KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Ruslanas Abdrachimovas

Objektinių modelių transformacijų realizavimas

Magistro darbas

Darbo vadovas
doc. V. Pilkauskas

Kaunas, 2004

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
(parašas) prof. habil. dr. H. Pranevičius
2004 05

Objektinių modelių transformacijų realizavimas

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
(parašas) dr. Jurgita Mikelionienė
2004 05

Vadovas
(parašas) doc. dr. V. Pilkauskas
2004 05

Recenzentas
dr. doc. Rimantas Butleris
(parašas)
2004 05

Atliko
FM-8/1 gr. stud.
(parašas) R. Abdrachimovas
2004 05

Kaunas, 2004

SANTRAUKA

Magistro darbe nagrinėjamas vieno svarbiausių OMG konsorciumo modeliais pagrįstos architektūros (MDA) elementų – modelių transformacijų realizavimo problemos.

Remiantis OMG specifikacijų ir literatūros šaltiniais, autorius analizuoja transformacijos modelių procesą, nustato modeliavimo ir metamodeliavimo reikšmė į UML panašių modeliavimo kalbų kūrimui bei modelių transformacijų realizavimui.

Darbe pateikiami sukurtų eksperimentinių modeliavimo kalbų metamodeliai: „Esybė – procesas“, Java metamodelis bei reliacinis metamodelis. Supažindinama su modeliavimo kalbų grafine reprezentacija modelių redaktoriuose, sukurtuose EMF programavimo aplinkos priemonėmis.

Analizės rezultatų pagrindu, aprašoma suprojektuota lanksti modelių transformacijų realizavimo architektūra pagrįsta filtrų architektūriniu šablonu, kuri leidžia lengvai dekomponuoti transformavimo procesą į atskiras aiškiai apibrėžtas stadijas.

Pasirinktos filtrų architektūros pagrindu, realizuojamos eksperimentinės transformacijos iš „Esybė – procesas“ modelių į Java ir reliacinį modelius. Pristatomi kokybiniai ir kiekybiniai eksperimentinių transformacijų rezultatai.

SUMMARY

Presented work covers one of the most important areas of OMG’s model driven architecture (MDA) – problems of object model transformations.

Based on research of OMG specifications and other sources, author analyzes transformation process, states importance of modeling and metamodeling for designing of UML like modeling languages.

Research work describes designed metamodels of experimental modeling languages: “Entity – process”, Java metamodel and relational metamodel. Author gives a short overview of model editors for these languages, created using EMF framework tools.

Based on analysis, author describes very flexible architecture of model transformation implementation, based on filter and pipes architectural pattern. Usage of this architecture gives flexibility to transformation implementation and allows easy and straightforward decomposition of transformation to separate stages.

Designed filter and pipes transformation architecture was used for experimental transformation implementation. Research work presents quality and quantity based results of experimental transformations.

TURINYS

LENTELĖS

PAVEIKSLAI

1 ĮVADAS

Paskutinio dešimtmečio informacinių technologijų (IT) vystimasis pasižymėjo didžiuliais pasikeitimais. Mūro dėsnis (*Moore's Law*) vis dar tinka aparatinės įrangos pramonei ir nuolat pasireiškė eksponentiniu procesorių, atmintinių ir duomenų saugyklų našumo didėjimu. Taip pat ir programinės įrangos vystimasis pasižymi didžiule kaita. Buvo pastebėta, kad skirtingi programinės įrangos abstraktumo lygiai pasižymi nevienoda kaitos sparta. Kaip taisyklė: kuo didesnis abstraktumo lygis – tuo mažesnis pasikeitimų tempas. Programavimo lygyje, atsiranda naujos programavimo kalbos, o jau egzistuojančios (pvz. Java arba C#) vystomos toliau. Jau daugelį metų programos skaidomos į skirtingo dydžio komponentes ir tų komponenčių realizavimo metodai ženkliai keitėsi bėgant laikui, EJB (*Enterprise Java Beans*) standartas taip pat ir Microsoft kompanijos .NET platforma gali būti paminėti kaip pavyzdžiai. Kitaip negu realizavimo metodai, programinės įrangos struktūrizavimas ir modeliavimas tapo pastovesnis, ypač kai atsirado standartizuotos modeliavimo kalbos tokios kaip OMG (*Object Management Group*) konsorciumo vieninga modeliavimo kalba UML (*Unified Modeling Language*).

