VILNIAUS UNIVERSITETAS

MARTAS AMBRAZIŪNAS

VEIKLOS ŽINIŲ BAZE IŠPLĖSTOS MODELIAIS GRINDŽIAMOS ARCHITEKTŪROS TAIKYMO INFORMACIJOS SISTEMŲ INŽINERIJOJE METODAS

Daktaro disertacija

Fiziniai mokslai, informatika (09P)

Vilnius, 2014

Disertacija parengta 2010 – 2014 metais Vilniaus universitete, Kauno humanitariniame fakultete, Informatikos katedroje.

Mokslinis vadovas:
Prof. dr. Audrius LOPATA (Vilniaus universitetas, fiziniai mokslai, Informatika – 09P).

Martas Ambraziūnas 2014

PADĖKOS

Disertacijos autorius dėkoja moksliniam vadovui prof. dr. Audriui Lopatai, už per doktorantūros studijas suteiktas vertingas mokslines konsultacijas bei įžvalgas, už patarimus ir pasiūlymus bei nuolatinį palaikymą, VU Kauno humanitarinio fakulteto informatikos katedros darbuotojams bei doktorantams už bendradarbiavimą, pagalbą ir supratimą.

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

SANTRUMPŲ SĄRAŠAS

CIM – *Computation Independant Model* – nuo skaičiavimų nepriklausomas modelis

DB – *Database* – duomenų bazė

EA – *Enterprise Architecture* – veiklos architektūra

EM - *Enterprise Model* – veiklos modelis

EML – *Enterprise Modeling Language* – veiklos modeliavimo kalba

EMM – *Enterprise Meta Model* – veiklos metamodelis

EMC – *Elementary Management Cycle* – elementarus valdymo ciklas

IS – *Information System* – informacijos sistema

ISO – *International Organization for Standardization* – tarptautinė standartizavimo organizacija

IT – *Information Technology* – informacinės technologijos

KB – *Knowledge Based* – žiniomis grindžiamas

MDA – *Model Driven Architecture* – modeliais grindžiama architektūra

OMG – Object Management Group – objektų standartizavimo organizacija

OS – *Operating system* – operacinė sistema

PIM – *Platform Independant Model* – nuo platformos nepriklausomas modelis

PĮ – Programinė įranga

PSM – *Platform Specific Model* – nuo platformos priklausomas modelis

SysML - *Systems Modeling Language* – sistemų modeliavimo kalba

UML – *Unified Modeling Language* – vieninga modeliavimo kalba

XMI – *XML Metadata Interchange* – Standartinė saityno metažymių kalba

XML – *Extensible Markup Language* – standartinė saityno metažymių kalba

MDD – *Model Driven Development* **–** modeliais grindžiamas kūrimas

GUI – *Grapthic User Interface* – grafinė vartotojo sąsaja

TERMINŲ ŽODYNAS

#	Terminas	Paaiškinimas
1	Nuo skaičiavimų nepriklausomas modelis (angl. *„CIM“*)	Modelis, kuris skirtas aprašyti vartotojo reikalavimus sistemai bei verslo procesus. Šis modelis neturi informacijos apie technologinę platformą, kurioje sistema bus realizuojama.
2	Nuo platformos nepriklausomas modelis (angl. *„PIM“*)	Modelis, kuris aprašo sistemos architektūrą ir elgseną be specifikacijų konkrečiai technologinei platformai. Modelį anotuojant konkrečios platformos informacija, yra sukuriamas nuo platformos priklausomas modelis.
3	Nuo platformos priklausomas modelis (angl. *„PSM“*)	Modelis, kuris aprašo sistemos architektūrą ir elgseną su specifikacijomis konkrečiai technologinei platformai (pvz. programavimo kalbai, operacinei sistemai).
4	Metamodelis	Metamodelis yra modeliavimo kalbos modelis, kuris nusako struktūrą, semantiką bei apribojimus, šia kalba kuriamiems modeliams [76].
5	Modeliavimo kalba	Dirbtinė kalba, kuri skirta išreikšti informaciją arba žinias, formatu, kurį apibrėžia pastovių taisyklių rinkinys.
6	Elementarus valdymo ciklas	Valdomą veiklos procesą formalizuotai aprašanti struktūra [23].
7	Modelis	Supaprastintas sudėtingos esybės ar proceso aprašas (pvz. formulių rinkiniu), atvaizdas ar pan.[53]
8	Žiniomis grindžiama sistema	Kompiuterinė programa, turinti deklaratyvių žinių bazę, kurioje laikomos užkoduotos žmogaus žinios problemų sprendimui [53].
9	Ontologija	Bendrai naudojamas formalus tam tikros srities sąvokų (konceptų), tipų, jų tarpusavio priklausomybės, ryšių, aksiomų, dėsningumų ir kt. aprašas [53].
10	Standartinė saityno metažymių kalba	Kalba, sudaranti galimybę kurti laisvai pasirenkamus žymių rinkinius, kuriais galima struktūrizuoti ir aprašyti bet kokios rūšies duomenis, numatant jų laikymo, peržiūrėjimo, apdorojimo būdus [53].
11	Žiniomis grindžiama IS inžinerija	Kompiuterizuota IS inžinerija, kai CASE sistemoje yra dalykinės srities žinių posistemis, sudarytas pagal veiklos modelį, kaupiantis žinias apie kompiuterizuojamą dalykinę sritį [25].
12	Modeliais grindžiama IS inžinerija	Programinės įrangos kūrimo inžinerijos kryptis, kuri akcentuoja dalykinės srities modelių sudarymą bei intensyvų jų naudojimą kūrimo procese.

TURINYS

ĮVADAS

Tyrimo aktualumas

Šiuolaikinius IS inžinerijos metodus (pvz. *RUP[35]*, *MDA[59]*, *SCRUM[75]*) nuolat vysto ir tobulina IT srities bendruomenės (pvz. *Agile Alliance[1]*, *OMG[65]*), komercinės (pvz. *IBM[34]*, *Microsoft[56]*) bei mokslo organizacijos, tačiau iš esmės šie metodai yra grindžiami empiriniais procesais t.y. dalykinės srities žinių surinkimo veikla priklauso nuo sistemų analitiko bei kliento (užsakovo) kompetencijos ir patirties. Praktinio darbo patirtis leidžia teigti, jog empiriškai išgautų žinių kokybė gali būti nepakankama sėkmingam projekto įgyvendinimui, nes netikslus ir/arba nepilnas vartotojo reikalavimų surinkimas neigiamai įtakoja visus programinės įrangos kūrimo etapus, o tai lemia didesnes laiko, žmogiškųjų, finansinių ir kitų išteklių sąnaudas bei didina projekto įgyvendinimo riziką.

Tyrimo objektas

Veiklos modelio integracija į MDA architektūra grindžiamą programinės įrangos kūrimo procesą.

Tyrimo tikslas ir uždaviniai

Darbo tikslas yra sukurti IS inžinerijos metodą, kuris įgalins empiriniais būdais surinktas žinias patikrinti formalių kriterijų (specifikuotų veiklos metamodelyje) atžvilgiu. Tai sąlygos mažesnę empirinių veiksnių (pvz.: nepakankamos analitiko bei kliento kompetencijos ir patirties) įtaką programinės įrangos kūrimo procesui. Šiam tikslui pasiekti žiniomis grindžiamos IS inžinerijos komponentai yra integruojami į MDA programinės įrangos kūrimo procesą. Disertacijos tikslui pasiekti buvo iškeltos šios užduotys:

1) Atlikti klasikinių programinės įrangos kūrimo metodų analizę, vartotojo reikalavimų surinkimo ir veiklos modeliavimo kontekste, siekiant įvertinti empirinių veiksnių įtaką šiems programinės įrangos kūrimo etapams.

2) Išanalizuoti žiniomis grindžiamų bei MDA metodų privalumus ir trūkumus, vartotojo reikalavimų specifikavimo ir veiklos modeliavimo etapuose.
3) Pasiūlyti žiniomis grindžiamą MDA metodą, iškeltam tikslui pasiekti, pateikiant realizacijos specifikaciją:
    1. specifikuoti metode taikomo veiklos modelio pakeitimus, būtinus integracijai į MDA procesą;
    2. sukurti veiklos modelio ir MDA modelių sąsajų žemėlapius (angl. *„mappings“*);
    3. detaliai specifikuoti žiniomis grindžiamo MDA metodo vykdymo procesą.
4) Siekiant patikrinti metodo funkcionalumą, sukurti dalykinės programos prototipą.
5) Eksperimentiškai patikrinti siūlomo metodo efektyvumą:
    1. taikant metodo funkcionalumą realizuojančios dalykinės programos prototipą bei žiniomis grindžiamo MDA metodo principus, sukurti mobiliems įrenginiams skirtą programėlę;
    2. aprašyti siūlomu metodu grindžiamą programėlės kūrimo procesą, pateikti eksperimentinio tyrimo išvadas.
6) Apibendrinti teoriškai ir eksperimentiškai gautus rezultatus. Pateikti rekomendacijas dėl siūlomo metodo panaudojimo galimybių ir tolimesnių metodo vystymo krypčių.

Tyrimo metodika

Darbe taikomi teoriniai ir eksperimentiniai tyrimo metodai. Dalykinės srities turinio analizės tyrimas apima sisteminę mokslinės literatūros analizę, nagrinėjančią programinės įrangos kūrimo metodų veiklos modeliavimo bei vartotojo reikalavimų procesus. Taikomi lyginamosios analizės bei apibendrinimo metodai. Teoriniams ir eksperimentiniams tyrimams atlikti buvo naudojamos CASE priemonės (*MagicDraw 17.0, MS Visio 2013*) ir kita programinė įranga (*MS SQLServer*, integruotos programavimo aplinkos *MS Visual Studio* bei *Eclipse).*

Ginamieji teiginiai

1) MDA architektūros išplėtimas veiklos modeliu padės užtikrinti dokumentacijos ir programinio kodo kokybę šiais aspektais: tiksliau ir pilniau bus specifikuojama dalykinė sritis, veiklos modelio pagrindu sugeneruota dokumentacija teigiamai (standartizuotas programinio kodo generavimas ir susiejimas su vartotojo reikalavimais) įtakos analitiko, projektuotojo ir programuotojo darbo rezultatus.
2) MDA architektūros išplėtimas veiklos modeliu sumažins laiko sąnaudas, kuriant identiško funkcionalumo dalykines programas skirtingoms operacinių sistemų platformoms.

Gauti rezultatai

1) Atlikus klasikinių programinės įrangos kūrimo procesų veiklos modeliavimo ir vartotojo reikalavimų specifikavimo analizę bei išanalizavus žiniomis grindžiamų ir MDA metodų privalumus ir trūkumus, nustatyta, kad tikslinga taikyti žiniomis grindžiamą MDA metodą iškeltam tikslui pasiekti.
2) Teoriškai aprašytas žiniomis grindžiamas MDA metodas.
3) Siekiant patikrinti metodo funkcionalumą, sukurtas žiniomis grindžiamo MDA metodo dalykinės programos prototipas ir mobiliems įrenginiams skirta programėlė.
4) Eksperimentiškai patikrintas siūlomo metodo efektyvumas, įvertinant laiko sąnaudas būtinas sukurti daugiaplatformėms sistemoms skirtas dalykines programas.
5) Apibendrinti teoriškai ir eksperimentiškai gauti rezultatai ir pateiktos metodinės rekomendacijos dėl siūlomo metodo taikymo IS inžinerijoje.

Mokslinis naujumas

Darbo metu naujais struktūriniais elementais papildytas VU mokslininkų (bendradarbiaujant su kitomis mokslo institucijomis) sukurtas veiklos metamodelis [24]. Šie pakeitimai atlikti, siekiant veiklos metamodelį integruoti į MDA procesą bei panaudoti dalykinės srities žinių tikrinimui formalių

kriterijų atžvilgiu [28] ir projektinių modelių generavimui. Sukurti transformavimo algoritmai, įgalinantys perkelti MDA CIM žinias į veiklos modelį ir veiklos modelyje saugomas dalykinės srities žinias į MDA PIM. Sukurta žiniomis grindžiamos IS inžinerijos ir MDA architektūros apjungimo koncepcija, suteikia galimybę vystyti progresyvius IS inžinerijos metodus, skirtus dalykinių programų ir programėlių, veikiančių daugiaplatformėse sistemose, kūrimui.

Praktinė reikšmė

Realizuoti vidinių CIM modelių (SysML *Use Case* ir *Activity* modelių*)* transformavimo į veiklos modelį algoritmai. Tai leidžia analitikui ir projektuotojui tikrinti šių modelių elementų tarpusavio sąsajas ir juose saugomų žinių atitikimą veiklos metamodelio taisyklėms. Tokiu būdu analitikas ir projektuotojas gali dirbti su standartiniais SysML ir UML modeliais bei lygiagrečiai naudotis veiklos modelio teikiamais privalumais (centralizuota žinių saugykla, žinių tikrinimas formalių kriterijų atžvilgiu, galimybė atvaizduoti žinias skirtingomis notacijomis).

Darbo metu sukurtas žiniomis grindžiamo MDA metodo dalykinės programos prototipas, kuris realizuoja dalį (*Use Case* ir *Activity* modelių transformavimas į veiklos modelį, tarpusavio suderinamumo tarp šių modelių tikrinimas, modeliuose apibrėžtų elementų, remiantis EVC kriterijais, tikrinimas) teoriškai apibrėžtų žiniomis grindžiamo MDA metodo funkcijų. Prototipas gali būti naudojamas kaip pagrindas kurti įskiepį konkrečiam modeliavimo įrankiui bei suteikti galimybę sistemų analitikams atlikti papildomą modelių tikrinimą veiklos metamodelio atžvilgiu.

Rezultatų aprobavimas

Disertacijos tematika buvo parengta ir publikuota 11 mokslinių straipsnių, iš jų – 5 (vienas spaudoje) straipsniai ISI Web of Science duomenų bazėje referuojamuose ir turinčiuose citavimo indeksą leidiniuose, 4 straipsniai (vienas spaudoje) – kituose ISI duomenų bazėse referuojamuose leidiniuose (proceedings ir kt.), 2 straipsniai – kituose recenzuojamuose mokslo

leidiniuose. Publikacijų sąrašas pateiktas 1 priede: „Autoriaus publikacijų sąrašas disertacijos tematika“.

Disertacijos tematika buvo skaityti 6 moksliniai pranešimai, iš jų – 3 pranešimai tarptautinėse mokslinėse konferencijose, 1 respublikinėje mokslinėje konferencijoje bei 2 tarptautiniuose mokslo seminaruose:

- Tarptautinis mokslo seminaras “Duomenų analizės metodai programų sistemoms 2013”, pranešimo pavadinimas: “Žiniomis grindžiamo MDA metodo įrankio prototipas”.
- Tarptautinis mokslo seminaras “Duomenų analizės metodai programų sistemoms 2012”, pranešimo pavadinimas: “Veiklos modelio taikymas žiniomis grindžiamame informacijos sistemų kūrimo procese”.
- Tarptautinė mokslo konferencija “International Conference on Business Information Systems 2011”, pranešimo pavadinimas: “Knowledge Based MDA Approach”.
- Tarptautinė mokslo konferencija “Information Technologies 2010”, pranešimo pavadinimas: “Knowledge Subsystem’s Integration into MDA Based Forward and Reverse IS Engineering”.
- Tarptautinė mokslo konferencija “International Conference on Business Information Systems 2010”, pranešimo pavadinimas: “MDA Compatable Knowledge Based IS Development Process”
- Tarpuniversitetinė magistrantų ir doktorantų mokslinėje konferencija „Informacinės technologijos 2010“, pranešimas „Veiklos žinių posisteme grindžiamas MDA metodas“.

Panaudojant disertacijos metu sukurtą metodą bei metodo dalykinės programos prototipą, buvo atliktas mokslinis tyrimas „Inovatyvių programinės įrangos kūrimo metodų taikymo išmaniesiems įrenginiams tyrimas„ (VP2-1.3-ŪM-05-K-02-057) pagal MITA „InočekiaiLT“ priemonę (VP2-1.3-ŪM-05). Tyrimo metu buvo detalizuotos siūlomo metodo panaudojimo galimybės, kuriant aplikacijas mobiliųjų įrenginių platformoms.

Disertacijos apimtis ir struktūra

Disertaciją sudaro santrumpų ir terminų žodynėliai, įvadas, keturi skyriai, išvados, panaudotos literatūros sąrašas, publikacijų sąrašas ir priedai. Bendra disertacijos apimtis yra 170 puslapiai, įskaitant 47 lentelių ir 50 paveikslų. Literatūros sąraše pateiktos 92 nuorodos.

Įvadinėje dalyje aptariama tiriamoji sritis, problemos aktualumas, darbo mokslinė reikšmė, aprašomas tyrimo objektas, formuojami darbo tikslas bei uždaviniai, pateikiamas darbo mokslinis naujumas, darbo praktinė reikšmė, ginamieji teiginiai, aprašoma tyrimo metodika. Įvado pabaigoje pristatomos disertacijos tema autoriaus paskelbtos publikacijos ir pranešimai konferencijose bei disertacijos struktūra.

Pirmasis skyrius skirtas programinės įrangos kūrimo metodikų apžvalgai vartotojo reikalavimų ir veiklos modeliavimo aspektu. Apžvelgiami tradicinių programinės įrangos kūrimo metodų reikalavimų surinkimo etapai, reikalavimų valdymo bei veiklos modeliavimo žiniomis grindžiamose IS inžinerijos metoduose bei modeliais grindžiamuose IS kūrimo metoduose problematika. Antrajame skyriuje aprašomas siūlomas programinės įrangos kūrimo metodas, jo sudėtis, veiklos procedūros, modelių sąsajų žemėlapiai, modelių transformavimo algoritmai. Trečioji dalis skirta aprašyti siūlomo metodo dalykinės programos prototipą, jo struktūrą ir funkcijas. Ketvirtojoje dalyje aprašomas, siūlomu metodu grindžiamas, mobiliesiems įrenginiams skirtos programėlės kūrimo procesas. Šioje dalyje taip pat pateikiamos eksperimentinio tyrimo išvados.

1. PROGRAMINĖS ĮRANGOS REIKALAVIMŲ SURINKIMO IR VEIKLOS MODELIAVIMO PROBLEMATIKA

Egzistuoja aibė programinės įrangos kūrimo metodų, koncepcijų ir idėjų, įskaitant klasikinį, bet jau atgyvenusį „krioklio“ modelį ir naujus, inovatyvius veiklos modeliais grindžiamus metodus. Vertinant programinės įrangos kūrimo metodus ir siekiant rasti tinkamiausią sprendimą atitinkantį konkretaus projekto įgyvendinimo ar konkrečios organizacijos poreikius, būtina atsižvelgti į tokius kriterijus kaip organizacijos dydis, darbuotojų patirtis ir kompetencija, turimi materialiniai ir žmogiškieji ištekliai, kuriamo produkto apimtis ir gyvavimo ciklas. Nepriklausomai nuo skirstymo, visos programinės įrangos kūrimo metodikos, apibrėžia šiuos esminius kūrimo etapus: reikalavimų surinkimas ir veiklos modeliavimas, programinės įrangos architektūros projektavimas, programinės įrangos programavimas, testavimas, diegimas ir palaikymas. Reikalavimų surinkimo ir veiklos modeliavimo etapai turi itin svarbią reikšmę [17], [44], [33] PĮ kūrimo procese, tačiau, dažnai šie etapai yra grindžiami empiriniais procesais [8], [41], kas įtakoja įvairių problemų atsiradimą (dėl nepilnai ir netiksliai surinktų reikalavimų, tampa sudėtinga tiksliai planuoti projekto biudžetą, įgyvendinimo terminus bei reikiamus išteklius ir kt.).

1.1 „Plan Driven“ metodai

„Plan Driven“ metodai, tokie kaip krioklio (angl. *„Waterfall“*) [71] ar RUP yra formalizuoti („sunkūs“), jų etapai griežtai atskirti vieni nuo kitų ir dokumentuoti. Kiekvienas etapas vykdomas tik pilnai pabaigus prieš tai buvusį etapą bei įsitikinus šio etapo rezultatų kokybe. Šie metodai yra naudojami didelės apimties ir sudėtingumo sistemų vystymui, kurias kuria ir prižiūri stambios kompanijos ar didelės specialistų grupės. Kritinis reikalavimas šių metodų naudojimui yra galimybė tiksliai specifikuoti sistemos reikalavimus bei surinktų reikalavimų pastovumas [91]. Minėtų metodų naudojimas užtikrina didesnę išteklių ir laiko kontrolę (egzistuoja aibė metrikų , kurios gali būti stebimos ir analizuojamos), taip pat įgalina paprasčiau integruoti naujus

specialistus į produkto kūrimo ir palaikymo procesą (egzistuoja išsami dokumentacija apie visus PĮ kūrimo etapus). Esminiai „Plan Driven“ metodų trūkumai yra papildomos laiko sąnaudos projekto administravimui, nepakankamas lankstumas keičiantis vartotojo reikalavimams, lėtas ir formalizuotas sprendimų priėmimo procesas. Reikalavimų surinkimo ir valdymo procesai „Plan Driven“ metoduose yra formalizuoti, iš esmės grindžiami panaudos atvejų (angl. „*Use Case*“) naudojimu, siekiant grafiškai bei tekstu aprašyti [70] reikalavimus. Reikalavimų surinkimo procesas yra skirstomas į tris pagrindinius etapus: informacijos apie dalykinės srities procesus specifikavimas, funkcinių reikalavimų specifikavimas, nefunkcinių reikalavimų specifikavimas. Reikalavimai surinkti panaudos atvejais yra aukšto abstrakcijos lygmens [20], todėl jie yra detalizuojami prieš perduodant specifikacijas programinės įrangos projektuotojams ir programuotojams. Detalizavimo proceso metu galimas loginių trūkių atsiradimas, kadangi šis procesas yra empiriškas t.y. sistemų analitikas bei projektuotojas skirtingai interpretuoja panaudos atvejais surinktą informaciją bei jos pagrindu kuria projektavimo lygmens artefaktus (struktūros, elgsenos modelius). Kitas svarbus „Plan Driven“ metodų trūkumas reikalavimų surinkimo ir valdymo prasme - pasikeitusių reikalavimų valdymas. Tai laiko ir darbo sąnaudoms imlus procesas, nes pasikeitus reikalavimams būtina atitinkamai pakeisti visus jais susijusius artefaktus (panaudos atvejus, architektūros ir veiklos modelius, programinį kodą), nepriklausomai nuo reikalavimo pakeitimo specifikos ir dydžio t.y. formalus procesas vykdomas nuosekliai, kas esant dinamiškai probleminei sričiai mažina galimybę operatyviai reaguoti.

1.2 Agilieji *(angl. „Agile“)* metodai

Agilieji [11] metodai apima aibę PĮ kūrimo metodų tokių kaip Scrum, XP[38], FDD[22] ir kitų [10], [92]. Šiems metodams būdingas lankstus požiūris į programinės įrangos kūrimą, pakartotinis grįžimas į prieš tai buvusius etapus, žmogiškųjų išteklių svarbos akcentavimas. Agilieji metodai įgalina operatyviai pateikti vartotojui dalinai veikiantį produktą, kuris gali būti

panaudotas kaip papildomas įrankis reikalavimų patikslinimui bei testavimui. Reikalavimų nepastovumas šio tipo programinės įrangos kūrimo metoduose yra valdomas efektyviau nei „Plan Driven“ metoduose [41]. Agilieji metodai yra naudojami projektams, kuriuos vykdo mažos, tačiau aukštos kvalifikacijos specialistų grupės. Pagrindiniai trūkumai – sudėtinga integruoti naujus specialistus į sistemos kūrimą (dokumentacija dažnai neegzistuoja arba yra fragmentiška), sudėtinga prognozuoti galutinę produkto kainą ir atlikimo terminus, stebėti pačią projekto eigą. Egzistuoja agilieji metodai (pvz. AgileUP [3]), kurie iš dalies įtraukia „Plan Driven“ metoduose naudojamas formalizavimo procedūras, neprarandant galimybės lanksčiai reaguoti į pasikeitusius reikalavimus. Reikalavimų surinkimo ir valdymo procesas agiliuose metoduose pradedamas nuo pirminio reikalavimų sąrašo sudarymo (angl. „*Requirements Backlog“).* Jo pagrindu yra sukuriamas produkto reikalavimų sąrašas (angl. *„Product Backlog“*) t.y. detalizuoti reikalavimai, kurie gali būti išreikšti kaip vartotojo istorijos, vizualizacijos ir kt. [60]. Produkto reikalavimams yra suteikiami prioritetai, remiantis jais reikalavimai yra atrenkami konkrečiai produkto kūrimo iteracijai. Reikalavimai dokumentuojami pasitelkiant lenteles, tekstinius aprašus, įvairių notacijų diagramas ar paveikslėlius. Toks reikalavimų specifikavimas įgalina prisitaikyti prie konkretaus vartotojo poreikių, tačiau sukuria prielaidas atsirasti loginiams trūkiams perduodant informaciją tarp skirtingų komandų bei itin stipriai priklauso nuo sistemų analitikų komandos kompetencijos t.y. empirinis faktorius agiliuose metoduose yra itin ryškus. Empiriškumo problema agiliuose metoduose yra sprendžiama naudojantis iteraciniu programinės įrangos kūrimu. Žemiau yra pateikta lentelė, kurioje yra pavaizduoti apibendrinti skirtingų metodologijų reikalavimų surinkimo etapų aprašymai.

Lentelė 1. Apibendrinti PĮ kūrimo metodų reikalavimų surinkimo etapų aprašymai

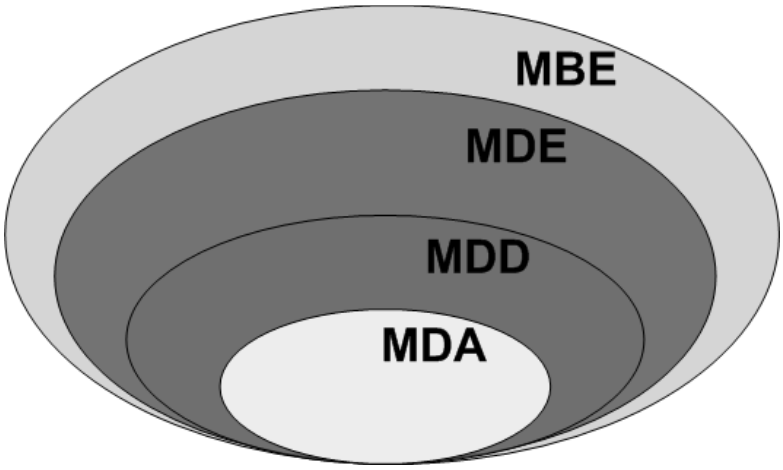
PĮ kūrimo metodas (Koncepcija)	Reikalavimų specifikavimo etapų pavadinimai	Apibendrintas reikalavimų surinkimo procesas ir dokumentacijos naudojimas	Reikalavimų tikrinimas
RUP (Plan Driven)	Business Modeling, Requirements acquisition	Naudojami reikalavimų aprašymo šablonai, grafiniai modeliai, egzistuoja reikalavimų surinkimo procedūrų aprašai ir detalūs procesai (pvz. Volere [72]). Reikalavimų surinkimo procesas yra nelankstus, tačiau suteikia galimybę dokumentaciją įtraukti į užsakymų sutartis.	Empirinis tikrinimas reikalavimų surinkimo etape, automatinis tikrinimas testavimo etape
SCRUM (Agilus)	Requirements BackLog, Product BackLog, Sprint Backlog	Formaliai dokumentuojami tik pagrindiniai vartotojo reikalavimus (ir iš jų sekantys sistemos reikalavimus), grafiniai modeliai naudojami retai, juos pakeičia „White Board“ naudojimas. Procesas yra lankstus ir adaptyvus, įgalina prisitaikyti prie kintančių reikalavimų bei suteikia greitą grįžtamąjį ryšį su vartotoju.	Empirinis tikrinimas reikalavimų surinkimo etape, empirinis tikrinimas programavimo etape, automatinis tikrinimas testavimo etape
MDA (MDD)	CIM	Dokumentacijoje dominuoja grafiniai modeliai, kas reikalauja žinoti grafines modeliavimo kalbas. Reikalavimai gali būti automatiškai transformuojami į kitų etapų artefaktus (t.y. mažesnis empiriškumas transformavimo metu), greitas reikalavimų pakeitimų perdavimas į kitus PĮ kūrimo etapus	Empirinis tikrinimas reikalavimų surinkimo etape, automatinis tikrinimas testavimo etape
Knowledge Based	Business Modeling, Requirements acquisition	Reikalavimai veiklos modelyje yra saugomi specifiniu formatu t.y. siekiant panaudoti kituose PĮ kūrimo etapuose turi būti transformuojami į standartizuotus formatus (UML, tekstą), reikalavimų tikrinimas yra tiek teisingas, kiek teisingas veiklos metamodelis. Surinkti reikalavimai automatiškai tikrinami formalių kriterijų atžvilgiu, reikalavimai saugomi vieningu formatu.	Empirinis ir formalus tikrinimas reikalavimų surinkimo etape, automatinis tikrinimas testavimo etape

Kaip matome iš pateiktos lentelės automatizuotai pagal formalias taisykles reikalavimai yra tikrinami tik žiniomis grindžiamuose IS inžinerijos metoduose. Naudojant kitus metodus reikalavimų tikrinimas yra empirinis t.y. jį atlieka ekspertai reikalavimų surinkimo ir peržiūros metu. Testavimo etape

reikalavimų tikrinimas yra atliekamas netiesiogiai t.y. per automatinių testų vykdymą (angl. *„Unit Testing“*) ir empiriškai testuojant sukurtą sistemą funkcionalumo požiūriu.

1.3 Modeliais grindžiama IS inžinerija

Modeliais grindžiama IS inžinerija [29] (angl. „*Model Driven Engineering – MDE“*) tai programinės įrangos kūrimo kryptis, kurioje pagrindinis dėmesys skiriamas įvairaus tipo modelių (pvz. dalykinės srities, architektūros, testavimo scenarijų ir kt.) sukūrimui ir jų taikymui standartiniuose programinės įrangos kūrimo gyvavimo ciklo etapuose (reikalavimų surinkimo, veiklos modeliavimo, projektavimo, programavimo, testavimo bei diegimo). Priklausomai nuo konkrečios MDE metodologijos modeliai gali būti naudojami kaip papildomas žinių šaltinis padedantis geriau išanalizuoti ir suprasti dalykinę sritį, vartotojo reikalavimus ir keliamus tikslus bei perteikti šiuos tikslus visiems komandos nariams arba kaip tiesioginis žinių šaltinis programinio kodo generavimui.



Pav. 1. Modeliais grindžiamų PĮ kūrimo metodų hierarchija [7]

MDE metodologiją realizuojantys PĮ kūrimo įrankiai dažniausiai turi integruotas programinio kodo generavimo iš modelių funkcijas. Pagal apibrėžimą, modeliais grindžiamos inžinerijos metodas turi aprašyti šiuos elementus: modeliavimo kalbas, modelius, transformacijas tarp modelių bei modelių sudarymo procesus [20]. Šie elementai turi būti detaliai specifikuoti,

siekiant realizuoti konkretų MDE metodą. Visos *modeliavimo kalbos* gali būti suskirstytos į dvi pagrindines grupes [21], [90]:

- Nuo dalykinės srities priklausomos modeliavimo kalbos [14] (angl. *„Domain Specific Modelling Languages – DSML“*). Šio tipo modeliavimo kalbos yra nesudėtingos, sudarytos iš nedidelio kiekio elementų, gerai suprantamos specifinėje srityje dirbantiems specialistams [86]. Šių kalbų privalumas yra tas, jog jos gali būti taikomos tiksliai specifikuoti siauros specializacijos dalykines sritis bei šioms sritims kuriamą programinę įrangą. Naudojant šio tipo modeliavimo kalbas, surinkti reikalavimai yra tikslūs, nedviprasmiški, gali būti automatizuotai transformuojami į kitos notacijos kalbas. Taikyti šias kalbas įvairialypėms dalykinėms sritims specifikuoti yra netikslinga, nes dėl kalbos ribotumo neįmanoma sukurti pilnos ir išsamios sistemos specifikacijos.
- Bendros paskirties modeliavimo kalbos [30] (angl. *„General Purpose Modeling Languages – GPML“*). Šio tipo modeliavimo kalbos skirtos aprašyti sudėtingas ir dideles sistemas nepriklausomai nuo dalykinės srities specifikos. Pagrindinis šių kalbų trūkumas yra jų sudėtingumas ir didelis elementų kiekis. Kalbas sudėtinga išmokti, nes jos turi sudėtingą sintaksę ir semantiką, egzistuoja daug elementų, kurie nėra naudojami konkrečios dalykinės srities modeliavimui, o specifinių elementų gali trūkti. Viena iš populiariausių šio tipo kalbų yra UML [68] (angl. „*Unified Modelling Language“*). Ši kalba gali būti išplėsta papildomais konkrečiai dalykinei sričiai būtinais elementais. UML kalbą atnaujina, tobulina ir vysto OMG (angl. *„Object Management Group“)* organizacija. Kalba yra standartizuota, sukurtas jos apsikeitimo formatas (XMI [69]) bei vieninga sertifikavimo sistema.

Kitas privalomas MDE metodų elementas yra *modelis*. Pagal apibrėžimą modelis yra „Supaprastintas sudėtingos esybės ar proceso aprašas (pvz. formulių rinkiniu), atvaizdas ar pan.“ [53]. Programinės įrangos kūrimo aspektu modeliai gali būti skirstomi į du tipus: verslo logikos (reikalavimų)

modeliai bei techniniai projektavimo modeliai. Verslo logikos modelius kuria sistemų analitikai interpretuodami iš užsakovo gautą informaciją. Techninės architektūros modelius kuria sistemų architektai bei programuotojai, remdamiesi sistemų analitiko sudarytais reikalavimų modeliais. Naudojamų modelių tipas, paskirtis, abstraktumo lygis yra skirtingai specifikuojami kiekvieno MDE metodo. Trečias MDE metodų elementas yra *modelių transformavimo procedūros*. Modelių transformavimas yra procesas, kurio metu vieno modelio elementai yra konvertuojami į kito modelio elementus. Šis procesas gali būti realizuotas tik tuo atveju, jeigu yra sukurtas modelių elementų sąsajų žemėlapis t.y. nurodymai, kaip ir kokiomis sąlygomis vieno modelio elementus atvaizduoti kitame modelyje. Egzistuoja du modelių transformavimo tipai: horizontalus ir vertikalus.

- Horizontalus transformavimas realizuojamas tame pačiame abstrakcijos lygmenyje tik skirtinga notacija arba pjūviu. Šio tipo transformacijos yra atliekamos naudojant sąsajų žemėlapius, kurie apibrėžia vieno modelio elementų atvaizdavimo į kito modelio elementus taisykles.
- Vertikalus transformavimas realizuojamas skirtingame abstrakcijos lygmenyje, bet naudojant tą pačia notaciją ir pjūvį. Minėta transformacija gali būti vieno iš dviejų tipų: iš didesnio abstrakcijos lygmens į mažesnį ir atvirkščiai. Pirmuoju atveju pradinis modelis turi būti detalizuojamas t.y. papildomas detalizuojančiai elementais (angl. *„tagging“*). Antruoju atveju pagal apibrėžtas taisykles dalis pradinio modelio elementų yra pašalinami.

MDE metodo *procesas* apibrėžia modelių kūrimo procedūras ir jų eiliškumą, modelių tikrinimo procedūras[74], modelių transformacijas ir jų eiliškumą, procesuose veikiančius aktorius ir kt. 2001 metais OMG pristatė MDE realizavimo metodologiją pavadintą MDA [59] (angl. *„Model Driven Architecture“*). MDA detalizuoja MDE koncepciją, bet tai nėra taikomoji MDE realizacija. Pagrindinis principas, kuris yra akcentuojamas MDA koncepcijoje - atskirti kuriamos programinės įrangos funkcinių reikalavimų modelius nuo jų realizavimo modelių bei pateikti procedūras, kurios leistų

automatizuotai transformuoti šiuos modelius tarpusavyje. Tokiu būdu siekiama išspręsti keletą programinės įrangos kūrimo problemų tokių kaip: sumažinti specialistų modeliavimo kalbų mokinimosi kaštus, išvengti loginių trūkių tarp modelių, panaudoti reikalavimų modelius kelių skirtingų platformų PĮ kūrimui ir kt.

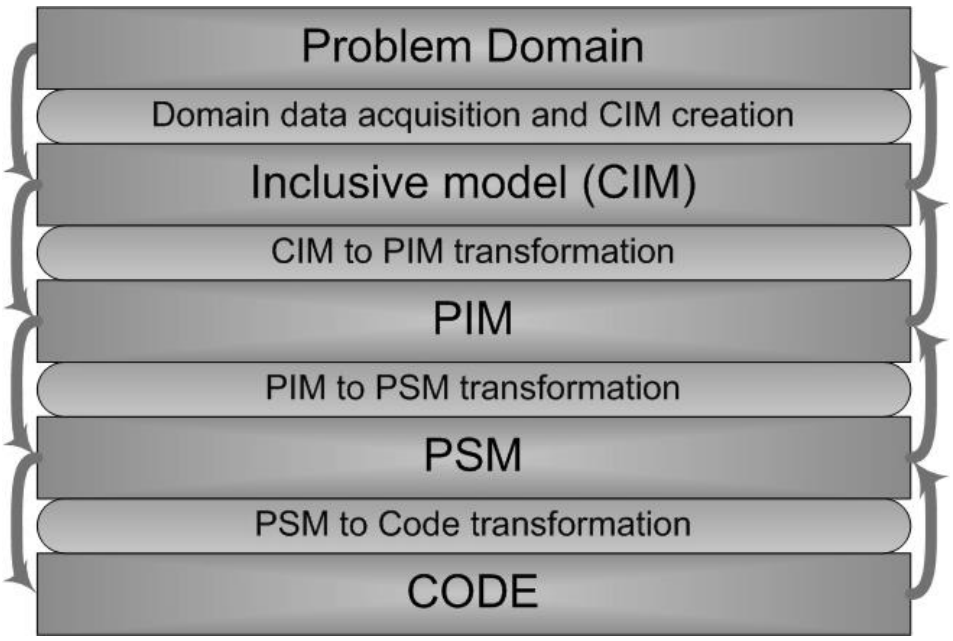
1.3.1 MDA

Modeliais grindžiamos architektūros (MDA) pagrindinė idėja yra atskirti sistemos funkcionalumą aprašančius modelius nuo modelių aprašančių, kaip tas funkcionalumas turi būti įgyvendinamas (techninių realizavimo aspektų) t.y. atskirti „ką daryti“ nuo „kaip daryti“. MDA išskiria tris kokybiškai skirtingus modelių tipus (sluoksnius), kurie yra naudojami kuriant programinę įrangą:

- CIM (angl. *„Computation Independant Model*“) – nuo skaičiavimų nepriklausomas modelis. Tai sistemos funkcinių ir nefunkcinių reikalavimų modelis. Šio modelio sudarymo procedūros, jo struktūra, pakeitimų valdymas yra reikalavimų surinkimo ir valdymo etapų atitikmuo klasikiniuose programinės įrangos kūrimo metoduose. MDA nėra tikslios specifikacijos šio modelio struktūrai, taip pat sudarymo procedūroms [40], todėl CIM sudarymas yra įvairiai interpretuojamas skirtingų autorių [13], [79], [88]. CIM gali būti sudarytas iš aibės modelių [2], [12], [37], [43], kurie atskirai aprašo statinę ir dinaminę sistemos informaciją (SysML/UML modeliai, darbų sekų modeliai ir kt.).
- PIM (angl. *„Platform Independant Model“*) – nuo platformos nepriklausomas abstraktus modelis, kuris turi pakankamai informacijos, kad iš jo, transformacijos metu, būtų sukurtas specifinės platformos modelis (PSM). Nuo platformos nepriklausomas modelis apibūdina sistemos struktūrą ir funkcijas architektūriniame lygmenyje, bet ne jų realizavimo būdą.

- PSM (angl. *„Platform Specific Model“*)– specifinei platformai sudarytas modelis, kuris turi informaciją ne tik apie tai ką sistema turi daryti (funkcinė informacija), bet ir kaip turi daryti (techninio realizavimo informacija). Šis modelis yra sukuriamas transformuojant PIM modelį t.y. naudojant papildomas žymes apie objektų tipus, platformai būdingus objektų ryšius ir elementus.

Pagal pateiktas OMG specifikacijas [59] kuriant programinę įrangą, yra sudaromas vienas CIM modelis, kuriame turi būti specifikuoti visi sistemos reikalavimai ir procesai, šio modelio pagrindu, naudojant transformacijas yra sukuriamas vienas PIM modelis. PSM modelių skaičius yra neribojamas (bet nemažiau kaip vienas) ir priklauso nuo konkretaus projekto poreikių. Kuriant programinę įrangą, remiantis MDA principais, procesas yra pradedamas nuo vartotojo reikalavimų specifikavimo ir CIM modelio kūrimo. Šis etapas yra empiriškas t.y. sistemų analitikas interpretuoja vartotojo pateikiamą informaciją apie sistemos reikalavimus ir formalizuoja juos naudojantis pasirinkta modeliavimo kalba.



Pav. 2. MDA modeliai ir procesas

Sukurtas modelis yra transformuojamas į architektūrinį modelį (PIM). Minėta transformacija gali būti atliekama automatiniu būdu su sąlyga, kad CIM modelis buvo aprašytas naudojant formalias modeliavimo kalbas ir kad

egzistuoja CIM į PIM sąsajų žemėlapiai. PIM į PSM transformacijos yra atliekamos automatiškai, pakeičiant bazinius PIM tipus į konkrečios platformos specifinius tipus ir ryšius. Programinio kodo generavimas iš PSM modelio taip pat yra atliekamas automatiškai naudojantis kodo generavimo įrankiais. Detalesnė įrankių apžvalga ir palyginimas yra pateiktas kitame skyriuje.

1.3.2 MDA įrankiai

MDA pagrindu sukurtuose programinės įrangos kūrimo metoduose yra naudojami dalinai ar pilnai MDA procesą automatizuojantys CASE įrankiai (pvz. "AndroMDA" [4], "Enterprise Architect" [83], "Acceleo" [85], "MagicDraw" [63]). Įrankiai, kurie yra sukurti laikantis atvirų OMG modeliavimo standartų, suteikia galimybę vienu įrankiu sukurtus modelius naudoti kitame įrankyje, tai leidžia apjungti kelių gamintojų įrankius į vientisą MDA procesą realizuojančią technologinę grandinę. MDA įrankiai gali būti skirstomi pagal šias atliekamas funkcijas: modelių testavimo, kūrimo, analizės, transformavimo, atvirkštinės inžinerijos, meta-valdymo. Lentelėje žemiau, remiantis UML modelių palyginimo tyrimu [16], yra pateiktas apibendrintas modeliavimo įrankių palyginimas pagal UML kalbos realizavimo lygį.

Lentelė 2. UML įrankių palyginimas

Pavadinimas	**Gamintojas**	**UML įvertinimas**	**Kodo generavimas**	**UML versija**	**XMI versija**
MagicDraw	**NoMagic**	**78.98%**	**23.00%**	**2.1**	**2.0/2.1**
Visual Paradigm for UML	Visual Paradigm	74.10%	40.00%	2.x	1.0,1.2,2.1
Enterprise Architect	Sparx Systems	71.44%	31.00%	2.1	1.3/2.1
Umodel	Altova	63.38%	20.00%	2.2	2.1
Together	Borland	62.21%	21.00%	2.0	2.0
Visual UML	Visual Object Modelers Inc.	57.86%	28.00%	1.x/2.0	1.0/1.1
Topcased	Topcased.org	55.31%	0.00%	2.0	
Poseidon	Gentleware	55.20%	28.00%	9 UML diagramos	1.2
Metamill	MetaMill	54.35%	22.00%	2.1	1.1, 1.2, 1.3, 2.0
Objecteering	Objecteering Software	50.85%	2.00%	2.0	

Kaip galima matyti iš informacijos pateiktos lentelėje pagal UML kalbos realizavimo lygį aukščiausias įvertinimas yra skirtas firmos *"NoMagic"* modeliavimo paketui *"MagicDraw"*. Šis paketas yra vienas iš įrankių, kurio pagalba yra atliktas disertacijos eksperimentinis tyrimas.

1.3.3 Vykdomasis UML

Vykdomasis UML [61] (angl. *„Executable UML"*) plačiąja prasme yra tokia UML modelių panaudojimo programinės įrangos kūrime koncepcija, kuri traktuoja modelius, kaip aukšto abstrakcijos lygmens programavimo kalba sukurtus artefaktus t.y. UML modeliai yra sukuriami, kompiliuoti, vykdomi (angl. *"run"*) ir testuojami kaip programinės įrangos kodas. Būtina minėtos koncepcijos realizavimo sąlyga yra ta, jog UML modeliai detaliai aprašytų probleminės srities struktūrinę bei elgsenos informaciją. Panaudojant vykdomąjį UML yra kuriami dalykinių programų prototipai, kurių pagalba gali būti tikrinami bei patikslinami vartotojo reikalavimai arba patys prototipai tobulinami iki pilnai funkcionuojančio produkto. Vykdomajame UML yra naudojamos programavimo (modeliavimo) kalbos, kurių abstrakcijos lygmuo yra aukštesnis už trečios kartos programavimo kalbų (3GL pvz. C++, C#, Java) abstrakcijos lygmenį. Tokiu būdu sukurti vykdomieji modeliai yra nepriklausomi nuo konkrečios platformos ir gali būti plačiai naudojami skirtingų operacinių sistemų programinei įrangai kurti. Vykdomajame UML naudojama veiksmų programavimo kalba (angl. *„Action Language"*) suteikia galimybę vykdyti ir testuoti modelių pagalba aprašytus procesus t.y. modeliai nėra naudojami kaip pagrindas programiniam kodui, jie yra naudojami kaip programinis kodas, kas sumažina loginį trūkį tarp dokumentacijos (klasiniu požiūriu modeliai yra dokumentacijos dalis) ir programavimo kalbos. Pagrindiniai modelių tipai, kurie yra naudojama vykdomuosiuose UML metoduose yra šie:

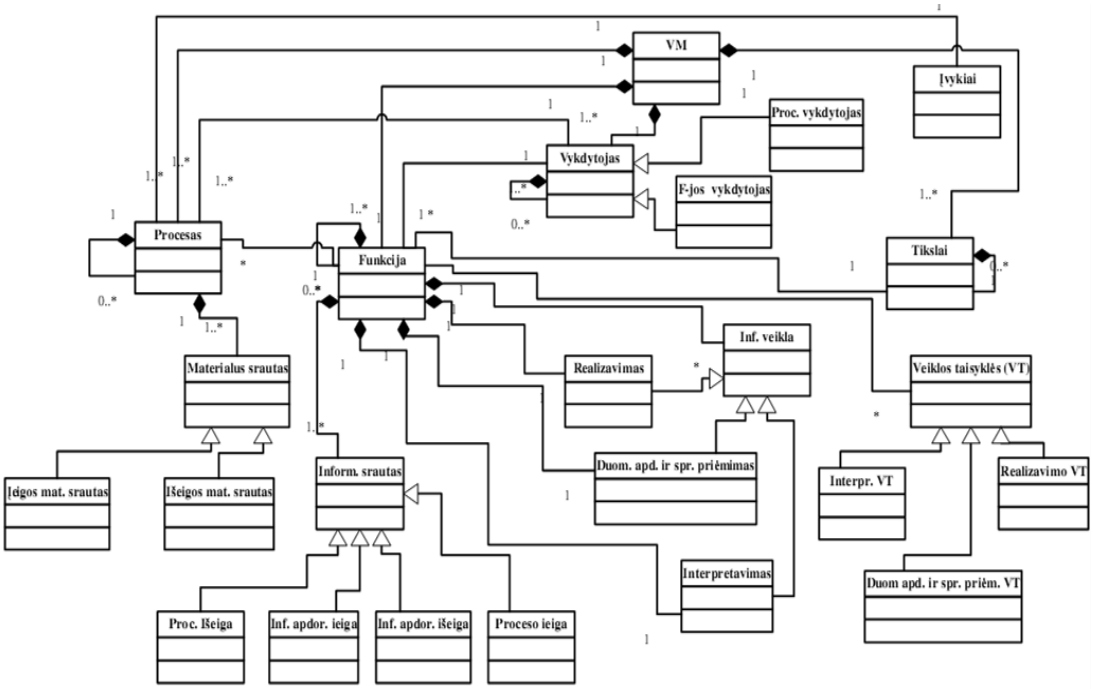
- Klasių modeliai (struktūros), kurie aprašo probleminės srities objektus bei ryšius.
- Būsenos/veiklų modeliai, kurie aprašo probleminės srities objektų būsenas, įvykius, būsenos pasikeitimo taisykles ir pan.

OMG apibrėžia savo „Executable UML“ metodą [66], kuris remiasi šiais standartais: UML, fUML, Alf. *fUML* yra UML pagrindu sukurta modelių aibė, kuri yra naudojama aprašyti kuriamos dalykinės programos struktūros ir elgsenos informaciją. *Alf* yra OMG pasiūlyta veiksmų programavimo kalba (angl. *„Action Language for fUML“*) [77]. OMG pasiūlytas „Executable UML“ gali būti integruotas į MDA grindžiamus metodus, su tikslus realizuoti PIM/PSM modelių transformavimą į veikiančią programinę įrangą.

1.4 Žiniomis grindžiama IS inžinerija

Žiniomis grindžiamos IS inžinerijos [53] pagrindinis bruožas yra veiklos metamodelio (angl. *„Enterprise Meta-Model“*) ir veiklos modelio (angl. *„Enterprise Model“*) naudojimas programinės įrangos kūrimo ciklo etapuose. Remiantis veiklos metamodeliu yra sukuriamas konkrečios organizacijos veiklos modelis, kuriame saugomos programinės įrangos kūrimo procesui būtinos dalykinės srities žinios (aktoriai, veiklos procesai, funkcijos, tikslai, srautai). Veiklos modelio užpildymas dalykinės srities žiniomis gali remtis klasikinių IS inžinerijos metodų procedūromis t.y. reikalavimai yra dokumentuojami pasitelkiant tekstinius aprašus, diagramas, vartotojo istorijas, panaudos atvejus ir kt. Sekantis etapas yra surinktos informacijos transformavimas į veiklos modelio objektus, remiantis veiklos metamodelyje apibrėžtomis taisyklėmis. Informaciją perkėlus į veiklos modelį gali būti atliekamas žinių integralumo tikrinimas metamodelio struktūros atžvilgiu. Veiklos modelį ir veiklos metamodelį tikslinga integruoti į tradicinius IS kūrimo metodus, nes tai teigimai įtakotų reikalavimų surinkimo ir veiklos modeliavimo etapus, kas leistų sumažinti IS kūrimo kaštus bei laiko sąnaudas. Egzistuoja aibė veiklos metamodelių standartų, tokių kaip IDEF [52], OMT [73], CIMOSA [32], GERAM[36], dodaf[15], UEML[87] bei kitų mokslininkų sukurtų veiklos modelių (pvz. veiklos taikyklėmis grindžiamų [42]). Didelė šių metodų įvairovė apsunkina programinės įrangos kūrimo įrankių, skirtų organizacijos veiklos modeliavimui ir programinės įrangos projektavimui, suderinamumą. Siekiant išspręsti minėtą problemą mokslinės grupės,

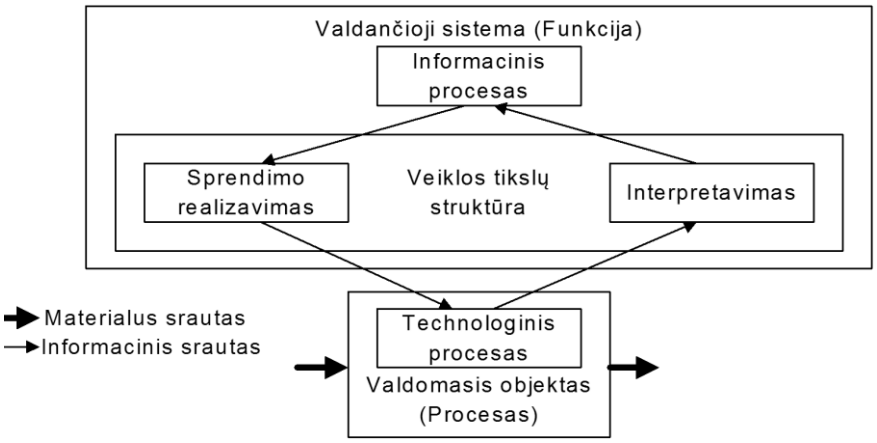
atviruosius standartus kuriančios organizacijos (pvz. OMG), programinės įrangos gamintojai („Oracle“, „Microsoft“) bei pagrindinės standartizacijos organizacijos (ISO, CEN) stengiasi sukurti bendrą standartą, kuriuo grindžiant būtų kuriami nauji organizacijos veiklos modeliavimo metodai, kalbos bei su jais suderinta programinė įranga. Šių pastangų rezultatu tapo ISO CEN ENV 40003 standartas [19], kuris sukurtas ISO 15704, ISO 14258 bei CEN ENV 12204 [18] standartų pagrindu. Nei vienas iš nagrinėjamų standartų pilnai neatitinka organizacinės sistemos projektuotojo poreikius ne tik dėl riboto konstruktų kiekio ir funkcijų, bet ir todėl, kad nei viename standarte nėra apibrėžta veiklos modelio sudėtis, būtina ir pakankama programinės įrangos projektiniams modeliams ir programiniam kodui generuoti. Taip pat veiklos modelių turinys ne visada tikrinamas formalių kriterijų atžvilgiu (pvz. veiklos modelio sudėtis nėra tikrinama semantiniu aspektu). CEN ENV 12204 ir ENV 40003 standartuose organizacija nėra modeliuojama valdymo požiūriu. UEML yra naudojami organizacijos modeliavimo valdymo požiūriu principai, tačiau jie nėra išvystyti iki taikomojo lygmens. Siekiant pašalinti apibūdintus trūkumus VU mokslininkai, bendradarbiaudami su kitomis institucijomis, sukūrė veiklos metamodelį [25], kurio sudėtis yra grindžiama valdymo teorijos [28] principais bei aukščiau išvardintų standartų elementais. Detalizuotas VU mokslininkų sukurto metamodelio klasių modelis yra pavaizduotas 4 pav.



