KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Ruslanas Vitiutinas

UML CASE įrankio išplėtimas modelių transformacijomis

Magistro darbas

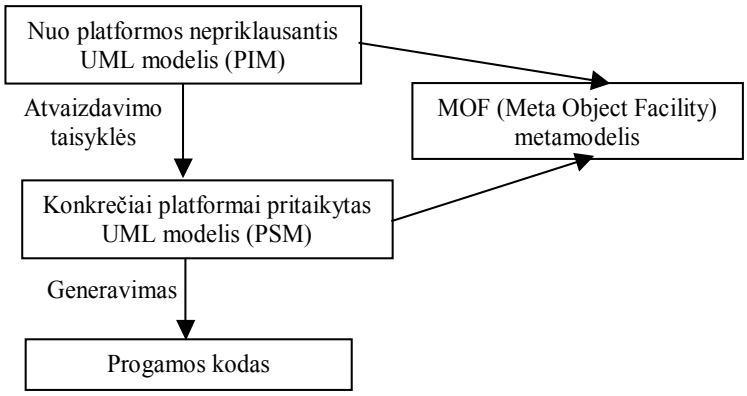
Kalbos konsultantė		Vadovas	
	Lietuvių kalbos konsultantė J.Mikelionienė		Doc. Lina Nemuraitė
2006-05-		2006-05-	
Recenzentas		Atliko	
	dr. Darius Šilingas		IFM/02 gr.stud.
2006-05-			Ruslanas Vitiutinas
			2006-05-

Kaunas, 2006

Turinys

1. ĮVADAS

Pastaruoju metu tobulėjant informacinėms technologijoms ir didėjant kuriamų informacinių sistemų skaičiui, atsirado poreikis padidinti kūrimo proceso automatizavimo laipsnį. Automatizavimo problemai spręsti galima taikyti organizacijos Object Management Group (OMG) [1] sukurtą Model Driven Architecture (MDA) sistemų kūrimo technologiją. Ši technologija paremta modelių naudojimu sistemų projektavimo procese. MDA pagrindas yra nuo realizavimo technologijos nepriklausomas sistemos modelis PIM (*Platform Independent Model*), kuris sudaromas unifikuota modeliavimo kalba UML. Kadangi šis modelis nepriklauso nuo konkrečios realizavimo platformos ir MDA automatiškai transformuojamas į programos kodą, jis gali būti greitai pritaikytas naujoms ar pasikeitusioms technologijoms [2]. Tokiu būdu MDA sistemų kūrimo procesas apima perėjimus nuo PIM (nuo platformos nepriklausomo modelio) prie PSM (platformai specifinio modelio) ir nuo PSM prie programos išeities kodo (1 pav).



1 pav. Sistemų kompiuterizavimo etapai, taikant OMG MDA principus [2]

Perėjimas nuo PIM prie PSM galimas keliais būdais: projektuotojas gali pats pritaikyti PIM konkrečiai platformai, panaudodamas iš anksto paruoštus šablonus, kurie palengvintų tokį pritaikymą, arba projektuotojui pasirinkus reikiamą platformą, CASE įrankiai automatiškai atliktų PIM pavertimą į PSM [3]. Kol kas, norėdami taikyti MDA metodiką, konkrečiai platformai skirtą generavimo įrankį turi susikurti patys programinės įrangos kūrėjai. Nors skelbiama, kad jau yra keletas įrankių, pagrįstų MDA metodika, realiai tokie įrankiai, kaip ir pati MDA architektūra, yra kūrimo stadijoje. Tačiau tikimasi, kad artimiausiais metais jie bus sukurti [3].

Informacinių sistemų kūrimą, remiantis MDA, galima pavaizduoti tokia transformacijų seka: RM $\xrightarrow{T1}$ PIM $\xrightarrow{T2}$ PSM $\xrightarrow{T3}$ PR, čia RM – reikalavimų sistemai modelis, PR – programinė realizacija pasirinktoje platformoje [3], T_2, T_3 – transformacijos, kurias siekia automatizuoti OMG.

Šio proceso eigoje mažai dėmesio skiriama PIM modelio teisingumui – patikrinimui ar kuriamos informacinės sistemos reikalavimų etapo modelis teisingas. Atlikti PIM išbaigtumo ir suderinamumo tikrinimą yra būtina norint gauti teisingai veikiančią sistemą. Transformacijų pagalba generuojant PIM modelį iš reikalavimų modelio, teisinga būtų modelio patikrinimą atlikti ankstyvesnėje kūrimo fazėje – reikalavimų modelio sudarymo procese.

1.1 Darbo tikslas

Suprojektuoti ir realizuoti programų sistemą skirtą reikalavimo modelio tikrinimui ir transformacijai į projekto modelį atlikti MagicDraw [4] UML CASE įrankio aplinkoje.

1.2 Uždaviniai

- Atlikti esamų MDA įrankių analizę, siekiant nustatyti jų savybes, naudojamas technologijas, integravimo su CASE įrankiais galimybes ir tinkamumą modelio į modelį transformacijoms vykdyti.
- Suprojektuoti ir realizuoti reikalavimų modelio transformavimą į projektą atliekančią programų sistemą.
- Integruoti sukurtą programų sistemą į MagicDraw įrankį.
- Atlikti įrankio funkcionalumo pranašumo, prieš rankinį modelio kūrimo būdą, tyrimą.

1.3 Literatūros apžvalga

Literatūros apžvalgai atlikti buvo naudojami keturių tipų literatūros šaltiniai:

- Techninės įrankių ir technologijų specifikacijos.
- Apžvalginiai įrankių straipsniai.
- Knygų, susijusių su duota tematika, elektroninės versijos.
- Konferencijų medžiaga ir straipsniai.

1.4 Darbo struktūra

Darbo analizės dalyje pateikta penkių MDA įrankių tiekiamo funkcionalumo ir integravimo su CASE įrankiais galimybių analizė. Nagrinėjamos analizuojamų MDA įrankių savybės, tinkamumas DIM modelio sudarymo ir transformavimo į PIM modelį uždaviniams spręsti. Šios dalies pabaigoje pateiktas analizės rezultatų apibendrinimas ir išvados.

Projektavimo dalyje aprašyti DIM modelio transformavimo į PIM modelį įrankio reikalavimai ir projektiniai sprendimai, kurie buvo pritaikyti įrankio kūrimo proceso metu.

Taip pat šioje dalyje aprašomas sukurto transformavimo įrankio architektūros modelis, integravimo su MagicDraw UML CASE įrankių sąsajos realizavimas.

Darbo trečiame skyriuje pateikti reikalavimo modelio sudarymo metodai, aprašyta projekto modelis PIM ir reikalavimo modelio transformavimo į projekto modelį metodika. Pateiktas reikalavimo modelio tikrinimo algoritmas.

Penktoje darbo dalyje aprašytas atliktas sukurto MagicDraw įskiepio naudingumo tyrimas, pateikti tyrimo rezultatai. Tyrimo tikslas - nustatyti automatinio PIM modelio generavimo iš DIM pranašumus, prieš rankinį PIM modelio kūrimo būdą. Pranašumas vertinamas modelio kūrimui reikalingo laiko sąnaudų palyginimu.

Pabaigoje pateikiamos atlikto darbo pagrindinės išvados ir rezultatai.

Prieduose pateikiami reikalavimų modelio ir sukurto įskiepio sugeneruoti projekto modelių pavyzdžiai.

2. MODELIAIS GRINDŽIAMO KŪRIMO TECHNOLOGIJŲ IR ĮRANKIŲ ANALIZĖ

DIM modelio transformacijai į PIM modelį atlikti reikalingas MDA įrankis, galintis transformacijos metų vykdyti sudėtingus modelio elementų atributų analizės veiksmus. Šiuo metu, egzistuoja daug sukurtų įrankių, daugiau ar mažiau atitinkančių MDA specifikaciją. Daugelis parduodamų produktų, klaidingai vadinami MDA įrankiais [5]. Dažniausiai tai kodo generavimo ir diagramų modeliavimo įrankiai, kurie gali būti naudojami MDA proceso metu. Pagrindinė MDA idėja – suskirstyti kuriamą programinę įrangą į tris modelius: CIM (*Computationally Independent Model*), PIM ir PSM. MDA įrankis turi tiksliai nusakyti kiekvieno MDA modelio ribas ir leisti atlikti transformacijas iš vieno modelio į kitą, o ne tik generuoti programos kodą iš klasių diagramų [5].

Analizei atlikti buvo pasirinkti MDA įrankiai atitinkantys MDA specifikaciją [6] ir turintys integravimo su CASE įrankiais galimybę.

2.1 OptimalJ MDA įrankis

Compuware įmonės kuriamas MDA įrankis OptimalJ gali atlikti transformacijas iš modelio į modelį, ir transformuoti modelį į programinį kodą. Šio įrankio modeliai aprašomi UML kalba. Modelio transformavimo į kodą metu gaunamas Java Interprase Edition (J2EE) [7] specifikaciją atitinkantis kodas. OptimalJ tai modeliais ir projektavimo šablonais grįstas informacinių sistemų kūrimo įrankis, skirtas J2EE taikomųjų programų kūrimui, integravimui ir priežiūrai. OptimalJ įrankio naudojami projektavimo šablonai palengvina sudėtingų sistemų kūrimą. Šablonų naudojimas gerina kodo kokybę ir struktūrą, padidina pakartotino panaudojimo galimybes.

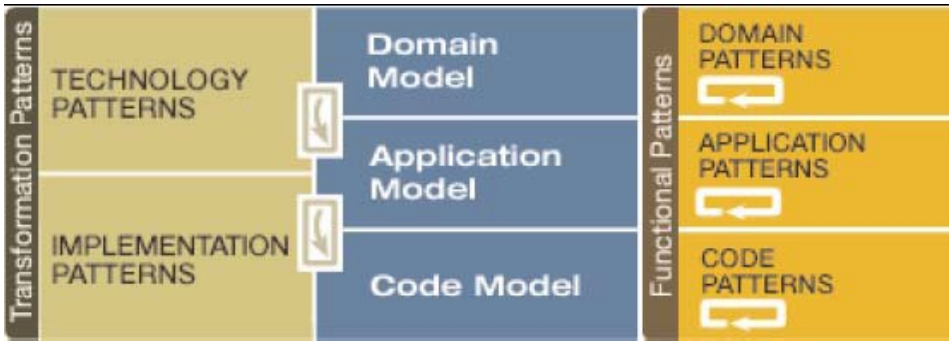
Informacinių sistemų kūrimo OptimalJ įrankių proceso metu naudojami:

- dalykinę sritį aprašantis PIM modelis
- nuo platformos priklausantis modelis PSM
- programos išeities kodas.

PIM modelio transformavimas į PSM modelį ir programinio išeities kodo generavimas vykdomas pagal vartotojo apibrėžtas taisykles.

2.1.1 OptimalJ modeliais grįstas programinės įrangos kūrimas

Modeliais valdomas programinės įrangos kūrimą OptimalJ įrankiu, iliustruoja paveikslas 2.



2 pav. OptimalJ įrankio architektūra [8]

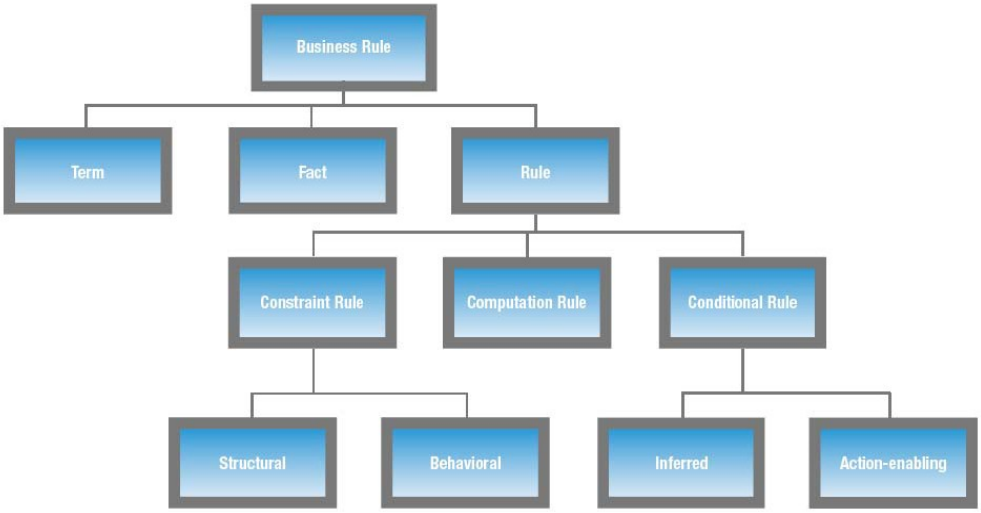
OptimalJ kūrimo aplinkoje naudojami du transformavimo šablonai – technologijos ir realizavimo (2 pav.). Šie šablonai naudojami PIM į PSM ir PSM į kodą transformacijoms aprašyti. Tuo tarpu, į funkcinių šablonų grupę įeina PIM, PSM ir programos kodo šablonai.

OptimalJ pagalba galima atvaizduoti kuriamą sistemą skirtingose abstrakcijų lygmenyse, kas palengvina sistemos pritaikymą pasikeitus verslo modelio reikalavimams. Taip pat OptimalJ gerina kuriamos sistemos kokybę, padidiną našumą ir palaikomumą atskiriant verslo modelį nuo architektūros modelio ir architektūros modelį nuo realizavimo programinės kalbos. OptimalJ [9] nustato kur reikalingas pakeitimų taikymas programoje. OptimalJ pakeitimai skirstomi:

- veiklos pakeitimai dalykinės srities modelyje
- architektūros pakeitimai architektūrą aprašančiame modelyje
- programos kodo pakeitimai

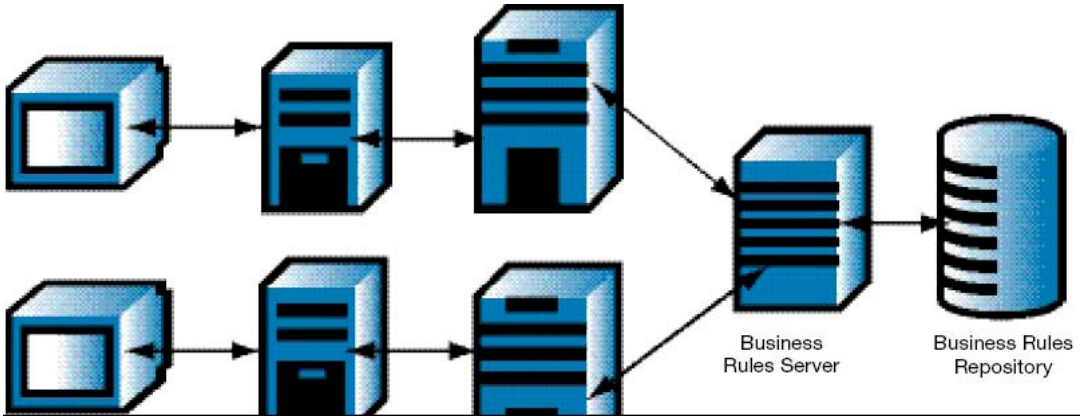
2.1.2 Verslo modelio taisyklės

Verslo modelio taisyklės atvaizduoja organizacijos veiklą, kuri diktuoja ką turi atlikti kuriama programinė įranga tai veiklai palengvinti arba automatizuoti. OptimalJ leidžia organizacijai dalyvauti programinės įrangos kūrimo procese [10]. OptimalJ įrankio pagalba galima aprašyti ir realizuoti verslo taisykles atskirai nuo kuriamos programos logikos. Norint sudaryti tokias taisykles reikia visų pirma nustatyti veiklą ir jos organizavimą. Paveikslas 3 iliustruoja verslo logikos taisyklių hierarchiją OptimalJ įrankyje.



3 pav. OptimalJ verslo taisyklių hierarchija [8]

Verslo logikos taisyklės yra aprašomos modeliais. OptimalJ paverčia šiuos modelius veikiančia J2EE taikomąja programa. Verslo taisyklės įsigalioja skirtingose taikomosios programos lygiuose. Darbe [10] rašoma, kad OptimalJ veiklos logikos taisyklės palaikomos ir stebimos atskirai nuo sistemos. Šios taisyklės saugomos saugykloje ir pasiekiamos verslo logikos serverio pagalba (4 pav.).



4 pav. Verslo logikos serverio naudojimo schema [8]

Veiklos logikos taisyklių centralizavimas palengvina pakeitimų pritaikymą ir padidina pakartotinio panaudojimo galimybes. Verslo logikos taisyklių serveris leidžia naudotis taisyklėmis programos vykdymo metu. Sistemų kūrėjai ir administratoriai gali tiesiogiai kreiptis į taisyklių serverį, modifikuoti taisykles. Atlikti pakeitimai matomi programoje. Toks taisyklių pakeitimas nereikalauja programos kodo pakartotinio perkompiliavimo ir įdiegimo. Jei programai prireikia verslo taisyklės, ji gali gauti ją iš taisyklių serverio. Kreipiniai į taisyklių serverį galimi iš bet kurio programos architektūrinio lygmens.

2.1.3 OptimalJ transformavimo šablonai

OMG dar ne specifikavo, kaip turi būti atliekamos transformacijos tarp PIM, PSM ir programos kodo. Tai leidžia kūrėjams savaip interpretuoti MDA transformacijas. OptimalJ įrankyje transformacijos atliekamos modeliams pritaikant transformavimo šablonus. OptimalJ šablonai skirstomi į transformacijų ir funkcionavimo šablonus. Transformavimo šablonai naudojami modelio elementams transformuoti į žymesnio lygio modelio elementus. Transformavimo šablonai skirstomi į Technologijos ir Realizavimo šablonus. Šie šablonai aprašo modelio transformavimą į modelį, ir modelio transformavimą į kodą. Kitaip tariant, Technologijos šablonai paverčia programos verslo specifikacijas į taikomosios programos lygmenį pritaikytą specifiniai technologijai. OptimalJ įrankis gali dirbti su Java ir JSP realizavimo technologijomis. Technologijos šablonai naudojami vaizdavimo, veiklos ir duomenų lygių komponentų kūrimui. Funkciniai šablonai modelyje naudojami produktyvumui aprašyti. Jų tikslas – suteikti programai pakartotinio panaudojimo galimybes. Šie šablonai taikomi pavieniuose modelio lygiuose.

Funkciniai šablonai būna trijų tipų:

- dalykinės srities šablonai PIM – tai dalykinės srities UML modelis, kuris naudojamas duomenų modelio lygyje.
- taikomųjų programų šablonai PSM – tai šablonai naudojami taikomųjų programų modeliams sudaryti.
- programos kodo šablonai. – tai kodo šablonai naudojami programos kodo generavimo metu. Jie aprašo taikomosios programos realizavimo strategijas.

Vartotojai, naudodami tiekiamų įrankių rinkinį gali modifikuoti realizavimo šablonus ir taikyti savo realizavimo strategijas. OptimalJ turi Meta modelį skirtą aukšto lygio realizavimo šablono struktūrai aprašyti. Šie modeliai sudaryti UML ir Meta-Object Facility pagrindu. MOF naudojamas visiems modelio tipams aprašyti. Tokiu būdu galima aprašyti pavyzdžiui EJB ir Web modelius, Meta modelius transformacijoms aprašyti. Vartotojai gali sudaryti savo norimus Meta modelius.

OptimalJ Realizacijos šablonai sudaromi TPL (*Template Pattern Language*) . TPL kalba sukurta palengvinti Realizacijos šablonų kodo rašymą. Ši kalba skirta MOF aprašomiems modeliams transformuoti į specifine sintakse aprašomas programavimo kalbas. Vykdant realizuotas transformacijas OptimalJ generuoja veikiančias taikomąsias programas. Galima išskirti tris pagrindinius darbo su OptimalJ aspektus:

- MDA koncepcija – MOF aprašytiems modeliams taikomi transformavimo šablonai
- Iteratyvus programinės įrangos kūrimas – šablonų modifikavimas ir

pritaikymas pasikeitusiems reikalavimams pakeičia kuriamą produktą, kur naudojami pasikeit šablonai.

- Skirtingi abstrakcijų lygiai – šablonai naudojami nuo verslo lygmens iki pat programinio kodo generavimo lygio.

2.1.4 Modelių sinchronizavimas

PSM modelio ir programos kodo sinchronizavimas atsiradus PIM modelyje pakeitimams vadinamas aktyvia sinchronizacija [8]. OptimalJ įrankis atlieka sinchronizaciją, kai sugeneruotą programos kodą keičia programuotojas. Ši sinchronizacija skirta programinio kodo pakeitimams atvaizduoti modelyje. Tokia sinchronizacija užtikrina, kad keičiant modelį, keisis žymesnio lygio modelis ir programos kodas. Aktyvi sinchronizacija atlieka dvi pagrindines funkcijas: atlieka PIM modelio sinchronizaciją su PSM modeliu, vykdo PSM ir kodo modelio sinchronizacijas. Sinchronizavimo metu reikalingi pakeitimai atliekami šablonų pagalba. OptimalJ gali atlikti sinchronizaciją tarp PSM ir Java kodo, tokiu būdu modelis tiksliai atvaizduos gaunamą programą. Atliekant sinchronizaciją, OptimalJ patikrina ar įvykdyti pakeitimai neišgadins modelio struktūros.

2.1.5 Integruotas diegimas

J2EE taikomųjų programų diegimas ir testavimas reikalauja tam tikro laiko ir gali būti sudėtingas specialistui, neturinčiam reikiamų įgūdžių. OptimalJ įrankyje realizuotas funkcionalumas padedantis išspręsti diegimo problemą ir pašalinti diegimo sunkumus. Taip pat OptimalJ supaprastina J2EE taikomųjų programų testavimą ir derinimą integruoto diegimo dėka. OptimalJ, kūrimo proceso metu automatiškai įdiegia taikomąją programą nurodytame programų serveryje. Kūrėjas gali nurodyti įdiegimui naudojamą programų serverį OptimalJ nustatymuose. Reikiamas įdiegimo aprašas bus sugeneruotas ir įtrauktas į programos įdiegimo archyvą. OptimalJ gali atlikti įdiegimą į J2EE specifikaciją atitinkančius serverius: JBoss, IBM WebSphere ir BEA WebLogic. OptimalJ turi integruotą testavimo aplinką, JBoss Open Source EJB serverį, Tomcat Web Serverį, Servlet Engine ir Solid duomenų bazę.

2.1.6 Tiekiamas funkcionalumas

Čia yra išvardinamas ne visas, o tik MDA taikomas OptimalJ funkcionalumas:

- PIM ir PSM modelių palaikymas
- J2EE platformos taikymas PSM modeliams.
- Modelių integravimo palaikymas
- Sistemos adaptavimo palaikymas

- Integravimas su kitais įrankiais – galimas UML modelių importavimas ir eksportavimas XMI failų pavidalu.
- Transformacijų modeliavimas naudojant TPL
- Šablonų taikymas
- Kodo restruktūrizavimo galimybė.

2.1.7 Kūrimo aplinka ir procesas

OptimalJ programinės įrangos kūrimo procese naudojami trys modeliai – dalykinės srities, taikomųjų programų ir kodo modeliai. Dalykinės srities modelis (*Domain Model*) skirtas aprašyti verslo veiklas ir verslo informaciją. Šis modelis neturi technologinių sprendimų ir yra verčiamas į PIM modelį. DM modelis aprašo sistemos funkcionalumą, nenaudojant realizacijos sprendimų. OptimalJ naudojami dviejų tipų DM modeliai: dalykinės srities klasių modelis ir dalykinės srities servisų modelis.

Pirmasis aprašo programos duomenų struktūrą ir atributus. Antrasis aprašo taikomosios programos tranzakcijas (pvz. užsakymo pateikimą, priėmimą ir kitus verslo veiksmus). Klasių modeliui galima taikyti įprastinius dalykinės srities šablonus, kurie padidins modelio išplėtimo galimybes. Naudojant Servisų modelį galima lengvai atlikti Session Beans komponentų generavimą.

Atlikus DM modelio sudarymą, OptimalJ automatiškai transformuoja jį į DM taikomosios programos modelį, kuris paskui verčiamas į J2EE PSM modelį. DM modelio transformavimas į PSM modelį aprašomas Technologiniais šablonais.

Transformacijos pasekoje gautas J2EE PSM trijų sluoksnių modelis. Jį sudaro:

- Web vaizdavimo modelis – aprašo duomenis reikalingus web puslapių generavimui.
- EJB veiklos logikos modelis – aprašo naudojamus tranzakcijų, duomenų apsaugos, saugojimo veiksmus.
- DBMS duomenų bazės modelis – aprašo duomenų saugojimui reikalingas struktūras.

Realizavimo šablonų pagalba, OptimalJ transformuoja PSM modelį į programos kodą. Kodas gali būti generuojamas visiems PSM modeliams vienu metu arba tik pasirinktiems modeliams.

2.1.8 OptimalJ integravimo su UML CASE įrankiais galimybė

Kuriant programinę įrangą su OptimalJ įrankiu, nereikalingi papildomi CASE įrankiai. OptimalJ turi UML modeliavimo įrankį, kurio pagalba sudaromi UML modeliai. Apsikeitimas modeliais tarp įrankių gali būti atliekamas XMI failų pagalba. Kitų integravimo

sąsajų OptimalJ neturi.

