų kodui, 14 metodų, pritaikyti 4 panaudojimai pagal asociacijas, sugeneruota 10 kintamųjų, pagal M2M iš 14 sekų veiksmų tipų priskirtus stereotipus M2T sugeneruoti visi 14 maršrutų ir sukurti atitinkami metodai, taip pat bendrai sugeneruota 276 kodo eilučių. Didesnio modelio, kuris buvo modeliuotas pagal veikiančia sistemos nepriklausomą funkcionalumo rinkinį atveju, įvykdžius abu testavimo scenarijus, pritaikyti 5 Laravel specifiniai stereotipai skirti klasėms, jie pritaikyti 71 klasei, taip pat pritaikyti 4 specifiniai stereotipai

skirti klasių operacijoms, jie pritaikyti viso 73 operacijoms. MT2 transformacijos metu, sukurta aplankų struktūrą ir joje sugeneruoti viso 84 programinio kodo failai, 138 metodai, pritaikyti 26 panaudojimai pagal asociacijas, sugeneruota 116 kintamųjų. Pagal M2M iš 73 sekų veiksmų tipų priskirtus stereotipus M2T sugeneruoti visi 73 maršrutai ir sukurti atitinkami metodai, taip pat bendrai sugeneruota 2699 kodo eilučių. Mažesnio modelio testavimo atveju, sugeneruotų PHP Laravel kodo fragmentų pavyzdžiai pateikiami 4 priede.

Metodikos sprendimo pagal rankinio testavimo scenarijus metu, atliekant transformacijas su mažesnės apimties ir didesnės apimties modeliais su nesklandumais beveik nebuvo susidurta. Testavimo metu aptiktas vienas nesklandumas, dėl Entity stereotipo neradimo M2M transformacijos metu, tačiau ši problema buvo išspręsta.

5. Eksperimentinis sprendimo tyrimas

5.1. Eksperimento planas

Sukurtai MDA architektūra grindžiamai metodikai, apimančiai modelių transformacijas ir PHP Laravel kodo generavimą, įvertinti atliekamas eksperimentas, kurio tikslas – nustatyti, ar modelių transformacijos ir kodo generavimas PHP Laravel karkasui palengvintų informacinių sistemų, realizuojamu šiuo karkasu, kūrimą. Eksperimentas sudarytas iš keturių dalių:

- Pirmojoje dalyje atliekama tyrimo objekto naudotojų apklausa, pateikiant klausimus apie sugeneruojamo kodo korektiškumą ir naudą. Apklausoje respondentams buvo pateikta viso 12 klausimų. Dalies apklausos klausimų teiginių įvertinimui pasirinkta naudoti Likerto skalę, kuri yra skirta vertinti respondentų nuomonei, suteikia galimybę respondentams pateikti savo sutikimo / nesutikimo su teiginiais laipsnį [26]. Likerto skalės statistinės analizės vidurkių, medianos ir modos apskaičiavimui pasirinktą naudoti SPSS statistinės analizės įrankį, naudojamą statistinei analizei atlikti [29]. Kiti klausimai skirti išsiaiškinti asmenų veiklos sritis, įvertinti respondentų patirtį su PHP programavimo kalba, Laravel karkasu, UML modelių sudarymu ir pasiteirauti respondentų, ar jie mano, kad modeliavimas ir kodo generavimas galėtų sumažinti technologinius migracijos kaštus ir kūrimo laiko kaštus kuriant informacines sistemas.

- Antrojoje dalyje atliekamas rankinis naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas, lyginant juos pagal įvairius kriterijus su veikiančios sistemos programiniu kodu.

- Trečiojoje ir ketvirtojoje dalyse atliekamas automatinis programinio kodo palyginimas, lyginant naują Laravel projektą ir iš modelio sugeneruotą kodą pagal įvairius kriterijus su veikiančios sistemos funkcionalumo rinkinio programiniu kodu. Veikianti sistema – realizuota draudimo paslaugas teikiančios įmonės informacinė sistema. Šios sistemos modelis pateikiamas 47 pav. Automatiniam programinio kodo palyginimui buvo naudoti du įrankiai: LaravelStats ir PHPLoc. LaravelStats yra įrankis skirtas analizuoti Laravel projekto sandarą ir parametrus. Šis įrankis gali pateikti kodo eilučių, testų kodo eilučių kiekį, klasių ir metodų skaičių ir santykį tarp klasių ir realizuojamų metodų [24]. PHPLoc įrankis skirtas išmatuoti ir analizuoti PHP projekto struktūrą. Šis įrankis gali pateikti: bendrą kodo eilučių skaičių, užkomentuotų kodo eilučių skaičių, neužkomentuotų kodo eilučių skaičių, loginių kodo eilučių skaičių, klasių skaičių, metodų skaičių, failų tipų ir struktūrų skaičių ir kitus aktualius PHP projekto parametrus [29].

Eksperimentą pasirinkta sudaryti viso iš keturių dalių, siekiant apžvelgti realizuotą metodiką skirtingais aspektais. Realizuotos metodikos apžvelgimas įvairiais aspektais padeda nustatyti ar modelių transformacijos ir kodo generavimas PHP Laravel karkasui palengvintų informacinių sistemų realizuojamu šia programavimo kalba kūrimą.

5.2. Eksperimento rezultatai

5.2.1. Tyrimo objekto naudotojų apklausa

Tyrimo objektų naudotojų apklausa organizuota apklausiant programuotojus ir analitikus apie sugeneruojamo kodo korektiškumą ir naudą. Apklausoje dalyvavo 53 respondentai (50 programuotojų, 3 analitikai). Dalis respondentams užduotų klausimų apie jų žinias, patirtį ir kt.

konkretizuojami 6 priede. Kitoje apklausos klausimų dalyje, respondentų buvo paprašyta įvertinti tris teiginius:

1. Jei kodo generavimo įrankis iš modelio sugeneruotų pavyzdyje pateiktą kodą, tai mane paskatintų naudoti UML modelius kuriant sistemas su Laravel karkasu.
2. Iš modelio sugeneruoto kodo fragmentas atitinka Laravel specifiką ir yra tinkamas naudoti kuriant sistemas su Laravel karkasu.
3. Pavyzdyje pateikto programinio kodo generavimas palengvintų sistemos kūrimą ir tolimesnį plėtojimą.

Respondentams pateikti klasių diagramų ir kodo fragmentų pavyzdžiai šiems teiginiams pateikiami 6 priede. Teiginiai turėjo būti įvertinami, pagal sugeneruoto kodo fragmentų pavyzdžius Laravel Model, Controller, Request, Routes klasėms. Teiginių įvertinimui pasirinkta naudoti Likerto skalę, o vidurkių, medianų ir modų apskaičiavimui SPSS statistinės analizės įrankį, naudojamą statistinei analizei atlikti [29]. Teiginių atsakymai yra vertinami taip, kad atitiktų kryptingumą – kiekvieno teiginio vertinimas gerėja, jei auga respondentų sutikimo kiekis. Šiai apklausai sudarytoje Likerto skalėje, respondentas vertinamus teiginius galėjo įvertinti pasirinkdamas vieną iš penkių galimų pasirinkimų – „Visiškai nesutinku“, „Nesutinku“, „Nei sutinku, nei nesutinku“, „Sutinku“ ir „Visiškai sutinku“. Respondentų atsakymams buvo priskiriami taškai: visiškai nesutinku variantui priskirtas 1 taškas, nesutinku variantui priskirti 2 taškai, nei sutinku nei nesutinku variantui priskirti 3 taškai, sutinku variantui priskirti 4 taškai, o visiškai sutinku variantui priskirti 5 taškai. Respondentų atsakymai į pateiktus teiginius pateikiami 6 priede. Toliau pateikiami šių atsakymų rezultatai. Pirmasis kodo fragmentas skirtas teiginių įvertinimui buvo Laravel Model. Kodo fragmento įvertinimo rezultatai pateikiami 18 lentelėje.

18 lentelė. Klausimo apie Laravel Model kodo fragmentą teiginių įvertinimo rezultatai

Teiginys	**Vidurkis**	**Mediana**	**Moda**
Jei kodo generavimo įrankis iš modelio sugeneruotų pavyzdyje pateiktą kodą, tai mane paskatintų naudoti UML modelius kuriant sistemas su Laravel karkasu.	3.21	3	4
Iš modelio sugeneruoto kodo fragmentas atitinka Laravel specifiką ir yra tinkamas naudoti kuriant sistemas su Laravel karkasu.	3.77	4	4
Pavyzdyje pateikto programinio kodo generavimas palengvintų sistemos kūrimą ir tolimesnį plėtojimą.	3.40	3	3

Atsakydami į klausimus apie pirmąjį kodo fragmentą, respondentai buvo linkę sutikti su visais trimis teiginiais. Pagal Likerto skalę įvertinus teiginių vidurkius pirmojo teiginio vidurkio rezultatas yra 3.21, antrojo teiginio rezultatas 3.77, o trečiojo teiginio vidurkis 3.40.

Antrasis kodo fragmentas skirtas teiginių įvertinimui buvo Laravel Controller. Kodo fragmento įvertinimo rezultatai pateikiami 19 lentelėje.

19 lentelė. Klausimo apie Laravel Controller kodo fragmentą teiginių įvertinimo rezultatai

Teiginys	**Vidurkis**	**Mediana**	**Moda**
Jei kodo generavimo įrankis iš modelio sugeneruotų pavyzdyje pateiktą kodą, tai mane paskatintų naudoti UML modelius kuriant sistemas su Laravel karkasu.	3.50	4	4
Iš modelio sugeneruoto kodo fragmentas atitinka Laravel specifiką ir yra tinkamas naudoti kuriant sistemas su Laravel karkasu.	3.98	4	5
Pavyzdyje pateikto programinio kodo generavimas palengvintų sistemos kūrimą ir tolimesnį plėtojimą.	3.64	4	4

Atsakydami į klausimus apie Laravel Controller kodo fragmentą, respondentai buvo linkę sutikti su visais trimis teiginiais. Įvertinus vidurkius, atsakiusieji labiausiai buvo linkę sutikti su antru teiginiu, kurio vidurkis 3.98, tačiau taip pat sutiko ir su pirmu teiginiu kurio vidurkis 3.5 ir trečiuoju teiginiu, kurio vidurkis 3.64. Dažniausiai pasikartojanti reikšmė teiginiams buvo sutikimo ir visiško sutikimo, o skaičių eilės vidurinis elementas gautas taip pat sutikimo.

Trečiasis kodo fragmentas skirtas teiginių įvertinimui buvo Laravel Request. Kodo fragmento įvertinimas pateikiamas 20 lentelėje.

20 lentelė. Klausimo apie Laravel Request kodo fragmentą teiginių įvertinimo rezultatai

Teiginys	**Vidurkis**	**Mediana**	**Moda**
Jei kodo generavimo įrankis iš modelio sugeneruotų pavyzdyje pateiktą kodą, tai mane paskatintų naudoti UML modelius kuriant sistemas su Laravel karkasu.	3.40	4	4
Iš modelio sugeneruoto kodo fragmentas atitinka Laravel specifiką ir yra tinkamas naudoti kuriant sistemas su Laravel karkasu.	3.79	4	4
Pavyzdyje pateikto programinio kodo generavimas palengvintų sistemos kūrimą ir tolimesnį plėtojimą.	3.55	4	4

Atsakydami į klausimus apie trečiąjį kodo fragmentą, respondentai buvo taip pat linkę sutikti su visais trimis teiginiais. Pirmojo teiginio apskaičiuotas vidurkis yra 3.4, antrojo teiginio 3.89, trečiojo teiginio 3.55. Skaičių eilės vidurinis elementas ir dažniausiai pasikartojanti reikšmė nurodo pasiskirstymą sutikimui.

Ketvirtasis kodo fragmentas skirtas teiginių įvertinimui buvo Laravel Routes. Kodo fragmento įvertinimo rezultatai pateikiami 21 lentelėje.

21 lentelė. Klausimo apie Laravel Routes kodo fragmentą teiginių įvertinimo rezultatai

Teiginys	**Vidurkis**	**Mediana**	**Moda**
Jei kodo generavimo įrankis iš modelio sugeneruotų pavyzdyje pateiktą kodą, tai mane paskatintų naudoti UML modelius kuriant sistemas su Laravel karkasu.	3.25	3	3
Iš modelio sugeneruoto kodo fragmentas atitinka Laravel specifiką ir yra tinkamas naudoti kuriant sistemas su Laravel karkasu.	3.68	4	4
Pavyzdyje pateikto programinio kodo generavimas palengvintų sistemos kūrimą ir tolimesnį plėtojimą.	3.45	4	4

Atsakydami į klausimus apie Laravel Routes kodo fragmentą, respondentai buvo linkę sutikti su pateiktais trimis teiginiais. Apskaičiuotas pirmojo teiginio vidurkis yra 3.25, antrojo teiginio 3.68, trečiojo teiginio 3.45. Apskaičiavus klausimų vidurkius kiekvienam iš klausimų, 22 lentelėje**.** pateikiama visų klausimų vidurkių statistika.

22 lentelė. Klausimų vidurkių įvertinimo pagal Likerto skalę apibendrinta vidurkių statistika

Klausimas	**Bendras klausimo vidurkis**
Klausimas apie Laravel Model	3.46
Klausimas apie Laravel Controller	3.71
Klausimas apie Laravel Request	3.58
Klausimas apie Laravel Routes	3.46

Klausimų vidurkių statistikoje kiekvienam iš klausimų galima įžvelgti, kad respondentai linkę sutikti su kiekvieno iš kodo fragmentų nauda. Bendrai apskaičiavus kodo fragmentų klausimų teiginių vidurkius, 23 lentelėje**.** pateikiama visų teiginių tipų atsakymų statistika.

23 lentelė. Teiginių vidurkių įvertintų pagal klausimų teiginius Likerto skalę statistika

Teiginys	**Bendras vidurkis**
Jei kodo generavimo įrankis iš modelio sugeneruotų pavyzdyje pateiktą kodą, tai mane paskatintų naudoti UML modelius kuriant sistemas su Laravel karkasu.	3.34
Iš modelio sugeneruoto kodo fragmentas atitinka Laravel specifiką ir yra tinkamas naudoti kuriant sistemas su Laravel karkasu.	3.81
Pavyzdyje pateikto programinio kodo generavimas palengvintų sistemos kūrimą ir tolimesnį plėtojimą.	3.51

Apibendrinant tyrimo objekto naudotojų apklausos rezultatus galima teigti, kad respondentai dalyvavę apklausoje, labiau linkę sutikti, kad modelių transformacijos ir kodo generavimas galėtų sumažinti technologinius migracijos kaštus ir kūrimo laiko kaštus kuriant informacines sistemas. Taip pat, respondentai linkę sutikti su visais trimis teiginiais, kurie buvo įvertinami pagal Likerto skalę. Labiausiai respondentai sutiko su kodo korektiškumo teiginiu, toliau, su sistemų kūrimo ir tolimesnio plėtojimo palengvinimo teiginiu ir mažesnė dalis sutiko su teiginiu, kad sugeneruojami kodo pavyzdžiai paskatintų juos naudoti UML modelius kuriant sistemas su Laravel karkasu.

5.2.2. Rankinis naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas

Rankinis naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas vykdytas rankiniu būdu. Palyginimui paruoštas naujas Laravel projektas ir naudojant komandinę eilutę paleistos Artisan komandos, kad sugeneruoti kuo daugiau kodo pagal draudimo paslaugas teikiančios įmonės veikiantį informacinės sistemos funkcionalumo rinkinį, tuomet paruoštas šios draudimo bendrovės veikiančios informacinės sistemos UML modelis ir iš šio modelio sugeneruotas kodas, vėliau atliktas palyginimas. Palyginimui, iš pradžių buvo suskaičiuotas vieno ir kito projekto bendras metodų skaičius, toliau suskaičiuotas veikiančių metodų skaičius, jų procentas, taip pat metodų turinčių tą patį pavadinimą kaip ir realizuotoje sistemoje skaičius ir procentas, bei realizuotų Request validacijos klasių skaičius ir procentas. Veikiantys metodai laikomi metodais, kurie realizavus sistemos dizaino dalį galėtų iš karto atlikti tokias vidines sistemos funkcijas kaip: informacijos pridėjimas, redagavimas, šalinimas ar kt. Visi šie apskaičiuoti parametrai buvo palyginti su jau realizuotos ir veikiančios informacinės sistemos parametrais. Rankinio palyginimo rezultatai pateikiami 24 lentelėje.