Pav. 3. VU mokslininkų sukurto veiklos metamodelio klasių modelis

Pagrindinis konceptas, kuris yra naudojamas minėtame veiklos metamodelyje yra elementarus valdymo ciklas (EVC). Šio koncepto esminis teiginys yra: kiekvienas procesas turi turėti jį valdančią funkciją. Esminiai metamodelio elementai – *procesas* ir *funkcija*. *Procesas* – atvaizduojamas kaip klasė, sauganti veiksmo, kuris yra vykdoma siekiant gauti konkretų materialų rezultatą, aprašymą. *Procesas* yra susietas su *materialiu srautu* t. y. *proceso* įeigos ir išeigos srautais. *Procesą* vykdo *aktorius*. *Funkcija* yra veikla, kuri kontroliuoja *procesą*. Tai sudėtinis elementas, galintis turėti hierarchinę struktūrą. Kiekvienas *procesas* privalo turėti bent vieną valdančiąją *funkciją* t. y. nekontroliuojamų *procesų* pagal šio veiklos metamodelio taisykles negali būti. *Procesas* teikia *funkcijai* valdymui reikalingus duomenis, kurie yra apdorojami funkcijos informacinių veiklų. Funkcijos *informacinės veiklos (IP, interpretavimas, realizavimas)* informacija keičiasi per *informacinių srautų* objektus. *Interpretavimas* yra atvaizduojamas kaip klasė, kuri saugo informaciją (taisykles) apie iš *procesų* gautų duomenų transformavimą. Šių taisyklių paskirtis – transformuoti *proceso* perduodamus duomenis į *funkcijos* veikloms priimtiną formatą. *IP* yra funkcijos elementas, kuris atlieka informacijos apdorojimo bei sprendimų priėmimo operacijas. *Informacinės veiklos*

organizuojamos pagal egzistuojančias arba naujas *verslo taisykles*. *Funkcijos* per savo veiklą privalo įgyvendinti vieną ar kelis organizacijos *tikslus*. Paveiksle žemiau yra pavaizduoja principinė EVC schema, taikoma technologinių procesų aprašymui.



Pav. 4. Elementaraus valdymo ciklo principinė schema [27]

P*rocesus* ir *funkcijas* atlieka *aktoriai* (specialistai, programinė bei techninė įranga). Aprašytas organizacijos veiklos modelis buvo naudojamas kituose moksliniuose tyrimuose: saugoti ir transformuoti žinias surinktas darbų sekų modeliais [49], [50], saugomų žinių transformavimui į klasių modelius [82], integracijai su Henderson IS kūrimo ciklo etapais [47]. Kuriant žiniomis grindžiamą MDA metodą bazinis veiklos metamodelis buvo išplėstas papildomais elementais. Išplėstas veiklos modelis yra aprašytas antrajame skyriuje.

1.5 Išvados

Remiantis atliktu tyrimu bei praktinio darbo patirtimi galima teigti jog, „Plan Driven“ metodų reikalavimų surinkimo ir valdymo procesai yra formalizuoti ir nelankstūs, netinkami greitai kintančios aplinkos sąlygomis, tačiau jie turi griežtai apibrėžtas reikalavimų surinkimo ir tikrinimo procedūras, kurių paskirtis yra užtikrinti surinktų reikalavimų kokybę. Reikalavimų tikrinimo procesas šiuose metoduose yra empirinis, tačiau jis yra kontroliuojamas per dokumentų šablonus ir nustatytas procedūras. Agiliųjų metodų reikalavimų surinkimo ir valdymo procesai yra mažai formalizuoti, remiasi juos atliekančių komandų kompetencija ir patirtimi. Tai suteikia

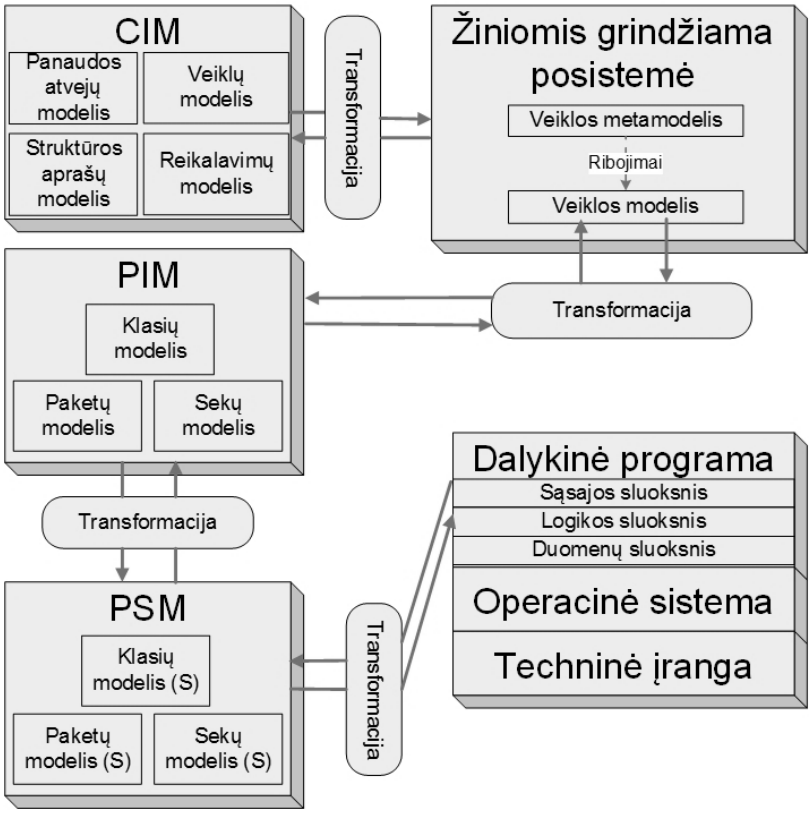
galimybę greitai reaguoti į pasikeitusius vartotojo poreikius, tačiau apsunkina projekto kontrolę ir būtinų resursų kiekio nustatymą. Reikalavimų tikrinimas yra atliekamas empiriškai, tačiau į šį procesą yra įtraukiami ir programuotojai t.y. sudaromos grįžtamojo ryšio sąlygos. Automatinis tikrinimas kaip ir „Plan Driven“ metoduose yra atliekamas netiesiogiai (ne patys reikalavimai yra tikrinami, o jų pagrindu sukurta programinė įranga) programinės įrangos testavimo metu (angl. *„Unit Testing“*). Pastebėti reikalavimų trūkumai yra šalinami dinamiškai (ne visada laikantis nustatyto proceso), taisant programinį kodą ir vėl testuojant. Žiniomis grindžiami IS inžinerijos metodai įgyvendina galimybę reikalavimus tikrinti automatiškai, remiantis metamodelio apibrėžtomis taisyklėmis, tačiau veiklos modeliavimo standartų siūlomų veiklos modelių sudėtis yra ribota, konstruktai dažnai nėra pakankami pilnai specifikuoti dalykinę sritį. Kita problema su kuria susiduriama naudojant šio tipo IS inžinerijos metodus yra reikalavimų saugojimo formato pakeitimas į vidinį veiklos modelio formatą (pvz. reikalavimai yra surenkami naudojantis WF diagramomis, tekstiniais failais, kuriuos reikia konvertuoti į veiklos modelio elementus). Transformacijoms realizuoti turi būti sukurti sąsajų žemėlapiai, transformavimo algoritmai bei su veiklos modeliu dirbantys analitikai privalo turėti specifinių žinių apie jo sudėtį ir panaudojimą. Taigi minėtos priežastys apsunkina veiklos modelių integravimą į tradicinius PĮ kūrimo metodus, ko pasėkoje atsisakoma tokių veiklos modelio naudojimo privalumų kaip mažesnio projektavimo klaidų, atsirandančių dėl netikslių ar nepilnų reikalavimų, kiekio, greitesnio reikalavimų surinkimo ir tikrinimo proceso. Metoduose, kuriuose yra taikomi žiniomis grindžiamos IS inžinerijos principai, reikalavimų tikrinimas atliekamas dviem būdais: empiriniu, kaip ir tradiciniuose metoduose (analitikas vertina reikalavimus jų surinkimo metu, remdamasis savo patirtimi), bei formaliu, kuris yra grindžiamas veiklos metamodelio semantinėmis taisyklėmis. MDA pagrindu sukurti metodai apibrėžia reikalavimų surinkimo koncepciją, kuri remiasi CIM modelio naudojimu t.y. reikalavimai yra surenkami ir formalizuojami panaudojant įvairaus tipo grafinius modelius. Modeliai yra sudaromi empiriškai ir

pagrindiniai kokybę užtikrinantys faktoriai yra sistemų analitiko patirtis ir kompetencija. Savo ruožtu MDA pagrindu sukurti metodai sumažina žmogiškųjų veiksnių įtaką programinės įrangos kūrimo etapams informacijos perdavimo tarp skirtingų etapų metu, kadangi informacijos pernešimas (transformavimas) turi būti atliekamas automatiškai, pagal iš anksto nustatytas taisykles. Apibendrinus galima teigti, jog klasikiniai programinės įrangos kūrimo metodai reikalavimų surinkimo ir tikrinimo problemas sprendžia dviem būdais: griežtai apibrėždami formalias procedūras, kuriomis turi vadovautis šioje srityje dirbantys specialistai arba kartojant reikalavimų surinkimo ir tikrinimo etapus, kol pasiekiamas patenkinamas rezultatas. Abu būdai savo esme yra empiriniai t.y. remiasi tai atliekančių žmonių kompetencija ir patirtimi. Šią problemą įmanoma išspręsti IS kūrimo metoduose panaudojant žiniomis grindžiamos IS inžinerijos principus, tačiau veiklos modelio integracija į klasikinius IS kūrimo metodus yra sudėtingas procesas, kuris reikalauja papildomų resursų. Metodai, į kurių sudėtį gali būti nuosekliai integruotas veiklos modelis, turi remtis iš principo panašia koncepcija t.y. plačiai taikyti modelius ir modelių transformacijas. MDA koncepcija ir apibrėžia tokio tipo metodus. Šių metodų reikalavimų surinkimo procesas yra empirinės prigimties ir iš esmės remiasi įvairios notacijos (WF, SysML, UML) modelių sudarymu. Taigi apjungus žiniomis grindžiamos IS inžinerijos principus su MDA koncepcija yra įmanoma sukurti metodą, kuris įgalina naudoti formalų ir automatinį reikalavimų tikrinimo procesą (žiniomis grindžiamos IS inžinerijos konceptas) bei nuoseklų modelių transformavimo procesą (MDA konceptas). Disertacijos autoriaus (su bendraautoriais) atliktų tyrimų, dėl veiklos modelio panaudojimo programinės įrangos kūrimo procese, rezultatai yra publikuoti straipsniuose „Enterprise Model and ISO Standards Based Informations System's Development Process“ (žr. proceedings publikacijų sąrašą nr.: 1), „Knowledge-Based Approach to Business and IT Alignment Modelling“ (žr. žurnalinių publikacijų sąrašą nr.: 5). Jungtinis MDA ir žiniomis grindžiamos IS inžinerijos metodas yra aprašytas antrame skyriuje.

2. ŽINIOMIS GRINDŽIAMAS MDA METODAS

2.1 Pagrindiniai žiniomis grindžiamo MDA metodo principai

MDA yra programinės įrangos kūrimo koncepcija, kuri remiasi modelių naudojimu programinės įrangos funkcinių bei nefunkcinių reikalavimų surinkimui, taip pat programinės įrangos projektavimui bei programinio kodo generavimui. MDA apibrėžia tik bendrus modelių panaudojimo principus, bet nepateikia detalių specifikacijų. Remiantis tyrimu [5], daugumai MDA pagrindu sukurtų IS inžinerijos metodų reikia tobulinti šias sritis: reikalavimų surinkimas, CIM konstravimas bei sistemos modelių tikrinimas dalykinės srities atžvilgiu.



Pav. 5. Koncepcinė MDA metodo, praplėstos žinių posisteme, schema

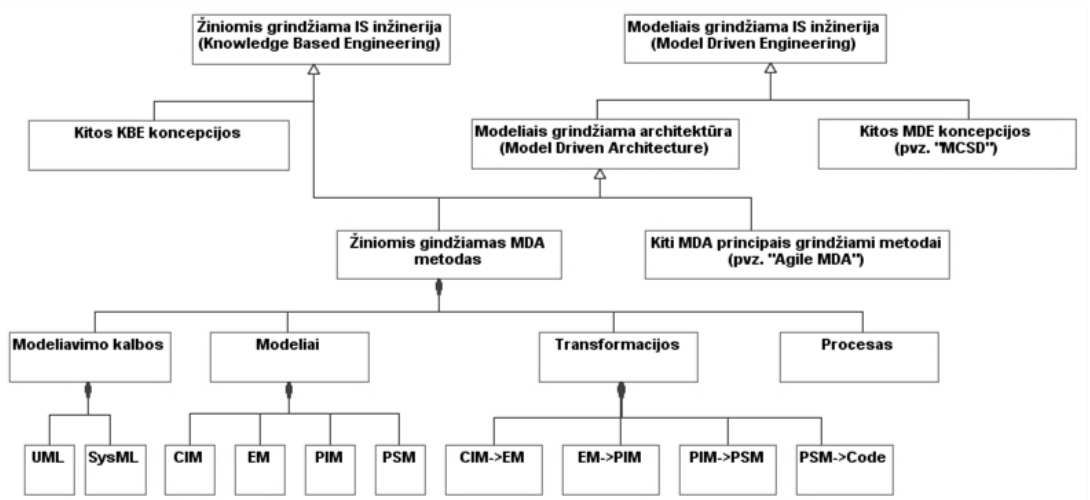
Minėtos problemos gali būti išspręstos papildžius MDA koncepciją žiniomis grindžiamos IS inžinerijos principais, kurių realizacija – veiklos modelis (veiklos metamodelio pagrindu) ir transformavimo bei tikrinimo algoritmai. Modelių transformavimo ir tikrinimo algoritmai yra naudojami atlikti transformacijas tarp MDA modelių (CIM, PIM) bei veiklos modelio. Veiklos

modelis yra „įterpiamas“ tarp CIM ir PIM modelių ir yra atsakingas už kuriamos informacinės sistemos vartotojo reikalavimų ir veiklos procesų tikrinimą. Pagrindinis žiniomis grindžiamo MDA metodo privalumas yra tai, jog MDA CIM modelyje surinkti reikalavimai gali būti tikrinami veiklos metamodelio taisyklių atžvilgiu, tokiu būdu kiti MDA modeliai (PIM bei PSM) yra kuriami su mažesne empirinių veikslių įtaka (tokia kaip nepakankama sistemų analitiko ar užsakovo kompetencija ir iš to sekantys nepilnai surinkti ar nenuoseklūs reikalavimai). Pirmoje dalyje atlikta analizė parodė, jog minėtų veiksnių įtaka gerai atsiskleidžia naudojant tradicinius IS inžinerijos metodus, kurie yra grindžiami empiriniu dalykinės srities reikalavimų surinkimo procesu. Surinkti reikalavimai dažnai būna fragmentiški, nepilni, prieštaringi. Taip pat gali egzistuoti aplinkybės, kai vartotojo reikalavimai neatitinka formalių verslo kriterijų ir reguliavimų. Šiuos neatitikimus išaiškinus vėlesniuose programinės įrangos kūrimo etapuose, visam programinės įrangos kūrimo procesui daromas itin neigiamas poveikis. Apibendrindami galime teigti, jog žiniomis grindžiamas MDA metodas apima tradicinės MDA koncepcijos modelių naudojimo IS inžinerijoje principus bei išplečią šią metodiką įvesdamas naują elementą - veiklos žinių modelį, kurio pagalba atliekamas empiriškai surinktų reikalavimų tikrinimas.

2.2 Žiniomis grindžiamas MDA metodas IS inžinerijos metodų hierarchijoje

Žiniomis grindžiamas MDA metodas apjungia pagrindinius MDA bei žiniomis grindžiamos IS inžinerijos principus. MDE apibrėžia pagrindinius dalykinės srities modelių panaudojimo principus programinės įrangos kūrimo procese, o MDA metodas yra sukurtas vadovaujantis šias principais. MDA yra labiau detalizuotas nei MDE koncepcija, tačiau vis dar nepakankamai detalus, kad jį patį būtų galima naudoti tiesiogiai kaip PĮ kūrimo metodą. Detalizuojant MDA yra kuriami metodai, kurie yra tiesiogiai naudojami PĮ kūrimui (pvz. *„Agile MDA“* [54]), šiems metodams priklauso ir žiniomis grindžiamas MDA metodas. Žemiau pateiktoje diagramoje yra pavaizduota žiniomis grindžiamo MDA metodo vieta MDE bei KBE metodų hierarchijoje. Žiniomis grindžiamo

MDA metodo sąntykis su modeliais grindžiamais ir žiniomis grindžiamais programinės įrangos kūrimo metodais yra pavaizduotas 6 paveiksle.

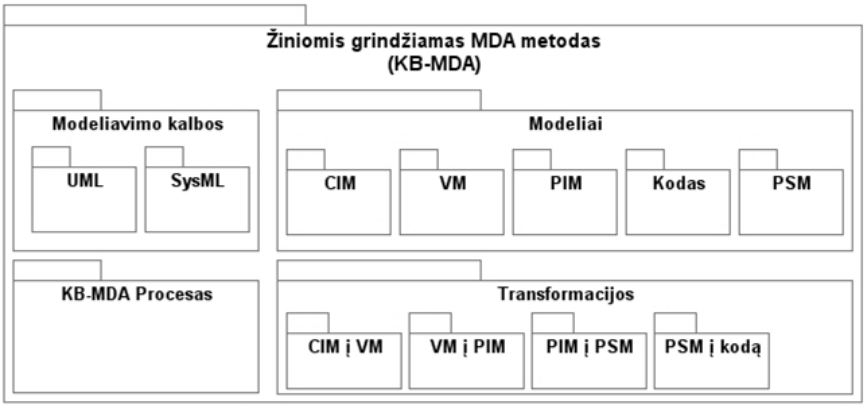


Pav. 6. Žiniomis grindžiamo MDA metodo vieta MDE ir KBE metodų hierarchijoje

Žiniomis grindžiamas MDA metodas išplečia MDA koncepciją papildydamas ją vienu modeliu (veiklos modeliu) bei dvejomis naujomis transformacijomis (CIM į EM ir EM į PIM) [46]. Šie elementai sudaro veiklos žinių posistemę.

2.3. Žiniomis grindžiamo MDA metodo sudėtis

Žiniomis grindžiamas MDA metodas naudoja SysML ir UML modeliavimo kalbas, standartinius MDA modelius (CIM, PIM, PSM) bei veiklos modelį (VM), standartines MDA transformacijas papildytas dvejomis naujomis transformacijomis (CIM į EM ir EM į PIM) bei aprašydamas savo specifinį procesą. Pagrindiniai žiniomis grindžiamo MDA metodo komponentai yra pavaizduoti 7 paveiksle.



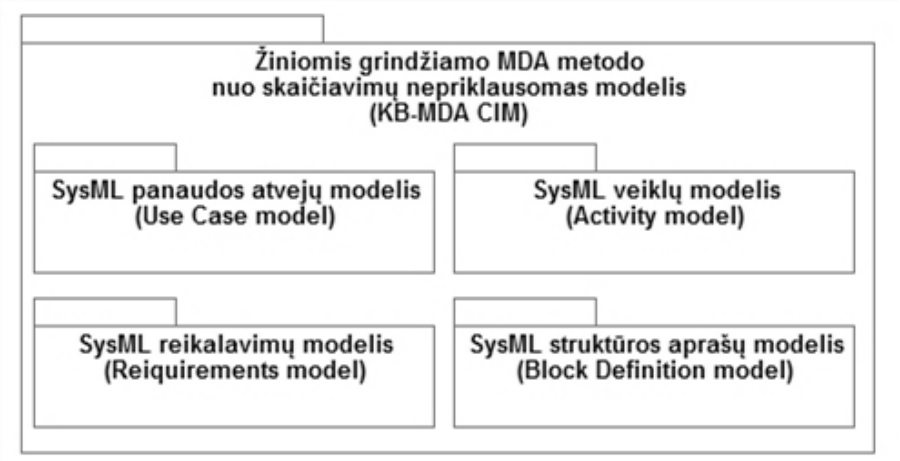
Pav. 7. Žiniomis grindžiamo MDA metodo sudedamieji komponentai

2.3.1 Modeliavimo kalbos

Žiniomis grindžiamo MDA metodo modeliams kurti yra pasirinktos dvi modeliavimo kalbos SysML [67], [89] ir UML. SysML kalba yra skirta veiklos modeliavimo ir vartotojo reikalavimų formalizavimui (vidinių CIM modelių sudarymui). UML kalba naudojama PIM ir PSM modelių sudarymui. Abi modeliavimo kalbos yra palaikomos OMG grupės ir yra suderinamos tarpusavyje, taigi įmanoma atlikti automatizuotas CIM į PIM transformacijas (SysML ir UML modeliai turi didelę dalį bendrų komponentų). Papildomas privalumas renkantis šias kalbas yra jų žinomumas ir pripažinimas *de facto* standartais tarp IT ir veiklos procesų modeliavimo specialistų.

2.3.2 Modeliai

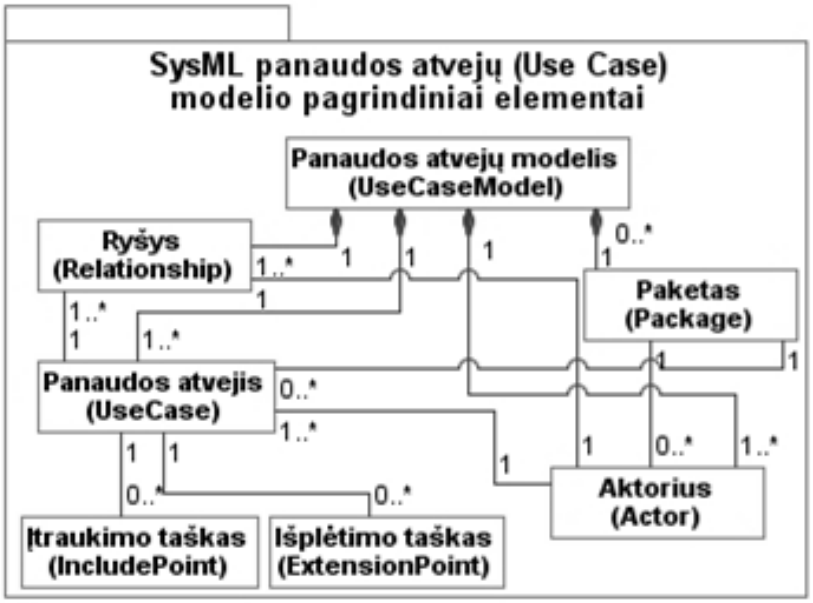
Žiniomis grindžiamas MDA metodas apibrėžia keturių tipų modelius: CIM, VM, PIM ir PSM. CIM modelis aprašo funkcinius ir nefunkcinius sistemos reikalavimus. Šį modelį sukuria sistemų analitikas kartu su užsakovo paskirtais darbuotojais, kurie turi perteikti sistemos funkcinius ir nefunkcinius reikalavimus. VM yra veiklos modelis, kurio struktūra yra pagrįsta veiklos metamodeliu, šis modelis yra generuojamas automatiškai iš CIM modelio. PIM yra sistemos architektūros modelis, kuriame atvaizduoti sistemos komponentai bei jų tarpusavio sąveika. Šiame modelyje nėra specifinių konkrečiai technologijai ar platformai būdingų elementų. Paveiksle žemiau yra pavaizduota CIM modelio struktūra. CIM modelis yra sudarytas iš keturių SysML modelių [45].



Pav. 8. Žiniomis grindžiamo MDA CIM modelio sudėtis

Panaudos atvejų (angl. *„Use Case“*) ir veiklų (angl. *„Activity“)* modeliai yra skirti surinkti dinaminę sistemos informaciją, struktūros aprašų (angl. *„Block Definition“)* modelis yra skirta statinės informacijos surinkimui, o reikalavimų (angl. *„Requirements“)* modelis yra naudojamas nefunkcinių reikalavimų specifikavimui. Detalesnis aprašymas yra pateiktas žemiau:

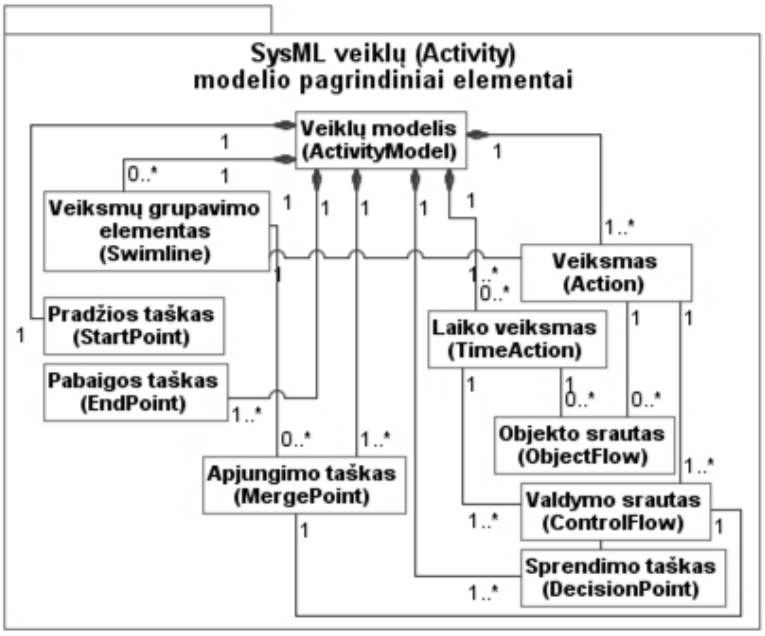
- *Panaudos atvejų* modeliai – šie modeliai skirti surinkti sistemos funkcinius reikalavimus bei pirminę sistemos elgsenos informaciją. Pagrindiniai šio tipo modelių elementai yra aktorius (angl. *„Actor“*), kuris nusako sistemos funkcijų vykdytoją ar naudotoją bei panaudos atvejis (angl. *„Use Case“*), kuris nusako aktoriaus atliekamą veiksmą. Panaudos atvejai gali būti susieti su aktoriais bei kitais panaudos atvejais. Panaudos atvejų modeliai dažniausiai yra pirmi sukurti modeliai, nuo kurių pradedamas reikalavimų surinkimo procesas. Šie modeliai naudojami kaip pagrindas kurti veiklų modelius.



Pav. 9. Pagrindiniai SysML panaudos atvejų modelio elementai

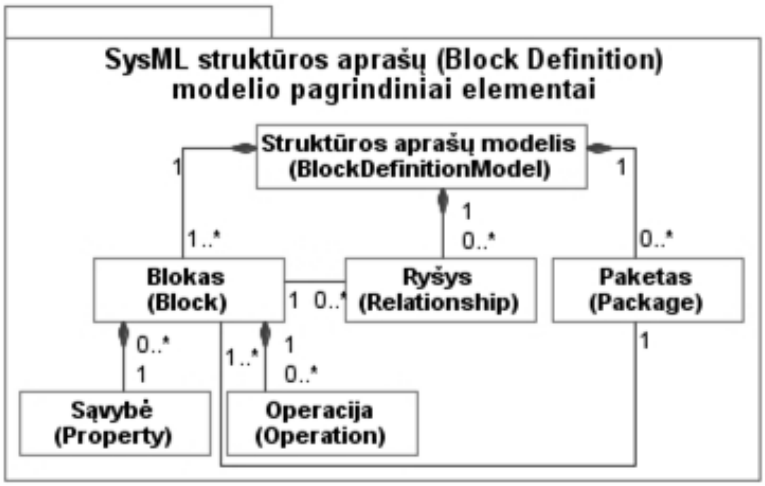
- *Veiklų* modeliai – šie modeliai skirti surinkti sistemos elgsenos informaciją. Pagrindinis elementas yra veiksmas (angl. *„Action“*), kuris nusako vieną aktoriaus atliekamą operaciją. Veiksmai yra susiejami kontrolės ir/arba objektų srautais. Modelis privalo turėti pradžios bei pabaigos elementus. Veiksmų išsišakojimai yra atvaizduojami naudojant

sprendimo bei apjungimo elementus (angl. „*Merge node*“, „*Decision node*“).



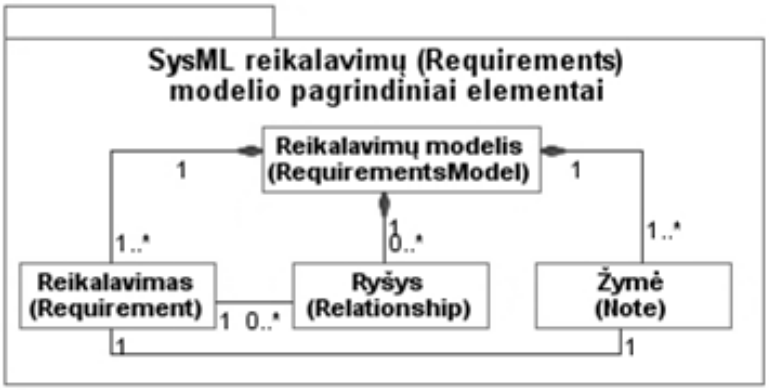
Pav. 10. Pagrindiniai SysML veiklų modelio elementai

- *Struktūros aprašų* modeliai – šie modeliai skirti surinkti informaciją apie statinę sistemos struktūrą. Pagrindinis modelio elementas yra blokas (angl. *„Block“*), kuris atvaizduoja tam tikrą vientisą sistemos komponentą. Blokai gali turėti savybes (angl. *„Property“*) bei operacijas (angl. *„Operation“*). Blokai yra susieti vieni su kitais ryšiais, tokiais kaip pvz.: asociacija (angl. *„Association“*), paveldėjimas (angl. *„Generalization“*).



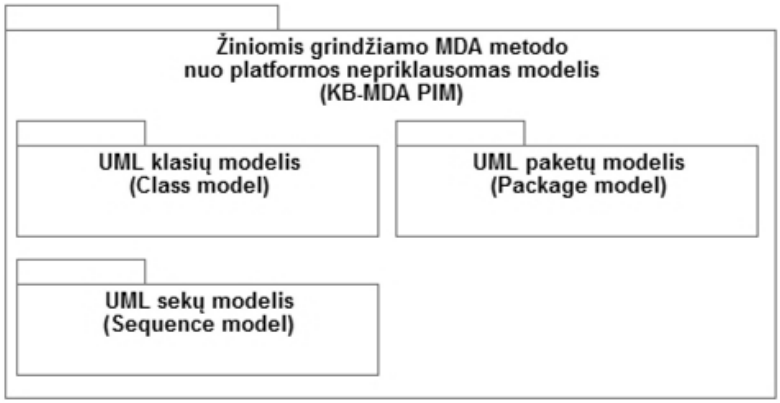
Pav. 11. Pagrindiniai SysML struktūros aprašų modelio elementai

- *Reikalavimų* modeliai – šie konkrečiu atveju yra skirti surinkti ir specifikuoti nefunkcinius sistemos reikalavimus. Pagrindinis elementas yra reikalavimas (angl. *„Requirement*“), tai elementas, kuris atvaizduoja tekstu aprašytą sistemos apribojimą. Reikalavimai gali būti susieti ryšiais (angl. *„Relatinship*“) ir sudaryti hierarchines struktūras.



Pav. 12. Pagrindiniai SysML reikalavimų modelio elementai

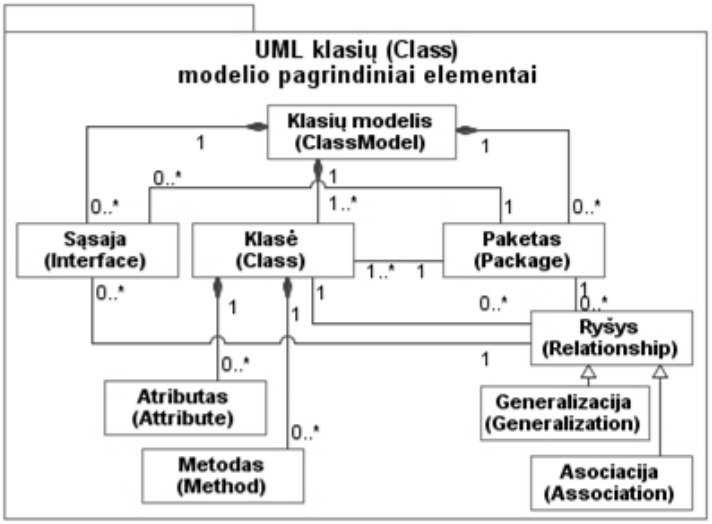
Aprašyti CIM modeliai įgalina surinkti statinę ir dinaminę kuriamos sistemos informaciją. Nefunkciniai reikalavimai yra surenkami naudojant SysML reikalavimų modelius, o taip pat papildomai anotuojant kitų modelių elementus žymėmis (angl. *„Note“*). Žymės yra skirtos surinkti specifinę modelių transformavimo informaciją (pvz. nusako ar veikla turi būti traktuojama kaip funkcija ar kaip procesas) ir kitą nestandartinę informaciją (pvz. konkrečius savybių tipus). Naudojant žymėse saugomą informaciją įmanoma surinkti nestandartizuotus duomenis, kurie gali būti interpretuojami ir naudojami veiklos žinių posistemės kituose žiniomis grindžiamo MDA metodo etapuose.



Pav. 13. Žiniomis grindžiamo MDA PIM modelio sudėtis

Žiniomis grindžiamo MDA metodo PIM modelis yra sudarytas iš trijų [6] UML modelių: klasių (angl. *„Class“*), sekų (angl. *„Sequence“*), paketų (angl. *„Package“*). Klasių modelis yra skirtas atvaizduoja struktūrinius sistemos elementus bei jų tarpusavio ryšius, sekų modelis yra skirtas atvaizduoti dinaminę sistemos informaciją, paketų modelis skirtas atvaizduoti sistemos suskirstymą į loginius blokus. Detalesnis aprašymas yra pateiktas žemiau:

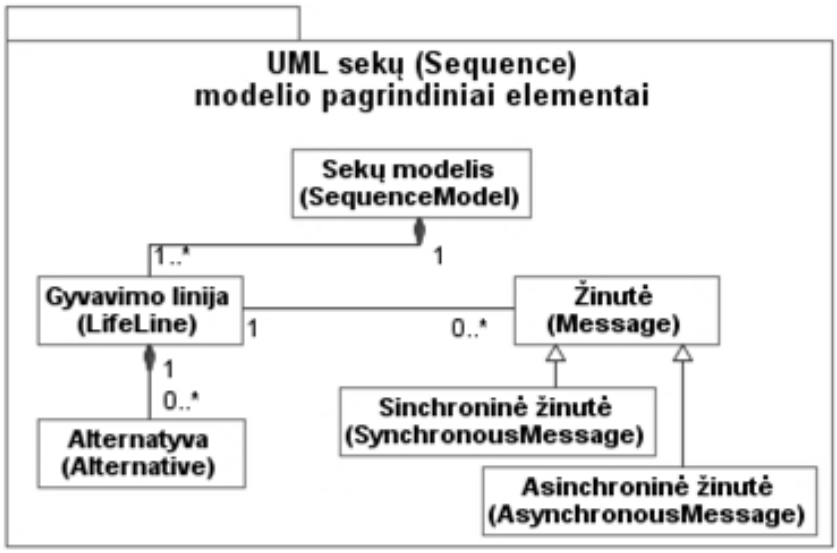
- *Klasių* modeliai – klasių modeliai atvaizduoja struktūrinę sistemos informaciją t.y. sistemos elementus bei jų tarpusavio ryšius. Pagrindinis modelio elementas yra klasė (angl. *„Class“*), kuri atvaizduoja konkretų sistemos elementą. Klasės gali turėti savybes (angl. *„Attribute“*) bei metodus (angl. *„Method“*). Klasės gali būti susietos ryšiais, tokiais kaip: asociacija (angl. *„Association“*), agregacija (angl. *„Aggregation“*), paveldėjimas (angl. *„Generalization“*). Klasių modeliai yra pagrindinis programinio kodo generavimo šaltinis.



Pav. 14. Pagrindiniai UML klasių modelio elementai

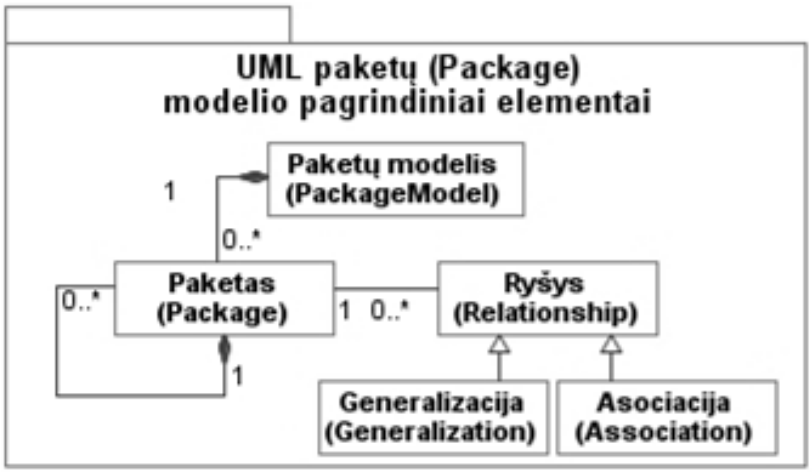
- *Sekų* modeliai – šie modeliai yra skirti atvaizduoti sistemos elgsenos informaciją. Sekų modeliai yra detalesni už veiklų modelius bei įgalina atvaizduoti įvykius nustatytu eiliškumu t.y. yra laiko dedamoji. Pagrindiniai šio tipo modelių elementai yra gyvavimo linija (angl. „*LifeLine“*), kuri nusako konkretaus objekto gyvavimo ciklą, asinchroniniai ir sinchroniniai pranešimai (angl.

„Synchronuos/Asynchronous Message“), kurie apibrėžia objektų bendravimą bei alternatyvos (angl. *„Alternative“*), kurios atlieka pasirinkimų valdymo funkciją.



Pav. 15. Pagrindiniai UML sekų modelio elementai

- *Paketų* modeliai – šie modeliai yra skirti atvaizduoti sistemą loginių posistemių aspektu. Paketai yra skirti grupuoti susijusios paskirties objektus (klases, sąsajas). Pagrindinis šio tipo modelių elementas yra paketas (angl. „*Package“*). Paketas gali būti sudarytas iš kitų paketų arba iš klasių (sąsajų). Paketai kaip ir klasės gali būti susieti ryšiais.

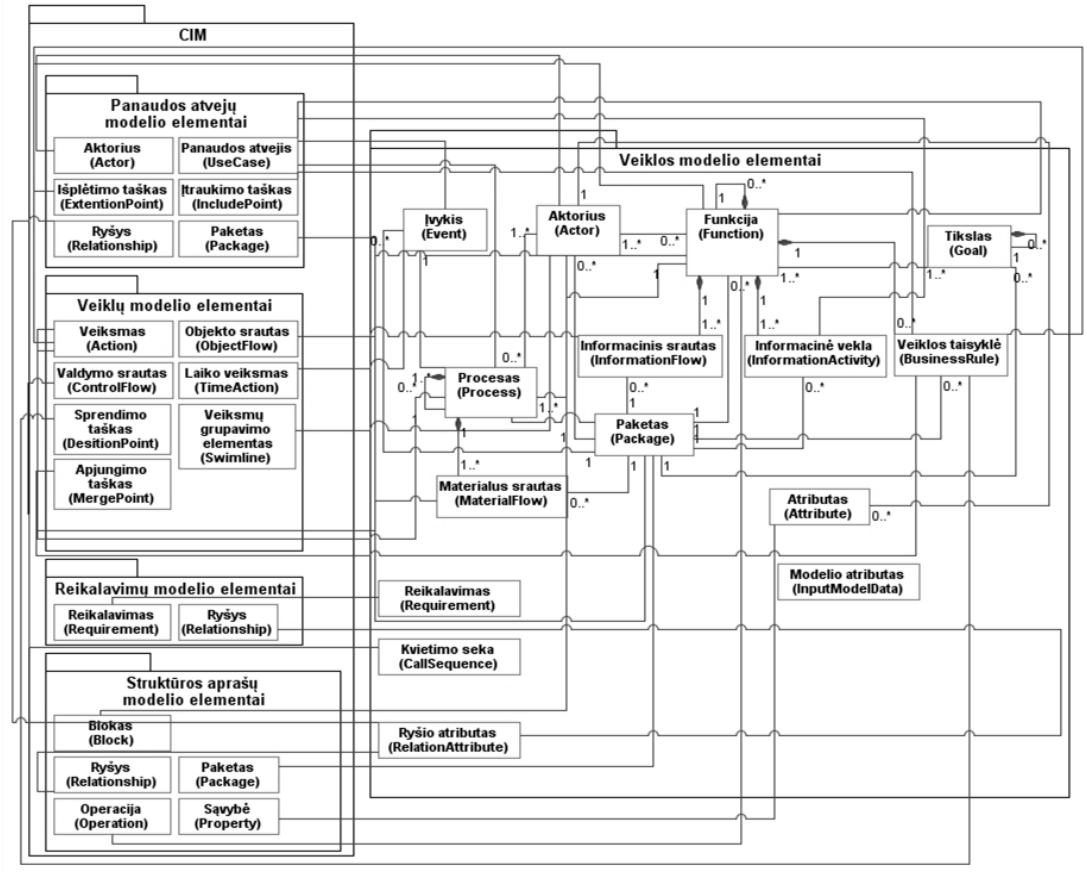


Pav. 16. Pagrindiniai UML paketų modelio elementai

Žiniomis grindžiamo MDA metodo PSM modelis yra sudarytas iš analogiškų UML modelių tipų kaip ir PIM, pastaruosius anotavus konkrečiai platformai būdingomis žymėmis, duomenų ir esybių tipais.

2.3.3.1 Metamodelių sąsajos: CIM ir VM

Veiklos modelis yra sudarytas iš aibės elementų, kurių pagrindiniai yra *procesas*, *funkcija* ir *aktorius*. Šie elementai gali turėti vidinę struktūrą (pvz. procesą sudaro vidiniai procesai, funkcija yra sudaryta iš informacinių veiklų). Diagramoje žemiau yra pavaizduotas žiniomis grindžiamo MDA metodo veiklos metamodelis bei CIM modelio vidiniai SysML modeliai ir pagrindiniai jų elementai bei sąsajos.



Pav. 17. Žiniomis grindžiamo MDA CIM ir veiklos metamodelio elementų sąsajos

Panaudos atvejų modelyje pagrindiniai elementai yra: aktorius (angl. *„Actor“*),panaudos atvejis (angl. *„Use Case“*), išplėtimo taškas (angl. *„Extension Point“*), įtraukimo taškas (angl. *„Include Point“),* paketas (angl. *„Package“*) ir ryšys (angl. *„Relationship“*). Ryšiai tarp elementų gali būti:

generalizacija (angl. *„Generalization“*), asociacija (angl. *„Association“*) ir agregacija (angl. *„Aggregation“*). Veiklų modelyje pagrindiniai elementai yra: veiksmas (angl. *„Action“*), valdymo srautas (angl. *„Control Flow“)*, sprendimo taškas (angl. *„Decision Point“*), apjungimo taškas (angl. *„Merge Point“*), objekto srautas (angl. *„Object Flow“*), priėmimo veiksmas (angl. *„Accept Event Action“*), veiksmų grupavimo elementas (angl. *„Swimline“*). Reikalavimų modelio pagrindiniai elementai yra: reikalavimas (angl. *„Requirement“*) ir ryšys (angl. *„Relationship“*). Struktūros aprašų modelio pagrindiniai elementai yra: blokas (angl. *„Block“*), sąsaja (angl. *„Interface“*), operacija (angl. *„Operation“*), savybė (angl. *„Property“*) ir ryšys (angl. *„Relationship“*). Detalus sąsajų žemėlapis yra pateiktas žemiau lentelėje. Šioje lentelėje kiekvienam veiklos modelio elementui yra priskirtas jį reprezentuojantis CIM modelio elementas.

Lentelė 3. Sąsajų žemėlapis tarp CIM ir VMM elementų

CIM modelio vidiniai modeliai	**Modelio elementas**	**Veiklos modelio elementai**	**Sąsajos aprašymas**
Reikalavimų modelis (Requirements modelis)	Reikalavimas (Requirement)	Reikalavimas (Requirement)	SysML reikalavimų modelio *reikalavimo* objektas yra atvaizduojami į veiklos modelio reikalavimą. Šiame objekte yra saugomi nefunkcinių reikalavimų aprašai. Papildoma anotacija yra reikalinga, kuri nusako su kokiu objektu yra susietas reikalavimas.
	Ryšys (Relationship)	Ryšio atributas (RelationAttribute)	SysML reikalavimų modelio ryšiai tarp reikalavimų yra atvaizduojami į ryšio atributų elementus.
Panaudos atvejų modelis (Use Case model)	Aktorius (Actor)	Aktorius (Actor)	SysML panaudos atvejų modelio aktoriaus objektai yra atvaizduojami į veiklos modelio aktoriaus objektus, pastariesiems priskiriant panaudos atvejų aktorių pavadinimus.
	Panaudos atvejis (UseCase)	Funkcija (Function), procesas (Process)	SysML panaudos atvejų objektai veiklos modelyje yra atvaizduojami į funkcijas arba procesus. Panaudos atvejų modelyje turi būti atlikta papildoma anotacija naudojantis žymų objektą, kuris nurodo, kokio tipo veikla yra konkretus panaudos atvejis.

	Įtraukimo taškas (IncludePoint)	Veiklos taisyklė (BusinesRule)	SysML panaudos atvejų modelio įtraukimo taško elementas yra atvaizduojamas į veiklos modelio veiklos taisyklės elementą. Veiklos taisyklė aprašo kviečiantį bei priimantį objektus, bei kvietimo sąlygą.
	Išplėtimo taškas (ExtentionPoint)	Veiklos taisyklė (BusinesRule)	SysML panaudos atvejų modelio išplėtimo taško elementas yra atvaizduojamas į veiklos modelio veiklos taisyklės elementą. Veiklos taisyklė aprašo kviečiantį bei priimantį objektus bei kvietimo sąlygą.
	Paketas (Package)	Paketas (Package)	SysML panaudos atvejų modelio paketas yra atvaizduojamas į veiklos modelio paketą. Panaudos atvejų modelio paketai grupuoja aktorius ir atitinkamai jų panaudos atvejus.
	Ryšys (Relationship)	Ryšio atributas (RelationAttribute)	SysML panaudos atvejų modelio ryšiai tarp aktorių ir panaudos atvejų yra atvaizduojami kaip konkretaus aktoriaus funkcijos ar procesai. Ryšiai tarp aktorių atvaizduojami į ryšio atributų elementus.
Veiklų modelis (Activity model)	Veiksmas (Action)	Funkcija (Function), procesas (Process)	SysML veiklų modelio veiksmo objektas gali būti atvaizduojamas į veiklos modelio procesą arba funkciją. Papildomas anotavimas yra būtinas, nes nusako veiksmo tipą.
	Valdymo srautas (ControlFlow)	Kvietimo seka (CallSequence)	SysML veiklų modelio elementas valdymo srautas yra atvaizduojamas į veiklos modelio elementą kvietimo seką. Šis elementas nusako veiklų iškvietimo eiliškumą.
	Objekto srautas (ObjectFlow)	Informacinis srautas (MaterialFlow)	SysML veiklų modelio elementas objekto srautas yra atvaizduojamas į veiklos modelio objektą materialų srautą (atitinka duomenis) arba į informacinį srautą (atitinka informaciją). Srauto tipas nustatomas pagal siunčiančio ir priimančio objekto tipus.
	Laiko veiksmas (TimeAction)	Įvykis (Event)	SysML veiklų modelio elementas laiko veiksmas yra atvaizduojamas į veiklos modelio elementą įvykis.
	Sprendimo taškas (DesitionPoint)	Veiklos taisyklė (BusinesRule)	SysML veiklų modelio sprendimo taško elementas yra atvaizduojamas į veiklos modelio veiklos taisyklės elementą. Veiklos taisyklė aprašo įėjimo bei išėjimo veiksmus bei pasirinkimo sąlygą.

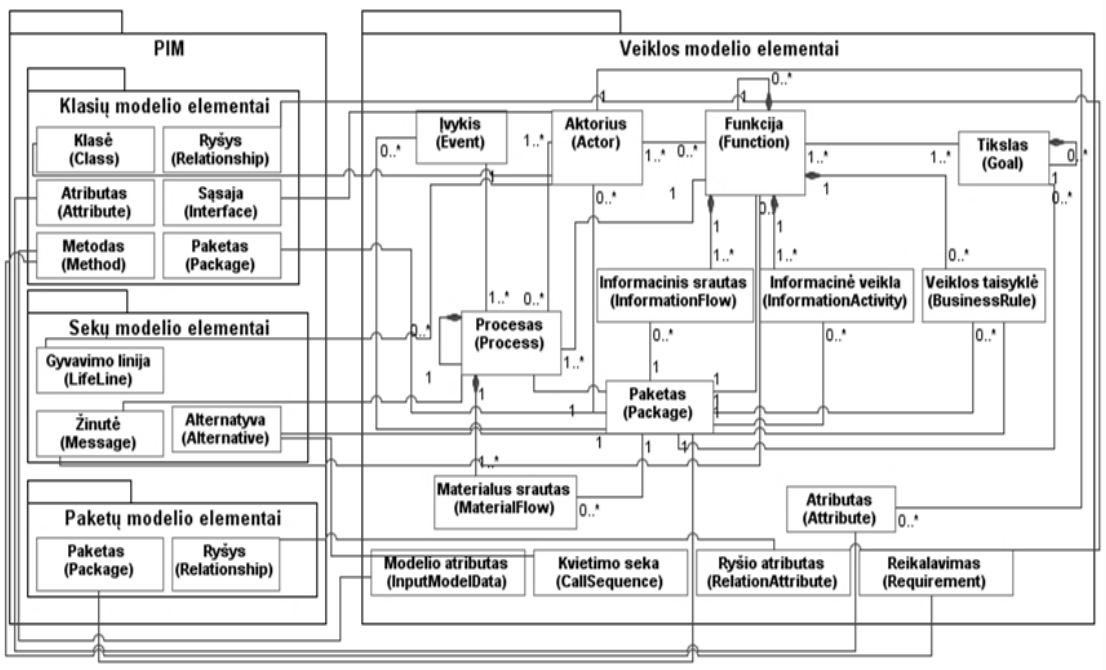
	Apjungimo taškas (MergePoint)	Veiklos taisyklė (BusinesRule)	SysML veiklų modelio sprendimo taško elementas yra atvaizduojamas į veiklos modelio veiklos taisyklės elementą. Veiklos taisyklė aprašo įėjimo bei išėjimo veiksmus.
	Veiksmų grupavimo elementas (Swimline)	Aktorius (Actor)	SysML veiklų modelio veiksmų grupavimo objektai yra atvaizduojami į veiklos modelio aktoriaus objektus.
Struktūros aprašų modelis (Block Definition model)	Blokas (Block)	Aktorius (Actor)	SysML struktūros aprašų bloko objektai yra atvaizduojami į veiklos modelio aktoriaus objektus. Bloko metodai yra atvaizduojami į funkcijas arba procesus, savybės į aktoriaus atributus, o ryšiai į ryšio atributus.
	Savybė (Property)	Atributas (Attribute)	SysML struktūros aprašų modelio bloko objekto savybės yra atvaizduojamos į veiklos modelio aktoriaus atributus.
	Operacija (Operation)	Funkcija (Function), procesas (Process)	SysML struktūros aprašų modelio bloko objekto metodai yra atvaizduojami į veiklos modelio aktoriaus funkcijas arba procesus. Papildoma anotacija yra reikalinga, kuri nusako metodo tipą. Metodo parametrai yra atvaizduojami į materialius arba informacinius rautus, priklausomai nuo metodo tipo.
	Ryšys (Relationship)	Ryšio atributas (RelationAttribute)	SysML struktūros aprašų modelio ryšiai tarp blokų yra atvaizduojami į ryšio atributų elementus.