OMG pasiūlė UML kalbą programinės įrangos (ir ypač objektiškai orientuotos) elementų ir sistemų aprašymui. Vidinė UML architektūra ir taikymų ribos dar nėra visai nusistovėję, nes paskutinė UML 2.0 kalbos versija ženkliai skiriasi nuo ankstesnių. Norėdama standartizuoti UML ir kitų į UML panašių modeliavimo kalbų kūrimo procesą, OMG sukūrė MOF (*Meta Object Facility*) modeliavimo kalbą. UML ir MOF yra esminės keturių lygių MDA modeliavimo aplinkos dalys [1, 2-2p.].

Faktai, kad IT kaita yra nuolatinė ir, kad kaitos sparta mažėja – didėjant abstraktumo lygiui, tapo OMG pagrindu naujai modeliavimo strategijai. Jeigu skirtingi abstraktumo lygiai gali būti tiksliai nustatyti, tai technologinių pasikeitimų įtaka galima apriboti modelio poaibiu. Vietoj to, kad perdaryti visą sistemos modelį, galima pakeisti tik technologijos pasikeitimo paveiktas dalis.

Modeliais Pagrįsta Architektūra (MDA) (*Model Driven Architecture*) – tai OMG iniciatyva, kuri pateikia efektyvaus programinės įrangos modelių kūrimo ir panaudojimo strategijas. MDA apibrėžia tokį IT sistemų specifikavimo būdą, kuris atskiria sistemos funkcionalumo specifikaciją nuo sistemos realizavimo specifikacijos tam tikrai technologinei platformai [2, 3p.]. MDA nėra nauja architektūra – tai nauja programinės įrangos modelių kūrimo strategija. Šitos strategijos vienas iš siekių – skirtingų realizavimo technologijų ir standartų

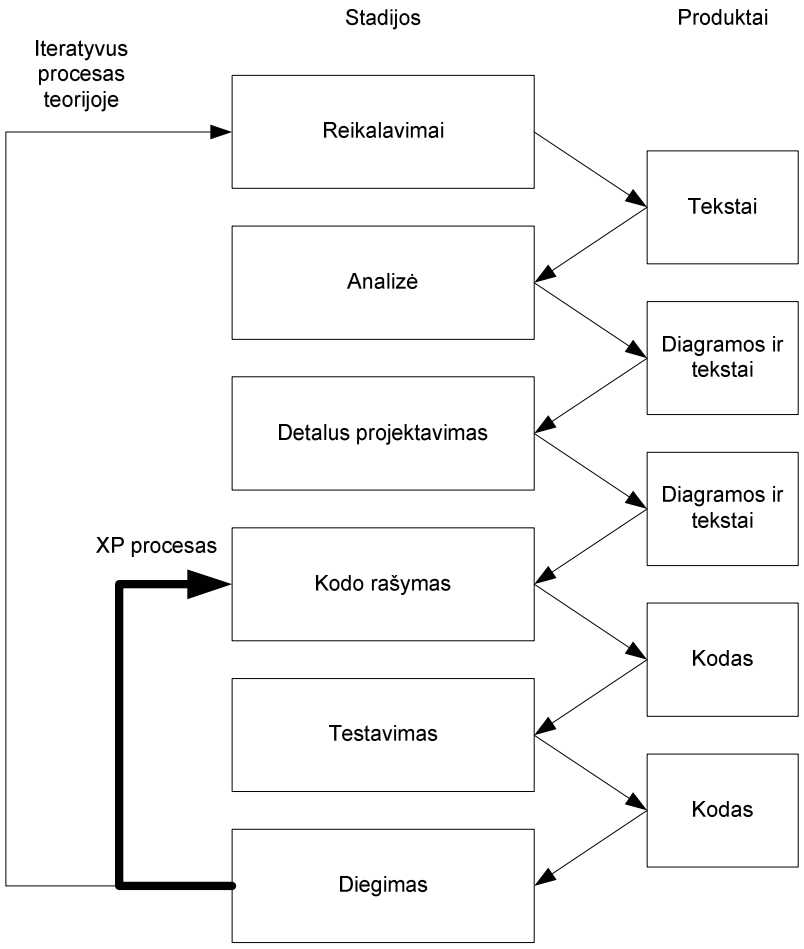
sambūvio galimybė. MDA tikslas – ne vieno uniforminio standarto įdiegimas, o aiškus skirtingų abstrakcijos lygių atskyrimas. MDA architektūros taikymas turėtų iš esmės pakeisti šiuolaikinę programų kūrimo praktiką. Toliau detaliau panagrinėsime programų kūrimo proceso problemas, kurias turėtų išspręsti programų kūrimo pagal MDA metodikos.