24 lentelė. Eksperimento rankinio palyginimo rezultatai

Palyginimas	**Laravel komandinės eilutės projektas**	**Iš modelio sugeneruotas kodas**
Viso metodų sk.	91 (yra perteklinių / nereikalingų metodų)	73
Veikiančių metodų sk.	0/77	27/77
Veikiančių metodų proc.	0%	35.06%

24 lentelė. Eksperimento rankinio palyginimo rezultatai

Palyginimas	Laravel komandinės eilutės projektas	Iš modelio sugeneruotas kodas
Metodų turinčių tą patį pavadinimą kaip ir realizuotoje sistemoje skaičius	71/77 (yra perteklinių / nereikalingų metodų)	73/77
Metodų turinčių tą patį pavadinimą kaip ir realizuotoje sistemoje procentas	92.21%	94.81%
Realizuotų Request validacijos klasių sk.	0/26	26/26
Realizuotų Request validacijos klasių procentas	0%	100%

Rankinio palyginimo metu nustatyta, kad naujas Laravel projektas sugeneruoja 93 metodus, o iš modelio sugeneruotas kodas 73, naujas projektas sugeneruoja 20 metodų daugiau. Tačiau, naujas projektas generuotas su komandine eilute sugeneruoja perteklinius / nereikalingus metodus, kurių nėra veikiančioje informacinėje sistemoje. Toliau, lyginant veikiančius metodus nustatyta, kad naujas Laravel projektas, kurtas su komandine eilute neturi nei vieno iš 77 veikiančios sistemos veikiančių metodų, o iš modelio sugeneruotas programinis kodas turi 27 veikiančius metodus, kas sudaro 35.06%. Palyginus metodų turinčių tą patį pavadinimą kaip ir realizuotoje sistemoje skaičių, nustatyta, kad naujas Laravel projektas turi 71 sutaptį, kas sudaro 92.21 % veikiančios sistemos metodų pavadinimų, tačiau turi perteklinių / nereikalingų metodų. Iš modelio sugeneruotas kodas turi 73 metodų pavadinimų sutaptis, sudarančias 94.81% veikiančios informacinės sistemos. Atlikus realizuotų validacijos klasių palyginimą, nustatyta, kad naujai sukurtas Laravel projektas neturi nei vienos realizuotos validacijos klasės, o iš modelio sugeneruotas kodas turi visas 26 veikiančio projekto realizuotas validacijos klases, kas sudaro 100 %.

5.2.3. Automatinis naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas su LaravelStats įrankiu

Vienas iš automatinių naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas buvo atliktas su LaravelStats įrankiu. Su įrankiu buvo atlikti skaičiavimai naujam Laravel projektui, kuris buvo kurtas su Artisan komandine eilute, iš modelio sugeneruotam Laravel kodui ir pilnai realizuotai draudimo paslaugas teikiančios įmonės informacinės sistemos funkcionalumo rinkiniui kurtam su Laravel karkasu. Iš modelio sugeneruoto kodo modelio klasių diagrama, maža dalis iš modelio sugeneruotų kodo fragmentų ir maža dalis veikiančios sistemos nepriklausomo funkcionalumo rinkinio kodo fragmentų pateikiami 7 priede esančiose lentelėse ir paveiksle. Kiekvienam iš trijų projektų tipų buvo skaičiuojamas Controllers, Models ir Requests klasių skaičius, metodų vidurkis esantis šiose klasėse, kodo eilučių skaičius ir loginių kodo eilučių skaičius. Loginių kodo eilučių skaičius, tai skaičius nurodantis vykdomų kodo eilučių kiekį, tačiau neįskaičiuojant antraščių, komentarų ir nevykdomo kodo. Pačios vykdomos kodo eilutės nurodo interpretatoriui (angl. Parser) kokį veiksmą reikia atlikti. Naujo Laravel projekto sugeneruotos klasės ir kodo eilučių kiekis pateikiami 25 lentelėje.

25 lentelė. LaravelStats įrankiu apskaičiuotos naujo Laravel projekto sugeneruotos klasės ir kodo eilutės

Tipas	**Klasių sk.**	**Metodai klasėje (vidurkis)**	**Kodo eilučių sk.**	**Loginių kodo eilučių sk.**
Controllers*	13	7	1105	20
Models	13	0	175	3
Requests	13	2	390	26
Viso:	**39**	**3**	**1670**	**49**

**Įrankis neapskaičiavo tikslaus Controllers klasių, eilučių, metodų skaičiaus, nes naujai sukurtame projekte, kuriant valdiklius su komandine eilute maršrutai web.php faile nėra sugeneruojami, tai neleidžia įrankiui aptikti valdiklių. Controllers klasės, eilutės ir metodų skaičius buvo apskaičiuoti rankiniu būdu.*

Su LaravelStats įrankio pagalba buvo apskaičiuota, kad naujai sugeneruotas Laravel projektas su Artisan komandine eilute turi viso 39 klases, vidutiniškai 3 metodus, 1670 kodo eilutes ir bendrai 49 logines kodo eilutes. Toliau, su įrankiu buvo apskaičiuojamas iš PIM modelio sugeneruotų klasių ir kodo eilučių kiekis, rezultatai pateikiami 26 lentelėje.

26 lentelė. LaravelStats įrankiu apskaičiuotos iš modelio sugeneruoto kodo klasės ir kodo eilutės

Tipas	**Klasių sk.**	**Metodai klasėje (vidurkis)**	**Kodo eilučių sk.**	**Loginių kodo eilučių sk.**
Controllers	13	5.62	655	27
Models	13	2	532	13
Requests	13	3	544	26
Viso:	**39**	**3.54**	**1731**	**66**

Su įrankiu apskaičiuota, kad iš modelio sugeneruotų klasių yra 39, metodų vidurkis klasėje yra 3.54, kodo eilučių bendrai 1731 ir loginių kodo eilučių kiekis 66. Toliau, su įrankiu buvo apskaičiuotas veikiančios sistemos funkcionalumo rinkinio klasių ir kodo eilučių kiekiai, rezultatai pateikiami 27 lentelėje.

27 lentelė. LaravelStats įrankiu apskaičiuotos veikiančios sistemos funkcionalumo rinkinio klasės ir kodo eilutės

Tipas	**Klasių sk.**	**Metodai klasėje (vidurkis)**	**Kodo eilučių sk.**	**Loginių kodo eilučių sk.**
Controllers	13	5.79	2195	806
Models	13	5	1117	165
Requests	13	2	746	26
Viso:	**39**	**4.26**	**4058**	**997**

Veikianti sistema turi 39 klases, 4.26 metodų klasėje vidurkį, 4058 kodo eilutes ir 997 logines kodo eilutes. 28 lentelėje pateikiamas apibendrinantis naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas, lyginant juos su veikiančia sistema.

28 lentelė. LaravelStats naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas su veikiančia sistema

Palyginimas	**Laravel komandinės eilutės projektas**	**Iš modelio sugeneruotas kodas**
Klasių sk.	39/39	39/39
Klasių proc.	100%	100%
Metodų klasėje vidurkis	3/4.26	3.54/4.26
Metodų klasėje vidurkio proc.	70.42%	83.09%
Kodo eilučių sk.	(1670/4058)	(1731/4058)
Kodo eilučių proc.	41.15%	42.66%
Loginių kodo eilučių sk.	49/997	66/997
Loginių kodo eilučių proc.	4.91%	6.62%

Po automatinio naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimo nustatyta, kad tiek Laravel komandinės eilutės projektas, tiek iš modelio sugeneruotas kodas sugeneruota tą patį klasių kiekį, kurį turi veikianti sistema. Naujo Laravel projekto kurto su komandine eilute kodo eilučių kiekis skiriasi tik 61 kodo eilute nuo iš modelio sugeneruoto kodo. Taip yra todėl, kad naujas Laravel projektas, kuriamas su komandine eilute, sugeneruoja nemažą kiekį tarpų ir komentarų erdvių kiekvienoje klasėje. Nepaisant to, iš modelio sugeneruoto kodo metodų klasėje vidurkis yra 12.67% didesnis, nei naujo Laravel projekto, lyginant juos su veikiančia sistema. Iš modelio sugeneruojamas kodas yra pranašesnis ir loginių kodo eilučių kiekyje, sugeneruodamas 1.71% daugiau, nei naujai kurtas projektas.

5.2.4. Automatinis naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas su PHPLoc įrankiu

Antrasis iš automatinių naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimų buvo atliktas su PHPLoc įrankiu. Su įrankiu buvo atlikti skaičiavimai naujam Laravel projektui, kuris buvo kurtas su Artisan komandine eilute, iš modelio sugeneruotam Laravel kodui ir pilnai realizuotai draudimo paslaugas teikiančios įmonės informacinei sistemai kurtai su Laravel karkasu. Iš modelio sugeneruoto kodo modelio klasių diagrama, maža dalis iš modelio sugeneruotų kodo fragmentų ir maža dalis veikiančios sistemos nepriklausomo funkcionalumo rinkinio kodo fragmentų pateikiami 7 priede esančiose lentelėse ir paveiksle. Kiekvienam iš trijų projektų tipų buvo skaičiuojamas kodo eilučių skaičius, užkomentuotų kodo eilučių skaičius, neužkomentuotų kodo eilučių skaičius, loginių kodo eilučių skaičius, klasių skaičius, metodų skaičius, Blades aplankų (aplankai skirti talpinti Blades šablono failus) skaičius ir Blades failų skaičius. Iš įrankio gauti rezultatai naujam Laravel projektui, pateikiami 29 lentelėje.

29 lentelė. PHPLoc įrankiu apskaičiuoti rezultatai naujam Laravel projektui

Tipas	**Kodo eilučių sk.**	**Užkomentuotų kodo eilučių sk.**	**Neužkomentuotų kodo eilučių sk.**	**Loginių kodo eilučių sk.**	**Klasių sk.**	**Metodų sk.**	**Blades Aplankų sk.**	**Blades Failų sk.**
Models	175	15	160	3	13	0	-	-
Controllers	1105	624	481	20	13	91	-	-
Requests	390	143	247	26	13	26	-	-
Blade	0	0	0	0	0	0	0	0
Routes web.php	16	10	6	1	0	0	-	-
Viso:	**1686**	**792**	**894**	**50**	**39**	**117**	**0**	**0**

Su PHPLoc įrankiu viso suskaičiuotos 1686 kodo eilutės, iš jų 792 užkomentuotos ir 894 neužkomentuotos, taip pat įrankis rado 50 loginių kodo eilučių, 39 klases, 117 metodų, 0 Blades aplankų ir failų. Toliau, skaičiavimai buvo atlikti iš PIM modelio sugeneruotam kodui, rezultatai pateikiami 30 lentelėje.

30 lentelė. PHPLoc įrankiu apskaičiuoti rezultatai iš modelio sugeneruotam programiniam kodui

Tipas	**Kodo eilučių sk.**	**Užkomentuotų kodo eilučių sk.**	**Neužkomentuotų kodo eilučių sk.**	**Loginių kodo eilučių sk.**	**Klasių sk.**	**Metodų sk.**	**Blades Aplankų sk.**	**Blades Failų sk.**
Models	532	156	376	13	13	26	-	-
Controllers	668	270	398	27	13	73	-	-
Requests	544	0	544	26	13	26	-	-
Blade	823	0	823	-	-	-	12	45
Routes web.php	132	10	122	89	0	0	-	-
Viso:	**2699**	**436**	**2263**	**168**	**39**	**138**	**12**	**45**

Įrankis suskaičiavo 2699 kodo eilutes, iš jų 436 užkomentuotos ir 2263 neužkomentuotos. Taip pat suskaičiuota 168 loginių kodo eilučių, 39 klasės, 138 metodai, 12 Blades aplankų ir 45 Blades failai. Toliau, skaičiavimai buvo atlikti veikiančios sistemos funkcionalumo rinkiniui, rezultatai pateikiami 31 lentelėje.

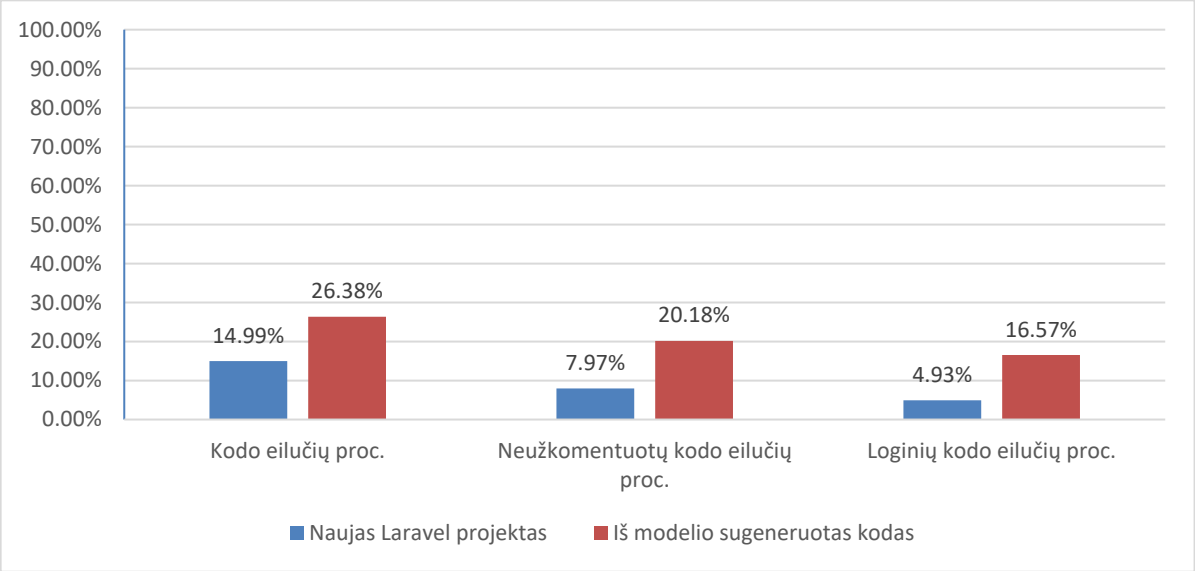
31 lentelė. PHPLoc įrankiu apskaičiuoti rezultatai veikiančios sistemos funkcionalumo rinkiniui

Tipas	**Kodo eilučių sk.**	**Užkomentuotų kodo eilučių sk.**	**Neužkomentuotų kodo eilučių sk.**	**Loginių kodo eilučių sk.**	**Klasių sk.**	**Metodų sk.**	**Blades Aplankų sk.**	**Blades Failų sk.**
Models	763	0	763	85	13	35	-	-
Controllers	2146	14	2132	795	13	77	-	-
Requests	746	0	746	26	13	26	-	-
Blade	7447	0	7447	-	-	-	12	53

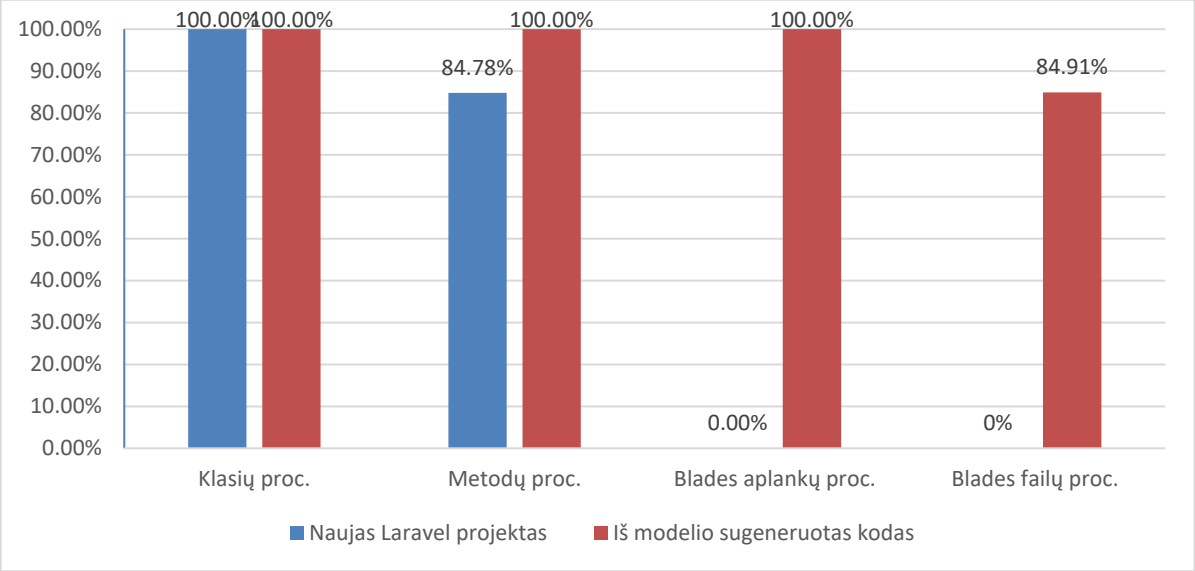
31 lentelė. PHPLoc įrankiu apskaičiuoti rezultatai veikiančios sistemos funkcionalumo rinkiniui

Tipas	**Kodo eilučių sk.**	**Užkomentuotų kodo eilučių sk.**	**Neužkomentuotų kodo eilučių sk.**	**Loginių kodo eilučių sk.**	**Klasių sk.**	**Metodų sk.**	**Blades Aplankų sk.**	**Blades Failų sk.**
Routes web.php	143	16	127	108	-	-	-	-
Viso:	**11245**	**30**	**11215**	**1014**	**39**	**138**	**12**	**53**

PHPLoc įrankis viso suskaičiavo 11245 kodo eilutes, iš kurių 30 kodo eilučių užkomentuota ir 11215 neužkomentuota. Taip pat apskaičiuota, kad veikianti sistema turi 1014 loginių kodo eilučių, 39 klases, 138 metodus, 12 Blades aplankų ir 53 Blades failus. 39 lentelėje ir 30 pav., 31 pav. pateikiamose diagramose pateikiamas apibendrinantis naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas, lyginant juos su veikiančia sistema. Diagramose pateikiami rezultatai pateikiami procentais nuo veikiančios sistemos kodo.