Iš lentelėje pateiktos informacijos galime matyti, kad dalis elementų gali turėti keletą atvaizdavimo variantų. Tokiu atveju elementas yra anotuojamas papildoma žyme (*„Note“* elementu), kuri nusako, koks veiklos modelio elementas turi būti sukurtas.

2.3.3.2 Metamodelių sąsajos: PIM ir VM

Sąsajos tarp veiklos modelio elementų ir PIM modelio yra reikalingos siekiant automatiniu ar pusiau automatiniu būdu generuoti PIM modelį iš veiklos modelio. Žiniomis grindžiamo MDA metodo PIM modelis yra sudarytas iš trijų UML modelių: klasių (angl. *„Class“)*, sekų (angl. *„Sequence“)* ir paketų (angl. *„Package“*). Klasių ir paketų modeliai yra naudojami aprašyti sistemos statinę informaciją bei suskirstymą į logines dalis, sekų modeliai yra skirti atvaizduoti sistemos elgsenos informaciją. Pagrindiniai sekų modelio elementai yra gyvavimo linija (angl. *„Life line“*), žinutė (angl.

„Message“) ir alternatyva (angl. *„Alternative“*). Žinutės gali būti asinchroninės ir sinchroninė, objektai gali turėti metodus ir duomenų laukus. Klasių modelio pagrindiniai elementai yra klasė (angl. *„Class“)* ir ryšys (angl. *„Relationship“).* Šių modelių elementai yra pavaizduoti paveiksle ir lentelėje žemiau.



Pav. 18. Žiniomis grindžiamo MDA PIM ir veiklos metamodelio elementų sąsajos

Klasė gali turėti metodus (angl. *„Method“*) ir savybes (angl. *„Attribute“*). Klasės gali būti susiejamos ryšiai tokiais kaip asociacija (angl. *„Association“)*, paveldėjimas(angl. *„Generalization“*), agregacija (angl. *„Aggregation“*), kompozicija (angl. *„Composition“*). Paketų modelio pagrindinis elementas yra paketas (angl. *„Package“*), kuris atvaizduoja tam tikrą loginę sistemos dalį. Paketai yra naudojami kaip susijusių klasių grupavimo elementai. Detalus VMM į PIM sąsajų žemėlapis yra pateiktas žemiau lentelėje. Šioje lentelėje kiekvienam veiklos modelio elementui yra priskirtas jį reprezentuojantis PIM modelio elementas.

Lentelė 4. Sąsajų žemėlapis tarp PIM ir VMM elementų

PIM modelio vidiniai modeliai	Modelio elementas	Veiklos modelio elementai	Sąsajos aprašymas
Klasių modelis (Class model)	Klasė (Class)	Aktorius (Actor)	Veiklos modelio aktoriaus objektai yra atvaizduojami į UML klasių modelio klasės objektus. Aktoriaus atributai yra atvaizduojami į klasės savybes, o aktoriaus atliekami procesai ir funkcijos gali būti atvaizduojami (priklausomai nuo detalumo lygmens) į klasės metodus.
	Atributas (Attribute)	Atributas (Attribute)	Veiklos modelio aktoriaus atributai yra atvaizduojami į klasių modelio klasės atributus.
	Metodas (Method)	Funkcija (Function), procesas (Process)	Veiklos modelio funkcijos ir procesai yra atvaizduojami į klasės metodus. Informaciniai ir materialūs srautai yra atvaizduojami į funkcijos įvesties ir išvesties parametrus.
	Ryšys (Relationship)	Ryšio atributas (RelationAttribute)	Veiklos modelio ryšio atributo elementas yra atvaizduojamas ryšio elementu klasių modelyje. Ryšio atributas nurodo kokie du elementai yra susieti bei koks ryšio tipas.
	Sąsaja (Interface)	Aktorius (Actor)	Veiklos modelio aktorius gali būti atvaizduojamas, kaip sąsajos elementas klasių modelyje. Tokio tipo aktorius turi būti anotuojamas papildoma žyme nurodančia šio tipo konvertavimą. Sąsajos metodai ir atributai yra sukuriami iš aktoriaus atributų ir funkcijų bei procesų.
	Paketas (Package)	Paketas (Package)	Veiklos modelio paketas yra atvaizduojamas į klasių modelio paketą. Veiklos modelio aktoriai, kurie yra grupuojami veiklos modelio pakete, yra atvaizduojami kaip klasių modelio klasės.

Sekų modelis (Sequence model)	Gyvavimo linija (LifeLine)	Aktorius (Actor)	Veiklos modelio aktoriaus objektai yra atvaizduojami į UML sekų modelio *gyvavimo linijos* elementus. Aktoriaus atliekami procesai ir funkcijos yra atvaizduojami į sekų modelio žinučių elementus.
	Žinutė (Message)	Funkcija (Function), procesas (Process)	Veiklos modelio funkcijos ir procesai yra atvaizduojami į sekų modelio žinutės elementus. Informaciniai ir materialūs srautai yra atvaizduojami į žinutės parametrus.
	Alternatyva (Alternative)	Kvietimo seka (CallSequence)	Veiklos modelio elementas „kvietimo seka“ gali būti atvaizduojamas į sekų modelio alternatyvos elementą. Šis elementas nusako metodų iškvietimo eiliškumą.
Paketų modelis (Package model)	Paketas (Package)	Paketas (Package)	Veiklos modelio paketas yra atvaizduojamas į klasių modelio paketą. Veiklos modelio aktoriai, kurie yra grupuojami veiklos modelio pakete, yra atvaizduojami kaip klasių modelio klasės.
	Ryšys (Relationship)	Ryšio atributas (RelationAttribute)	Veiklos modelio ryšio atributo elementas yra atvaizduojamas ryšio elementu klasių modelyje. Ryšio atributas nurodo kokie du elementai yra susieti bei koks ryšio tipas.

Iš lentelėje pateiktos informacijos galime matyti, kad dalis elementų gali turėti keletą atvaizdavimo variantų. Tokiu atveju papildomai apibrėžiamos sąlygos, kurios nusako kokias atvejais konkretus veiklos modelio elementas yra atvaizduojamas į PIM modelio elementą.

2.3.3 Transformacijos

Žiniomis grindžiamas MDA metodas apibrėžia keturias modelių transformacijas CIM į VM, VM į PIM, PIM į PSM ir PSM į programinį kodą. PIM į PSM ir PSM į programinį kodą yra standartinės MDA transformacijos, o CIM į VM ir VM į PIM yra žiniomis grindžiamo MDA metodo išskirtinės transformacijos. Tam kad būtų įmanoma atlikti minėtas transformacijas yra

sudaryti sąsajų žemėlapiai, kurie nusako kaip konkretus vieno modelio elementas (pvz. aktorius) turi būti atvaizduojamas kitame modelyje. Žiniomis grindžiamo MDA metodo specifines transformacijas bei sąsajų žemėlapius detalizuosime.

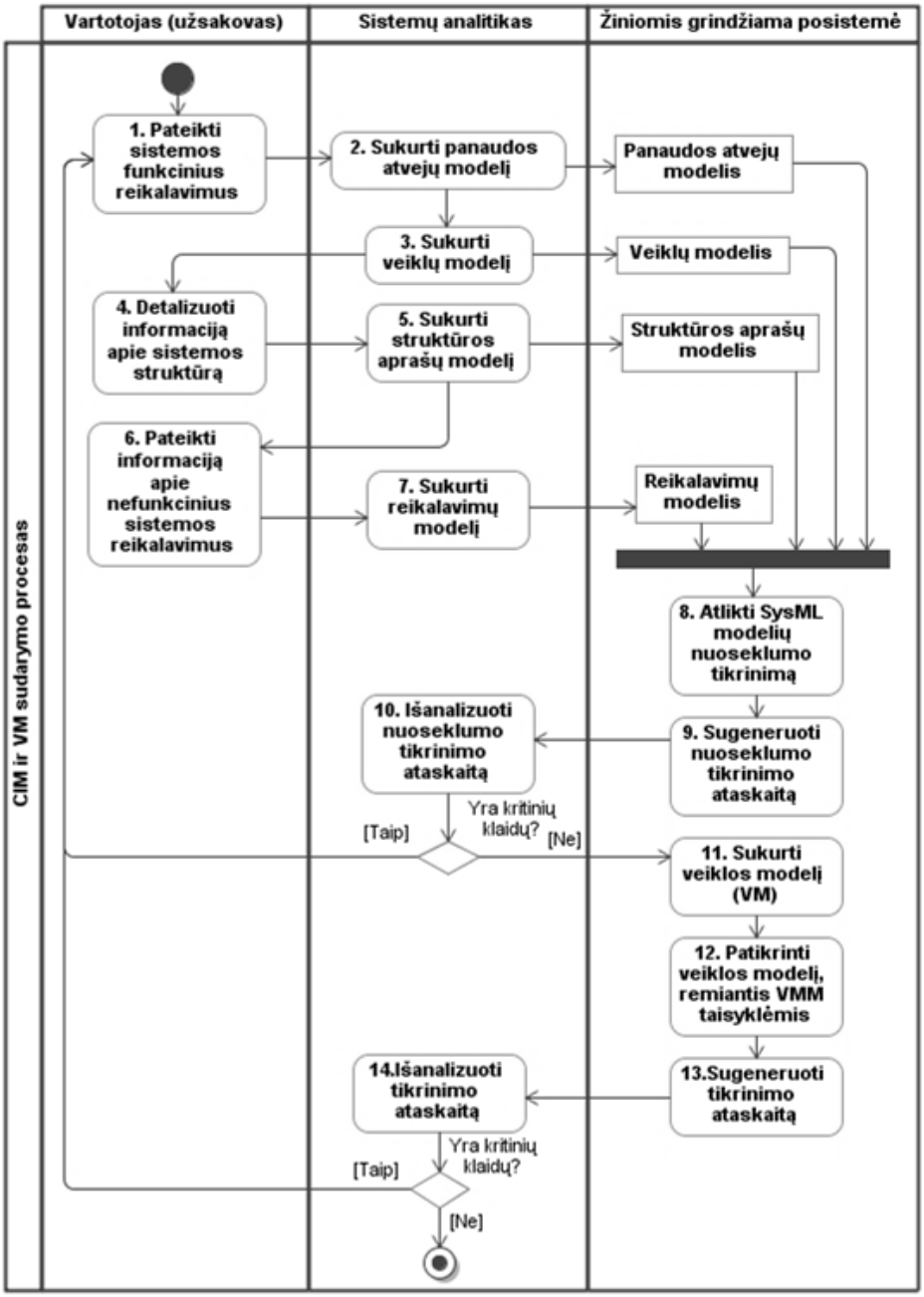
2.3.4 Procesas

Žiniomis grindžiamas MDA metodas apibrėžia penkių tipų aktorius: vartotojas, sistemų analitikas, žiniomis grindžiama posistemė (angl. *„Knowledge Based Subsystem" (KBS)*), MDA modelių transformavimo įrankis bei architektas (programuotojas). Pradinis žiniomis grindžiamo MDA metodo etapas – CIM modelio sudarymas dalykinei sričiai. Šį žingsnį atlieka sistemos analitikas kartu su vartotoju, naudodamas įvairias dalykinės srities žinių surinkimo technikas [12], [78]. Atlikus vidinių CIM modelių tarpusavio suderinamumo patikrinimą ir ištaisius esamas klaidas, šių modelių pagrindu yra kuriamas veiklos modelis. Veiklos modelis yra generuojamas automatiniu būdu, naudojant egzistuojančius metamodelių sąsajų žemėlapius bei transformavimo algoritmus. Kitas etapas yra patikrinti sukurtą veiklos modelį veiklos metamodelio taisyklių atžvilgiu t.y. remiantis EVC apibrėžimu. Šio patikrinimo metu yra sugeneruojama klaidų ataskaita, kuri pateikiama sistemų analitikui. Jeigu ataskaitoje pateikiami veiklos modelio trūkumai yra esminiai, veiklos modelis turi būti atnaujintas t.y. SysML modeliai, kurių pagrindu buvo sukurtas veiklos modelis turi būti papildyti trūkstama informacija arba turi būti ištaisytos juose egzistuojančios klaidos. Ciklas CIM => VM=> atnaujintas CIM => VM gali turėti keletą iteracijų. Esant teisingam veiklos modeliui iš jo elementų yra generuojamas PIM modelis. PIM modelis yra generuojamas konkrečiai pasirinktai sričiai arba visiems veiklos modelio elementams. Sukūrus PIM modelį bei pasirinkus konkrečią programinės įrangos realizavimo platformą ir naudojant MDA transformavimo įrankius yra kuriamas PSM modelis. Galutinis proceso etapas yra programinio kodo generavimas iš PSM modelio. Kituose skyriuose detaliau aptarsime visus minėtus etapus.

2.3.4.1 CIM modelio sudarymo procesas

CIM modelio sudarymas yra pirmas žiniomis grindžiamos MDA metodo etapas. Šiame etape yra siekiama surinkti reikalavimus bei juos patikrinti formalių kriterijų atžvilgiu. Etape dalyvauja trijų tipų aktoriai: vartotojas, sistemų analitikas bei veiklos žinių posistemė (ją realizuojantis KB-MDA įrankis). Detali proceso schema yra pavaizduota 19 paveiksle. Pirmas žingsnis yra informacijos apie sistemos reikalavimus suteikimas (žingsnis *"1. Pateikti sistemos funkcinius reikalavimus"*). Šio žingsnio metu vartotojas nusako pageidaujamą sistemos elgseną bei pirminius veiklos scenarijus t.y. pateikia funkcinius reikalavimus. Sistemų analitikas naudodamas suteiktą informaciją sukuria panaudos atvejų (*„Use Case"*) modelius. Panaudos atvejų modeliai yra naudojami kaip veiklos scenarijus aprašantis duomenų šaltinis bei kaip pagrindas veiklų (*„Activity")* modelių kūrimui. Sudarius dalykinę sritį apibūdinančius panaudos atvejų ir veiklų modelius, CIM gali būti traktuojamas, kaip užpildytas modeliuojamos sistemos elgsenos informacija. Ketvirto žingsnio (*"4. Detalizuoti informaciją apie sistemos struktūrą")* metu vartotojas detalizuoja sistemos architektūrinius elementus. Sistemų analitikas naudodamasis šia informacija sukuria struktūros aprašų (*„Block Definition")* modelius. Paskutiniame reikalavimų surinkimo etape vartotojas pateikia informaciją apie sistemos nefunkcinius reikalavimus tokius kaip operacijų maksimali vykdymo trukmė, komunikavimo su kitomis sistemomis standartai ir pan. Šie reikalavimai specifikuojami naudojant reikalavimų (*„Requirements")* modelius. Nefunkciniai reikalavimai yra susiejami su funkcinių reikalavimų objektais pvz. panaudos atvejais, aktoriais, veiksmais. Sukūrus visus CIM modelius yra vykdomas modelių tikrinimo etapas (*"8. Atlikti SysML modelių nuoseklumo tikrinimą "*). Šiame etape atliekamas modelių nuoseklumo tikrinimas t.y. modeliai yra tikrinami vienas kito atžvilgiu naudojantis nustatytais kriterijais (pvz. ar visi panaudos atvejų modeliuose specifikuoti aktoriai yra specifikuoti ir veiklų modeliuose, ar nėra pasikartojančių ar trūkstamų elementų ir kt.). Kitas žingsnis tikrinimo ataskaitos generavimas (*"10. Išanalizuoti nuoseklumo tikrinimo ataskaitą"*).

Sistemų analitikas išanalizuoja sugeneruotą tikrinimo ataskaitą bei nusprendžia, kuriuose modeliuose ir kokius pakeitimus būtina atlikti. Jeigu sugeneruotoje ataskaitoje klaidų nėra, sekantis žingsni yra veiklos modelio duomenų bazės užpildymas informacija, kuri buvo surinka CIM. Šiame etape, pasinaudojant sąsajų žemėlapiais, SysML modelių elementai yra transformuojami į veiklos modelio elementus.



Pav. 19. Pagrindiniai CIM modelio generavimo žingsniai

Kitas etapas yra veiklos modelio tikrinimas veiklos metamodelio taisyklių atžvilgiu ("*12. Patikrinti veiklos modelį, remiantis VMM taisyklėmis*"). Jeigu

tikrinimo rezultatas yra teigiamas, toliau yra atliekami kiti žiniomis grindžiamo MDA proceso etapai. Priešingu atveju veiklos modelis turi būti atnaujinamas per CIM pakeitimus. Tiesiogiai redaguoti veiklos modelį yra neįmanoma, tokiu būdu siekiama užtikrinti žiniomis grindžiamo MDA metodo proceso nuoseklumą ir vientisumą. Veiklos modelio užpildymo duomenimis ir tikrinimo procesas yra iteracinis ir turi būti kartojamas tol, kol veiklos modelyje yra ištaisomos klaidos. SysML modeliai turi hierarchines struktūras, todėl jų nuskaitymui ir konvertavimui į veiklos modelio elementus yra naudojami rekursinius algoritmus. Pagrindinis principas yra nuoseklus iteravimas per konkretaus modelio elementus „platyn ir gilyn“. Pasirinkus pirmąjį elementą yra sukuriamas jį atitinkantis veiklos modelio elementas, tuomet patikrinama ar šis elementas turi susijusių elementų (angl. *“child elements”*), jeigu turi, tuomet iteruojama per šių elementų sąrašą tokiu pačiu principu t.y. kiekvienas elementas yra traktuojamas kaip potencialus “tėvo” tipo elementas (angl. *“parent element”*). Šis principas yra naudojamas nuskaitant SysML modelius, skirtus veiklos modelio generavimui (panaudos atvejų, veiklų, struktūros aprašų, reikalavimų). Detalizuoti CIM lygmens žingsniai yra aprašyti lentelėje žemiau.

Lentelė 5. Žiniomis grindžiamo MDA metodo CIM modelio sudarymo žingsniai

#	**Žingsnio pavadinimas**	**Vykdytojas**	**Etapo aprašymas**
1	Pateikti sistemos funkcinius reikalavimus	Vartotojas	Vartotojas bendrauja su sistemų analitiku ir perteikia jam funkcinius sistemos reikalavimus t.y. sistemos elgsenos informaciją.
2	Sukurti panaudos atvejų modelį	Sistemų analitikas	Remiantis iš vartotojo gauta informacija sistemų analitikas sukuria panaudos atvejų diagramą.
3	Sukurti veiklos modelį	Sistemų analitikas	Remiantis iš vartotojo gauta informacija sistemų analitikas sukuria veiklos diagramą.
4	Detalizuoti informaciją apie sistemos struktūrą	Vartotojas	Vartotojas bendrauja su sistemų analitiku ir detalizuoja sistemos elementų struktūros informaciją.
5	Sukurti struktūros aprašų modelį	Sistemų analitikas	Remiantis iš vartotojo gauta informacija sistemų analitikas sukuria struktūros aprašų diagramą.
6	Pateikti informaciją apie nefunkcinius sistemos reikalavimus	Vartotojas	Vartotojas bendrauja su sistemų analitiku ir perteikia jam nefunkcinius sistemos reikalavimus.

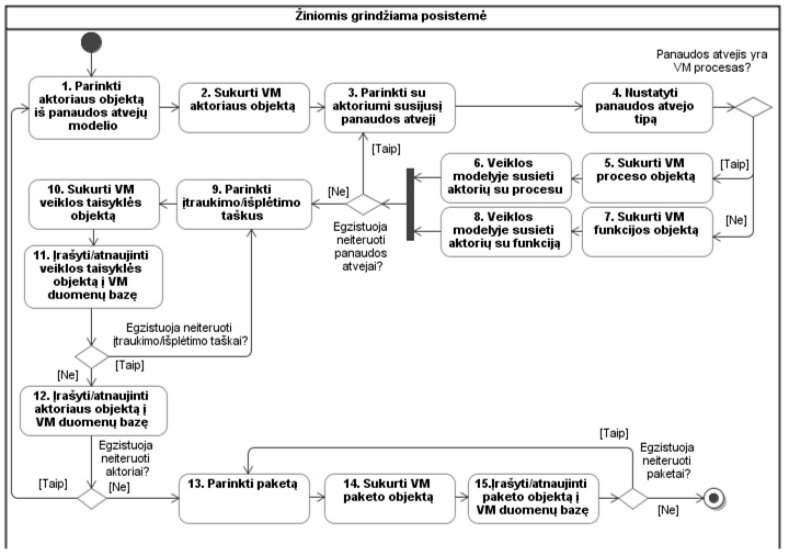
7	Sukurti reikalavimų modelį	Sistemų analitikas	Remiantis iš vartotojo gauta informacija sistemų analitikas sukuria reikalavimų diagramą.
8	Atlikti SysML modelių nuoseklumo tikrinimą	Žiniomis grindžiama posistemė	SysML modelių nuoseklumo tikrinimas yra atliekamas naudojantis sistemų analitiko nustatytomis taisyklėmis. Šio etapo metu yra nustatomi trūkstami arba pertekliniai SysML modelių elementai.
9	Sugeneruoti nuoseklumo tikrinimo ataskaitą	Žiniomis grindžiama posistemė	Žiniomis grindžiama posistemė sugeneruoja modelių nuoseklumo tikrinimo rezultatų ataskaitą. Ataskaita įtraukia perspėjimus (angl. *„Warnings“*) ir klaidas (angl. „Errors“).
10	Išanalizuoti nuoseklumo tikrinimo ataskaitą	Sistemų analitikas	Sistemų analitikas išanalizuoja nuoseklumo ataskaitą ir nusprendžia, kokie pakeitimai turi būti padaryti CIM modelyje.
11	Sukurti veiklos modelį	Žiniomis grindžiama posistemė	Remiantis CIM => VM sąsajų žemėlapiais, CIM esanti informacija yra transformuojama į veiklos modelyje naudojamą notaciją.
12	Patikrinti veiklos modelį, remiantis VMM taisyklėmis	Žiniomis grindžiama posistemė	Sukurtas veiklos modelis yra patikrinamas veiklos metamodelyje apibrėžtų taisyklių atžvilgiu.
13	Sugeneruoti tikrinimo ataskaitą	Žiniomis grindžiama posistemė	Žiniomis grindžiama posistemė sugeneruoja veiklos modelio tikrinimo rezultatų ataskaitą. Ataskaita įtraukia perspėjimus (angl. *„Warnings“*) ir klaidas (angl. „*Errors*“).
14	Išanalizuoti tikrinimo ataskaitą	Sistemų analitikas	Sistemų analitikas išanalizuoja ataskaitą ir nusprendžia ar galima pradėti kitus žiniomis grindžiamo MDA metodo etapus ar būtina kartoti CIM modelio etapus.
15	Pradėti kitus žiniomis grindžiamo MDA metodo etapus	Žiniomis grindžiama posistemė	Pagal nustatytus parametrus atliekamas PIM modelio generavimas iš veiklos modelio. Generavimą atlieka žiniomis grindžiama posistemė.

Kadangi veiklos modelio pildymas naujais elementais (CIM transformavimas į VM) yra linijinis procesas, SysML modelių nuskaitymas ir veiklos modelių elementų kūrimas yra atliekamas nustatytu eiliškumu [11]. Panaudos atvejų modeliai yra nuskaitomi pirmieji. Šie modeliai yra naudojami specifikuoti aukšto abstrakcijos lygmens funkcinius reikalavimus ir veiklas. Veiklų modeliai yra nuskaitomi antrieji, kadangi jie detalizuoja panaudos atvejų modeliais surinktą informaciją. Abu minėti modelių tipai apibrėžia dalykinės srities dinaminę informaciją. Trečiajame etape yra nuskaitomi struktūros aprašų modeliai, kurie yra skirti aprašyti sistemos struktūrą, jos elementus, sandarą ir ryšius. Paskutinis žingsnis yra reikalavimų modelių nuskaitymas. Šie

modeliai apibrėžia sistemos nefunkcinius reikalavimus. Nuskaičius visus SysML modelius yra atliekamas modelių nuoseklumo tikrinimas. Modelių nuoseklumo tikrinimo procesas turi užtikrinti, jog skirtinguose modeliuose surinkta informacija yra neprieštaringa ir pilna (kitų modelių atžvilgiu) t.y. visi modeliai turi būti tarpusavyje suderinami. Gali būti naudojami įvairūs modelių tarpusavio suderinamumo kriterijai [9], [31], [51], [64] (pvz. ar kiekvienas panaudos atvejis turi jį detalizuojantį veiklų modelį). Žiniomis grindžiamo MDA metodo dalykinės programos prototipe yra tikrinami panaudos atvejų modelio aktorių ryšiai su veiklų modelio grupavimo elementais. Detalūs CIM modelių nuskaitymo algoritmai yra pateikti kituose skyriuose.

2.3.4.1.1 VM elementų generavimas iš SysML panaudos atvejų modelio

Panaudos atvejų modeliai yra skirti surinkti ir atvaizduoti pagrindinius sistemos elgsenos scenarijus. Esminiai šių modelių elementai yra: aktorius, panaudos atvejis, paketas, išplėtimo ir įtraukimo ryšiai. Visi minėti elementai yra naudojami veiklos modelio elementų kūrimui. Panaudos atvejų modelio nuskaitymas pradedamas iteruojant per visus aktoriaus tipo elementus („*[UC]Actor“*) Šio tipo elementų informacija yra naudojama kuriant veiklos modelio aktoriaus tipo objektus („*[VM]Actor“*). Bazinis panaudos atvejų modelių nuskaitymo algoritmas yra pavaizduotas paveiksle 20.



Pav. 20. Pagrindiniai veiklos modelio elementų generavimo iš panaudos atvejų modelio žingsniai

Kiekvienas *[UC]Actor* elementas yra susijęs su vienu ar daugiau panaudos atvejų elementų („*[UC]Use Case“*), kuris atitinkamai gali turėti vieną ar kelis išplėtimo („*[UC]Include Point“*) ar įtraukimo („*[UC]Extention Point“*) elementus. Panaudos atvejų elementai veiklos modelyje yra atvaizduojami kaip funkcijos (*„[VM]Function“*) arba kaip procesai (*„[VM]Process“*), priklausomai nuo veiklos tipo. *[UCM]Include Point* ir *[UCM]Extention Point* elementai veiklos modelyje yra atvaizduojami kaip veiklos taisyklių („*[VM]Business Rule“*) elementai. Lentelėje žemiau yra aprašyti algoritmo įvesties elementai (panaudos atvejų modelio elementai) ir išvesties elementai (veiklos modelio elementai).

Lentelė 6. SysML panaudos atvejų ir veiklos modelio elementų sąsajos

SysML panaudos atvejų modelio elementas	**Veikos modelio elementas**	**Sąsajų aprašymas**
Aktorius *([UC]Actor)*	Aktorius (*[VM]Actor*)	Veiklos modelyje aktoriaus objektas yra procesų ir funkcijų vykdytojas. Panaudos atvejų modeliuose aktorius vykdo panaudos atvejus. Iš panaudos atvejų *[UC]Actor* objektų yra sukuriami *[VM]Actor* objektai, kurie po šios transformacijos turi tik bazinę informaciją (pavadinimą ir ryšius).
Panaudos atvejis *([UC]Use Case)*	Funkcija, procesas (*[VM]Function, [VM]Process)*	*[UC]Use Case* yra panaudos atvejų elementas, kuris apibūdina aktoriaus atliekamas veiklas t.y. dinaminę sistemos informaciją. Panaudos atvejis yra dekomponuojamas naudojantis Activity diagrama. veiklos modelyje yra atvaizduojamas kaip funkcija arba procesas (veiklos tipas anotuojamas naudojantis *[UC]Note* elementu).
Įtraukimo taškas *([UC]Include Point)*	Veiklos taisyklė (*[VM]Business Rule*)	*[UC]Include Point* elementas apibrėžia veiklos taisykles, kurios turi būti patenkintos, kad konkretus panaudos atvejis galėtų būti įgyvendintas. Šis elementas veiklos modelyje atvaizduojamas kaip *[VM]Business Rule* elementas.
Išplėtimo taškas *([UC] Extension point)*		*[UC]Extension Point* elementas apibrėžia veiklos taisykles, turi būti patenkintos, kad pagrindinį panaudos atvejį išplečiantis panaudos atvejis, galėtų būti įgyvendintas. Šis elementas veiklos modelyje atvaizduojamas kaip *[VM]Business Rule* elementas.
Paketas *([UC] Package)*	Paketas *([VM]Package)*	*[UC]Package* yra grupavimo elementas, kuris apjungia panašios paskirties elementus. Veiklos modelyje jis yra atvaizduojamas kaip *[VM]Package* elementas.

Panaudos atvejų modelių nuskaitymas yra pirmas žingsnis veiklos modelio

kūrimo procese. Šio žingsnio pabaigoje turi būti sukurti pagrindiniai veiklos modelio elementai (aktoriai, funkcijos, procesai). Šie elementai dar nebūna pilnai specifikuoti, kadangi dalis informacijos yra išgaunama nuskaitant kitus SysML modelius (veiklų, struktūros aprašų, reikalavimų). Kitas žingsnis yra veiklų modelių nuskaitymas. Šie modeliai apibrėžia sistemos veiklos procesus bei jų eiliškumą, taip pat veiklos taisykles.

2.3.4.1.2 VM elementų generavimas iš SysML veiklų modelio

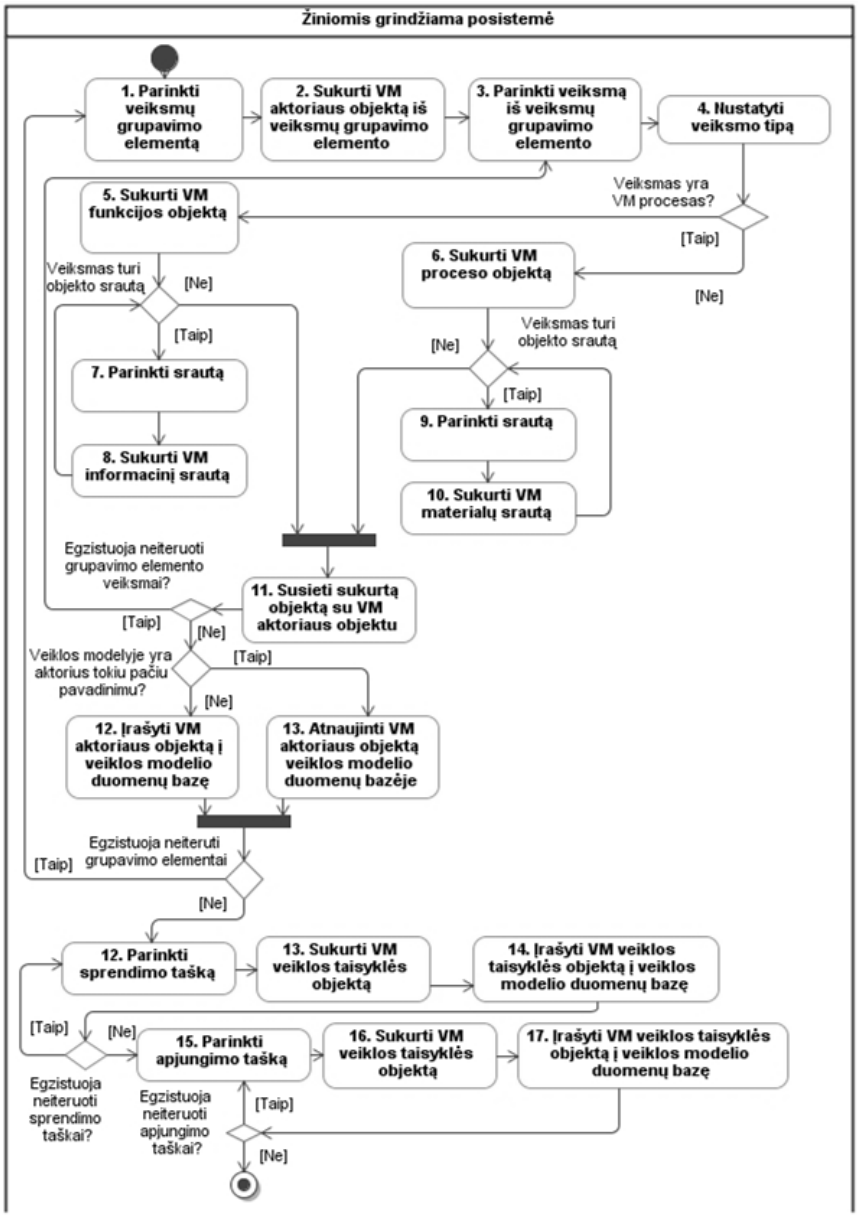
Veiklų modeliai yra skirti surinkti sistemos elgsenos informaciją ir aprašyti bendravimą tarp sistemos elementų. Šie modeliai yra glaudžiai susiję su panaudos atvejų modeliais, kadangi detalizuoja panaudos atvejais surinktą informaciją. Pagrindiniai veiklų modelių elementai yra šie: veiksmų grupavimo elementas, veiksmas, sprendimo taškas ir apjungimo taškas, objekto srautas ir valdymo srautas. Veiklų modelio nuskaitymo algoritmas yra aprašytas 7 lentelėje ir pavaizduotas paveiksle 21.

Lentelė 7. SysML veiklų ir veiklos modelio elementų sąsajos

SysML veiklų modelio elementas	**Veiklos modelio elementas**	**Sąsajų aprašymas**
Veiksmų grupavimo elementas *([AM]Swimlane)*	Aktorius *([VM]Actor)*	*[AM]Swimlane* elementas yra atvaizduojamas į veiklos modelio *[VM]Actor* elementą. Kiekvienas veiksmų grupavimo elementas turi turėti jį atitinkantį aktoriaus elementą. Grupavimo elemente esantys veiksmai yra priskiriami konkrečiam aktoriaus objektui, o duomenų srautai yra traktuojami kaip informaciniai arba materialūs srautai veiklos modelyje.
Veiksmas *([AM]Action)*	Funkcija, procesas, įvykis *([VM]Function, [VM]Process, [VM] Event)*	*[AM]Action* elementas yra gali būti atvaizduojamas į vieną iš šių elementų *[VM]Function, [VM]Process, [VM]Event*. *[AM]Action* gali turėti įeinančius arba išeinančius objekto srautus *[AM]ObjectFlow*. *[AM]Event Action* elementas yra specifinis *[AM]Action* atvejis, kuris apibrėžia veiksmus atliekamus laiko arba išorinės aplinkos
Valdymo srautas *([AM]Control Flow)*		*[AM]Control Flow* elementai apibrėžia veiklų vykdymo eiliškumą. Jie yra atvaizduoja į *[VM]CallFlow* elementus.
Objekto srautas *([AM]Object Flow)*	Materialus, informacinis srautas (*[VM]Materi*	*[AM]Object Flow* elementai aprašo duomenis, kuriais keičiasi *[AM]Action* elementai. [AM]Action elementas gali priimti įeinančius parametrus, juos apdoroti ir perduoti kitam [AM]Action objektui.

	al/Inf Flow)	
Sprendimo/ apjungimo taškas (*[AM]Decision/ Merge Point*)	Veiklos taisyklė (*[VM]Business Rule*)	*[AM]Decision/Merge Point* elementai apibrėžia sprendimų priėmimo taisykles (sąlygas). Šis elementas veiklos modelyje atvaizduojamas kaip *[VM]Business Rule* elementas.

Kiekvienas veiksmų grupavimo elementas *(„[AM]Swimlane“)* reprezentuoja veiklos modelio *[VM]Actor* elementą. Veiksmas *(„[AM]Action“)* gali būti atvaizduojamas į veiklos modelio *[VM]Process* arba *[VM]Function* elementus



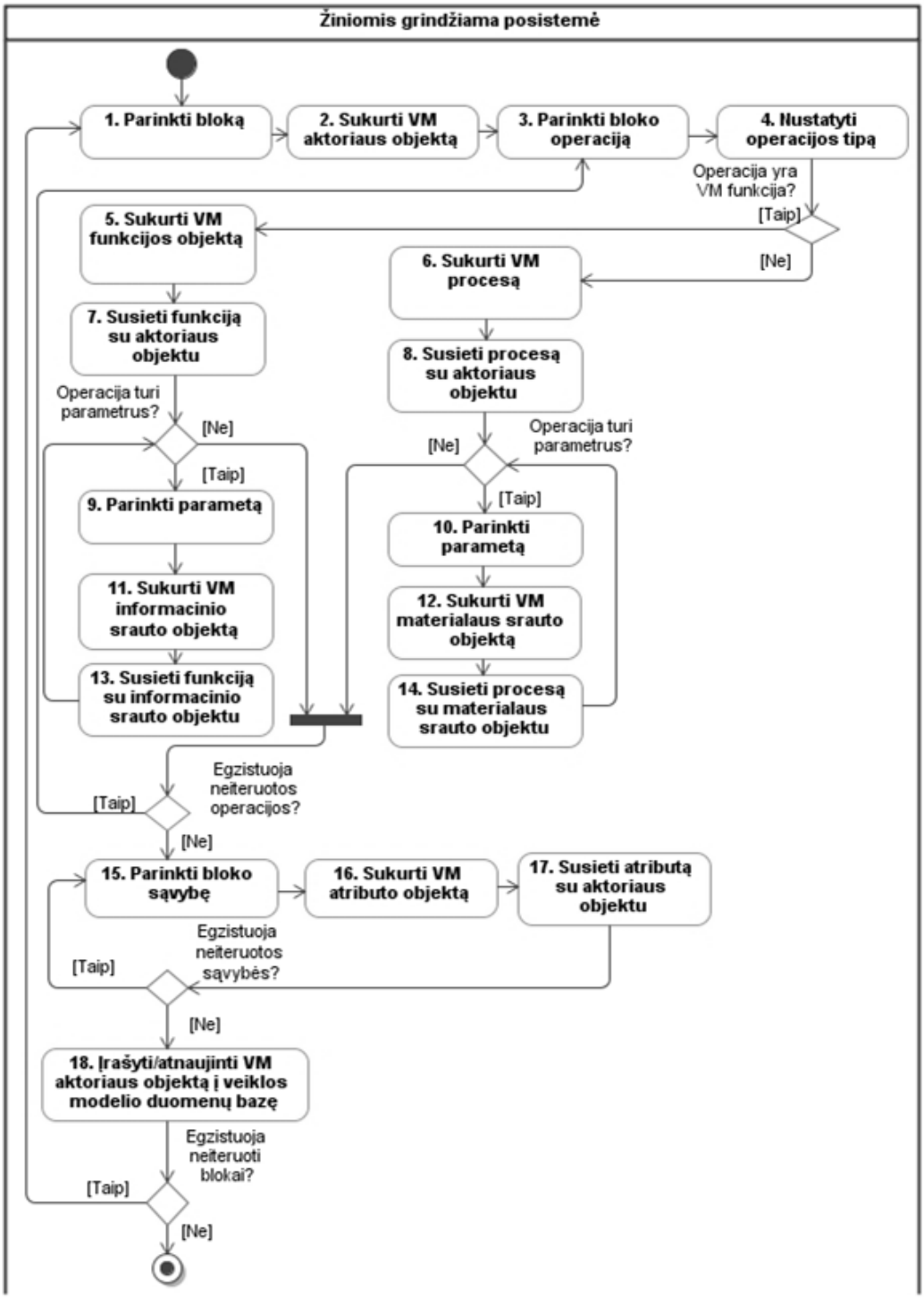
Pav. 21. Pagrindiniai veiklos modelio elementų generavimo iš veiklų modelio žingsniai

Sistemų analitikas turi papildomai anotuoti *[AM]Action* elementą (*[AM]Note elementu*), nurodydamas kuris veiklos modelio elementas turi būti sukurtas. Valdymo srautų („*[AM]Control Flow"*) elementai apibrėžia veiklos modelio veiklų eiliškumą, o objektų srautų elementai („*[AM]Object Flow"*) nurodo kokie objektai yra perduodami tarp veiklų. Atitinkamai *[AM]Object Flow* yra atvaizduojamas į *[VM]InformationaFlow* (funkcijos atveju) arba *[VM]MaterialFlow* (proceso atveju). Veiklos modelio *[VM]Business Rule*s elementai yra kuriami remiantis *[VM]DecisionPoint* ir *[VM] MergePoint* elementais. Lentelėje žemiau yra aprašyti algoritmo įvesties elementai (veiklų modelio elementai) ir išvesties elementai (veiklos modelio elementai).

SysML veiklų modelių nuskaitymas yra antras žingsnis veiklos modelio kūrimo procese. Šio žingsnio pabaigoje egzistuojantys veiklos modelio elementai yra papildomi nauja informacija bei nauji veiklos modelio elementai yra sukuriami (pvz. *[VM]Event*). Kitas žingsnis yra struktūros aprašų modelio nuskaitymas. Šie modeliai apibrėžia sistemos architektūrą ir detalizuoja sistemos elementų sandarą.

2.3.4.1.3 VM elementų generavimas iš SysML struktūros aprašų modelio

Struktūros aprašų modeliai yra skirti surinkti informaciją apie sistemos struktūrą bei jos elementus. Šie modeliai yra nuskaitomi po panaudos atvejų ir veiklų modelių t.y. veiklos modelio elementai, kurie buvo sukurti pirmų dviejų etapų metu, yra papildomi nauja informacija, kuri apima savybes, parametrus, paveldėjimo ryšius ir kt. Pagrindiniai struktūros aprašų modelių elementai yra šie: blokas, operacija, operacijos parametras, apribojimas ir savybė. Struktūros aprašų modelio nuskaitymo algoritmas yra pavaizduotas paveiksle 22 ir aprašytas lentelėje 8.



Pav. 22. Pagrindiniai veiklos modelio elementų generavimo iš struktūros aprašų modelio žingsniai

Lentelė 8. SysML struktūros aprašų ir veiklos modelio elementų sąsajos

SysML struktūros aprašų modelio elementas	**Veiklos modelio elementas**	**Sąsajų aprašymas**
Blokas *([BDM]Block)*	Aktorius *([VM]Actor)*	*[BDM]Block* elementas yra atvaizduojamas į veiklos modelio

		aktoriaus elementą. *[BDM]Block* elemento pagrindinė paskirtis detalizuoti aktoriaus objekto statinius laukus (atributus) bei atliekamų funkcijų ir procesų įvesties ir išvesties parametrus. *[BDM]Block* elementai gali turėti generalizavimo ryšius, todėl yra naudojami išreikšti aktorių abstraktumo lygmenis.
Operacija ([BDM]Operation)	Funkcija, procesas (*[VM]Function, [VM]Process)*	*[BDM]Operation* yra *[BDM]Block* atliekamos procedūros. Jos yra atvaizduojamos į *[VM]Function* arba *[VM]Process* elementus. Šie objektai yra papildomai anotuojami *[BDM]Note* elementais su tipo informacija.
Operacijos parametras *([BDM]Operation Parameter)*	Materialus, informacinis srautas (*[VM]Material Flow, [VM]Informational Flow)*	*[BDM]Operation Parameter* elementai aprašo duomenis, kurie yra perduodami (arba grąžinami) *[BDM]Operation* objektui
Savybė *([BDM]Property)*	Atributas *([VM]Attribute)*	*[BDM]Property* yra *[BDM]Block* vidinis elementas, kuris nusako bloko statinę informaciją. Savybė turi pavadinimą ir reikšmę. Ji yra atvaizduojama kaip *[VM]Attribute* elementas veiklos modelyje.
Apribojimas *([BDM]Constraint)*	Veiklos taisyklė (*[VM]Business Rule*)	*[BDM]Constraints* nusako *[BDM]Block* taikomus apribojimus. Šis elementas veiklos modelyje atvaizduojamas kaip *[VM]Business Rule* elementas.

Struktūros aprašų modelių nuskaitymas yra trečias žingsnis veiklos modelio kūrimo procese. Šio žingsnio pabaigoje egzistuojantys veiklos modelio elementai yra papildomos statinę informaciją aprašančiais laukais.Paskutinis žingsnis veiklos modelio objektų kūrime yra reikalavimų modelių nuskaitymas. Šie modeliai apibrėžia nefunkcinius sistemos reikalavimus.

2.3.4.1.4 VM elementų generavimas iš SysML reikalavimų modelio

Reikalavimų modeliai yra skirti surinkti informaciją apie nefunkcinius sistemos reikalavimus. Surinkti reikalavimai yra nuskaitomi ir susiejami su konkrečiais veiklos modelio elementais. Pagrindiniai reikalavimų modelio elementai yra šie: reikalavimas ir savybė. Reikalavimas yra elementas, kuris tekstiniu formatu (atributuose) aprašo konkretų nefunkcinį sistemos reikalavimą. Reikalavimai yra susiejami su kitais veiklos modelio elementais

(pvz. funkcijomis, procesais, aktoriais). Reikalavimų modelio nuskaitymo algoritmas yra pavaizduotas paveiksle 23.

Pav. 23. Pagrindiniai veiklos modelio elementų generavimo iš reikalavimų modelio žingsniai

Lentelėje žemiau yra aprašyti algoritmo įvesties elementai (reikalavimų modelio elementai) ir išvesties elementai (veiklos modelio elementai).

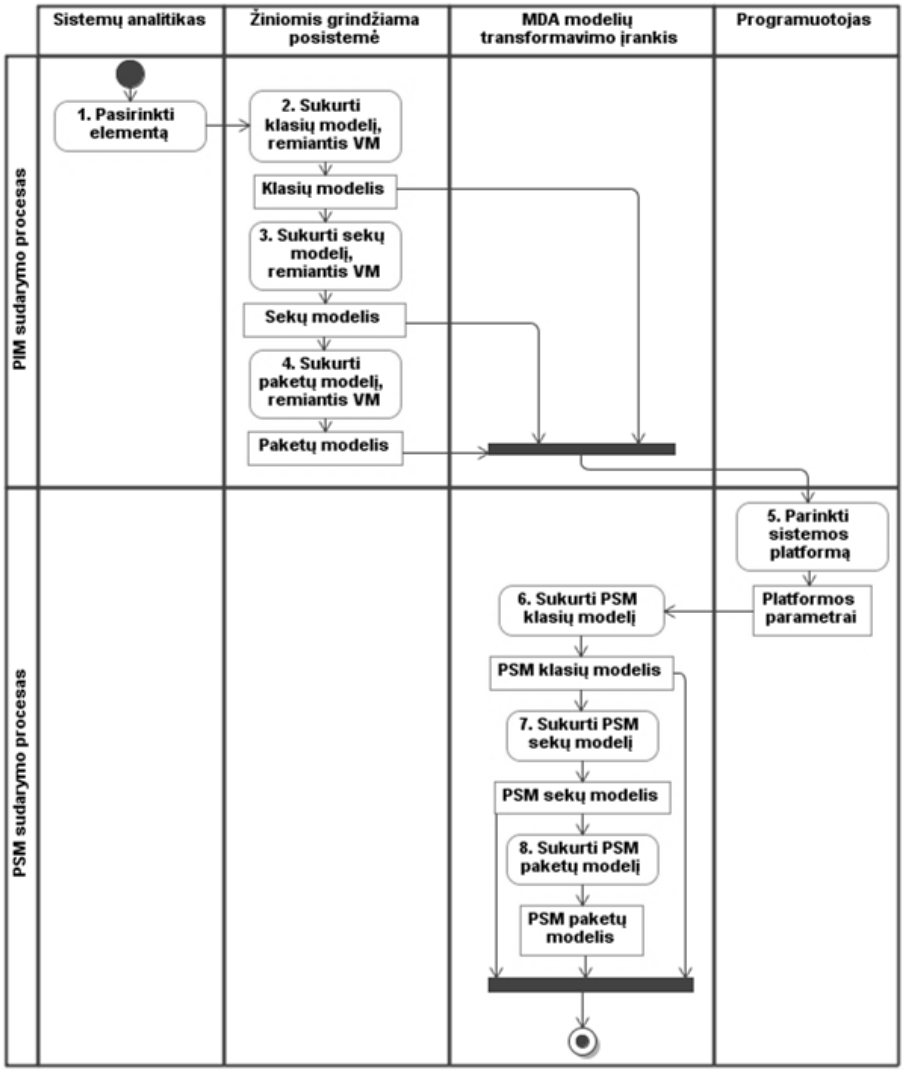
Lentelė 9. SysML reikalavimų ir veiklos modelio elementų sąsajos

SysML reikalavimų modelio elementas	**Veiklos modelio elementas**	**Sąsajų aprašymas**
Reikalavimas *([RM]Requirement)*	Nefunkcinis reikalavimas *[VM]NonFunctionalRequirement*	*[RM]Requirement* yra elementas, kuris skirtas aprašyti nefunkcinius sistemos reikalavimus tekstu. Šis elementas yra atvaizduojamas į *[VM]NonFunctional Requirement* elementą ir gali būti susietas su pasirinktu veiklos modelio elementu. Susiejimo informacija yra saugoma *[RM]Note* elementų pavidalu.
Atributas *([RM]Attribute)*	Atributas *([VM]Attribute)*	*[RM]Attribute* yra elementas, kuris saugo reikalavimo parametrus ir yra atvaizduojamas į *[VM]Attribute* elementą. *[VM]Attribute* turi pavadinimą ir reikšmę.

Reikalavimų modelių nuskaitymas yra paskutinis žingsnis, kurio metu surenkama informacija veiklos modelio generavimui. Kiti CIM proceso etapai apima modelių tikrinimą bei konkrečių modelių transformacijų realizavimą.

2.3.4.2 PIM ir PSM modelių generavimo procesas

PIM modelio generavimo procesas yra pradedamas nuo veiklos modelio elemento parinkimo („1. Parinkti elementą“), kuris bus naudojamas, kaip centrinis elementas PIM modelio generavimui t.y. elementui ir su juo susijusiems elementams yra sukuriamos UML diagramos: klasių (Class) (atvaizduoja statinę sistemos informaciją), sekų (Sequence) (atvaizduoja dinaminę sistemos informaciją), paketų (Package) (atvaizduoja statinę sistemos informaciją). CIM sudarymo procesas yra pateiktas paveiksle 24.



Pav. 24. Pagrindiniai PIM ir PSM modelių generavimo žingsniai

PIM vidiniai modeliai neturi su platforma susijusios informacijos, šios diagramos yra naudojamos PSM vidinių modelių kūrimui, papildant jas konkrečiais platformai būdingais elementais. Prieš sudarant PSM modelį,

būtina parinkti kuriamos programinės įrangos darbinės aplinkos platformą (“5. Parinkti sistemos platformą”). Platformą parenka sistemos architektas, remdamasis informacija surinkta specifikuojant sistemos nefunkcinius reikalavimus. Atlikus platformos parinkimą PIM modelis yra transformuojamas į PSM modelį. Tai yra standartinis MDA metodo etapas. PIM į PSM transformavimo procese dalyvauja MDA modelių transformavimo įrankis. Šio įrankio paskirtis anotuoti PIM modelio elementus konkrečiai platformai būdingais atributais (duomenų tipais, kitais architektūriniais elementais). Detalizuoti PIM ir PSM modelių generavimo etapai yra aprašyti lentelėje žemiau.

Lentelė 10. Žiniomis grindžiamo MDA metodo PIM ir PSM modelių sudarymo žingsniai

#	**Žingsnio pavadinimas**	**Vykdytojas**	**Žingsnio aprašymas**
1	Pasirinkti elementą	Sistemų analitikas	Sistemų analitikas parinka pirminį objektą (pvz. aktorius, funkcija, reikalavimas), kuriam bus generuojamas klasių modelis. Elemento parinkimo metu yra nustatomas generavimo sąsajų gylis t.y. kiek ryšių turi apimti generuojamas klasių modelis.
2	Sukurti klasių modelį, remiantis VM	Žiniomis grindžiama posistemė	Žiniomis grindžiamos posistemės modelių transformavimo algoritmas konvertuoja veiklos modelio elementus į klasių modelio elementus. Sukuriamas XMI failas, kuris gali būti nuskaitomas su UML modeliavimo įrankiu, kuris palaiko XMI formatą. Generavimo algoritmai turi turėti išplėtimo galimybę, tiems atvejams, kai UML įrankis pilnai nesilaiko XMI standarto taisyklių.
3	Sukurti sekų modelį, remiantis VM	Žiniomis grindžiama posistemė	Žiniomis grindžiamos posistemės modelių transformavimo algoritmas konvertuoja veiklos modelio elementus į sekų modelio elementus. Sukuriamas XMI failas, kuris gali būti nuskaitomas su UML modeliavimo įrankiu, kuris palaiko XMI formatą. Generavimo algoritmai turi turėti išplėtimo galimybę, tiems atvejams, kai UML įrankis pilnai nesilaiko XMI standarto taisyklių.
4	Sukurti paketų modelį, remiantis VM	Žiniomis grindžiama posistemė	Žiniomis grindžiamos posistemės modelių transformavimo algoritmas konvertuoja veiklos modelio elementus į paketų modelio elementus. Sukuriamas XMI failas, kuris gali būti nuskaitomas su UML modeliavimo įrankiu, kuris palaiko XMI formatą. Generavimo algoritmai turi turėti išplėtimo galimybę, tiems atvejams, kai UML įrankis pilnai nesilaiko XMI standarto taisyklių.