Pirmoji problema – tai programos kūrimo proceso produktyvumas. Programos kūrimo procesas, kokį mes jį žinome iš šiuolaikinės praktikos, didžiąja dalimi remiasi detaliu projektavimu ir programos kodo rašymu [3, 2p.]. Tipinis programos kūrimo procesas pereina tokias stadijas:

1. Reikalavimų surinkimas;
2. Analizė ir funkcinis aprašymas;
3. Projektavimas;
4. Kodo rašymas;
5. Testavimas;
6. Diegimas.

Šio proceso metu, nuo 1 iki 3 fazės, sukuriami reikalavimų tekstai, diagramos, programos funkcinės specifikacijos dokumentai ir daugybe UML diagramų. Gaunamas dokumentų kiekis būna gana didelis ir jų kūrimui panaudojama daug nemažai ir laiko, tačiau – tai tik dokumentai. Kai tik pradedama programos kodo rašymo stadija – dokumentų vertė pradeda sparčiai nykti. Tęsiantis programos kodo rašymui, dokumentų ir realizacijos neatitikimas vis didėja. Keičiantis jau sukurtai sistemai, pakeitimai dėl laiko ir išteklių stokos dažnai daromi tik kodo lygyje. Iškyla klausimas: Kam gaišti tiek brangaus laiko aukšto lygmens specifikacijoms, jeigu be tinkamo atnaujinimo jos greitu laiku praranda vertę?

Ekstremalaus programavimo XP (*Extreme Programming*) metodika pasidarė gana populiari praktikoje, nes pasižymi didele programos kūrimo proceso sparta. XP remiasi tuo, kad programos kodas yra lemiantis programos kūrimo elementas, todėl realiai tik programavimas ir testavimas yra produktyvios šio proceso stadijos (žr. 1 pav.). Tačiau ši metodika išsprendžia tik dalį problemos [3, 2p.]. Kol projekte dirba ta pati komanda – aukšto lygio žynių apie sistemą yra pakankamai, kad užtikrinti projekto eigą. Tačiau po pirmos programos versijos išleidimo projekto komanda yra išskaidoma ir kiti žmonės perima programos palaikymą ir klaidų taisymą. Programos palaikymas yra praktiškai neįmanomas turint tik kodą ir testus.