30 pav. Pirma duomenų dalis PHPLoc naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimui su veikiančia sistema



31 pav. Antra duomenų dalis PHPLoc naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimui su veikiančia sistema

Po automatinio naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimo su PHPLoc įrankiu nustatyta, kad, iš modelio sugeneruotas kodas sugeneruoja 11.39% kodo eilučių daugiau, nei naujai kurtas Laravel projektas. Tiek naujas projektas, tiek iš modelio sugeneruotas kodas turi didesnį kiekį užkomentuoto kodo eilučių, nei veikianti sistema. Taip yra todėl, nes naujai sugeneruotame projekte automatiškai priskiriama daug tuščių komentarų, o iš modelio sugeneruotame kode komentarais išskiriama erdvė nepergeneruojamam programiniam kodui. Toliau, iš modelio sugeneruotas kodas turi 12.21% daugiau neužkomentuoto kodo eilučių ir 11.64% daugiau loginių kodo eilučių, nei naujas Laravel projektas. Abu projektai sugeneruoja visas reikiamas veikiančios sistemos klases, tačiau iš modelio sugeneruotas kodas turi visus veikiančios sistemos metodus, o naujo Laravel projekto kodas turi 84.78% veikiančios sistemos metodų. Lyginant sugeneruotus Blades aplankus ir failus, naujame Laravel projekte juos sugeneruoti galimybės nėra, o iš modelio sugeneruotame kode sugeneruojami visi veikiančios sistemos Blades aplankai ir 84.91% veikiančios sistemos Blades failų.

5.3. Sprendimo veikimo ir savybių analizė, kokybės kriterijų įvertinimas

Sukurtai MDA architektūra grindžiamai metodikai, apimančiai modelių transformacijas ir PHP Laravel kodo generavimą, įvertinti buvo atliktas eksperimentas, kuris susidėjo iš keturių dalių ir kurio tikslas – nustatyti, ar modelių transformacijos ir kodo generavimas PHP Laravel karkasui palengvintų informacinių sistemų realizuojamu šiuo karkasu kūrimą. Pirmojoje eksperimento dalyje, atlikus tyrimo objekto naudotojų apklausą, nustatyta, kad respondentai dalyvavę apklausoje linkę sutikti, kad modelių transformacijos ir kodo generavimas galėtų sumažinti technologinius migracijos kaštus ir kūrimo laiko kaštus kuriant informacines sistemas. Taip pat, kad apklausoje pateikti sugeneruoto kodo pavyzdžiai yra korektiški ir tinkami naudoti, kuriant sistemas su Laravel karkasu. Likusiose trijose dalyje, atlikus rankinį ir du automatinius programinio kodo palyginimus su LaravelStats ir PHPLoc įrankiais, tarp naujo Laravel projekto sugeneruoto su Artisan komandomis, iš modelio sugeneruoto kodo ir veikiančios sistemos projekto, nustatyta, kad iš modelio sugeneruotas kodas yra pranašesnis, nei Laravel projektas kurtas su Artisan komandomis. Iš modelio sugeneruotas kodas, padengia daugiau veikiančios sistemos kodo funkcijų, nei naujas projektas ir tuo pačiu leidžia turėti sistemos UML modelį. Pagal eksperimento rezultatus, galima teigti, kad modelių transformacijos ir

kodo generavimas PHP Laravel karkasui palengvintų informacinių realizuojamu šiuo karkasu ir programavimo kalba kūrimą.

5.4. Sprendimo taikymo rekomendacijos

Realizuotą pasiūlytos metodikos sprendimą labiausiai tinka naudoti, jei norima kurti naują informacinę sistemą grįstą Laravel karkasu, tačiau galima naudoti ir norint sudaryti esamos sistemos modelį ir taip tęsti esamos sistemos funkcijų plėtojimą. Sprendimo nerekomenduojama naudoti, jei planuojama kurti sistema yra mažos apimties (1-2 modeliai, valdikliai ir rodiniai) ir toliau nenumatomas tokios sistemos plėtojimas. Taip pat sprendimas nebūtų tinkamas naudoti modeliuojant jau esamos sistemos modelį ir generuojant jai programinį kodą, jei sistema yra suprogramuota ne su 8, ar aukštesne Laravel versija, kadangi transformacijų ir kodo generavimo sprendimas pritaikytas generuoti Laravel 8+ PHP programinį kodą. Sprendimui yra taikomas vienas apribojimas – modeliai turi būti sudaromi su įrankiais, kurie palaiko XMI eksportavimo ir importavimo galimybes.

Modeliais grindžiamos PHP Laravel programinio kodo generavimo metodikos sprendimo taikymas apima įrankių įsidiegimą ir pati generavimo procesą. Sprendimo taikymo instrukcija susideda iš šešių pagrindinių dalių:

1. Į darbinę aplinką reikia įdiegti MagicDraw modeliavimo įrankį ir Eclipse Modeling Tools aplinką, į kurią papildomai reikia įsirašyti ATL ir Acceleo įrankius.
2. Sudaryti sistemos modelį su MagicDraw modeliavimo įrankiu. Sudarius modelį, jį išeksportuoti Eclipse 2 UML XMI formatu.
3. ATL įrankio konfigūracijoje nurodyti išeksportuoto modelio lokaciją, Laravel profilio lokaciją ir UML metamodelį iš Eclipse bibliotekos.
4. Paleisti transformaciją su ATL įrankiu iš PIM į PSM.
5. Atidaryti Acceleo transformacijų aplinką ir atlikti konfigūraciją. Konfigūracija atliekama nurodant UML metamodelį iš Eclipse bibliotekos, modelio, kurį norima transformuoti, lokaciją ir aplanką, į kurį Acceleo įrankis išsaugos sugeneruotą programinį kodą.
6. Paleisti transformaciją, po kurios įrankis sugeneruos programinį kodą. Šį programinį kodą galima įsikelti į savo Laravel projektą ir pabaigti programuoti reikiamas informacinės sistemos funkcijas.

Išvados

1. Pagrindinis MDA principas yra suteikti galimybę iš modelio sugeneruoti programinį kodą, o modelių transformacijas ir programinio kodo generavimą galima naudoti įvairioms programavimo kalboms ir jų karkasams. Išanalizavus UML sąsajas su PHP kalba ir šios kalbos karkasais nustatyta, kad galima pritaikyti modelių transformacijas ir kodo generavimą PHP kalbos Laravel karkasui.
2. MDA architektūros kontekste nerasta egzistuojančių MDA transformacijų ir kodo generavimo sprendimų PHP kalbos Laravel karkasui, todėl nuspręsta PHP kodo generavimui pritaikyti MDA transformacijų ir kodo generavimo principus: M2M transformacijoms, skirtoms transformuoti PIM modelį į Laravel specifinį modelį (PSM) naudoti ATL transformacijų įrankį, o Laravel programinio kodo generavimui (M2T transformacija) – Acceleo transformacijų įrankį.
3. Atlikus egzistuojančių modelių transformacijų MDA kontekste, jų įrankių ir egzistuojančių kodo generavimo iš PSM sprendimų analizę, nustatyta, kad egzistuoja pritaikytų sprendimų PHP kalbos CodeIgniter karkasui, Java kalbai ir jos karkasams, tačiau didžioji dalis šių sprendimų dažniau generuoja kodą iš UML struktūrinio tipo diagramų, rečiau – iš UML elgsenos diagramų.
4. Sprendimo specifikacijoje buvo apibrėžti iš PIM į PSM ir iš PSM į PHP Laravel programinį kodą transformacijoms aktualūs klasių ir sekų PIM ir PSM metamodelių fragmentai, formaliai apibrėžta metodika apimanti PIM ir PSM modelių struktūros specifikacijas, paruoštas PHP programavimo kalbos Laravel karkasui pritaikytas profilis, apibrėžti algoritmų aprašai ir pavyzdiniai modelių ir kodo fragmentai. Apibrėžtas metodikos taikymas transformacijoms leidžia panaudoti ne tik struktūrinio tipo klasių diagramą, tačiau ir elgsenos tipo sekų diagramą.
5. Realizavus ir ištestavus metodiką su paruoštais modeliais, iš kurių buvo sugeneruotas PHP Laravel programinis kodas, nustatyta, kad metodiką ir šios metodikos algoritmus galima sėkmingai pritaikyti transformacijoms iš PIM į PSM ir iš PSM į Laravel programinį kodą.
6. Eksperimento metu vykdytos apklausos metu, pagal gautus statistinės analizės rezultatus nustatyta, kad tyrimo objekto naudotojai linkę sutikti (3.55 iš 5, pagal Likerto skalę), kad modelių transformacijos ir kodo generavimas PHP Laravel karkasui palengvintų informacinių sistemų realizuojamu šiuo karkasu ir programavimo kalba kūrimą.
7. Atlikus palyginimo eksperimentus, lyginant naują Laravel projektą, kurtą su komandine eilute, iš modelio sugeneruotą kodą ir veikiančios sistemos nepriklausomą funkcionalumo rinkinį, nustatyta, kad iš modelio sugeneruotas kodas padengia daugiau veikiančios sistemos funkcionalumo (33.63% daugiau), nei programinis kodas, kurtas su komandine eilute.

Magistro baigiamojo projekto tematika buvo parašytas ir pristatytas mokslinis straipsnis tarptautinėje konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2020) [30].

Literatūros sąrašas

1. OMG, „Unified Modeling Language,“ 2017. [Tinkle]. Available: https://www.omg.org/spec/UML/. [Kreiptasi 24 10 2019].
2. K. Arrhioui, S. Mbarki, O. Betari, S. Roubi ir M. Erramdani, „A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications,“ *Transactions on Machine Learning and Artificial Intelligence,* t. 5, nr. 4, pp. 1-8, 2017.
3. S. D. Rathod, „Automatic code generation with business logic by capturing attributes from user interface via XML,“ įtraukta *International Conference on Electrical, Electronics, and Optimization Techniques*, 2016.
4. M. Laaziria, K. Benmoussa, S. Khoulji ir M. L. Kerkeb, „A Comparative study of PHP frameworks performance,“ *Procedia Manufacturing,* t. 32, pp. 864-871, 2019.
5. A. Srai, F. Guerouate, N. Berbiche ir H. D. Lahsini, „Applying MDA approach for Spring MVC Framework,“ *International Journal of Applied Engineering Research,* t. 12, nr. 14, pp. 4372-4381, 2017.
6. M. Ozkaya, „Are the UML modelling tools powerful enough,“ *IET Software,* t. 13, nr. 5, pp. 338-354, 2019.
7. E. Planas ir J. Cabot, „How are UML class diagrams built in practice? A usability study of two UML tools: Magicdraw and Papyrus,“ *Computer Standards & Interfaces,* t. 67, nr. 103363, 2020.
8. OMG, „ABOUT THE META OBJECT FACILITY SPECIFICATION VERSION 2.5.1,“ [Tinkle]. Available: https://www.omg.org/spec/MOF. [Kreiptasi 14 11 2019].
9. F. Deeba, S. Kun, M. Shaikh, F. A. Dharejo, S. Hayat ir P. Suwansrikham, „Data Transformation of UML Diagram by Using Model Driven Architecture,“ įtraukta *2018 the 3rd IEEE International Conference on Cloud Computing and Big Data Analysis*, 2018.
10. OMG, „ABOUT THE XML METADATA INTERCHANGE SPECIFICATION VERSION 2.5.1,“ [Tinkle]. Available: https://www.omg.org/spec/XMI. [Kreiptasi 14 11 2019].
11. O. 2. E. Team, „Meta-Modeling and the OMG Meta Object Facility (MOF),“ 01 03 2017. [Tinkle]. Available: https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf. [Kreiptasi 04 01 2020].
12. M. Brambilla, J. Cabot ir M. Wimmer, Model-Driven Software Engineering in Practice: Second Edition, 2017.
13. W. B. A. Karaa, Z. B. Azzouz, A. Singh, N. Dey, A. S. Ashour ir H. B. Ghazala, „Automatic builder of class diagram (ABCD): an application of UML generation from functional requirements,“ *Software: Practice and Experience,* t. 46, nr. 11, pp. 1443-1458, 2016.
14. D. J. M. Siegel, „MDA Guide rev. 2.0,“ Bostonas, 2014.
15. H. Benouda, M. Azizi, R. Esbai ir M. Moussaoui, „MDA Approach to Automate Code Generation for Mobile Applications,“ įtraukta *Mobile and Wireless Technologies 2016*, 2016.

16. K. Ma, B. Yang, Z. Chen ir A. Abraham, „A Relational Approach to Model Transformation with QVT Relations Supporting Model Synchronization," *Journal of Universal Computer Science,* t. 17, nr. 13, pp. 1863-1883, 2011.

17. N. Kahani, M. Bagherzadeh ir J. Cordy, „Survey and classification of model transformation tools.," *Software and Systems Modeling,* t. 18, nr. 37, p. 2361–2397, 2019.

18. „ATL | The Eclipse Foundation," [Tinkle]. Available: https://www.eclipse.org/atl/. [Kreiptasi 25 02 2020].

19. „Eclipse Acceleo," [Tinkle]. Available: https://projects.eclipse.org/projects/modeling.m2t.acceleo. [Kreiptasi 25 02 2020].

20. K. Benmoussa, M. Laaziri, S. Khoulji, M. L. Kerkeb ir A. E. Yamami, „A new model for the selection of web development frameworks: application to PHP frameworks," *International Journal of Electrical and Computer Engineering (IJECE) ,* t. 9, nr. 1, pp. 695-703, 1 vasaris 2019.

21. M. Laaziri, K. Benmoussa, S. Khoulji, K. M. Larbi ir A. E. Yamami, „A comparative study of laravel and symfony PHP frameworks," *International Journal of Electrical and Computer Engineering (IJECE),* t. 9, nr. 1, pp. 704-712, 2019.

22. „Laravel Documentation," [Tinkle]. Available: https://laravel.com/. [Kreiptasi 03 03 2020].

23. I. Essebaa, S. Chantit ir M. Ramdani, „MoDAr-WA: Tool Support to Automate an MDA Approach for MVC Web Application," *Computers 2019,* t. 8, nr. 89, pp. 1-23, 2019.

24. G. Paolone, M. Marinelli, R. Paesani ir P. Di Felice, „Automatic Code Generation of MVC Web Applications," *Computers 2020,* t. 9, nr. 3, p. 56, 2020.