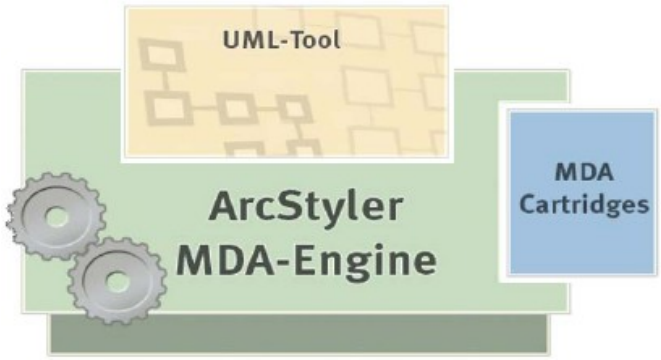
2.2 ArcStyler įrankio analizė

Interactive Object sukurtas ArcStyler įrankis, skirtas programinės įrangos kūrimui pagal MDA. Šis įrankis atlieka modelio į modelį ir į programos kodą transformacijas. Pradinis modelis aprašomas UML kalba. ArcStyler gali generuoti Java ir .NET platformoms skirtą programinį kodą. Modelio sudarymo procese vartotojas gali naudotis ArcStyler įrankio UML modelių kūrimo aplinka arba kitu UML CASE įrankiu. Taikomųjų programų kūrimas ArcStyler įrankių vykdomas trims žingsniais:

- Pradinio modelio sudarymas
- Norimos PSM technologijos pasirinkimas
- Transformavimas ir įdiegimas.

ArcStyler įrankį sudaro keturi pagrindiniai komponentai (5 pav.):

- UML modeliavimo komponentas
- MDA veiklos komponentas
- MDA kasetės
- UML/MOF/JMI saugykla



5 pav. ArcStyler įrankio architektūra [11]

2.2.1 UML komponentas modeliavimo kokybės kontrolei.

UML komponentas tai UML įrankis, turintis UML sąrankos ir MOF modeliavimo galimybes. ArcStyler Tolo Adapteri Standarto (TAS) API pagalba galimas kito (nebūtinai UML) įrankio integravimas su ArcStyler. Naudodamas ArcStyler XMI Edition vartotojas gali pritaikyti ArcStyler XMI generatorių darbui su standartiniais UML/XMI modeliais., priklausomai nuo to, kokios versijos XMI failas reikalingas UML CASE įrankiui.

2.2.2 MDA kasetės

Lengvai įdiegiamos ir praplečiamos MDA kasetės yra viena pagrindinių ArcStyler savybių. Šių kasečių viduje saugomi transformavimui reikalingi duomenys. Tai duomenys apie modelio transformavimo operacijas, validavimo algoritmus, šablonus, praplečiamus generatorius. Kelios MDA kasetės gali būti naudojamos vienu metu. Taip gali būti generuojamos multiplatforminės sistemos. ArcStyler įrankiui yra sukurtos Java, J2EE, C# ir .NET MDA kasetės. Esant poreikiui šios kasetės gali būti praplečiamos.

2.2.3 UML/MOF/JMI saugykla

Visi ArcStyler moduliai naudojasi bendra, atvira UML/MOF/JMI specifikacijas atitinkančia saugykla, kuri gali būti integruojama su kitomis, minėtas specifikacijas atitinkančiomis saugyklomis. Toks saugyklų integravimas, ne tik panaikina importavimo/eksportavimo problemas, bet ir leidžia naudoti ArcStyler MDA kasetėse saugomą informaciją kituose įrankiuose, keistis transformacijomis ir šablonais.

2.2.4 ArcStyler funkcionalumas

Pagal [11] ArcStyler tiekia tokį funkcionalumą:

- Taikomosios programos logikos atskyrimą nuo realizacijos – ArcStyler naudoja MDA metodus, kurie atskiria sistemos logikos kūrimo darbus, nuo jos realizacijos konkrečioje platformoje darbų .
- Galimybę naudotis ArcStyler UML modeliavimo įrankiais arba išoriniais UML CASE įrankiais modelio sudarymo proceso metu. Šis funkcionalumas leidžia projektuotojui naudotis jam įprastu ir patogiu UML CASE įrankiu, taip palengvinant modelių kūrimo procesą. Jeigu projektuotojas neturi UML CASE įrankio arba jis neprieinamas, jis visada gali pasinaudoti ArcStyler UML įrankiu. ArcStyler aplinkos UML įrankis leidžia sumažinti kaštus skirtus papildomiems UML CASE įrankiams įsigyti.
- MDA automatizuotas variklis, apdorojantis MDA kasetėse pateikiamus transformavimo duomenis, validavimo taisykles, projektavimo šablonus.
- Integruojama, atvira modelių saugykla – galimas saugomo modelio naudojimas ne tik UML CASE įrankiuose, atitinkančiuose saugyklos specifikaciją, bet ir kituose saugyklos sąsajas turinčiuose MDA įrankiuose.

2.2.5 Palaikomos platformos

ArcStyler generuoja Java/J2EE arba C# .NET projektus. Projekto generavimas priklauso nuo naudojamų MDA kasečių. Kodo generavimo taisyklių valdymas atliekamas

modifikuojant MDA kasečių duomenis. J2EE projektui, ArcStyler sugeneruoja EJB Java išeities kodą ir aprašus, JUnit modulių testus, SQL duomenų bazių skriptus, ANT projekto konstravimo ir įdiegimo skriptus, JBuilder, Eclipse ir kitų IDE projektų failus. .Net projektui sugeneruojami J2EE atitinkantis komponentai. Bet koks rankinis sugeneruoto programinio kodo keitimas iššaukia sinchronizavimo su modeliu veiksmus.

2.2.6 Programinės įrangos kūrimo procesas ArcStyler įrankiu

Programinės įrangos kūrimo procesą ArcStyler įrankiu galima suskirstyti į tris žingsnius:

- Pirmasis žingsnis - kuriamos sistemos modelio sudarymas aprašant verslo objektus, ryšius ir procesus tarp jų UML kalba. Pradinis sistemos modelis gali būti sukurtas nuo nulio, importuotas iš egzistuojančio UML modelio arba automatiškai sugeneruotas iš programos Java kodo. Modelio generavimas iš sukurtos sistemos leidžia praplėsti esamos sistemos funkcionalumą, modifikuoti sistemą pagal pasikeitusius reikalavimus arba perkelti sistemą į kitą platformą. Esamos sistemos perkėlimas į modeliavimo lygį su galimybe sugeneruoti sistemos išeities kodą sudaro geras sąlygas sistemos priežiūrai ir paveldėtų sistemų rekonstravimo darbams atlikti.
- Antras žingsnis – reikiamos platformos pasirinkimas. ArcStyler palaiko kelias populiarias platformas: IBM WebSphere, BEA WebLogic, Microsoft .NET. Galimas ArcStyler konfigūravimas ir pritaikymas specifiniai vartotojo platformai ar aplinkai.
- Paskutinis žingsnis transformavimo vykdymas, kurio metu sukurtas modelis transformuojamas į programos kodą, generuojama reikiama infrastruktūra, paleidžiami duomenų bazės, web serveriai, testavimo klientai ir web programos.

2.2.7 ArcStyler integravimas su kitais įrankiais

ArcStyler ir kitų UML CASE įrankių integravimas galimas ArcStyler modeliu saugyklos dėka. Ši saugykla yra atvira ir ja galima naudotis per JMI sąsają. Šios bendros saugyklos naudojimas leidžia keistis modeliais tarp ArcStyler ir skirtingų UML CASE įrankių. ArcStyler turi modelių importavimo ir eksportavimo XMI failų pavidalu funkciją. Į ankstesnes ArcStyler įrankio versijas buvo integruota Rational Rose UML modeliavimo aplinka. Dabartinis ArcStyler, UML modeliavimui, naudoja MagicDraw UML modeliavimo aplinką.

2.3 Eclipse modeliavimo aplinka

Eclipse modeliavimo aplinka (Eclipse Modeling Framework, toliau EMF) tai IBM kompanijos sukurta atviro kodo aplinka, skirta modeliais valdomam programinės įrangos kūrimui. Pradiniai modeliai EMF aplinkoje gali būti aprašomi UML arba XML schemomis. Modelio transformacijos į kodą metu generuojamas Java programinis kodas. Sugeneruotas Java kodas gali būti naudojamas pradinio modelio duomenų grafiniam atvaizdavimui ir manipuliavimui, nuskaitymui, išsaugojimui ir modifikavimui.

2.3.1 Eclipse modeliavimo aplinkos analizė

EMF aplinka sudaro:

EMF – Eclipse modeliavimo aplinkos branduolys. EMF branduolį sudaro:

- Ecore [12] meta modelis, skirtas modelių aprašymui.
- XMI generavimo ir nuskaitymo komponentas, atliekantis modelio išsaugojimo/nuskaitymo funkcijas.
- Programinė modelių valdymo sąsaja.

EMF.Edit – tai aplinka, skirta kurti EMF modelių redagavimo įrankius. EMF.Edit sudaro turinio, atributų ir kitų išeities kodo savybių klasės , kurios leidžia atvaizduoti EMF modelius standartinėmis (JFace) vaizdavimo priemonėmis, o jų savybes savybių kortelėse.

EMF.Codegen – tai EMF modelių redagavimo įrankių kodo generavimo aplinka. Šios aplinkos nustatymų keitimas ir generavimo proceso valdymas atliekamas grafinės vartotojo sąsajos pagalba. Kodo generavimui naudojamas JDT (*Java Development Tooling*) Eclipse komponentas.

Kodo generavimas skirstomas į tris lygius [12]:

- Modelio kodo generavimas – visoms modelio klasėms generuojamos Java sąsajų ir realizacijų klasės. Papildomai sugeneruojamos gamybos (Factory) ir paketo meta duomenų klasės.
- Pritaikymo klasių generavimas – tai realizavimo klasių, skirtų pritaikyti modelio klases atvaizdavimo ir modifikavimo veikloms, generavimas.
- Redagavimo komponento generavimas – sugeneruojamas modelio redagavimo komponentas, galintis modifikuoti Eclipse modeliavimo aplinkos modelius.

Vartotojui atliekant sugeneruotų klasių modifikavimą įvyksta klasių regeneravimas ir pakeitimų sinchronizavimas su sistemos modeliu.

2.3.2 Eclipse modeliavimo aplinkos funkcionalumas

EMF aplinka leidžia saugoti modelius ir kitus objektus XMI failuose tolesniam duomenų apsikeitimui su kitais įrankiais ir programų sistemomis. EMF modeliai gali būti kuriami Java arba XML schemų pagalba. Galimas išoriniu UML CASE įrankiu sukurto modelio importavimas į EMF. Kodo generatorius paverčia modelį Java klasių rinkiniu. Šios klasės gali būti praplečiamos ir regeneruojamos. Vartotojas gali papildyti šias klases norimais metodais ir atributais. Modifikavus modelį, vartotojas gali sugeneruoti šias klases iš naujo. Generuojamos bus tik tos klasės, kurias įtakojo modelio pakeitimai. EMF aplinkoje vykdoma abipuse modelio ir modelio klasių sinchronizacija – modelis keičiamas atlikus pakeitimus klasėse, o klasės yra keičiamos atlikus modelio pakeitimus.

2.3.3 Programinės įrangos kūrimas Eclipse modeliavimo aplinkoje

EMF Modelio sukūrimas gali būti atliekamas keliais būdais. Kadangi EMF aprašomas XMI dokumentu, galimi keli modelio sukūrimo būdai [12:

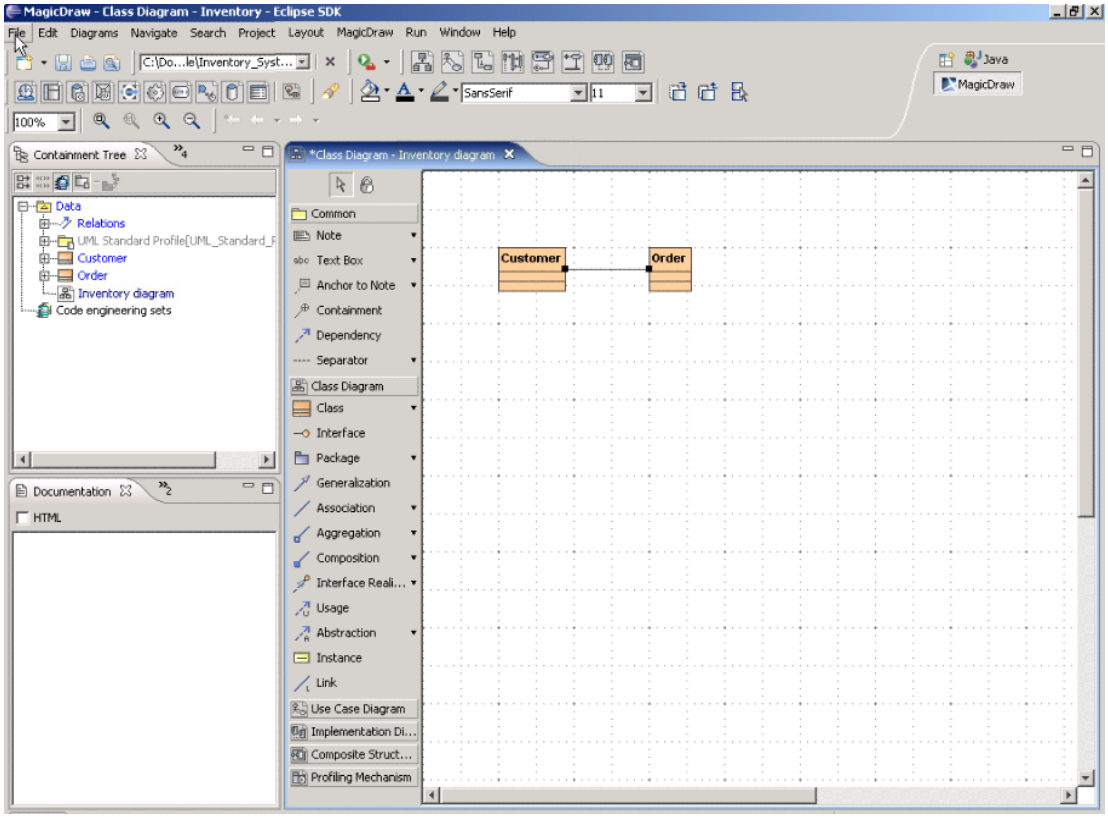
- Sukurti XMI dokumentą naudojant XML ar kitokį teksto redaktorių.
- Eksportuoti XMI dokumentą iš UML CASE įrankio.
- Aprašyti modelio savybes Java sąsajų pagalba.
- Panaudoti XML schemas aprašančias duomenų nuskaitymo taisykles ir nusiskaityti modelį iš atitinkamos saugyklos.

Pirmasis būdas nėra labai patogus. Jis yra skirtas vartotojams gerai išmanantiems XML sintaksę ir semantiką. Modelio XMI dokumento eksportavimui iš UML įrankio visų pirma reikalingas pats UML CASE įrankis. Šis būdas tinka vartotojams turintiems tam tikrų UML žinių ir darbo modeliavimo įrankiu, įgūdžius. Trečiasis būdas, aprašo patį paprasčiausią modelio sudarymo būdą naudojant Java sąsajas. Šios sąsajos gali būti kuriamos Eclipse aplinkoje arba kokiu nors kitu Java kūrimo įrankiu, ar net tekstiniu redaktoriumi. Šis būdas tinka vartotojams išmanantiems Java kalbos sintaksę. Ketvirtasis būdas rečiau naudojamas. Jis reikalauja programinės įrangos palaikančios tam tikro XML failo, aprašyto XML schema, sukūrimo.

2.3.4 Eclipse modeliavimo aplinkos integravimas su UML CASE įrankiais

Eclipse modeliavimo aplinkos modelis aprašomas XMI dokumentu, todėl galimas modelių apsikeitimas su kitais, XMI importavimo/eksportavimo funkcijas turinčiais UML CASE įrankiais. Eclipse integruota kūrimo aplinka gali būti praplečiama įskiepių pagalba. Lanksti Eclipse įskiepių sistema leidžia naudotis visu Eclipse funkcionalumu. Tokiu būdu galimas Eclipse įskiepio realizavimas, kuris atliktų sąsajos tarp UML CASE įrankio ir EMF aplinkos vaidmenį. Tokius įskiepius jau turi, kai kurie UML CASE įrankiai: ArgoUML,

Poseidon, MagicDraw, Rational Rose. MagicDraw UML CASE įrankio kūrėjai integravo savo įrankio UML modeliavimo komponentus į Eclipse aplinką (6 pav.).



pav. 6: MagicDraw integravimas į Eclipse

2.4 AndroMDA

AndroMDA – tai atviro kodo MDA įrankis, skirtas modelių ir programų kodo generavimui. Šis įrankis neturi savo UML modeliavimo aplinkos. Modeliai yra importuojami iš UML CASE įrankių. PSM modelio transformavimui į kodą naudojamos AndroMDA transformavimo kasetės.

2.4.1 Veikimo aprašymas

Šiuo metu AndroMDA neturi grafinės vartotojo aplinkos. Grafinė aplinka, skirta AndroMDA yra kūrimo stadijoje, todėl visos komandos iškviečiamos konsolės rėžime. Įrankio užduočių vykdymui naudojami Ant [13]arba Maven [14] automatizavimo įrankiai. Įvykdžius generavimo komandą, dialogo su vartotojo metu, sugeneruojamas projektas su reikiama katalogų struktūra ir pradiniu XMI modeliu. Sugeneruotas XMI modelis gali būti redaguojamas UML CASE įrankio aplinkoje. Modifikuotas XMI modelis naudojamas programos kodo generavimui.

2.4.2 Tiekiamas funkcionalumas

AndroMDA įrankio funkcionalumas apima importuoto modelio konvertavimą į kodą. Pradinis XMI modelis generuojamas konkrečiai vartotojo nurodytai platformai. Tokiu būdu praleidžiamas PIM modelio kūrimo etapas. Kodas generuojamas iš sugeneruoto XMI modelio.

Kodo generavimas atliekamas naudojant transformavimo kasetes. Transformavimo kasetės, tai transformavimo taisyklių, naudojamų šablonų, valdymo ir meta klasių rinkiniai. Dabartinėje AndroMDA įrankio versijoje (3.2) realizuotos transformavimo kasetės leidžiančios generuoti Spring, EJB, Webservices, Hibernate, Struts, JSF, Java, XSD kodą.

2.4.3 Programinės įrangos kūrimas AndroMDA įrankiu

Čia bus aprašyta tipinė sistemų kūrimo proceso AndroMDA įrankių veiksmų eiga Java platformai.

1. Sugeneruojamas naujas AndroMDA projektas įvykdant komandą „maven andromdapp:generate“
2. Dialogo su vartotoju metu nustatomos norimos konfigūracijos. Vartotojo klausiama, projekto pavadinimo, norimos platformos ir su platforma susijusių nustatymų.
3. Sugeneruoto PSM modelio XMI failas modifikuojamas UML CASE įrankių. AndroMDA kūrėjai siūlo tam naudoti MagicDraw UML CASE įrankį.
4. Modifikuotas PSM modelio XMI failas išsaugomas. Įvykdomos komandos skirtos programos kodo iš modelio generavimui.
5. Sugeneruojama pasirinktos platformos kodo katalogų struktūra, sukuriami atitinkami išeities kodo failai.

2.4.4 AndroMDA integravimas su UML CASE įrankiais

AndroMDA įrankio modelio apsikeitimas su UML CASE įrankiais vykdomas XMI failų pavidalu. XMI failai naudojami tik modelio įvedimui į AndroMDA. AndroMDA negeneruoja XMI modelių, išskyrus pradinio modelio XMI. AndroMDA XMI modelio importavimo į UML CASE įrankius bandymų rezultatai [15] parodė, kad geriausiai XMI modelio modifikavimui tinkamas MagicDraw UML modeliavimo įrankis. Taip pat įmanomas XMI modelio redagavimas ArgoUML, Poseidon ir Sparx Enterprise Architect UML CASE įrankiais.

Šiuo metu, yra kuriamas įskiepis Android [16], kurio pagalba AndroMDA bus integruojamas į Eclipse įrankio aplinką. Kitas sukurtas įskiepis skirtas ArgoUML [17] CASE įrankiui, leidžia naudotis AndroMDA įrankio funkcionalumu ArgoUML aplinkoje. Šis

įskiepis taip pat yra kūrimo stadijoje.

2.5 Codagen Architect

Šis MDA įrankis skirtas programų kodo generavimui iš modelio, todėl įrankis neturi UML modelio sudarymo galimybių. Tam yra naudojama Codagen Architect add-ins [18] praplėtimų sąsaja, leidžianti vartotojui modelio sudarymui, naudoti kitų gamintojų UML CASE įrankius. Codagen Architect atlieka tik modelio transformavimą į kodą. Generuojamas išėjimo kodas yra skirtas .NET ir Java platformoms.

2.5.1 Veikimo aprašymas

Codagen Architect įrankis transformuoja duomenų modelį į vieną ar daugiau vartotojo nurodytos platformos PSM modelių. Darbą su įrankiu galima aprašyti trimis fazėmis:

- Architektūros dizaino fazė
- Architektūros realizavimo fazė
- Kodo generavimo fazė

2.5.1.1.Architektūros dizaino fazė

Šios fazės metu projektuotojas turi sudaryti architektūros specifikaciją. Specifikacijos sudarymas susideda iš kelių veiksmų:

- Kuriamos sistemos, architektūros sluoksnių nustatymas.
- Kiekvienam architektūros sluoksniui nurodomos užduotys, kurias būtina atlikti sluoksnio realizavimui.
- Kiekvienai sluoksnio užduočiai nurodomi sprendimai. Šiuos spendimus pateikia programuotojai. Sprendimai gali remtis projektavimo šablono taikymu, stereotipų ir specifinių parametrų naudojimu.
- Kiekvienas sprendimas aprašomas UML.
- Kiekvienai užduoties ir jos sprendimo porai nurodomas sprendimas pagal nutylėjimą.

2.5.1.2.Architektūros realizavimo fazė

Architektūros realizavimo fazės metu projektuotojas sudaro XML architektūros specifikaciją, kuri naudojama Codagen XML transformacijos šablonams sudaryti. Šie šablonai aprašo PIM modelio transformavimo į PSM taisykles kartu su sluoksnių užduočių sprendimais.

2.5.1.3.Kodo generavimo fazė

Sudarytose Codagen XML transformacijos šablonuose projektuotojas gali pakeisti norimą sluoksnio užduoties sprendimą arba palikti sprendimą pagal nutylėjimą. PIM modelio

transformavimui į PSM modelį projektuotojas naudoja „software factory“ Codagen generavimo įrankį. Šis įrankis, nurodytam PIM modeliui pritaiko projektuotojo sukurtus Codagen XML transformacijos šablonus – tokiu būdu gaunamas PSM modelis. Codagen Architect įrankis gali generuoti PSM modelį skirtą J2EE, .NET ir CORBA platformoms.

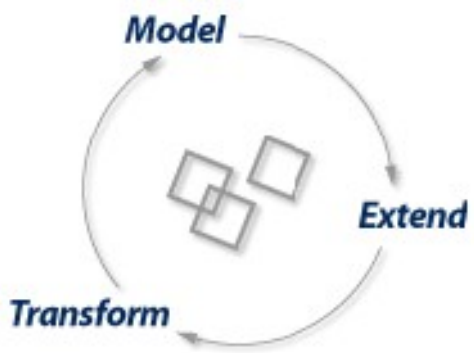
2.5.2 Tiekiamas funkcionalumas

Vienas iš svarbesnių Codagen Architect įrankio funkcionalumų – Codagen Technologijos akseleratorius. Technologijos akseleratoriai, tai paruošti transformacijų šablonai ir UML sąrankos, skirti naudoti platformai specifinių modelių generavimui. Šie akseleratoriai grupuojami pagal generuojamo modelio platformas [18] .

1. J2EE
    - Enterprise Java Beans (EJB) – EJB Technologijos akseleratorius leidžia vartotojui generuoti EJB komponentų programos kodą iš UML klasių diagramų. Akseleratorius gali generuoti Session Beans ir Entity Beans komponentus.
    - Oracle JDeveloper – šio akseleratoriaus pagalba galimas klasių diagramų transformavimas į JDeveloper duomenų bazės diagramas.
    - Java Server Pages – šis akseleratorius generuoja bazinį J2EE web puslapių sluoksnio kodą. Generavimui naudojami J2EE Web aplinkos šablonai, kurių pagrindą sudaro MVC architektūra. Šis akseleratorius sugeneruoja JSP puslapius, XML konfigūracijos failus vaizdinio ir valdiklio MVC šablono sluoksniams. Sugeneruojamos UML veiklos diagramos, vaizduojančios perėjimų tarp puslapių tvarką.
2. .NET
    - ADO.NET – ADO technologijos akseleratorius leidžia naudoti Codagen Architect duomenų bazės lygio UML klasių diagramas, duomenų bazės struktūros generavimui. Kartu generuojamas sukurtos duomenų bazės valdymo kodas.
    - ASP.NET – šis akseleratorius skirtas ASP.NET web programoms generuoti. Kaip ir JSP akseleratoriuje, naudojamas MVC architektūrinis šablonas. Šis generatorius sugeneruoja ASP.NET konfigūracijos failus, paprasčiausios vartotojo sąsajos HTML puslapius.