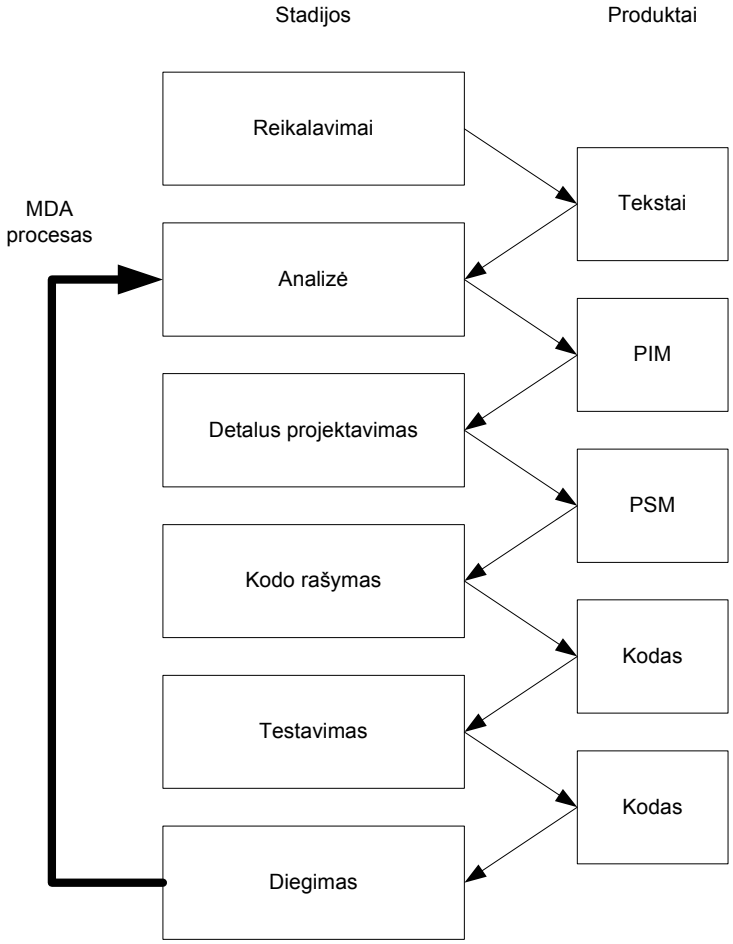


1 pav. -- Programų kūrimo stadijos (pagal [3, 3p.]).

Kita problema su kuria susiduria programinės įrangos pramonė – tai sparti technologijų kaita. Įmonės investuoja didžiules lėšas programų kūrimui šiuolaikinėmis technologijos, tačiau dar net dalinai neatsipirkus sukurtam produktui atsiranda naujos technologijos arba pasikeičia esamos. Dauguma bibliotekų ir programavimo priemonių tiekėjų palaiko tik paskutines tris versija, todėl neatnaujinus sistemos – galima likti be palaikymo iš tiekėjų pusės. Įmonės konkurencijos sąlygomis negali sau leisti atsilikti ir tenka adaptuosis prie naujų technologijų.

Galima jau egzistuojančią ir veikiančią sistemą naudoti ir toliau, tačiau čia iškyla sąveikavimo ir integravimo problemos (*interoperability problem*) – juk senoji sistema turi veikti su naujai sukurtąja. Programinės sistemos retai būna izoliuotos. Daugumai programinių produktų tenka sąveikauti su jau egzistuojančiomis programomis. Taip pat paminėtina, kad net jeigu programa rašoma iš naujo, ji dažniausiai naudoja keletą technologijų, bibliotekų ir programavimo aplinkų (*frameworks*), ir naujai sukurtų ir jau seniai naudojamų. Atskiriems programų komponentėms rašyti naudojamos tam labiausiai tinkamos technologijos, tačiau komponentės turi sąveikauti norint, kad sistema veiktų [3, 5p.].

MDA – tai aplinka (*framework*) programų kūrimui, pasiūlyta OMG konsorciumo, kuri turėtu išspręsti bent dalį anksčiau paminėtas programavimo proceso problemas ir suteikti naujas programinių sistemų atnaujinimo ir integravimo galimybes. Programų kūrimo proceso fazės pagal MDA labai panašios į klasikinio, tačiau stadijų produktai yra skirtingi (žr. 2 pav.).



2 pav. -- Programos kūrimo pagal MDA procesas (pagal [3, 7p.])

Antros ir trečios fazės produktai – tai formalūs programinės sistemos modeliai, t.y. – šie modeliai ne tik suprantami žmonėms, bet gali būti apdoroti kompiuteriu. MDA programų kūrimas yra pagrįstas modeliais (*model-driven*), nes modeliai nukreipia sistemos kūrimą, modifikavimą ir diegimą [4, 2-2p.].