25. E. Foundation, „Acceleo/Getting Started," Eclipse, [Tinkle]. Available: https://wiki.eclipse.org/Acceleo/Getting_Started#Run_as_Java_Application. [Kreiptasi 21 04 2021].

26. „The 4,5, and 7 Point Likert Scale," [Tinkle]. Available: https://www.formpl.us/blog/point-likert-scale. [Kreiptasi 04 04 2021].

27. IBM, „IBM SPSS software," [Tinkle]. Available: https://www.ibm.com/analytics/spss-statistics-software. [Kreiptasi 21 04 2021].

28. „Laravel Stats," [Tinkle]. Available: https://github.com/stefanzweifel/laravel-stats. [Kreiptasi 15 02 2021].

29. „PHPLoc," [Tinkle]. Available: https://github.com/sebastianbergmann/phploc. [Kreiptasi 15 02 2021].

30. M. Ražinskas ir L. Čeponienė, „MDA Approach for Laravel Framework Code Generation," įtraukta *INFORMATION SOCIETY AND UNIVERSITY STUDIES*, Kaunas, 2020.

31. G. Paolone, M. Marinelli, R. Paesani ir a. P. D. Felice, „Automatic Code Generation of MVC," *Computers 2020,* t. 9, nr. 3, p. 56, 2020.

32. A. Srai, F. Guerouate, N. Berbiche ir H. D. Lahsini, „Applying MDA approach for Spring MVC Framework," *International Journal of Applied Engineering Research,* t. 12, nr. 14, pp. 4372-4381, 2017.

33. OMG, „Unified Modeling Language," 2017. [Tinkle]. Available: https://www.omg.org/spec/UML/. [Kreiptasi 24 10 2019].

34. M. Laaziria, K. Benmoussa, S. Khoulji ir M. L. Kerbeb, „A Comparative study of PHP frameworks performance," *Procedia Manufacturing,* t. 32, pp. 864-871, 2019.

35. M. Laaziri, K. Benmoussa ir M. L. K. Samira Khoulji, „A Comparative study of PHP frameworks performance," *Procedia Manufacturing,* t. 32, pp. 864-871, 2019.

36. M. Laaziri, K. Benmoussa, S. Khoulji, K. M. Larbi ir A. E. Yamami, „A comparative study of laravel and symfony PHP frameworks," *International Journal of Electrical and Computer Engineering (IJECE),* t. 9, nr. 1, pp. 704-712, 2019.

37. A. Kilicdagi ir H. I. Yilmaz, Laravel Design and Best Practices, Packt Publishing, 2014.

38. I. Essebaa ir C. Salima, „QVT transformation rules to get PIM model from CIM," įtraukta *Europe and MENA Cooperation Advances in Information and Communication Technologies*, Springer International Publishing, 2017, pp. 195-207.

39. I. Essebaa, S. Chantit ir M. Ramdani, „MoDAr-WA: Tool Support to Automate an MDA Approach for MVC Web Application," *Computers 2019,* t. 8, nr. 89, pp. 1-23, 2019.

40. M. Brambilla, J. Cabot ir M. Wimmer, Model-Driven Software Engineering in Practice: Second Edition, 2017.

41. K. Benmoussa, M. Laaziri, S. Khoulji, M. L. Kerkeb ir A. E. Yamami, „A new model for the selection of web development frameworks: application to PHP frameworks," *International Journal of Electrical and Computer Engineering (IJECE),* t. 9, nr. 1, pp. 695-703, 1 vasaris 2019.

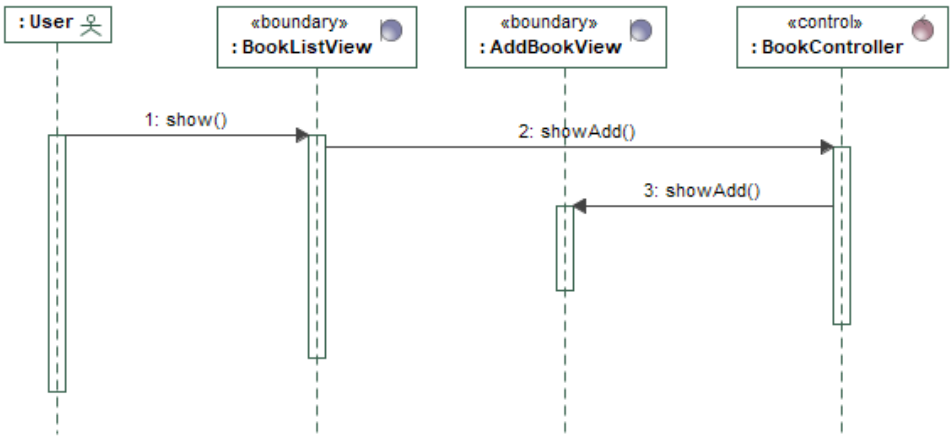
42. K. Arrhioui, S. Mbarki, O. Betari, S. Roubi ir M. Erramdani, „A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications," *Transactions on Machine Learning and Artificial Intelligence,* t. 5, nr. 4, pp. 1-8, 2017.

43. Mellor S.J., Scott K., Uhl A., Weise D., „Model-Driven Architecture.," įtraukta *Advances in Object-Oriented Information Systems. OOIS 2002. Lecture Notes in Computer Science, vol 2426. Springer*, Berlin, Heidelberg, 2002.

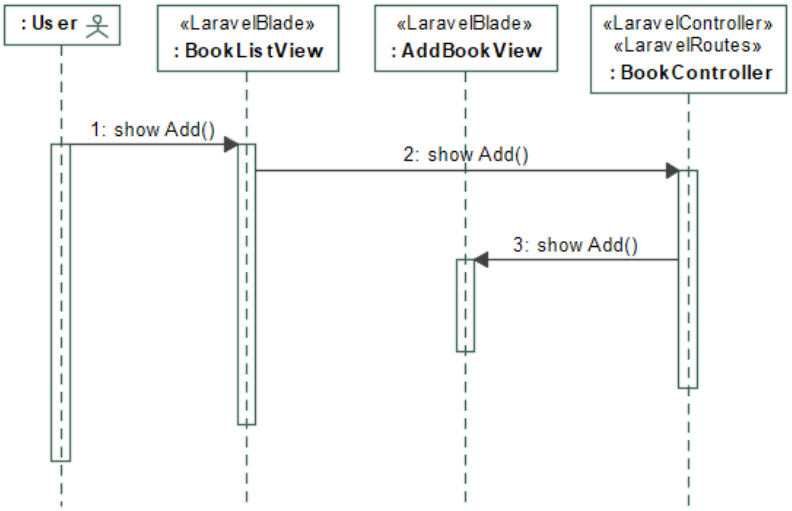
44. Sebastián, G., Gallud, J. A., Tesoriero, R., „Code generation using model driven architecture: A systematic mapping study," *Journal of Computer Languages,* t. Volume 56, nr. 100935, 2020.

Priedai

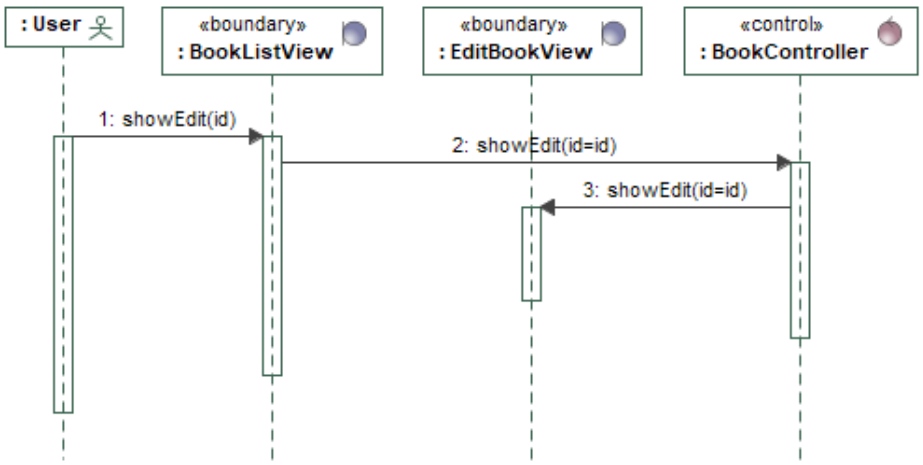
1 priedas. Pavyzdinio PSM fragmentai sudaryti iš sekų diagramų



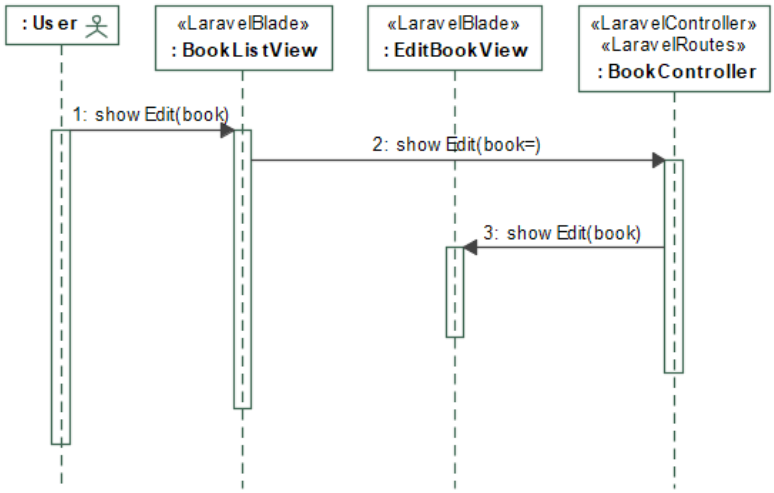
32 pav. PIM – knygų parduotuvės naujos knygos pridėjimo lango atidarymo sekų diagrama



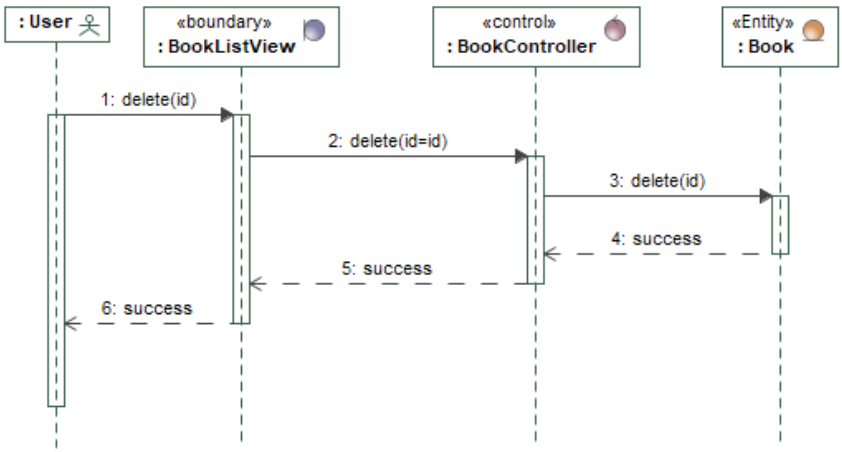
33 pav. PSM – knygų parduotuvės naujos knygos pridėjimo peržiūros lango atidarymo sekų diagrama



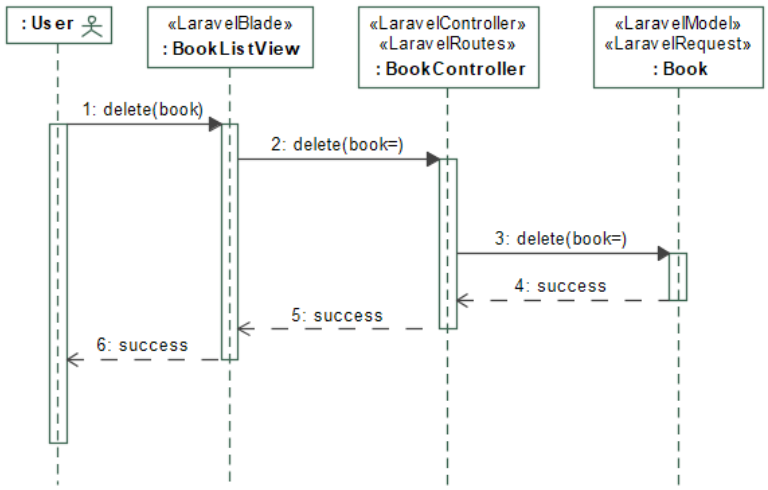
34 pav. PIM – knygų parduotuvės knygos redagavimo lango atidarymo sekų diagrama



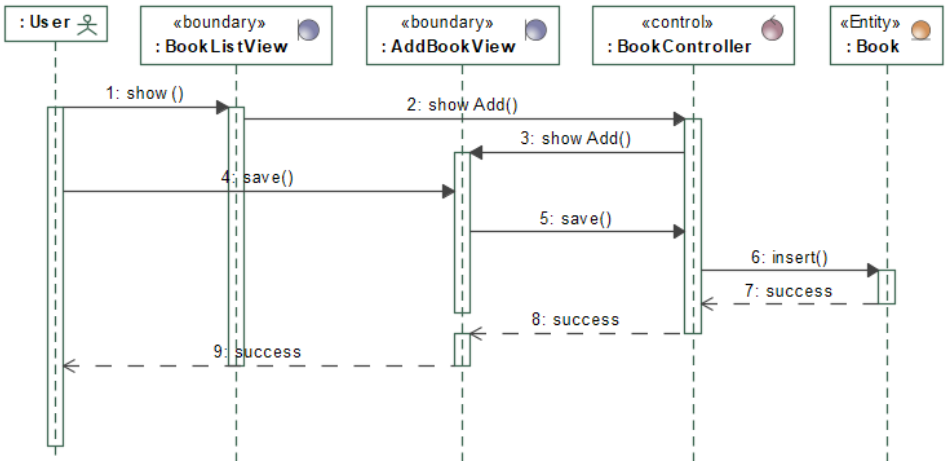
35 pav. PSM – knygų parduotuvės knygos redagavimo lango atidarymo sekų diagrama



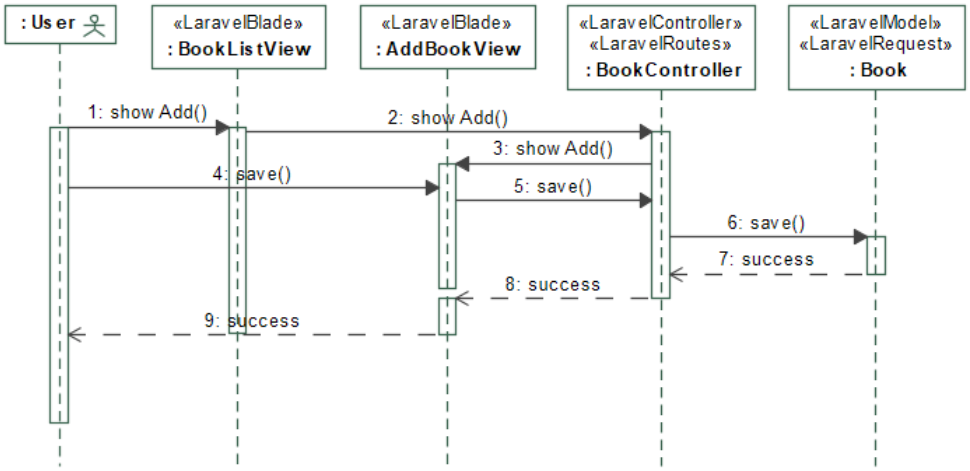
36 pav. PIM – knygų parduotuvės knygos pašalinimo sekų diagrama