2.5.3 Programinės įrangos kūrimas Codagen Architect aplinkoje

Programų sistemų kūrimas Codagen Architect aplinkoje susideda iš modeliavimo (Model), praplėtimo (Extend) ir transformavimo (Transform) etapų (7 pav.)



7 pav. Sistemų kūrimo etapai Codagen Architect aplinkoje [18]

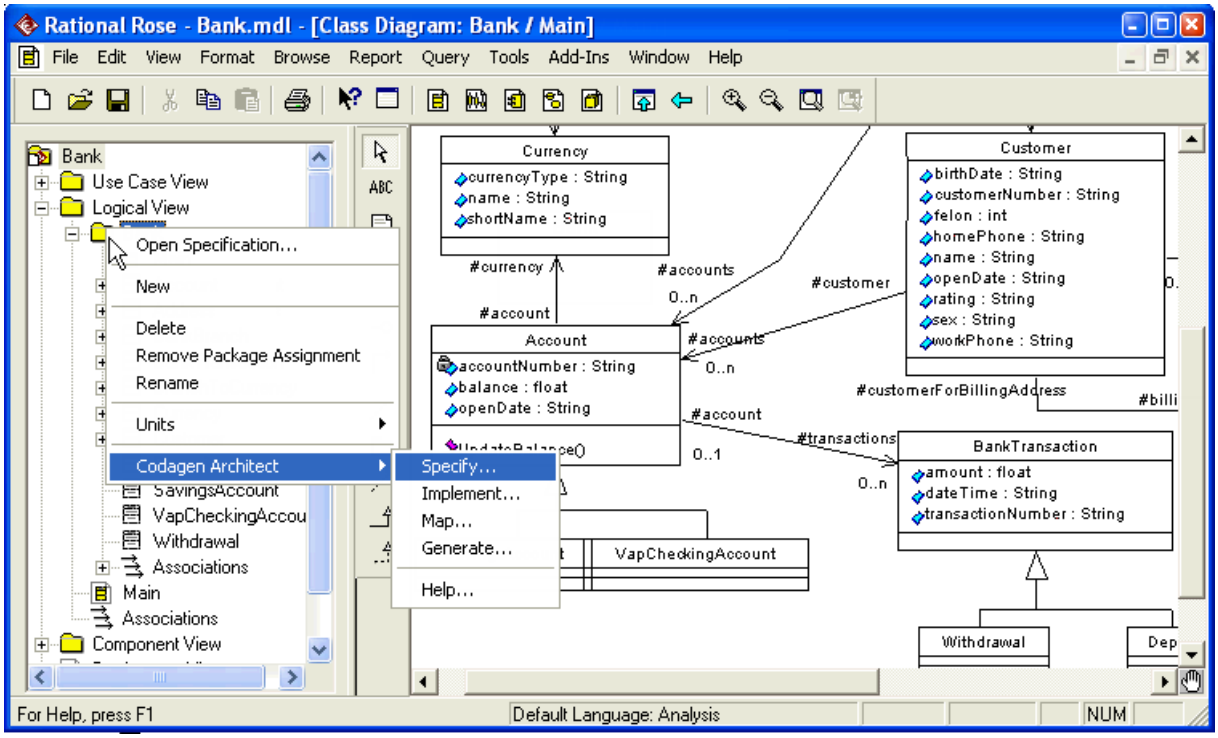
Modeliavimo etapo metu naudojamas išorinis UML CASE įrankis. Codagen Architect gali būti integruojamas į:

- IBM Rational Rose 2000
- Microsoft Visio UML – veikia su Microsoft Visio UML for Enterprise Architects versija Enterprise Architect programų sistemų pakete.
- TogetherTogether Control Center ne mažesnės nei 6.0 versijos.

UML CASE įrankio pagalba sudaromas kuriamos sistemos dalykinės srities modelis. Praplėtimo fazėje sudarytas modelis papildomas transformacijoms atlikti reikalingais atributais. Šie atributai aprašo vartotojo pasirinktos platformos architektūros dizaino sprendimus. Atitinkamos platformos technologijos akseleratoriaus pagalba, atributais praplėstas modelis transformuojamas į PSM modelį, iš kurio generuojamas programinis kodas.

2.5.4 Codagen Architect integravimas su UML CASE įrankiais

Codagen Architect neturi UML modeliavimo galimybių. Modelis yra importuojamas iš UML CASE įrankių. Modelio importavimas galimas XMI failo pavidalu. Codagen Architect integravimui į UML CASE įrankį, naudojama Codagen add-in sąsaja. Ši sąsaja leidžia naudotis Codagen Architect funkcionalumu iš kitų įrankių. Šiuo metu yra sukurti Codagen add-in įskiepiai skirti Rational Rose, Microsoft Visio ir Togeher Control Center. Codagen Architect funkcionalumo valdymas UML CASE įrankiuose atliekamas kontekstinio meniu pagalba (8 pav.).



8 pav. Integruoto Codagen Architect į Rational Rose funkcionalumo valdymas

2.6 Analizės apibendrinimas

Atlikus MDA įrankių apžvalgą buvo nustatyta, kad ne visi apžvelgti įrankiai, palaiko modelio į modelį transformacijas (lentelė 1), kas yra vienas pagrindinių MDA principų [6]. PIM modelio transformacijai į PSM MDA įrankių gamintojai skiria mažiau dėmesio. Daugiau dėmesio skiriama specifinės platformos kodo generavimui.

Visi nagrinėti MDA įrankiai generuoja kodą skirtą J2EE platformai (Lentelė 1). Galima manyti, kad ateityje J2EE platforma taps generuojamo kodo de-fakto standartu.

Bendras XMI failų naudojimas ne visada užtikrina apsikeitimą modeliais tarp įrankių. Šios problemos priežastis – įrankių skirtingų XMI versijų palaikymas. Ši problema pasireiškė modifikuojant UML CASE įrankiu AndroMDA sugeneruotą XMI modelį. Iš 22 testuojamų UML įrankių, tik 4 įrankiai galėjo modifikuoti AndroMDA XMI modelį[15]. Ir tik MagicDraw tai atliko visiškai teisingai [15].

Apžvelgus MDA įrankių integravimo su UML CASE įrankiais galimybes pastebėtas abipusis įrankių integravimas (lentelė 2). MDA įrankių funkcionalumas papildo UML CASE įrankių funkcionalumą ir atvirkščiai – UML CASE įrankiai praplečia MDA įrankių modeliavimo galimybes. Galima teigti, kad netolimoje ateityje skirtumai tarp MDA ir modeliavimo įrankių išnyks. Kaip pavyzdį galima pateikti OptimalJ MDA įrankį.

Šiame darbe nagrinėjamai DIM modelio į PIM transformavimo problemai spręsti galima

naudoti OptimalJ ir ArcStyler MDA įrankius.

Lentelė 1: MDA įrankių transformavimo galimybės

MDA įrankis	Transformacijos		PSM platforma
	Modelis -> Modelis	Modelis -> Kodas	
OptimalJ	Yra	Yra	J2EE
ArcStyler	Yra	Yra	J2EE
Eclipse EMF	Nėra	Yra	J2EE, .NET
AndroMDA	Nėra	Yra	J2EE, .NET
Codagen Architect	Nėra	Yra	CORBA, J2EE, .NET

Lentelė 2: MDA įrankių integravimo su CASE įrankiais galimybės.

MDA įrankis	Integravimas su UML CASE įrankiais			
	Nuosavas įrankis	Integruojasi į CASE įrankius	CASE įrankiai integruojasi į	Bendras duomenų formatas/sąsaja
OptimalJ	Yra	Ne	Ne	XMI
ArcStyler	Yra	Ne	Taip	XMI, ATL, JMI
Eclipse EMF	Nėra	Ne	Taip	XMI, PDE
AndroMDA	Nėra	Taip	Ne	XMI, Profiles
Codagen Architect	Nėra	Taip	Ne	XMI, Adds-in

2.7 Analizės išvados

- Analizėje apžvelgti MDA įrankiai nepalaiko pilnos MDA kūrimo proceso koncepcijos.
- MDA įrankių kūrėjai savaip interpretuoja ir realizuoja MDA kūrimo koncepcijos principus.
- Nagrinėjami MDA įrankiai orientuoti į kodo generavimą iš PSM modelių – tik du įrankiai palaiko modelio į modelį transformaciją.
- Visi nagrinėti įrankiai atlieka kodo generavimą skirtą J2EE platformai – J2EE tampa pagrindine platforma.
- Įrankių apsikeitimas modeliais aprašytais XMI, ne visada įmanomas dėl skirtingų įrankių palaikomų XMI versijų.
- DIM modelio į PIM transformavimui atlikti galima naudoti OptimalJ ir ArcStyler MDA įrankius.

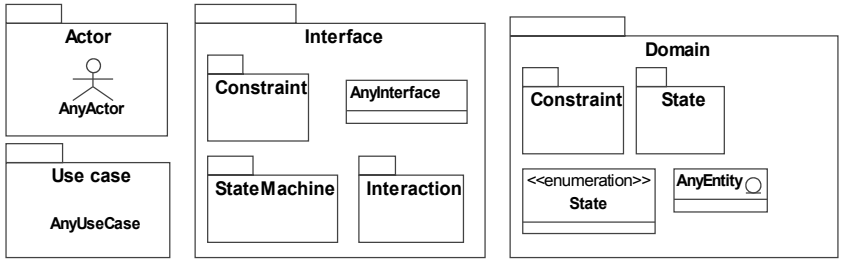
3. REIKALAVIMO MODELIO TRANSFORMAVIMO Į PROJEKTĄ METODIKA

3.1 Reikalavimų modelis DIM ir jo sudarymo metodika

Reikalavimų modelio sudarymui naudojamos panaudojimo atvejų, klasių ir būsenų diagramos. Šis reikalavimų modelis buvo pavadintas DIM (angl. *Design Independent Model*). Tai detalizuotas reikalavimų modelis, kuris išsamiai aprašo kuriamos sistemos struktūrą ir elgseną ir skiriasi nuo projekto modelio tuo, kad jame dar nėra projektinių sprendimų [2]. Tai reiškia, kad DIM neturi būti valdymo ar komponentų klasių. DIM modelį sudaro dalykinės srities esybės ir elgsenos sąsajos, apimančios konceptualių operacijų aibes. Sudarant reikalavimų modelį DIM, kiekvienam panaudojimo atvejui specifikuojamos sistemos sąsajos, kurios aprašo operacijų aibę. Kiekvienos operacijos specifikaciją sudaro jos signatūra, išankstinės (*pre-condition*) ir rezultato (*post-condition*) sąlygos; tarp operacijų ir sąsajų galimi apribojimai.

Programinėje realizacijoje naudojamas DIM modelis, kurį sudaro paketai, pavaizduoti paveiksle 9:

- *Actor* – paketas aprašantis aktorių modelį, kuris vaizduoja projektuojamos sistemos aktorius ir ryšius tarp jų.
- *Domain* – dalykinės srities modelio paketas, skirtas sistemoje naudojamoms esybėms saugoti. Šio paketo viduje sukuriamas *State* paketas, kuriame saugomos esybių būsenos ir jų kaitos scenarijai. Esybės gali turėti apribojimus – invariantus, kurie saugomi pakete *Constraint*.
- *Interface* – sistemos sąsajų modelio paketas, aprašantis sistemos aktoriams tiekiamas paslaugas ir paslaugų tarpusavio ryšius. *Interface* pakete yra sukuriamas apribojimų, išankstinių ir rezultato sąlygų paketas *Constraint;* būsenų mašinos ir sąveikos saugomos paketuose *StateMachine* ir *Interaction*.



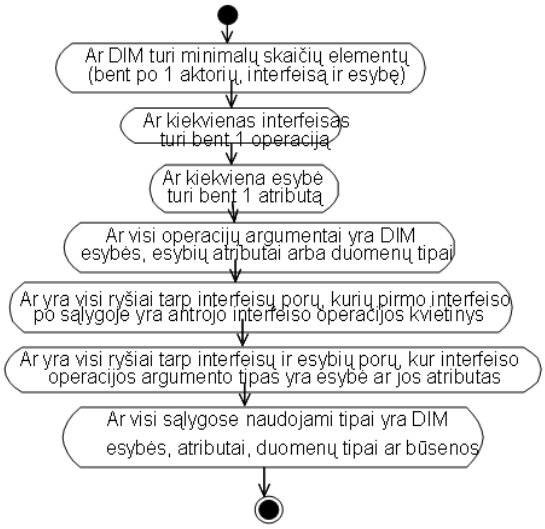
9 pav. DIM sudėtis

Reikalavimų modelį sudaro sąsajos, esybės, jų tarpusavio ryšiai, apribojimai, atributų ir operacijų specifikacijos, esybių būsenų vardai. Būsenų mašinos ir sąveikos aprašo sistemos

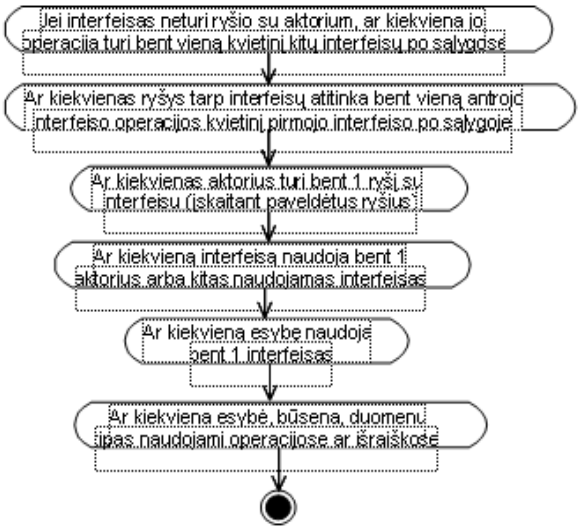
elgseną ir sukuriamos tam, kad galutinės specifikacijos elementai būtų tarpusavyje suderinti. Modelio suderinamumą užtikrina abipusis atvaizdavimas tarp sąveikų modelio ir būsenų mašinų [19], [20], [21]. Šiame darbe laikoma, kad DIM suderinamumas patikrintas. Tačiau tikrinamos jo savybės, kurių neužtikrina sąveikų ir būsenų suderinimo algoritmas.

3.2 DIM modelio patikra

DIM modelio tikrinimas turėtų apimti modelio išsamumo, susietumo, nepertekliškumo ir sintaksės patikrinimą. Kol kas dar nėra sukurtos konkrečios metodikos, skirtos pilnam DIM modelio tikrinimui atlikti. Galimos modelio išsamumo ir pertekliškumo tikrinimo veiklų sekos pavaizduotos paveiksluose 10 ir 11. Modelio išsamumo patikra turėtų nustatyti ar yra visi reikalingi elementai. Pertekliškumo tikrinimas nustatytų, ar modelyje nėra nereikalingų elementų.



pav. 10 Galimos modelio išsamumo tikrinimo veiklos.



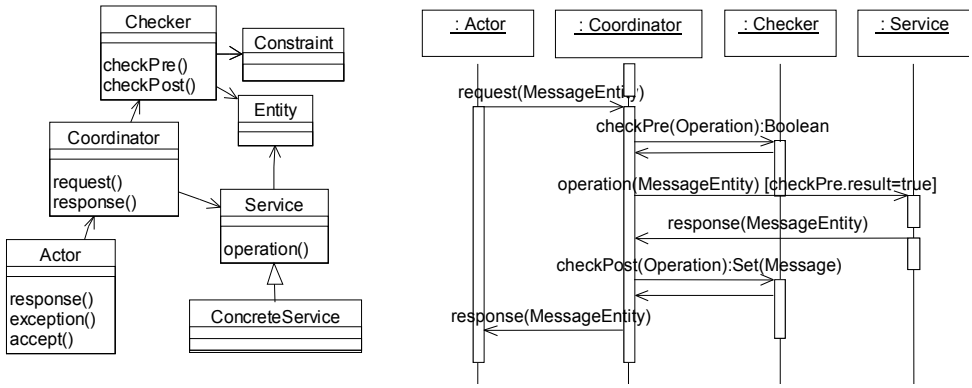
pav. 11 Galimos modelio perteklingumo tikrinimo veiklos.

3.3 Projekto modelis (PIM)

PIM – tai modelis, nepriklausantis nuo konkrečios realizavimo technologijos, sudarytas nenaudojant specifiniai programavimo kalbai būdingų duomenų tipų, struktūrų ir kitų atributų. PIM modelis aprašomas UML kalba, kas leidžia sudaryti modelį naudojant bendrus objektinės programavimo kalbos aprašus ir duomenų tipus. DIM modelyje yra aprašytos projektuojamos sistemos tiekiamos funkcijos. Tuo tarpu, PIM modelyje pateikiami sistemos funkcionalumo realizavimui reikalingi projektavimo sprendimai. Tai galėtų būti klasikiniai GOF [22] arba naujai sukurti projektavimo šablonai. Sistemos funkcionalumo realizavimo projektavimui galimas bendras šių šablonų naudojimas.

3.4 DIM į PIM transformacija

DIM modelio transformacija į projekto modelį PIM atliekama modelio elementams taikant transformavimo taisykles pagal būsenų koordinatoriaus (angl. *State Coordinator)* šabloną [3]. *State Coordinator* šablonas jungia standartinius būsenų (angl. *State)* ir fasado (angl. *Facade*) šablonus. Šis šablonas aprašo veiklos procesų vykdymo logikos valdymą (12 pav.).



12 pav. *State Coordinator* šablono elementai ir veikimo principas

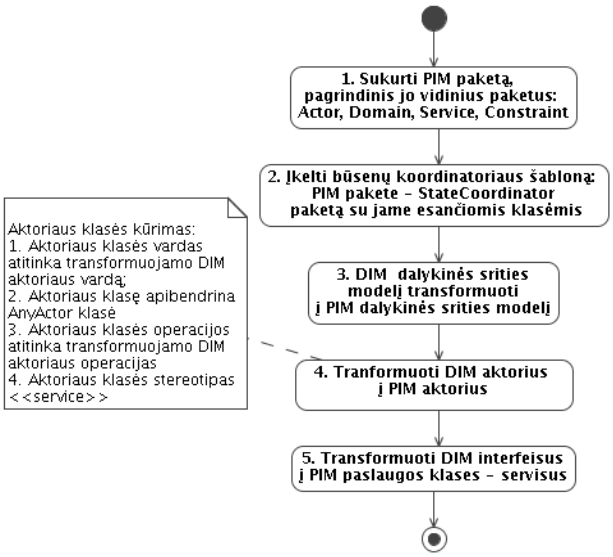
Norėdamas atlikti operaciją, aktorius siunčia užklausą koordinatoriui. Iš užklausos pranešimo koordinatorius išskiria operacijos, kurią reikia iškviesti, pavadinimą ir kreipiasi į tikrintoją (angl. *Checker*), kuris grąžina kviečiamos operacijos išankstinės sąlygos patikros rezultatus. Jeigu išankstinė sąlyga patenkinta, koordinatorius iškviečia užklausoje nurodytą reikiamos paslaugos operaciją ir tikrina pranešimo sąlygą: kokius pranešimus reikia siųsti po operacijos iškvietimo. Atėjus atsakui, *Coordinator* priima jį, kaip, bet kurį kitą pranešimą. Jei tai galutinis atsakas, pranešimas siunčiamas aktoriui. Operacijos išankstinės ir rezultato

sąlygos aprašomos OCL kalba [23]. Tikrinant išankstines sąlygas, duomenys imami iš paslaugų sistemos duomenų bazės. Sąlygos saugomos atskirai nuo paslaugų ir apdorojamos atskiru, tam skirtu komponentu (*Checker*), todėl pasikeitus sistemos veiklos taisyklėms, pačios paslaugos gali likti nepakitusios, keičiasi tik jų naudojimo sąlygos. Sistemos sąsajos išlieka tokios pat, kaip reikalavimų modelyje.

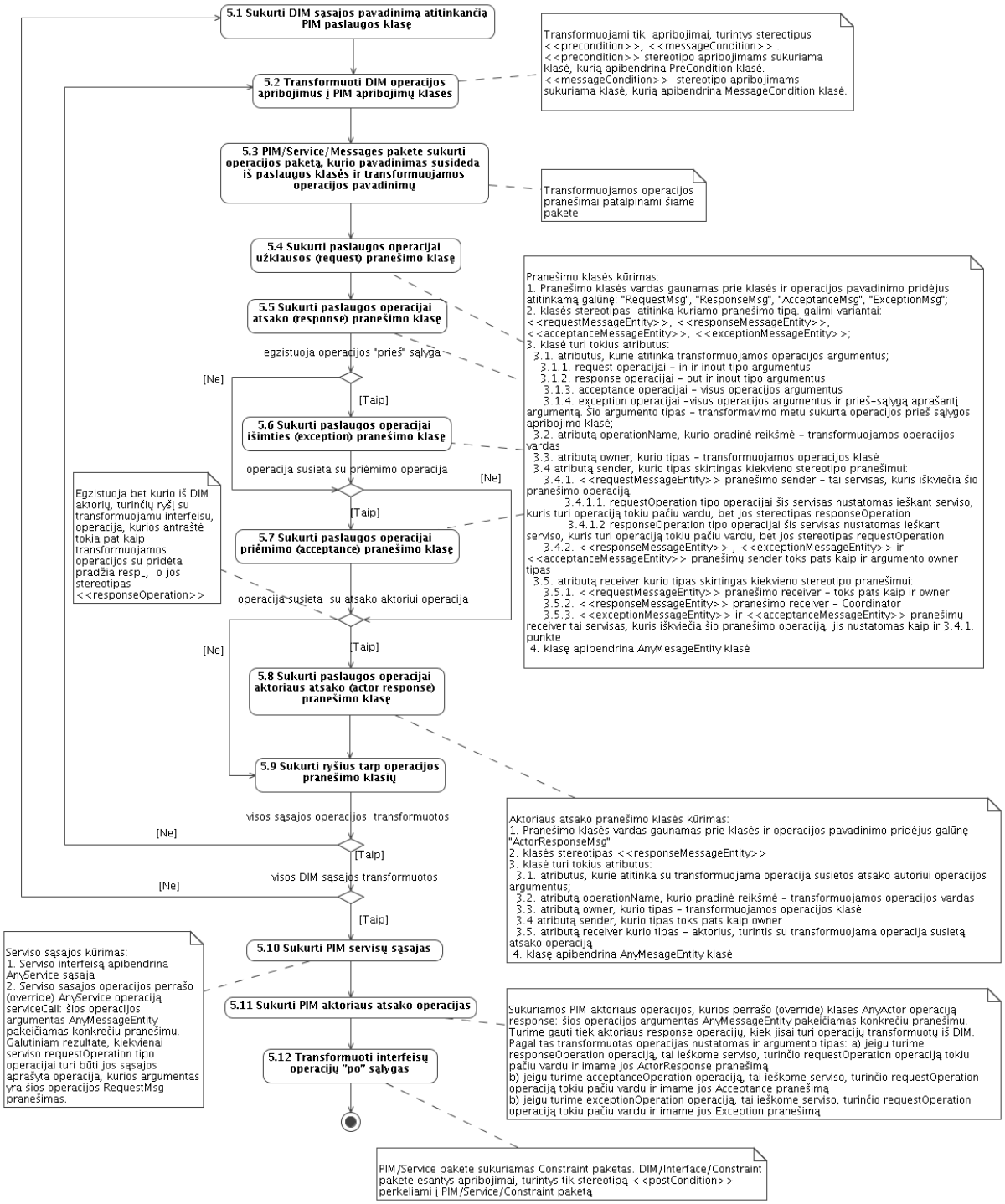
DIM transformacija į PIM modelį vykdoma pagal [3] aprašytas transformacijos taisykles, kurios nusako kiekvieno DIM modelio elemento transformavimą į PIM modelio elementą. DIM modelio transformacija atliekama tokia tvarka (13 pav.):

1. PIM pakete sukuriami reikiami vidiniai paketai.
2. Į PIM modelį įkeliamas *State Coordinator* šablonas.
3. DIM modelio dalykinės srities modelis transformuojamas į PIM dalykinės srities modelį be pokyčių.
4. DIM aktoriai transformuojami į PIM aktorius realizuojančius *StateCoordinator AnyActor* sąsają.
5. DIM modelio sąsajos transformuojamos į PIM paslaugų klases – servisus. Šios klasės realizuoja *State Coordinator* šablono *Service* sąsają. Išorinės paslaugų sąsajos perkeliamos iš DIM į PIM modelį.