5	Parinkti sistemos platformą	Programuotojas	Sistemų analitikas parenka platformą, kuriai bus generuojamas PSM modelis iš sukurto PIM modelio.
6	Sukurti PSM klasių modelį	MDA modelių transformavimo įrankis	MDA modelių transformavimo įrankis anotuoją PIM klasių modelį konkrečiai platformai būdingais elementais (duomenų, ryšių tipais).
7	Sukurti PSM sekų modelį	MDA modelių transformavimo įrankis	MDA modelių transformavimo įrankis anotuoją PIM sekų modelį konkrečiai platformai būdingais elementais (duomenų, ryšių tipais).
8	Sukurti PSM paketų modelį	MDA modelių transformavimo įrankis	MDA modelių transformavimo įrankis anotuoją PIM paketų modelį konkrečiai platformai būdingais elementais (duomenų, ryšių tipais).

Esant sukurtam PSM modeliui gali būti atliekamas kitas MDA etapas – programinio kodo generavimas.

2.3.4.2.1 Klasių modelio generavimas iš veiklos modelio

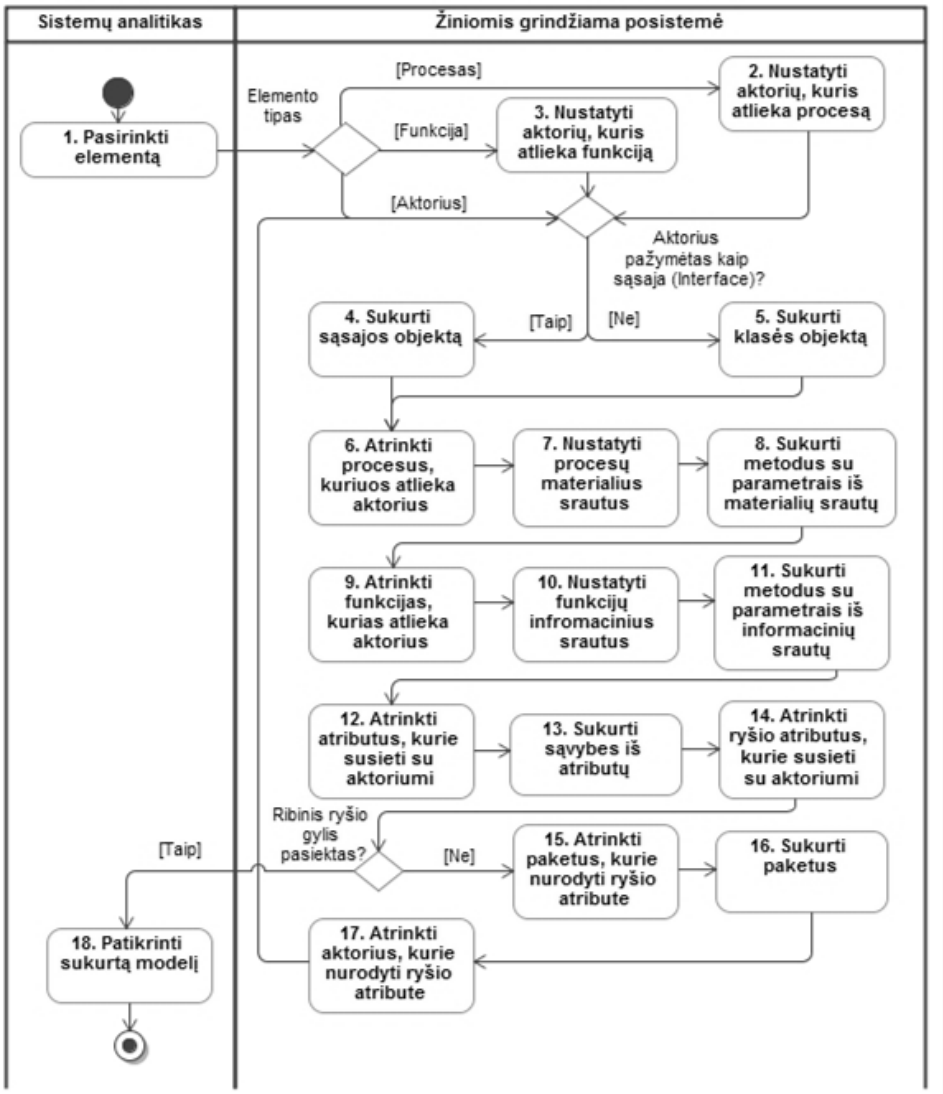
Šiame skyrelyje detalizuosime klasių modelio elementų generavimo iš veiklos modelio procesą. UML klasių modeliai yra plačiai naudojami [48] programinę įrangą kuriančių specialistų (analitikų, architektų, programuotojų) su tikslu perteikti kuriamos sistemos struktūrinę informaciją. Klasikiniu atveju klasių modeliai yra kuriami atlikus pirminę vartotojo reikalavimų analizę ir specifikavimą (sukūrus panaudos atvejų modelius ar/ir aprašius reikalavimus tekstu). Žiniomis grindžiamo MDA metodo atveju klasių modelis yra generuojamas iš veiklos modelio, kuris savo ruožtu yra sukurtas iš CIM modelio. Pradinis klasių modelio generavimo žingsnis yra veiklos modelio elemento (pvz. *[VM]Aktoriaus*, *[VM]Proceso*, *[VM]Funkcijos*), kuriam bus generuojamas klasių modelis, parinkimas. Priklausomai nuo pasirinkto elemento tipo tolimesni veiksmai gali būti:

- *[VM]Aktorius*. Klasė ir klasės savybės yra sukuriami iš veiklos modelio aktoriaus objekto ir jo atributų. Metodai (*[KM]Metodas*) yra kuriami iš pasirinkto *[VM]Aktorius* objekto procesų (*[VM]Procesas*) ir/arba funkcijų (*[VM]Funkcija*). Proceso materialūs srautai (*[VM]Materialus srautas*) ir funkcijos informaciniai srautai (*[VM]Informacinis srautas*) yra

konvertuojami į metodo parametrus. Aktoriaus atributai yra transformuojami į klasės savybes *([KM]Savybė).*

- *[VM]Procesas* – Iš procesų/aktorių sąsajų lentelės yra nustatomas procesą atliekantis aktorius, kiti veiksmai yra atliekami tuo pačiu eiliškumu kaip ir pasirinkus aktoriaus objektą.
- *[VM]Funkcija* - Iš funkcijų/aktorių sąsajų lentelės yra nustatomas procesą atliekantis aktorius, kiti veiksmai yra atliekami tuo pačiu eiliškumu kaip pasirinkus aktoriaus objektą.

Detalizuotas klasių modelio generavimo iš veikos modelio algoritmas yra pavaizduotas žemiau paveiksle ir aprašytas 11 lentelėje.



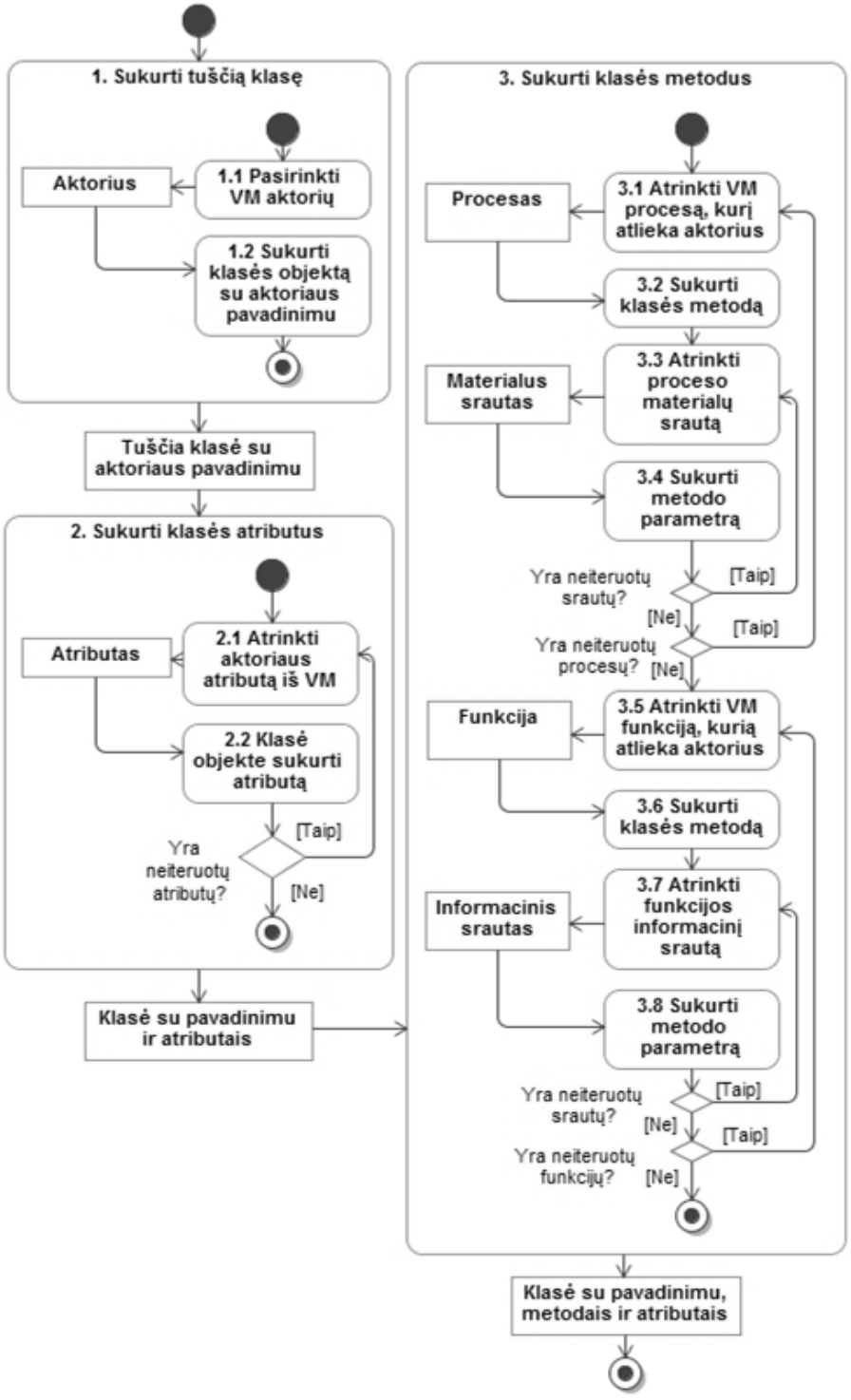
Pav. 25. Pagrindiniai klasių modelių generavimo žingsniai

Lentelė 11. Pagrindiniai klasių modelių generavimo žingsniai

#	Žingsnio pavadinimas	Vykdytojas	Žingsnio aprašymas
1	Pasirinkti elementą	Sistemų analitikas	Klasių modelio generavimo procesas yra pradedamas nuo veiklos modelio elemento parinkimo. Sistemų analitikas parenka aktorių, procesą arba funkciją.
2	Nustatyti aktorių, kuris atlieka procesą	Žiniomis grindžiama posistemė	Jeigu pasirinktas veiklos modelio elementas yra procesas, tokiu atveju nustatomas procesą atliekantis aktorius ir jis naudojamas kaip centrinis klasių modelio generavimo elementas.
3	Nustatyti aktorių, kuris atlieka funkcijų	Žiniomis grindžiama posistemė	Jeigu pasirinktas veiklos modelio elementas yra funkcija, tokiu atveju nustatomas funkciją atliekantis aktorius ir jis naudojamas kaip centrinis klasių modelio generavimo elementas.
4	Sukurti sąsajos objektą	Žiniomis grindžiama posistemė	Jeigu aktorius turi sąsajos anotaciją, sukuriamas klasių modelio sąsajos (angl. *„Interface*“) objektas. Objektui priskiriamas veiklos modelio aktoriaus pavadinimas.
5	Sukurti klasės objektą	Žiniomis grindžiama posistemė	Jeigu aktorius neturi sąsajos anotaciją, sukuriamas klasių modelio klasės (Class) objektas. Objektui priskiriamas veiklos modelio aktoriaus pavadinimas.
6	Atrinkti procesus, kuriuos atlieka aktorius	Žiniomis grindžiama posistemė	Iš veiklos modelyje saugomų procesų yra atrenkami tie procesai, kuriuos atlieka pasirinktas aktorius. Procesai yra naudojami kaip šablonas klasės metodams sukurti.
7	Nustatyti procesų materialius srautus	Žiniomis grindžiama posistemė	Iš veiklos modelyje saugomų materialių srautų yra atrinkami tie srautai, kuriuos naudoja atrinktas procesas. Srautai yra konvertuojami į metodo parametrus.
8	Sukurti metodus su parametrais iš materialių srautų	Žiniomis grindžiama posistemė	Klasės objektas yra papildomas nauju metodu, kuris buvo sukurtas iš aktoriaus atliekamo proceso, o metodo parametrai yra proceso naudojami srautai.
9	Atrinkti funkcijas, kurias atlieka aktorius	Žiniomis grindžiama posistemė	Iš veiklos modelyje saugomų funkcijų yra atrenkamos funkcijos, kurias atlieka pasirinktas aktorius. Funkcijos yra naudojami kaip šablonas klasės metodams sukurti.
10	Nustatyti funkcijų informacinius srautus	Žiniomis grindžiama posistemė	Iš veiklos modelyje saugomų informacinių srautų yra atrenkami tie srautai, kuriuos naudoja konkreti aktoriaus funkcija. Srautai yra konvertuojami į metodo parametrus.
11	Sukurti metodus su parametrais iš informacinių srautų	Žiniomis grindžiama posistemė	Klasės objektas yra papildomas nauju metodu, kuris buvo sukurtas iš aktoriaus atliekamos funkcijos, o metodo parametrai yra funkcijos naudojami srautai.

12	Atrinkti atributus, kurie susieti su aktoriumi	Žiniomis grindžiama posistemė	Iš veiklos modelyje saugomų atributų atrenkami su aktoriumi susiję atributai.
13	Sukurti savybes iš atributų	Žiniomis grindžiama posistemė	Klasės objektas yra papildomas naujomis savybėmis, kurios sukurtos iš veiklos modelio aktoriaus atributų.
14	Atrinkti ryšio atributus, kurie susieti su aktoriumi	Žiniomis grindžiama posistemė	Iš veiklos modelyje saugomų ryšio atributų atrenkami su aktoriumi susiję atributai. Aktoriaus ryšio atributai gali susieti aktorius arba paketus.
15	Atrinkti paketus, kurie nurodyti ryšio atribute	Žiniomis grindžiama posistemė	Iš aktoriaus ryšio atributų atrenkami ryšiai, kurie susieja aktorių su paketais.
16	Sukurti paketus	Žiniomis grindžiama posistemė	Klasių modelyje yra sukuriami paketai.
17	Atrinkti aktorius, kurie nurodyti ryšio atribute	Žiniomis grindžiama posistemė	Iš aktoriaus ryšio atributų atrenkami ryšiai, kurie susieja aktorių su kitais aktoriais. Iš ryšio atributų atrenkami aktoriai. Kiekvienam atrinktam aktoriui procesas pradedamas nuo 4 arba 5 žingsnio, priklausomai nuo aktoriaus tipo.
18	Patikrinti sugeneruotą modelį	Sistemų analitikas	Sistemų analitikas patikrina sukurtą modelį.

Pagrindinis klasių modelio elementas yra klasė. Tai elementas, kuris reprezentuoją konkretų sistemos dalyvį, jo savybes bei galimus atlikti veiksmus. Klasės objekto sukūrimas susideda iš trijų pagrindinių etapų: tuščio klasės šablono sukūrimas. Šiame etape yra sukuriamas klasės objektas su konkretaus veiklos modelio aktoriaus pavadinimu ir tuščiais atributų (savybių) ir metodų sąrašais. Kitas etapas yra užpildyti klasės atributų sąrašą, jeigu atributai yra priskirti konkrečiam aktoriui. Atributai yra atrenkami iš aktoriaus atributų lentelės. Atributą apibrėžia jo pavadinimas ir tipas. Tipas gali būti tekstinis, skaitinis ar objektinis. Šio etapo pabaigoje klasė turi pilnai užpildytą atributų sąrašą. Paskutinis klasės objekto generavimo etapas yra metodų sąrašo užpildymas. Metodą apibrėžia jo pavadinimas bei įvesties ir išvesties parametrai. Metodo parametrai yra sukuriami iš įeinančių ir išeinančių funkcijos ar proceso srautų. Detalizuotas klasės generavimo algoritmo aprašymas yra pateiktas lentelėje 12 ir pavaizduotas paveiksle 26.



Pav. 26. Klasės objekto generavimo žingsniai

Lentelė 12. Klasės objekto generavimo žingsniai

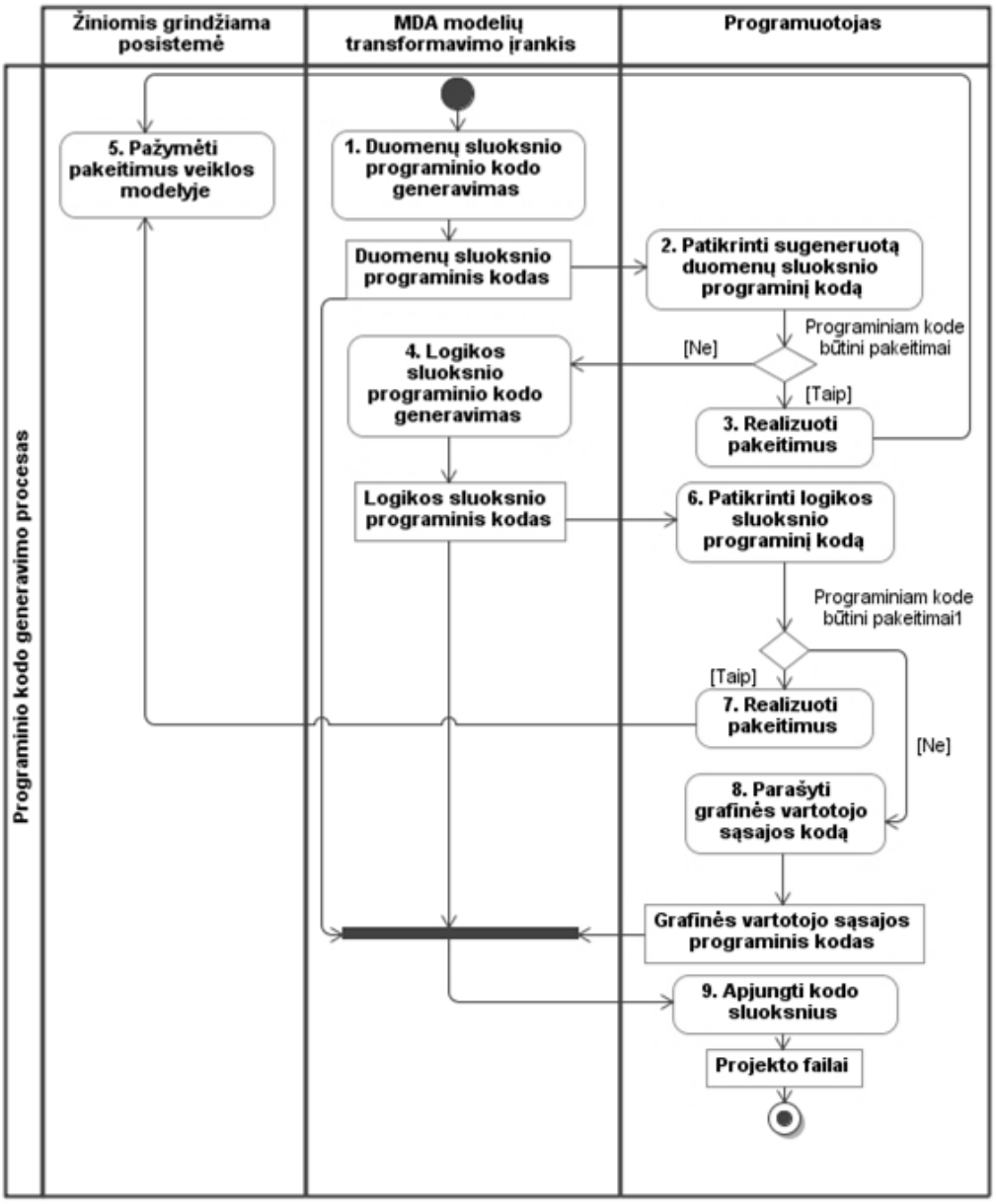
#	Žingsnio pavadinimas	Žingsnio aprašymas
1.1	Pasirinkti VM aktorių	Algoritmas nuosekliai iteruoja per veiklos modelio aktorių sąrašą ir kiekvienam aktoriui inicijuoja naujos klasės sukūrimo procesą.

1.2	Sukurti klasės objektą su aktoriaus pavadinimu	Iš veiklos duomenų bazės yra nuskaitomas pasirinktas aktoriaus objektas. Aktoriaus pavadinimas yra priskiriamas naujai sukurtam klasės tipo objektui.
2.1	Atrinkti aktoriaus atributą iš VM	Vykdoma užklausa į veiklos žinių bazę, kurios metu grąžinamas atributų, priskirtų aktoriui, sąrašas.
2.2	Klasės objekte sukurti atributą	Nuosekliai iteruojama per aktoriaus atributų sąrašą ir kiekvienam atributui sukuriamas savybės laukas aktoriaus objekte.
3.1	Atrinkti VM procesą, kurį atlieka aktorius	Vykdoma užklausa į veiklos žinių bazę, kurios metu grąžinamas procesų, priskirtų aktoriui, sąrašas.
3.2	Sukurti klasės metodą	Nuosekliai iteruojama per aktoriaus procesų sąrašą ir kiekvienam procesui sukuriamas metodas aktoriaus objekte.
3.3	Atrinkti proceso materialųjį srautą	Vykdoma užklausa į veiklos žinių bazę, kurios metu grąžinamas srautų, priskirtų procesui, sąrašas.
3.4	Sukurti metodo parametrą	Nuosekliai iteruojama per proceso srautų sąrašą ir kiekvienam srautui sukuriamas metodo parametras aktoriaus objekte.
3.5	Atrinkti VM funkciją, kurą atlieka aktorius	Vykdoma užklausa į veiklos žinių bazę, kurios metu grąžinamas funkcijų, priskirtų aktoriui, sąrašas.
3.6	Sukurti klasės metodą	Nuosekliai iteruojama per aktoriaus funkcijų sąrašą ir kiekvienai funkcijai sukuriamas metodas aktoriaus objekte.
3.7	Atrinkti funkcijos informacinių srautą	Vykdoma užklausa į veiklos žinių bazę, kurios metu grąžinamas srautų, priskirtų funkcijai, sąrašas.
3.8	Sukurti metodo parametrą	Nuosekliai iteruojama per funkcijos srautų sąrašą ir kiekvienam srautui sukuriamas metodo parametras aktoriaus objekte.

2.3.4.3 Programinio kodo generavimo procesas

Programinio kodo generavimo procesas yra grindžiamas trijų sluoksnių PĮ architektūra, kurią sudaro šie sluoksniai: vartotojo sąsajos sluoksnis (angl. *„GUI“*), logikos sluoksnis (angl. *„Business Logic Layer“*), duomenų sluoksnis (angl. *„Data Layer“*). Procesas yra pradedamas nuo duomenų sluoksnio programinio kodo generavimo (*“1 Duomenų sluoksnio programinio kodo generavimas ”*). Duomenų sluoksnio programinis kodas yra generuojamas iš UML klasių modelio. Klasių modeliai aprašo verslo logikos sluoksnio objektus, todėl atlikus duomenų sluoksnio kodo generavimą tam tikri kodo optimizavimo darbai (pvz. duomenų bazės lentelių normalizavimas) gali būti reikalingi Priklausomai nuo pasirinktos platformos gali būti generuojami SQL skriptai, XML schemos, CSV failų antraštės ir pan. Jeigu atliekamas optimizavimas įtakoja PIM modelio elementus ir reikalauja jų pakeitimo, VM turi būti atnaujintas, kad atitiktų šiuos pakeitimus. Kitas etapas yra logikos sluoksnio elementų generavimas (“4. Logikos sluoksnio programinio kodo

generavimas"). Šio sluoksnio elementai yra generuojami naudojant informaciją iš visų trijų PSM modelio UML modelių: klasių, sekų ir paketų.



Pav. 27. Pagrindiniai programinio kodo generavimo žingsniai

Klasių modelio elementai (klasės) yra naudojami kaip programinio kodo objektų karkasas, paketų diagramos suteikia grupavimo elementų funkcionalumą (pvz. *„Namespace"* .Net programavimo kalboje), o sekų modelio informacija apibrėžia objektų sukūrimo ir metodų kvietimo eiliškumą. Sugeneruoti verslo logikos sluoksnio elementai turi būti patikrinti sistemų architekto ar programuotojo, ir jeigu yra būtini tam tikri kodo pakeitimai, jie gali būti atliekami naudojant programinės įrangos kūrimo įrankius (angl.

„Integrated Development Enviroment - IDE“). Jeigu pakeitimai turi įtakos VM struktūrai šie pakeitimai turi būti įtraukti į veiklos modelį. Programuotojo atlikti programinio kodo pakeitimai yra registruojami veiklos modelyje, tačiau pats veiklos modelis nėra automatiškai sinchronizuojamas, kadangi veiklos modelis gali būti redaguojamas tik per CIM pakeitimus (tokiu būdu išlaikomas žiniomis grindžiamo MDA metodo nuoseklumas). Sistemų analitikas turi priėjimą prie informacijos apie atliktus pakeitimus ir atitinkamai turi atnaujinti CIM modelį bei pergeneruoti veiklos modelį. Detalizuoti programinio kodo generavimo žingsniai yra aprašyti lentelėje:

Lentelė 13. Žiniomis grindžiamo MDA metodo programinio kodo generavimo žingsniai

#	Žingsnio pavadinimas	Vykdytojas	Žingsnio aprašymas
1	Duomenų sluoksnio programinio kodo generavimas	MDA modelių transformavimo įrankis	MDA modelių transformavimo įrankis transformuoja PSM modelį į duomenų sluoksnio programinį kodą (pvz. SQL skriptai). Transformacija atliekama iš klasių modelio.
2	Patikrinti sugeneruotą duomenų sluoksnio programinį kodą	Programuotojas	Programuotojas patikrina sugeneruotą kodą ir nustato kokie pakeitimai yra būtini.
3	Realizuoti pakeitimus	Programuotojas	Programuotojas realizuoja programinio kodo pakeitimus t.y. papildo duomenų programinį kodą papildomų laukų aprašymu arba savybėmis (pvz. *auto increment*, *primary key*, *foreign key*).
4	Logikos sluoksnio programinio kodo generavimas	MDA modelių transformavimo įrankis	MDA modelių transformavimo įrankis transformuoja PSM modelį į logikos sluoksnio programinį kodą (pasirinktos programavimo kalbos programinį kodą). Transformacija atliekama iš klasių, sekų ir paketų modelių.
5	Pažymėti pakeitimus veiklos modelyje	Žiniomis grindžiama posistemė	Veiklos modelyje *ChangeSet* elementuose yra įrašomi atlikti pakeitimai (klasių, metodų, savybių pavadinimų pakeitimai, hierarchijos pakeitimai, iškvietimo eiliškumo pakeitimai, paketų sudėties pakeitimai). Pakeitimų stebėjimas yra reikalingas siekiant išlaikyti sinchronizaciją tarp skirtingų modelių.
6	Patikrinti logikos sluoksnio programinį kodą	Programuotojas	Programuotojas patikrina sugeneruotą kodą ir nustato kokie pakeitimai yra būtini.
7	Realizuoti pakeitimus	Programuotojas	Programuotojas realizuoja programinio kodo pakeitimus (papildo klasių hierarchijos struktūrą. pakeičia klasių, savybių, metodų pavadinimus, įtraukia papildomą anotaciją, komentarus).

8	Parašyti grafinės vartotojo sąsajos kodą	Programuotojas	Programuotojas sukuria vartotojo sąsajos programinį kodą.
9	Apjungti kodo sluoksnius	Programuotojas	Visi trys kodo sluoksniai apjungiami į vientisą projektą, atliekamas integravimo klaidų taisymas.

Sukūrus duomenų ir verslo logikos sluoksniu, gali būti generuojamas vartotojo sąsajos sluoksnis. Šiam sluoksniui sukurti gali būti naudojamos modifikuotos klasių diagramos [81] arba šio sluoksnio objektai yra sukuriami programuotojo. Tam tikrais atvejais grafinė vartotojo sąsaja nėra būtina (pvz. kuriamos tik klasių bibliotekos ar programos funkcijos iškviečiamos naudojantis konsole). Esant sugeneruotiems visiems programinio kodo sluoksniams, jie yra apjungiami į vieną projektą, naudojantis pasirinkta IDE. Atlikus sluoksnių apjungimą į vieną projektą, galima pradėti dalykinės programos testavimą. Testavimui yra naudojami veiklos scenarijai aprašyti panaudos atvejų ir veiklų modeliuose.

2.4 Išvados

Žiniomis grindžiamas MDA metodas apjungia dviejų IS inžinerijos krypčių principus (žiniomis grindžiamos ir modeliais grindžiamos). Metodo pagrindu yra OMG grupės sukurta modelių panaudojimo programinės įrangos kūrimo procese koncepcija – MDA. Iš šios koncepcijos žiniomis grindžiamame MDA metode yra naudojami pagrindiniai modelių tipai (CIM, PIM, PSM) bei modelių transformavimo idėja. Žiniomis grindžiamos IS inžinerijos principų integravimas yra atliekamas per žinių posistemės naudojimą, kurią sudaro: veiklos modelis, veiklos metamodelis, modelių transformavimo ir tikrinimo algoritmai. Žiniomis grindžiamo MDA metodo sudėtiniai elementai yra:

- Modeliavimo kalbos: SysML ir UML;
- Modeliai: CIM, VM, PIM, PSM, programinis kodas;
- Transformacijos: CIM => VM, VM=>PIM, PIM=>PSM, PSM=>Code;
- Žiniomis grindžiamo MDA metodo procesas.

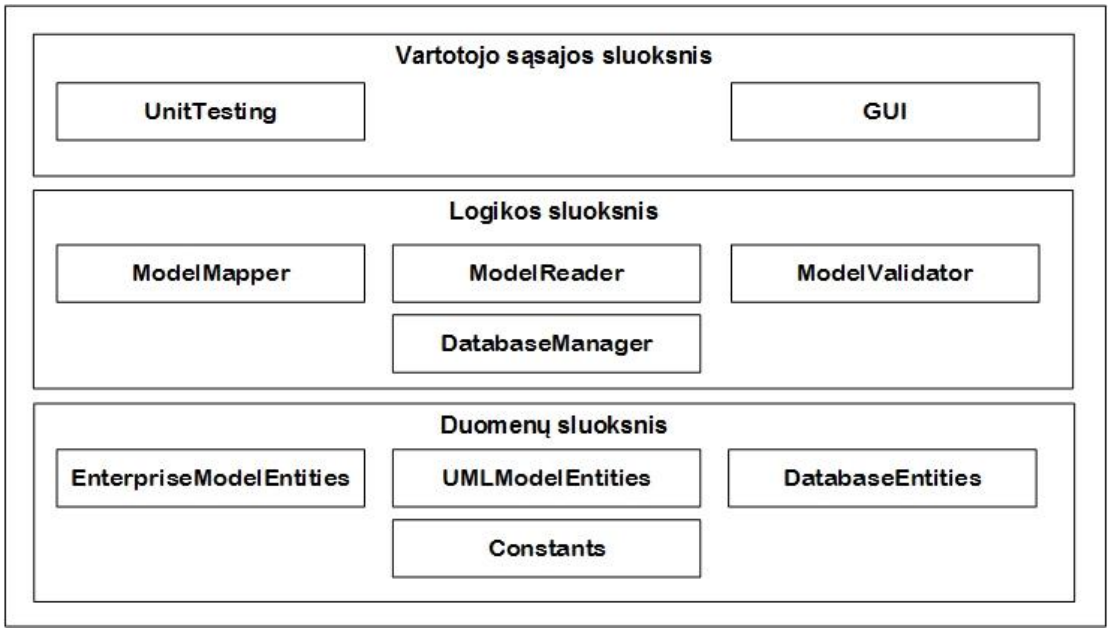
Skyriuje detaliai aprašyta: veiklos modelio sudėtis, modelių sąsajų žemėlapiai (angl. *mappings*), žiniomis grindžiamo MDA metodo proceso etapai, SysML ir

UML modelių transformavimo į veiklos modelį algoritmai. Rezultatus publikuojantys straipsniai yra išpausdinti mokslo žurnaluose „Electronics And Electrical Engineering“ (žr. žurnalų publikacijų sąrašą nr.: 2, 4) „Transformations in Business & Economics“ (žr. žurnalų publikacijų sąrašą nr.: 3), proceedings (Lecture Notes in Business Information Processing/ Lecture Notes in Computer Science) (žr. proceedings publikacijų sąrašą nr.: 2, 3, 4).

3 ŽINIOMIS GRINDŽIAMO MDA METODO DALYKINĖS PROGRAMOS PROTOTIPAS

3.1 Žiniomis grindžiamo MDA metodo prototipo komponentai

Žiniomis grindžiamo MDA metodo dalykinės programos įrankio prototipas yra sukurtas naudojant Microsoft Visual Studio 2012 programavimo aplinką (IDE) [58], .Net Framework 4.5 programavimo karkasą [55] bei Resharper [39] programinio kodo optimizavimo įrankį. Duomenų bazei sukurti buvo naudojamas Microsoft SQL Server Express [57] duomenų bazės valdymo sistema. Prototipas buvo kuriamas remiantis modulių koncepcija t.y. tokiu būdu kad daugumą elementų būtų įmanoma panaudoti kituose tyrimuose kaip atskiras bibliotekas. Prototipo architektūra yra trisluoksnė t.y. sudaryta iš šių kokybiškai skirtingų sluoksnių (angl. *„Layer“)*: vartotojo sąsajos sluoksnis (angl. *„UI“*), logikos sluoksnis, duomenų sluoksnis. Vartotojo sąsajos sluoksnio elementai negali tiesiogiai kreiptis į duomenų sluoksnio modulius t.y. visi veiksmai turi būti atliekami naudojant logikos sluoksnio objektus ir funkcijas. Bendras architektūros vaizdas ir trumpas modulių aprašymas yra pateiktas žemiau, o detalus aprašymas - kituose skyriuose.



Pav. 28. Žiniomis grindžiamo MDA metodo dalykinės programos prototipo moduliai

Žiniomis grindžiamo MDA metodo įrankio dalykinės programos prototipas yra sudarytas iš šių modulių:

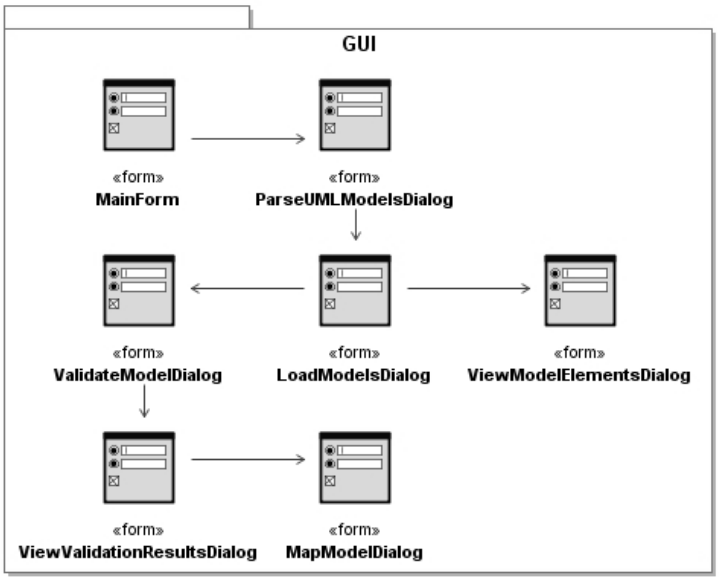
- Vartotojo sąsajos sluoksnis:
  - *Grafinės vartotojo sąsajos* (angl. *„GUI“*) modulis apjungia grafinės vartotojo sąsajos elementus tokius kaip: formos ir dialogai. Šio modulio paskirtis priimti vartotojo įvedamus duomenis, pateikti vartotojui pasirinkimus bei atvaizduoti atliktų veiksmų rezultatus.
  - *Testavimo* (angl. *„Unit Testing“*) modulis yra skirtas atlikti automatinį dalykinės programos testavimą (verslo logikos bei duomenų sluoksnių). Šio modulio klasės yra naudojamos tik testavimo tikslams. Automatinis testavimas turi užtikrinti dalykinės programos integralumą bei teisingą veikimą, ją plečiant naujais elementais ir moduliais.
- Logikos sluoksnio moduliai (angl. *„Logic Layer“*):
  - *Bylų nuskaitymo* (angl. *„ModelReader“*) modulis apjungia dalykinės programos elementus, kurie yra skirti *MagicDraw* UML failo, išsaugoto XMI formatu nuskaitymui. Nuskaityti duomenys (UML modelio elementai ir ryšiai) yra išsaugomi kaip modulio *UMLModelsEntities* objektai programos atmintyje.
  - *Modelių tikrinimo* (angl. *„ModelValidator“*) modulis apjungia dalykinės programos elementus, kurių paskirtis - UML modelių tikrinimas įvairių kriterijų (pvz. nuoseklumo) atžvilgiu.
  - *Modelių susiejimo* (angl. *„ModelMapper“)* modulis apjungia dalykinės programos elementus, kurių paskirtis transformuoti UML modelio objektus (*UMLModelEntities*) į veiklos modelio objektus (*EnterpriseModelEntities*).
  - *Duomenų bazės valdymo modulis* (angl. *„DatabaseManager“*) yra atsakingas už veiklos modelio elementų transformavimą į duomenų bazės elementus (*DatabaseEntities*) bei jų išsaugojimą ir nuskaitymą iš duomenų bazės.
- Duomenų sluoksnio moduliai (angl. *„Data Layer“)*:
  - *UML modelio esybių modulis* (angl. *„UMLModelEntities“*) yra skirtas perkelti iš XMI failo nuskaitytus duomenis į programos atmintyje sukuriamus objektus. Šie objektai yra naudojami UML modelių

tikrinimui bei transformavimui į veiklos modelio elementus.

- *Veiklos modelio esybių modulis* (angl. *„EMEntities“*). Šio modulio klasės yra naudojamos kuri veiklos modelio objektus programos atmintyje. Veiklos modelio objektai yra naudojami atlikti įvairius veiksmus (analizė, tikrinimas, transformavimas) su veiklos modeliu veiksmus.
- *Duomenų bazės esybių modulis* (angl. *„Database Entities“*). Šiame modulyje yra ORM įrankio sukurtos klasės, kurios naudojamos duomenų bazės įrašų atnaujinimui, sukūrimui, trynimui ir kitiems veiksmams.
- *Konstantų modulis* (angl. *„Constants“*). Šiame modulyje yra saugoma statinė dalykinės programos prototipo informacija, tokia kaip konstantų reikšmės, žinučių ir klaidų tekstai ir pan.

3.1.1 Vartotojo sąsajos modulis (GUI)

Grafinės vartotojo sąsajos modulis yra atsakingas už sąsajos su vartotoju valdymą. Pagrindinis šio modulio elementas yra forma. Visas prototipe naudojamas formas galima suskirstyti į dvi grupes: duomenų įvedimo bei rezultatų atvaizdavimo. Dalykinės programos prototipe naudojamų formų sąrašas yra pavaizduotas ir aprašytas žemiau.



Pav. 29. Vartotojo sąsajos modulio elementai

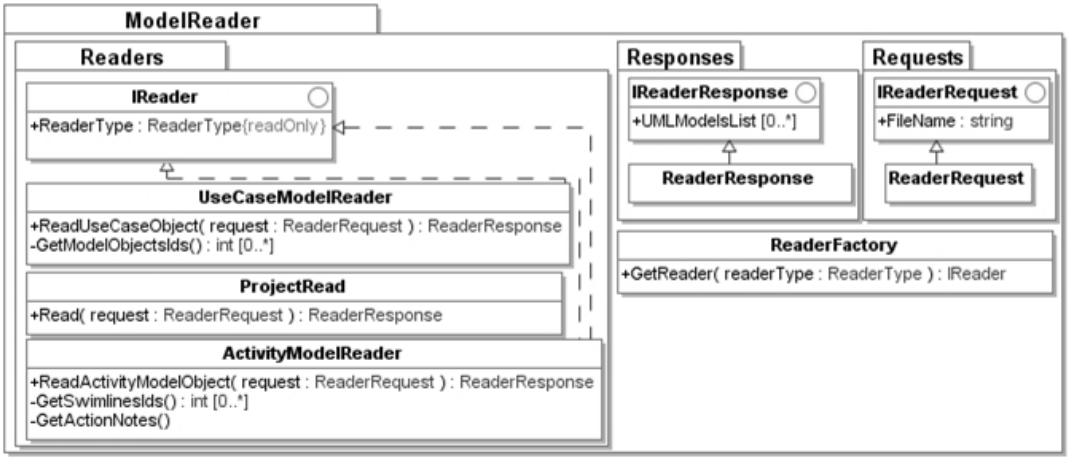
- *Pagrindinė forma* (angl. *„MainForm“*). Tai pirminė programos forma. Ji yra parodoma programos startavimo metu. Formoje yra meniu, kuris yra skirtas valdyti vartotojo navigaciją į kitas formas.
- *UML modelių nuskaitymo forma* (angl. *„ParseUMLModelsDialog“*). Ši forma yra skirta nuskaityti MagicDraw modeliavimo įrankiu sukurtus UML modelių failus. Formoje vartotojas turi nurodyti kelią iki failo. Failai turi būti išsaugoti XMI formatu.
- *Modelių tikrinimo forma* (angl. *„ValidateModelDialog“*). Ši forma yra skirta parinkti parametrus UML modelių nuoseklumo tikrinimui. Vartotojas turi pasirinkti pagrindinį modelį (angl. *„MainModel“*), kuris naudojamas, kaip tikrinimo etalonas (angl. *„Baseline“*), bei veidrodinį modelį (angl. *„MirrorModel“*).
- *Nuskaitytų modelių sąrašo forma* (angl. *„LoadModelsDialog“*). Ši forma yra skirta parodyti iš XMI failo nuskaitytų modelių sąrašą. Sąraše yra rodomas modelio pavadinimas ir tipas. Prototipas gali nuskaityti dviejų tipų modelius – panaudos atvejų (angl. *„Use Case“*) ir veiklų (angl. *„Activity“*).
- *Modelių elementų peržiūros forma* (angl. *„ViewModelElementsDialog“*). Šioje formoje vartotojui yra pateikiamas vieno konkretaus UML modelio elementų (aktorių, panaudos atvejų, veiklų ir kt.) sąrašas.
- *Tikrinimo rezultatų forma* (angl. *„ViewValidationResultsDialog“*). Ši forma yra skirta atvaizduoti modelių nuoseklumo tikrinimo rezultatus. Forma pateikia informaciją apie elementus, kurie egzistuoja/yra trūkstami pagrindiniame/veidrodiniame modeliuose.
- *Modelių susiejimo forma* (angl. *„MapModelDialog“*). Ši forma yra skirta pateikti informaciją vartotojui apie taip, kaip UML modelio elementai bus konvertuojami į veiklos modelio elementus. Šioje formoje vartotojas gali nustatyti tam tikrus konvertavimo parametrus.

Visos aprašytos dalykinės programos prototipo formos gali atlikti veiksmus su *panaudos atvejų* ir *veiklų* modeliais. Formos yra sukurtos tokiu būdu, kad jų

išplėtimas būtų nesudėtingas t.y. naudojami dizaino šablonai (angl. *„Design Pattern“*) tokie kaip objektų gamyklos (angl. *„Factory“)* taip pat objektų architektūra pagrįsta sąsajų (angl. „*Interface“*) naudojimu. Formos yra sukurtos naudojant *.Net Form* objektus bei užpildytos grafiniais *.Net* bibliotekos objektais tokiais kaip teksto įvedimo laukai (angl. „*Textbox“*), užrašai (angl. „*Label“*), duomenų lentelės (angl. „*DataGridView“*).

3.1.2 UML modelių bylų nuskaitymo modulis (ModelReader)

UML modelių bylų nuskaitymo modulis yra skirtas nuskaityti MagicDraw [20] modeliavimo įrankiu sukurtas XMI formato bylas. XMI nuskaitymui naudojamos *XMLDocument*, *XMLNode .NET* bibliotekos klasės skirtos darbui su XML failais. Šio modulio architektūros pagrindas yra bazinė skaitytuvo (angl. „*Reader“*) sąsaja (angl. „*Interface“*). Kiekvienas modelis (*panaudos atvejų, veiklų, reikalavimų, struktūros aprašų*) privalo turėti jam skirtą skaitytuvą, kuris realizuoja sąsajos *IReader* funkciją *ReadModelObjects* ir savybę *ReaderType*. Dalykinės programos prototipe yra sukurti *panaudos atvejų* ir *veiklų* modelių skaitytuvai. Skaitytuvai yra kuriami *ReaderFactory* objektui nurodant reikiamo skaitytuvo tipą. Be pagrindinio *ReaderObjectModel* metodo kiekvienas skaitytuvas gali turėti specifinius metodus ir savybes, kurios gali būti reikalingos siekiant nuskaityti konkretaus tipo modelį. Žemiau pateikta skaitytuvų modulio sudėtis ir ryšiai bei lentelėje aprašyti elementai.



Pav. 30. Modelių bylų nuskaitymo modulio pagrindiniai elementai

Lentelė 14. Bylų nuskaitymo modulio elementai

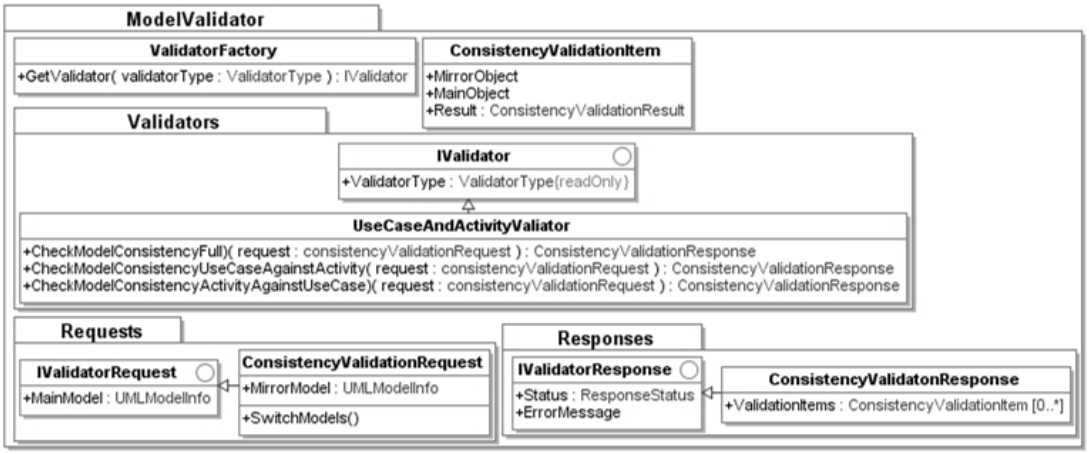
Elementas	Aprašymas
IReaderResponse	Skaitytuvo *(Reader)* rezultato sąsaja, aprašanti grąžinamo rezultato būseną ir klaidos pranešimą.
IReaderRequest	Skaitytuvo *(Reader)* užklausos sąsaja, aprašanti UML modelio failą.
IReader	Skaitytuvo sąsaja aprašanti būtinus skaitytuvų metodus.
ReaderResponse	Skaitytuvo rezultatų klasė.
ReaderRequest	Skaitytuvo užklausos klasė.
ReaderFactory	Klasė skirta sukurti konkretų skaitytuvą priklausomai nuo modelio tipo.
ActivityModelReader	Klasė, kuri realizuoja IReader sąsają ir yra skirta nuskaityti Activity modelio duomenis.
UseCaseModelReader	Klasė, kuri realizuoja IReader sąsają ir yra skirta nuskaityti Use Case modelio duomenis.
ProjectReader	Klasė, kuri realizuoja IReader sąsają ir yra skirta nuskaityti visus XMI faile esančius modelius.

ProjectReader yra klasė, kuri realizuoja *IReader* sąsają, bet skirtingai nei kiti skaitytuvai ši klasė nuskaito visus XMI faile esančius UML modelius nepriklausomai nuo jų tipo. Pateikta skaitytuvų architektūra įgalina lengvai išplėsti egzistuojantį modulį naujais skaitytuvais, tačiau visi nauji skaitytuvai turi realizuoti *IReader* sąsajos metodus ir savybes bei turi būti įtraukti į *ReaderFactory* ir *ProjectReader* klases.

3.1.3 UML modelių tikrinimo modulis (ModelValidator)

UML modelių tikrinimo modulis yra naudojamas apjungti UML modelių tikrinimui skirtas klases. Dalykinės programos prototipe yra realizuotas modelių nuoseklumo tikrinimas tarp dviejų modelių tipų :*panaudos atvejų* ir *veiklos*. Nuoseklumo tikrinimo procese modeliai yra skirstomi į pagrindinius ir veidrodinius. Pagrindinis modelis yra traktuojamas kaip etalonas ir esant neatitikimams klaidingu yra laikomas veidrodinis modelis. Prototipe yra tikrinama ar panaudos atvejų modelio elementas *aktorius* yra pavaizduotas kaip *veiklų grupavimo* elementas *veiklų* modelyje. Jeigu šio elemento nėra tai gali reikšti, kad modelio sudarytojas suklydo rašydamas pavadinimą (tokiu atveju bus rodomas *veiklų* modelio elementas, kuris neturi sau poros su *panaudos atvejų* modelio elementu), arba kitu atveju skirsis elementų skaičius, kas reiškia jog modeliuose yra perteklinis elementas, kuris turi būti pašalintas iš sistemos, arba trūkstamas elementas, kuris turi būti įtrauktas į sistemą. Šiais

abiem atvejais sistemų analitikas turi patikrinti konkrečius UML modelius, remiantis klaidų sąrašu, ir pašalinti esamus trūkumus. Žemiau pateikta modelių tikrinimo modulio sudėtis ir ryšiai bei lentelėje aprašyti elementai.



Pav. 31. Modelių tikrinimo modulio pagrindiniai elementai

Lentelė 15. Modelių tikrinimo modulio elementai

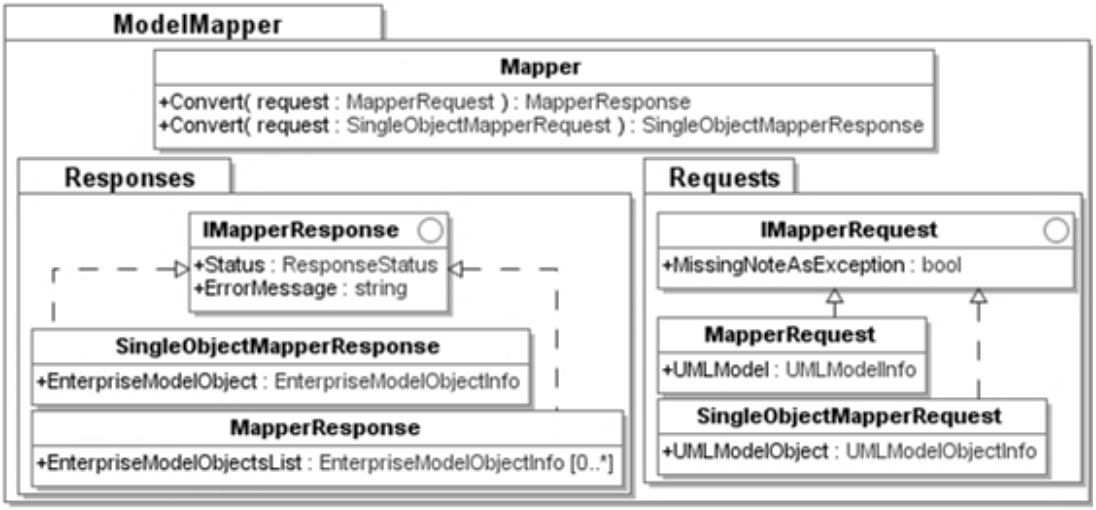
Elementas	Aprašymas
IValidator	Bendra sąsaja, kurią turi realizuoti visos tikrinimą atliekančios klasės *(Validators)*.
UseCaseAndActivityValidator	Klasė, kuri atlieka *Use Case* ir *Activity* modelių tarpusavio nuoseklumo tikrinimą. Klasė realizuoja IValidator sąsają.
ConsistencyValidationRequest	Klasė, kuri yra skirta perduoti parametrus tikrinimą atliekančiai klasei.
ConsistencyValidationResponse	Klasė, kuri skirta grąžinti tikrinimo rezultatus.
ConsistencyValidationItem	Klasė, kuri skirta gražinti konkrečių elementų tikrinimo rezultatus.