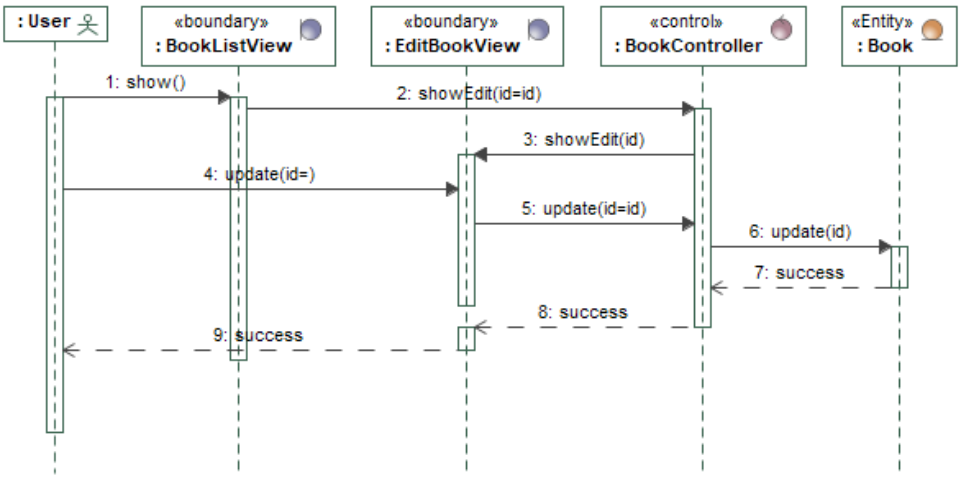
37 pav. PSM – knygų parduotuvės knygos pašalinimo sekų diagrama



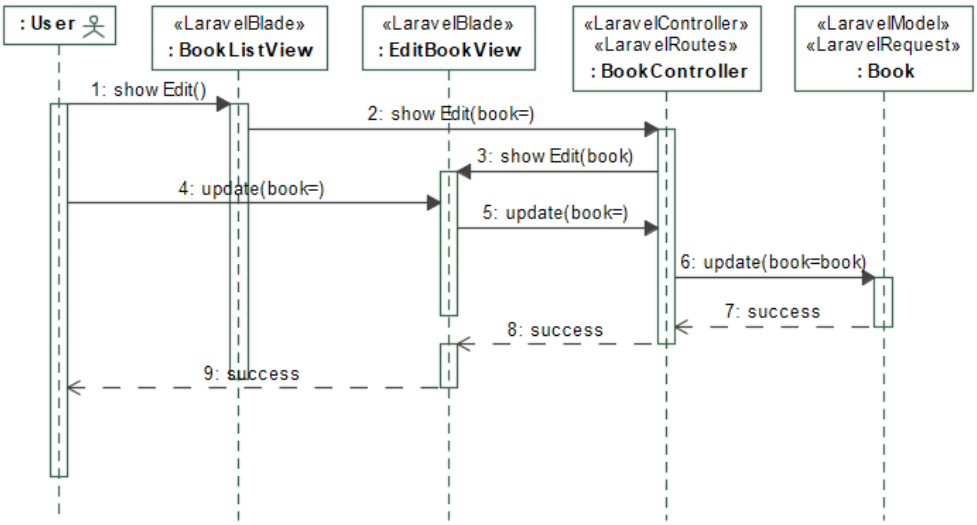
38 pav. PIM – knygų parduotuvės naujos knygos pridėjimo sekų diagrama



39 pav. PSM – knygų parduotuvės naujos knygos pridėjimo sekų diagrama



40 pav. PIM – knygų parduotuvės esamos knygos redagavimo sekų diagrama



41 pav. PSM – knygų parduotuvės esamos knygos redagavimo sekų diagrama

2 priedas. PIM transformacijos algoritmas

32 lentelė. ATL transformacijos iš PIM į PSM kodo fragmentas

```
-- @atlcompiler emftvm
-- @nsURI UML2=http://www.eclipse.org/uml2/5.0.0/UML

module Transformation;
create OUT : UML2 from IN : UML2, INPROFILE : UML2, LARAVELPROFILE : UML2;

helper context UML2!Element def: hasStereotype(stereotype : String) : Boolean =
   self.getAppliedStereotypes() -> collect(st | st.name) -> includes(stereotype);

helper context UML2!Element def: hasOperation(operation : String) : Boolean =
   self.getAllOperations() -> collect(op | op.name) -> includes(operation);

helper def : getStereotype(name : String) : UML2!Stereotype =
   UML2!Stereotype.allInstances()->select(p | p.name = name)->first();

helper def: fragmentOut : String = '';
helper def: lifelineOut : String = '';
helper def: ownedAttr : String = '';
helper def: sclass : UML2!"uml::Class" = '';
helper def: rsclass : UML2!"uml::Class" = '';
helper def: output : Sequence(UML2!Element) = Sequence{};
helper def: outputClass : Sequence(UML2!Element) = Sequence{};
helper def: outputMessage : Sequence(UML2!Element) = Sequence{};
helper def: counter : Integer = 1;

helper def: stereotypedValues: Sequence(UML2!Element) = Sequence{};
helper def: stereotypedClasses: Sequence(UML2!Element) = Sequence{};
helper def: savedMessages: Sequence(UML2!Element) = Sequence{};

-- ========================================================================
-- Stereotype copying rules begin
-- ========================================================================

lazy rule ApplyStereotypes {
   from s : UML2!"uml::Class", opas : UML2!"uml::Operation" in IN
   using {
      t : UML2!"uml::Class" = s.resolve();
      top : UML2!"uml::Operation" = opas.resolve();
   }
   do {
      for(singleClassArray in thisModule.stereotypedClasses){
         for(singleClassNameArray in thisModule.stereotypedValues){
            if(singleClassArray.size() = 3){
               if(singleClassNameArray.size() = 3){
                  if(singleClassArray.at(1).hasStereotype('LaravelBlade') and
singleClassArray.at(2).hasStereotype('LaravelController') and
singleClassArray.at(3).hasStereotype('LaravelBlade')){
                     for(outOp in thisModule.savedMessages){
                        if(outOp.size() = 3){
                           if(opas.owner.name = singleClassNameArray.at(2) and outOp->exists(p |
p.startsWith(opas.name))){
                              for(ops in opas.getApplicableStereotypes()){
                                 if(ops.name = 'RouteGET' and top.hasStereotype('RouteGET') = false and
top.hasStereotype('RouteDELETE') = false and top.hasStereotype('RoutePOST') = false and
top.hasStereotype('RoutePUT') = false and opas.owner.hasStereotype('control')){
                                    top.applyStereotype(ops);
                                 }
                              }
                           }
                        }
                     }
                  }
               }
            }
         }
      }
```

```
            }
            if(singleClassArray.size() >= 5){
                if(singleClassNameArray.size() >= 5){
                    if(singleClassArray.at(1).hasStereotype('LaravelBlade') and
singleClassArray.at(2).hasStereotype('LaravelController') and
singleClassArray.at(3).hasStereotype('LaravelModel')){
                        for(outOp in thisModule.savedMessages){
                            if(outOp.size() >= 5 and outOp.size() <= 7){
                                if(opas.owner.name = singleClassNameArray.at(2) and outOp->exists(p |
p.startsWith(opas.name)) and opas.ownedParameter.size() = 1){
                                    for(ops in opas.getApplicableStereotypes()){
                                        if(ops.name = 'RouteDELETE' and top.hasStereotype('RouteDELETE') =
false and top.hasStereotype('RouteGET') = false and top.hasStereotype('RoutePOST') = false and
top.hasStereotype('RoutePUT') = false and opas.owner.hasStereotype('control')){
                                            top.applyStereotype(ops);
                                        }
                                    }
                                }
                            }
                            if(outOp.size() >= 5){
                                if(opas.owner.name = singleClassNameArray.at(2) and outOp->exists(p |
p.startsWith(opas.name)) and opas.ownedParameter.size() = 0){
                                    for(ops in opas.getApplicableStereotypes()){
                                        if(ops.name = 'RoutePOST' and top.hasStereotype('RoutePOST') = false
and top.hasStereotype('RouteDELETE') = false and top.hasStereotype('RouteGET') = false and
top.hasStereotype('RoutePUT') = false and opas.owner.hasStereotype('control')){
                                            top.applyStereotype(ops);
                                        }
                                    }
                                }
                            }
                        }
                    }

                    if(singleClassArray.at(1).hasStereotype('LaravelBlade') and
singleClassArray.at(2).hasStereotype('LaravelController') and
singleClassArray.at(3).hasStereotype('LaravelBlade') and
singleClassArray.at(5).hasStereotype('LaravelController') and
singleClassArray.at(6).hasStereotype('LaravelModel')){
                        for(outOp in thisModule.savedMessages){
                            if(outOp.size() >= 7){
                                if(opas.owner.name = singleClassNameArray.at(2) and outOp->exists(p |
p.startsWith(opas.name)) and opas.ownedParameter.size() = 1){
                                    for(ops in opas.getApplicableStereotypes()){
                                        if(ops.name = 'RoutePUT' and top.hasStereotype('RoutePUT') = false and
top.hasStereotype('RouteGET') = false and top.hasStereotype('RoutePOST') = false and
top.hasStereotype('RouteDELETE') = false and opas.owner.hasStereotype('control')){
                                            top.applyStereotype(ops);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        if(s.hasStereotype('boundary')) {
            for(as in s.getApplicableStereotypes()){
                if(as.name = 'LaravelBlade' and t.hasStereotype('LaravelBlade') = false){
                    t.applyStereotype(as);
                    for (a in as.getAllAttributes()) {
                        if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {
                            t.setValue(as, a.name, s.getValue(as, a.name));
                        }
```

```
                    }
                }
            }
        }
        else if(s.hasStereotype('control')) {
            for(as in s.getApplicableStereotypes()){
                if(as.name = 'LaravelController' and t.hasStereotype('LaravelController') = false){
                    t.applyStereotype(as);
                    for (a in as.getAllAttributes()) {
                        if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {
                            t.setValue(as, a.name, s.getValue(as, a.name));
                        }
                    }
                }
                if(as.name = 'LaravelRoutes' and t.hasStereotype('LaravelRoutes') = false){
                    t.applyStereotype(as);
                    for (a in as.getAllAttributes()) {
                        if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {
                            t.setValue(as, a.name, s.getValue(as, a.name));
                        }
                    }
                }
            }
        }
        else if(s.oclIsTypeOf(UML2!"uml::Class")){
            for(as in s.getApplicableStereotypes()){
                if(as.name = 'LaravelModel' and t.hasStereotype('LaravelModel') = false){
                    t.applyStereotype(as);
                    for (a in as.getAllAttributes()) {
                        if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {
                            t.setValue(as, a.name, s.getValue(as, a.name));
                        }
                    }
                }
                if(as.name = 'LaravelRequest' and t.hasStereotype('LaravelRequest') = false){
                    t.applyStereotype(as);
                    for (a in as.getAllAttributes()) {
                        if (not a.name.startsWith('base_Class') and s.hasValue(as, a.name)) {
                            t.setValue(as, a.name, s.getValue(as, a.name));
                        }
                    }
                }
            }
        }
    }
}

endpoint rule ApplyAllStereotypes() {
    do {
        for (element in thisModule.traces.defaultSourceElements
                ->collect(e|e.object)
                ->select(o|o.oclIsKindOf(UML2!"uml::Interaction"))) {
            thisModule.FindSequenceRelation(element);
        }
        for (element in thisModule.traces.defaultSourceElements
                ->collect(e|e.object)
                ->select(o|o.oclIsKindOf(UML2!"uml::Class"))) {
            for (elementop in thisModule.traces.defaultSourceElements
                ->collect(e|e.object)
                ->select(o|o.oclIsKindOf(UML2!"uml::Operation"))) {
            thisModule.ApplyStereotypes(element, elementop);
                }
        }
    }
}

-- ==========================================================================
```

```
-- Stereotype copying rules end
-- ================================================================================

-- ================================================================================
-- Sequence finding rules begin
-- ================================================================================


lazy rule FindSequenceRelation {
   from s : UML2!"uml::Interaction" in IN
   using {
      t : UML2!"uml::Interaction" = s.resolve();
   }
   do {
            thisModule.counter <- 1;

            for(i in s.message){
            if(s.message->size() >= thisModule.counter){
            thisModule.outputMessage <- thisModule.outputMessage.append(s.message-
>at(thisModule.counter).name);
            thisModule.fragmentOut <- s.fragment -> select(frg | frg.__xmiID__ = s.message-
>at(thisModule.counter).receiveEvent.__xmiID__) -> collect(frg | frg.covered);
            for(fragment in thisModule.fragmentOut){
               for(lif in fragment){
                  thisModule.lifelineOut <- s.lifeline -> select(ll |ll.__xmiID__ = lif.__xmiID__) ->
collect(ll | ll.represents.__xmiID__);
                  for(lrepresents in thisModule.lifelineOut){
                     thisModule.ownedAttr <- s.ownedAttribute -> select(oa | oa.__xmiID__ = lrepresents)
->collect(oa | oa.type.__xmiID__);
                     for(owned in thisModule.ownedAttr){
                        thisModule.sclass <- UML2!"uml::Class".allInstances() -> select(cl |
cl.__xmiID__ = owned) -> collect(cl | cl);
                        thisModule.rsclass <- UML2!"uml::Class".allInstances() -> select(cl |
cl.__xmiID__ = owned) -> collect(cl | cl.name);
                        for(class in thisModule.sclass){
                           thisModule.outputClass <- thisModule.outputClass.append(class);
                        }
                        for(rclass in thisModule.rsclass){
                           thisModule.output <- thisModule.output.append(rclass);
                        }
                     }
                  }
               }
            }

            thisModule.counter <- thisModule.counter + 1;
            }else{
               thisModule.counter <- 1;
            }
         }
            thisModule.stereotypedValues<-thisModule.stereotypedValues.append(thisModule.output);
            thisModule.stereotypedClasses<-thisModule.stereotypedClasses.append(thisModule.outputClass);
            thisModule.savedMessages<-thisModule.savedMessages.append(thisModule.outputMessage);

            thisModule.output <- Sequence{};
            thisModule.outputClass <- Sequence{};
            thisModule.outputMessage <- Sequence{};
   }
}

-- ================================================================================
-- Sequence finding rules end
-- ================================================================================


rule EAnnotation {
    from s : UML2!"ecore::EAnnotation" in IN
```

```
    to t : UML2!"ecore::EAnnotation" (
        __xmiID__ <- s.__xmiID__,
        source <- s.source,
        eAnnotations <- s.eAnnotations,
        details <- s.details,
        contents <- s.contents,
        references <- s.references)
}

rule EStringToStringMapEntry {
    from s : UML2!"ecore::EStringToStringMapEntry" in IN
    to t : UML2!"ecore::EStringToStringMapEntry" (
        __xmiID__ <- s.__xmiID__,
        key <- s.key,
        value <- s.value)
}


rule Comment {
    from s : UML2!"uml::Comment" in IN
    to t : UML2!"uml::Comment" (
        __xmiID__ <- s.__xmiID__,
        body <- s.body,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        annotatedElement <- s.annotatedElement)
}

rule Package {
    from s : UML2!"uml::Package" in IN (s.oclIsTypeOf(UML2!"uml::Package"))
    to t : UML2!"uml::Package" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        packageMerge <- s.packageMerge,
        packagedElement <- s.packagedElement,
        profileApplication <- s.profileApplication)
}

rule Dependency {
    from s : UML2!"uml::Dependency" in IN (s.oclIsTypeOf(UML2!"uml::Dependency"))
    to t : UML2!"uml::Dependency" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        supplier <- s.supplier,
        client <- s.client)
}

rule ElementImport {
    from s : UML2!"uml::ElementImport" in IN
    to t : UML2!"uml::ElementImport" (
        __xmiID__ <- s.__xmiID__,
        visibility <- s.visibility,
```

```
        alias <- s.alias,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        importedElement <- s.importedElement)
}

rule PackageImport {
    from s : UML2!"uml::PackageImport" in IN
    to t : UML2!"uml::PackageImport" (
        __xmiID__ <- s.__xmiID__,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        importedPackage <- s.importedPackage)
}

rule Constraint {
    from s : UML2!"uml::Constraint" in IN (s.oclIsTypeOf(UML2!"uml::Constraint"))
    to t : UML2!"uml::Constraint" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        constrainedElement <- s.constrainedElement,
        specification <- s.specification)
}

rule Association {
    from s : UML2!"uml::Association" in IN (s.oclIsTypeOf(UML2!"uml::Association"))
    to t : UML2!"uml::Association" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        isDerived <- s.isDerived,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        generalization <- s.generalization,
        powertypeExtent <- s.powertypeExtent,
        redefinedClassifier <- s.redefinedClassifier,
        substitution <- s.substitution,
        representation <- s.representation,
        collaborationUse <- s.collaborationUse,
        ownedUseCase <- s.ownedUseCase,
        useCase <- s.useCase,
        memberEnd <- s.memberEnd,
        ownedEnd <- s.ownedEnd,
      navigableOwnedEnd <- s.navigableOwnedEnd)
}

rule Parameter {
    from s : UML2!"uml::Parameter" in IN
    to t : UML2!"uml::Parameter" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
```

```
        isOrdered <- s.isOrdered,
        isUnique <- s.isUnique,
        direction <- s.direction,
        isException <- s.isException,
        isStream <- s.isStream,
        effect <- s.effect,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        type <- s.type,
        templateParameter <- s.templateParameter,
        upperValue <- s.upperValue,
        lowerValue <- s.lowerValue,
        parameterSet <- s.parameterSet,
        defaultValue <- s.defaultValue)
}

rule Property {
    from s : UML2!"uml::Property" in IN (s.oclIsTypeOf(UML2!"uml::Property"))
    to t : UML2!"uml::Property" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isStatic <- s.isStatic,
        isOrdered <- s.isOrdered,
        isUnique <- s.isUnique,
        isReadOnly <- s.isReadOnly,
        isDerived <- s.isDerived,
        isDerivedUnion <- s.isDerivedUnion,
        aggregation <- s.aggregation,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        type <- s.type,
        upperValue <- s.upperValue,
        lowerValue <- s.lowerValue,
        templateParameter <- s.templateParameter,
        deployment <- s.deployment,
        redefinedProperty <- s.redefinedProperty,
        defaultValue <- s.defaultValue,
        subsettedProperty <- s.subsettedProperty,
        association <- s.association,
        qualifier <- s.qualifier)
}

rule Operation {
    from s : UML2!"uml::Operation" in IN
    to t : UML2!"uml::Operation" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isStatic <- s.isStatic,
        isAbstract <- s.isAbstract,
        concurrency <- s.concurrency,
        isQuery <- s.isQuery,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        ownedParameter <- s.ownedParameter,
        method <- s.method,
        raisedException <- s.raisedException,
        ownedParameterSet <- s.ownedParameterSet,
```

```
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        precondition <- s.precondition,
        postcondition <- s.postcondition,
        redefinedOperation <- s.redefinedOperation,
        bodyCondition <- s.bodyCondition)
}

rule Class {
    from s : UML2!"uml::Class" in IN (s.oclIsTypeOf(UML2!"uml::Class"))
    to t : UML2!"uml::Class" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        isActive <- s.isActive,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        generalization <- s.generalization,
        powertypeExtent <- s.powertypeExtent,
        redefinedClassifier <- s.redefinedClassifier,
        substitution <- s.substitution,
        representation <- s.representation,
        collaborationUse <- s.collaborationUse,
        ownedUseCase <- s.ownedUseCase,
        useCase <- s.useCase,
        ownedAttribute <- s.ownedAttribute,
        ownedConnector <- s.ownedConnector,
        ownedBehavior <- s.ownedBehavior,
        classifierBehavior <- s.classifierBehavior,
        interfaceRealization <- s.interfaceRealization,
        nestedClassifier <- s.nestedClassifier,
        ownedOperation <- s.ownedOperation,
        ownedReception <- s.ownedReception)
}

rule Model {
    from s : UML2!"uml::Model" in IN
    to t : UML2!"uml::Model" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        viewpoint <- s.viewpoint,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        packageMerge <- s.packageMerge,
      packagedElement <- s.packagedElement,
        profileApplication <- s.profileApplication)
}

rule DataType {
```

```
    from s : UML2!"uml::DataType" in IN (s.oclIsTypeOf(UML2!"uml::DataType"))
    to t : UML2!"uml::DataType" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        generalization <- s.generalization,
        powertypeExtent <- s.powertypeExtent,
        redefinedClassifier <- s.redefinedClassifier,
        substitution <- s.substitution,
        representation <- s.representation,
        collaborationUse <- s.collaborationUse,
        ownedUseCase <- s.ownedUseCase,
        useCase <- s.useCase,
        ownedAttribute <- s.ownedAttribute,
        ownedOperation <- s.ownedOperation)
}

rule ProfileApplication {
    from s : UML2!"uml::ProfileApplication" in IN
    to t : UML2!"uml::ProfileApplication" (
        __xmiID__ <- s.__xmiID__,
        isStrict <- s.isStrict,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        appliedProfile <- s.appliedProfile)
}

rule Actor {
    from s : UML2!"uml::Actor" in IN
    to t : UML2!"uml::Actor" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        generalization <- s.generalization,
        powertypeExtent <- s.powertypeExtent,
        redefinedClassifier <- s.redefinedClassifier,
        substitution <- s.substitution,
        representation <- s.representation,
        collaborationUse <- s.collaborationUse,
        ownedUseCase <- s.ownedUseCase,
        useCase <- s.useCase,
        ownedBehavior <- s.ownedBehavior,
        classifierBehavior <- s.classifierBehavior,
        interfaceRealization <- s.interfaceRealization)
}
```

```
rule Lifeline {
    from s : UML2!"uml::Lifeline" in IN
    to t : UML2!"uml::Lifeline" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        represents <- s.represents,
        selector <- s.selector,
        decomposedAs <- s.decomposedAs,
        coveredBy <- s.coveredBy)
}


rule CollaborationUse {
    from s : UML2!"uml::CollaborationUse" in IN
    to t : UML2!"uml::CollaborationUse" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        type <- s.type,
        roleBinding <- s.roleBinding)
}

rule Collaboration {
    from s : UML2!"uml::Collaboration" in IN
    to t : UML2!"uml::Collaboration" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        generalization <- s.generalization,
        powertypeExtent <- s.powertypeExtent,
        redefinedClassifier <- s.redefinedClassifier,
        substitution <- s.substitution,
        representation <- s.representation,
        collaborationUse <- s.collaborationUse,
        ownedUseCase <- s.ownedUseCase,
        useCase <- s.useCase,
        ownedBehavior <- s.ownedBehavior,
        classifierBehavior <- s.classifierBehavior,
        interfaceRealization <- s.interfaceRealization,
        ownedAttribute <- s.ownedAttribute,
        ownedConnector <- s.ownedConnector,
        collaborationRole <- s.collaborationRole)
}

rule Message {
    from s : UML2!"uml::Message" in IN
    to t : UML2!"uml::Message" (
        __xmiID__ <- s.__xmiID__,
```

```
        name <- s.name,
        visibility <- s.visibility,
        messageSort <- s.messageSort,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        receiveEvent <- s.receiveEvent,
        sendEvent <- s.sendEvent,
        connector <- s.connector,
        argument <- s.argument)
}

rule Interaction {
    from s : UML2!"uml::Interaction" in IN
    to t : UML2!"uml::Interaction" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        isActive <- s.isActive,
        isReentrant <- s.isReentrant,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        generalization <- s.generalization,
        powertypeExtent <- s.powertypeExtent,
        redefinedClassifier <- s.redefinedClassifier,
        substitution <- s.substitution,
        representation <- s.representation,
        collaborationUse <- s.collaborationUse,
        ownedUseCase <- s.ownedUseCase,
        useCase <- s.useCase,
        ownedAttribute <- s.ownedAttribute,
        ownedConnector <- s.ownedConnector,
        ownedBehavior <- s.ownedBehavior,
        classifierBehavior <- s.classifierBehavior,
        interfaceRealization <- s.interfaceRealization,
        nestedClassifier <- s.nestedClassifier,
        ownedOperation <- s.ownedOperation,
        ownedReception <- s.ownedReception,
        redefinedBehavior <- s.redefinedBehavior,
        ownedParameter <- s.ownedParameter,
        precondition <- s.precondition,
        postcondition <- s.postcondition,
        ownedParameterSet <- s.ownedParameterSet,
        specification <- s.specification,
        covered <- s.covered,
        generalOrdering <- s.generalOrdering,
        lifeline <- s.lifeline,
        fragment <- s.fragment,
        action <- s.action,
        formalGate <- s.formalGate,
        message <- s.message)
}

rule MessageOccurrenceSpecification {
    from s : UML2!"uml::MessageOccurrenceSpecification" in IN
    to t : UML2!"uml::MessageOccurrenceSpecification" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
```

```
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        covered <- s.covered,
        generalOrdering <- s.generalOrdering,
        toBefore <- s.toBefore,
        toAfter <- s.toAfter,
        message <- s.message)
}

rule BehaviorExecutionSpecification {
    from s : UML2!"uml::BehaviorExecutionSpecification" in IN
    to t : UML2!"uml::BehaviorExecutionSpecification" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        covered <- s.covered,
        generalOrdering <- s.generalOrdering,
        start <- s.start,
        finish <- s.finish,
        behavior <- s.behavior)
}

rule ExecutionOccurrenceSpecification {
    from s : UML2!"uml::ExecutionOccurrenceSpecification" in IN
    to t : UML2!"uml::ExecutionOccurrenceSpecification" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        covered <- s.covered,
        generalOrdering <- s.generalOrdering,
        toBefore <- s.toBefore,
        toAfter <- s.toAfter,
        execution <- s.execution)
}

rule CombinedFragment {
    from s : UML2!"uml::CombinedFragment" in IN (s.oclIsTypeOf(UML2!"uml::CombinedFragment"))
    to t : UML2!"uml::CombinedFragment" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        interactionOperator <- s.interactionOperator,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        covered <- s.covered,
        generalOrdering <- s.generalOrdering,
        operand <- s.operand,
        cfragmentGate <- s.cfragmentGate)
}

rule OpaqueExpression {
    from s : UML2!"uml::OpaqueExpression" in IN
    to t : UML2!"uml::OpaqueExpression" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        body <- s.body,
        language <- s.language,
```

```
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        type <- s.type,
        behavior <- s.behavior)
}

rule InteractionOperand {
    from s : UML2!"uml::InteractionOperand" in IN
    to t : UML2!"uml::InteractionOperand" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        covered <- s.covered,
        generalOrdering <- s.generalOrdering,
        guard <- s.guard,
        fragment <- s.fragment)
}

rule InteractionConstraint {
    from s : UML2!"uml::InteractionConstraint" in IN
    to t : UML2!"uml::InteractionConstraint" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        constrainedElement <- s.constrainedElement,
        specification <- s.specification,
        minint <- s.minint,
        maxint <- s.maxint)
}

rule LiteralInteger {
    from s : UML2!"uml::LiteralInteger" in IN
    to t : UML2!"uml::LiteralInteger" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        value <- s.value,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        --clientDependency <- s.clientDependency,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        type <- s.type)
}

rule LiteralString {
    from s : UML2!"uml::LiteralString" in IN
    to t : UML2!"uml::LiteralString" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        value <- s.value,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
```

```
        templateParameter <- s.templateParameter,
        type <- s.type)
}

rule LiteralBoolean {
    from s : UML2!"uml::LiteralBoolean" in IN
    to t : UML2!"uml::LiteralBoolean" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        value <- s.value,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        type <- s.type)
}

rule LiteralNull {
    from s : UML2!"uml::LiteralNull" in IN
    to t : UML2!"uml::LiteralNull" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        type <- s.type)
}

rule LiteralUnlimitedNatural {
    from s : UML2!"uml::LiteralUnlimitedNatural" in IN
    to t : UML2!"uml::LiteralUnlimitedNatural" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        value <- s.value,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        nameExpression <- s.nameExpression,
        templateParameter <- s.templateParameter,
        type <- s.type)
}
```