    - DIM sąsajų operacijų išankstinės sąlygos ir pranešimų sąlygos (rezultato sąlygų dalis) transformuojami į PIM *Constraint,* kurias tikrina *Checker*. Likusi rezultato sąlygų dalis transformuojama į paslaugų klasių operacijų metodų apibrėžimus (*BodyConditions*).
    - Kiekvienai serviso operacijai sugeneruojami užklausos, atsako, išimties, priėmimo ir aktoriaus atsako pranešimai. Paslaugų klasių pranešimų generavimo algoritmas pateiktas paveiksle 14.

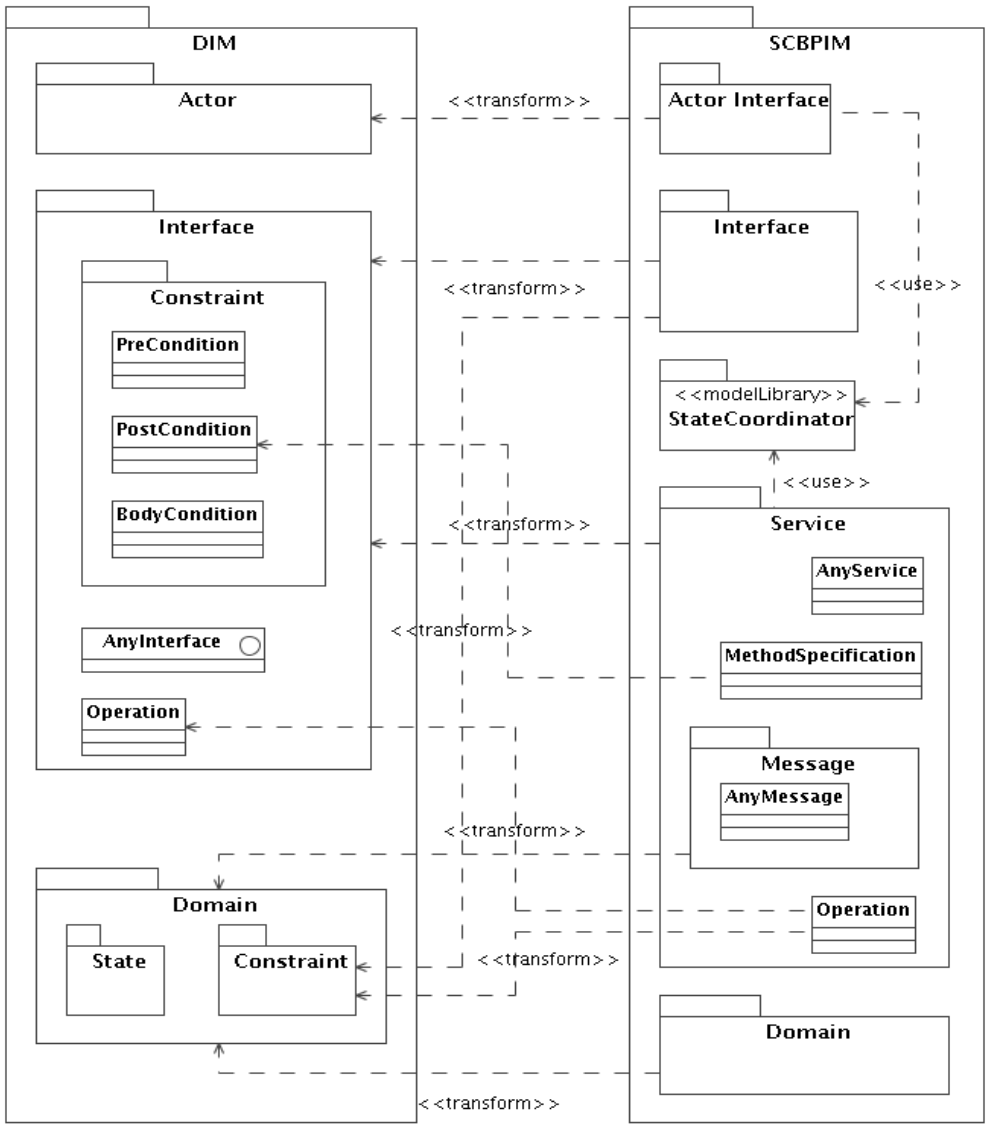


13 pav. DIM į PIM modelio transformavimo veiksmų seka



14 pav. Paslaugų klasės pranešimų generavimas

Paveiksle 15 vaizduojami transformacijos metu gauti PIM modelio paketai ir jų priklausomybės nuo DIM modelio paketų .



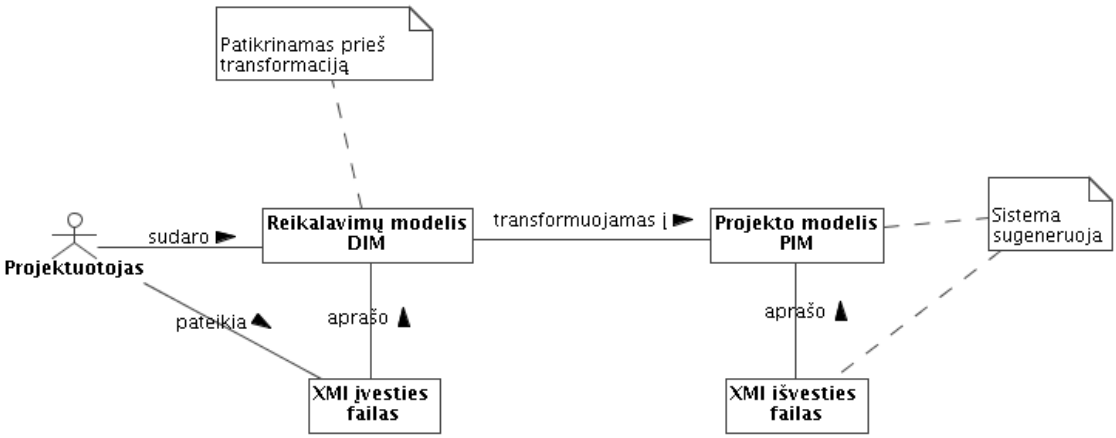
15 pav. Principinė DIM transformacijos į PIM schema

DIM sudarymo, tikrinimo ir transformavimo į PIM metodika realizuota DIM2PIM įrankyje. Metodikos realizacija aprašyta projekto dalyje.

4. CASE ĮRANKIO IŠPLĖTIMO PROJEKTAS

4.1 Funkciniai reikalavimai sistemai

Kuriama sistema turi atlikti reikalavimų modelio DIM transformavimą į projekto modelį. Prieš transformavimą turi būti atliekamas reikalavimų modelio verifikavimas siekiant patikrinti modelio sintaksę, išsamumą, susietumą ir nepertekliškumą. Reikalavimų modelis gali būti įvedamas UML CASE įrankio pagalba grafiniame pavidale arba nuskaitomas iš XMI failo. Transformacijos pabaigoje gautas projekto modelis PIM pateikiamas CASE įrankyje grafiniu pavidalu arba atvaizduojamas XMI išeities failu, kuris gali būti importuojamas į UML CASE įrankį tolesniam sistemos kūrimui. Kuriamos sistemos koncepcinė schema pavaizduota paveiksle 16.



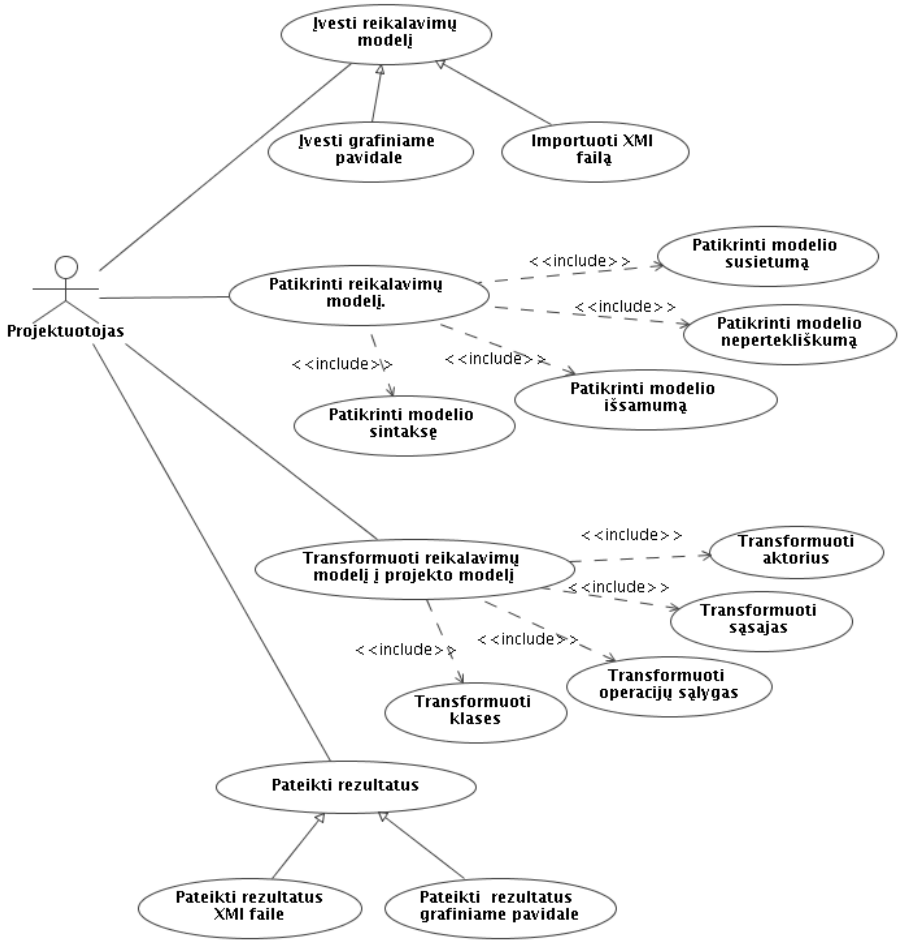
16 pav. Sistemos konceptualus modelis.

Iš užsakovo pateiktų reikalavimų buvo sudarytas panaudos atvejų sąrašas:

1. Įvesti reikalavimų modelį. Įvedimas galimas tiesiogiai arba importuojant XMI failą , pavyzdžiui, iš Rational Rose, Poseidon, ArgoUML.
2. Patikrinti reikalavimų modelį. Patikrinimas susideda iš:
    - Patikrinti modelio apribojimų sintaksę – OCL išraiškų tikrinimas.
    - Patikrinti modelio išsamumą – ar visi reikiami elementai apibrėžti (pvz., operacijų argumentai).
    - Patikrinti nepertekliškumą – ar nėra elementų (esybių, operacijų), kurie niekur nenaudojami.
    - Patikrinti susietumą – ar yra klasių ryšiai, atitinkantys pranešimų siuntimus operacijų rezultato sąlygose

3. Transformuoti reikalavimų modelį į projekto modelį. Transformavimas bus atliekamas, pritaikant užsakovo pateiktą algoritmą. Transformavimas susideda iš tokių panaudojimo atvejų:
   3.1. Transformuoti aktorius – sistemos aktoriai transformuojami į aktorių paslaugų klases.
   3.2. Transformuoti sąsajas – sąsajos virsta servisais.
   3.3. Transformuoti operacijų sąlygas – išankstinių ir rezultato operacijų sąlygų transformavimas.
   3.4. Transformuoti klases – reikalavimų modelio klasės virsta projekto modelio klasėmis
4. Pateikti gautą projekto modelį.
5. PIM modelis pateikiamas grafiniame pavidale.
6. PIM modelis pateikiamas XMI failo pavidalu.

Panaudos atvejų vaizdas pateikiamas paveiksle 17.



17 pav. Sistemos panaudos atvejų diagrama

4.2 Nefunkciniai reikalavimai sistemai

4.2.1 Reikalavimai sistemos išvaizdai (Look and feel)

Reikalavimas #: 8	**Reikalavimo tipas: 5.0**	**Įvykis/panaudojimo atvejis #:** 1-4	
Aprašymas:	Grafinė vartotojo sąsaja turi atrodyti vienodai visose platformose.		
Pagrindimas:	Projektuotojas turi lengvai išmokti dirbti su sistema skirtingose platformose.		
Šaltinis:	Projektuotojas.		
Tinkamumo kriterijus:	Grafinė vartotojo sąsajos išvaizda visose sistemos veikimo platformose vienoda.		
Užsakovo patenkinimas:	3	**Užsakovo nepatenkinimas:**	2
Priklausomybės:	Nėra	**Konfliktai:**	Nėra.
Papildoma medžiaga:	Nėra.		
Istorija:	Užregistruotas 2005m. kovo 10 d.		

Reikalavimas #: 9	**Reikalavimo tipas: 5.0**	**Įvykis/panaudojimo atvejis #:** 1-4	
Aprašymas:	Grafinės sąsajos komponentai turi būti išdėstomi dinamiškai, atsižvelgiant į antraščių ilgį, šrifto dydį ir naudojamą monitoriaus raišką.		
Pagrindimas:	Kuriant multiplatforminę sistemą reikalingas automatinis grafinės sąsajos komponentų prisitaikymas prie esamos platformos.		
Šaltinis:	Projektuotojas.		
Tinkamumo kriterijus:	Grafinė vartotojo sąsaja veikia teisingai su skirtingais antraščių ilgiais, šrifto dydžiais ir monitoriaus raiškomis.		
Užsakovo patenkinimas:	5	**Užsakovo nepatenkinimas:**	5
Priklausomybės:	Nėra	**Konfliktai:**	Nėra.
Papildoma medžiaga:	Nėra.		
Istorija:	Užregistruotas 2005m. kovo 10 d.		

4.2.2 Reikalavimai panaudojamumui (Usability)

Reikalavimas #:10	**Reikalavimo tipas: 6.0**	**Įvykis/panaudojimo atvejis #:** 1.0	
Aprašymas:	Sistemoje turi būti patogus duomenų apie reikalavimų modelį įvedimas,		
Pagrindimas:	Projektuotojui neturi kilti sunkumų įvedant reikalavimų modelį dėl programos sudėtingumo.		
Šaltinis:	Projektuotojas.		
Tinkamumo kriterijus:	Projektuotojai intuityviai supranta ir lengvai išmoksta įvesti reikalavimų modelį į sistemą.		
Užsakovo patenkinimas:	4	**Užsakovo nepatenkinimas:**	3
Priklausomybės:	Nėra	**Konfliktai:**	Nėra.
Papildoma medžiaga:	Nėra.		
Istorija:	Užregistruotas 2005m. kovo 10 d.		

Reikalavimas #:11	**Reikalavimo tipas: 6.0**	**Įvykis/panaudojimo atvejis #:** 1.0	
Aprašymas:	Įvedamas reikalavimų modelis gali turėti esybių pavadinimus ir kitus žymėjimus įvairiomis nacionalinėmis kalbomis.		
Pagrindimas:	Reikalavimų modelio aprašymui gali būti naudojamos įvairios kalbos, nes gali pasitaikyti projektuotojų kalbančių kitomis kalbomis.		
Šaltinis:	Projektuotojas.		
Tinkamumo kriterijus:	Sistema veikia teisingai nepriklausomai nuo specifikacijose naudojamų kalbų.		
Užsakovo patenkinimas:	4	**Užsakovo nepatenkinimas:**	4
Priklausomybės:	Nėra	**Konfliktai:**	Nėra.
Papildoma medžiaga:	Nėra.		
Istorija:	Užregistruotas 2005m. kovo 10 d.		

Kuriamajai sistemai greičio reikalavimai nėra taikomi, bet algoritmo programinė realizacija turi būti optimizuota, kad neverstų vartotojo pernelyg ilgai laukti darbo rezultatų.

4.2.3 Reikalavimai sistemos priežiūrai (Maintainability and portability)

Reikalavimas #:12	**Reikalavimo tipas: 7.0**	**Įvykis/panaudojimo atvejis #:** Nėra	
Aprašymas:	Kuriama sistema lengvai papildoma naujais skaičiavimo algoritmais.		
Pagrindimas:	Modelių transformavimo ir validavimo algoritmai turi būti realizuoti kaip atskiras, nepriklausomas ir lengvai modifikuojamas sistemos modulis.		
Šaltinis:	Sistemos administratorius.		
Tinkamumo kriterijus:	Lengvas naujų algoritmų papildymas ir modifikavimas.		
Užsakovo patenkinimas:	5	**Užsakovo nepatenkinimas:**	5
Priklausomybės:	Nėra	**Konfliktai:**	Nėra.
Papildoma medžiaga:	Nėra.		
Istorija:	Užregistruotas 2005m. kovo 10 d.		

4.2.4 Teisiniai reikalavimai

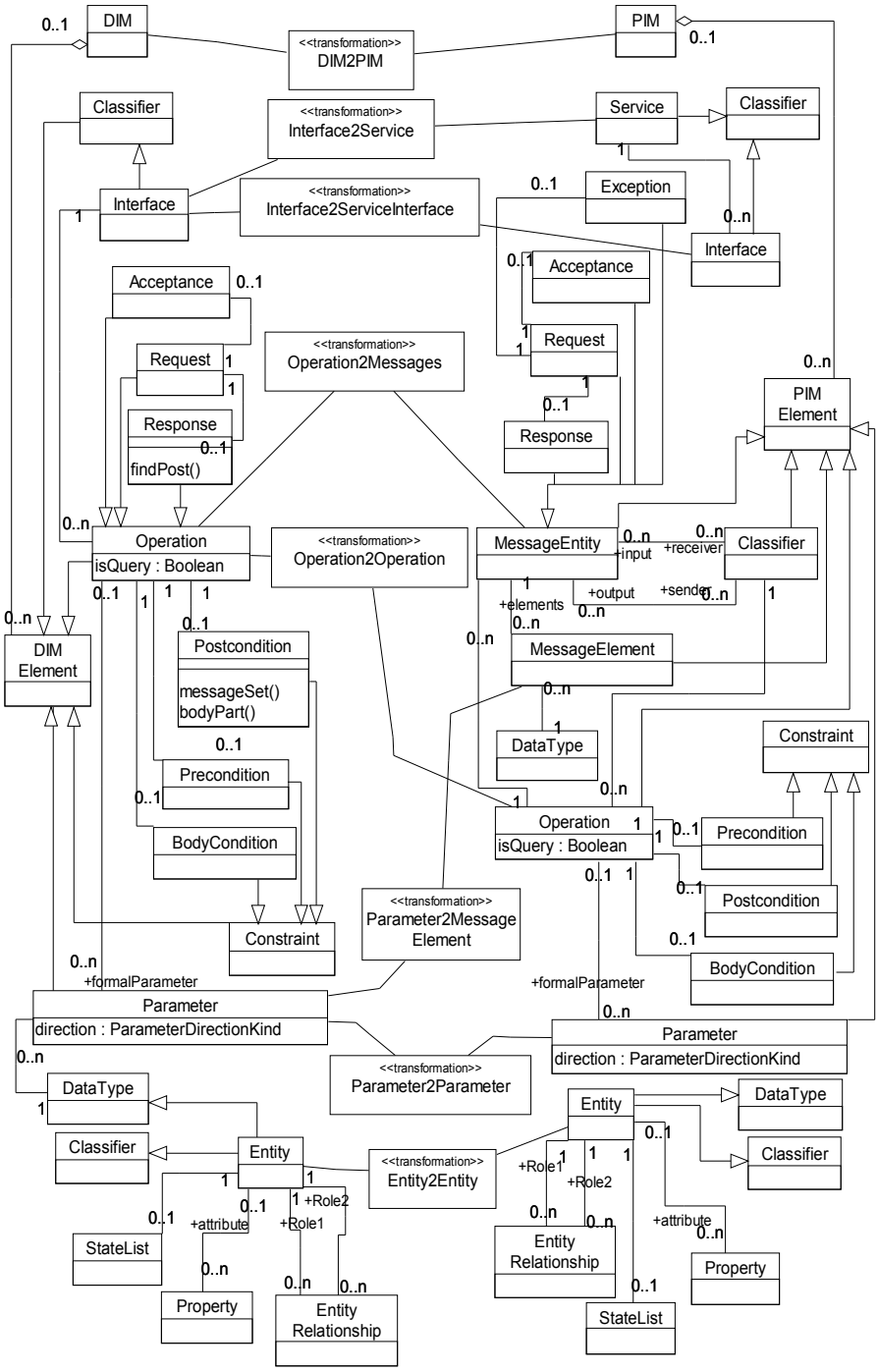
Reikalavimas #:13	**Reikalavimo tipas: 7.0**	**Įvykis/panaudojimo atvejis #:**	
Aprašymas:	Visos technologijos ir produktai panaudoti sistemos kūrimui turi būti laisvai prieinami ir nemokami.		
Pagrindimas:	Sistema turi būti kuriama nemokamų produktų pagrindu, kas leistų sumažinti iki minimumo licenzijų pirkimo kaštus.		
Šaltinis:	Užsakovas.		
Tinkamumo kriterijus:	Visos sistemai sukurti reikalingos technologijos ir produktai yra nemokami.		
Užsakovo patenkinimas:	5	**Užsakovo nepatenkinimas:**	5
Priklausomybės:	Nėra	**Konfliktai:**	Nėra.
Papildoma medžiaga:	Nėra.		
Istorija:	Užregistruotas 2005m. kovo 10 d.		

4.3 Duomenų struktūra

Sistemoje naudojama užsakovo pateikta duomenų struktūra (pav. 18). Ši duomenų struktūra skirta atvaizduoti probleminės srities reikalavimų ir projekto modeliuose

naudojamus duomenis. Pateiktas duomenų modelis, tai praplėsta OMG UML specifikacijoje aprašoma struktūra, reikalinga šiame darbe nagrinėjamos problemos sprendimui.

Kuriamos sistemos analizės darbo metu, buvo nustatytas dar vienas reikalavimas duomenų modeliui – duomenų modelio atitikimas Java Meta data Interface (JMI) specifikacija. Šią specifikaciją atitinkantis duomenų modelis gali būti naudojamas kartu su JMI sąsaja turinčiais modeliavimo įrankiais, XMI failų generatoriais ir analizatoriais.



18 pav. Sistemos duomenų modelis

4.4 Sistemos architektūra

Kuriamos sistemos architektūros tikslai:

- Nepriklausomumas nuo platformos – sistema turi veikti keliose plačiai naudojamose platformose.
- Daugiakalbiškumas - sistemoje galima naudoti skirtingų kalbų simbolius duomenų įvedimui ir išvedimui.
- Sistemos integravimo su kitomis sistemomis galimybė – sistema gali būti integruota su kitomis sistemomis, kaip posistemė arba veikti, kaip atskira sistema bendraujanti su sistemom per tam tikras sąsajas.

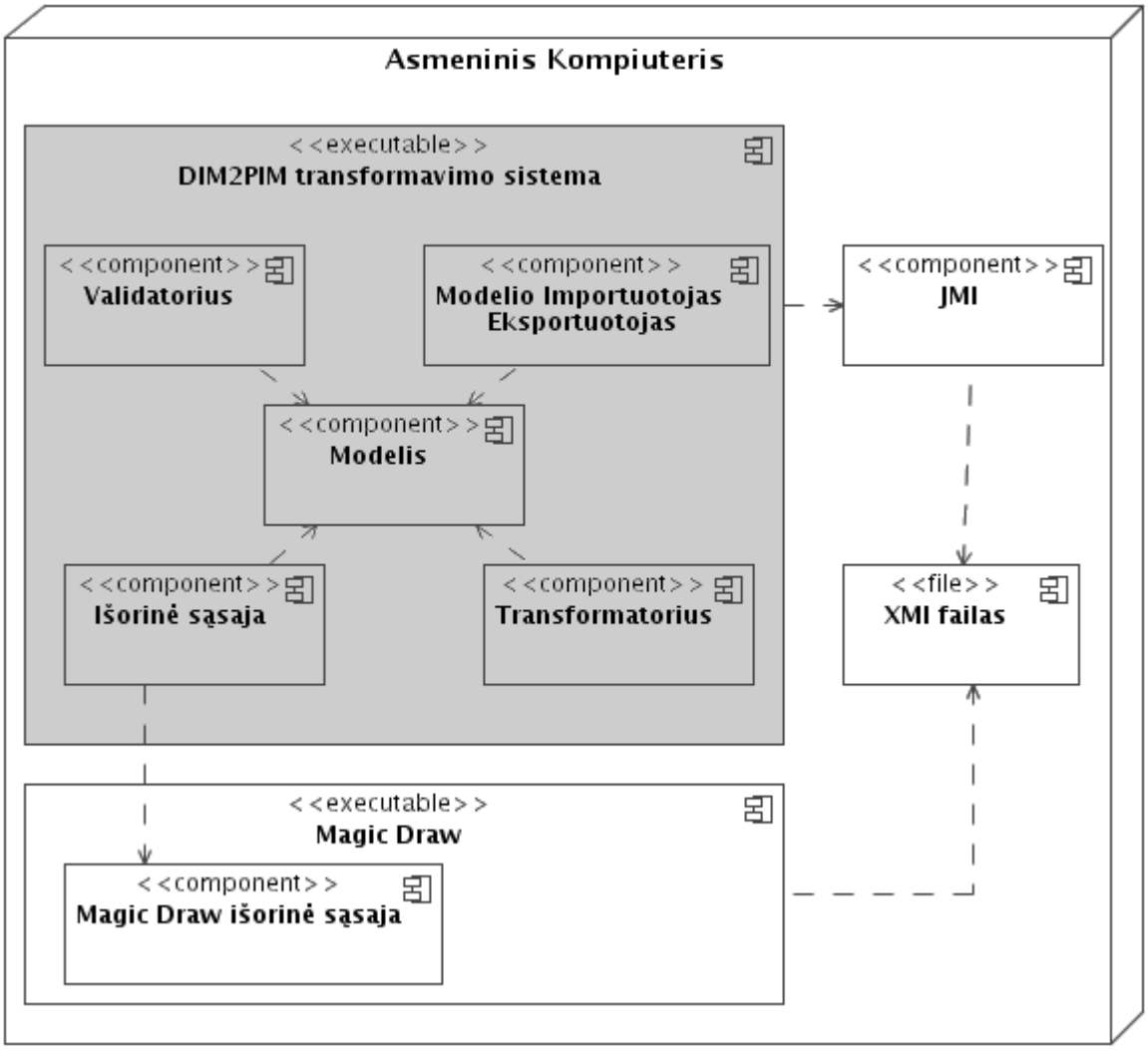
Kuriamai sistemai taikomi architektūros apribojimai:

- Veiklos logika turi būti realizuota remiantis užsakovo pateiktu duomenų struktūros šablonu.
- Kuriama sistema turi būti integruota į MagicDraw UML CASE įrankį.
- Įvedimo/išvedimo duomenims apdoroti turi būti naudojama JMI sąsaja.