Visos naujos modelių tikrinimo klasės turi realizuoti *IValidator* sąsają bei joje aprašytus metodus ir savybes.

3.1.4 Modelių susiejimo modulis (ModelMapper)

Modelių susiejimo modulis yra skirtas konvertuoti UML modelių elementus į veiklos modelio (VM) elementus. Pagrindinis modulio elementas yra susiejimo klasė (angl. „*Mapper*“). Šios klasės metodas kaip įvestį priima UML modelio elementus (*MapperRequest* objektas, kuris turi sąrašą UML modelio elementų *UMLModelObjectInfo*) ir grąžina veiklos modelio elementų sąrašą (*EnterpriseModeObjectInfo*). Modelių susiejimo klasė yra statinio tipo, kadangi ji yra nepriklausoma nuo modelio tipo ir turi informaciją apie visus *UMLModelObjectInfo* ir *EnterpriseModelObjectInfo* realizuojančius objektus.

Vartotojas gali perduoti papildomus parametrus konvertavimo metodui. Žemiau pateikta modelių susiejimo modulio sudėtis ir ryšiai bei lentelėje aprašyti elementai.



Pav. 32. Modelių susiejimo modulio pagrindiniai elementai

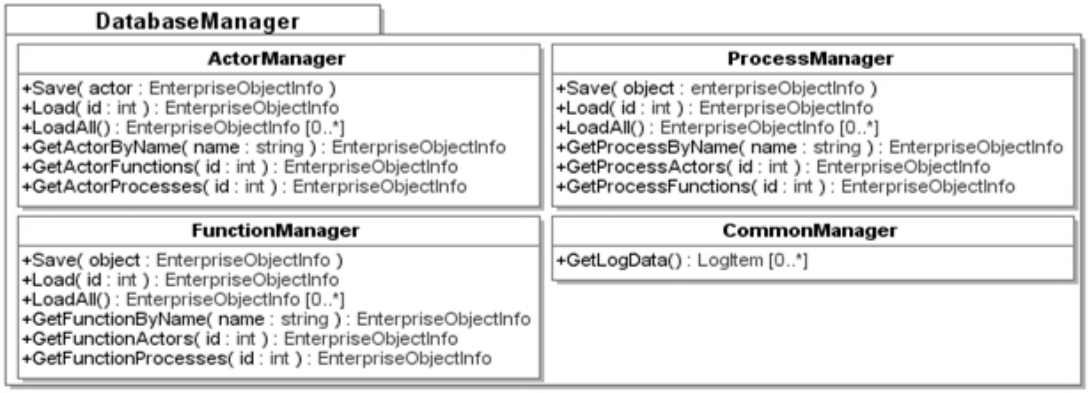
Lentelė 16. Modelių susiejimo modulio elementai

Elementas	Aprašymas
Mapper	Klasė, kuri skirta atlikti modelių elementų transformacijas. *UMLModelEntities* objektai yra transformuojami į *EnterpriseModelEntities* objektus. Transformacijai parametrai yra nurodomi vartotojo arba parenkami automatiškai.
IMapperResponse	*Mapper* klasės užklausos sąsaja.
IMapperRequest	*Mapper* klasės rezultato sąsaja.
MapperRequest	Klasė skirta perduoti parametrus objektų transformavimo procesui.
SingleObjectMapperResponse	Klasė skirta grąžinti konkretaus modelio elemento transformavimo rezultatą.
MapperResponse	Klasė skirta grąžinti transformavimo rezultatą aprašančius parametrus.
SingleObjectMapperRequest	Klasė skirta perduoti konkretaus modelio elemento transformavimui reikalingus parametrus.

3.1.5 Duomenų bazės funkcijų modulis (Database Manager)

Duomenų bazės funkcijų modulis (angl. „*DatabaseManager*“) yra skirtas realizuoti *CRUD* (angl. „*Create, Read, Update, Delete*“) veiksmus su duomenų baze. Pagrindiniai šio modulio objektai yra valdikliai (angl. „*Manager*“). Kiekvienas veiklos modelio elementas turi jam skirtą duomenų bazės valdiklį (pvz. *ActorManager, FunctionManager* ir kt.). Dalis valdiklių metodų atlieka kokybiškai identiškas funkcijas, tokias kaip: sukurti naują

objektą, ištrinti objektą, kiti metodai yra specializuotos paskirties ir skirti tam tikram veiksmui su konkrečiu veiklos modelio elementu. Bendras valdiklis (angl. „*CommonManager*“) yra skirtas atlikti visiems veiklos modelio elementams bendrus veiksmus arba priešingai, specifinius veiksmus, kurie yra neįtraukti nė vieno valdiklio funkcijas (pvz. duomenų bazės lentelės išvalymas). Žemiau pateikta duomenų bazės funkcijų modulio sudėtis ir ryšiai bei lentelėje aprašyti elementai.



Pav. 33. Duomenų valdymo modulio pagrindiniai elementai

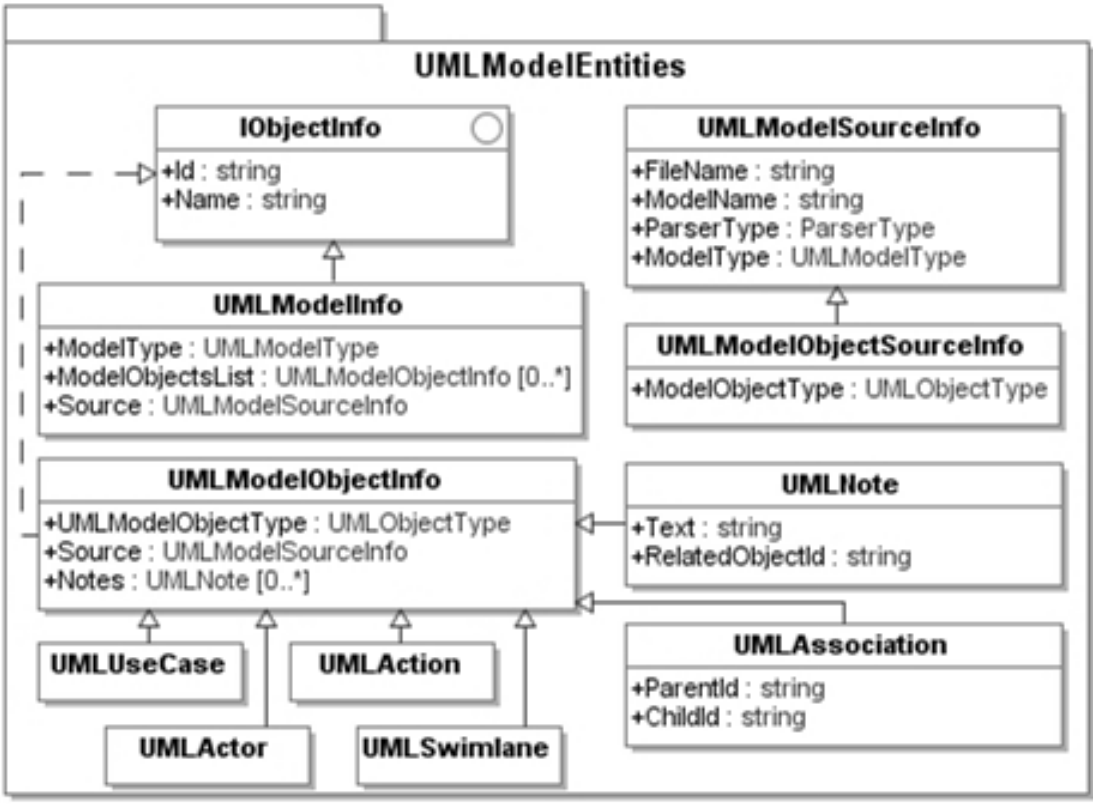
Lentelė 17. Duomenų valdymo modulio elementai

Elementas	Aprašymas
ActorManager	Klasės, kurios skirtos konkrečių veiklos modelio elementų veiksmams su duomenų baze atlikti (išsaugoti, nuskaityti, atnaujinti, ištrinti).
FunctionManager	
ProcessManager	
CommonManager	Klasė skirta atlikti bendriems veiksmams su duomenų baze.

3.1.6 UML modelių esybių modulis (UMLModelEntities)

UML modelių esybių modulis (angl. *„UMLModelEntities“*) yra sudarytas iš aibės klasių, kurios yra skirtos saugoti UML modelio elementų informaciją, nuskaitytą iš XMI bylų, programos vykdymo metu (angl. *„Runtime“*). Klasės turi metodus, kurie yra naudojami tikrinimo bei transformavimo funkcijoms atlikti, o klasių pavadinimai atitinka UML modelio elementų pavadinimus su priešdėliu UML. Pvz. UML objekto *Actor* iš *panaudos atvejų* modelio duomenys yra saugomi sukuriant naują objektą *UMLActor* ir į jo savybių sąrašą įrašant atributus nuskaitytus iš XMI bylos. Kiekvienas šio modulio elementas yra paveldėtas iš bazinės klasės *UMLModelObjectInfo*. Ši klasė turi pagrindinius laukus, kurie yra bendri visoms pasiveldinčioms klasėms, tokius

kaip: elemento pavadinimas, Id, tipas bei šaltinis (angl. „*Id, Name, UMLObjectType, Source*“). Paveiksle žemiau yra pateikta šio modulio sudėtis ir ryšiai, o lentelėje detalizuoti elementai.



Pav. 34. UML modelių esybių modulio pagrindiniai elementai

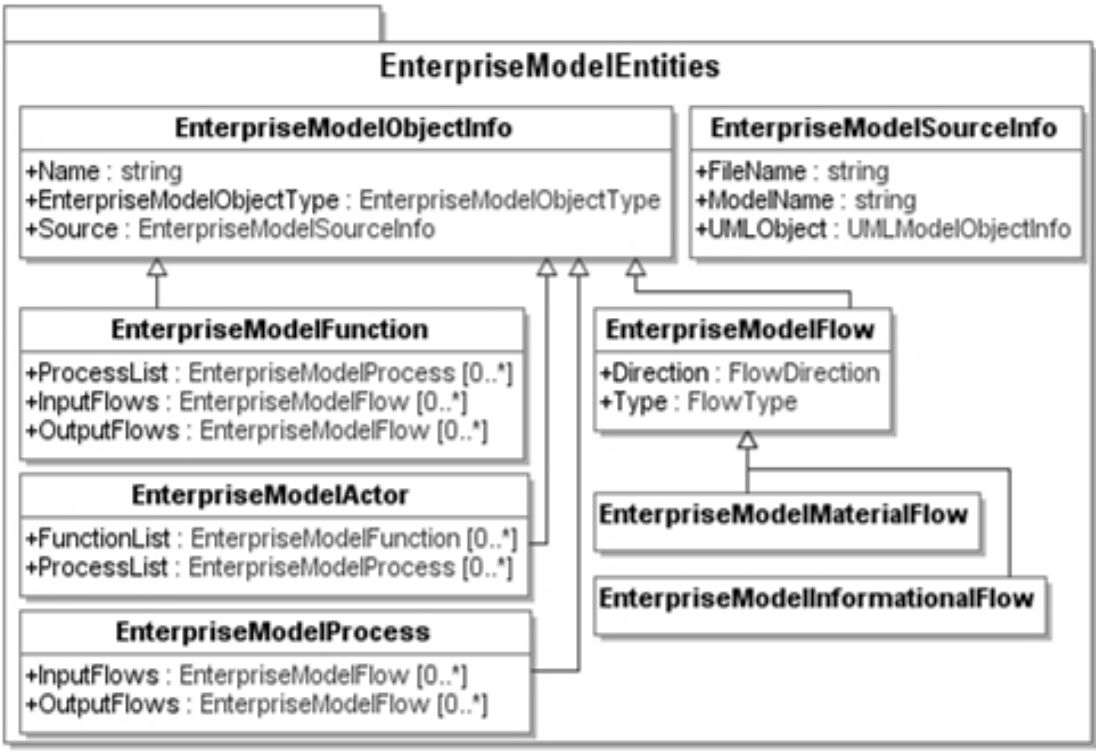
Lentelė 18. UML modelių esybių modulio elementai

Elementas	Aprašymas
UMLModelObjectInfo	Bazinė klasė iš kurios pasiveldi visos kitos *UMLModelEntities* klasės. Ši klasė realizuoja *IObjectInfo* sąsają.
UMLModelInfo	Klasė, kuri skirta saugoti bendrą UML modelio informaciją, tokią kaip modelio tipas, pavadinimas, modelio elementų sąrašas ir kt.
UMLModelSourceInfo	Klasė, kuri skirta saugoti modelio kilmės informaciją, tokią kaip pradinio failo pavadinimas, skaitytuvo tipas, modelio pavadinimas.
UMLUseCase, UMlActor, UMLAction, UMLSwimline, UMLAssociation, UMLNote	Klasės, kurios pasiveldi *UMLModelObjectInfo* klasė. Šio klasės saugo konkretaus UML modelio informaciją programos atmintyje.

Aprašytos esybės apima tik pagrindinius *UseCase* ir *Activity* modelių elementus, bet sukurta architektūra turi būti naudojama kaip pagrindas kuriant elementus kitiems UML modeliams.

3.1.7 Veiklos modelio esybių modulis (Enterprise Model Entities)

Veiklos modelio esybių modulis (angl. *„EnterpriseModelEntities“*) yra sudarytas iš aibės klasių, kurios saugo veiklos modelio elementų informaciją programos vykdymo metu (angl. *„Runtime“*). Veiklos modelio elementai yra sukuriami iš UML modelio elementų, atliekant elementų transformacijas. Visi veiklos modelio elementai pasiveldi iš bazinės klasės *EnterpriseModelObject*. Bendri paveldimi laukai yra pavadinimas, tipas, šaltinis (angl. „*Name, EnterpriseModelObjectType, Source“*). Šaltinis yra *EnterpriseModelSourceInfo* tipo objektas. Paveiksle žemiau yra pateikta šio modulio sudėtis ir ryšiai, o lentelėje detalizuoti elementai.



Pav. 35. Veiklos modelio esybių modulio pagrindiniai elementai

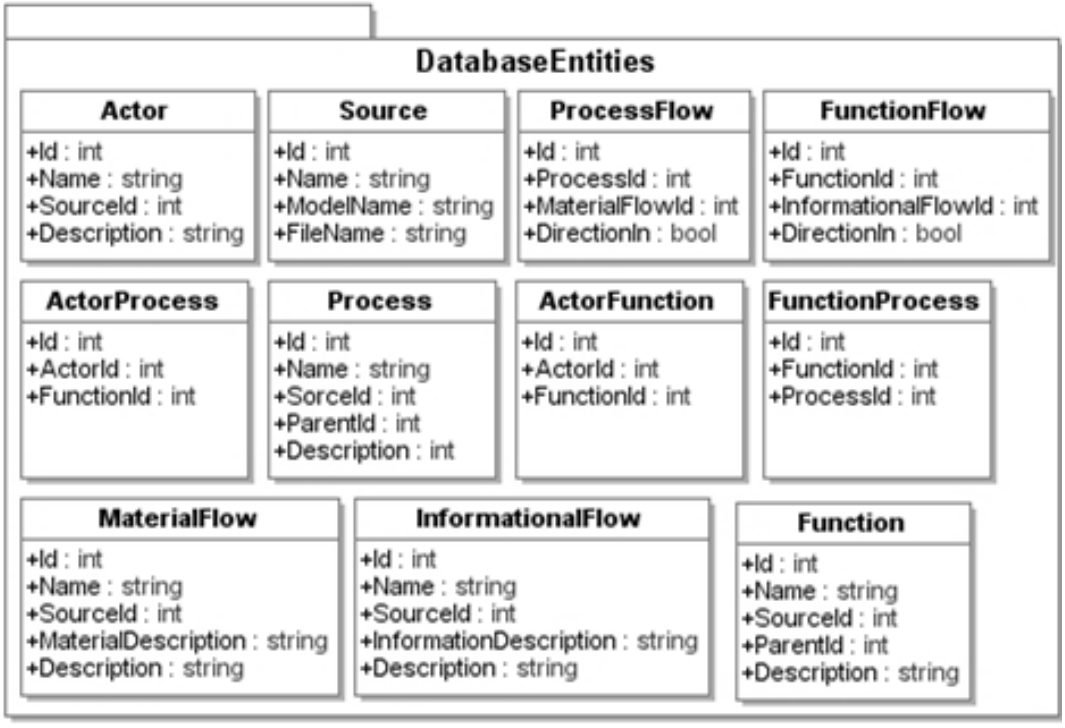
Lentelė 19. Veiklos modelio esybių modulio elementai

Elementas	Aprašymas
EnterpriseModelObjectInfo	Bazinė klasė iš kurios pasiveldi visos kitos *EnterpriseModelEntities* klasės.
EnterpriseModelSourceInfo	Klasė, kuri skirta saugoti veiklos modelio kilmės informaciją, tokią kaip pradinio failo pavadinimas, UML modelio pavadinimas.

EnterpriseModelFunction, EnterpriseModelActor, EnterpriseModelFlow, EnterpriseModelProcess, EnterpriseModelMaterialFlow, EnterpriseModelInformationalFlow	Klasės, kurios pasiveldi *EnterpriseModelEntities* klasė. Šio klasės saugo konkretaus veiklos modelio elementų informaciją programos atmintyje.

3.1.8 Duomenų bazės esybių modulis (Database Entities)

Duomenų bazės esybių modulis (angl. *„DatabaseEntities“)* yra sudarytas iš aibės klasių, kurios yra sugeneruotos ORM (angl. *„Object Relationship Mapper*“) įrankio (konkrečiu atveju .Net *Entity Framework*). Kiekviena duomenų bazės lentelė turi jai sukurtą duomenų sluoksnio klasę ir kitas sudėtines klases (pvz. atvaizdų (angl. *„View“*)). Šios klasės yra naudojamos kaip tarpininkas tarp logikos sluoksnio klasių ir duomenų bazės lentelių. Paveiksle žemiau yra pateikta šio modulio sudėtis ir ryšiai, o lentelėje detalizuoti elementai.



Pav. 36. Duomenų bazės esybių pagrindiniai elementai

Lentelė 20. Duomenų bazės esybių modulio elementai

Elementas	Aprašymas
Actor, Function, Process, FunctionProcess, ActorFunction, ActorProcess	Automatiškai ORM įrankio sugeneruotos klasės. Šios klasės realizuoja duomenų bazės objektus, programos logikos sluoksnyje. Klasės turi tokius duomenų bazei būdingus laukus kaip *pirminis raktas(primary key).*

3.1.9 Konstantų modulis (Constants)

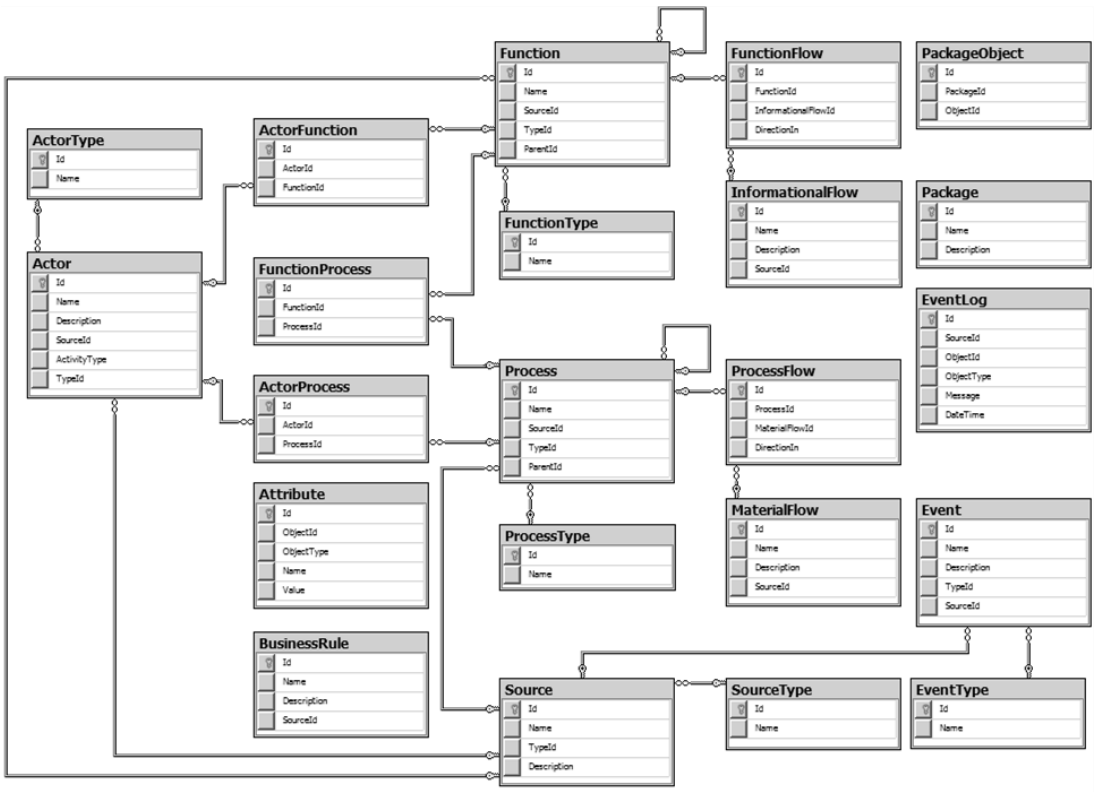
Konstantų modulis (angl. *„Constants“*) yra sudarytas iš klasių, kurios saugo statinę dalykinės programos prototipo informaciją, tokią kaip klaidų kodai, žinučių tekstai, pasirinkimo sąrašai (angl. *„Enums“*) ir pan. Duomenys yra saugomi kaip konstantos tipo laukai (angl. *„Fields/Members“*). Šį modulį naudoja logikos sluoksnio klasės bei esybių moduliai tokiems klasių laukams kaip objekto arba šaltinio tipas bei vartotojo sąsajos sluoksnis pranešimų formavimui.

3.1.10 Testavimo Modulis (UnitTesting)

Testavimo modulis (angl. *„UnitTesting“*) yra sudarytas iš klasių, kurios yra skirtos tik testavimui. Šiame modulyje yra sukuriami automatizuoti testavimo atvejai (angl. *„Test Cases*), kurie yra naudojami siekiant užtikrinti dalykinės programos prototipo architektūros vientisumą ir veikimo teisingumą, plėtojant prototipą. Testai yra rašomi visiems logikos sluoksnio elementams ir funkcijoms (UML modelių nuskaitymui, tikrinimui, konvertavimui, įrašymui į duomenų bazę).

3.1.11 Duomenų bazės ER schema ir aprašymas

Dalykinės programos prototipo veiklos modelio duomenų bazę sudaro 20 lentelių, kuriose saugomi duomenys apie veiklos modelio elementus ir ryšius tarp jų. Duomenų bazė yra sukurta naudojant *Microsoft SQL Server Express* duomenų bazės serverį. Duomenų bazės esybių- ryšių diagrama yra pavaizduota žemiau.



Pav. 37. Duomenų bazės esybių-ryšių diagrama

Lentelė 21. Duomenų bazės lentelių aprašymas

Lentelė	Aprašymas
Actor	Lentelė yra skirta saugoti informaciją apie veiklos modelio aktorius.
ActorType	Lentelė yra skirta saugoti veiklos modelio aktorių tipus.
ActorFunction	Lentelė yra skirta saugoti ryšius tarp veiklos modelio aktorių ir funkcijų.
ActorProcess	Lentelė yra skirta saugoti ryšius tarp veiklos modelio aktorių ir procesų.
Function	Lentelė yra skirta saugoti informaciją apie veiklos modelio funkcijas.
FunctionType	Lentelė yra skirta saugoti veiklos modelio funkcijų tipus.
FunctionProcess	Lentelė yra skirta saugoti ryšius tarp veiklos modelio funkcijų ir procesų.
FunctionFlow	Lentelė yra skirta saugoti ryšius tarp veiklos modelio funkcijų ir srautų.
InformationalFlow	Lentelė yra skirta saugoti informaciją apie veiklos modelio informacinius srautus.
Process	Lentelė yra skirta saugoti informaciją apie veiklos modelio procesus.
ProcessType	Lentelė yra skirta saugoti veiklos modelio procesų tipus.
ProcessFlow	Lentelė yra skirta saugoti ryšius tarp veiklos modelio procesų ir srautų.
MaterialFlow	Lentelė yra skirta saugoti informaciją apie veiklos modelio materialius srautus.
PackageObject	Lentelė yra skirta saugoti informaciją apie veiklos modelio elementų priklausymą konkrečiam grupavimo elementui.
Package	Lentelė yra skirta saugoti informaciją apie veiklos modelio grupavimo elementus
EventLog	Lentelė yra skirta saugoti aplikacijos įvykius.

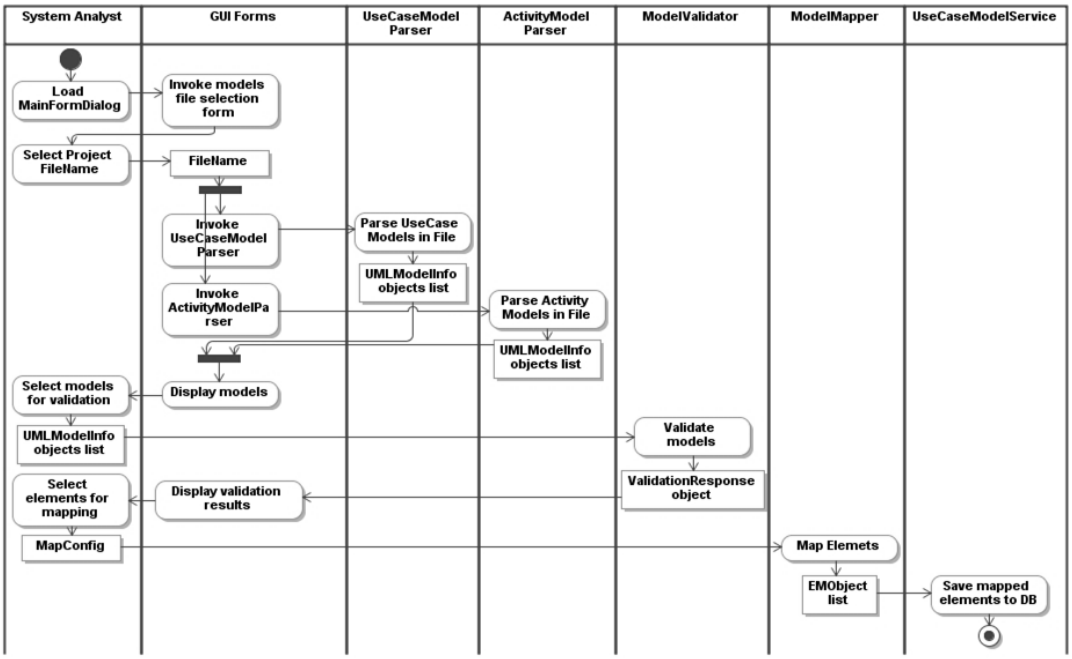
Event	Lentelė yra skirta saugoti informaciją apie veiklos modelio įvykius.
EventType	Lentelė yra skirta saugoti veiklos modelio įvykių tipus.
Source	Lentelė yra skirta saugoti veiklos modelio elementų šaltinių informaciją.
SourceType	Lentelė yra skirta saugoti veiklos modelio elementų šaltinių tipus.

Duomenų bazėje nėra naudojamos procedūros ir funkcijos. Šių elementų funkcijas atlieka *Entity Framework* klasės bei *LINQ* bibliotekos funkcijos ir procedūros.

3.2 Žiniomis grindžiamo MDA metodo įrankio prototipo funkcionalumas

3.2.1 Proceso sekų diagrama ir aprašymas

Standartinis žiniomis grindžiamo MDA metodo procesas yra pradedamas dalykinės srities CIM modelio sudarymu. Sistemų analitikas naudodamasis *MagicDraw* UML modeliavimo paketu sukuria antrame skyriuje aprašytus SysML (UML) modelius ir išsaugo juos XMI formatu. Eksperimentinėje dalyje aprašytas dalykinės programos prototipas gali nuskaityti *panaudos atvejų* ir *veiklų* modelius iš XMI bylų, todėl veiksmai su šiais modeliais gali būti automatizuoti. Sistemų analitikas pagrindiniame dalykinės programos prototipo meniu turi pasirinkti modelių importavimo langą. Šiame lange yra nurodoma XMI byla, kurią norima nuskaityti. Nuskaičius pasirinktą XMI bylą, dalykinės programos prototipas pateikiami visų nuskaitytų modelių tipus ir pavadinimus. Pasirinkus vieną konkretų modelį, galima peržiūrėti jo elementų sąrašą. Kitas etapas yra modelių tikrinimas. Šiame etape sistemų analitikas parenkama pagrindinį ir veidrodinį modelius. Pagrindinis modelis yra laikomas etalonu ir jo atžvilgiu yra atliekamas nuoseklumo tikrinimas su veidrodiniu modelius. Tikrinimo rezultatai parodomi atskiroje formoje. Jeigu tikrinimo metu nebuvo nustatyta klaidų, yra aktyvuojama modelių konvertavimo funkcija. SysML (UML) modelių elementai yra transformuojami į veiklos modelio elementus, kurie vėliau yra panaudojami veiklos modelio analizėje. Galutiniame etape sukurti veiklos modelio elementai yra įrašomi į duomenų bazę. Žemiau pateikta viso proceso veiklos diagrama.

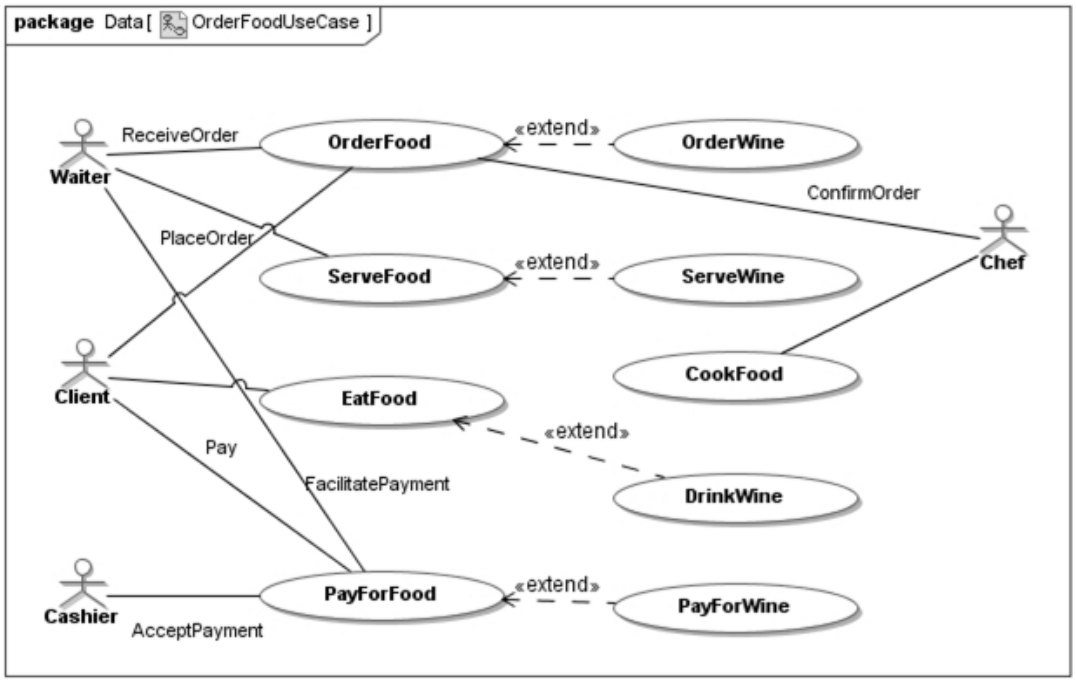


Pav. 38. Žiniomis grindžiamo MDA metodo dalykinės programos prototipo naudojimo diagrama

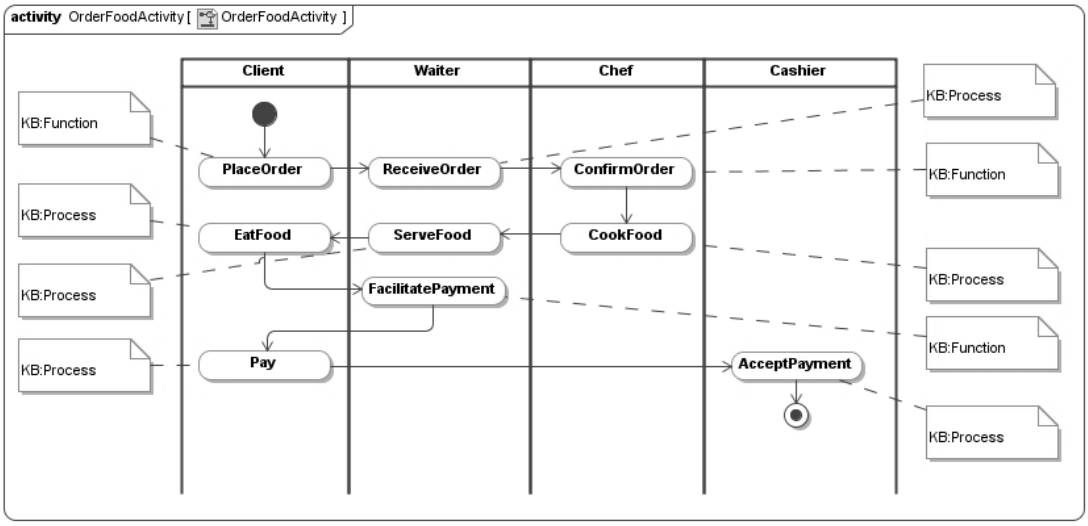
Pateiktoje diagramoje yra pavaizduota žiniomis grindžiamo MDA metodo proceso dalis, kurioje buvo panaudotas dalykinės programos prototipas. Aprašyti veiksmai apima CIM vidinių modelių nuoseklumo tikrinimą, jų transformavimą į veiklos modelį bei veiklos modelio elementų išsaugojimą duomenų bazėje.

3.2.2 Žiniomis grindžiamo MDA prototipo veiklos pavyzdys

Šiame skyriuje pateiksime žiniomis grindžiamo MDA metodo dalykinės programos prototipo panaudojimo pavyzdį konkrečiam atvejui. Aprašoma veikla yra patiekalo užsakymas restorane. Joje dalyvauja keturi aktoriai: *Waiter, Client, Cashier, Chef*. Aktoriai ir panaudos atvejai yra speficikuojami UML *panaudos atvejų* modelyje - *OrderFoodUseCase*. *Panaudos atvejų* modeliui detalizuoti yra sukuriamas UML *veiklų modelis* - *OrderFoodActivity*, kuriame yra pateikiama veiksmų eiliškumo seka be to veiksmai yra anotuojami papildoma žyme, kuri nusako veiksmo tipą (funkcija ar procesas). Abu modeliai yra sukuriami ir išsaugomi vienoje MagicDraw projekto byloje XMI formatu. Sukurti dalykinės srities modeliai yra pateikti paveiksle žemiau.



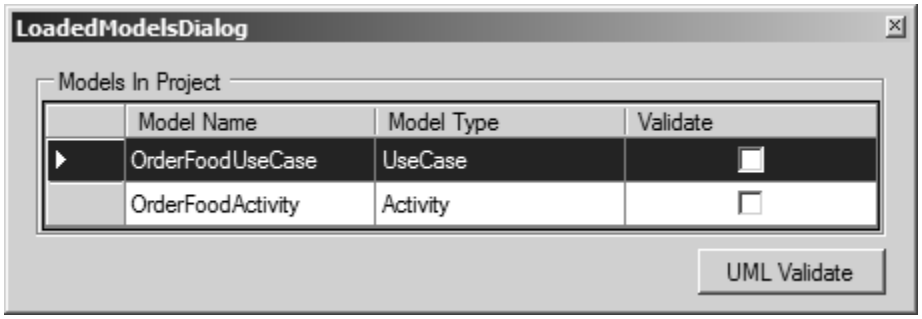
Pav. 39. Pavyzdinė maisto užsakymo restorane panaudos atvejų diagrama



Pav. 40. Pavyzdinė maisto užsakymo restorane veiklų diagrama

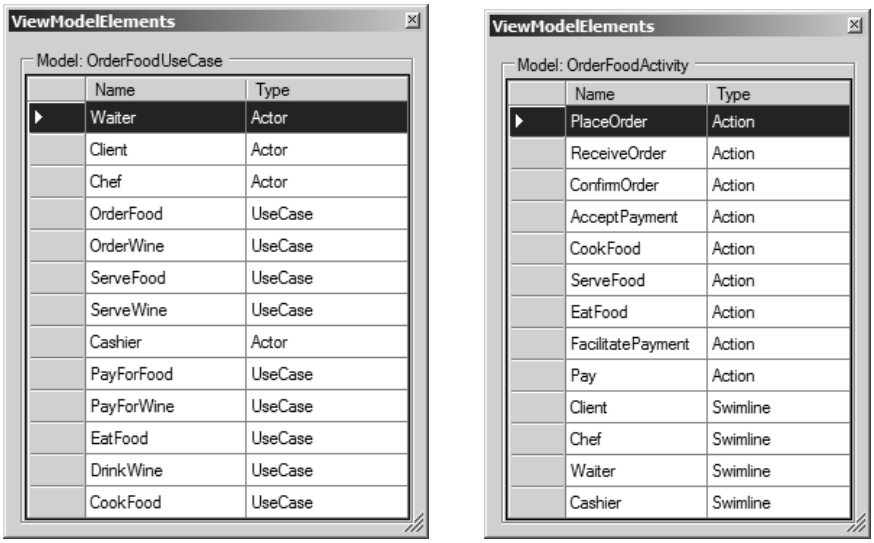
Projektinė XMI formato byla yra nuskaitoma žiniomis grindžiamo MDA metodo dalykinės programos prototipo pagalba. Nuskaitytiems *panaudos atvejų* ir *veiklų modelio* objektams yra sukuriami atitinkami loginio sluoksnio objektai (UML modelių esybių klasių pagrindu). Prototipe yra sukuriami du UML modelių objektai (tipas *UMLModelInfo*) su pavadinimais *OrderFoodUseCase* ir *OrderFoodActivity*. Šie objektai turi tuos pačius elementus (aktorius, panaudos atvejus, grupavimo elementus, veiksmus) kaip ir objektai išsaugoti XMI byloje t.y. duomenys buvo perkelti iš XMI failo į

veikiančios programos atmintį. Nuskaitytų modelių sąrašas parodomas prototipo lange lentelės pavidalu (žr. pav. 41):



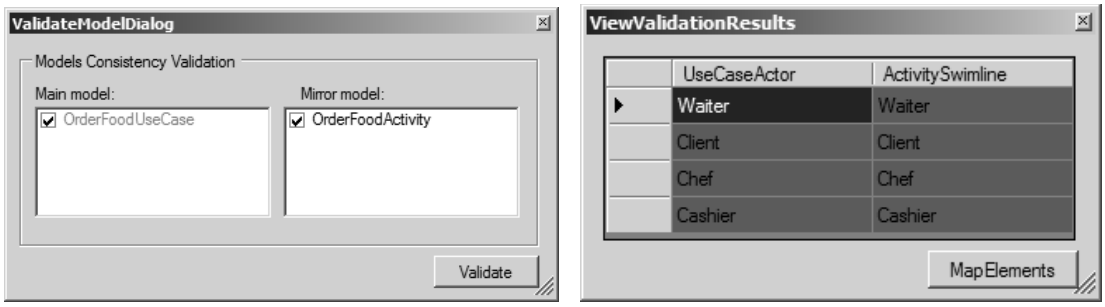
Pav. 41. Nuskaitytų UML modelių sąrašas

Kiekvienam *panaudos atvejų* modelio *aktoriaus* tipo elementui yra sukuriamas ir užpildomas duomenimis *UMLActor* objektas. Kiekvienam *panaudos atvejų* modelio *panaudos atvejo* tipo elementui yra sukuriamas *UMLUseCase* objektas. Atitinkamai kiekvienam *veiklų* modelio *grupavimo elementui* yra sukuriamas *UMLSwimline* objektas ir kiekvienam *veiksmui* yra sukuriamas *UMLAction* objektas. Nuskaitytiems *Note* elementams yra sukuriami *UMLNote* objektai. Šie objektai programoje susiejami su konkrečiu UML elementu, kuriam jie yra priskirti. Konkrečiu atveju *UMLNote* elementai yra naudojamas *ModelMapper* objekto tam, kad nustatyti veiksmo (*Action*) objekto tipą (funkcija ar procesas). Nuskaitytų *OrderFoodUseCase* ir *OrderFoodActivity* modelių elementai atvaizduoti prototipo lange.



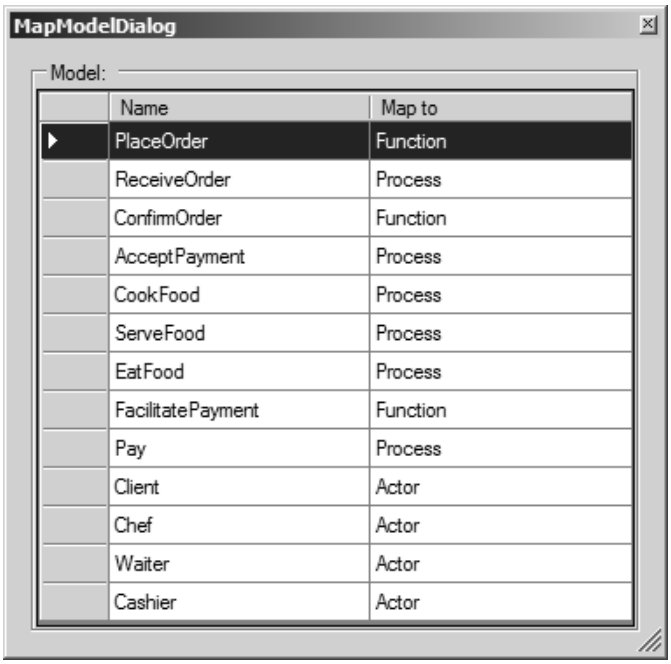
Pav. 42. Nuskaityto UML modelio elementų sąrašas

Kitas etapas, kuris atliekamas naudojantis dalykinės programos prototipo funkcijomis yra modelių nuoseklumo tikrinimas. Tikrinimo rezultatai pateikiami lentelėje, susieti laukai yra pažymimi viena spalva, trūkstami - kita. Jeigu yra neteisingų laukų, UML objektų konvertavimas į veikos modelio objektus yra negalimas. Prototipe yra realizuotas modelių tarpusavio tikrinimas tarp panaudos atvejų ir veiklų modelių, tačiau tikrinimo modulio architektūra yra sukurta tokia būdu, jog būtų įmanoma šį modulį būtų įmanoma išplėsti kitiems UML modeliams skirtais tikrinimo algoritmais. Žemiau pateiktas tikrinimo rezultatų langas.



Pav. 43. Prototipo modelių tikrinimo parametrų ir rezultatų langai

Paskutinis prototipo panaudojimo funkcionalumas yra modelių konvertavimas. Prototipe realizuotas modelio elementų konvertavimas yra atliekamas remiantis *UMLNote* objekto pateikta informacija.



Pav. 44. Modelių susiejimo rezultatų langas

Jeigu UML diagramoje elementas neturi jam priskirto *Note* su konvertavimo nurodymais, vartotojui pateikiamas pranešimas apie trūkstamą informaciją.

3.3 Išvados

Sukurtas prototipas yra skirtas dalinai realizuoti teorinėje dalyje aprašytus žiniomis grindžiamo MDA metodo proceso etapus. Prototipas gali atlikti veiksmus su panaudos atvejų („*Use Case*“) ir veiklų („*Activity*“) modeliais (nuskaityti, tikrinti, transformuoti, išsaugoti žinių duomenų bazėje). Aprašytas prototipo panaudojimo pavyzdys pademonstruoja modelių nuskaitymo iš XMI formato bylų galimybes, modelių nuoseklumo tikrinimą ir modelių elementų transformavimą į veiklos modelio elementus.

Prototipas remiasi trijų sluoksnių architektūros koncepcija (duomenų sluoksnis, logikos sluoksnis ir vartotojo sąsajos sluoksnis). Duomenų sluoksniui realizuoti buvo naudojama *MS SQL Server Express* duomenų bazių valdymo sistema. Logikos ir vartotojo sąsajos sluoksnis realizuoti naudojant *MS .Net Framework* karkasą. Prototipo architektūra buvo projektuota su orientacija į išplėtimo galimybes: naudojamos sąsajos ir projektavimo šablonai (*„Interface“* ir *„Design Pattern“*).

Skyriuje detaliai aprašyta: prototipą sudarantys moduliai ir jų tarpusavio ryšiai, vidinė modulių sudėtis (klasės, sąsajos) ir ryšiai tarp elementų, duomenų bazės sudėtis, prototipo panaudojimo pavyzdys. Šiame skyriuje aprašomi rezultatai yra publikuoti straipsnyje „SysML and UML models usage in Knowledge Based MDA process“ mokslo žurnale „Electronics and Electrical Engineering“ (žr. žurnalų publikacijų sąrašą nr.: 1).

4 PAŠTO SIUNTŲ STEBĖJIMO PROGRAMĖLĖS PROTOTIPAS

4.1 Atskiro atvejo (angl. *„Case Study“*) tyrimas

Siekiant įvertinti žiniomis grindžiamo MDA metodo efektyvumą (lyginant su tradiciniai IS inžinerijos metodais) buvo nuspręsta panaudoti šį metodą bei metodo dalykinės programos prototipą, kuriant „realaus gyvenimo“ (angl. *„real life“*) programą. Konkrečiu atveju buvo pasirinkta sukurti pašto siuntų stebėjimo programėlę mobiliesiems įrenginiams. Metodo efektyvumas buvo vertinamas ekspertiškai, remiantis programinės įrangos kūrimo metodų efektyvumo kriterijais (produkto pristatymo trukmė, programinio kodo kokybė bei programinio kodo klaidų kiekis [84]).

4.2 Mobiliems įrenginiams skirtos PĮ kūrimo specifika ir metodai

Kurti programinę įrangą vieno tipo mobilaus įrenginio platformai tikslinga naudojant tradicines PĮ kūrimo metodologijas. Tačiau dažniausiai programėlės mobiliems įrenginiams yra kuriamos bent dviem skirtingom platformom (pvz. *„Android“ ir „iOS“*). Taigi mobiliesiems įrenginiams skirtų programėlių ypatumas yra tas, jog šios programėlės turi būti pritaikytos veikti keliose skirtingose operacinėse sistemose. 2014 metų duomenimis pagrindinės mobiliųjų įrenginių operacinės sistemos (platformos) yra *„Android“*, *„iOS“*, *„Windows Phone“*. Minėtos trys operacinės sistemos užima ~95% mobiliųjų įrenginių rinkos [62], todėl organizacijoms siekiant pasiekti maksimalų vartotojų skaičių būtina kurti programėles šioms dominuojančioms operacinėms sistemoms. Minėtoms platformos programėlės yra kuriamos skirtingomis programavimo kalbomis (*„Android“* platformai programėlės yra kuriamos *„Java“* kalba, *„Windows Phone“* naudojant - *.Net* karkasą ir jo palaikomas kalbas (pvz. *C#*), *„iOS“* yra naudojama *Objective-C* programavimo kalba). Visos kalbos yra objektiškai orientuotos, tačiau jų sintaksė ir semantika yra skirtinga, taigi programuotojas mokantis vieną iš kalbų negali greitai ir efektyviai persiorientuoti prie kitos kalbos (pvz. *Java* kalboje egzistuoja daugialypis paveldėjimas, o *C#* jis negalimas, *Objective-C* kintamieji skirstomi į mutable ir immutable, kas su tam tikrais skirtumais

galėtų atitikt *Java „read only“* ar C# konstantas (arba *readonly* kintamuosius)). Taigi pagrindinis uždavinys kuriant programėles skirtingoms mobiliųjų įrenginių operacinėms sistemoms yra išlaikyti tą patį funkcionalumą ir vartotojo sąsają (*feel and look*), tačiau tą pasiekti reikia naudojant skirtingas programavimo kalbas. Idealiu atveju programėlės turėtų būti kuriamos aukštesnio lygio programavimo kalba nei minėtos trečios kartos (3G) programavimo kalbos t.y. kalba kuri yra nepriklausoma nuo platformos. Šį reikalavimą iš esmės atitinka MDA pagrindu sukurti programinės įrangos kūrimo metodai. MDA atveju programinės įrangos reikalavimai yra specifikuojami nepriklausomai nuo konkrečios platformos, jų pagrindu yra sukuriamas nuo platformos nepriklausomas architektūrinis modelis, o vėliau transformavimo dėka šis modelis yra pritaikomas konkrečiai operacinei sistemai. Būtina sąlyga, kad metodas būtų efektyvus – užtikrinti surinktų reikalavimų kokybę, kadangi klaidos ar netikslumai reikalavimuose įtakos ne vieną programėlę, o visas programėles, kurios yra kuriamos skirtingoms operacinėms sistemoms. Taigi galima teigti, jog kuriant programinę įrangą, skirtą kelioms mobiliųjų įrenginių platformoms reikalingas efektyvus reikalavimų valdymas ir tikrinimas. Šią užduotį gali atlikti žiniomis grindžiami programinės įrangos kūrimo metodai, savo ruožtu formalizuojant reikalavimus ir sistemos architektūrą gali būti panaudojami MDA koncepcija grindžiami metodai. Abu kriterijus atitinka žiniomis grindžiamas MDA metodas. Siekiant patikrinti šio metodo taikymo galimybes realiems uždaviniams spręsti, buvo nuspręsta sukurti mobiliems įrenginiams skirtą programėlę, vadovaujantis siūlomo metodo principais.

4.3 Pašto siuntų stebėjimo programėlės kūrimo įrankiai

Pašto siuntų stebėjimo programėlės prototipas buvo kuriama naudojant „*Eclipse“* programavimo aplinką (*IDE*). Programėlės nustatymas ir duomenims saugoti buvo panaudotos XML bylos mobilaus įrenginio vidinėje atmintyje. Darbui su XML bylomis buvo naudota biblioteka *simple-xml-2.6.9.jar* [80]. Programėlė naudoja HTTP GET protokolo užklausas duomenų iš interneto puslapių parsiuntimui. Internetiniai puslapiai yra nuskaitomi

naudojantis *jsoup-1.7.1.jar* [40] biblioteka. Programėlės reikalavimai buvo aprašyti *panaudos atvejų, veiklų* bei *reikalavimų* modeliais, programėlės architektūros specifikuoti buvo panaudoti *struktūros aprašų* modeliai. Panaudos atvejų bei veiklų modeliai buvo tikrinama naudojant žiniomis grindžiamo MDA metodo dalykinės programos prototipą ir jo modelių tikrinimo funkcijas. Programėlė buvo testuota naudojant Android versijas (2.x, 4.x) bei šiuos mobilius įrenginius: *„AsusTransfomer TF300TG“* su Android versija 4.0, *„Samsung GT-S5660“* su Android versija 2.3.4, *„LG P700“* su Android versija 4.0.3. Testavimui naudojami realių siuntų duomenys iš Lietuvos, Honkongo, Jungtinių Valstijų bei Kinijos. Detalūs programėlės reikalavimai, architektūros aprašymas, veiklos pavyzdžiai bei programinis kodas yra pateiktas kituose skyriuose.