Nuo platformos nepriklausomas modelis PIM (*Platform Independent Model*) – tai sistemos atvaizdas (*view*) nuo platformos nepriklausomu požiūriu [4, 2-6p.]. PIM aprašo sistemą abstrakčiai ir tam tikru laipsniu nepriklauso nuo realizacijos detalių tam tikrai platformai. PIM kuriama taip, kad kuo tiksliau aprašytų probleminę sritį (*business domain*).

Kitas modelis kurį aprašo MDA – tai platformai specifinis sistemos modelis PSM (*Platform Specific Model*). PSM aprašo sistemą terminais specifiniais tam tikrai platformai

kurioje bus realizuota programinė sistema [4, 2-6p.]. Kuriant programą MDA aplinkoje – analizės fazės metu sukurtas PIM modelis transformuojamas į vieną ar kelis PSM. Kadangi platformai specifinis modelis aprašo tam tikrai technologinei platformai būdingas detales, jis gali būti lengvai transformuotas į programos kodą [3, 6p.].

Paminėsime dar viena modelių tipą – tai nuo skaičiavimų nepriklausomas modelis CIM (*Computationally Independent Model*). CIM yra bazė PIM modeliui, tačiau jis visiškai skirtas tik probleminės srities esybėms aprašyti ir visai neturi informacijos apie pačią skaičiavimo sistemą ir jos skaičiavimų aplinką (*computational environment*), dėl šitos priežasties jis nebuvo paminėtas MDA programos kūrimo procese.

Tradicinio programavimo proceso metu, perėjimas nuo vieno modelio prie kito ar perėjimas nuo modelio prie programinio kodo atliekamas rankiniu būdu. Naudojant kodo generavimo įrankius galima dalį programinio kodo sugeneruoti, tačiau generacijos rezultatas – tai tik šablonas, ir vis tiek didžiąją dali jo tenka užpildyti rankomis (žr. 1 lent.). Tačiau MDA transformacijos visada vykdomos automatiškai naudojant programinius įrankius. Dauguma klasikinių automatinio kodo generavimo įrankių moka sugeneruoti programinį kodą iš PSM, nes šio modelio elementai ir sąryšiai tiesiogiai atitinka konkrečią platformą. Tai ką MDA apibrėžia naujo – tai PIM aukšto abstraktumo modelio automatinis transformavimas į tam tikrai platformai specifinį PSM modelį [3, 8p.].

1 lent. -- Programos kodo generacijos efektyvumas (ištrauka iš [5, 9p.])

	Neautomatizuotas kodo rašymas	**Paprastas UML modeliavimas**	**MDA**
Rankiniu būdu parašytų kodo eilučių skaičius	126	71	20
Sugeneruotų eilučių skaičius	0	55	106
Visa kodo apimtis eilutėmis	126	126	126
Rankomis parašytų eilučių kiekis [%]	100	56	16

Matome, kad modelių transformavimas yra esminis programų kūrimo pagal MDA procesas, todėl nustatyti tokie šio tiriamojo darbo tikslai:

1. Susipažinti su transformacijų MDA aplinkoje ypatumais ir sąsaja su OMG standartais;
2. Sukurti eksperimentinius išeities ir tikslo metamodelius;

3. Išanalizuoti eksperimentinių modelių transformacijų procesą ir suprojektuoti transformacijos algoritmą;
4. Sukurti eksperimentinės transformacijos algoritmo realizaciją;
5. Įvykdyti eksperimentines modelių transformacijas.

Sekančiame skyriuje pagrindžiamas tolesnis eksperimentinių metamodelių konstravimas, bei parenkamos transformacijų realizavimo priemonės.