Paveiksle 19 pavaizduotas sistemos išdėstymo vaizdas atitinkantis sistemos architektūros reikalavimus.

Kuriamą sistemą sudaro penki pagrindiniai komponentai:

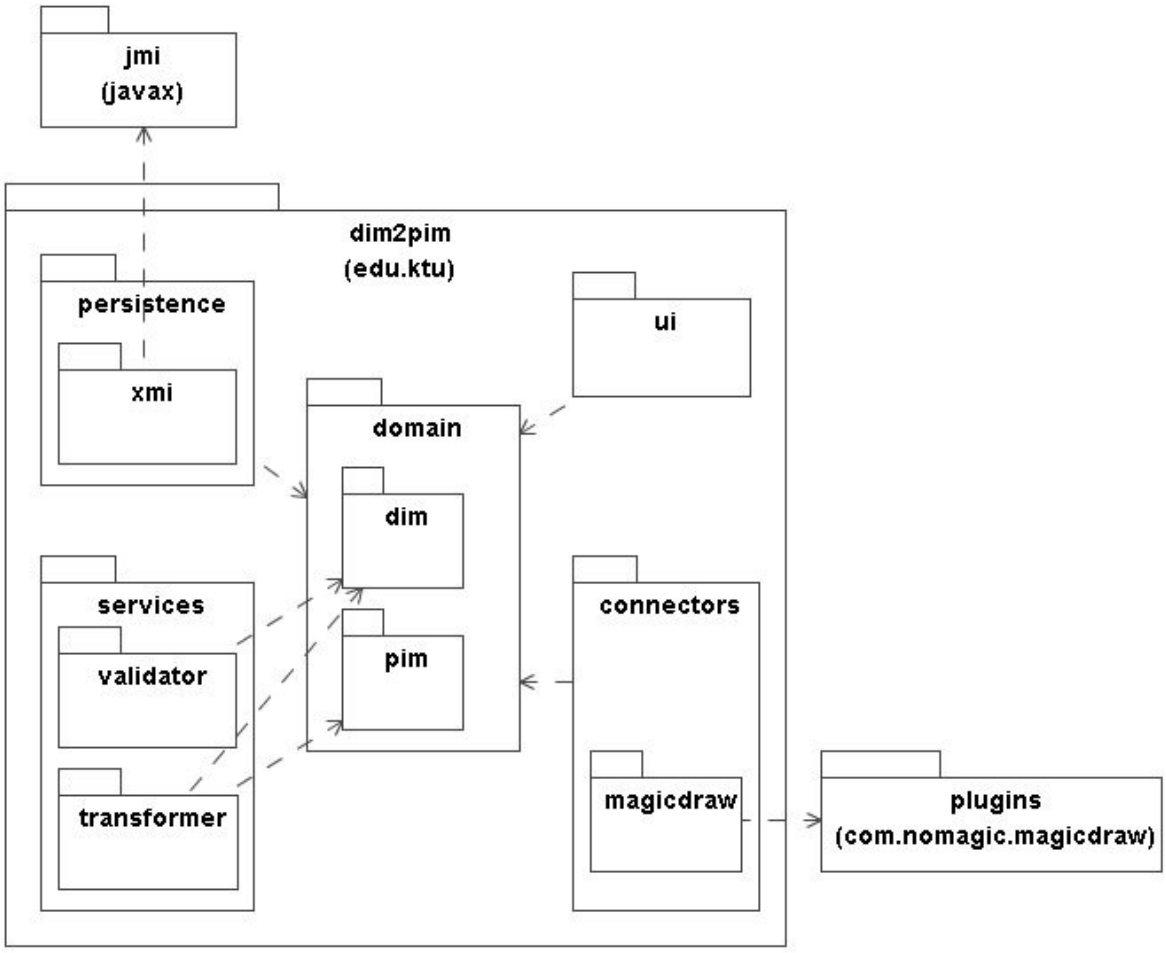
- Modelis – tai sistemoje naudojamų duomenų komponentas. Šio komponento duomenų klasės naudojamos kituose komponentuose.
- Validatorius – komponentas skirtas DIM modelio validavimui. Galimas šio komponento praplėtimas naujais validavimo algoritmais.
- Transformatorius – komponentas skirtas DIM modelio transformavimui į PIM modelį. Komponentą galima praplėsti papildomais transformavimo algoritmais.
- Išorinės sąsajos komponentas – modelio iš UML CASE įrankio importavimo ir eksportavimo sąsajų rinkinys.
- Modelio Importavimo/Eksportavimo komponentas – JMI sąsajos komponentas, skirtas XMI DIM modelio importavimui ir XMI PIM modelio eksportavimui.



19 pav. Sistemos išdėstymo vaizdas. Pilka spalva pažymėti komponentai sukurti šio darbo metu.

4.5 Paketų struktūra

Kuriamos sistemos realizacijai reikalinga paketų struktūra pavaizduota paveiksle 20. Tokia paketų struktūra reikalinga bendram sistemos ir kitų modeliavimo įrankių funkcionavimui realizuoti. Integravimo į CASE įrankius sąsajos laikomos *connectors* pakete. *Javax.jmi* ir *com.nomagic.magicdraw.plugins* yra papildomi išoriniai paketai



20 pav. Sistemos paketų struktūra

4.5.1 Paketų aprašymas

javax.jmi – JMI (*Java Metadata Interface)*, tai Sun Microsystems sukurta dinaminė, nuo platformos nepriklausoma sąsaja, kurios realizacijos skirtos valdyti meta duomenų kūrimą, saugojimą, pasiekimą ir apsikeitimą tarp įrankių.

edu.ktu.transformer.persistence.xmi – paketas skirtas XMI failų importavimui į sistemą, projekto modelio XMI failų sukūrimui.

edu.ktu.transformer.domain – paketas skirtas sistemoje naudojamų duomenų struktūroms ir tipams saugoti. Pagrindinės sistemoje naudojamos duomenų struktūros pavaizduotos paveiksle.

edu.ktu.transformer.connectors – išorinių programų bendravimo sąsajų paketas.

edu.ktu.transformer.connectors.magicdraw – realizuota sąsaja, skirta sistemos bendravimui su MagicDraw modeliavimo įrankiu paketo **com.nomagic.magicdraw.plugins** pagalba .

com.nomagic.magicdraw.plugins – MagicDraw įrankio išorinė sąsajos paketas, kuris leidžia

įskiepiams naudotis įrankio funkcionalumu.

edu.ktu.transformer.ui – vartotojo sąsajos klasių paketas. Šios sąsajos pagalba galima valdyti sistemos funkcionalumą. Šis paketas gali būti naudojamas sistemos grafinės vartotojo sąsajos kūrimo metu.

edu.ktu.transformer.models.dim – reikalavimo modelio esybių paketas.

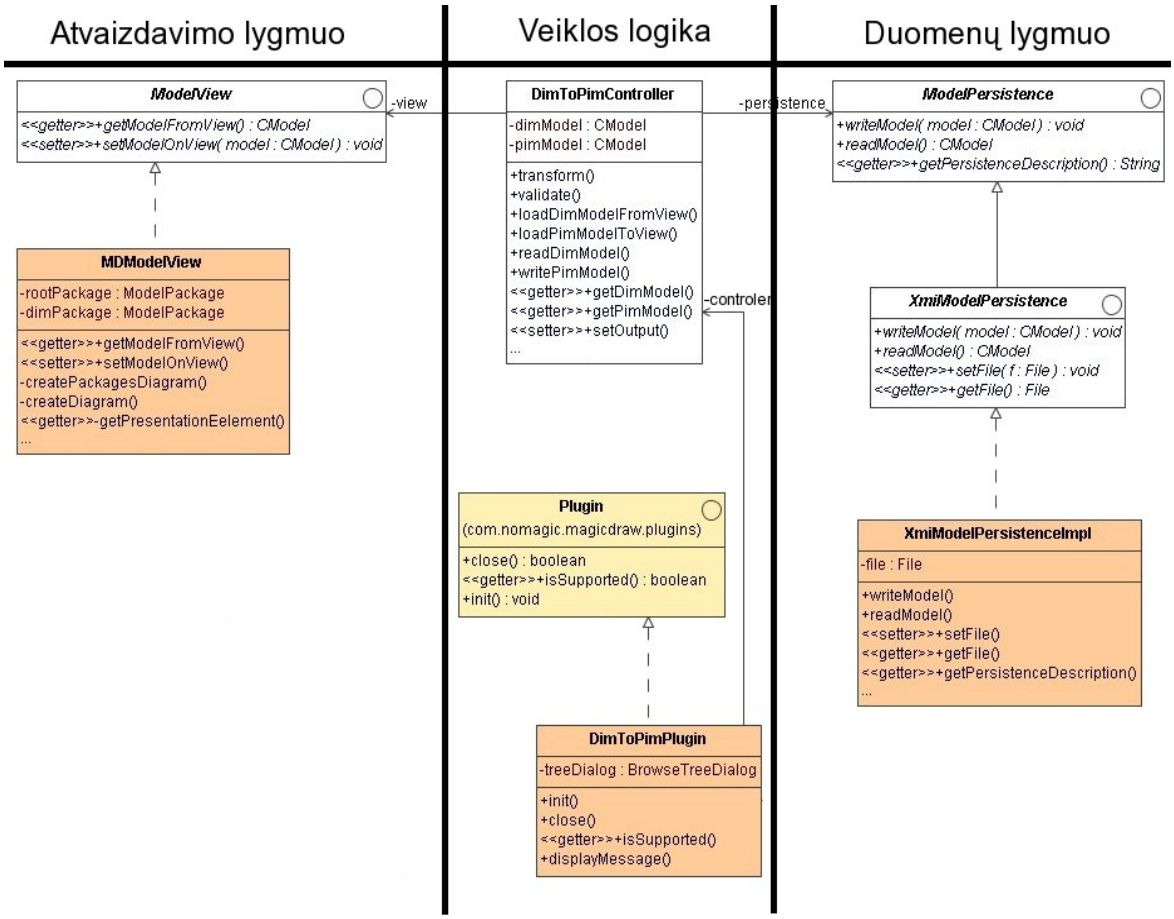
edu.ktu.transformer.models.pim – projekto modelio esybių paketas.

edu.ktu.transformer.services.validator – reikalavimo modelio tikrinimą realizuojančios klasės. Visos modelio validavimą atliekančios klasės turi realizuoti *ModelValidator* sąsają.

edu.ktu.transformer.services.transformer – reikalavimo modelio į projekto modelį transformavimo klasės.

4.6 Esminiai projektavimo sprendimai

DIM2PIM sistema suprojektuota, taikant trijų lygių architektūros modelį (21 pav.). Toks architektūros modelis leidžia lengvai integruoti sistemą su CASE įrankiais nekeičiant jos veiklos logikos, keisti duomenų saugojimo formatą, bei metodą, nuskaitymo ir įrašymo būdą.



21 pav. Sistemos trijų lygių architektūros realizavimas

Veiklos logikos komponentai bendravimui su atvaizdavimo ir duomenų lygių komponentais naudoja sąsajas – *ModelView* ir *ModelPersistence*.

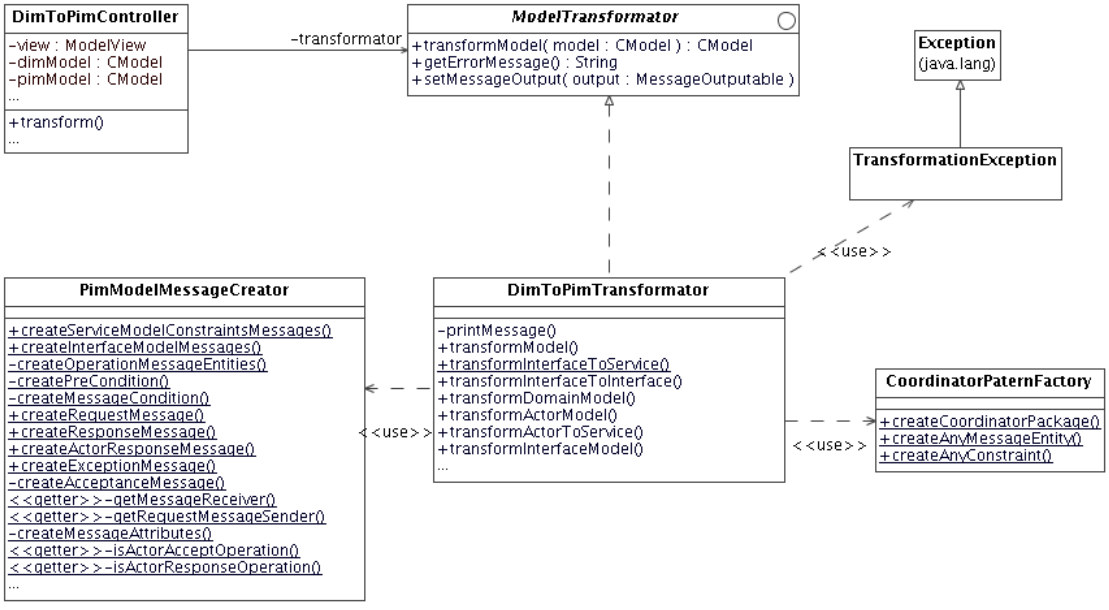
Sąsaja *ModelView* skirta transformuojamam modeliui gauti ir atvaizduoti grafiškai UML CASE įrankiuose. Šią sąsają realizuojanti klasė *MDModelView* atlieka DIM modelio nuskaitymą ir PIM modelio atvaizdavimą MagicDraw įrankio aplinkoje. Integruojant sistemą su kitu CASE įrankiu turi būti sukurta tam įrankiui skirta *ModelView* sąsajos realizacija.

ModelPersistence – sąsaja skirta modelio nuskaitymui ir saugojimui. Skirtingos šios sąsajos realizacijos gali atlikti modelio saugojimą į failus ar duomenų bazę. Modelio saugojimui į XMI failus sukurta *ModelPersistence* praplečianti *XMIModelPersistence* sąsaja. Ši bendra XMI saugojimo sąsaja leidžia kurti realizacijas dirbančias su skirtingomis XMI versijomis. Dabartinė *XMIModelPersistence* sąsajos realizacija *XMIModelPersistenceImpl* modelio nuskaitymui ir saugojimui naudoja XMI 1.2 versijos failų specifikaciją.

DIM2PIM įrankio funkcionalumo valdymas vykdomas *DimToPimController* klasės pagalba. Ši klasė naudojama modelio nuskaitymo, tikrinimo, transformavimo ir saugojimo funkcijoms iškviesti iš *DimToPimPlugin* klasės. *DimToPimPlugin* realizuoja MagicDraw įrankio *Plugin* sąsają, todėl šią klasę galima integruoti į įrankio aplinką.

4.6.1 Modelių transformavimo realizavimas

Modelio transformavimo funkcija iškviečiama iš *DimToPimController* klasės (22 pav.). Ši klasė naudoja *ModelTransformator* sąsają transformavimą realizuojančioms klasėms valdyti. Bendros transformacijos klasių sąsajos naudojimas leidžia papildyti sistemą naujomis transformavimo algoritmo realizacijomis.



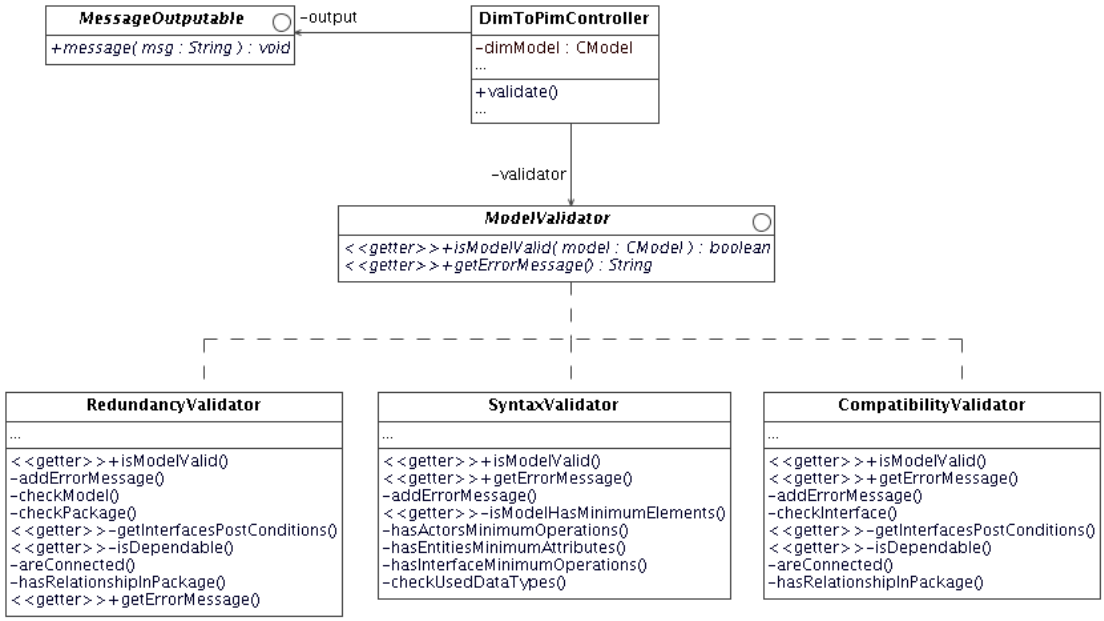
22 pav. DIM modelio į PIM transformavimo klasės

ModelTransformator sąsają realizuojanti klasė *DimToPimTransformator* atlieka

modelio transformavimą iš DIM į PIM. StateCoordinator šablonui sukurti naudojama *CoordinatorPatternFactory* klasė. Klasė *PimModelMessageCreator* atlieką StateCoordinator PIM modelio servisų bendravimo pranešimų generavimą. Įvykus klaidai transformavimo metu, generuojamas *TransformationException* išimties klasės egzempliorius saugantis informaciją apie klaidą: jos atsiradimo priežastį ir vietą.

4.6.2 Modelio tikrinimo realizavimas

DIM modelio tikrinimo funkcijos iškvietimas atliekamas *DimToPimController* klasės pagalba. Visos sistemoje naudojamos tikrinimo klasės realizuoja *ModelValidator* sąsają. *DimToPimController* naudoja šią sąsają tikrinimui atlikti ir jo rezultatams gauti. Šiuo metu yra sukurtos *RedundancyValidator (pertekliškumo)*, *SyntaxValidator (sintaksės)* ir *CompatibilityValidator (susietumo)* tikrinimo klasės (23 pav.). Tikrinimo algoritmui praplėsti reikia sukurti *ModelValidator* sąsajos norimą tikrinimo realizaciją. *MessageOutputable* sąsajos realizacijos naudojamos vartotojo pranešimams atvaizduoti.

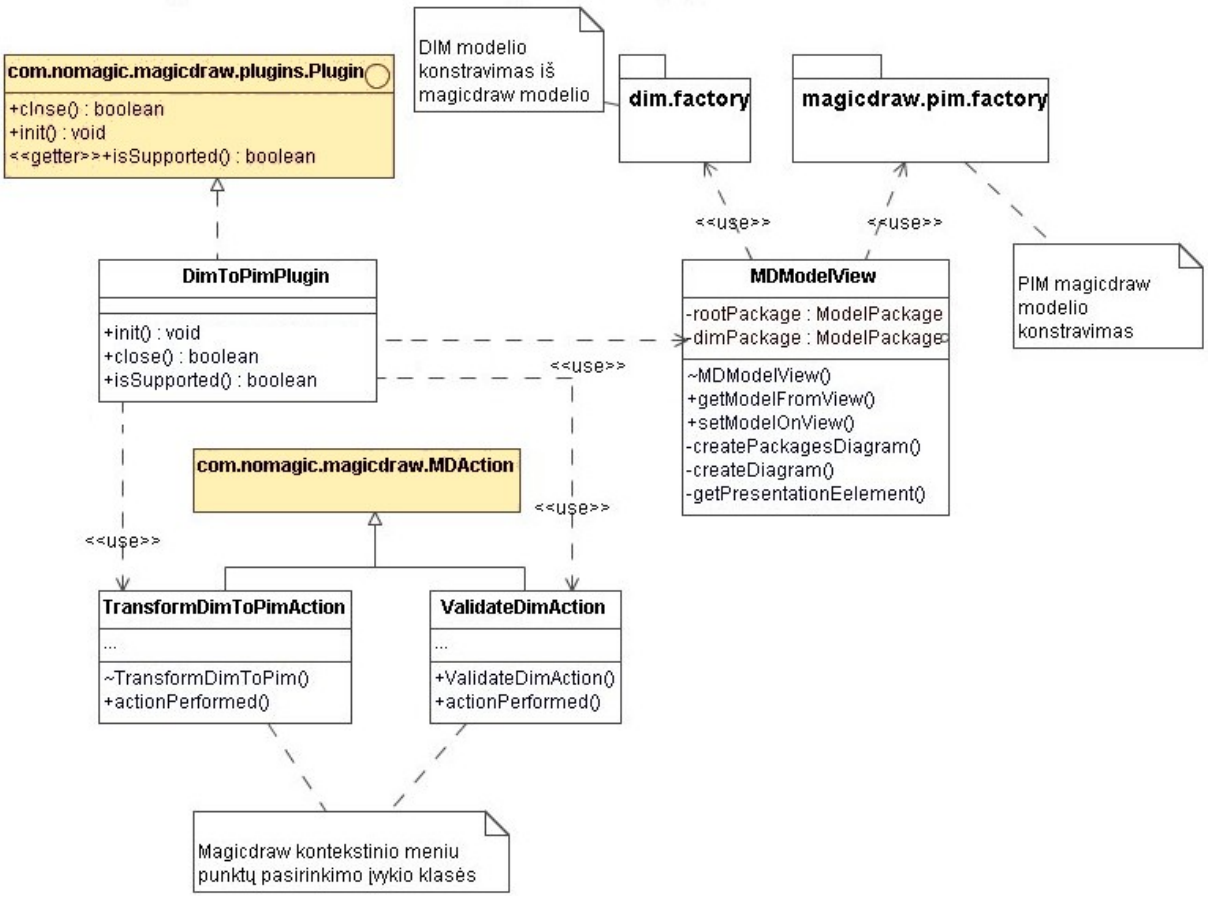


23 pav. DIM modelio tikrinimo sąsajos realizavimas

4.6.3 Sukurtos sistemos integravimas į MagicDraw

MagicDraw įrankio funkcionalumas išplečiamas įskiepių pagalba. Visi MagicDraw įskiepiai realizuoja *com.nomagic.magicdraw.plugins* pakete esančią bendrą sąsają *Plugin* (24 pav.). Ši sąsaja aprašo *init()* ir *close()* metodus, kurie yra iškviečiami MagicDraw įrankio darbo pradžios ir pabaigos metu. Šiuose metoduose aprašomi įskiepio inicializavimui reikalingi veiksmai ir panaudotų resursų atlaisvinimo operacijos, kurios iškviečiamos įskiepio darbo pabaigoje.