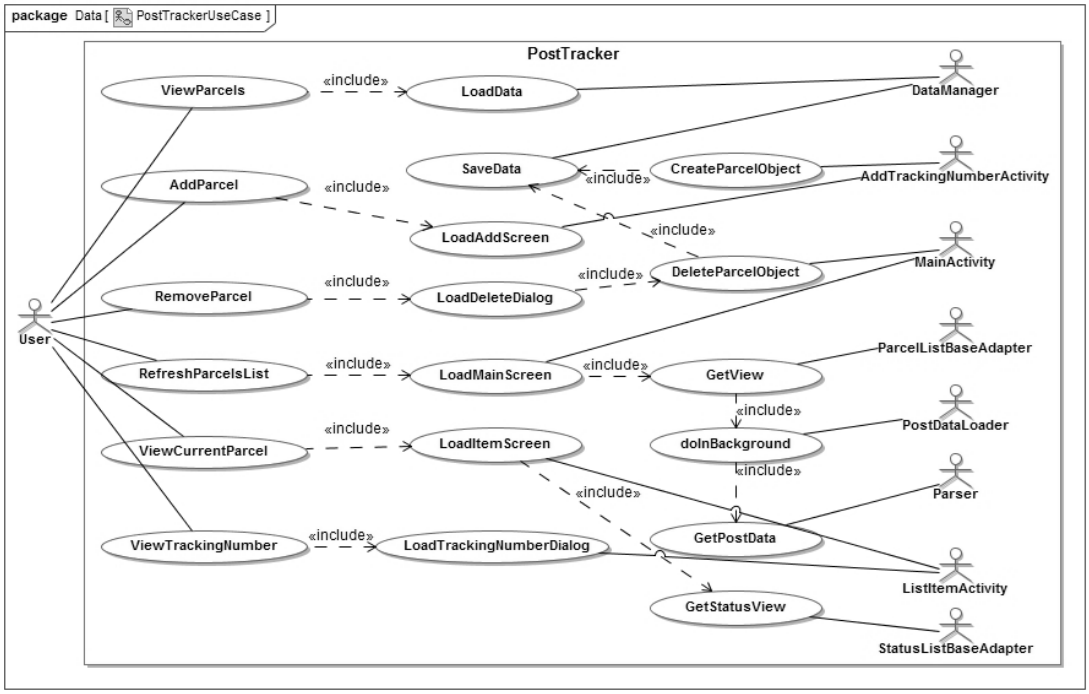
4.4 Pašto siuntų stebėjimo programėlės reikalavimai

Pašto siuntų programėlės reikalavimai buvo suskirstyti į funkcinius ir nefunkcinius. Funkciniams reikalavimams specifikuoti buvo naudojami UML *panaudos atvejų* ir *veiklų* modeliai, nefunkciniams reikalavimams - UML *reikalavimų* modelis, o detalizuoti sistemos struktūrą buvo panaudotas *struktūros aprašų* modelis. Pagrindinė programėlės užduotis - nuskaityti siuntos duomenis iš Lietuvos pašto sistemos ir pateikti juos vartotojui. Vartotojas įveda siuntos numerį, programėlė suformuoja GET užklausą į Lietuvos pašto interneto svetainę, gautas atsakymas yra nuskaitomas (atrenkami duomenys apie siuntą)) bei parodomi vartotojui. Papildomas reikalavimas buvo sukurti lengvai išplečiamą siuntėjo pašto sistemų duomenų nuskaitymo architektūrą ir pateikti šios architektūros naudojimo pavyzdį. Tai buvo padaryta su Honkongo bei Jungtinių Valstijų (USPS) pašto sistemomis.

4.4.1 Funkciniai reikalavimai

Panaudos atvejų modelyje yra specifikuoti 9 aktoriai. Vartotojas (angl. „*User*“) sąveikauja su sistema (*„PostTracker“*) tam kad pasiektų savo tikslus. Vidiniai sistemos aktoriai yra skirti specifikuoti pagrindines sistemos klases.

Šie aktoriai turi savo atitikmenis *veiklų* bei *struktūros aprašų* modeliuose. Panaudos atvejų modelis yra pateiktas paveiksle 45.



Pav. 45. Pašto siuntų stebėjimo programėlės reikalavimai aprašyti panaudos atvejų diagrama

Vartotojas dalyvauja atliekant 6 panaudos atvejus: *sukurti naują siuntą*, *pašalinti siuntą*, *peržiūrėti siuntas*, *atnaujinti siuntų sąrašą*, *peržiūrėti siuntą*, *peržiūrėti siuntos numerį*. Panaudos atvejai detalizuoti lentelėje 22

Lentelė 22. Programėlės funkcijos, kurias inicijuoja vartotojas

Pavadinimas	Aprašymas
AddParcel (Sukurti naują siuntą)	Vartotojas turi galėti sukurti naują siuntą. Vartotojas paspaudžia meniu paveikslėlį ir yra nukreipiamas į naujos siuntos sukūrimo langą. Vartotojas turi įvesti siuntos pavadinimą ir siuntos numerį. Vartotojas paspaudžia išsaugoti mygtuką ir yra grąžinamas į pradinį langą.
RemoveParcel (Pašalinti siuntą)	Vartotojas turi galėti pašalinti siuntą. Vartotojas ilgai paspaudžia konkrečios siuntos objektą. Pasirodo kontekstinis meniu su vienu punktu: ištrinti siuntą. Paspaudus šį punktą siuntą yra ištrinama ir atnaujinamas siuntų sąrašas pagrindiniame lange.
ViewParcels (Peržiūrėti siuntas)	Vartotojas turi turėti galimybę peržiūrėti visas egzistuojančius siuntas. Vartotojas įsijungia programėlę ir pagrindiniame lange gali matyti egzistuojančias siuntas. Sąraše prie kiekvieno siuntos objekto turi būti rodomas siuntos pavadinimas, siuntos numeris, paskutinė atnaujinta būsena.
RefreshParcelsList (Atnaujinti siuntų sąrašą)	Vartotojas turi galėti atnaujinti siuntų būsenas. Vartotojas paspaudžia menu punkto „Atnaujinti“

	paveikslėlį. Siuntų būsenos pradedamos asinchroniškai atnaujinti. Proceso metu siuntos būsena būną pakeista į „Atnaujinama“. Sąrašas atnaujinamas asinchroniškai ir atvaizduojamas vartotojui.
ViewCurrentParcel (Peržiūrėti siuntą)	Vartotojas turi turėti galimybę peržiūrėti detalią siuntos informaciją. Vartotojas trumpai paspaudžia siuntos įrašą pagrindiniame lange. Atidaromas siuntos detalių langas. Lango viršuje yra pateikiamas siuntos pavadinimas ir siuntos numeris. Pagrindinė lango dalis „Tab“ elementų pagalba yra padalinta į dvi dalis. Vienoje dalyje yra rodomas informacija gauta iš Lietuvos pašto sistemos, kitoje informacija iš siuntėjo šalies pašto sistemos. Vartotojas gali pasirinkti kurią informaciją peržiūrėti. Jeigu informacijos nėra, vartotojui turi būti rodomi tušti sąrašai.
ViewTrackingNumber (Peržiūrėti siuntos numerį)	Vartotojas turi turėti galimybę padidintu šriftu matyti pasirinktos siuntos numerį. Vartotojas paspaudžia meniu punktą „peržiūrėti siuntinį“ Siuntos peržiūros lange pasirenka meniu punktą padidinti. Vartotojui parodomas kontekstinius meniu su didesnio šrifto siuntos numeriu. Šriftas turi būti lengvai įskaitomas, nes šis funkcionalumas bus naudojamas nurodant siuntos numerį pašto darbuotojui, atsiimant siuntą.

PostTracker vidiniai aktoriai reprezentuoja pagrindines programėlės klases: DataManager, AddTrackingNumberActivity, MainActivity, ParcelListBaseAdapter, PostDataLoader, Parser, ListItemActivity, StatusListBaseAdapter. Detalizuotas aktorių aprašymas pateiktas lentelėje 23:

Lentelė 23. Pašto siuntų stebėjimo programėlės aktoriai ir funkcijos

Aktorius	Funkcija
DataManager	Duomenų nuskaitymas ir įrašymas. Duomenys saugomi XML formatu vidinėje įrenginio atmintyje.
AddTrackingNumberActivity	Naujos siuntos lango atvaizdavimas, siuntos objekto sukūrimas, DataManager iškvietimas
MainActivity	Pagrindinio programėlės lango atvaizdavimas, DataManager iškvietimas, asinchroninio duomenų nuskaitymo inicializavimas, siuntos trynimas, navigacijos į kitus langus valdymas
ParcelListBaseAdapter	Siuntų sąrašo atnaujinimas gavus asinchroniškai nuskaitytus duomenis.
PostDataLoader	Atlieka asinchroninį duomenų nuskaitymą iš pašto sistemų.
Parser	Pašto sistemų duomenų nuskaitymas
ListItemActivity	Atvaizduoja konkrečios siuntos duomenis, iškviečia siuntos numerio dialogą
StatusListBaseAdapter	Siuntų būsenų atvaizdavimas ListView elemente.

Panaudos atvejų diagramoje specifikuoti aktoriai ir panaudos atvejai yra žemo abstrakcijos lygmens t.y. reprezentuoja ne abstrakčius sistemoje veikiančius

elementus, bet konkrečias klases. Aukštas abstrakcijos lygmuo nesudaro sąlygų/apsunkina galimybę atlikti nuoseklumo tikrinimą tiesiogiai su struktūros aprašų modeliu.

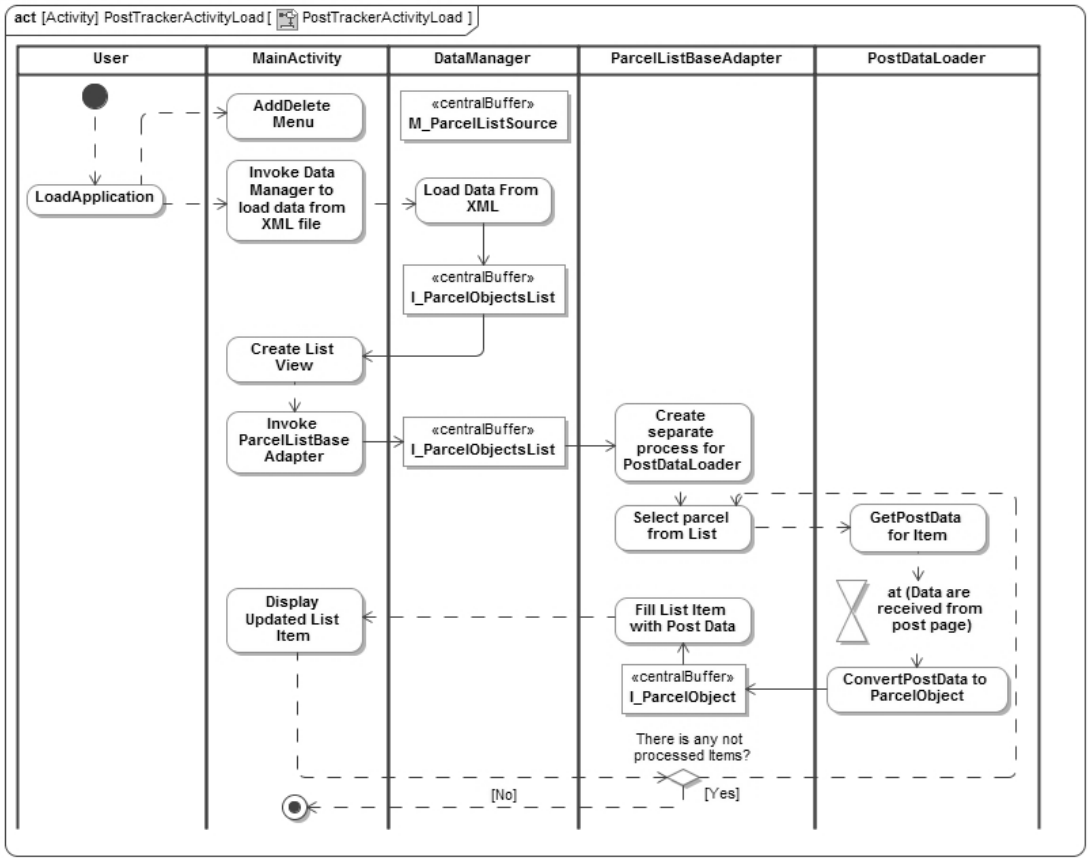
4.4.2 Pašto siuntų stebėjimo programėlės elgsenos aprašymas

Šiame skyriuje naudojant UML veiklų modelius detalizuojami du panaudos atvejai: *LoadMainScreen* (pagrindinio lango užkrovimas) ir *AddParcel* (naujos siuntos įkėlimas). Veiklų modeliuose procesai yra specifikuojami techniniame lygmenyje, atlikus modelių sudarymą atliekamas nuoseklumo tikrinimas panaudojant žiniomis grindžiamo MDA metodo dalykinės programos prototipą. Konkrečiu atveju buvo tikrinamas aktorių ir grupavimo elementų suderinamumas.

4.4.2.1 Pagrindinio lango rodymas

Pagrindinio programėlės lango rodymą inicijuoja *vartotojas*, mobiliajame įrenginyje paspausdamas programėlę startuojančią ikoną ir tokiu būdu sukurdamas *MainActivity* procesą. *MainActivity* inicializuoja grafinį programėlės langą, sukuria kontekstinius ir paprastus meniu bei sukuria *DataManager* objektą, kuris nuskaito įrenginio atmintyje XML failuose išsaugotus vartotojo duomenis. Nuskaityti duomenys yra įrašomi į *ParcelObjectsList* objektą, kuris sudarytas iš *ParcelObject* objektų.

Pagrindinio lango iškvietimoveiklų modelis yra pateiktas paveiksle 46.

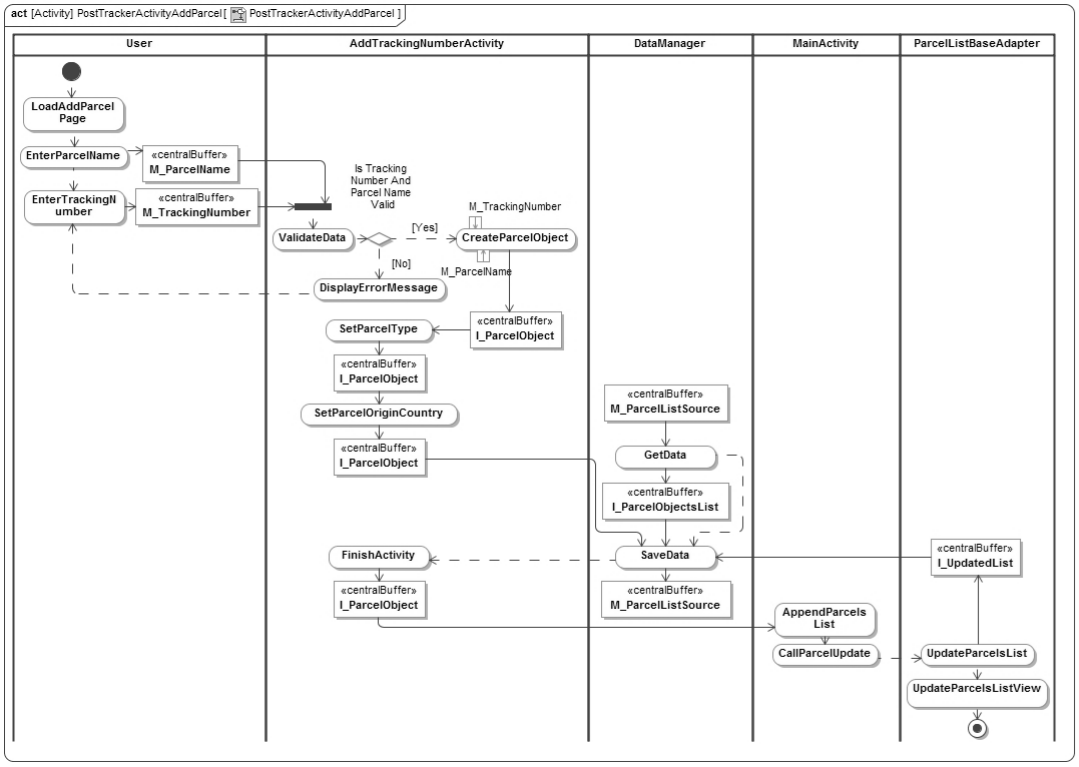


Pav. 46. Pagrindinio lango rodymo veiklų modelis

ParcelObjectsList objektas kaip parametras yra perduodamas *MainActivity* procesui, kuris naudodamas šiame sąraše aprašytų objektų laukus sukuria grafinio sąrašo (*ListView*) objektus. *ListView* užpildymas objektas yra atliekamas iškviečiant *ParcelListBaseAdapter*, kuris kiekvienam *ParcelObject* sukuria atskirą *PostLoader* objektą asinchroniniam duomenų nuskaitymui iš pašto sistemos. Nuskaitęs pašto sistemos duomenis, *PostLoader* objektas informuoja *ParcelListBaseAdapter* apie būsenos pasikeitimą ir perduoda rezultatus. *ParcelListBaseAdapter* objektas inicijuoja grafinį sąrašo atnaujinimą nauja informacija. Kadangi kiekvienam siuntos objektui (siuntiniui) yra sukurtas atskiras *PostDataLoader* objektas, siuntų duomenys nuskaitomi nepriklausomai vienas nuo kito ir nepriklausoma seka.

4.4.2.2 Naujos siuntos įkėlimas

Naujos siuntos įkėlimą iniciuoja vartotojas, pasirinkęs *„Pridėti naują siuntą“* meniu punktą ir tokiu būdu sukurdamas *AddTrackingNumberActivity* objektą. Atsidariusiame lange vartotojas įveda siuntos numerį ir pavadinimą. Abu parametrai perduodami duomenų tikrinimo metodui, kuris tikrina, ar siuntos numeris būtų sudarytas iš 13 simbolių bei kad siuntos pavadinimas nebūtų tuščias. Naujos siuntos įkėlimo veiklų modelio diagrama yra pateikta paveiksle 47.



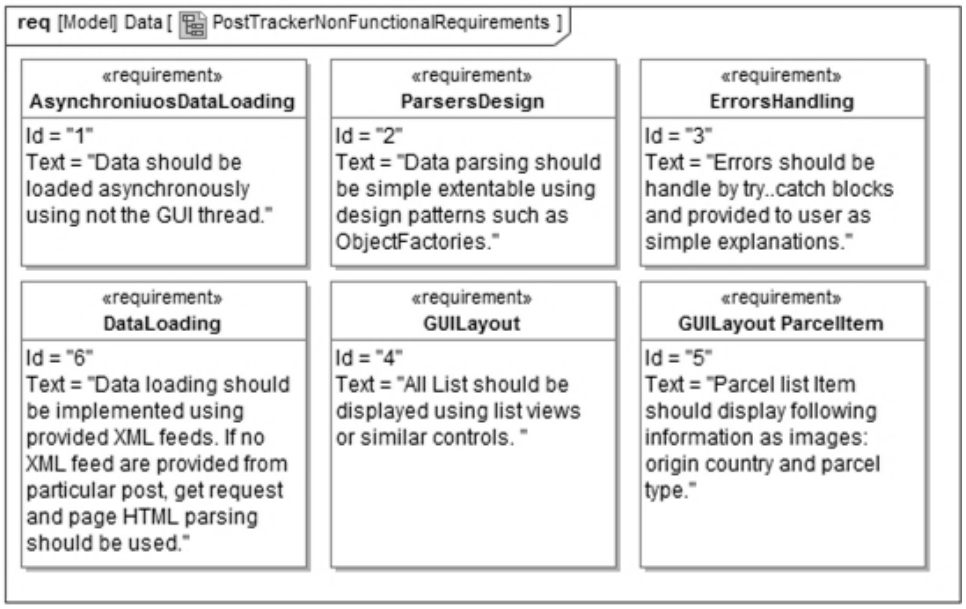
Pav. 47. Naujos siuntos duomenų įkėlimo veiklų modelis

Jeigu vartotojo įvesti duomenys yra su klaidomis, vartotojas apie tai informuojamas programėlės lange ir laukia, kol duomenys bus pakoreguoti. Jeigu duomenys yra teisingi, programėlė sukuria naują *ParcelObject* objektą, kuriam nustatomas tipas bei kilmės šalis pagal siuntos numerį. Sukūrus *ParcelObject* objektą yra iškviečiamas *DataManager* objektas, kuris nuskaito programėlės atmintyje esančias siuntas ir sukuria *PercelListObject* objektą. Šis objektas yra papildomas naujos siuntos objektu bei išsaugomas įrenginio atmintyje. Atnaujintas siuntų sąrašas (*PercelListObject objektas*) yra perduodamas *MainActivity* procesui, kuris inicijuoja siuntų statusų atnaujinimą

pagrindiniame programėlės lange, tokiu būdu įtraukiant naują siuntą atliekamas ir visų egzistuojančių siuntų būsenų atnaujinimas.

4.4.3 Nefunkciniai reikalavimai

Nefunkciniai reikalavimai apibūdina programėlės veikimo efektyvumo kriterijus, reikalavimus programėlės architektūrai bei reikalavimus grafinei vartotojo sąsajai. Jų sąrašas yra pateiktas žemiau.



Pav. 48. Pašto siuntų stebėjimo programėlės reikalavimų diagrama

- **Asinchroninis duomenų nuskaitymas**. Duomenų parsiuntimas iš pašto sistemų turi būti atliekamas asinchroniškai. Šis procesas yra neapibrėžtas laike (priklauso nuo įvairių faktorių tokių kaip: įrenginio tipas, bevielio tinklo sparta ir kt.), taip pat pakartotinai iškviečiamas jeigu pirminis kreipinys nebuvo įvykdytas, todėl šių veiksmų negalima atlikti pagrindiniame programėlės procese. Duomenų turi būti nuskaitomi sukuriant atskirus procesus bei apie proceso baigtį informuojant pagrindinį procesą per parametrus.
- **Programėlės praplečiamumas naujais pašto siuntų stebėjimo moduliais**. Pašto sistemų duomenų nuskaitymui naudojamos procedūros ir objektai turi būti lengvai išplečiami siekiant įtraukti papildomų pašto sistemų siuntų stebėjimo galimybes. Šiam tikslui pasiekti turi būti naudojami projektavimo šablonai (angl. *„Design*

Pattern“), sąsajomis (angl. *„Interface“*) grindžiamos klasės bei abstrakčios klasės (angl. *„Abstract Class“*).

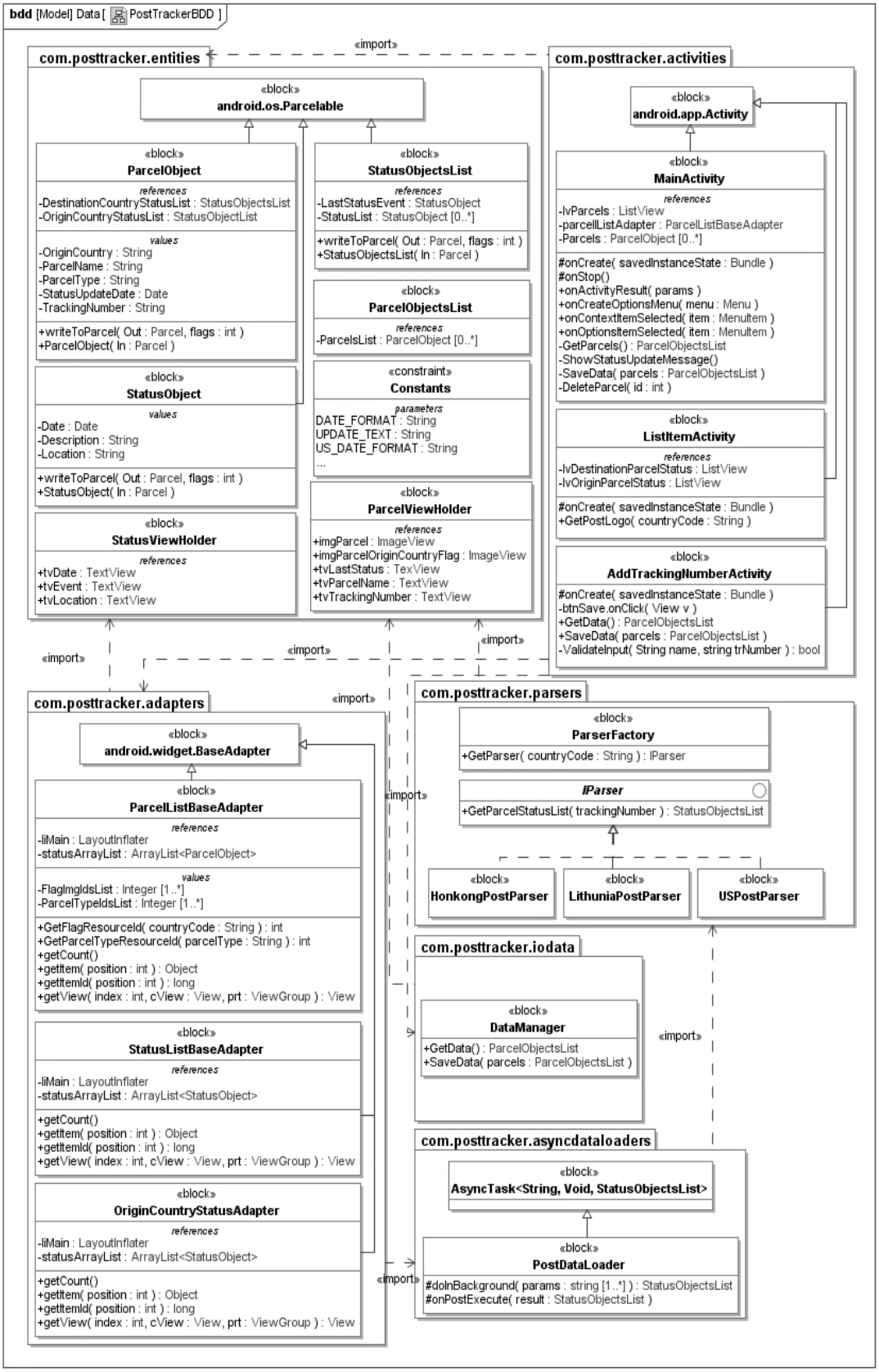
- **Klaidų valdymas.** Vartotojas neturi matyti techninių klaidų pranešimų. Visos klaidos turi būti užfiksuojamos panaudojant tam skirtas programavimo kalbos funkcijas (pvz. „*try...catch“* blokus), apdorojamos ir pateikiamos vartotojui suprantamu būdu. Vartotojas neturi būti informuojamas apie jam nereikšmingas klaidas.
- **Duomenų nuskaitymas.** Siuntų duomenų nuskaitymas turi būti atliekamas naudojantis pašto sistemų suteikiamais XML srautais (angl. *„XML feed*“). Jeigu pašto sistema nepateikia informacijos minėtais srautais, duomenys turi būti nuskaitomi iš pašto sistemų puslapių, nuskaitant HTML tekstą bei informaciją apie siuntos būseną. Užklausos turi būti siunčiamos GET arba POST protokolu priklausomai nuo konkrečios pašto sistemos.
- **Grafinė vartotojo sąsaja.** Grafinė vartotojo sąsaja turi būti paprasta ir minimalistinė. Visi sąrašai turi būti pateikiami naudojant sąrašo (angl. „*ListView*“) arba lentelės komponentus. Sąrašų skaidymui naudoti puslapiavimo (angl. *„Tab“*) komponentus. Siuntų sąraše kiekvienam siuntiniui turi būti rodoma ši informacija: siuntos pavadinimas, siuntos numeris, paskutinė siuntos būsena nuskaityta iš Lietuvos pašto sistemos.

4.4.4 Funkcinių reikalavimų tikrinimas veiklos metamodelio taisyklių atžvilgiu

Nagrinėjamas naujos siuntos įkėlimo į programėlę panaudos atvejis. Vartotojas turi įvesti siuntos stebėjimo numerį, kuris privalo būti 13 ženklų ilgio: pirmos dvi raidės nusako siuntos tipą, devyni skaitmenys yra unikalus numeris, paskutiniai du ženklai yra šalies siuntėjos kodas (pvz. *MP100382863LT*). Jeigu vartojas įveda formato neatitinkantį siuntos sekimo numerį aplikacija negali teisingai jo interpretuoti ir pateikti vartotojui su siunta susijusių duomenų. t.y. būtina tikrinti siuntos numerio struktūrą. Remiantis asmenine darbo patirtimi galiu teigti, jog panašūs tikrinimai dažnai yra nespecifikuojami ir klaidos surandamos testavimo metu arba, kai produktas jau

yra darbinėje aplinkoje. Šią klaidą galima nustatyti panaudojant formalias veiklos procesų tikrinimo procedūras, naudojant veiklos metamodelio taisykles (EVC tikrinimą). Žiniomis grindžiamo MDA metodo aplinkoje, vartotojo duomenų įvedimo veikla yra priskiriama procesams (visos veiklos kurios nekeičia duomenų, o tik transformuoja jų atvaizdavimo/pernešimo formatą yra traktuojamos kaip procesai). Procesas privalo turėti jį valdančią funkciją, kuri kontroliuoja proceso veiklas. Taigi siuntos numerio įvedimas veikla reikalauja, kad egzistuotų šia veiklą (procesą) valdanti funkcija t.y. žvelgiant iš programavimo perspektyvos metodas, kuris tikrina įvedamus duomenis ir pateikia atsakymą apie jų korektiškumą. Jeigu tokia funkcija (metodas) neegzistuoja t.y. nėra specifikuotas veiklų modelyje, modelio tikrinimas panaudojant VM taisykles sugeneruos klaidos pranešimą, kuris bus pateiktas sistemos analitikui, atlikus klaidų tikrinimo procesą. Konkrečiu atveju yra du procesai kurie atsakingi už duomenų įvedimą „*EnterParcelName*“ ir *EnterTrackingNumber*“ šiuos procesus atlieka vartotojas. Veiklų modelyje matome (žr. pav. 47), jog egzistuoja veikla „*ValidateData*“ (funkcija), kuri atsakinga už įvestų duomenų tikrinimą t.y. valdo procesą. *„ValidateData“* metodas gali grąžinti dvi reikšmes: duomenys teisingai suformatuoti arba duomenys klaidingai suformatuoti.

4.5 Pašto siuntų stebėjimo programėlės architektūra



Pav. 49. Pašto siuntų stebėjimo programėlės komponentai

com.posttracker.entities

Šis modulis yra sudarytas iš duomenų saugojimui skirtų klasių.

ParcelObject klasė yra skirta saugoti ir manipuliuoti vienos siuntos. Šios klasės objektai yra išsaugomi įrenginio atmintyje XML formatus. Paleidus programėlę duomenys yra nuskaitomi iš atmintis ir sukuriami ParcelObjectList objektai, kuriais yra manipuliuojama aplikacijoje. Kiekvienas ParcelObject objektas turi Status objektų sąrašą.

Lentelė 24. Klasės *ParcelObject* savybių sąrašas

Savybė	Aprašymas	Tipas
DestinationCountryStatusList	Elementas kuriame yra saugomi iš Lietuvos pašto sistemos gauti siuntos būsenos duomenys. Elemento tipas yra StatusObjectList iš esmės tai yra StatusObject elementų sąrašas.	StatusObjectList
OriginCountryStatusList	Elementas kuriame yra saugomi iš kilmės pašto sistemos gauti siuntos būsenos duomenys. Elemento tipas yra StatusObjectList iš esmės tai yra StatusObject elementų sąrašas.	StatusObjectList
TrackingNumber	Elementas kuriame yra saugomas siuntos numeris.	String
OriginCountry	Elementas kuriame yra saugomas kilmės šalies kodas	String
ParcelName	Elementas kuriame yra saugomas siuntos pavadinimas.	String
ParcelType	Elementas kuriame yra saugomas siuntos tipo kodas.	String
StatusUpdateDate	Elementas kuriame yra saugomas paskutinės atliktos siuntos būsenos atnaujinimo data.	Date

Lentelė 25. Klasės *ParcelObject* metodų sąrašas

Metodas	Aprašymas
writeToParcel	Metodas skirtas suformatuoti objektą perdavimui tarp skirtingų Activity objektų. Šis metodas yra apibrėžtas *Parcelable* sąsajoje.
ParcelObject	Objekto konstruktorius naudojamas atkurti objektą po perdavimo kitam Activity objektui. Šis metodas yra apibrėžtas *Parcelable* sąsajoje.

StatusObject. Klasė yra skirta saugoti siuntos būsenų duomenis. Yra saugoma data, vietovė ir aprašymas. Ši klasė yra anotuota XML žymomis, tokiu būdu ji gali būti išsaugota XML formatu įrenginio atmintyje.

Lentelė 26. Klasės *StatusObject* savybių sąrašas

Savybė	Aprašymas	

Date	Elementas kuriame yra saugoma siuntos būsenos įvykio data.	Date
Description	Elementas kuriame yra saugoma siuntos būsenos įvykio aprašymas.	String
Location	Elementas kuriame yra saugoma siuntos būsenos įvykio vietovės aprašymas.	String

Lentelė 27. Klasės *StatusObject* metodų sąrašas

Metodas	Aprašymas
writeToParcel	Metodas skirtas suformatuoti objektą perdavimui tarp skirtingų Activity objektų. Šis metodas yra apibrėžtas *Parcelable* sąsajoje.
ParcelObject	Objekto konstruktorius naudojamas atkurti objektą po perdavimo kitam Activity objektui. Šis metodas yra apibrėžtas *Parcelable* sąsajoje.

ParcelObjectList. Klasė yra skirta saugoti siuntų sąrašą. Naudojant XML anotavimą ir šia klasę visas vartotojo siuntų sąrašas gali būti nuskaitomas ir išsaugomas Įrenginio atmintyje. Programoje egzistuoja tik vienas siuntų sąrašas.

Lentelė 28. Klasės *ParcelObjectList* savybių sąrašas

Savybė	Aprašymas	
ParcelsList	Elementas kuriame yra saugomas vartotojo siuntų sąrašas. Ši klasė yra skirta apjungti visus siuntų aprašymus.	List<Parcel Object>

StatusObjectList. Klasėje yra saugomas siuntos būsenų sąrašas. Kiekviena siunta turi du būsenos sąrašus. Vienas siuntos sąrašas suformuojamas iš siuntėjo pašto sistemos duomenų, o kitas iš Lietuvos pašto sistemos duomenų.

Lentelė 29. Klasės *StatusObjectList* savybių sąrašas

Savybė	Aprašymas	
LastStatusEvent	Elementas kuriame yra saugomas paskutinis siuntos būsenos įvykis.	StatusObject
StatusList	Elementas kuriame yra saugomas vartotojo siuntų sąrašas. Ši klasė yra skirta apjungti visus siuntų aprašymus.	List< StatusObject>

Lentelė 30. Klasės *StatusObjectList* metodų sąrašas

Metodas	Aprašymas
writeToParcel	Metodas skirtas suformatuoti objektą perdavimui tarp skirtingų Activity objektų. Šis metodas yra apibrėžtas *Parcelable* sąsajoje.
ParcelObject	Objekto konstruktorius naudojamas atkurti objektą po perdavimo kitam Activity objektui. Šis metodas yra apibrėžtas *Parcelable* sąsajoje.

ParcelViewHolder. Grafinės vartotojo sąsajos elementai, skirti atvaizduoti siuntos informacijos vieną įrašą. Elementas yra sukomponuotas iš trijų tekstinių laukų: Siuntos pavadinimo, siuntos numerio, paskutinio įvykio aprašymo bei dviejų grafinių laukų atitinkamai pavaizduojančių siuntos tipą ir siuntos kilmės šalį

StatusViewHolder. Grafinės vartotojo sąsajos elementai, skirti atvaizduoti siuntos būsenos informacijos vieną įrašą. Elementas yra sukomponuotas iš trijų tekstinių laukų: Apibūdinimo, Datos, Vietos.

Constants. Ši klasė savyje saugoja programėlėje naudojamas konstantas tokias kaip klaidų kodai, vertimai, datų formatai ir pan.

com.posttracker.activities

Šis modulis yra sudarytas iš programėlės logiką realizuojančių klasių: MainActivity, AddTrackingNumberActivity, ListItemActivity. Šio klasės yra atsakingos už grafinių elementų valdymą, sąveikos su vartotoju valdymą bei kitų objektų inicializavimą (pvz. DataManager) nustatyta tvarka.

MainActivity. Tai pagrindinė programėlės klasė, kuri yra atsakinga pagrindinio lango sukūrimą ir navigaciją. Joje atliekamas duomenų nuskaitymas ir saugojimas yra įrenginio atmintį. O taip pat iš šios klasės iniciuojami kiti langai bei asinchroninio duomenų nuskaitymo užduotys.

Lentelė 31. Klasės *MainActivity* savybių sąrašas

Savybė	Aprašymas	Tipas
Parcels	Elementas kuriame yra saugomi visų vartotojo siuntų duomenys. Elemento tipas yra ParcelObject masyvas.	List<ParcelObject>
lvParcels	Elementas kuriame yra grafiškai atvaizduojami siuntų duomenys.	listView
parcelListAdapter	Elementas kuriame yra saugomas siuntos numeris.	ParcelListBaseAdapter

Lentelė 32. Klasės *MainActivity* metodų sąrašas

Metodas	Aprašymas
onCreate	Metodas, kuris yra iškviečiamas sukuriant MainActivity procesą. Jame yra inicializuojamos kintamųjų reikšmės, nuskaitomi duomenys iš įrenginio atmintis, inicializuojami grafiniai bei meniu elementai.
onStop	Metodas, kuris iškviečiamas MainActivity procesui baigiant darbą. Jame atliekami duomenų išsaugojimo darbai.
onActivityResult	Metodas, kuris iškviečiamas kai procesas yra gauna

	duomenis iš kito proceso.
onCreateOptpnsMenu	Metodas, kuriame aprašoma meniu elementų sukūrimo logika.
onContextItemSelected	Metodas, kuris iškviečiamas pažymėjus kontekstinio meniu elementą. Šiame metode aprašoma veiksmų seka,
onOptionsItemSelected	Metodas, kuris iškviečiamas pažymėjus meniu elementą. Šiame metode aprašoma veiksmų seka,
GetParcels	Metodas skirtas užkrauti siuntų duomenis iš įrenginio atminties. Duomenys yra nuskaitomi inicializavus DataManager objektą.
ShowStatusUpdateMessage	Metodas skirtas sugeneruoti pranešimą vartotojui apie siuntos trynimo negalimumą atnaujinant siuntos informaciją.
SaveData	Metodas skirtas išsaugoti siuntų duomenis įrenginio atmintyje. Duomenys yra išsaugomi iškviečiant DataManager objektą ir jam perduodant ParcelObjectList objektą.
DeleteParcel	Metodas skirtas pašalinti siuntą iš sistemos. Siuntą yra ištrinama iš įrenginio atmintis.

AddTrackingNumberActivity. Klasės skirta įtraukti naujos siuntos duomenis į sistemą. Vartotojas šio proceso metu privalo įvesti teisingą siuntos numerį bei nurodyti siuntos pavadinimą.

Lentelė 33. Klasės *AddTrackingNumberActivity* metodų sąrašas

Metodas	Aprašymas
onCreate	Metodas, kuris yra iškviečiamas sukuriant AddTrackingNumberActivity procesą. Jame yra inicializuojamos kintamųjų reikšmės, nuskaitomi duomenys iš įrenginio atmintis, inicializuojami grafiniai bei meniu elementai.
GetData	Metodas skirtas nuskaityti duomenis iš XML failo ir sukurti egzistuojančių siuntų sąrašą.
SaveData	Metodas, kuris iškviečiamas AddTrackingNumberActivity procesui baigiant darbą. Jame atliekami duomenų išsaugojimo į XML failą veiksmai.
ValidateInput	Metodas skirtas patikrinti vartotojo įvedamus duomenis (siuntos pavadinimą ir siuntos numerio formatą)

ListItemActivity Klasė skirta atvaizduoti konkrečios siuntos būsenų duomenis vartotojui. Rodomi duomenys yra iš Lietuvos pašto sistemos bei kilmės šalies sistemos, jeigu tam yra sukurtas atitinkamas *Parser* klasės objektas

Lentelė 34. Klasės *ListItemActivity* savybių sąrašas

Savybė	Aprašymas	Tipas
lvlDestinationParcelStatus	Elementas kuriame yra grafiškai atvaizduojami siuntų būsenų duomenys iš Lietuvos pašto sistemos.	listView
lvOriginParcelStatus	Elementas kuriame yra grafiškai	listView

	atvaizduojami siuntų duomenys iš siuntos kilmės pašto sistemos.	

Lentelė 35. Klasės *ListItemActivity* metodų sąrašas

Metodas	Aprašymas
onCreate	Metodas, kuris yra iškviečiamas sukuriant ListItemActivity. Jame yra inicializuojamos kintamųjų reikšmės, nuskaitomi duomenys iš įrenginio atmintis, inicializuojami grafiniai bei meniu elementai.
GetPostLogo	Metodas, kuri susieja siuntos kodą su siuntą reprezentuojančiais paveikslėliais.

com.posttracker.adapters

ParcelListBaseAdapter Klasė kuri skirta sukurti grafinius siuntos duomenis atvaizduojančius elementus t.y. siuntų sąrašo objektą.

Lentelė 36. Klasės *ParcelListBaseAdapter* savybių sąrašas

Savybė	Aprašymas	Tipas
parcelsArrayList	Elementas kuriame yra saugomi siuntų duomenys. Elemento tipas yra ParcelObjectList iš esmės tai yra ParcelObject elementų sąrašas.	List<ParcelObject>
liMain	Elementas, kuris naudojamas sukurti grafinės posistemės funkcijas.	listView
FlagImgIdsList	Klasėje naudojamų vėliavų paveikslėlių identifikatorių sąrašas.	List<int>
ParcelTypeIdsList	Klasėje naudojamų siuntos tipo paveikslėlių identifikatorių sąrašas.	List<int>

Lentelė 37. Klasės *ParcelListBaseAdapter* metodų sąrašas

Metodas	Aprašymas
GetFlagResoureId	Metodas, kuris gražina vėliavos paveikslėlio Id pagal parametru perduotą šalies kodą.
GetParcelTypeResourceId	Metodas, kuris gražina siuntinio tipo paveikslėlio Id pagal parametru perduotą siuntos kodą.
getCount	Metodas, kuris gražina adapteryje saugomų elementų skaičių.
getItem	Metodas, kuris gražina adapteryje saugomą elementą.
getItemId	Metodas, kuris gražina adapteryje saugomą elementą pagal elemento Id.
getView	Metodas, kuris sugeneruoja ir grąžina grafinę elemento reprezentaciją.

OriginCountryStatusAdapter Klasė kuri skirta sukurti grafinius siuntos būsenų duomenis iš kilmės šalies pašto sistemos atvaizduojančius elementus t.y. siuntų būsenų sąrašo objektą.

Lentelė 38. Klasės *OriginCountryStatusAdapter* savybių sąrašas

Savybė	Aprašymas	Tipas
statusArrayList	Elementas kuriame yra saugomi iš	List<StatusObject>

	kilmės šalies pašto sistemos gauti siuntos būsenos duomenys. Elemento tipas yra StatusObjectList iš esmės tai yra StatusObject elementų sąrašas.	
liMain	Elementas, kuris naudojamas sukurti grafinės posistemės funkcijas.	listView

Lentelė 39. Klasės *OriginCountryStatusAdapter* metodų sąrašas

Metodas	Aprašymas
getCount	Metodas, kuris gražina adapteryje saugomų elementų skaičių.
getItem	Metodas, kuris gražina adapteryje saugomą elementą.
getItemId	Metodas, kuris gražina adapteryje saugomą elementą pagal elemento Id.
getView	Metodas, kuris sugeneruoja ir grąžina grafinę elemento reprezentaciją.

StatusListBaseAdapter. Klasė kuri skirta sukurti grafinius siuntos būsenų duomenis iš Lietuvos pašto sistemos atvaizduojančius elementus t.y. siuntų būsenų sąrašo objektą.

Lentelė 40. Klasės *StatusListBaseAdapter* savybių sąrašas

Savybė	Aprašymas	Tipas
statusArrayList	Elementas kuriame yra saugomi iš Lietuvos pašto sistemos gauti siuntos būsenos duomenys.	List< StatusObject >
liMain	Elementas, kuris naudojamas sukurti grafinės posistemės funkcijas.	listView

Lentelė 41. Klasės *StatusListBaseAdapter* metodų sąrašas

Metodas	Aprašymas
getCount	Metodas, kuris gražina adapteryje saugomų elementų skaičių.
getItem	Metodas, kuris gražina adapteryje saugomą elementą.
getItemId	Metodas, kuris gražina adapteryje saugomą elementą pagal elemento Id.
getView	Metodas, kuris sugeneruoja ir grąžina grafinę elemento reprezentaciją.

com.posttracker.parsers

ParserFactory klasė yra skirta sukurti ir grąžinti konkretaus pašto duomenims nuskaityti reikalingą *Parser* objektą.

Lentelė 42. Klasės *PaserFactory* metodų sąrašas

Metodas	Aprašymas
GetParser	Metodas skirtas grąžina nustatyto tipo Parser objektą. Metodui per parametrus perduodamas reikalaujamo objekto tipas.

IReader. Sąsaja, kurią turi realizuoti visos *Parser* tipo klasės. Ši sąsaja turi vieną metodą „GetParcelStatusList“, kuris turi grąžinti siuntos būsenų sąrašą.

Lentelė 43. Sąsajos *IReader* metodų sąrašas

Metodas	Aprašymas
GetParcelStatusList	Metodas skirtas grąžintis siuntos būsenų sąrašą. Metodui perduodamas siuntos numeris (String tipo).

HonkongPostParser klasė realizuojanti *IReader* sąsają ir skirta nuskaityti duomenis iš Honkongo pašto sistemos.

LithuaniaPostParser klasė realizuojanti *IReader* sąsają ir skirta nuskaityti duomenis iš Lietuvos pašto sistemos.

USPostParser klasė realizuojanti *IReader* sąsają ir skirta nuskaityti duomenis iš Jungtinių Valstijų pašto sistemos.

com.posttracker.iodata

DataManager klasė yra skirta atlikti duomenų sluoksnio veiksmus t.y. į XML failą išsaugoti siuntų duomenis bei juos nuskaityti.

Lentelė 44. Klasės *DataManager* metodų sąrašas

Metodas	Aprašymas
GetData	Metodas, kuris skirtas nuskaityti siuntų duomenis iš XML failo.
SaveData	Metodas, kuris skirtas išsaugoti siuntų duomenis į XML failą.

com.posttracker.asyncdataloaders

PostDataLoader klasė yra skirta parsiųsti siuntos duomenis iš pašto sistemos.

Lentelė 45. Klasės *PostDataLoader* metodų sąrašas

Metodas	Aprašymas
doInBackgoround	Metodas, kuris skirtas asinchroniškai nuskaityti siuntų duomenis iš pašto sistemų.
onPostExecute	Metodas, kuris skirtas grąžinti iš pašto sistemų nuskaitytus siuntų duomenis.

4.6 Testavimo duomenys

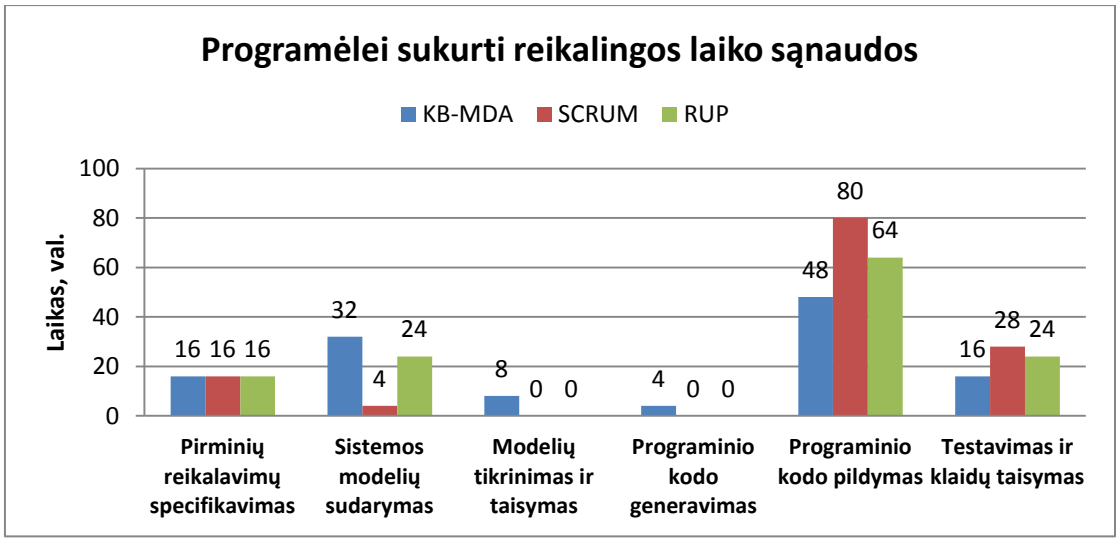
Testavimui buvo naudojami realūs siuntų duomenys. Siuntos informacija buvo stebima nuo jos išsiuntimo iš kilmės šalies iki gavimo Lietuvos pašte bei pristatymo vartotojui..

Lentelė 46. Testavimui naudotų siuntų numeriai bei kilmės šalys

Siuntos numeris	Kilmės šalis
MP100382863LT	Lietuva
CG085160116US	JAV
RT159081832HK	Honkongas
RB714936039CN	Kinija

4.7 Išvados

Programėlės kūrimas apėmė šiuos etapus: pirminis reikalavimų specifikavimas, sistemos modelių sudarymas, modelių tikrinimas (nuoseklumo tikrinimas, VM tikrinimas) ir taisymas, programinio kodo generavimas iš sukurtų modelių, programinio kodo koregavimas ir pildymas, testavimas ir klaidų taisymas. Paveiksle žemiau yra pateiktos laiko sąnaudos kiekvienam etapui, kuriant programėlę vadovaujantis žiniomis grindžiamo MDA metodo principais (palyginimui pateiktos teorinės laiko sąnaudos remiantis „RUP“ ir „Scrum“ metodais).



Pav. 50. Laiko sąnaudos vienai platformai pagal programinės įrangos kūrimo etapus

Lentelė 47. Laiko sąnaudos vienai platformai pagal programinės įrangos kūrimo etapus

Etapas (laikas,val.) / Metodas	**KB-MDA**	**SCRUM**	**RUP**
Pirminių reikalavimų specifikavimas	16	16	16
Sistemos modelių sudarymas	32	4	24
Modelių tikrinimas ir taisymas	8	0	0
Programinio kodo generavimas	4	0	0
Programinio kodo pildymas	48	80	64
Testavimas ir klaidų taisymas	16	28	24
Viso:	124	128	128

Programėlės SysML modelių sudarymas yra imlus laikui ir darbui procesas. Pagrindinis sunkumas su kuriuo buvo susidurta sudarant modelius, tai CIM modelio abstrakcijos lygmens pasirinkimas. Pagal apibrėžimą CIM modelis turi aprašyti veiklos procesus vartotojui suprantamu būdu, tai iš principo lemia aukštą abstrakcijos lygmenį, kuris yra nepakankamas tiksliam reikalavimų surinkimui į veiklos modelį. Šią problemą galima spręsti sudarant kelių abstrakcijos lygmenų CIM modelius. Taigi pirminis (*CIM0*) modelis yra sukuriamas vartotojo ir analitiko, o iš jo sukuriamas detalizuotas, mažesnio abstrakcijos lygmens (*CIM1*) modelis, kurį sudaro sistemos analitikas. Šis modelis anotuojamas žymėmis nusakančiomis veiklos modelio funkcijoms (tikrinimui, transformavimui) reikalingą informaciją. Tik atlikus papildomą anotavimą žymėmis *CIM1* modelis gali būti perduodamas žiniomis grindžiamo MDA įrankio analizei. Galutinis *CIM1* modelio variantas pagal savo detalumą ir sudėtingumą tapo artimas PIM modeliui. Iš dalies tai lėmė SysML kalbos specifika, todėl vertėtų išanalizuoti galimybę sukurti SysML (UML dialektą), kuris aprašytų bendrai naudojamus mobiliems įrenginiams būdingus elementus (pvz. asinchroninio duomenų nuskaitymo klases) ir tokiu būdu sumažintų CIM modelio sudėtingumą.

Naudotas modelių nuoseklumo tikrinimas leido pastebėti semantines (pvz. trūkstamus blokus struktūros aprašų modelyje) ir sintaksines (skirtingi elementų pavadinimai) modelių klaidas, taigi užtikrino modelių kokybę. Tikrinimas veiklos modelio atžvilgiu sudarė sąlygas tiksliais atskirti duomenų tikrinimo ir transformavimo (apdorojimo) procesus ir tokiu būdu nustatyti trūkstamas duomenų tikrinimo funkcijas.

Dėl laikui imlaus modelių sudarymo proceso, vienai platformai skirtos programėlės sukūrimo laikas nebuvo trumpesnis nei pasirinkus konkretų agilųjį ar „Plan Driven“ metodą. Tačiau remiantis ilgalaike darbo patirtimi galima teigti, jog laiko ekonomija atsiranda kuriant programėles dviems ar daugiau operacinių sistemų dėl šių veiksnių: pilnesnio vartotojo reikalavimų surinkimo bei jų pateikimo formato programuotojui (detalūs SysML/UML modeliai), automatinio programinio kodo generavimo, trumpesnio testavimo etapo.

IŠVADOS

1) Atlikus klasikinių PĮ kūrimo procesų veiklos modeliavimo ir vartotojo reikalavimų specifikavimo etapų analizę, nustatyta, kad analitiko ir projektuotojo patirtis turi didelę įtaką šiuose etapuose vykstantiems procesams, todėl nuspręsta, kad tikslinga sukurti metodą, kuris sumažintų empirinių veiksnių įtaką.
2) Empirinių veiksnių įtakos mažinimui tikslinga taikyti perspektyvią IS inžinerijos šaką – žiniomis grindžiamą IS inžineriją, kurios vienas iš esminių taikymo privalumų yra surinktų dalykinės srities žinių tikrinimas formalių kriterijų atžvilgiu, o tai įtakoja efektyvesnį (lyginant su klasikiniais IS kūrimo metodais) IS inžinerijos procesą (detalesnį vartotojo reikalavimų specifikavimą, mažesnį loginių trukių skaičių tarp programinės įrangos kūrimo dalyvių).
3) Problemai spręsti pasirinkta modeliais grindžiamos IS kūrimo krypties OMG sukurta MDA koncepcija, UML ir SysML modeliavimo kalba ir XMI modelių pernešimo formatas, todėl kad šiomis priemonėmis veiklos modelį galima funkcionaliai integruoti su vidiniais MDA modeliais (CIM ir PIM), išsaugant galimybę automatizuoti IS kūrimo procesą.
4) Žiniomis grindžiamas MDA metodas sukurtas klasikinę MDA architektūrą papildant nauju elementu - veiklos modeliu. Šis veiklos modelis papildytas naujais elementais, kurie įgalina jo integraciją su vidiniais MDA modeliais (CIM, PIM). Sukurti sąsajų žemėlapiai tarp CIM=>VM ir VM=>PIM, suprojektuoti CIM=>VM ir VM=>PIM modelių transformavimo algoritmai, sukurtas modelių transformavimo ir tikrinimo prototipas (panaudos atvejų ir veiklų modeliams);
5) Siekiant nustatyti metodo efektyvumą buvo atliktas empirinis tyrimas, panaudojant sukurtą dalykinės programos prototipą. Tyrimo rezultatai parodė, kad šį metodą tikslinga naudoti daugiaplatformių sistemų kūrimo atvejais bei taikyti sistemoms, kurioms svarbūs kokybės reikalavimai bei išplečiamumo galimybės. Tai grindžiama atliktu

eksperimentu, kurio metu buvo sukurta pašto siuntų stebėjimo programėlė „Android“ platformai.