3 priedas. PSM transformacijos algoritmas

33 lentelė. Acceleo transformacijos iš PSM į programinį kodą kodo fragmentas

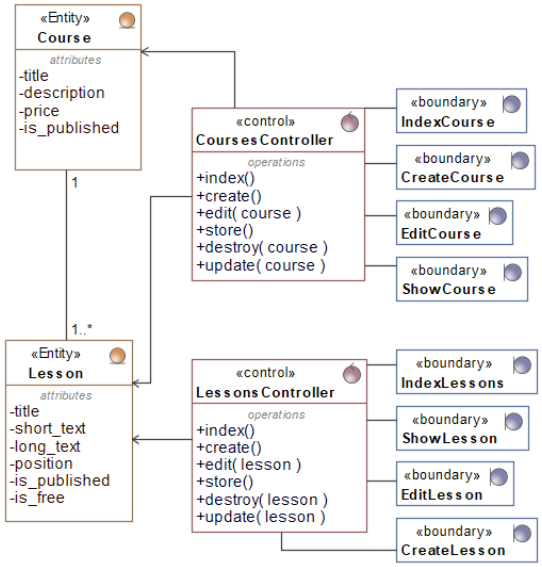
```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/uml2/5.0.0/UML',
'http://www.eclipse.org/uml2/5.0.0/UML/Profile/Standard')]


[template public generateElement(aModel : Model)]
[comment @main/]
[for (p:Package | aModel.eContents(Package))]
   [for (c:Class | p.eAllContents(Class))]

   [generateControllerClassDeclarations(c)/]
   [generateModelClassDeclarations(c)/]
   [generateBladeClassDeclarations(c)/]
   [generateRequestDeclarations(c)/]
   [generateRouteClassDeclarations(c)/]
   [/for]

[/for]
[/template]

[template public generateControllerClassDeclarations(c:Class) ? (c.hasStereotype('LaravelController'))]
[file ('app/Http/Controllers/' + c.name + '.php', false, 'UTF-8')]
<?php

namespace App\Http\Controllers;

[for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelModel'))]
            [if (rel.name <> c.name and c.name.startsWith(rel.name))]
use App\Models\[rel.name/];
            [/if]
         [/if]
         [if (rel.hasStereotype('LaravelRequest'))]
            [if (rel.name <> c.name and c.name.startsWith(rel.name))]
use App\Http\Requests\[rel.name/]Request;
            [/if]
         [/if]
      [/for]
[/for]
//[protected (c.name)]
//Nepergeneruojama sasaju su kitais modeliais apibrezimo dalis
//[/protected]
use Illuminate\Http\Request;

class [c.name/] extends Controller
{

   [for (op:Operation | c.getOperations()) ]
[if (op.hasStereotype('RouteGET'))]
[for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelModel'))]
            [if (rel.name <> c.name and c.name.startsWith(rel.name))]
[op.visibility/] function [op.name/]([if (op.getOwnedMembers()->notEmpty())][rel.name/] ['$'+
op.getOwnedMembers().name/][/if]) {
   //[protected (op.name)]
    // RouteGET metodo kodo nepergeneruojama dalis
   //[/protected]
   }

            [/if]
         [/if]
```

```
      [/for]
[/for]
         [/if]
   [if (op.hasStereotype('RoutePOST'))]
[for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelModel'))]
            [if (rel.name <> c.name and c.name.startsWith(rel.name))]
[op.visibility/] function [op.name/]([rel.name/]Request $request) {
    //[protected (op.name)]
    $[rel.name.toLower()/] = [rel.name/]::create($request->all());
   //[/protected]
   }
[/if]
         [/if]
      [/for]
[/for]
   [/if]
   [if (op.hasStereotype('RoutePUT'))]
[for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelModel'))]
            [if (rel.name <> c.name and c.name.startsWith(rel.name))]
[op.visibility/] function [op.name/]([rel.name/]Request $request, [rel.name/] $[rel.name.toLower()/]) {
    //[protected (op.name)]
    $[rel.name.toLower()/]->update($request->all());
   //[/protected]
   }

[/if]
         [/if]
      [/for]
[/for]
   [/if]
   [if (op.hasStereotype('RouteDELETE'))]
[for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelModel'))]
            [if (rel.name <> c.name and c.name.startsWith(rel.name))]
[op.visibility/] function [op.name/]([if (op.getOwnedMembers()->notEmpty())][rel.name/] ['$'+
op.getOwnedMembers().name/][/if]) {
    //[protected (op.name)]
    $[rel.name.toLower()/]->delete();
   //[/protected]
   }
[/if]
         [/if]
      [/for]
[/for]
   [/if]
   [/for]
}
//[protected (c.name)]
//Erdve kitam kodui
//[/protected]
[/file]
[/template]

[template public generateModelClassDeclarations(c:Class) ? (c.hasStereotype('LaravelModel'))]
[file ('app/Models/' + c.name + '.php', false, 'UTF-8')]
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
//[protected (c.name)]
```

```
//Erdve importams
//[/protected]
class [c.name/] extends Model
{
   use HasFactory;

   protected $fillable = ['['/]
    [for (p: Property | c.attribute) separator(',')]
   [if (p.name <> '')]
   '[p.name/]'
   [/if]
   [/for]
    [']'/];

   [for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelModel'))]
            [if (rel.name <> c.name)]
   public function [rel.name.toLower()/]() {
   //[protected (rel.name)]
   //Erdve belongsTo, hasMany.. rysiams
   return $this->([rel.name/]);
   //[/protected]
   }
            [/if]
         [/if]
      [/for]
[/for]
//[protected (c.name)]
//Erdve kitam kodui
//[/protected]
}
[/file]
[/template]

[template public generateBladeClassDeclarations(c:Class) ? (c.hasStereotype('LaravelBlade'))]
[for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelController'))]
            [for (ass1:Association | rel.getAssociations()) ]
               [for (rel1:Class | ass1.relatedElement) ]
                  [if (rel1.hasStereotype('LaravelModel'))]
                     [if (rel.name.startsWith(rel1.name))]
[file ('resources/views/'+ rel1.name.toLower()+ '/' + c.name.toLower() + '.blade.php', false, 'UTF-8')]
//[protected (c.name)]
<form method="GET/PUT/POST" action="{{ route('[rel1.name.toLower()/].method') }}">

<div class="form">
[for (p: Property | rel1.attribute)]
[if (p.name <> '')]<p>[p.name/]:{{ $[rel1.name.toLower()/]->[p.name/] }}</p>[/if]
[/for]
     </form>
 </div>
</div>
//[/protected]
[/file]
[/if]
                  [/if]
               [/for]
            [/for]
         [/if]
      [/for]
[/for]
[/template]

[template public generateRequestDeclarations(c:Class) ? (c.hasStereotype('LaravelRequest'))]
[file ('app/Http/Requests/' + c.name + 'Request.php', false, 'UTF-8')]
```

```
<?php
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class [c.name/]Request extends FormRequest
{
   public function authorize()
   {
       return true;
   }
   public function rules()
   {
       return ['['/]
    [for (p: Property | c.attribute) separator(',')]
   [if (p.name <> '')]
   '[p.name/]' => 'required'
   [/if]
   [/for]
    [']'/];
   }
   public function messages()
   {
       return ['['/]
    [for (p: Property | c.attribute) separator(',')]
   [if (p.name <> '')]
   '[p.name/].required' => 'This field is required'
   [/if]
   [/for]
    [']'/];
   }
}

[/file]
[/template]
[template public generateRouteClassDeclarations(c:Class) ? (c.hasStereotype('LaravelRoutes'))]
[file ('routes/web.php', true, 'UTF-8')]
[for (ass:Association | c.getAssociations()) ]
      [for (rel:Class | ass.relatedElement) ]
         [if (rel.hasStereotype('LaravelModel'))]
            [if (rel.name <> c.name and c.name.startsWith(rel.name))]

[for (op:Operation | c.getOperations()) ]
         [if (op.hasStereotype('RouteGET'))]
Route::get('[rel.name.toLower()/]s[if (op.getOwnedMembers()->notEmpty())]/['{'+ op.getOwnedMembers().name +
'}'/][/if]','[c.name/]@[op.name/]');
         [/if]
   [if (op.hasStereotype('RoutePOST'))]
Route::post('[rel.name.toLower()/]s','[c.name/]@[op.name/]');
   [/if]
   [if (op.hasStereotype('RoutePUT'))]
Route::put('[rel.name.toLower()/]s[if (op.getOwnedMembers()->notEmpty())]/['{'+ op.getOwnedMembers().name +
'}'/][/if]','[c.name/]@[op.name/]');
   [/if]
   [if (op.hasStereotype('RouteDELETE'))]
Route::get('[rel.name.toLower()/]s[if (op.getOwnedMembers()->notEmpty())]/['{'+ op.getOwnedMembers().name +
'}'/][/if]','[c.name/]@[op.name/]');
   [/if]
   [/for]
[/if]
         [/if]
      [/for]
[/for]
[/file]
[/template]
[query public hasStereotype (e:uml::Element, value: String):Boolean = not e.getAppliedStereotypes()-
>select(e:uml::Stereotype|e.name=value) -> isEmpty() /]
```