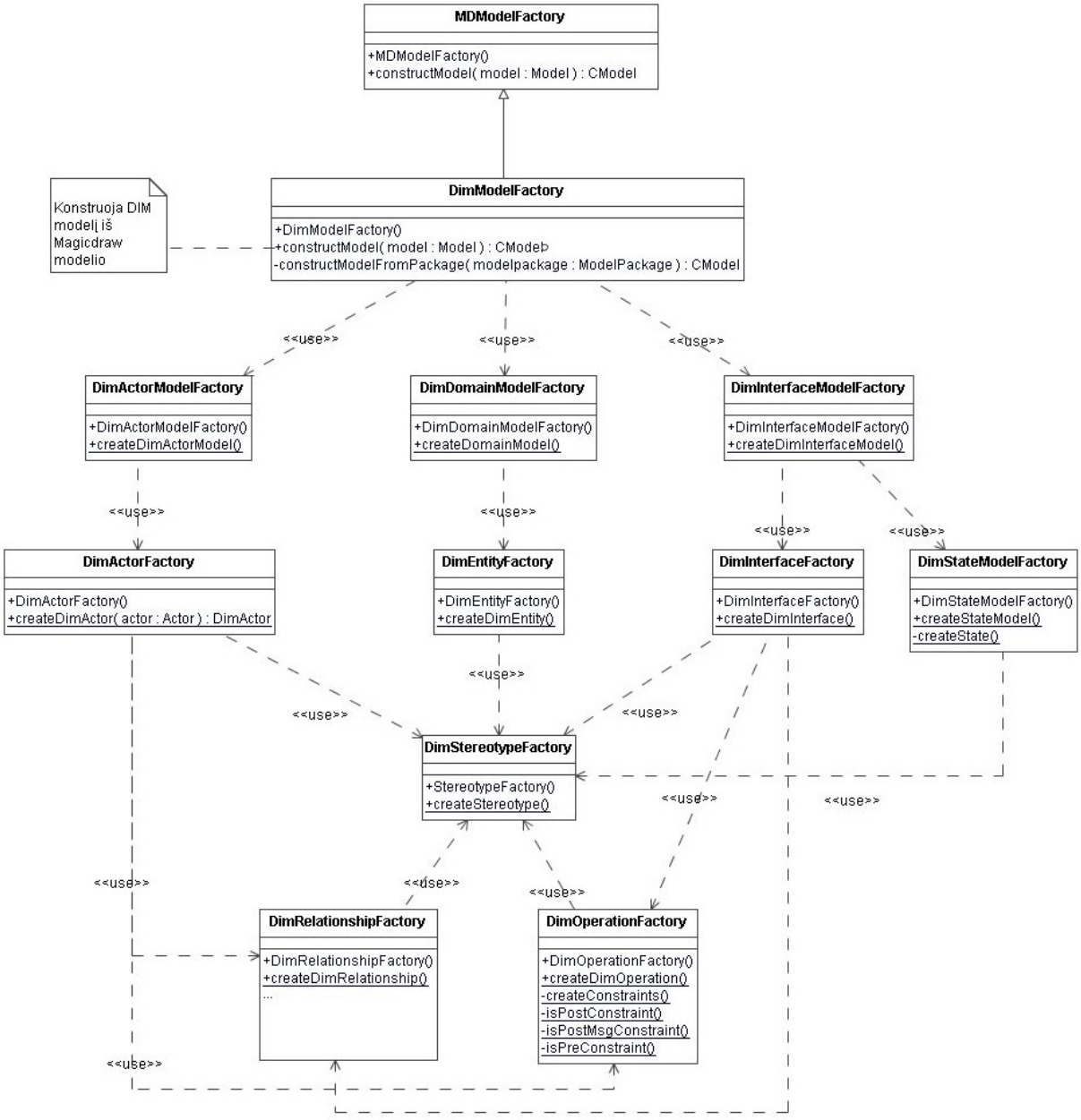
Sukurtai sistemai integruoti į MagicDraw reikalingas bendros *com.nomagic.magicdraw.plugins* sąsajos realizavimas .



24 pav. DIM2PIM integravimo į MagicDraw sąsajos realizavimas (pilkai pažymėtos MagicDraw sąsajos)

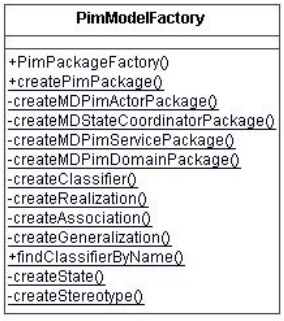
Įskiepio funkcionalumas valdomas MagicDraw grafine aplinka. Tam tikslui į kontekstinį meniu buvo įterpti validavimo ir transformavimo funkcijų iškvietimo meniu punktai. *MDAction* sąsaja aprašo *actionPerformed* metodą, kuris yra iššaukiamas kiekvieną kartą vartotojui pasirinkus atitinkamą meniu punktą (24 pav.).

MagicDraw aplinkoje sukurto DIM modelio konvertavimui į sistemos DIM modelį naudojamas dim.factory paketo klasės. Šios klasės konvertuoja MagicDraw įrankio duomenų modelio elementus į sistemoje naudojamus modelio elementus. Konvertavimo metu vyksta rekursinis visų duoto DIM modelio paketų perėjimas. Kiekvienam paketo modelio elementui sukonstruoti iškviečiamas atitinkamos konstravimo klasės *create()* metodas. DIM modelio konstravimo klasių hierarchija pavaizduota paveiksle 25. DIM Modelio konstravimas vykdomas iššaukiant *constructModel* metodą. Metodo parametras *model:Model* - MagicDraw aplinkoje sukurtas modelis.



25 pav. DIM modelio konstravimo klasių hierarchija

DIM2PIM sistemos PIM modelio atvaizdavimas į MagicDraw aplinkos duomenų modelį yra nesudėtingas, todėl jis realizuotas vienoje konstravimo klasėje (pav. 26).



26 pav. MagicDraw PIM modelio konstravimo klasė

4.7 Sukurtos programų sistemos charakteristikos

Sukurta programų sistema pasižymi praplėčiamumu, pakartotiniu panaudojimu, integralumo savybėmis. Sistema gali būti praplėsta naujais modelio tikrinimo ir transformavimo algoritmais. Sistemą galima pritaikyti darbui su skirtingais duomenų saugojimo, nuskaitymo formatais ir metodais. Komponentų suskirstymas į lygius ir jų naudojimo sąsajų apibrėžimas sudaro sąlygas sistemos komponentų ar atskirų jos dalių pakartotiniam panaudojimui kitose sistemose. Sistemos modelio atvaizdavimo sąsajos realizacijų pagalba galima reikalavimo modelio sudarymui ir projekto modelio vaizdavimui naudoti skirtingus UML CASE įrankius.

Trečioje lentelėje pateikti apskaičiuoti vidutiniai sistemos parametrų įverčiai. Šie įverčiai yra artimi standartiniams, teisingai suprojektuotų ir realizuotų sistemų įverčiams, kurių stengiasi laikytis profesionalūs programinės įrangos kūrėjai.

Lentelė 3: Sukurtos sistemos metrikos

Parametro pavadinimas	Dydis	Įvertis	St. Įvertis
Statinių metodų kiekis	64	1.89	1.77
Klasių kiekis	77	3.72	3.89
Atributų kiekis	105	2.74	4.13
Metodų kiekis	291	3.79	4.28
Perkrautų metodų kiekis	7	0.19	0.29
Maks. Parametrų metode kiekis	5	1.03	1.04
Sąsajų kiekis	6	0.27	0.54
Paveldėjimų kiekis	36	1.38	1.52
Paketų kiekis	22		
Kodo eilučių kiekis	3955		

4.8 Sistemos ateities tobulinimo darbai

Reikalavimų modelio tikrinimo ir transformavimo algoritmai yra įprogramuoti į sistemą. Naujų algoritmų kūrimą arba esamų algoritmų modifikavimą gali atlikti tik programuotojas. Tam reikalingas sistemos perkompiliavimas ir pakartotinas įdiegimas. Šiai problemai spręsti, ateityje, turi būti sukurtas komponentas galintis atlikti dinaminį papildomo programos kodo užkrovimą sistemos veikimo metu.

Dabartinė sistema yra pritaikyta dirbti su konkrečios, nustatytos struktūros reikalavimų modeliu. Ateityje reikalingas reikalavimų modelio struktūros aprašymo mechanizmo projektavimas ir jo realizavimas.

5. SUKURTO ĮSKIEPIO NAUDINGUMO TYRIMAS

MDA sistemų kūrimo proceso metu PIM modelio sudarymą atlieka projektuotojas. Sukurtas DIM2PIM transformavimo įskiepis automatizuoja PIM modelio kūrimą. Automatinis PIM modelio generavimas turėtų sumažinti sistemos projektavimo laiką, lyginant su rankiniu PIM modelio sudarymo būdu.

Šis tyrimas atliekamas siekiant nustatyti automatinio PIM modelio generavimo pranašumus, prieš įprastą, rankinį PIM modelio kūrimo būdą. Pranašumo vertinimo kriterijai yra PIM modeliui sukurti reikalingos laiko sąnaudos.

5.1 Tyrimo dalyviai

Vardas, pavardė	Darbovietė	Pareigos	Dalyvio kodas
Lina Čeponienė	KTU Informacinių sistemų ir duomenų bazių projektavimo modelio laboratorija.	Projektuotojas	D1
Milda Balandytė		Projektuotojas	D2
doc. dr. Lina Nemuraitė	KTU Informacinių sistemų katedra	Projektuotojas	D3
dr. Darius Šilingas	UAB „Baltijos programinė įranga“	Mokymų skyriaus vadovas.	D4

Visi tyrimo dalyviai turi pakankamus darbo su MagicDraw įrankių įgūdžius ir reikiamas UML modeliavimo žinias.

5.2 Tyrimui naudojama medžiaga ir priemonės

PIM modelio rankinio projektavimo laiko sąnaudoms įvertinti buvo panaudoti du, skirtingo sudėtingumo PIM modelio sudarymo aprašai.

DIM modelio sudarymui, reikalingo laiko tyrimams atlikti, panaudoti trys DIM modelio sudarymo aprašai. Pagal šiuos aprašus sukurti DIM modeliai transformuojami į PIM modelius, atitinkančius, rankinio projektavimo tyrimo metu kuriamus PIM.

Modelių projektavimui naudojamas 9.5 versijos MagicDraw UML CASE įrankis. DIM modelio transformacijoms į PIM modelį atlikti naudojamas sukurtas DIM2PIM įskiepis.

5.3 PIM modelio kūrimo laiko sąnaudų tyrimas

Šio tyrimo metu, siekiama įvertinti PIM modeliui sukurti reikalingas laiko sąnaudas ir palyginti gautus rezultatus su PIM automatinio generavimo laiko sąnaudomis.

5.3.1 Tyrime naudojamų užduočių sudėtingumo įvertinimas

Tyrimo metu naudojami du, skirtingo sudėtingumo PIM ir trys DIM modeliai. Modelių sudėtingumas įvertinamas modelio elementų skaičių suma:

- AK – sistemos aktorių skaičius
- PS – paketų skaičius.
- DS – diagramų skaičius.
- KS – klasių skaičius.
- SS – sąsajų skaičius.
- Atr.S – atributų skaičius.
- OS – operacijų skaičius.
- Par.S – parametrų skaičius.
- As.S – asociacijų skaičius.
- GS – generalizacijos ryšių skaičius.
- RS – realizacijos ryšių skaičius.

Lentelėse 4, 5, 6 pateikti tyrime naudojamų PIM modelių elementų skaičiai.

Lentelė 4: PIM1 modelio elementų skaičius

PIM1											
Modelis	AS	PS	KS	DS	SS	Atr.S	OS	Par.S	As.S	GS	RS
Actor	0	1	1	1	1	0	1	1	0	0	1
Domain	0	2	2	1	0	4	0	0	1	0	0
Service	0	10	14	23	1	18	2	3	20	4	2
Modelio elementų skaičius: 113											

Lentelė 5: PIM2 modelio elementų skaičius

PIM2											
Modelis	AS	PS	KS	DS	SS	Atr.S	OS	Par.S	As.S	GS	RS
Actor	0	1	3	1	1	0	5	6	0	0	3
Domain	0	2	4	1	0	7	0	0	3	3	0
Service	0	30	41	31	1	199	16	32	42	37	6
Modelio elementų skaičius: 474											

Lentelė 6: PIM3 modelio elementų skaičius

PIM3											
Modelis	AS	PS	KS	DS	SS	Atr.S	OS	Par.S	As.S	GS	RS
Actor	0	1	4	1	1	0	19	31	0	0	4
Domain	0	2	8	1	0	44	0	0	6	2	0
Service	0	40	120	41	1	615	58	90	112	113	8
Modelio elementų skaičius: 1322											

Lentelėse 7, 8, 9 pateikti tyrimo metu naudojamų DIM modelių elementų skaičiai.

Lentelė 7: DIM1 modelio elementų skaičius

DIM1											
Modelis	AS	PS	KS	DS	SS	Atr.S	OS	Par.S	As.S	GS	RS
Actor	1	1	0	1	0	0	1	1	0	0	0
Domain	0	2	2	1	0	4	0	0	1	0	0
Interface	0	1	0	1	2	0	2	2	2	0	0
Modelio elementų skaičius: 24											

Lentelė 8: DIM2 modelio elementų skaičius

DIM2											
Modelis	AS	PS	KS	DS	SS	Atr.S	OS	Par.S	As.S	GS	RS
Actor	3	1	0	1	0	0	0	0	0	2	0
Domain	0	2	4	1	0	7	0	0	3	3	0
Interface	0	3	0	1	6	0	16	30	6	0	0
Modelio elementų skaičius: 88											

Lentelė 9: DIM3 modelio elementų skaičius

DIM3											
Modelis	AS	PS	KS	DS	SS	Atr.S	OS	Par.S	As.S	GS	RS
Actor	4	1	0	1	0	0	19	34	0	0	0
Domain	0	2	8	1	0	44	0	0	6	2	0
Interface	0	2	0	1	8	0	21	45	9	0	5
Modelio elementų skaičius: 213											

5.3.2 Tyrimo eiga

Kiekvienas dalyvis, MagicDraw įrankio pagalba, naudodamasis PIM modelio aprašu, turėjo sukurti PIM modelį. Buvo skaičiuojamas PIM modelio sukūrimo laikas. Tyrimo dalyviai kūrė PIM modelius, pradedant PIM modelio aprašų, turinčių mažiausią modelių elementų skaičių. Pagal DIM modelių aprašą, kiekvienam sukurtam PIM modeliui buvo kuriamas atitinkamas DIM modelis ir fiksuojamas DIM modelio sukūrimui reikalingas laikas, atliekamas DIM į PIM transformacijos trukmės įvertinimas.

5.3.3 Tyrimo rezultatai

Tyrimo pabaigoje buvo gautas kiekvieno dalyvio sugaištas modeliui sudaryti laikas. Buvo įvertintas kiekvienos modelio dalies sudarymo laikas. PIM ir DIM Duomenų modelio sudarymo laikas tyrime nenaudojamas, nes Duomenų paketas abiejuose modeliuose yra vienodas. Modelio sudarymo laikas tai Aktorių ir Servisų paketų sudarymo laiko suma. Skirtingų modelių projektavimo laikas parodytas lentelėse 10, 11, 12, 13. Paveiksluose 27, 26, 28 pavaizduotas DIM ir PIM modelių kūrimo laiko palyginimas.

Lentelė 10: DIM1 modelio sudarymo laikas.

DIM1			
Dalyviai	Aktorių modelio sudarymo laikas (minutėmis)	Sąsajų modelio sudarymo laikas (minutėmis)	Viso sugaišta laiko
D1	2	7	9
D2	3	7	10
D3	2	7	9
D4	1.4	5	6.4

Lentelė 11: PIM1 modelio sudarymo laikas.

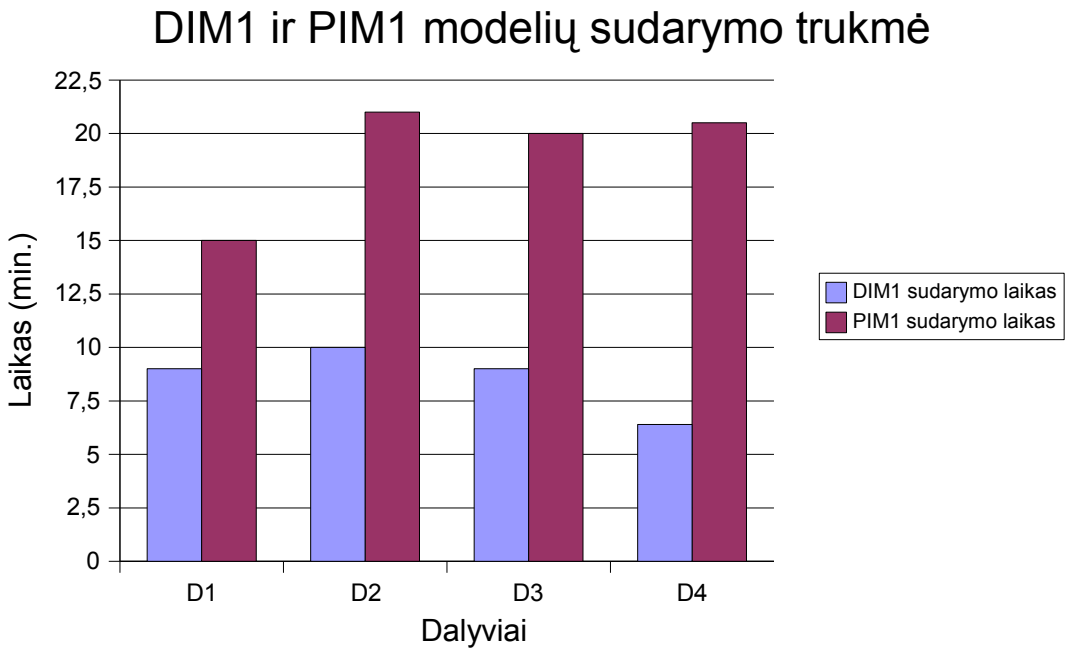
PIM1			
Dalyviai	Aktorių modelio sudarymo laikas (minutėmis)	Servisų modelio sudarymo laikas (minutėmis)	Viso sugaišta laiko
D1	3	12	15
D2	4	17	21
D3	4	16	20
D4	2.5	18	20.5

Lentelė 12: DIM2 modelio sudarymo laikas.

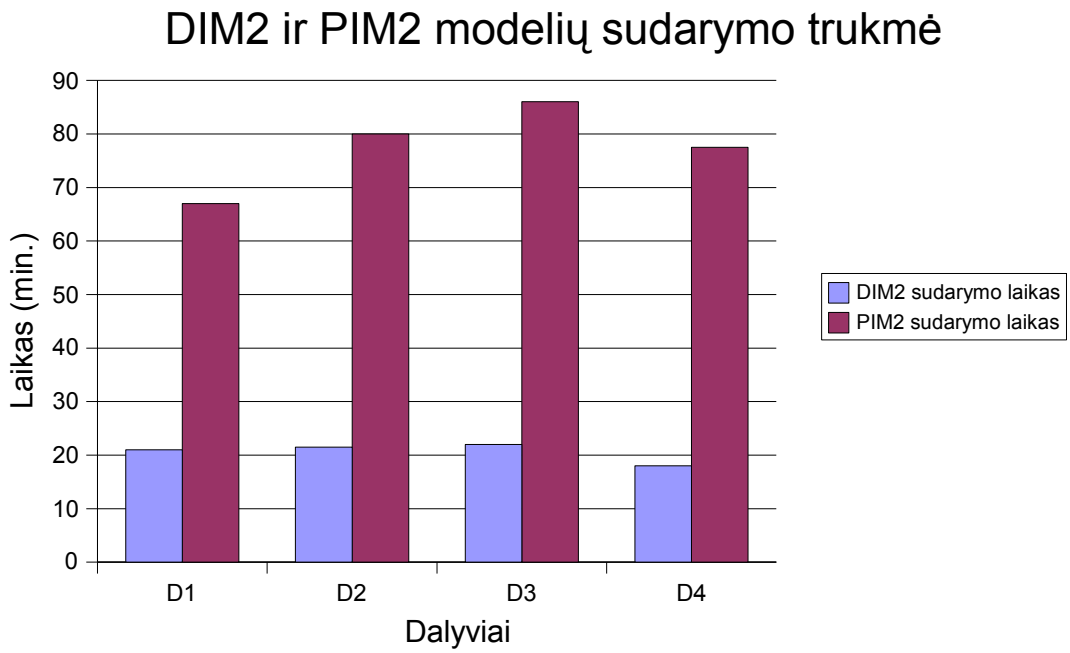
DIM2			
Dalyviai	Aktorių modelio sudarymo laikas (minutėmis)	Sąsajų modelio sudarymo laikas (minutėmis)	Viso sugaišta laiko
D1	3	18	21
D2	2.5	19	21.5
D3	3	19	22
D4	2	16	18

Lentelė 13: PIM2 modelio sudarymo laikas.

PIM2			
Dalyviai	Aktorių modelio sudarymo laikas (minutėmis)	Servisų modelio sudarymo laikas (minutėmis)	Viso sugaišta laiko
D1	4	63	67
D2	5	75	80
D3	5	81	86
D4	3.5	74	77.5



27 pav. DIM1 ir PIM1 modelių sudarymo trukmės palyginimas.



28 pav. DIM2 ir PIM2 modelių sudarymo trukmės palyginimas.

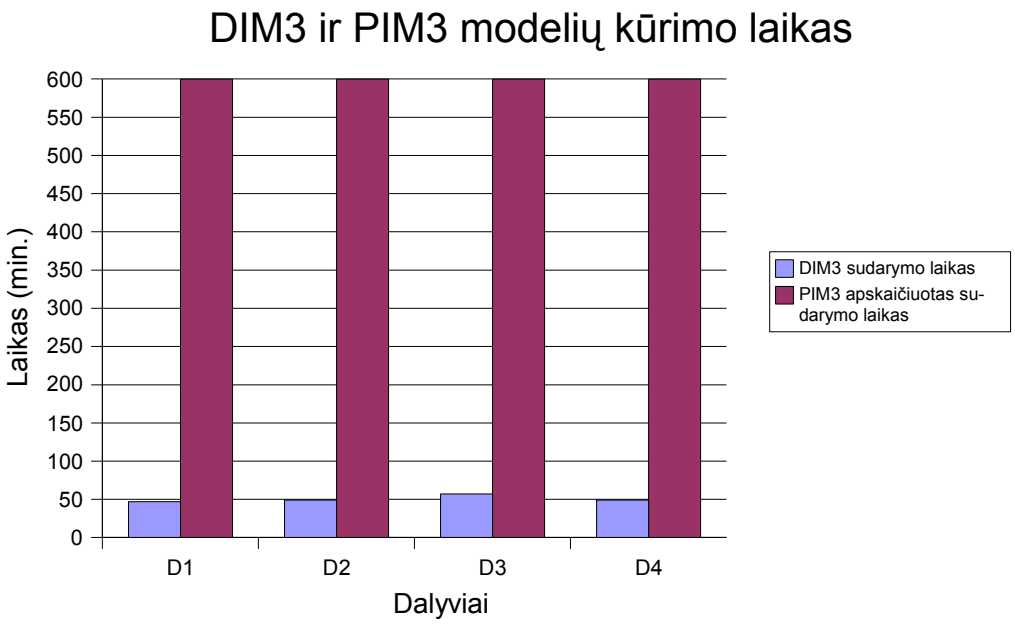
Lentelė 14: DIM3 modelio sudarymo laikas.

DIM3			
Dalyviai	Aktorių modelio sudarymo laikas (minutėmis)	Sąsajų modelio sudarymo laikas (minutėmis)	Viso sugaišta laiko
D1	18	29	47
D2	15	34	49
D3	19	38	57
D4	14	35	49

PIM3 modelis per daug sudėtingas tam, kad jis būtų sudaromas rankiniu būdu. Jo sudarymas užimtų daug laiko, todėl laikas reikalingas PIM3 modelio sudarymui apskaičiuojamas remiantis ankstesniais modelio kūrimo trukmės įvertinimais.

Tokiu būdu, vidutinis PIM Aktorių modelio elemento kūrimo laikas = (D1 PIM aktorių modelio sudarymo laikas/elementų skaičius +D2 PIM aktorių modelio sudarymo laikas/ elementų skaičius + D3 PIM aktorių modelio sudarymo laikas/ elementų skaičius +D4 PIM aktorių modelio sudarymo laikas/ elementų skaičius) / 4 = 0.36, o vidutinis PIM Servisų modelio elemento kūrimo laikas = (D1 PIM Servisų modelio sudarymo laikas/elementų skaičius + D2 PIM Servisų modelio sudarymo laikas/elementų skaičius + D3 PIM Servisų modelio sudarymo laikas/elementų skaičius + D4 PIM Servisų modelio sudarymo laikas/elementų skaičius) / 4 = 0.53.

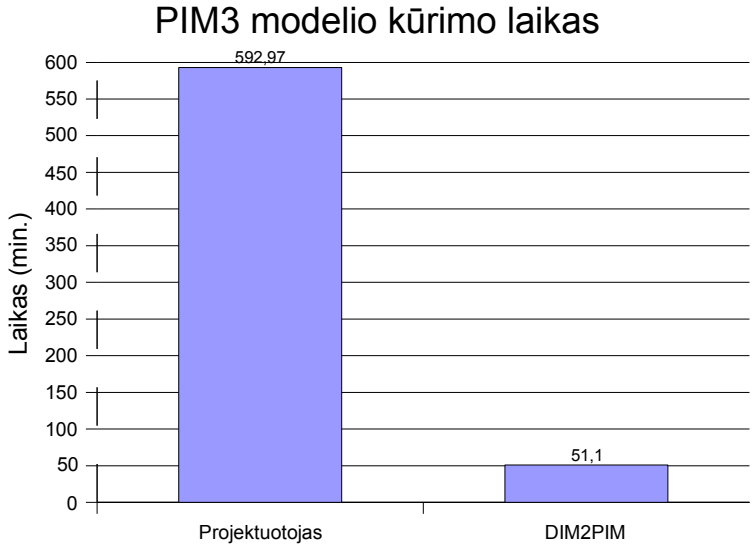
Apskaičiavę Servisų ir Aktorių modelio elementų sukūrimo vidurkį, gauname, kad vidutinis modelio elemento sukūrimo laikas : ~0,4485 min. PIM3 modelio sudarymo rankiniu būdu trukmė lygi Aktorių ir Servisų modelių elementų skaičiui padaugintam iš vidutinio modelio elemento sukūrimo laiko. T.y. 1259*0,4485 = 592,97min = 9.41 val. Paveiksle 29 pavaizduotas DIM3 ir PIM3 modelių sudarymo trukmės palyginimas.