6) Tyrimo metu nustatyta, kad taikyti žiniomis grindžiamą MDA metodą IS inžinerijoje tikslinga nes: detaliau dokumentuojami vartotojo reikalavimai (jie yra tikrinami formalių kriterijų atžvilgiu), sumažinama loginių trukių atsiradimo galimybė (tarp programinės įrangos kūrimo dalyvių), daugiaplatforminiuose sprendimuose sumažinamos projekto įgyvendinimo laiko sąnaudos (dėka automatinio kodo generavimo iš patikrintų modelių).

LITERATŪROS SĄRAŠAS

1. **Agile Alliance** [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.agilealliance.org/>
2. **Ambler S. W.** *Agile Modeling* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: http://www.agilemodeling.com/essays/inclusiveModels.htm
3. **Ambler S. W.** *Agile Unified Process* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.] Prieiga per internetą: <http://www.ambysoft.com/unifiedprocess/agileUP.html>
4. **AndroMDA** [interaktyvus]. 2012 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.andromda.org/index.html>
5. **Asadi, M.; Ramsin, R.** MDA-based Methodologies: An Analytic Survey. Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications. pp. 419-431, Berlin (2008)
6. **Butleris R.; Lopata. A.; Ambraziunas M.; Gudas. S.:** *The Main Principles of Knowledge-Based Information Systems Engineering.* Electronics And Electrical Engineering, Vol. 120, No 4 (2012), pp. 99-102, ISSN 1392-1215, 2012
7. **Cabot, J.:** *Clarifying concepts: MBE vs MDE vs MDD vs MDA* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 07 d.]. Prieiga per internetą: <http://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/>
8. **Chanda, J.; Kanjilal, A.; Sengupta, S.; Bhattacharya, S.** *Traceability of requirements and consistency verification of UML use case, activity and Class diagram: A Formal approach.* In Methods and Models in Computer Science, ICM2CS 2009, 2009.
9. **Chown, B.; Lange, M.** *Modernizing System Development: Requirements-Based, Model-Driven Design, Implementation and Test.* Embedded Real Time Software and Systems [interaktyvus]. 2012 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.erts2012.org/Site/0P2RUC89/TA-1.pdf>
10. **CMS Office of Information Service**. *Selecting a development approach* [interaktyvus]. 2009 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf>.

11. **Cohen, D.; Lindvall, M.; Costa, P.:** *An introduction to agile methods*, Advances in Computers, New York, Elsevier Science, 2004, pp. 1-66. Prieiga per internetą: <http://robertfeldt.net/courses/agile/cohen_2004_intro_to_agile_methods.pdf>

12. **Čeponienė, L.; Nemuraitė, L.; Vedrickas, G.** *Separation of event and constraint rules in UML&OCL models of service oriented information systems.* Information technology and control, Vol. 38, No. 1. p.p. 29–37, 2009

13. **D. Silingas, R. Vitiutinas.**: *Towards UML-Intensive Framework for Model-Driven Development.* B. Meyer, J.R. Nawrocki, and B. Walter (Eds.):Second IFIPTC Central and East European Conference on Software Engineering Techniques (CEE-SET 2007), Poznan, Poland , Lecture Notes in Computer Science, Vol. 5082, 2008, 116-128

14. **Dalgarno, M.; Fowler, M.:** *UML vs. Domain-Specific Languages* [interaktyvus]. 2008 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.methodsandtools.com/archive/archive.php?id=71/>

15. **Department of Defence.** *DoD Architecture Framework Version 2.0 Volume 2: Architectural Data and Models Architect's Guide* [interaktyvus]. 2009 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://dodcio.defense.gov/Portals/0/Documents/DODAF/DoDAF%20V2%20-%20Volume%202.pdf>

16. **Eichelberger, H.; Eldogan, Y.; Schmid, K.** *A Comprehensive Analysis of UML Tools, their Capabilities and thei Compliance* [interaktyvus]. 2011 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.uni-hildesheim.de/media/fb4/informatik/AG_SSE/PDFs/UML-Tools-2.0.pdf>.

17. **Ellis K.** *The Impact of Business Requirements on the Success of Technology Projects*. Benchmark, IAG Consulting (2008)

18. **European Committee for Standardization.** *ENV 12 204: Advanced Manufacturing Technology – Systems Architecture – Constructs for Enterprise Modelling*, CEN TC 310/WG1, 1996, p.42.

19. **European Committee for Standardization.** *ENV 40 003: Computer Integrated Manufacturing – Systems Architecture – Framework for Enterprise Modelling*, CEN/CENELEC, 1990.

20. **Fox, M. S., Gruninger, M.** *Enterprise Modeling.* American Association for Artificial Intelligence [interaktyvus] 1998 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: http://www.eil.utoronto.ca/enterprise-modelling/papers/fox-aimag98.pdf

21. **Fraternali, P.** *General Purpose or Domain Specific: not all modeling languages are equal [interaktyvus]*. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://blog.webratio.com/2011/04/05/general-purpose-or-domain-specific-not-all-modeling-languages-are-equal/>.

22. **Goyal, S.:** *Agile Techniques for Project Management and Software engineering* [interaktyvus]. 2007 [žiūrėta 2014 m. gegužės 07 d.]. Prieiga per internetą: <http://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd.pdf>

23. **Gudas, S.** *Informacijos sistemų inžinerijos teorijos pagrindai.* Monografija. Vilnius: Vilniaus universiteto leidykla, 2012, 384 p. ISBN 978-609-459-075-7

24. **Gudas, S.; Lopata, A.** *Meta-model based development of use case model for business function.*Information technology and control, ISSN 1392-124X. 2007, Vol. 36, No. 3, p.p 302-309.

25. **Gudas, S.; Lopata, A.** *Žiniomis grindžiama informacijos sistemų inžinerija.* Informacijos mokslai: mokslo darbai, Vilnius: Vilniaus universiteto leidykla, 2004, Vol. 30, p 90-98. ISSN 1392-0561

26. **Gudas, S.; Lopata, A.; Skersys T.** *Approach to Enterprise Modelling for Information Systems Engineering.* Informatica, Vol. 16, No. 2, Institute of Mathematics and Informatics, Vilnius, 2005, pp. 175-192., 2005

27. **Gudas, S; Pakalnickas, E.** *Enterprise Management view based Specification of Business Components.* Proceedings of 15-th international conference on Information and software technologies, IT'2009, Kaunas, Technologija, 2009, p. 417-426, 2009 ISSN 2029-0020

28. **Gupta, M.M.; Sinha N. K.** *Intelligent Control Systems: Theory and Applications.* The Institute of Electrical and Electronic Engineers Inc., New York. 1996.

29. **Haan J. D.** *Model Driven Engineering* [interaktyvus]. 2008 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.theenterprisearchitect.eu/blog/2008/03/14/model-driven-engineering/>

30. **Haan, J. D.** *Combining general purpose languages and domain specific languages for Model Driven Engineering* [interaktyvus]. 2008 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.theenterprisearchitect.eu/blog/2008/04/15/combining-general-purpose-languages-and-domain-specific-languages-for-model-driven-engineering/>

31. **Hazra, R.; Dey, S.;** *Consistency between Use Case, Sequence and Timing Diagram for Real Time Software Systems.* International Journal of Computer Applications, Vol. 85, No. 16, ISSN 0975 - 8887, 2014

32. **Heuluy, B.; Vernadat, F.B.** *The CIMOSA Enterprise Ontology*. Proceedings of the IFAC Workshop–MIM'97, Vienna, 1997. p. 37–41.

33. **Hull, E.; Jackson K.; Dick J.:** *Requirements Engineering, Second Edition*. Springer, ISBN 1-85233-879-2, 2005

34. **IBM** [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.ibm.com/us/en/>

35. **IBM.** *RUP: Best practices for design, implementation and effective project management*[interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www-01.ibm.com/software/rational/rup/>

36. **IFIP–IFAC Task Force on Architectures for Enterprise Integration.** *GERAM:Generalised Enterprise Reference Architecture and Methodology* [interaktyvus]. 1999 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą < http://www.cit.gu.edu.au/~bernus/taskforce/geram/versions/geram1–6–3/v1.6.3.html>

37. **Yamin, M.; Zuna, V.; Bugami A.** *Requirements Analysis and Traceability at CIM Level.* Journal of Software Engineering and Applications, ISSN 1945-3124, Vol.3 No.9, p.p. 845-851, 2010

38. **Jeffries, R.** *Extreme Programming* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: < http://xprogramming.com/book/whatisxp/>**.**

39. **JetBrains.** *Resharper* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.jetbrains.com/resharper/>

40. **JSoup.** 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://jsoup.org/>

41. **Juhani, W.; Abrahamssom, P.; Salo, O.; j. Ronkainen.:** Agile software development methods [interaktyvus], ISBN 951-38-6010-8. 20024 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: < http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>

42. **Kapocius, K; R. Butleris.:** *Repository for Business Rules Based IS requirements.* Informatica, Vol.17, No.4, 2006, 503-518. //Butlerio EM

43. **Kardos, M.; Drozdova M.:** *Analytical Method of CIM to PIM Transformation in Model Driven Architecture (MD*A). Journal of Information and Organizational Sciences, Vol. 34, No. 1, 2010

44. **Leszek M.,** Requirements Analysis and Systems Design (3rd Edition), June 22, 2007 | ISBN-10: 0321440366 | ISBN-13: 978-0321440365 | Edition: 3rd

45. **Lopata A., Ambraziūnas M.** „Knowledge-Based MDA Approach“ // BIS 2011 International Workshops, Poznan, Poland, June 15-17, 2011, Series: Lecture Notes in Business Information Processing, Vol. 97, Abramowicz, Witold; Maciaszek, Leszek; Węcel, Krzysztof (Eds.), 2011, XV, 307p., Softcover ISBN: 978-3-642-25369-0

46. **Lopata A., Ambraziunas M.** MDA Compatible Knowledge– Based IS Engineering Approach. AICI 2010 (LNCS), Volume 6319, p 230-238.

47. **Lopata A., Ambraziūnas M., Gudas S.** Knowledge-Based Approach to Bussines and IT Aligment Modelling. Transformations in Business & Economics, Vol. 10, No 2, p. 60-73, 2011.

48. **Lopata A., Ambraziūnas M., Gudas S., Butleris R., Butkienė R.** „Enterprise Knowledge-Based Generation of Class Model“, Electronics And Electrical Engineering, Vol. 19, No 2 (2013), pp. 79-82, ISSN 1392-1215, 2013

49. **Lopata A., Gudas S.** Veiklos modeliu grindžiamas kompiuterizuotas funkcinių vartotojo reikalavimų specifikavimo metodas, daktaro laipsnio disertacija, VU KHF, 2005.

50. **Lopata A., Gudas S.** Workflow- Based Acquisition and Specification of Functional Requirements. Proceedings of 15th International Conference on Information and Software Technologies IT2009, Kaunas, Lithuania April 23-24. p. 417- 426 ISSN 2029-0020.

51. **Lucas, F. J.; Molina, F.; Toval, A.** *A Systematic Review of UML Model Consistency Management*. Information and Software Technology, 51, 1631-1645. 2009

52. **Mayer, R.** *IDEF Family of Methods for Concurrent Engineering and Business Re–engineering Applications* [interaktyvus]. 1992 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: < http://www.idef.com/Downloads/pdf/IDEFFAMI.pdf>.

53. **Maskeliūnas, S.** Žinių technologijų (ir saityno technologijų) terminų žodynėlis (2012-03-01 d. versija). [interaktyvus]. 2012 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://eta.ktl.mii.lt/~mask/LIKS-IS/Z%27iniu_technologiju_z%27odyne%27lis.pdf>

54. **Mellor S. J.** *Agile MDA* [Interaktyvus]. [žiūrėta 2008 11 10]. Prieiga per internetą: www.omg.org/mda/mda_files/AgileMDA.pdf.

55. **Microsoft.** .*Net Framework 4.5* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: < http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2.aspx>.

56. **Microsoft.** [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: < http://www.microsoft.com>.

57. **Microsoft.** *SQL Express* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: < http://msdn.microsoft.com/en-us/evalcenter/dn434042.aspx>.

58. **Microsoft.** *Visual Studio 2012* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: < http://www.microsoft.com/visualstudio/eng>.

59. **Miller J., Mukerji J.** MDA Guide Version 1.0.1 [Interaktyvus]. [žiūrėta 2009 02 08]. Puslapio nuoroda: www.omg.org/docs/omg/03-06-01.pdf.

60. **Moccia, J.:** *Agile Requirements Definition and Management* interaktyvus]. 2012 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.scrumalliance.org/community/articles/2012/february/agile-requirements-definition-and-management>

61. **Moliz.** *Testing UML models based on fUML*[interaktyvus]. 2012 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.modelexecution.org/?page_id=524>

62. **Nielsen.:** *Top U.S. Smarthphone operating systems by market share* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.nielseneurope.pl/us/en/newswire/2014/smartphone-milestone-half-of-americans-ages-55-own-smartphones.html>

63. **NoMagic.** *MagicDraw* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.nomagic.com/products/magicdraw.html>

64. **Noraini, I.; Rosziati, I.; Mohd Zainuri, S.; Dzahar, M.; Tutut H.** *Consistency Rules between UML Use Case and Activity Diagrams Using Logical Approach.* International Journal of Software Engineering and Its Applications. Vol. 5 No. 3, July, 2011

65. **Object Management Group.** OMG [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.omg.org >.

66. **Object Management Group.** *Semantics of a Foundational Subset for Executable UML Models (fUML)* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.omg.org/spec/FUML/1.1/PDF>.

67. **Object Management Group.** *Systems Modeling Language* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.omg.org/spec/SysML/1.3/PDF>.

68. **Object Management Group.** *Unified Modeling Language (UML)* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.omg.org/spec/UML/2.4.1/>.

69. **Object Management Group.** *XML Metadata Interchange* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.omg.org/spec/XMI/2.4.2/PDF>.

70. **Passing, J.**: *Requirements Engineering in the Rational Unified Process.*Hasso Plattner Insitute for Software Systems Engineering, 2006 [žiūrėta 2014 m. gegužės 14 d.]. Prieiga per internetą: <http://int3.de/res/RUP/RUP_Paper_JohannesPassing.pdf>

71. **Petersen, K.; Wohlin, C.** *The effect of moving from a plan-driven to an incremental software development approach with agile practices.* Empirical Software Engineering, Vol. 15, No. 6, 2010 p.p. 654-693

72. **Robertson, S.; Robertson, J.:** *Volere Requirements Techniques: an Overviev* [interaktyvus]. 2008 [žiūrėta 2014 m. liepos 20 d.]. Prieiga per internetą:http://www.softed.com/Resources/Docs/VolereOverview1.pdf

73. **Rumbaugh, J.** *OMT Insights: Perspective on Modeling from the Journal of Object–Oriented Programming.* Signature sounds recording, 1997. p. 412. ISBN: 0138469652.

74. **Schmidt, D.C.:** *Model-Driven Engineering* [interaktyvus]. 2006 [žiūrėta 2014 m. gegužės 07 d.]. Prieiga per internetą: <https://www.dre.vanderbilt.edu/~schmidt/GEI.pdf>

75. **Schwaber, K.; Sutherland, J.** *The Scrum Guide* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <https://www.scrum.org>.

76. **Scott, K.; Uhl, A.; Weise, D.; Mellor S. J.** *MDA Distilled: Principles of Model-Driven Architecture* 2004

77. **Seidewitz, E.** *Programming in UML: An Introduction to fUML and Alf* [interaktyvus]. 2011 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.omg.org/news/meetings/tc/agendas/va/xUML_pdf/Seidewitz_Tutorial.pdf>.

78. **Sharifi, H. R.; Mohsenzadeh M.** *A New Method for Generating CIM Using Business and Requirement Models.* World of Computer Science and Information Technology Journal (WCSIT), ISSN: 2221-0741, Vol. 2, No. 1, p.p. 8-12, 2012

79. **Silingas, D.; Butleris, R.**: *Towards Implementation a Framework for Modelling Software Requirements in MagicDraw UML.* Information Technology and Control Vol. 38 No. 2, p.p. 153–164 (2009)

80. **Silingas, D.; Vitiutinas, R.; Nemuraite, L.; Pavalkis, S.:** *Integrating GUI Prototyping Into UML Toolkit*. Proceedings of 16th International Conference on Information Technologies IT 2010, pp 224-232, 2010, ISSN 2029-0020

81. **Simple.** *SimpleXML* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://simple.sourceforge.net/ >.

82. **Skersys T.** *Business Knowledge-Based Generation of the System Class Model*. Informatica, Vol. 37, No. 2, Vilnius, 2008, p. 145 - 153, 2008

83. **SparxSystems.** *Enterprise Architect* [interaktyvus]. 2012 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: http://www.andromda.org/index.html

84. **Sudhakar, G. P.; Farroq, A.; Patnaik, S.:** *Measuring productivity of software development teams* [interaktyvus]. 2012 [žiūrėta 2014 m. liepos 20 d.]. Prieiga per internetą: http://www.sjm06.com/SJM%20ISSN1452-4864/7_1_2012_May_1_170/7_1_65-75.pdf

85. **The Eclipse Foundation.** *Acceleo* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 06 d.]. Prieiga per internetą: <http://www.eclipse.org/acceleo/>

86. **Tolvanen, J. P.; Kelly, S.** *Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences.* SPLC 2005, LNCS 3714, pp. 198 – 209, 2005.

87. **Vernadat, F.:** *UEML: towards a unified enterprise modelling language.* International Journal of Production Research, Vol. 40, no. 17, p. 4309–4321, 2002

88. **Vidmantas, M.; Kazanavičius, E.:** *Conception of a Multi-Platform System Software and Firmware Development Tool.* Informacijos mokslai, Vol. 50, ISSN 1392-0561, 2009

89. **Vrancken, J.; Santos Soares, M.:** *Model-Driven User Requirements Specification using SysML.* Journal Of Software, Vol. 3, No. 6, ISSN 1796-217X, pp. 57-68, 2008 http://www.academypublisher.com/jsw/vol03/no06/jsw03065768.pdf

90. **Watson, A.:** *UML® vs. DSLs: A false dichotomy* [interaktyvus]. 2009 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <http://www.omg.org/cgi-bin/doc?omg/08-09-03.pdf>

91. **Watson, A.:** *Visual Modelling: past, present and future* [interaktyvus]. 2014 [žiūrėta 2014 m. gegužės 05 d.]. Prieiga per internetą: <www.uml.org/Visual_Modeling.pdf>

92. **Williams, L.:** *A Survey of Agile Development Methodologies* [interaktyvus]. 2007 [žiūrėta 2014 m. gegužės 07 d.]. Prieiga per internetą: http://agile.csc.ncsu.edu/SEMaterials/AgileMethods.pdf

PRIEDAI

1. Autoriaus publikacijos disertacijos tematika

Publikacijos periodiniuose mokslo leidiniuose, turinčiuose cituojamumo rodiklį (Impact Factor) Thomson Reuters Web of Knowledge duomenų bazėje:

Priimta spausdinti:

1. Lopata A., Ambraziūnas M., Veitaitė I., Masteika S., Butleris R. „SysML and UML models usage in Knowledge Based MDA process“, Electronics and Electrical Engineering, ISSN 1392-1215, 2013 (Spaudoje)

Atspausdinta:

2. Lopata A., Ambraziūnas M., Gudas S., Butleris R., Butkienė R. „Enterprise Knowledge-Based Generation of Class Model“, Electronics And Electrical Engineering, Vol. 19, No 2 (2013), pp. 79-82, ISSN 1392-1215, 2013

3. Lopata A., Ambraziūnas M., Gudas, S. „Knowledge Based MDA Requirements Specification and Validation Technique“, Transformations in Business & Economics, Vol. 11, No 1 (25), pp. 248-261, ISSN 1648 - 4460, 2012

4. Lopata A., Ambraziūnas M., Gudas S., Butleris R. „The Main Principles of Knowledge-Based Information Systems Engineering“, Electronics And Electrical Engineering, Vol. 120, No 4 (2012), pp. 99-102, ISSN 1392-1215, 2012

5. Lopata, A., Ambraziunas, M., Gudas, S. “Knowledge-Based Approach to Business and IT Alignment Modelling”, Transformations in Business & Economics, Vol. 10, No 2 (23), pp. 60-73, ISSN 1648 - 4460, 2011

Publikacijos kituose ISI duomenų bazėse referuojamuose leidiniuose (proceedings ir kt)

Priimta spausdinti:

1. Veitaitė I., Ambraziūnas M., Lopata A. „Enterprise Model and ISO Standards Based Informations System’s Development Process“ // BIS 2014 International Workshops, Larnaca Cyprus May 22-23, 2014, Series: Lecture Notes in Business Information Processing.

Atspausdinta:

2. Lopata A., Ambraziūnas M. „Knowledge-Based MDA Approach“ // BIS 2011 International Workshops, Poznan, Poland, June 15-17, 2011, Series: Lecture Notes in Business Information Processing, Vol. 97, Abramowicz, Witold; Maciaszek, Leszek; Węcel, Krzysztof (Eds.), 2011, XV, 307p., Softcover ISBN: 978-3-642-25369-0
3. Lopata A. Ambraziūnas M. „MDA Compatible Knowledge - Based IS Engineering Approach“ // International Conference, AICI 2010, Sanya, China, October 23-24, 2010, Proceedings, Part II, Series: Lecture Notes in Computer Science, Vol. 6320, Subseries: Lecture Notes in Artificial Intelligence, Wang, Fu Lee; Deng, Hepu; Lei, Jingsheng (Eds.) 1st Edition., 2010, XXII, 386 p., Softcover ISBN: 978-3-642-16526-9
4. Lopata A., Ambraziunas M. „MDA Compatible Knowledge Based IS Development Process“ // BIS 2010 International Workshop, Berlin, Germany, May 3-5, 2010, Series: Lecture Notes in Business Information Processing, Vol. 57, Abramowicz, Witold; Tolksdorf, Robert; Wecel, Krzysztof (Eds.) 1st Edition., 2010, XVI, 324 p., Softcover ISBN: 978-3-642-15401-0

Publikacijos kituose recenzuojamuose mokslo leidiniuose:

1. Lopata A. Ambraziunas M. „Knowledge Subsystem’s Integration into MDA Based Forward and Reverse IS Engineering“ // Proceedings of 16th International Conference on Information and Software Technologies “Information Technologies 2010”. , Kaunas, Lithuania April 21-23, 2010 / Kaunas University of Technology, Kaunas: Technologija. 2010, p. 205-210, ISSN 2029-0020
2. Lopata A. Ambraziūnas M. „Veiklos žinių posisteme grindžiamas MDA metodas“ // Vytauto Didžiojo Universitetas: 15-osios tarpuniversitetinės magistrantų ir doktorantų konferencijos „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2010) medžiaga 2010.05.13 Kaunas. Kaunas: Vytauto Didžiojo Universitetas 2011. p. 5-9, ISSN 2029-249X

2. Žiniomis grindžiamo MDA metodo dalykinės programos prototipo programinis kodas

LoadedModelsDialog.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using UMLModelEntities;

namespace KBMDATool
{
   public partial class LoadedModelsDialog : Form
   {
      public LoadedModelsDialog()
      {
         InitializeComponent();
      }
      private void LoadedModelsDialog_Load(object sender, EventArgs e)
      {
         foreach (UMLModelInfo modelInfo in Program.AppManager.LoadedUMLModels)
         {
            DataGridViewRow row = new DataGridViewRow();
            row.CreateCells(this.dgvModels);
            row.SetValues(modelInfo.Name, modelInfo.ModelType, false);
            row.Tag = modelInfo;
            this.dgvModels.Rows.Add(row);
         }
      }

      private void dgvModels_CellContentClick(object sender, DataGridViewCellEventArgs e)
      {
         DataGridViewCheckBoxCell ch1 = new DataGridViewCheckBoxCell();
         ch1 =
(DataGridViewCheckBoxCell)this.dgvModels.Rows[this.dgvModels.CurrentRow.Index].Cells[2];
         if (ch1.Value == null)
            ch1.Value=false;
         switch (ch1.Value.ToString())
         {
            case "True":
               ch1.Value = false;
               break;
            case "False":
               ch1.Value = true;
               break;
         }
      }

      private void btnUMLValidate_Click(object sender, EventArgs e)
      {
         foreach (DataGridViewRow row in this.dgvModels.Rows.Cast<DataGridViewRow>().Where(row
=> Convert.ToBoolean(row.Cells[2].Value)))
         {
            Program.AppManager.SelectedForCompareUMLModel = (UMLModelInfo)row.Tag;
         }
         ValidateModelDialog validateModelDialog = new ValidateModelDialog();
```

```
            validateModelDialog.ShowDialog(this);
        }

        private void dgvModels_RowHeaderMouseDoubleClick(object sender,
DataGridViewCellMouseEventArgs e)
        {
            Program.AppManager.SelectedForViewUMLModel =
(UMLModelInfo)this.dgvModels.Rows[e.RowIndex].Tag;
            ViewModelElementsDialog modelParsingDialog = new ViewModelElementsDialog();
            modelParsingDialog.ShowDialog(this);
        }
    }
}
```

UseCaseModelParser.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using Constants;
using ModelParsers.Interfaces;
using ModelParsers.Responses;
using UMLModelEntities;

namespace ModelParsers.Parsers
{
    public class UseCaseModelParser : IReader
    {
        public IReaderResponse Parse(string filePath)
        {
            UMLModelParserResponse response = new UMLModelParserResponse();
            XmlDocument modelXml = new XmlDocument();
            modelXml.Load(filePath);
            XmlNode diagramsNode = modelXml.SelectSingleNode(".//mdOwnedDiagrams");
            XmlNode modelDataNode = modelXml.ChildNodes[1].ChildNodes[2];

            if (diagramsNode != null)
            {
                foreach (XmlNode diagramNode in diagramsNode.ChildNodes)
                {
                    if (diagramNode.ChildNodes[0].ChildNodes[2].InnerXml == "Use Case Diagram")
                    {
                        UMLModelInfo useCaseDiagramInfo = new UMLModelInfo()
                        {
                            Name = diagramNode.Attributes["name"].Value,
                            Id = diagramNode.Attributes["xmi:id"].Value,
                            ModelType = Enums.UMLModelType.UseCase,
                            Source = new UMLModelSourceInfo()
                            {
                                ModelName = filePath,
                                ParserType = Enums.ParserType.UseCase
                            }
                        };
                        List<string> tempObjIds =
GetModelObjectsIds(diagramNode.SelectSingleNode(".//mdOwnedViews"));

                        if (modelDataNode != null)
                        {
                            foreach (XmlNode objectNode in modelDataNode.ChildNodes)
```

```
                {
                    if (objectNode.Attributes["xmi:type"] != null &&
                        objectNode.Attributes["xmi:id"] != null &&
                        tempObjIds.Contains(objectNode.Attributes["xmi:id"].Value))
                    {
                        UMLModelObjectInfo umlModelObjectInfo = ParseUseCaseObject(objectNode);
                        if (umlModelObjectInfo != null)
                        {
                            useCaseDiagramInfo.ModelObjectsList.Add(umlModelObjectInfo);
                        }
                    }
                }
            }
            response.UMLModelsList.Add(useCaseDiagramInfo);
        }
    }
  }
  return response;
}

/// <summary>
/// Parses the uml use case objects from use case
/// </summary>
/// <param name="objectNode"></param>
/// <returns></returns>
private UMLModelObjectInfo ParseUseCaseObject(XmlNode objectNode)
{
  if (objectNode.Attributes["xmi:type"].Value.Equals("uml:UseCase"))
  {
    return new UMLUseCase()
    {
      Name = objectNode.Attributes["name"].Value,
      UMLObjectType = Enums.UMLModelObjectType.UseCase
    };
  }
  if (objectNode.Attributes["xmi:type"].Value.Equals("uml:Actor"))
  {
    return new UMLActor()
    {
      Name = objectNode.Attributes["name"].Value,
      UMLObjectType = Enums.UMLModelObjectType.Actor
    };
  }
  return null;
}

/// <summary>
/// Gets diagram related objects ids
/// </summary>
/// <param name="diagramNode"></param>
/// <returns></returns>
private List<string> GetModelObjectsIds(XmlNode diagramNode)
{
  List<string> tempObjIds = new List<string>();
  foreach (XmlNode objectNode in diagramNode.ChildNodes)
  {
    if (!objectNode.Attributes["elementClass"].Value.Equals("DiagramFrame"))
    {
      tempObjIds.Add(objectNode.FirstChild.Attributes["xmi:idref"].Value);
    }
  }
  return tempObjIds;
}
```

```
    }
}
```

Constants.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Constants
{
    /// <summary>
    /// Main class for application constants
    /// </summary>
    public static class Constants
    {
        /// <summary>
        /// Error messages constants
        /// </summary>
        public sealed class ErrorMessages
        {
            public const string ERROR_MESSAGE = "Error message";
            public const string INNER_ERROR_STACK_TRACE = "Inner error stack trace";
        }
    }

    /// <summary>
    /// Main class for application enums
    /// </summary>
    public class Enums
    {
        //----------------UML related enums----------------------------//

        /// <summary>
        /// Model type enum
        /// </summary>
        public enum ParserType
        {
            MagicDrawProject = 0,
            BlockDefinition = 1,
            UseCase = 2,
            Requirements = 3,
            Activity = 4,
            Class
        }

        /// <summary>
        /// Model type enum
        /// </summary>
        public enum UMLModelType
        {
            UseCase = 1,
            Requirements = 2,
            Activity = 3,
            Class = 4,
        }

        /// <summary>
        /// Model object type enum
        /// </summary>
        public enum UMLModelObjectType
```

```
        {
            Block = 1,
            Class = 2,
            UseCase= 3,
            Requirement = 4,
            Action = 5,
            Property = 6,
            Operation = 7,
            Actor = 8,
            Swimline = 9,
            Note = 10
        }

        //----------------Enterprise model related enums--------------//
        /// <summary>
        /// Model object type enum
        /// </summary>
        public enum EnterpriseModelObjectType
        {
            Function = 1,
            Process = 2,
            Actor = 3
        }

        //---------------Other Enums----------------------------------//
        /// <summary>
        /// ResponseStatus enums
        /// </summary>
        public enum ResponseStatus
        {
            OK = 1,
            ERROR = -1,
        }

        /// <summary>
        /// Validation results enums
        /// </summary>
        public enum ValidationResult
        {
            SUCCESS = 1,
            FAILURE = -1,
        }
    }
}
```

EnterpriseModelObjectInfo.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Constants;

namespace EnterpriseModelEntities
{
    public class EnterpriseModelObjectInfo
    {
        #region
        public string Name { get; set; }
        public Enums.EnterpriseModelObjectType EnterpriseModelObjectType { get; set; }
```

```
        public EnterpriseModelSourceInfo Source { get; set; }
        #endregion
    }
}
```

UMLModelObjectInfo.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Constants;

namespace UMLModelEntities
{
    public class UMLModelObjectInfo
    {
        #region Fields
        private List<UMLNote> notes;
        #endregion

        #region Properties
        public string Name { get; set; }
        public string Id { get; set; }
        public Enums.UMLModelObjectType UMLObjectType { get; set; }
        public UMLModelSourceInfo Source { get; set; }
        public List<UMLNote> Notes
        {
            get
            {
                if (this.notes == null)
                {
                    this.notes = new List<UMLNote>();
                }
                return this.notes;
            }
            set { this.notes = value; }
        }
        #endregion
    }
}
```

Tests.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Constants;
using ModelMapper;
using ModelMapper.Requests;
using ModelParsers;
using ModelParsers.Interfaces;
using ModelParsers.Parsers;
using ModelParsers.Responses;
using ModelValidator;
using ModelValidator.Requests;
using ModelValidator.Responses;
```

```
using NUnit.Framework;
using UMLModelEntities;

namespace UnitTesting
{
   public static class CommonTestingConstants
   {
      public const string ROOT = @"C:\Development\KBMDA_TestingData\";
      public const string USE_CASE_SUBDIR = @"UseCaseAndActivity";
      public const string USE_CASE_1_SUCCESS = @"KBMDA_UseCaseAndActivityTest_Success.xml";
      public const string USE_CASE_1_FAILURE = @"KBMDA_UseCaseAndActivityTest_Failure.xml";
   }

   [TestFixture]
   public class ParsingTests
   {
      [SetUp]
      public void Initialization()
      {
      }

      [Test]
      [Category("Parsing")]
      public void UseCaseParserTest()
      {
         IReader useCaseModelParser = ParserFactory.GetParser(Enums.ParserType.UseCase);
         IReaderResponse modelParserResponse =
useCaseModelParser.Parse(HelperRoutines.GetModelPath(CommonTestingConstants.ROOT,
CommonTestingConstants.USE_CASE_SUBDIR, CommonTestingConstants.USE_CASE_1_SUCCESS));
         Assert.Pass(string.Join(",", modelParserResponse.UMLModelsList[0].ModelObjectsList.Select(q =>
q.Name)));
      }

      [Test]
      [Category("Parsing")]
      public void ActivityParserTest()
      {
         IReader activityModelParser = ParserFactory.GetParser(Enums.ParserType.Activity);
         IReaderResponse modelParserResponse =
activityModelParser.Parse(HelperRoutines.GetModelPath(CommonTestingConstants.ROOT,
CommonTestingConstants.USE_CASE_SUBDIR, CommonTestingConstants.USE_CASE_1_SUCCESS));
         Assert.Pass(string.Join(",", modelParserResponse.UMLModelsList[0].ModelObjectsList.Select(q =>
q.Name)));
      }

      [Test]
      [Category("Parsing")]
      public void MagicDrawProjectParserTest()
      {
         IReader magicDrawProjectParser =
ParserFactory.GetParser(Enums.ParserType.MagicDrawProject);
         IReaderResponse projectParserResponse =
magicDrawProjectParser.Parse(HelperRoutines.GetModelPath(CommonTestingConstants.ROOT,
CommonTestingConstants.USE_CASE_SUBDIR, CommonTestingConstants.USE_CASE_1_SUCCESS));
         Assert.AreEqual(20, projectParserResponse.UMLModelsList.Count);
      }
   }

   [TestFixture]
   public class MappingTests
   {
      [SetUp]
      public void Initialization()
```

```
        {
        }

        [Test]
        [Category("Mapping")]
        public void MapUseCase()
        {
            IReader useCaseModelParser = ParserFactory.GetParser(Enums.ParserType.UseCase);
            IReaderResponse modelParserResponse =
useCaseModelParser.Parse(HelperRoutines.GetModelPath(CommonTestingConstants.ROOT,
CommonTestingConstants.USE_CASE_SUBDIR, CommonTestingConstants.USE_CASE_1_SUCCESS));
            UMLModelInfo useCaseModel = modelParserResponse.UMLModelsList.First(q => q.ModelType
== Enums.UMLModelType.UseCase);
            ModelMapperRequest mapRequest = new ModelMapperRequest();
            mapRequest.UMLModel = useCaseModel;
            Assert.IsTrue(Mapper.Convert(mapRequest).ResponseStatus == Enums.ResponseStatus.OK);
        }
    }

    [TestFixture]
    public class ValidationTests
    {
        [SetUp]
        public void Initialization()
        {
        }

        [Test]
        [Category("Validation")]
        public void ValidateUseCaseToActivitySuccess()
        {
            IReader useCaseModelParser = ParserFactory.GetParser(Enums.ParserType.UseCase);
            IReaderResponse useCaseModelParserResponse =
useCaseModelParser.Parse(HelperRoutines.GetModelPath(CommonTestingConstants.ROOT,
CommonTestingConstants.USE_CASE_SUBDIR, CommonTestingConstants.USE_CASE_1_FAILURE));
            UMLModelInfo useCaseModel = useCaseModelParserResponse.UMLModelsList[0];

            IReader activityModelParser = ParserFactory.GetParser(Enums.ParserType.Activity);
            IReaderResponse activityModelParserResponse =
activityModelParser.Parse(HelperRoutines.GetModelPath(CommonTestingConstants.ROOT,
CommonTestingConstants.USE_CASE_SUBDIR, CommonTestingConstants.USE_CASE_1_FAILURE));
            UMLModelInfo activityModel = activityModelParserResponse.UMLModelsList[0];

            ConsistencyValidationRequest consistencyUseCaseAgainstActivityValidationRequest = new
ConsistencyValidationRequest
            {
                MainModel = useCaseModel,
                MirrorModel = activityModel
            };
            ConsistencyValidatonResponse responseUseCaseAgainstActivity =
Validator.UseCaseAndActivity.CheckModelConsistencyUseCaseAgainstActivity(consistencyUseCaseAgai
nstActivityValidationRequest);
            Assert.IsTrue(responseUseCaseAgainstActivity.ValidationItems.Any(q => q.Result ==
Enums.ValidationResult.FAILURE));

            ConsistencyValidationRequest consistencyActivityAgaistUseCaseValidationRequest = new
ConsistencyValidationRequest
            {
                MainModel = activityModel,
                MirrorModel = useCaseModel
            };
```

```
        ConsistencyValidatonResponse responseActivityAgaistUseCase =
Validator.UseCaseAndActivity.CheckModelConsistencyUseCaseAgainstActivity(consistencyActivityAgais
tUseCaseValidationRequest);
        Assert.IsFalse(responseActivityAgaistUseCase.ValidationItems.Any(q => q.Result ==
Enums.ValidationResult.FAILURE));

        consistencyActivityAgaistUseCaseValidationRequest.SwitchModels();
        ConsistencyValidatonResponse responseFull =
Validator.UseCaseAndActivity.CheckModelConsistencyFull(consistencyActivityAgaistUseCaseValidation
Request);
        Assert.IsTrue(responseFull.ValidationItems.Any(q => q.Result ==
Enums.ValidationResult.FAILURE));
      }
   }

   #region Helper routines
   public static class HelperRoutines
   {
      /// <summary>
      /// creates path
      /// </summary>
      /// <param name="root"></param>
      /// <param name="subFolder"></param>
      /// <param name="fileName"></param>
      /// <returns></returns>
      public static string GetModelPath(string root, string subFolder, string fileName)
      {
         return Path.Combine(root, subFolder, fileName);
      }
   }
   #endregion

}
```

3. Žiniomis grindžiamo MDA metodo dalykinės programos prototipo duomenų bazės generavimo kodas

```
USE [master]
GO
/****** Object:  Database [EnterpriseModelDB]    Script Date: 07/28/2013 17:45:48 ******/
CREATE DATABASE [EnterpriseModelDB] ON  PRIMARY
( NAME = N'KBMDA', FILENAME = N'c:\Program Files\Microsoft SQL
Server\MSSQL10_50.SQLEXPRESS\MSSQL\DATA\KBMDA.mdf' , SIZE = 3072KB , MAXSIZE = UNLIMITED,
FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'KBMDA_log', FILENAME = N'c:\Program Files\Microsoft SQL
Server\MSSQL10_50.SQLEXPRESS\MSSQL\DATA\KBMDA_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB ,
FILEGROWTH = 10%)
GO
ALTER DATABASE [EnterpriseModelDB] SET COMPATIBILITY_LEVEL = 100
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [EnterpriseModelDB].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [EnterpriseModelDB] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET ANSI_NULLS OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET ANSI_PADDING OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET ARITHABORT OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET AUTO_CLOSE OFF
GO
```

```
ALTER DATABASE [EnterpriseModelDB] SET AUTO_CREATE_STATISTICS ON
GO
ALTER DATABASE [EnterpriseModelDB] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [EnterpriseModelDB] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET CURSOR_DEFAULT  GLOBAL
GO
ALTER DATABASE [EnterpriseModelDB] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET  DISABLE_BROKER
GO
ALTER DATABASE [EnterpriseModelDB] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET TRUSTWORTHY OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [EnterpriseModelDB] SET READ_COMMITTED_SNAPSHOT OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET HONOR_BROKER_PRIORITY OFF
GO
ALTER DATABASE [EnterpriseModelDB] SET  READ_WRITE
GO
ALTER DATABASE [EnterpriseModelDB] SET RECOVERY SIMPLE
GO
ALTER DATABASE [EnterpriseModelDB] SET  MULTI_USER
GO
ALTER DATABASE [EnterpriseModelDB] SET PAGE_VERIFY CHECKSUM
GO
ALTER DATABASE [EnterpriseModelDB] SET DB_CHAINING OFF
GO
USE [EnterpriseModelDB]
GO
/****** Object:  Table [dbo].[PackageObject]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PackageObject](
	[Id] [int] NOT NULL,
	[PackageId] [int] NOT NULL,
	[ObjectId] [int] NOT NULL,
 CONSTRAINT [PK_PackageObject] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Package]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Package](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[Description] [nvarchar](255) NOT NULL,
 CONSTRAINT [PK_Package] PRIMARY KEY CLUSTERED
(
	[Id] ASC
```

```
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[MaterialFlow]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MaterialFlow](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[Description] [nvarchar](255) NOT NULL,
	[SourceId] [int] NOT NULL,
 CONSTRAINT [PK_MaterialFlow] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[InformationalFlow]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[InformationalFlow](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[Description] [nvarchar](255) NOT NULL,
	[SourceId] [int] NOT NULL,
 CONSTRAINT [PK_InformationalFlow] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FunctionType]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FunctionType](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
 CONSTRAINT [PK_FunctionType] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[BusinessRule]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[BusinessRule](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[Description] [nvarchar](1000) NOT NULL,
	[SourceId] [int] NOT NULL,
 CONSTRAINT [PK_BusinessRule] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Attribute]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE TABLE [dbo].[Attribute](
	[Id] [int] NOT NULL,
	[ObjectId] [int] NOT NULL,
	[ObjectType] [int] NOT NULL,
	[Name] [nvarchar](50) NOT NULL,
	[Value] [nvarchar](255) NOT NULL,
 CONSTRAINT [PK_Attribute] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[ActorType]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ActorType](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
 CONSTRAINT [PK_ActorType] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[EventType]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[EventType](
	[Id] [int] NOT NULL,
	[Name] [nchar](10) NOT NULL,
 CONSTRAINT [PK_EventType] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[EventLog]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[EventLog](
	[Id] [int] NOT NULL,
	[SourceId] [int] NOT NULL,
	[ObjectId] [int] NOT NULL,
	[ObjectType] [int] NOT NULL,
	[Message] [nvarchar](255) NOT NULL,
	[DateTime] [date] NOT NULL,
 CONSTRAINT [PK_EventLog] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[ProcessType]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ProcessType](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
 CONSTRAINT [PK_ProcessType] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
```

```
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[SourceType]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[SourceType](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NULL,
 CONSTRAINT [PK_SourceType] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Source]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Source](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[TypeId] [int] NOT NULL,
	[Description] [nvarchar](255) NOT NULL,
 CONSTRAINT [PK_Source] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Event]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Event](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[Description] [nvarchar](255) NOT NULL,
	[TypeId] [int] NOT NULL,
	[SourceId] [int] NOT NULL,
 CONSTRAINT [PK_Event] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Process]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Process](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[SourceId] [int] NOT NULL,
	[TypeId] [int] NOT NULL,
	[ParentId] [int] NOT NULL,
 CONSTRAINT [PK_Process] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Function]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE TABLE [dbo].[Function](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[SourceId] [int] NOT NULL,
	[TypeId] [int] NOT NULL,
	[ParentId] [int] NOT NULL,
 CONSTRAINT [PK_Function] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Actor]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Actor](
	[Id] [int] NOT NULL,
	[Name] [nvarchar](255) NOT NULL,
	[Description] [nvarchar](255) NOT NULL,
	[SourceId] [int] NOT NULL,
	[ActivityType] [int] NOT NULL,
	[TypeId] [int] NOT NULL,
 CONSTRAINT [PK_Actor] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[ProcessFlow]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ProcessFlow](
	[Id] [int] NOT NULL,
	[ProcessId] [int] NOT NULL,
	[MaterialFlowId] [int] NOT NULL,
	[DirectionIn] [bit] NOT NULL,
 CONSTRAINT [PK_ProcessFlow] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FunctionProcess]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FunctionProcess](
	[Id] [int] NOT NULL,
	[FunctionId] [int] NOT NULL,
	[ProcessId] [int] NOT NULL,
 CONSTRAINT [PK_FunctionProcess] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FunctionFlow]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FunctionFlow](
	[Id] [int] NOT NULL,
	[FunctionId] [int] NOT NULL,
	[InformationalFlowId] [int] NOT NULL,
	[DirectionIn] [bit] NOT NULL,
```

```
 CONSTRAINT [PK_FunctionFlow] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[ActorProcess]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ActorProcess](
	[Id] [int] NOT NULL,
	[ActorId] [int] NOT NULL,
	[ProcessId] [int] NOT NULL,
 CONSTRAINT [PK_ActorProcess] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[ActorFunction]    Script Date: 07/28/2013 17:45:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ActorFunction](
	[Id] [int] NOT NULL,
	[ActorId] [int] NOT NULL,
	[FunctionId] [int] NOT NULL,
 CONSTRAINT [PK_ActorFunction] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  ForeignKey [FK_Source_SourceType]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Source]  WITH CHECK ADD  CONSTRAINT [FK_Source_SourceType] FOREIGN KEY([TypeId])
REFERENCES [dbo].[SourceType] ([Id])
GO
ALTER TABLE [dbo].[Source] CHECK CONSTRAINT [FK_Source_SourceType]
GO
/****** Object:  ForeignKey [FK_Event_EventType]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Event]  WITH CHECK ADD  CONSTRAINT [FK_Event_EventType] FOREIGN KEY([TypeId])
REFERENCES [dbo].[EventType] ([Id])
GO
ALTER TABLE [dbo].[Event] CHECK CONSTRAINT [FK_Event_EventType]
GO
/****** Object:  ForeignKey [FK_Event_Source]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Event]  WITH CHECK ADD  CONSTRAINT [FK_Event_Source] FOREIGN KEY([SourceId])
REFERENCES [dbo].[Source] ([Id])
GO
ALTER TABLE [dbo].[Event] CHECK CONSTRAINT [FK_Event_Source]
GO
/****** Object:  ForeignKey [FK_Process_Process]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Process]  WITH CHECK ADD  CONSTRAINT [FK_Process_Process] FOREIGN KEY([ParentId])
REFERENCES [dbo].[Process] ([Id])
GO
ALTER TABLE [dbo].[Process] CHECK CONSTRAINT [FK_Process_Process]
GO
/****** Object:  ForeignKey [FK_Process_ProcessType]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Process]  WITH CHECK ADD  CONSTRAINT [FK_Process_ProcessType] FOREIGN KEY([TypeId])
REFERENCES [dbo].[ProcessType] ([Id])
GO
ALTER TABLE [dbo].[Process] CHECK CONSTRAINT [FK_Process_ProcessType]
GO
/****** Object:  ForeignKey [FK_Process_Source]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Process]  WITH CHECK ADD  CONSTRAINT [FK_Process_Source] FOREIGN KEY([SourceId])
REFERENCES [dbo].[Source] ([Id])
GO
ALTER TABLE [dbo].[Process] CHECK CONSTRAINT [FK_Process_Source]
GO
```

```
/****** Object:  ForeignKey [FK_Function_Function]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Function]  WITH CHECK ADD  CONSTRAINT [FK_Function_Function] FOREIGN KEY([ParentId])
REFERENCES [dbo].[Function] ([Id])
GO
ALTER TABLE [dbo].[Function] CHECK CONSTRAINT [FK_Function_Function]
GO
/****** Object:  ForeignKey [FK_Function_FunctionType]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Function]  WITH CHECK ADD  CONSTRAINT [FK_Function_FunctionType] FOREIGN
KEY([TypeId])
REFERENCES [dbo].[FunctionType] ([Id])
GO
ALTER TABLE [dbo].[Function] CHECK CONSTRAINT [FK_Function_FunctionType]
GO
/****** Object:  ForeignKey [FK_Function_Source]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Function]  WITH CHECK ADD  CONSTRAINT [FK_Function_Source] FOREIGN KEY([SourceId])
REFERENCES [dbo].[Source] ([Id])
GO
ALTER TABLE [dbo].[Function] CHECK CONSTRAINT [FK_Function_Source]
GO
/****** Object:  ForeignKey [FK_Actor_ActorType]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Actor]  WITH CHECK ADD  CONSTRAINT [FK_Actor_ActorType] FOREIGN KEY([TypeId])
REFERENCES [dbo].[ActorType] ([Id])
GO
ALTER TABLE [dbo].[Actor] CHECK CONSTRAINT [FK_Actor_ActorType]
GO
/****** Object:  ForeignKey [FK_Actor_Source]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[Actor]  WITH CHECK ADD  CONSTRAINT [FK_Actor_Source] FOREIGN KEY([SourceId])
REFERENCES [dbo].[Source] ([Id])
GO
ALTER TABLE [dbo].[Actor] CHECK CONSTRAINT [FK_Actor_Source]
GO
/****** Object:  ForeignKey [FK_ProcessFlow_MaterialFlow]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[ProcessFlow]  WITH CHECK ADD  CONSTRAINT [FK_ProcessFlow_MaterialFlow] FOREIGN
KEY([MaterialFlowId])
REFERENCES [dbo].[MaterialFlow] ([Id])
GO
ALTER TABLE [dbo].[ProcessFlow] CHECK CONSTRAINT [FK_ProcessFlow_MaterialFlow]
GO
/****** Object:  ForeignKey [FK_ProcessFlow_Process]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[ProcessFlow]  WITH CHECK ADD  CONSTRAINT [FK_ProcessFlow_Process] FOREIGN
KEY([ProcessId])
REFERENCES [dbo].[Process] ([Id])
GO
ALTER TABLE [dbo].[ProcessFlow] CHECK CONSTRAINT [FK_ProcessFlow_Process]
GO
/****** Object:  ForeignKey [FK_FunctionProcess_Function]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[FunctionProcess]  WITH CHECK ADD  CONSTRAINT [FK_FunctionProcess_Function] FOREIGN
KEY([FunctionId])
REFERENCES [dbo].[Function] ([Id])
GO
ALTER TABLE [dbo].[FunctionProcess] CHECK CONSTRAINT [FK_FunctionProcess_Function]
GO
/****** Object:  ForeignKey [FK_FunctionProcess_Process]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[FunctionProcess]  WITH CHECK ADD  CONSTRAINT [FK_FunctionProcess_Process] FOREIGN
KEY([ProcessId])
REFERENCES [dbo].[Process] ([Id])
GO
ALTER TABLE [dbo].[FunctionProcess] CHECK CONSTRAINT [FK_FunctionProcess_Process]
GO
/****** Object:  ForeignKey [FK_FunctionFlow_Function]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[FunctionFlow]  WITH CHECK ADD  CONSTRAINT [FK_FunctionFlow_Function] FOREIGN
KEY([FunctionId])
REFERENCES [dbo].[Function] ([Id])
GO
ALTER TABLE [dbo].[FunctionFlow] CHECK CONSTRAINT [FK_FunctionFlow_Function]
GO
/****** Object:  ForeignKey [FK_FunctionFlow_InformationalFlow]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[FunctionFlow]  WITH CHECK ADD  CONSTRAINT [FK_FunctionFlow_InformationalFlow]
FOREIGN KEY([InformationalFlowId])
REFERENCES [dbo].[InformationalFlow] ([Id])
GO
ALTER TABLE [dbo].[FunctionFlow] CHECK CONSTRAINT [FK_FunctionFlow_InformationalFlow]
GO
/****** Object:  ForeignKey [FK_ActorProcess_Actor]    Script Date: 07/28/2013 17:45:50 ******/
```

```
ALTER TABLE [dbo].[ActorProcess]  WITH CHECK ADD  CONSTRAINT [FK_ActorProcess_Actor] FOREIGN
KEY([ActorId])
REFERENCES [dbo].[Actor] ([Id])
GO
ALTER TABLE [dbo].[ActorProcess] CHECK CONSTRAINT [FK_ActorProcess_Actor]
GO
/****** Object:  ForeignKey [FK_ActorProcess_Process]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[ActorProcess]  WITH CHECK ADD  CONSTRAINT [FK_ActorProcess_Process] FOREIGN
KEY([ProcessId])
REFERENCES [dbo].[Process] ([Id])
GO
ALTER TABLE [dbo].[ActorProcess] CHECK CONSTRAINT [FK_ActorProcess_Process]
GO
/****** Object:  ForeignKey [FK_ActorFunction_Actor]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[ActorFunction]  WITH CHECK ADD  CONSTRAINT [FK_ActorFunction_Actor] FOREIGN
KEY([ActorId])
REFERENCES [dbo].[Actor] ([Id])
GO
ALTER TABLE [dbo].[ActorFunction] CHECK CONSTRAINT [FK_ActorFunction_Actor]
GO
/****** Object:  ForeignKey [FK_ActorFunction_Function]    Script Date: 07/28/2013 17:45:50 ******/
ALTER TABLE [dbo].[ActorFunction]  WITH CHECK ADD  CONSTRAINT [FK_ActorFunction_Function] FOREIGN
KEY([FunctionId])
REFERENCES [dbo].[Function] ([Id])
GO
ALTER TABLE [dbo].[ActorFunction] CHECK CONSTRAINT [FK_ActorFunction_Function]
GO
```