4 priedas. Sprendimo testavimo sugeneruoto programinio kodo fragmentai



42 pav. Mažesnis modelio klasių diagramos fragmentas skirtas sprendimo testavimui

34 lentelė. Mažesnio modelio testavimo metu sugeneruoto kodo CoursesController.php fragmentas

```
<?php
namespace App\Http\Controllers;
use App\Models\Course;
use App\Http\Requests\CourseRequest;
//Start of user code CoursesController
//End of user code
use Illuminate\Http\Request;
class CoursesController extends Controller
{
public function store(CourseRequest $request) {
    //Start of user code store
    $course = Course::create($request->all());
   //End of user code
   }
public function create() {
   //Start of user code create
   //End of user code
   }
public function index() {
   //Start of user code index
   //End of user code
   }
public function update(CourseRequest $request, Course $course) {
    //Start of user code update
    $course->update($request->all());
   //End of user code
   }
public function edit(Course $course) {
   //Start of user code edit
   //End of user code
   }
public function destroy(Course $course) {
    //Start of user code destroy
    $course->delete();
   //End of user code
   }
}
//Start of user code CoursesController
//End of user code
```

35 lentelė. Mažesnio modelio testavimo metu sugeneruoto kodo CourseRequest.php fragmentas

```
<?php
namespace App\Http\Requests;
```

```
use Illuminate\Foundation\Http\FormRequest;
class CourseRequest extends FormRequest
{
   public function authorize()
   {
       return true;
   }
   public function rules()
   {
       return [
   'title' => 'required',
   'description' => 'required',
   'price' => 'required',
   'is_published' => 'required',
    ];
   }
   public function messages()
   {
    return [
   'title.required' => 'This field is required',
   'description.required' => 'This field is required',
   'price.required' => 'This field is required',
   'is_published.required' => 'This field is required',
    ];
   }
}
```

36 lentelė. Mažesnio modelio testavimo metu sugeneruoto kodo Course.php fragmentas

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
//Start of user code Course
//End of user code
class Course extends Model
{
   use HasFactory;
   protected $fillable = [
   'title',
   'description',
   'price',
   'is_published',
    ];
   public function lesson() {
   //Start of user code Lesson
   //End of user code
   }
//Start of user code Course
//End of user code
}
```

37 lentelė. Mažesnio modelio testavimo metu sugeneruoto kodo web.php maršrutų fragmentas

```
Route::post('courses','CoursesController@store');
Route::get('courses','CoursesController@create');
Route::get('courses','CoursesController@index');
Route::put('courses/{course}','CoursesController@update');
Route::get('courses/{course}','CoursesController@edit');
Route::get('courses/{course}','CoursesController@destroy');
```

38 lentelė. Mažesnio modelio testavimo metu sugeneruoto Blades kodo indexcourse.blade.php fragmentas

```
//Start of user code IndexCourse
<form method="GET/PUT/POST" action="{{ route('course.method') }}">
<div class="form">
<p>title:{{ $course->title }}</p>
<p>description:{{ $course->description }}</p>
<p>price:{{ $course->price }}</p>
<p>is_published:{{ $course->is_published }}</p>
     </form>
 </div>
</div>
//End of user code
```

5 priedas. Eksperimento rezultatų lentelė

39 lentelė. PHPLoc naujo Laravel projekto ir iš modelio sugeneruoto kodo palyginimas su veikiančia sistema

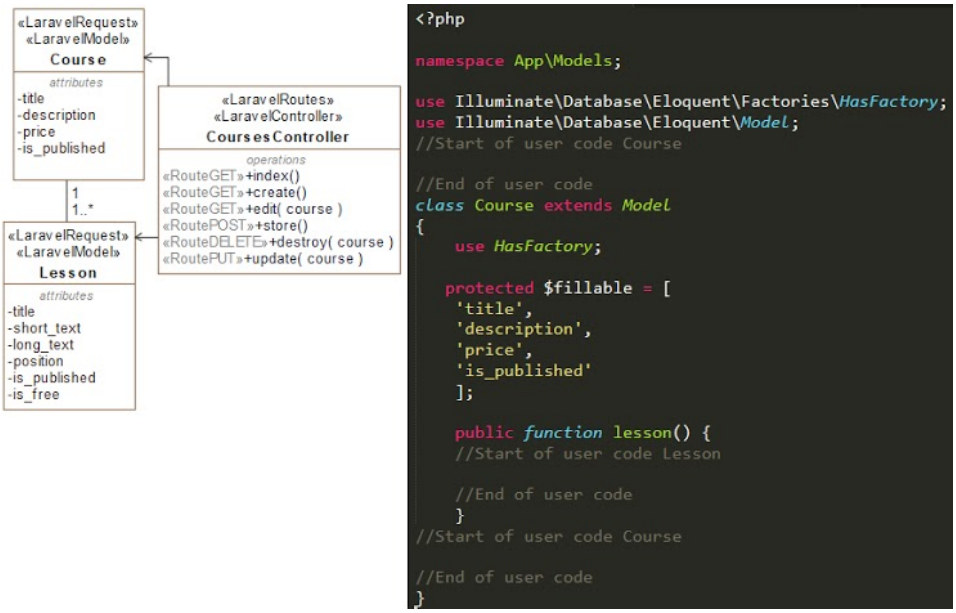
Palyginimas	**Laravel komandinės eilutės projektas**	**Iš modelio sugeneruotas kodas**
Kodo eilučių sk.	1686/11245	2966/11245
Kodo eilučių proc.	14.99%	26.38%
Užkomentuotų kodo eilučių sk.	792/30	436/30
Užkomentuotų kodo eilučių proc.	2640%	1453.33%
Neužkomentuotų kodo eilučių sk.	894/11215	2263/11215
Neužkomentuotų kodo eilučių proc.	7.97%	20.18%
Loginių kodo eilučių sk.	50/1014	168/1014
Loginių kodo eilučių proc.	4.93%	16.57%
Klasių sk.	39/39	39/39
Klasių proc.	100%	100%
Metodų sk.	117/138	138/138
Metodų proc.	84.78%	100%
Blades aplankų sk.	0/12	12/12
Blades aplankų proc.	0%	100%
Blades failų sk.	0/53	45/53
Blades failų proc.	0%	84.91%

6 priedas. Eksperimento naudotojų apklausos rezultatai

40 lentelė. Dalis klausimų užduotų respondentams apklausos metu

Klausimas	**Atsakymų apibendrinimas**
Kokia yra Jūsų specialybė?	50 programuotojų, 3 analitikai.
Kiek metų patirties turite su PHP programavimo kalba?	17 respondentų (32.1 %) atsakė turintys 5 metus ir daugiau patirties su PHP programavimo kalba, 16 respondentų (30.2 %) atsakė, kad turi nuo 1 iki 3 metų patirties, 14 respondentų (26.4 %) atsakė, kad turi nuo 3 iki 5 metų patirties, 4 respondentai (7.5 %) turi iki 1 metų patirties ir 2 respondentai (3.8 %) atsakė, kad neturi patirties su šia programavimo kalba.
Kiek metų patirties turite su Laravel karkasu?	16 respondentų (30.2 %) atsakė turintys nuo 1 iki 3 metų patirties su Laravel karkasu, 16 respondentų (30.2 %) atsakė, kad turi iki vienerių metų patirties, 9 respondentai (17 %) atsakė turintys nuo 3 iki 5 metų, 6 respondentai (11.3 %) patirties neturi, 6 respondentai (11.3 %) turi 5 ir daugiau metų patirties.
Ar žinote UML modeliavimo kalbą?	48 respondentai (90.6 %) atsakė žinantys UML modeliavimo kalbą, likę 5 respondentai (9.4 %) atsakė šios modeliavimo kalbos nežinantys.
Kaip vertintumėte savo UML modeliavimo įgūdžius?*	36 respondentai (75 %) savo modeliavimo įgūdžius vertina, kaip pradedančiojo, 10 respondentų (20.8 %) vertina, kad turi patirties ir 2 respondentai (4.2 %) vertina savo įgūdžius pažengusiu lygmeniu.
Ar naudojate UML modeliavimo įrankius (MagicDraw, Eclipse ar kt.) kurdami informacines sistemas?*	28 respondentai (58.3 %) atsakė, kad nenaudoja UML modeliavimo įrankių kurdami informacines sistemas, 20 respondentų (41.7 %) atsakė, kad naudoja.
Ar kurdami informacines sistemas naudojate įrankius, kurie sugeneruoja programinį kodą iš sumodeliuoto UML modelio?*	45 respondentai (93.8 %) kurdami informacines sistemas nenaudoja įrankių, kurie sugeneruoja programinį kodą iš sumodeliuoto UML modelio, 3 respondentai (6.3 %) naudoja įrankius sugeneruojančius programinį kodą iš UML modelio.
Ar manote, kad sistemos modeliavimas ir kodo generavimas iš modelio galėtų sumažinti technologinius migracijos kaštus ir kūrimo laiko kaštus kuriant informacines sistemas?	27 respondentai (50.9 %) mano, kad sistemų modeliavimas ir kodo generavimas iš modelio galėtų sumažinti technologinius migracijos kaštus ir kūrimo laiko kaštus kuriant informacines sistemas, kiti 26 respondentai (49.1 %) mano priešingai.

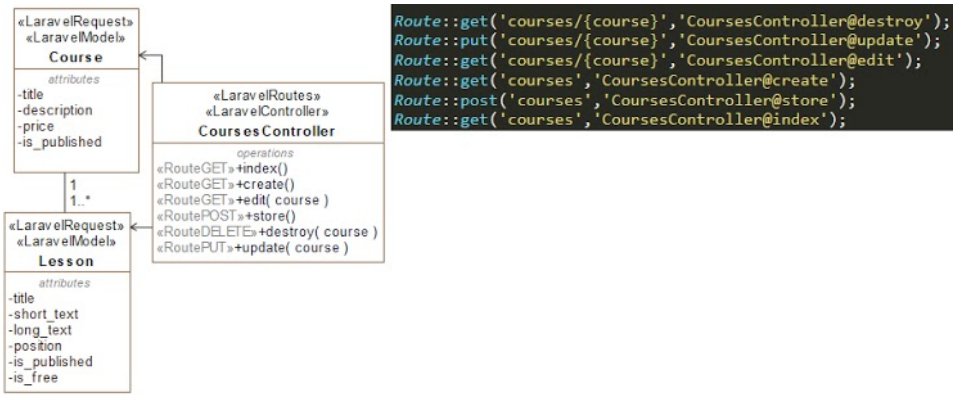
**Klausimai apie modeliavimo įgūdžius ir įrankių naudojimą buvo pateikiami tik asmenims žinantiems UML modeliavimo kalbą.*



43 pav. Klausime apie Laravel Model kodo fragmentą respondentams pateiktas klasių diagramos ir kodo fragmento pavyzdys

44 pav. Klausime apie Laravel Controller kodo fragmentą respondentams pateiktas klasių diagramos ir kodo fragmento pavyzdys

45 pav. Klausime apie Laravel Request kodo fragmentą respondentams pateiktas klasių diagramos ir kodo fragmento pavyzdys



46 pav. Klausime apie Laravel Routes kodo fragmentą respondentams pateiktas klasių diagramos ir kodo fragmento pavyzdys

41 lentelė. Teiginių apie Laravel Model kodo fragmentą rezultatų įvertinimas pagal Likerto skalę

Teiginys	**Visiškai nesutinku (1)**	**Nesutinku(2)**	**Nei sutinku, nei nesutinku (3)**	**Sutinku (4)**	**Visiškai sutinku (5)**
Pirmasis	6	8	15	17	7
Antrasis	1	4	14	21	13
Trečiasis	4	6	17	17	9

42 lentelė. Teiginių apie Laravel Controller kodo fragmentą rezultatų įvertinimas pagal Likerto skalę

Teiginys	**Visiškai nesutinku (1)**	**Nesutinku (2)**	**Nei sutinku, nei**	**Sutinku (4)**	**Visiškai sutinku (5)**

			nesutinku (3)		
Pirmasis	4	8	10	20	11
Antrasis	3	1	10	19	20
Trečiasis	4	5	11	19	14

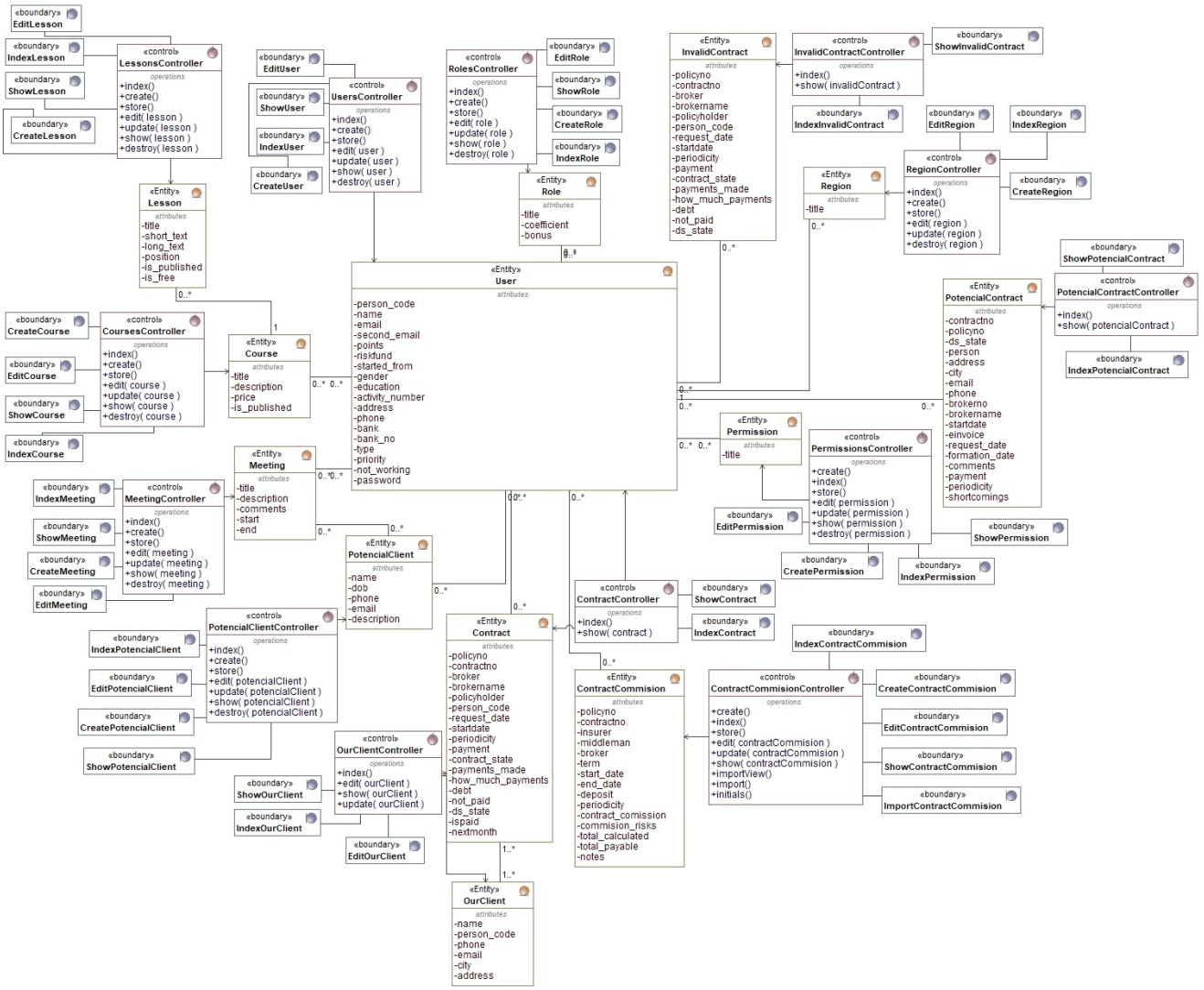
43 lentelė. Teiginių apie Laravel Request kodo fragmentą rezultatų įvertinimas pagal Likerto skalę

Teiginys	**Visiškai nesutinku (1)**	**Nesutinku (2)**	**Nei sutinku, nei nesutinku (3)**	**Sutinku (4)**	**Visiškai sutinku (5)**
Pirmasis	4	7	14	20	8
Antrasis	2	3	12	23	13
Trečiasis	4	3	15	22	9

44 lentelė. Teiginių apie Laravel Routes kodo fragmentą rezultatų įvertinimas pagal Likerto skalę.

Teiginys	**Visiškai nesutinku (1)**	**Nesutinku (2)**	**Nei sutinku, nei nesutinku (3)**	**Sutinku (4)**	**Visiškai sutinku (5)**
Pirmasis	6	7	17	14	9
Antrasis	4	3	13	19	14
Trečiasis	6	3	16	17	11

7 priedas. Eksperimento automatinio programinio kodo palyginimui naudota diagrama ir programinio kodo fragmentai



47 pav. Eksperimentui skirtos veikiančios sistemos funkcionalumo rinkinio klasių diagrama

45 lentelė. Eksperimento metu iš modelio sugeneruoto kodo Meeting.php Model klasės fragmentas

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
//Start of user code Meeting
//End of user code
class Meeting extends Model
{
    use HasFactory;

   protected $fillable = [
    'title',
    'description',
    'comments',
    'start',
    'end',
     ];

    public function potencialclient() {
    //Start of user code PotencialClient
    //End of user code
    }
```

```
    public function user() {
    //Start of user code User
    //End of user code
    }
//Start of user code Meeting
//End of user code
}
```