29 pav. DIM3 ir PIM3 modelių sudarymo trukmės palyginimas.

Atliekant PIM modelio generavimą iš DIM, PIM modelio kūrimo laikas yra DIM modelio kūrimo laikas + DIM2PIM transformacijos laikas. DIM2PIM įskiepis atlieka DIM modelio transformaciją į PIM modelį per ~10 sekundžių. Tada PIM3 modelio generavimo laikas = vidutinis DIM3 modelio kūrimo laikas + 0.6 = 51.1

Skirtumas tarp PIM3 modelio kūrimo rankiniu ir generavimo būdu matomas paveiksle 30.



30 pav. PIM3 modelio sudarymo skirtingai būdais trukmės palyginimas.

5.4 Tyrimo išvados

DIM modelio sudėtingumui didėjant, sparčiai didėja PIM modelio sudėtingumas. Šį sudėtingumo didėjimą sąlygoja sugeneruoti PIM modelio servisų užklausos, atsako, išimties, priėmimo ir aktoriaus atsako pranešimai. Šių pranešimų kūrimas rankiniu būdu, reikalauja daug laiko sąnaudų, todėl ilgėja PIM modelio sudarymo laikas.

PIM modelio generavimas iš DIM modelio yra spartesnis, nei PIM modelio kūrimas rankiniu būdu. PIM3 modelio kūrimo atveju automatinis PIM modelio generavimas 11.6 karto greitesnis už įprastinį rankinį kūrimą.

6. IŠVADOS

1. Atlikta šiuolaikinių MDA krypties CASE įrankių analizė parodė, kad labai mažai įrankių atlieka taip vadinamas horizontalias transformacijas iš vienų modelių į kitus. Tokios transformacijos galėtų automatizuoti ankstesnius projektavimo proceso etapus ir taip sumažinti projektuotojo darbo sąnaudas bei užtikrinti projekto modelių korektiškumą.
2. Sukurta reikalavimų modelio (DIM) transformavimo į PIM programinė realizacija, kuri gali būti integruojama į esamus CASE įrankius. Šios programinės realizacijos sukūrimas rodo, kad galima automatizuoti informacinių sistemų projektavimą, programiškai įgyvendinant detalaus reikalavimų modelio transformavimą į projekto modelį.
3. Reikalavimų etapo modelio transformavimas į projektą atliktas tam tikru metodu, naudojant būsenų koordinatoriaus šabloną. Panašiai galima automatizuoti ir kitus projektavimo metodus, kuriuos galima aprašyti kompiuterizuojamomis taisyklėmis. Taigi projektuojančioms organizacijoms atsiranda perspektyva automatizuoti savo individualius projektavimo metodus.
4. Transformacijas atliekantis įskiepis buvo sukurtas CASE įrankiui MagicDraw, remiantis UML 2.0, MOF, XMI, JMI standartais. Transformavimo programinė įranga atskirta nuo MagicDraw sąsajos, todėl sukurtą komponentą galima integruoti į kitus CASE įrankius, palaikančius šiuos standartus.
5. Atliktas tyrimas, skirtas ištirti DIM ir PIM modelių kūrimui reikalingas laiko sąnaudas. Ištirtas sukurto DIM2PIM įskiepio taikymas PIM modelio kūrimui reikalingo laiko sąnaudų mažinimui. Tyrimo rezultatai rodo, kad DIM2PIM įskiepis sumažina projektuotojo darbo laiko sąnaudas priklausomai nuo kuriamo modelio sudėtingumo. DIM2PIM įskiepio naudojimas sudėtingiems PIM modeliams kurti gali sumažinti darbo laiko sąnaudas iki dešimties kartų.
6. Magistro tezių pagrindu buvo paruoštas straipsnis ir padarytas pranešimas konferencijoje „Informacinės technologijos 2006“.

LITERATŪRA

[1] About the Object Management Group™ (OMG™) [interaktyvus]. [žiūrėta 2006-02-09]. Prieiga per internetą: <http://www.omg.org/gettingstarted/gettingstartedindex.htm>

[2] ČEPONIENĖ, L.; NEMURAITĖ L. Design independent modeling of information systems using UML and OCL: *Databases and Information Systems'2004*; Amsterdam, 2005, p. 224-237.

[3] ČEPONIENĖ, L.; NEMURAITĖ L. Transformation from Requirements to Design for Service Oriented Information Systems: *Advances in Databases and Information Systems*, Tallinn, 2005, p. 164-177.

[4] MagicDraw feature list. [žiūrėta 2006-05-19]. Prieiga per internetą: <http://www.magicdraw.com/files/brochures/a4/MagicDrawDataSheet_A4.pdf?NMSESSID=a62fe4580f43ce78edebaabbd707d74c>.

[5] KONTIO, M. *Choosing MDA tools* [interaktyvus]. [žiūrėta 2006-03-19]. Prieiga per internetą: <http://www-128.ibm.com/developerworks/wireless/library/wi-arch18.html >

[6] MDA specifikacija [interaktyvus]. [žiūrėta 2006-05-04] Prieiga per internetą: <http://www.omg.org/cgi-bin/doc?omg/03-06-01>.

[7] Simplified Guide to the Java™ 2 Platform, Enterprise Edition [interaktyvus]. [žiūrėta 2006-05-04]. Prieiga per internetą: <http://java.sun.com/j2ee/reference/whitepapers/j2ee_guide.pdf>

[8] OptimalJ: How business rules enable rapid response to change. [žiūrėta 2006-04-29]. Prieiga per internetą: <http://www.compuware.com/dl/busrules.pdf>

[9] An evaluation of Compuware OptimalJ Professional Edition as a MDA tool. [žiūrėta 2006-04-29]. Prieiga per internetą: <http://www.compuware.com/media.asp?cid=30AAX1 >

[10] MUZAFFAR, I. Analysis of available MDA support tools. [žiūrėta 2006-05-08]. Prieiga per internetą: <https://doc.telin.nl/dscgi/ds.py/Get/File-50548/d4.1.pdf>.

[11] ArcStyler Overview. [žiūrėta 2006-04-25]. Prieiga per internetą: <http://www.iosoftware.com/as_support/brochures/ArcStyler_Overview_250703.pdf>

[12] Ecore meta-model of EMF [interaktyvus]. [žiūrėta 2006-05-01]. Prieiga per internetą: <http://download.eclipse.org/tools/emf/javadoc/org/eclipse/emf/ecore/ >

[13] Apache Ant projektas [interaktyvus]. [žiūrėta 2006-04-26] Prieiga per internetą: <http://ant.apache.org>

[14] Maven Apache projektas [interaktyvus]. [žiūrėta 2006-04-26] Prieiga per internetą: <http://maven.apache.org>

[15] AndroMDA palaikančių UML CASE įrankių sąrašas [interaktyvus]. [žiūrėta 2006-04-27] Prieiga per internetą: < http://galaxy.andromda.org/docs/case-tools.html>.

[16] Android AndroMDA Eclipse įskiepis [interaktyvus]. [žiūrėta 2006-04-27] Prieiga per internetą: <http://www.andromda.org/updatesite>.

[17] ArgoUML CASE įrankis [interaktyvus]. [žiūrėta 2006-04-27] Prieiga per internetą: <http://www.argouml.org>.

[18] Codagen Architect features [interaktyvus]. [žiūrėta 2006-04-25]. Prieiga per internetą: <http://www.codagen.com/products/architect/features.htm>

[19] ČEPONIENĖ, L.; NEMURAITĖ L. Patobulintas UML klasių, būsenų ir sekų suderinimo algoritmas: *Informacinės technologijos'2004*, p. 502-511.

[20] ČEPONIENĖ, L.; NEMURAITĖ L. UML klasių, būsenų ir sekos diagramų suderinimas*: Informacinės Technologijos'2003*. - Kaunas: Technologija, 2003, p. XIV-62 – XIV-67

[21] ČEPONIENĖ, L.; NEMURAITĖ L. Transformations of UML Diagrams for Reconciliation of Requirements: Information Systems Development. Advances in Theory, Practice, and Education. Springer, 2005, XXVIII, ISBN: 0-387-25026-3, p. 289 – 301

[22] MOF 2.0 / XMI Mapping Specification, v2.1 [interaktyvus]. [žiūrėta 2005-10-07]. Prieiga per internetą: < http://www.omg.org/technology/documents/formal/xmi.htm>

[23] Unified Modeling Language: *OCL Version 2.0*. OMG document ptc/03-08-08 (2003) [žiūrėta 2005-10-07]. Prieiga per internetą: <http://www.omg.org>

[24] GAMMA, E. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994

SUMMARY

UML CASE tool extension with model transformations

In this master thesis the advanced design methodology is presented by which information systems requirements represented with precise UML models are transformed to project.

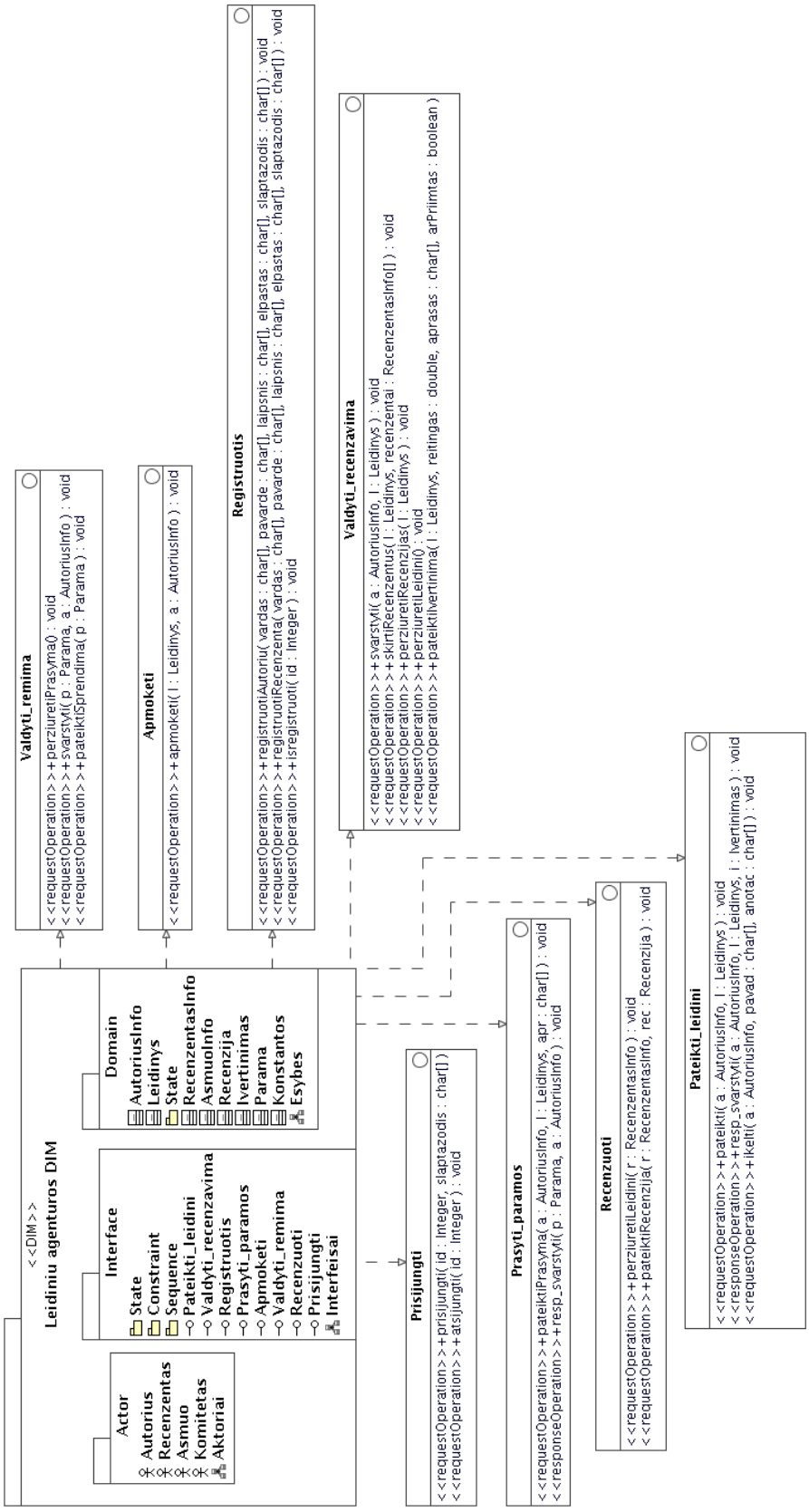
The first section describes a research of five existing MDA tools comparison according to evaluation criteria. This criteria covers model to model and model to code transformations approach, platform of generated code and integration with UML CASE tools abilities.

For requirements model transformation to design, plug-in for CASE tool MagicDraw is created according MDA standards. Model transformation plug-in takes system requirement model as input and generates design model. In this way it is possible to implement design methods that have explicit design rules. Created plug-in requirements, functional specification and architecture described in Project section.

The investigation section describes investigation of the developed plug-in. In this section were investigate the working efficiency of designer increasing and the quality of design models ensuring with create MagicDraw plug-in.

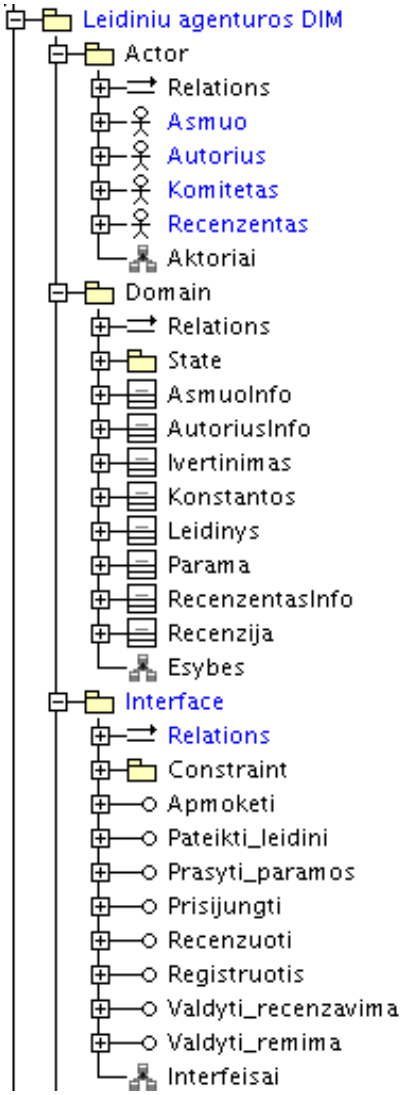
PRIEDAI

1. DIM modelio pavyzdys



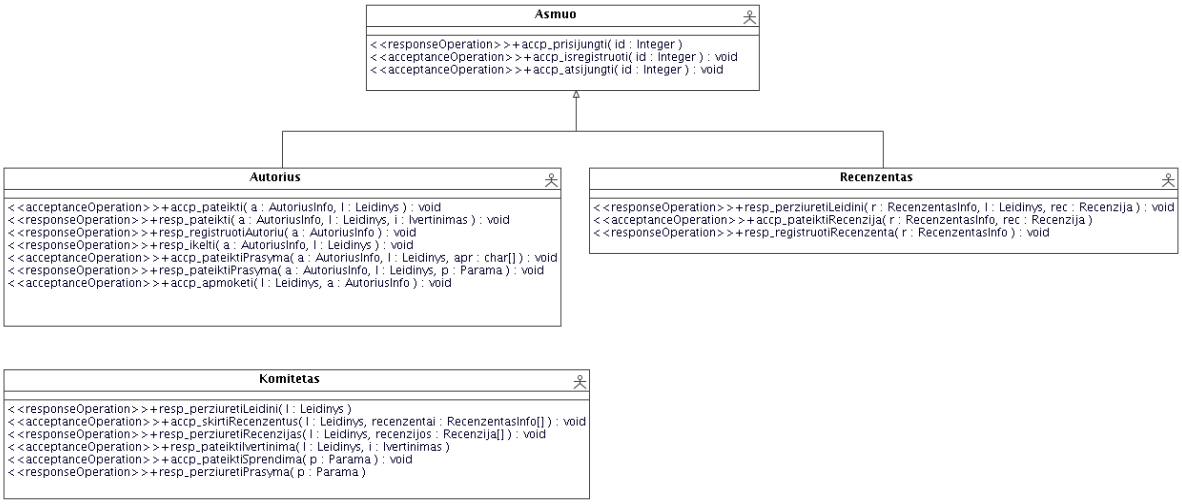
31 pav. Leidinių agentūros DIM modelis (bendras vaizdas)

2. DIM modelio struktūra

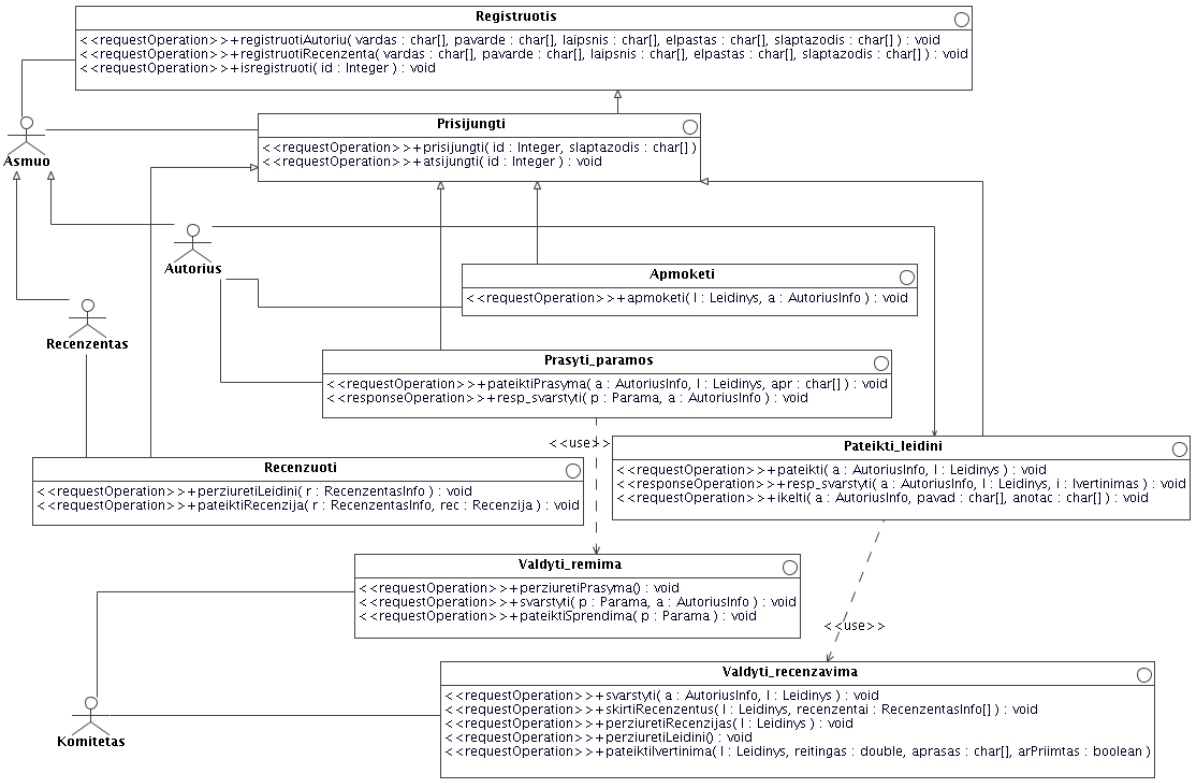


pav. 32: Leidinių agentūros DIM modelio elementai

DIM aktorių modelis

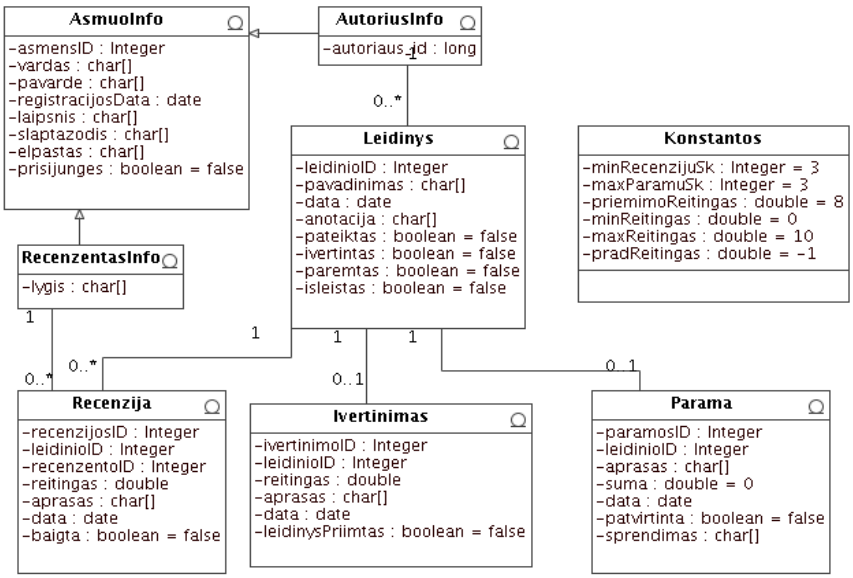


DIM sąsajų modelis



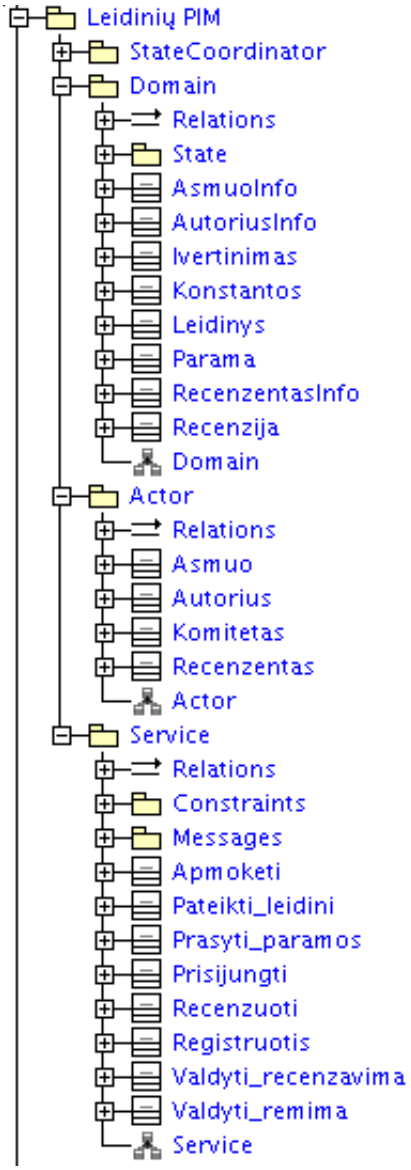
33 pav. Leidinių agentūros DIM sąsajų modelis

DIM esybių modelis



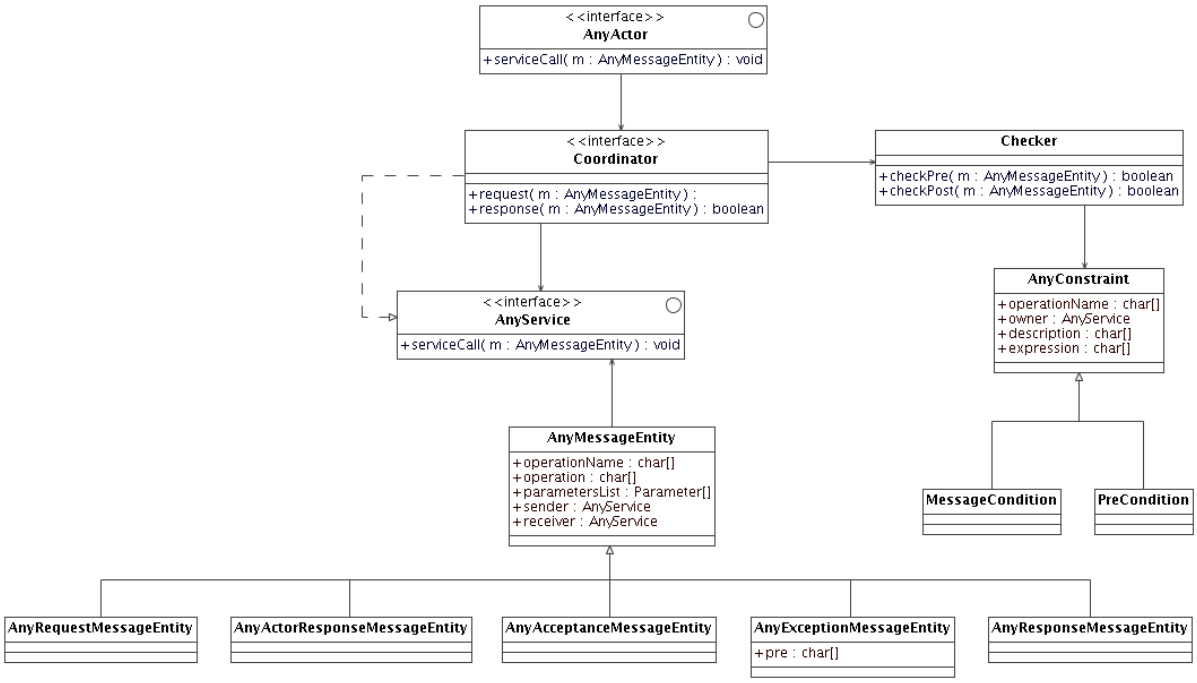
34 pav. Leidinių agentūros DIM esybių modelis