4. Testavimo XMI failas su veiklų modelio elementais

```
-<mdElement visibility="public" name="OrderFoodActivity"
xmi:id="_17_0_6_154803fb_1372966312328_498435_2228"
ownerOfDiagram="_17_0_6_154803fb_1372966312328_813895_2230"
elementClass="Diagram"
context="_17_0_6_154803fb_1372966312328_813895_2230">-<mdElement
xmi:id="_17_0_6_154803fb_1372966312328_963339_2229"
elementClass="DiagramPresentationElement"><elementID
xmi:idref="_17_0_6_154803fb_1372966312328_498435_2228"/><type>Acti
vity Diagram</type><umlType>Activity Diagram</umlType><zoomFactor
xmi:value="1.0"/><diagramOpened
xmi:value="true"/><diagramFrameInitialSizeSet
xmi:value="true"/><requiredFeature>;UML_Standard_Profile.xml</requiredF
eature><diagramWindowBounds>3, 25, 1244,
580</diagramWindowBounds><diagramScrollPositionX
xmi:value="0"/><diagramScrollPositionY xmi:value="0"/><maximized
xmi:value="false"/><active xmi:value="true"/>-<mdOwnedViews>-
<mdElement xmi:id="_17_0_6_154803fb_1372966312333_732357_2242"
elementClass="DiagramFrame"><elementID
xmi:idref="_17_0_6_154803fb_1372966312328_498435_2228"/><geometry>
5, 5, 1090, 526</geometry><compartment
compartmentID="TAGGED_VALUES"/></mdElement>-<mdElement
xmi:id="_17_0_6_154803fb_1372966330411_198815_2259"
elementClass="Swimlane"><geometry>35, 46, 432, 187</geometry>-
<mdOwnedViews>-<mdElement
xmi:id="_17_0_6_154803fb_1372966330411_350053_2260"
elementClass="SwimlaneHeader"><elementID
xmi:idref="_17_0_6_154803fb_1372966330394_278790_2251"/><geometry>
```

35, 46, 144, 21</geometry><compartment
compartmentID="TAGGED_VALUES"/><verticalCenterline>_17_0_6_1548
03fb_1372966330408_757115_2257</verticalCenterline><rotated
xmi:value="false"/></mdElement>-<mdElement
xmi:id="_17_0_6_154803fb_1372966330401_903628_2253"
elementClass="SwimlaneCell"><geometry>35, 67, 144, 166</geometry>-
<mdOwnedViews>-<mdElement
xmi:id="_17_0_6_154803fb_1372966397877_477205_2297"
elementClass="PseudoNode"><elementID
xmi:idref="_17_0_6_154803fb_1372966397874_579055_2294"/><geometry>
91, 82, 18, 18</geometry><compartment
compartmentID="TAGGED_VALUES"/><verticalCenterline>_17_0_6_1548
03fb_1372966378388_901520_2283</verticalCenterline><horizontalCenterlin
e>_17_0_6_154803fb_1372966397875_639750_2296</horizontalCenterline>
</mdElement>-<mdElement
xmi:id="_17_0_6_154803fb_1372966400692_810244_2303"
elementClass="ControlFlow"><elementID
xmi:idref="_17_0_6_154803fb_1372966400665_339020_2301"/><linkFirstE
ndID
xmi:idref="_17_0_6_154803fb_1372966378387_205726_2282"/><linkSecon
dEndID
xmi:idref="_17_0_6_154803fb_1372966397877_477205_2297"/><geometry>
100, 124; 100, 100; </geometry><compartment
compartmentID="TAGGED_VALUES"/><nameVisible
xmi:value="true"/></mdElement>-<mdElement
xmi:id="_17_0_6_154803fb_1372966406398_616212_2320"
elementClass="ControlFlow"><elementID
xmi:idref="_17_0_6_154803fb_1372966406391_246140_2318"/><linkFirstE
ndID
xmi:idref="_17_0_6_154803fb_1372966404107_685974_2306"/><linkSecon
dEndID
xmi:idref="_17_0_6_154803fb_1372966378387_205726_2282"/><geometry>
200, 135; 150, 135; </geometry><compartment
compartmentID="TAGGED_VALUES"/><nameVisible
xmi:value="true"/></mdElement>-<mdElement
xmi:id="_17_0_6_154803fb_1372966378387_205726_2282"
elementClass="CallBehaviorAction"><elementID
xmi:idref="_17_0_6_154803fb_1372966378385_20736_2281"/><geometry>5
0, 124, 100, 22</geometry><compartment
compartmentID="TAGGED_VALUES"/><verticalCenterline>_17_0_6_1548
03fb_1372966378388_901520_2283</verticalCenterline><horizontalCenterlin
e>_17_0_6_154803fb_1372966484674_207804_2354</horizontalCenterline>
</mdElement></mdOwnedViews></mdElement>-<mdElement
xmi:id="_17_0_6_154803fb_1372966350205_401114_2279"
elementClass="SwimlaneHeader">i:idref="_17_0_6_154803fb_13729663365
42_734123_2276"/><geometry>179, 46, 142, 21</geometry><compartment

compartmentID="TAGGED_VALUES"/><verticalCenterline>_17_0_6_154803fb_1372966453270_103830_2349</verticalCenterline><horizontalCenterline>_17_0_6_154803fb_1372966484674_207804_2354</horizontalCenterline></mdElement></mdOwnedViews></mdElement>-<mdElement xmi:id="_17_0_6_154803fb_1372966336545_977545_2278" elementClass="SwimlaneCell"><geometry>179, 67, 142, 166</geometry>-<mdOwnedViews>-<mdElement xmi:id="_17_0_6_154803fb_1372966433688_414023_2338" elementClass="ControlFlow"><elementID xmi:idref="_17_0_6_154803fb_1372966433680_131428_2336"/><linkFirstEndID xmi:idref="_17_0_6_154803fb_1372966429482_261552_2323"/><linkSecondEndID xmi:idref="_17_0_6_154803fb_1372966404107_685974_2306"/><geometry>344, 135; 300, 135; </geometry><compartment compartmentID="TAGGED_VALUES"/><nameVisible xmi:value="true"/></mdElement>-<mdElement xmi:id="_17_0_6_154803fb_1372966404107_685974_2306" elementClass="CallBehaviorAction"><elementID xmi:idref="_17_0_6_154803fb_1372966404105_119466_2305"/><geometry>200, 124, 100, 22</geometry><compartment compartmentID="TAGGED_VALUES"/><verticalCenterline>_17_0_6_154803fb_1372966336544_188776_2277</verticalCenterline><horizontalCenterline>_17_0_6_154803fb_1372966484674_207804_2354</horizontalCenterline></mdElement></mdOwnedViews></mdElement></mdOwnedViews></mdElement></mdOwnedViews></mdElement></mdElement></mdOwnedDiagrams>

5. Pašto siuntų stebėjimo programėlės programinis kodas

Modulis com.posttracker.entities

AddTrackingNumberActivity.java

```
package com.posttracker.activities;
import java.util.Calendar;
import com.posttracker.activities.R;
import com.posttracker.entities.Constants;
import com.posttracker.entities.ParcelObject;
import com.posttracker.entities.ParcelObjectsList;
import com.posttracker.iodata.DataManager;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class AddTrackingNumberActivity extends Activity {
	private int btnLoadCount = 0;
		@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_add_tracking_number);
		final EditText txtParcelName = (EditText) findViewById(R.id.txtParcelName);
		final EditText txtTrackingNumber = (EditText) findViewById(R.id.txtTrackingNumberText);
		final Button button = (Button) findViewById(R.id.btnSave);
	button.setOnClickListener(new View.OnClickListener() {
```

```
        public void onClick(View v) {
                ParcelObjectsList list = GetData();
                ParcelObject parcel = new ParcelObject();
                if(Validate(txtParcelName, txtTrackingNumber))
                {
                parcel.TrackingNumber(capitalizeString(txtTrackingNumber.getText().toString()));
                parcel.ParcelName(txtParcelName.getText().toString());
                parcel.OriginCountry(parcel.TrackingNumber().substring(11,13));
                parcel.ParcelType(parcel.TrackingNumber().substring(0,2));

                        Calendar calendar = Calendar.getInstance();
                calendar.add(Calendar.MINUTE, -10);
                parcel.StatusUpdateDate(calendar.getTime());
                list.Parcels().add(parcel);
        SaveData(list);
        Intent resultIntent = new Intent();
        resultIntent.putExtra("item", parcel);
        setResult(Activity.RESULT_OK, resultIntent);
        finish();
                }
    }
});
        }

        @Override
        public Object onRetainNonConfigurationInstance() {
           return btnLoadCount;
        }
        private void SaveData(ParcelObjectsList list)
        {
        DataManager.SaveData(this, list);
        }
        private ParcelObjectsList GetData()
        {
        return DataManager.GetData(this);
        }
        private boolean Validate(EditText txtParcelName, EditText txtTrackingNumber)
        {
                if(txtParcelName.getText().toString().length() == 0 )
                {
                        txtParcelName.setError(Constants.PARCEL_NAME_ERROR);
                        return false;
                }
                if(txtTrackingNumber.getText().toString().length() != 13 )
                {
                        txtTrackingNumber.setError(Constants.TRACKING_NUMBER_ERROR);
                        return false;
                }
                return true;
        }
        private String capitalizeString(String value)
        {
           String string = value;
           String capitalizedString = "";
           System.out.println(string);
           for(int i = 0; i < string.length(); i++)
           {
              char ch = string.charAt(i);
              ch = Character.toUpperCase(ch);
              capitalizedString += ch;
           }
           return capitalizedString;
        }
}
```

ListItemActivity.java

```
package com.posttracker.activities;
import java.util.ArrayList;
import com.posttracker.activities.R;
import com.posttracker.adapters.OriginCountryStatusAdapter;
import com.posttracker.adapters.StatusListBaseAdapter;
import com.posttracker.entities.ParcelObject;
import com.posttracker.entities.StatusObject;
import android.os.Bundle;
```

```
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Intent;
import android.graphics.drawable.Drawable;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.ListView;
import android.widget.TabHost;
import android.widget.TabHost.TabSpec;
import android.widget.TextView;
public class ListItemActivity extends Activity {
	ListView lvDestinationParcelStatus;
	ListView lvOriginParcelStatus;
	String trackingNumber;
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_list_item);
		Intent i = getIntent();
		final ParcelObject myParcelableObject = (ParcelObject) i.getParcelableExtra("item");

		TabHost tabHost=(TabHost)findViewById(R.id.tabHost);
		tabHost.setup();

		TabSpec spec1= tabHost.newTabSpec("tbLithuaniaPost");
		spec1.setIndicator("", GetPostLogo("LT"));
		spec1.setContent(R.id.tbLithuaniaPost);

		TabSpec spec2=tabHost.newTabSpec("tbOriginCountryPost");
		spec2.setIndicator("", GetPostLogo(myParcelableObject.TrackingNumber().substring(11,13)));
		spec2.setContent(R.id.tbOriginCountryPost);

		tabHost.addTab(spec1);
		tabHost.addTab(spec2);

		TextView tvParcelName = (TextView)findViewById(R.id.tvParcelName);
		TextView tvTrackingNumber = (TextView)findViewById(R.id.tvTrackingNumber);
		this.lvDestinationParcelStatus = (ListView)findViewById(R.id.lvDestinationStatus);
		this.lvOriginParcelStatus = (ListView)findViewById(R.id.lvOriginCountryStatus);

		this.trackingNumber = myParcelableObject.TrackingNumber();
		tvParcelName.setText(myParcelableObject.ParcelName());
		tvTrackingNumber.setText(trackingNumber);
		this.lvDestinationParcelStatus.setAdapter(new StatusListBaseAdapter(this,
				(ArrayList<StatusObject>)
myParcelableObject.DestinationCountryStatusList().Statuses()));
		this.lvOriginParcelStatus.setAdapter(new OriginCountryStatusAdapter(this,
				(ArrayList<StatusObject>)
myParcelableObject.OriginCountryStatusList().Statuses()));
	}

	private void ShowDialog(String message)
	{
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    LayoutInflater inflater = this.getLayoutInflater();
    View dialogView = inflater.inflate(R.layout.tracking_number_dialog, null);
    TextView tr = (TextView)dialogView.findViewById(R.id.lblTrackingNumberSingle);
    tr.setText(message);
    builder.setView(dialogView);
    AlertDialog dialog = builder.create();
    dialog.show();
    dialog.getWindow().setLayout(420,200);
	}

  @Override
  public boolean onCreateOptionsMenu(Menu menu)
  {
    MenuInflater menuInflater = getMenuInflater();
    menuInflater.inflate(R.layout.list_item_menu, menu);
    return true;
  }
```

```
   @Override
   public boolean onOptionsItemSelected(MenuItem item)
   {
      switch (item.getItemId())
      {
                  case R.id.menu_zoom_tracking_number:
                  {
                        ShowDialog(this.trackingNumber);
                     return true;
                  }
                  default:
                     return super.onOptionsItemSelected(item);
      }
   }

            public Drawable GetPostLogo(String countryCode)
            {
                        if(countryCode.equalsIgnoreCase("HK"))
                        {
                                    return getResources().getDrawable(R.drawable.icon_honkong_post_tab);
                        }
                        if(countryCode.equalsIgnoreCase("US"))
                        {
                                    return getResources().getDrawable(R.drawable.icon_us_post_tab);
                        }
                        return getResources().getDrawable(R.drawable.icon_lithuania_post_tab);
             }
}
```

MainActivity.java

```
package com.posttracker.activities;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.view.*;
import android.view.ContextMenu.ContextMenuInfo;
import java.util.ArrayList;
import java.util.Calendar;
import com.posttracker.activities.R;
import com.posttracker.adapters.ParcelListBaseAdapter;
import com.posttracker.entities.Constants;
import com.posttracker.entities.ParcelObject;
import com.posttracker.entities.ParcelObjectsList;
import com.posttracker.iodata.DataManager;

public class MainActivity extends Activity {

            TextView display;
            ListView lvParcels;
            ArrayList<ParcelObject> parcels;
            private ProgressDialog progressDialog;
            private ParcelListBaseAdapter parcellListAdapter;

            @Override
            protected void onCreate(Bundle savedInstanceState) {
                        super.onCreate(savedInstanceState);
                        progressDialog = ProgressDialog.show(MainActivity.this, "", Constants.LOADING_TEXT);
                        setContentView(R.layout.activity_main);
                        lvParcels = (ListView)findViewById(R.id.lvParcels);
                        parcels = GetParcels();
                        registerForContextMenu(lvParcels);
                        progressDialog.dismiss();
                        parcellListAdapter = new ParcelListBaseAdapter(this, parcels);
                        this.lvParcels.setAdapter(parcellListAdapter);
                        this.lvParcels.setOnItemClickListener(new OnItemClickListener() {
      public void onItemClick(AdapterView<?> parent, View view,
         int position, long id) {
            Object o = lvParcels.getItemAtPosition(position);
```

```
            ParcelObject obj_itemDetails = (ParcelObject)o;
          Intent i = new Intent(getApplicationContext(), ListItemActivity.class);
          i.putExtra("item", obj_itemDetails);
          startActivity(i);
        }
      });
            }

    @Override
    protected void onStop() {
       super.onStop();
       ParcelObjectsList list = new ParcelObjectsList();
       list.Parcels(parcels);
       SaveData(list);
    }

            @Override
            public void onActivityResult(int requestCode, int resultCode, Intent data) {
              super.onActivityResult(requestCode, resultCode, data);
              switch(requestCode) {
                case (10) : {
                  if (resultCode == Activity.RESULT_OK) {
                              ParcelObject currentParcel = (ParcelObject)data.getParcelableExtra("item");
                              boolean exists = false;
                              //Work around for orientation change, cause then duplicated item is added
                              for(ParcelObject existingParcel: parcels)
                              {
                                        if(existingParcel.ParcelName().equalsIgnoreCase(currentParcel.ParcelName()) &&

existingParcel.TrackingNumber().equalsIgnoreCase(currentParcel.TrackingNumber()))
                                        {
                                                  exists = true;
                                                  break;
                                        }
                              }
                              if(!exists)
                              {
                                        parcels.add(currentParcel);
                                        this.parcellListAdapter.notifyDataSetChanged();
                              }
                  }
                  break;
                }
              }
            }

    // Initiating Menu XML file (menu.xml)
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
       MenuInflater menuInflater = getMenuInflater();
       menuInflater.inflate(R.layout.menu, menu);
       return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {

       switch (item.getItemId())
       {
                  case R.id.menu_add:
                  {
                     Intent i = new Intent(getApplicationContext(), AddTrackingNumberActivity.class);
                     startActivityForResult(i, 10);
                     return true;
                  }
                  case R.id.menu_refresh:
                  {
                              Calendar calendar = Calendar.getInstance();
                              calendar.add(Calendar.MINUTE, -5);
                        for (ParcelObject parcel : parcels)
                        {
                              parcel.StatusUpdateDate(calendar.getTime());
                        }
```

```
                        this.parcellListAdapter.notifyDataSetChanged();
                        return true;
                    }
                    default:
                       return super.onOptionsItemSelected(item);
        }
    }

            private ArrayList<ParcelObject> GetParcels(){
                        ArrayList<ParcelObject> results = new ArrayList<ParcelObject>();
                        ParcelObjectsList list = DataManager.GetData(this);

                        if(list != null && list.Parcels().size() != 0)
                        {
                                    for(ParcelObject parcel : list.Parcels())
                                    {
                                                results.add(parcel);
                                    }
                        }
                        return results;
            }

            private void SaveData(ParcelObjectsList list)
            {
            DataManager.SaveData(this, list);
            }

            @Override
    public void onCreateContextMenu(ContextMenu menu, View v,ContextMenuInfo menuInfo) {
                          super.onCreateContextMenu(menu, v, menuInfo);
                           menu.add(0, v.getId(), 0, "Delete");
            }

            @Override
    public boolean onContextItemSelected(MenuItem item) {
                        AdapterView.AdapterContextMenuInfo info =
(AdapterView.AdapterContextMenuInfo)item.getMenuInfo();
                        if(item.getTitle()=="Delete")
                        {
                                    DeleteParcel(info.position);
                        }
                  else
                  {
                        return false;
            }
                    return true;
            }

            public void ShowMessageStatusUpdateMessage()
            {
       Toast.makeText(this, "Cannot delete, object status is updating", Toast.LENGTH_SHORT).show();
    }

            public void DeleteParcel(int id){
                        Object o = lvParcels.getItemAtPosition(id);
                        ParcelObject parcel = (ParcelObject)o;
                        if(parcel.DestinationCountryStatusList().LastStatusEvent().Description() == Constants.UPDATE_TEXT)
                        {
                                    ShowMessageStatusUpdateMessage();
                        }
                        else
                        {
                                    parcels.remove(parcel);
                                    this.parcellListAdapter.notifyDataSetChanged();
                        }
    }
}
```

Modulis com.posttracker.adapters

OriginCountryStatusAdapter.java

```
package com.posttracker.adapters;
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
import com.posttracker.activities.R;
import com.posttracker.entities.Constants;
import com.posttracker.entities.StatusObject;
import com.posttracker.entities.StatusViewHolder;

public class OriginCountryStatusAdapter extends BaseAdapter {
	private static ArrayList<StatusObject> statusArrayList;

	private LayoutInflater liMain;

	public OriginCountryStatusAdapter(Context context, ArrayList<StatusObject> statusArrayList) {
		OriginCountryStatusAdapter.statusArrayList = statusArrayList;
		this.liMain = LayoutInflater.from(context);
	}

	public int getCount() {
	 return OriginCountryStatusAdapter.statusArrayList.size();
	}

	public Object getItem(int position) {
	 return OriginCountryStatusAdapter.statusArrayList.get(position);
	}

	public long getItemId(int position) {
	 return position;
	}

	public View getView(final int index, View convertView, ViewGroup parent) {
	 final StatusViewHolder holder;
	 if (convertView == null) {
		convertView = liMain.inflate(R.layout.destination_list_item, null);
		holder = new StatusViewHolder();
		holder.tvEvent = (TextView) convertView.findViewById(R.id.tvEvent);
		holder.tvLocation = (TextView) convertView.findViewById(R.id.tvLocation);
		holder.tvDate = (TextView) convertView.findViewById(R.id.tvDate);
		convertView.setTag(holder);
	 }
	 else {
		holder = (StatusViewHolder)convertView.getTag();
	 }
	 holder.tvEvent.setText(statusArrayList.get(index).Description());
	 holder.tvLocation.setText(statusArrayList.get(index).Location());
	 SimpleDateFormat sdfDate = new SimpleDateFormat(Constants.DATE_FORMAT);
	 holder.tvDate.setText(sdfDate.format(statusArrayList.get(index).date()));

	 return convertView;
	}
}
```

ParcelListBaseAdapter.java

```
package com.posttracker.adapters;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import com.posttracker.activities.R;
import com.posttracker.asyncdataloaders.PostDataLoader;
import com.posttracker.entities.Constants;
import com.posttracker.entities.ParcelObject;
import com.posttracker.entities.StatusObject;
import com.posttracker.entities.StatusObjectsList;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
```

```
import android.widget.ImageView;
import android.widget.TextView;
public class ParcelListBaseAdapter extends BaseAdapter {
	private static ArrayList<ParcelObject> parcelsArrayList;
	private Integer[] imgParcelTypeIdsList = {
				R.drawable.question_icon,
				R.drawable.package_icon,
				R.drawable.mail_icon
	};
	 private Integer[] imgFlagImgIdsList = {
				R.drawable.question_flag,
				R.drawable.lithuania_flag,
				R.drawable.hongkong_flag,
				R.drawable.us_flag
	};
	  private LayoutInflater l_Inflater;
	  public ParcelListBaseAdapter(Context context, ArrayList<ParcelObject> results) {
			parcelsArrayList = results;
	 l_Inflater = LayoutInflater.from(context);
	}

	public int getCount() {
	 return parcelsArrayList.size();
	}
	 public Object getItem(int position) {
	 return parcelsArrayList.get(position);
	}
	 public long getItemId(int position) {
	 return position;
	}

	public View getView(final int index, View convertView, ViewGroup parent) {
	 final ViewHolder holder;
	 if (convertView == null) {
	  convertView = l_Inflater.inflate(R.layout.list_item, null);
	  holder = new ViewHolder();
	  holder.lblParcelName = (TextView) convertView.findViewById(R.id.lblParcelName);
	  holder.lblTrackingNumber = (TextView) convertView.findViewById(R.id.lblTrackingNumber);
	  holder.lblLastStatus = (TextView) convertView.findViewById(R.id.lblLastStatus);
	  holder.imgParcel = (ImageView) convertView.findViewById(R.id.imgParcel);
	  holder.imgParcelOriginCountryFlag = (ImageView)
convertView.findViewById(R.id.imgParcelOriginCountryFlag);
	  convertView.setTag(holder);
	 } else {
	  holder = (ViewHolder) convertView.getTag();
	 }

	 holder.lblParcelName.setText(parcelsArrayList.get(index).ParcelName());
	 holder.lblTrackingNumber.setText(parcelsArrayList.get(index).TrackingNumber());
	 holder.lblLastStatus.setText(Constants.UPDATE_TEXT);
	 holder.imgParcel.setImageResource(GetParcelTypeResourceId(parcelsArrayList.get(index).ParcelType()));

holder.imgParcelOriginCountryFlag.setImageResource(GetFlagResourceId(parcelsArrayList.get(index).OriginCountry()));

	 Calendar calendar = Calendar.getInstance();
	 calendar.add(Calendar.MINUTE, -5);
	 Date lastUpdateDate = parcelsArrayList.get(index).StatusUpdateDate();
	 Date tmp = calendar.getTime();
	 if(lastUpdateDate.compareTo(tmp)<0)
	 {
			new PostDataLoader()
			{
			   @Override
			   public void onPostExecute(StatusObjectsList result)
			   {
					parcelsArrayList.get(index).StatusUpdateDate(Calendar.getInstance().getTime());
					parcelsArrayList.get(index).OriginCountryStatusList(result);
			   }
			}.execute(parcelsArrayList.get(index).TrackingNumber(),
parcelsArrayList.get(index).TrackingNumber().substring(11,13));

			new PostDataLoader()
			{
			   @Override
			   public void onPostExecute(StatusObjectsList result)
```

```
                        {
                                SimpleDateFormat sdfDate = new SimpleDateFormat(Constants.DATE_FORMAT);
                                parcelsArrayList.get(index).StatusUpdateDate(Calendar.getInstance().getTime());
                                parcelsArrayList.get(index).DestinationCountryStatusList(result);
                                if(result != null)
                                {
                                        StatusObject lastStatusEvent = result.LastStatusEvent();
                                        holder.lblLastStatus.setText(lastStatusEvent.Description() + "\n" +
sdfDate.format(lastStatusEvent.date()));// + sdfDate.format(parcelsArrayList.get(index).StatusUpdateDate()));
                                }
                                else
                                {
                                                parcelsArrayList.get(index).StatusUpdateDate(
Calendar.getInstance().getTime());
                                        holder.lblLastStatus.setText(Constants.UPDATE_TEXT + "\n" +
sdfDate.format(parcelsArrayList.get(index).StatusUpdateDate()));
                                }
                        }
                       }.execute(parcelsArrayList.get(index).TrackingNumber(), "LT");
               }
               else
               {
               StatusObject lastStatusEvent = parcelsArrayList.get(index).DestinationCountryStatusList().LastStatusEvent();
               SimpleDateFormat sdfDate = new SimpleDateFormat(Constants.DATE_FORMAT);
                        holder.lblLastStatus.setText(lastStatusEvent.Description() + "\n" +
sdfDate.format(parcelsArrayList.get(index).StatusUpdateDate())); //sdfDate.format(lastStatusEvent.date()));
               }
               return convertView;
              }

              public int GetFlagResourceId(String countryCode) {
                        if(countryCode.equalsIgnoreCase("LT"))
                        {
                                return imgFlagImgIdsList[1];
                        }
                        if(countryCode.equalsIgnoreCase("HK"))
                        {
                                return imgFlagImgIdsList[2];
                        }
                        if(countryCode.equalsIgnoreCase("US"))
                        {
                                return imgFlagImgIdsList[3];
                        }
                        return imgFlagImgIdsList[0];
              }

              public int GetParcelTypeResourceId(String parcelType) {
                        if(parcelType.equalsIgnoreCase("CG"))
                        {
                                return imgParcelTypeIdsList[1];
                        }
                        if(parcelType.equalsIgnoreCase("RT") || parcelType.equalsIgnoreCase("MP"))
                        {
                                return imgParcelTypeIdsList[2];
                        }
                        return imgParcelTypeIdsList[0];
              }

              static class ViewHolder {
               TextView lblParcelName;
               TextView lblTrackingNumber;
               TextView lblLastStatus;
               ImageView imgParcel;
               ImageView imgParcelOriginCountryFlag;
              }
}
```

StatusListBaseAdapter.java

```
package com.posttracker.adapters;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import com.posttracker.activities.R;
import com.posttracker.entities.Constants;
import com.posttracker.entities.StatusObject;
```

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

public class StatusListBaseAdapter extends BaseAdapter {
		private static ArrayList<StatusObject> statusArrayList;

		private LayoutInflater l_Inflater;

		public StatusListBaseAdapter(Context context, ArrayList<StatusObject> results) {
			StatusListBaseAdapter.statusArrayList = results;
			l_Inflater = LayoutInflater.from(context);
		}

		public int getCount() {
		 return statusArrayList.size();
		}

		public Object getItem(int position) {
		 return statusArrayList.get(position);
		}

		public long getItemId(int position) {
		 return position;
		}

		public View getView(final int index, View convertView, ViewGroup parent) {
		 final ViewHolder holder;
		 if (convertView == null) {
			convertView = l_Inflater.inflate(R.layout.destination_list_item, null);
			holder = new ViewHolder();
			holder.tvEvent = (TextView) convertView.findViewById(R.id.tvEvent);
			holder.tvLocation = (TextView) convertView.findViewById(R.id.tvLocation);
			holder.tvDate = (TextView) convertView.findViewById(R.id.tvDate);
			convertView.setTag(holder);
		 }
		 else {
			holder = (ViewHolder) convertView.getTag();
		 }

		 holder.tvEvent.setText(statusArrayList.get(index).Description());
		 holder.tvLocation.setText(statusArrayList.get(index).Location());
		 SimpleDateFormat sdfDate = new SimpleDateFormat(Constants.DATE_FORMAT);
		 holder.tvDate.setText(sdfDate.format(statusArrayList.get(index).date()));

		 return convertView;
		}

		static class ViewHolder {
		 TextView tvEvent;
		 TextView tvLocation;
		 TextView tvDate;
		}
}
```

Modulis com.posttracker.asyncdataloaders

PostDataLoader.java

```
package com.posttracker.asyncdataloaders;
import com.posttracker.entities.StatusObjectsList;
import com.posttracker.parsers.IParser;
import com.posttracker.parsers.ParserFactory;
import android.os.AsyncTask;
public class PostDataLoader extends AsyncTask<String, Void, StatusObjectsList> {
   @Override
   protected StatusObjectsList doInBackground(String... params) {
		Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
		IParser parser = ParserFactory.GetParser(params[1]);
		StatusObjectsList parcelStatusList = parser.GetParcelStatusList(params[0]);
      return parcelStatusList;
```

```
    }
    @Override
    protected void onPostExecute(StatusObjectsList result) {
            super.onPostExecute(result);
    }
    @Override
    protected void onPreExecute() {
    }
    @Override
    protected void onProgressUpdate(Void... values) {
    }
}
```

Modulis com.posttracker.entities;

Constants.java

```
package com.posttracker.entities;

public class Constants {
    private Constants() { }

            ///Lithuanian translation///
            public static final String DATE_FORMAT = "yyyy-MM-dd HH:mm";
            public static final String US_DATE_TIME_FORMAT = "MMMM dd, yyyy, HH:mm a";
            public static final String US_DATE_FORMAT = "MMMM dd, yyyy";
            public static final String UPDATE_TEXT = "Atnaujinama siuntos būsena...";
            public static final String LOADING_TEXT =  "Atnaujinama...";
            public static final String TRACKING_NUMBER_TEXT =  "Siuntos numeris";

            public static final String LOCATION_EMPTY = "Informacijos apie vietovę nėra.";
            public static final String DESCRIPTION_EMPTY = "Informacijos apie siuntą nėra.";

            public static final String LOCATION_EMPTY_LT_POST = "Lietuva";
            public static final String DESCRIPTION_EMPTY_LT_POST = "Lietuvos paštas neturi duomenų apie siuntą.";

            public static final String INTERNET_CONNECTION_ERROR = "Neįmanoma prisijungti prie paštos sistemos.
            Patikrinkite interneto ryšį.";
            public static final String REFRESH_ERROR = "Neįmanoma prisijungti prie paštos sistemos. Patikrinkite interneto
            ryšį.";
            public static final String UNKNOWN_ERROR = "Nežinoma klaida...sveikiname Jūs sulaužėte programėlę >:[";

            public static final String EMPTY_LOCATION = "Informacijos apie vietovę nėra.";

                    public static final String TRACKING_NUMBER_ERROR = "Siuntos numeris ilgis netinkamas. Jis turi
            būti 13 simbolių ilgio.";
            public static final String PARCEL_NAME_ERROR = "Įveskite siuntinio pavadinimą.";

             ///English translations///
    /*
            public static final String DATE_FORMAT = "yyyy-MM-dd HH:mm";
            public static final String US_DATE_TIME_FORMAT = "MMMM dd, yyyy, HH:mm a";
            public static final String US_DATE_FORMAT = "MMMM dd, yyyy";
            public static final String UPDATE_TEXT = "Updating parcel status...";
            public static final String LOADING_TEXT =  "Loading...";
            public static final String TRACKING_NUMBER_TEXT =  "Tracking number";

            public static final String LOCATION_EMPTY = "There is no location information.";
            public static final String DESCRIPTION_EMPTY = "There is no description provided.";

            public static final String LOCATION_EMPTY_LT_POST = "Lietuva";
            public static final String DESCRIPTION_EMPTY_LT_POST = "Lietuvos paštas neturi duomenų apie siuntą.";

            public static final String INTERNET_CONNECTION_ERROR = "Cannot connect to post service. Check internet
            connection.";
            public static final String REFRESH_ERROR = "Cannot connect to post service. Please try to refresh.";
            public static final String UNKNOWN_ERROR = "Gratz You've broke it...";

            public static final String EMPTY_LOCATION = "";

            public static final String TRACKING_NUMBER_ERROR = "Tracking number should be 13 chrachters lenght.";
            public static final String PARCEL_NAME_ERROR = "Please enter parcel name.";
            */
}
```

ParcelObject.java

```
package com.posttracker.entities;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.simpleframework.xml.Element;
import org.simpleframework.xml.Root;
import android.os.Parcel;
import android.os.Parcelable;
@Root
public class ParcelObject implements Parcelable
{
		@Element
		private String parcelName;
		public String ParcelName() {
			return this.parcelName;
		}
		public void ParcelName(String parcelName) {
			this.parcelName = parcelName;
		}
		@Element
		private String trackingNumber;
		public String TrackingNumber() {
			return this.trackingNumber;
		}
		public void TrackingNumber(String trackingNumber) {
			this.trackingNumber = trackingNumber;
		}
		@Element
		private StatusObjectsList destinationCountryStatusList = new StatusObjectsList();
		public StatusObjectsList DestinationCountryStatusList() {return this.destinationCountryStatusList;}
		public void DestinationCountryStatusList(StatusObjectsList statusList) {this.destinationCountryStatusList =
statusList;}

		@Element
		private StatusObjectsList originCountryStatusList = new StatusObjectsList();
		public StatusObjectsList OriginCountryStatusList() {return this.originCountryStatusList;}
		public void OriginCountryStatusList(StatusObjectsList statusList) {this.originCountryStatusList = statusList;}

		@Element
		private String originCountry;
		public String OriginCountry() {
			return this.originCountry;
		}
		public void OriginCountry(String originCountry) {
			this.originCountry = originCountry;
		}
		@Element
		private String parcelType;
		public String ParcelType() {
			return this.parcelType;
		}
		public void ParcelType(String parcelType) {
			this.parcelType = parcelType;
		}
		@Element
		private Date statusUpdateDate;
		public Date StatusUpdateDate() {
			return this.statusUpdateDate;
		}
		public void StatusUpdateDate(Date statusUpdateDate) {
			this.statusUpdateDate = statusUpdateDate;
		}

   public ParcelObject() {}

   public int describeContents() {
      return 0;
   }

   public void writeToParcel(Parcel out, int flags) {
```

```
        out.writeString(this.trackingNumber);
        out.writeString(this.parcelName);

        out.writeParcelable(this.destinationCountryStatusList,  0);
        out.writeParcelable(this.originCountryStatusList,  0);

        out.writeString(this.originCountry);
        out.writeString(this.parcelType);

        SimpleDateFormat sdfDate = new SimpleDateFormat(Constants.DATE_FORMAT);
        out.writeString(sdfDate.format(this.statusUpdateDate));
    }

    public static final Parcelable.Creator<ParcelObject> CREATOR = new Parcelable.Creator<ParcelObject>() {
        public ParcelObject createFromParcel(Parcel in) {
            return new ParcelObject(in);
        }

        public ParcelObject[] newArray(int size) {
            return new ParcelObject[size];
        }
    };

    public ParcelObject(Parcel in) {
        this.trackingNumber= in.readString();
        this.parcelName= in.readString();

        this.destinationCountryStatusList = in.readParcelable(StatusObjectsList.class.getClassLoader());
        this.originCountryStatusList = in.readParcelable(StatusObjectsList.class.getClassLoader());

        this.originCountry= in.readString();
        this.parcelType= in.readString();

        SimpleDateFormat formatter = new SimpleDateFormat(Constants.DATE_FORMAT);
        try {
                                this.statusUpdateDate = (Date)formatter.parse(in.readString());
                        } catch (ParseException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
    }
}
```

ParcelObjectsList.java

```
package com.posttracker.entities;
import java.util.ArrayList;
import java.util.List;
import org.simpleframework.xml.ElementList;
import org.simpleframework.xml.Root;
@Root
public class ParcelObjectsList {
  @ElementList
  private List<ParcelObject> list = new ArrayList<ParcelObject>();
  public List<ParcelObject> Parcels() {
    return this.list;
  }
    public void Parcels(ArrayList<ParcelObject> parcels) {
            this.list = parcels;
  }
}
```

StatusObject.java

```
package com.posttracker.entities;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.simpleframework.xml.Element;
import org.simpleframework.xml.Root;
import android.os.Parcel;
import android.os.Parcelable;

@Root
```

```
public class StatusObject implements Parcelable {

		@Element
		private String description;
		public String Description() {
			return this.description;
		}
		public void Description(String description) {
			this.description = description;
		}

		@Element
		private String location;
		public String Location() {
			return this.location;
		}
		public void Location(String location) {
			this.location = location;
		}

		@Element
		private Date date;
		public Date date() {
			return this.date;
		}
		public void Date(Date date) {
			this.date = date;
		}

   public StatusObject() {}

   public void writeToParcel(Parcel out, int flags) {
      out.writeString(this.description);
      out.writeString(this.location);

      SimpleDateFormat sdfDate = new SimpleDateFormat(Constants.DATE_FORMAT);
      out.writeString(sdfDate.format(this.date));
   }

   public static final Parcelable.Creator<StatusObject> CREATOR = new Parcelable.Creator<StatusObject>() {
      public StatusObject createFromParcel(Parcel in) {
         return new StatusObject(in);
      }

      public StatusObject[] newArray(int size) {
         return new StatusObject[size];
      }
   };

   public StatusObject(Parcel in) {
      this.description= in.readString();
      this.location= in.readString();

      SimpleDateFormat formatter = new SimpleDateFormat(Constants.DATE_FORMAT);
      try {
				this.date = (Date)formatter.parse(in.readString());
			} catch (ParseException e) {
				e.printStackTrace();
			}
   }

   public int describeContents() {
      return 0;
   }
}
```

StatusObjectsList.java

```
package com.posttracker.entities;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import org.simpleframework.xml.Element;
import org.simpleframework.xml.ElementList;
import org.simpleframework.xml.Root;
```

```
import android.os.Parcel;
import android.os.Parcelable;
@Root
public class StatusObjectsList implements Parcelable {
		@Element
		private StatusObject lastStatusEvent = new StatusObject();
		public StatusObject LastStatusEvent() {
			if(this.list.size() != 0)
			{
				this.lastStatusEvent = this.list.get(0);
			}
			else
			{
				StatusObject statusObject = new StatusObject();
				statusObject.Description(Constants.DESCRIPTION_EMPTY_LT_POST);
				statusObject.Location(Constants.LOCATION_EMPTY_LT_POST);
				statusObject.Date(Calendar.getInstance().getTime());
				this.lastStatusEvent = statusObject;
			}
			return this.lastStatusEvent;
		}
			@ElementList
		private List<StatusObject> list = new ArrayList<StatusObject>();
		public List<StatusObject> Statuses() {
			return this.list;
		}

		public void Statuses(ArrayList<StatusObject> statuses) {
			this.list = statuses;
		}

		public StatusObjectsList() {
			if(this.list.size() != 0)
			{
				this.lastStatusEvent = this.list.get(this.list.size()-1);
			}
			else
			{
				StatusObject statusObject = new StatusObject();
				statusObject.Description(Constants.DESCRIPTION_EMPTY_LT_POST);
				statusObject.Location(Constants.LOCATION_EMPTY_LT_POST);
				statusObject.Date(Calendar.getInstance().getTime());
				this.lastStatusEvent = statusObject;
			}
		}
   public void writeToParcel(Parcel out, int flags) {
      out.writeParcelable(lastStatusEvent, 0);
      out.writeArray(list.toArray());
   }
   public static final Parcelable.Creator<StatusObjectsList> CREATOR = new Parcelable.Creator<StatusObjectsList>() {
      public StatusObjectsList createFromParcel(Parcel in) {
         return new StatusObjectsList(in);
      }
      public StatusObjectsList[] newArray(int size) {
         return new StatusObjectsList[size];
      }
   };
   public StatusObjectsList(Parcel in) {
      this.lastStatusEvent = in.readParcelable(StatusObject.class.getClassLoader());
      in.readList(this.list, StatusObject.class.getClassLoader());
   }
      public int describeContents() {
      return 0;
   }
}
```

StatusViewHolder.java

```
package com.posttracker.entities;
import android.widget.TextView;
public class StatusViewHolder {
		public TextView tvEvent;
		public TextView tvLocation;
		public TextView tvDate;
```

```
}
```

Modulis com.posttracker.iodata

DataManager.java

```
package com.posttracker.iodata;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;

import org.simpleframework.xml.Serializer;
import org.simpleframework.xml.core.Persister;

import com.posttracker.entities.ParcelObjectsList;

import android.app.Activity;
import android.content.Context;

public class DataManager {

	public static void SaveData(Activity currentActivity, ParcelObjectsList list){

		final String filename = "Data.xml";
		FileOutputStream outputStream;

		File dir = new File(currentActivity.getFilesDir() + "/UserData");
		if(dir.exists()==false)
		{
			try{
			dir.mkdirs();
			}catch (Exception e) {
				e.printStackTrace();
			}
		}
		try {

			Serializer serializer = new Persister();
			outputStream = currentActivity.openFileOutput(filename, Context.MODE_PRIVATE);
			serializer.write(list, outputStream);
		} catch (Exception e)
		{
		  e.printStackTrace();
		}
	}

	public static ParcelObjectsList GetData(Activity currentActivity){
		final String filename = "Data.xml";
		ParcelObjectsList list = new ParcelObjectsList();
		FileInputStream inputStream;

		File dir = new File(currentActivity.getFilesDir() + "/UserData");
		if(dir.exists()==false)
		{
			return new ParcelObjectsList();
		}
		else
		{
			try {
				Serializer serializer = new Persister();
				inputStream = currentActivity.openFileInput(filename);
				serializer.read(list, inputStream);
			} catch (Exception e)
			{
			  e.printStackTrace();
			  return new ParcelObjectsList();
			}
		}
		return list;
	}

}
```

Modulis com.posttracker.parsers

HonkongPostParser.java

```
package com.posttracker.parsers;
import java.net.ConnectException;
import java.net.SocketTimeoutException;
import java.net.URL;
import java.net.UnknownHostException;
import java.util.Calendar;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;

import com.posttracker.entities.Constants;
import com.posttracker.entities.StatusObject;
import com.posttracker.entities.StatusObjectsList;

public class HonkongPostParser implements IParser {
	@Override
	public StatusObjectsList GetParcelStatusList(String trackingNumber) {
		StatusObjectsList parcelStatusList = new StatusObjectsList();
		Document doc;
		try
		{
			try
			{
				doc =
Jsoup.connect("http://app3.hongkongpost.com/CGI/mt/mt2result.jsp?tracknbr="+trackingNumber).get();
			}
			catch (SocketTimeoutException  e)
			{
				URL url = new
URL("http://app3.hongkongpost.com/CGI/mt/mt2result.jsp?tracknbr="+trackingNumber);
				doc = Jsoup.parse(url, 10000);
			}

			Element statusElement = doc.select(".ieFix").first();
			StatusObject tmp = new StatusObject();
		tmp.Description(statusElement.childNode(12).toString());
		tmp.Location("HonkongPost");
		Calendar calendar = Calendar.getInstance();
	tmp.Date(calendar.getTime());
		parcelStatusList.Statuses().add(tmp);
		}
		catch (ConnectException  e)
		{
			StatusObject tmp = new StatusObject();
			tmp.Description(Constants.REFRESH_ERROR);
			tmp.Location(Constants.EMPTY_LOCATION);
		Calendar calendar = Calendar.getInstance();
	tmp.Date(calendar.getTime());
		parcelStatusList.Statuses().add(tmp);
			return parcelStatusList;
		}
		catch (UnknownHostException  e)
		{
			StatusObject tmp = new StatusObject();
			tmp.Description(Constants.REFRESH_ERROR);
			tmp.Location(Constants.EMPTY_LOCATION);
		Calendar calendar = Calendar.getInstance();
	tmp.Date(calendar.getTime());
		parcelStatusList.Statuses().add(tmp);
			return parcelStatusList;
		}
		catch (Exception e)
		{
			StatusObject tmp = new StatusObject();
			tmp.Description(Constants.UNKNOWN_ERROR + e);
			tmp.Location(Constants.EMPTY_LOCATION);
		Calendar calendar = Calendar.getInstance();
	tmp.Date(calendar.getTime());
		parcelStatusList.Statuses().add(tmp);
```

```
                        return parcelStatusList;
                }
                return parcelStatusList;
        }
}
```

IParser.java

```
package com.posttracker.parsers;
import com.posttracker.entities.StatusObjectsList;
public interface IParser {
        public StatusObjectsList GetParcelStatusList(String trackingNumber);
}
```

LithuniaPostParser.java

```
package com.posttracker.parsers;

import java.net.ConnectException;
import java.net.SocketTimeoutException;
import java.net.URL;
import java.net.UnknownHostException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Collections;
import java.util.Date;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import com.posttracker.entities.Constants;
import com.posttracker.entities.StatusObject;
import com.posttracker.entities.StatusObjectsList;

public class LithuniaPostParser implements IParser {
        @Override
        public StatusObjectsList GetParcelStatusList(String trackingNumber) {
                System.setProperty("http.keepAlive", "false");
                StatusObjectsList parcelStatusList = new StatusObjectsList();
                Document doc;
                try
                {
                        try
                        {
                                doc = Jsoup.connect("http://www.post.lt/lt/pagalba/siuntu-
paieska/index?num="+trackingNumber).get();
                        }
                        catch (SocketTimeoutException  e)
                        {
                                URL url = new URL("http://www.post.lt/lt/pagalba/siuntu-
paieska/index?num="+trackingNumber);
                                doc = Jsoup.parse(url, 10000);
                        }
                        catch(java.net.SocketException ex)
                        {
                                URL url = new URL("http://www.post.lt/lt/pagalba/siuntu-
paieska/index?num="+trackingNumber);
                                doc = Jsoup.parse(url, 10000);
                        }

                        Elements tables = doc.select("tbody");
                for (Element table : tables) {
                        for (Element row : table.children()) {
                                if(row != null)
                                {
                                        StatusObject tmp = new StatusObject();
                                        Elements columns = row.getElementsByTag("td");
                                        tmp.Description(columns.get(0).text());
                                        tmp.Location(columns.get(1).text());
                                SimpleDateFormat formatter = new SimpleDateFormat(Constants.DATE_FORMAT);
                                tmp.Date((Date)formatter.parse(columns.get(2).text()));
                                        parcelStatusList.Statuses().add(tmp);
```

```
                                        }
                                        }
                                }
                        Collections.reverse(parcelStatusList.Statuses());
                                }
                                catch (ConnectException  e)
                                {
                                        StatusObject tmp = new StatusObject();
                                        tmp.Description(Constants.INTERNET_CONNECTION_ERROR);
                                        tmp.Location(Constants.EMPTY_LOCATION);
                                Calendar calendar = Calendar.getInstance();
        tmp.Date(calendar.getTime());
                                parcelStatusList.Statuses().add(tmp);
                                        return parcelStatusList;
                                }
                                catch (UnknownHostException  e)
                                {
                                        StatusObject tmp = new StatusObject();
                                        tmp.Description(Constants.REFRESH_ERROR);
                                        tmp.Location(Constants.EMPTY_LOCATION);
                                Calendar calendar = Calendar.getInstance();
        tmp.Date(calendar.getTime());
                                parcelStatusList.Statuses().add(tmp);
                                        return parcelStatusList;
                                }
                                catch (Exception e)
                                {
                                        StatusObject tmp = new StatusObject();
                                        tmp.Description(Constants.UNKNOWN_ERROR + e);
                                        tmp.Location(Constants.EMPTY_LOCATION);
                                Calendar calendar = Calendar.getInstance();
        tmp.Date(calendar.getTime());
                                parcelStatusList.Statuses().add(tmp);
                                        return parcelStatusList;
                                }
                                return parcelStatusList;

                }
}
```

ParserFactory.java

```
package com.posttracker.parsers;
public class ParserFactory {

        public static IParser GetParser(String postID)
        {
                IParser parser = new LithuniaPostParser() ;
                if(postID.equalsIgnoreCase("HK"))
                {
                        parser = new HonkongPostParser();
                }
                if(postID.equalsIgnoreCase("LT"))
                {
                        parser = new LithuniaPostParser();
                }
                if(postID.equalsIgnoreCase("US"))
                {
                        parser = new USPostParser();
                }
                return parser;
        }
}
```

USPostParser.java

```
package com.posttracker.parsers;
import java.net.ConnectException;
import java.net.SocketTimeoutException;
import java.net.URL;
import java.net.UnknownHostException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
```

```
import java.util.Date;
import java.util.Locale;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import com.posttracker.entities.Constants;
import com.posttracker.entities.StatusObject;
import com.posttracker.entities.StatusObjectsList;
public class USPostParser implements IParser {
	@Override
	public StatusObjectsList GetParcelStatusList(String trackingNumber) {
		StatusObjectsList parcelStatusList = new StatusObjectsList();
		Document doc;
		try
		{
			try
			{
				doc =
Jsoup.connect("https://tools.usps.com/go/TrackConfirmAction_input?strOrigTrackNum="+trackingNumber).get();
			}
			catch (SocketTimeoutException  e)
			{
				URL url = new
URL("https://tools.usps.com/go/TrackConfirmAction_input?strOrigTrackNum="+trackingNumber);
				doc = Jsoup.parse(url, 10000);
			}
			Elements tables = doc.select("tbody");
		for (Element table : tables) {
			for (Element row : table.children()) {
			if(row != null)
			{
				if(row.children().size() != 0)
				{
					StatusObject tmp = new StatusObject();
					String description = row.select(".td-status").text();
					String location = row.select(".td-location").text();
					String date = row.select(".td-date-time").text();

					if(description.isEmpty())
					{
						tmp.Description(Constants.DESCRIPTION_EMPTY);
					}
					else
					{
						tmp.Description(description);
					}

					if(location.isEmpty())
					{
						tmp.Location(Constants.LOCATION_EMPTY);
					}
					else
					{
						tmp.Location(location);
					}
				SimpleDateFormat formatter = new
SimpleDateFormat(Constants.US_DATE_TIME_FORMAT, Locale.US);
				Date dateParsed;
				try
				{
				  dateParsed = (Date)formatter.parse(date);
				}
				catch (Exception e)
				{
				  SimpleDateFormat onlyDateFormatter = new
SimpleDateFormat(Constants.US_DATE_FORMAT, Locale.US);
				  dateParsed = (Date)onlyDateFormatter.parse(date);
				}
			tmp.Date(dateParsed);
					parcelStatusList.Statuses().add(tmp);
				 }
			}
			}
```

```
                    }
                    }
                    catch (ConnectException  e)
                    {
                              StatusObject tmp = new StatusObject();
                              tmp.Description(Constants.REFRESH_ERROR);
                              tmp.Location(Constants.EMPTY_LOCATION);
                    Calendar calendar = Calendar.getInstance();
          tmp.Date(calendar.getTime());
                    parcelStatusList.Statuses().add(tmp);
                              return parcelStatusList;
                    }
                    catch (UnknownHostException  e)
                    {
                              StatusObject tmp = new StatusObject();
                              tmp.Description(Constants.REFRESH_ERROR);
                              tmp.Location(Constants.EMPTY_LOCATION);
                    Calendar calendar = Calendar.getInstance();
          tmp.Date(calendar.getTime());
                    parcelStatusList.Statuses().add(tmp);
                              return parcelStatusList;
                    }
                    catch (Exception e)
                    {
                              StatusObject tmp = new StatusObject();
                              tmp.Description(Constants.UNKNOWN_ERROR + e);
                              tmp.Location(Constants.EMPTY_LOCATION);
                    Calendar calendar = Calendar.getInstance();
          tmp.Date(calendar.getTime());
                    parcelStatusList.Statuses().add(tmp);
                              return parcelStatusList;
                    }
                    return parcelStatusList;
               }
}
```