46 lentelė. Eksperimentui skirtos veikiančios sistemos funkcionalumo rinkinio Meeting.php klasės fragmentas

```
<?php

namespace App\Models;

use Carbon\Carbon;
use App\Traits\Auditable;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use \DateTimeInterface;

class Meeting extends Model
{
    use Auditable, HasFactory;

    public $table = 'meetings';

    protected $dates = [
        'start',
        'end',
        'created_at',
        'updated_at',
        'deleted_at',
    ];

    protected $fillable = [
        'title',
        'description',
        'comments',
        'start',
        'end',
        'created_at',
        'updated_at',
        'deleted_at',
    ];


    public function potencialclients()
    {
        return $this->belongsToMany(PotencialClient::class);
    }
    public function brokers()
    {
        return $this->belongsToMany(User::class);
    }
}
```

47 lentelė. Eksperimento metu iš modelio sugeneruoto kodo MeetingController.php Controller klasės fragmentas

```
<?php
namespace App\Http\Controllers;
use App\Models\Meeting;
use App\Http\Requests\MeetingRequest;
//Start of user code MeetingController
//End of user code
use Illuminate\Http\Request;

class MeetingController extends Controller
{

public function edit(Meeting $meeting) {
    //Start of user code edit
    //End of user code
    }
```

```
public function update(MeetingRequest $request, Meeting $meeting) {
    //Start of user code update
    $meeting->update($request->all());
   //End of user code
   }

public function show(Meeting $meeting) {
   //Start of user code show
   //End of user code
   }

public function create() {
   //Start of user code create
   //End of user code
   }

public function destroy(Meeting $meeting) {
    //Start of user code destroy
    $meeting->delete();
   //End of user code
   }
public function index() {
   //Start of user code index
   //End of user code
   }

public function store(MeetingRequest $request) {
    //Start of user code store
    $meeting = Meeting::create($request->all());
   //End of user code
   }
}
//Start of user code MeetingController
//End of user code
```

48 lentelė. Eksperimentui skirtos veikiančios sistemos funkcionalumo rinkinio MeetingController.php klasės fragmentas

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use App\Http\Requests\MeetingRequest;
use App\Models\Meeting;
use App\Models\PotencialClient;
use App\Models\User;
use Gate;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;
use Yajra\DataTables\Facades\DataTables;

class MeetingController extends Controller
{
    public function index(Request $request)
    {
        abort_if(Gate::denies('meeting_access'), Response::HTTP_FORBIDDEN, '403 Forbidden');

        if ($request->ajax()) {
            if(auth()->user()->roles->contains(1) || auth()->user()->roles->contains(2)){
                $query   =   Meeting::with(['potencialclients',   'brokers'])->select(sprintf('%s.*',   (new
Meeting)->table));

            }
            else if (auth()->user()->roles->contains(3)) {
             $query = Meeting::with(['potencialclients', 'brokers'])->whereHas('brokers', function($q) {
            $q->where('id', auth()->user()->id)->orWhere('users.region_id', auth()->user()->region_id);
            })->select(sprintf('%s.*', (new Meeting)->table));
         }
         else if (auth()->user()->roles->contains(4) || auth()->user()->roles->contains(5)|| auth()->user()-
>roles->contains(6)) {
             $query = Meeting::with(['potencialclients', 'brokers'])->whereHas('brokers', function($q) {
                $q->where('managing_id',       auth()->user()->id)->orWhere('id',       auth()->user()->id)-
>where('department_id', auth()->user()->department_id)->where('region_id', auth()->user()->region_id);
            })->select(sprintf('%s.*', (new Meeting)->table));
         }
```

```
         else{
             $query = Meeting::with(['potencialclients', 'brokers'])->whereHas('brokers', function($q) {
            $q->where('id', auth()->user()->id);
            })->select(sprintf('%s.*', (new Meeting)->table));
         }
            //$query = Meeting::with(['potencialclients', 'brokers'])->select(sprintf('%s.*', (new Meeting)-
>table));
            $table = Datatables::of($query);

            $table->addColumn('placeholder', ' ');
            $table->addColumn('actions', ' ');

            $table->editColumn('actions', function ($row) {
                $viewGate      = 'meeting_show';
                $editGate      = 'meeting_edit';
                $deleteGate    = 'meeting_delete';
                $crudRoutePart = 'meetings';

                return view('partials.datatablesActions', compact(
                    'viewGate',
                    'editGate',
                    'deleteGate',
                    'crudRoutePart',
                    'row'
                ));
            });

            $table->editColumn('id', function ($row) {
                return $row->id ? $row->id : "";
            });
            $table->editColumn('title', function ($row) {
                return $row->title ? $row->title : "";
            });
            $table->editColumn('description', function ($row) {
                return $row->description ? $row->description : "";
            });
            $table->editColumn('comments', function ($row) {
                return $row->comments ? $row->comments : "";
            });

            $table->editColumn('potencialclient', function ($row) {
                $labels = [];

                foreach ($row->potencialclients as $potencialclient) {
                    $labels[]    =    sprintf('<span    class="label    label-info    label-many">%s</span>',
$potencialclient->name);
                }

                return implode(' ', $labels);
            });
            $table->editColumn('broker', function ($row) {
                $labels = [];

                foreach ($row->brokers as $broker) {
                    $labels[]  =  sprintf('<span  class="label  label-info  label-many">%s</span>',  $broker-
>name);
                }

                return implode(' ', $labels);
            });

            $table->rawColumns(['actions', 'placeholder', 'potencialclient', 'broker']);

            return $table->make(true);
        }

        $potencial_clients = PotencialClient::get();
        if(auth()->user()->roles->contains(1) || auth()->user()->roles->contains(2)){
        $users = User::get();

    }
    else if (auth()->user()->roles->contains(3)) {
       $users  =  User::where('id',  auth()->user()->id)->orWhere('region_id',  auth()->user()->region_id)-
>get();
   }
   else  if  (auth()->user()->roles->contains(4)  ||  auth()->user()->roles->contains(5)||  auth()->user()-
>roles->contains(6)) {
```

```
        $users    =    User::where('managing_id',    auth()->user()->id)->orWhere('id',    auth()->user()->id)-
>where('department_id',    auth()->user()->department_id)->where('region_id',    auth()->user()->region_id)-
>get();
    }
    else{
        $users = User::where('id', auth()->user()->id)->get();
    }

        return view('admin.meetings.index', compact('potencial_clients', 'users'));
    }

    public function create()
    {
        abort_if(Gate::denies('meeting_create'), Response::HTTP_FORBIDDEN, '403 Forbidden');

        $potencialclients = PotencialClient::all()->pluck('name', 'id');

        $brokers = User::all()->pluck('name', 'id');

        return view('admin.meetings.create', compact('potencialclients', 'brokers'));
    }

    public function store(MeetingRequest $request)
    {
        $meeting = Meeting::create($request->all());
        $meeting->potencialclients()->sync($request->input('potencialclients', []));
        $meeting->brokers()->sync(auth()->user()->id);

        return redirect()->route('admin.meetings.index');
    }

    public function edit(Meeting $meeting)
    {
        abort_if(Gate::denies('meeting_edit'), Response::HTTP_FORBIDDEN, '403 Forbidden');

        $potencialclients = PotencialClient::all()->pluck('name', 'id');

        $brokers = User::all()->pluck('name', 'id');

        $meeting->load('potencialclients', 'brokers');

        return view('admin.meetings.edit', compact('potencialclients', 'brokers', 'meeting'));
    }

    public function update(MeetingRequest $request, Meeting $meeting)
    {
        $meeting->update($request->all());
        $meeting->potencialclients()->sync($request->input('potencialclients', []));
        $meeting->brokers()->sync(auth()->user()->id);

        return redirect()->route('admin.meetings.index');
    }

    public function show(Meeting $meeting)
    {
        abort_if(Gate::denies('meeting_show'), Response::HTTP_FORBIDDEN, '403 Forbidden');

        $meeting->load('potencialclients', 'brokers');

        return view('admin.meetings.show', compact('meeting'));
    }

    public function destroy(Meeting $meeting)
    {
        abort_if(Gate::denies('meeting_delete'), Response::HTTP_FORBIDDEN, '403 Forbidden');

        $meeting->delete();

        return back();
    }

    public function massDestroy(MassDestroyMeetingRequest $request)
    {
        Meeting::whereIn('id', request('ids'))->delete();

        return response(null, Response::HTTP_NO_CONTENT);
    }
```

```
}
```

49 lentelė. Eksperimento metu iš modelio sugeneruoto kodo MeetingRequest.php Request klasės fragmentas

```
<?php
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class MeetingRequest extends FormRequest
{
   public function authorize()
   {
       return true;
   }
   public function rules()
   {
       return [
    'title' => 'required',
    'description' => 'required',
    'comments' => 'required',
    'start' => 'required',
    'end' => 'required',
     ];
   }
   public function messages()
   {
       return [
    'title.required' => 'This field is required',
    'description.required' => 'This field is required',
    'comments.required' => 'This field is required',
    'start.required' => 'This field is required',
    'end.required' => 'This field is required',
     ];
   }
}
```

50 lentelė. Eksperimentui skirtos veikiančios sistemos funkcionalumo rinkinio MeetingRequest.php klasės fragmentas

```
<?php
namespace App\Http\Requests;
use Gate;
use Illuminate\Foundation\Http\FormRequest;

class MeetingRequest extends FormRequest
{
   public function authorize()
   {
       return Gate::allows('meeting_create');
   }
   public function rules()
   {
       return [
    'title' => 'required',
    'description' => 'required',
    'comments' => 'required',
    'start' => 'required',
    'end' => 'required',
    ];
   }
   public function messages()
   {
       return [
    'title.required' => 'Pavadinimo laukas yra privalomas',
    'description.required' => 'Aprašymo laukas yra privalomas',
    'comments.required' => 'Komentarų laukas yra privalomas',
    'start.required' => 'Susitikimo pradžios laikas yra privalomas',
    'end.required' => 'Susitikimo pabaigos laikas yra privalomas',
    ];
   }
}
```

51 lentelė. Eksperimento metu iš modelio sugeneruoto kodo maršrutų klasės fragmentas Meeting klasės maršrutams

```
Route::get('meetings/{meeting}','MeetingController@edit');
Route::put('meetings/{meeting}','MeetingController@update');
Route::get('meetings/{meeting}','MeetingController@show');
Route::get('meetings','MeetingController@create');
Route::get('meetings/{meeting}','MeetingController@destroy');
Route::get('meetings/{meeting}','MeetingController@massDestroy');
Route::get('meetings','MeetingController@index');
Route::post('meetings','MeetingController@store');
```

52 lentelė. Eksperimentui skirtos veikiančios sistemos funkcionalumo rinkinio maršrutų klasės fragmentas Meeting klasės maršrutams

```
Route::get('meetings/{meeting}','MeetingController@edit');
Route::put('meetings/{meeting}','MeetingController@update');
Route::get('meetings/{meeting}','MeetingController@show');
Route::get('meetings','MeetingController@create');
Route::get('meetings/{meeting}','MeetingController@destroy');
Route::get('meetings','MeetingController@index');
Route::post('meetings','MeetingController@store');
```