3. Sugeneruoto PIM modelio struktūra



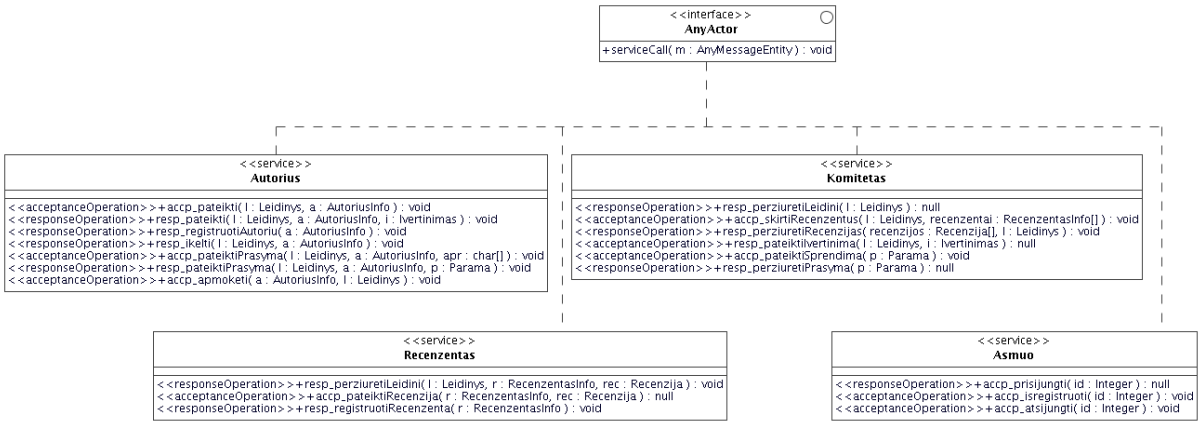
35 Pav. Leidinių agentūros sugeneruoto PIM modelio struktūra

PIM State Coordinator šablonas



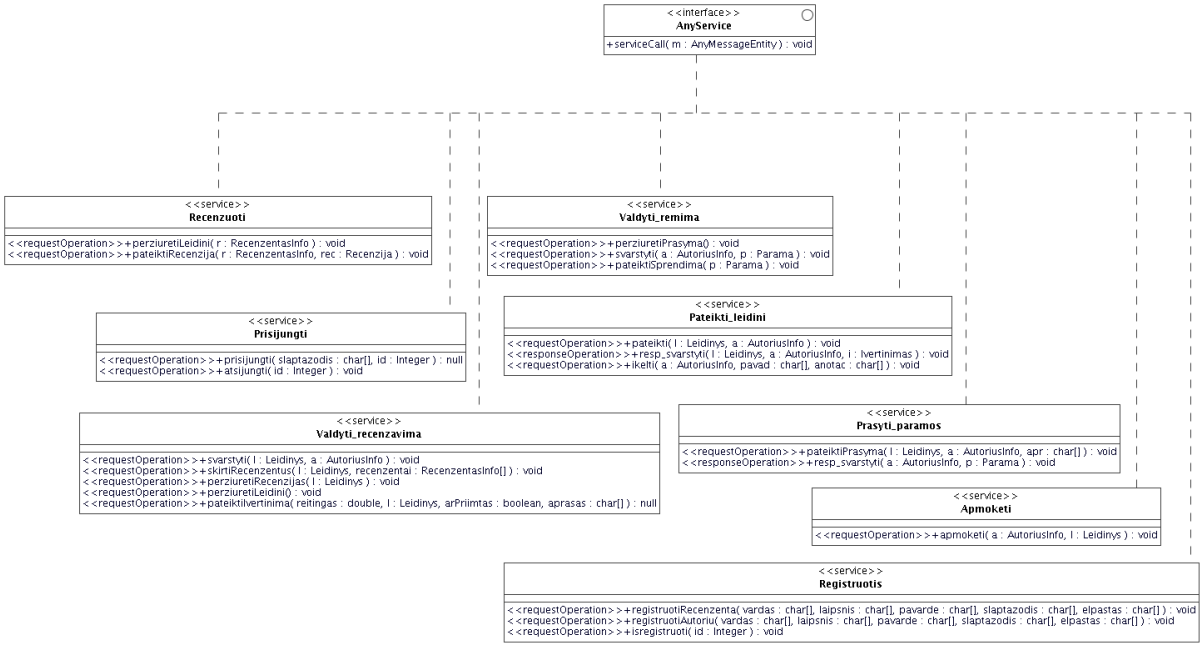
36 pav. Transformavimo metu taikomas *State Coordinator* šablonas

PIM aktorių modelis



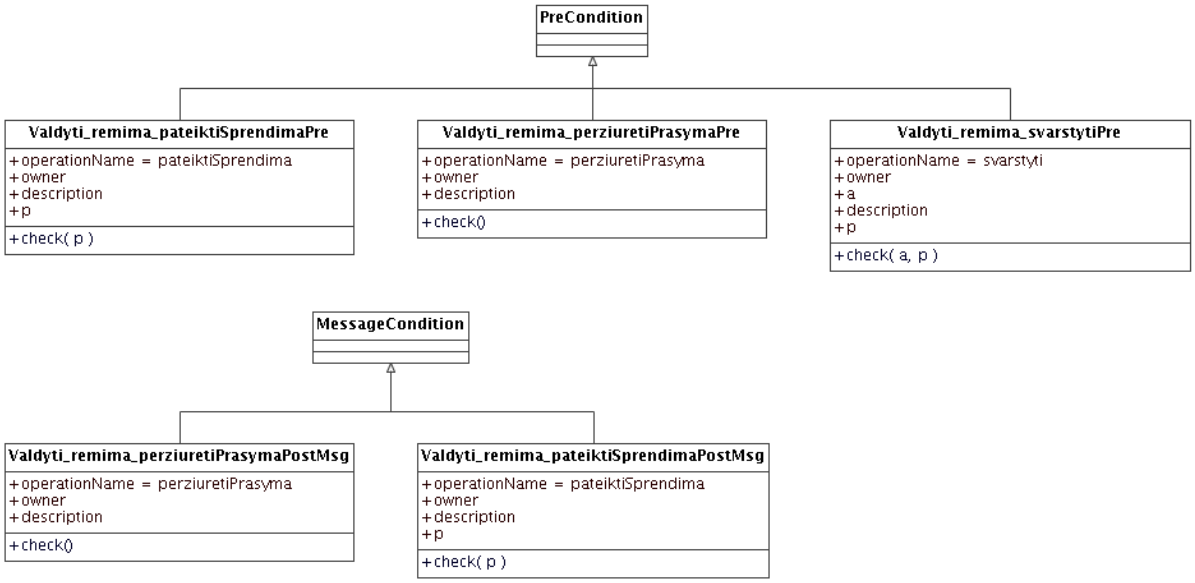
37 pav. Leidinių agentūros sugeneruotas PIM aktorių modelis

PIM paslaugų klasių (servisų) modelis

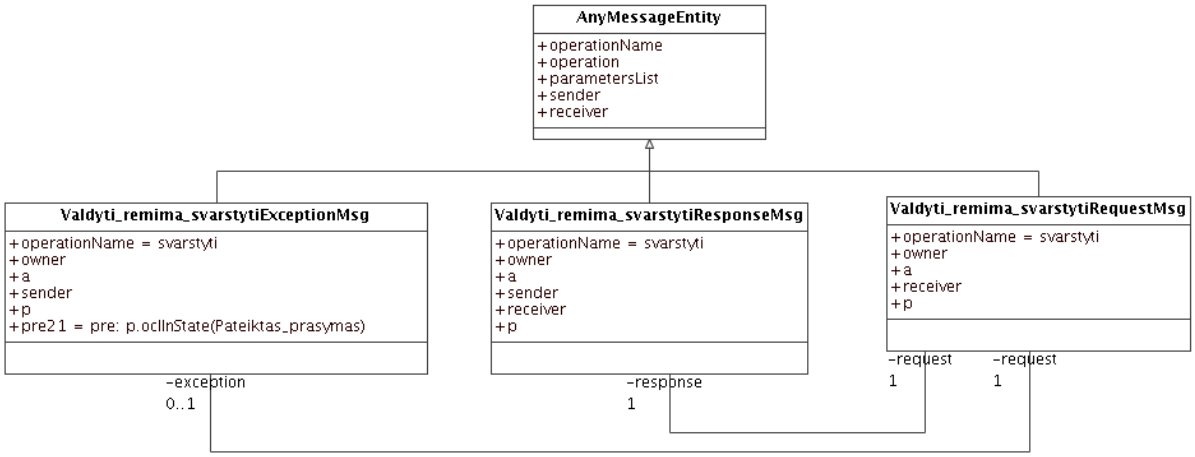


38 pav. Leidinių agentūros sugeneruotas PIM servisų modelis

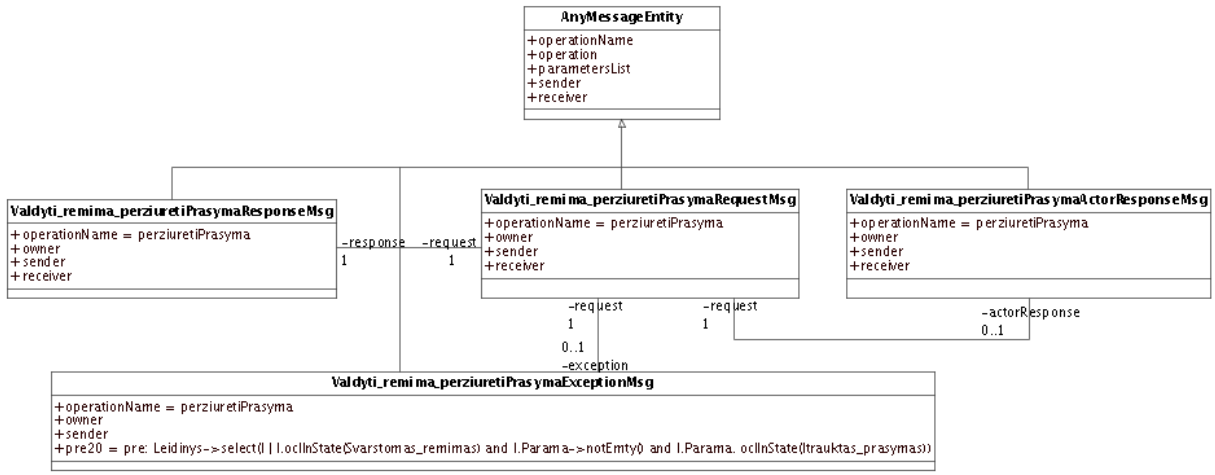
PIM servisų klasių sąlygos ir pranešimai



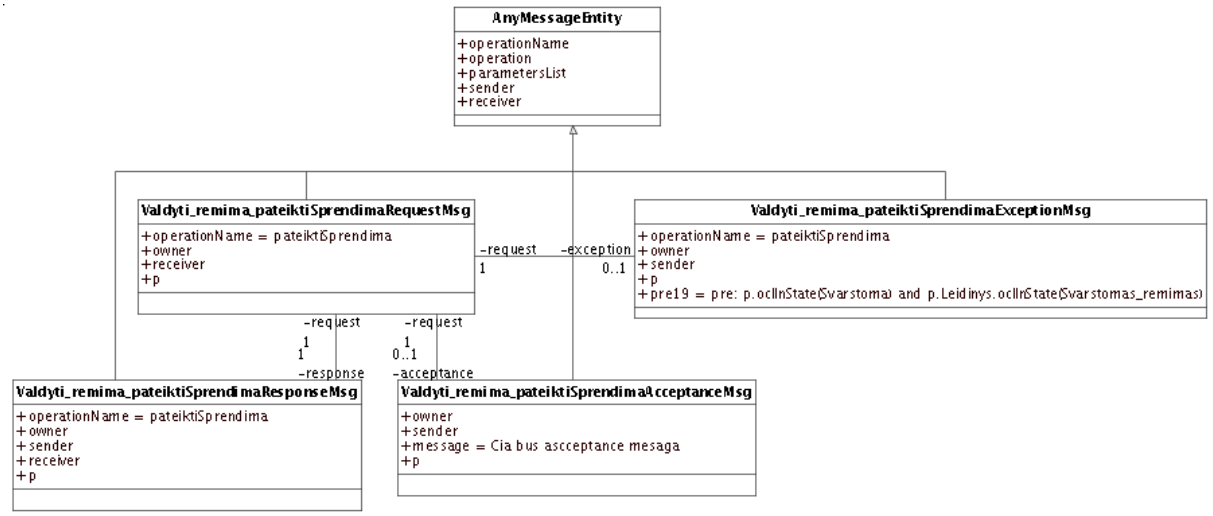
39 pav. *Valdyti_remima* serviso sugeneruotos sąlygų klasės



40 pav. Serviso *Valdyti_remima* operacijos *svarstyti* sugeneruotos pranešimų klasės

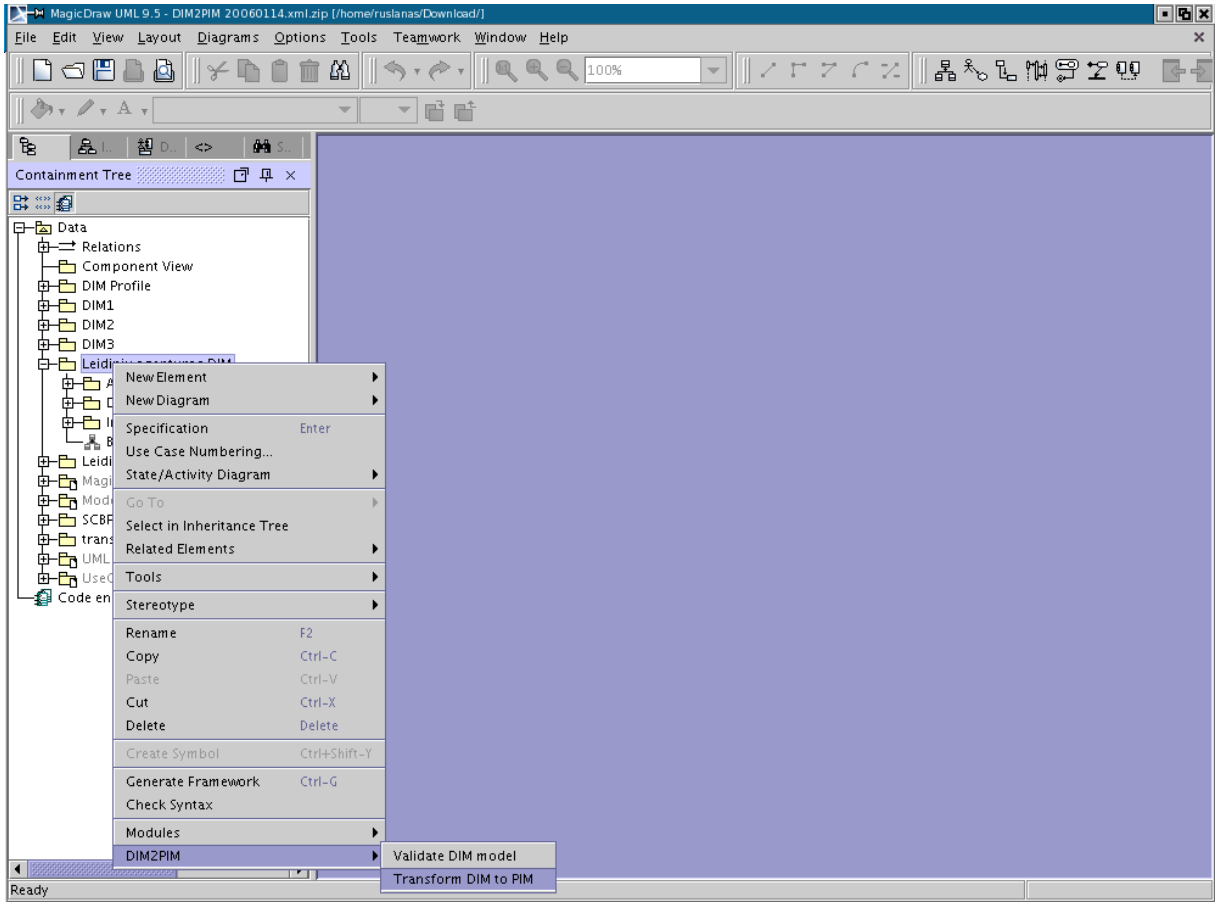


41 pav. Serviso *Valdyti_remima* operacijos *perziuretiPrasyma* sugeneruotos pranešimų klasės

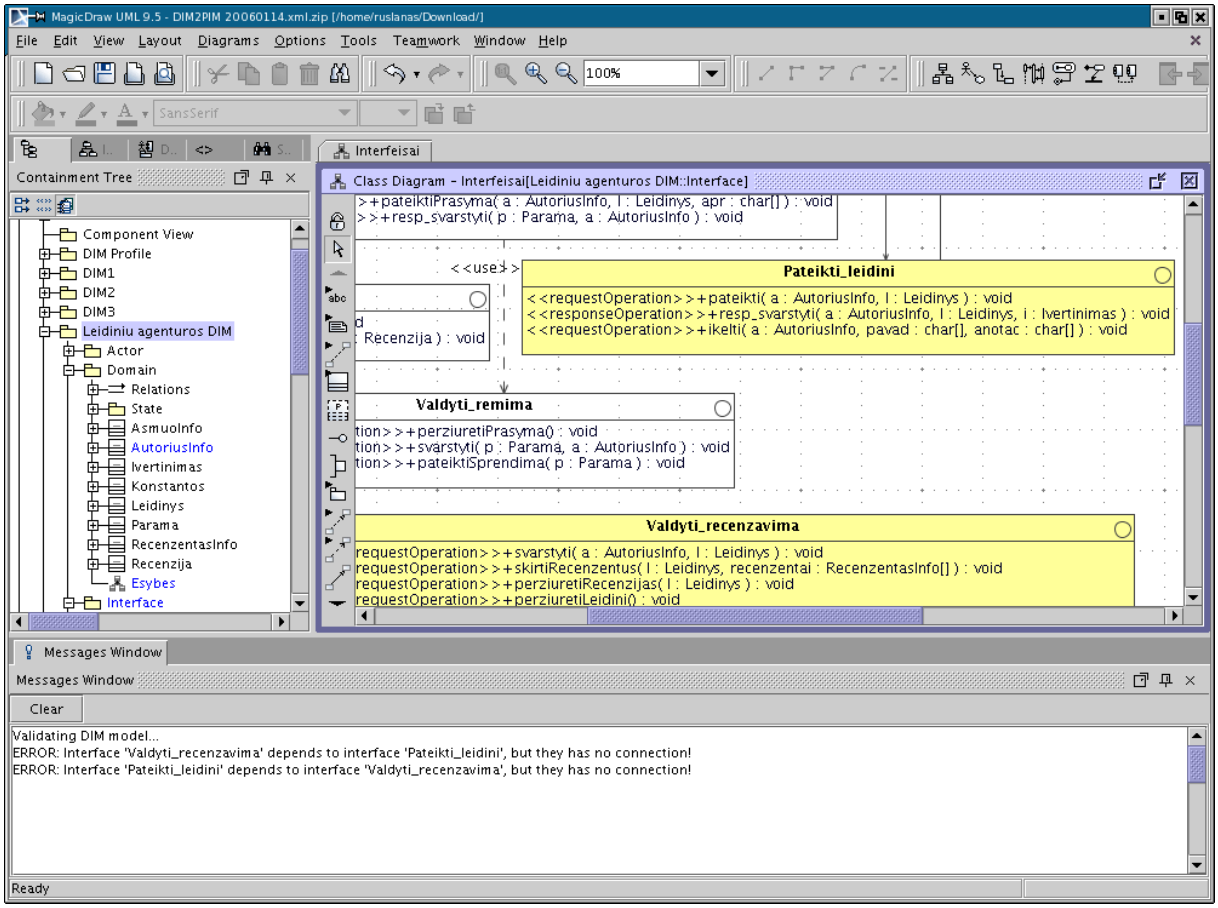


42 pav. Serviso *Valdyti_remima* operacijos *pateiktiSprendima* sugeneruotos pranešimų klasės

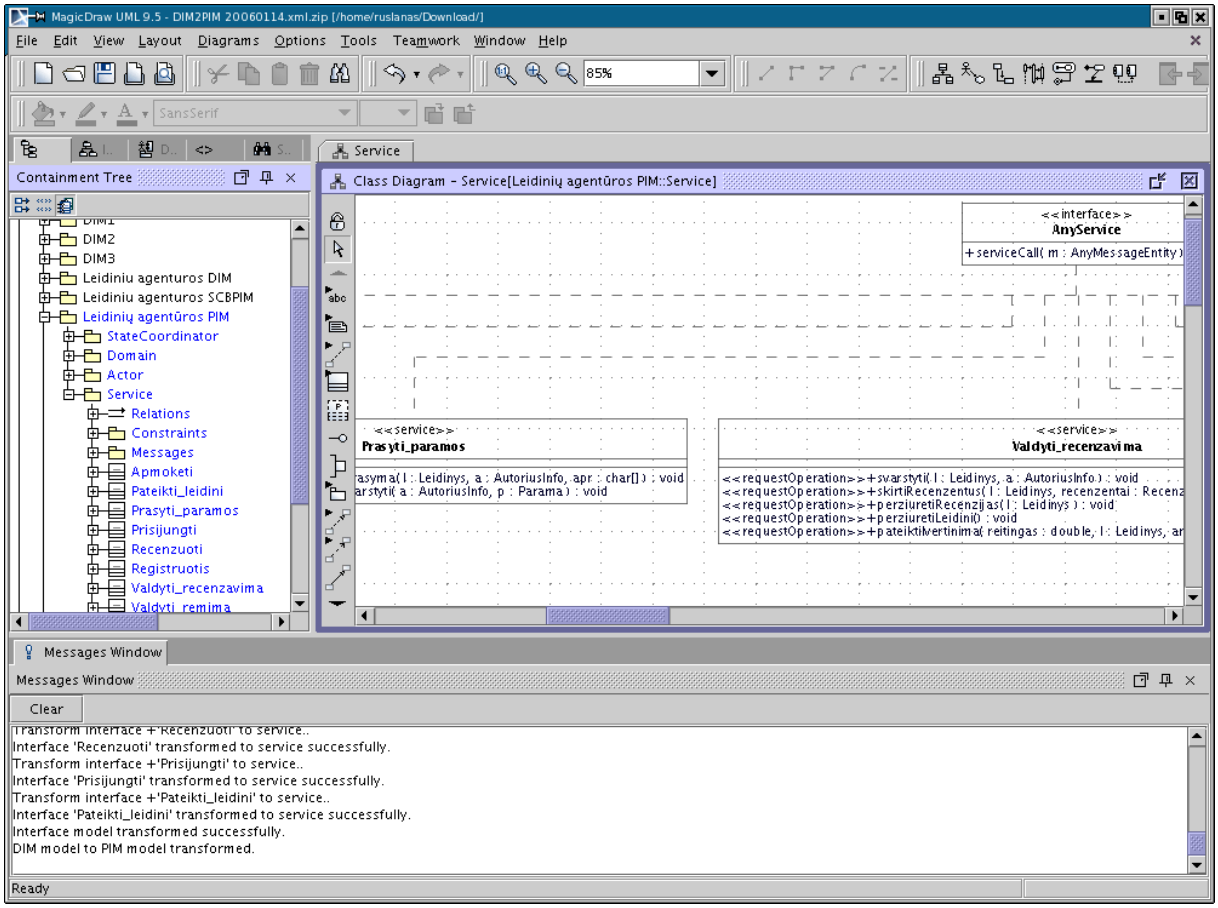
4. Sukurto DIM2PIM įskiepio naudojimas MagicDraw aplinkoje



43 pav. DIM2PIM įrankio funkcionalumo valdymas



44 pav. DIM modelyje rastos susietumo klaidos pranešimas



45 pav. Transformacijos metu gautas PIM servisų modelis