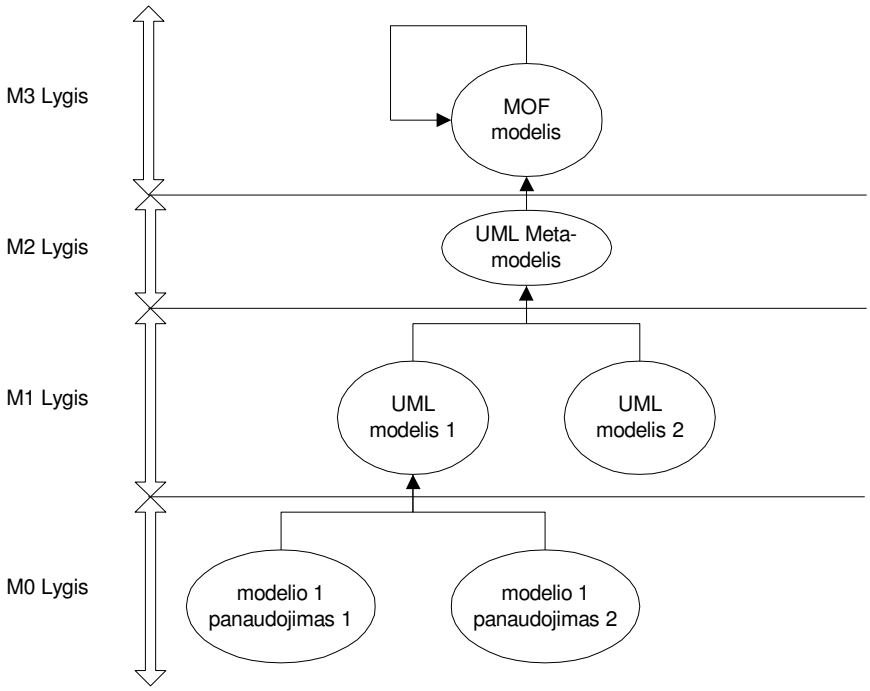
2 MODELIŲ TRANSFORMACIJOS MDA ARCHITEKTŪROJE

MDA architektūros esminis elementas – programinės sistemos ir probleminės srities modeliavimas. Tačiau modeliavimas MDA iš esmės nesiskirtu nuo kitų, kodo generavimu pagystų, programų kūrimo metodikų, jeigu perėjimai nuo PIM prie PSM modelių nebūtų vykdomi automatizuotai. Tokiu būdu, norint pasinaudoti visais MDA pranašumais – reikia realizuoti įrankius kurie vykdytų PIM – PIM ir PIM – PSM transformacijas. 2.1 skyriuje panagrinėsime kokią reikšmę modelių transformacijų realizavimui turi metamodeliavimas. 2.2 skyriuje apžvelgsime transformavimo strategijas. Pasirinktos transformacijos strategijos realizavimui, 2.4 skyriuje, bus palygintos ir parinktos eksperimentinės realizacijos aplinkos.

2.1 Metamodeliavimo reikšmė modelių transformacijoms

OMG konsorciumas sukūręs vieningą modeliavimo kalbą UML pateikė nauja standartą objektinės programinės įrangos modeliavimui ir projektavimui. Modelis yra sistemos ar jos dalies aprašymas formalia modeliavimo kalba [3, 16p.]. Tačiau, kad modelis būtų formalus jis taip pat turi būti apibrėžtas formaliai. Formaliems kalbų aprašymams naudojamos formalios kalbos vadinamos metakalbomis. Pavyzdžiui Pascal kalbos gramatiką galima aprašyti BNF kalbos priemonėmis. Tokiems formaliems metaduomenų ir modeliavimo kalbų aprašymams kurti OMG konsorciumas sukūrė MOF (Meta Object Facility) kalbą [6, 1-15p.]. Formalių modeliavimo kalbų aprašymų specifikavimas vadinamas metamodeliavimu. MOF kalba turėjo didelę įtaką esminių MDA principų kūrimui. MOF kalba[1] ypatinga tuo, kad ji yra save aprašanti, t.y. MOF kalba galima aprašyti ne tik kitų kalbų metamodelius, bet ir jos pačios modelį. MOF – tai ne tik gali būti naudojama naujų modeliavimo kalbų specifikavimui, bet ir universalus metaduomenų aprašymo ir saugojimo standartas [1, 2-2 ir 2-3p.]. MOF kalba atitinka OMG keturių lygmenų metadomenų architektūros aukščiausiąjį lygmenį (žr. 3 pav.).

[1] Paminėtina, kad sąvokos „kalba“, „metakalba“ ir „metametakalba“ yra reliatyvios, t.y. priklauso nuo nagrinėjamo abstrakcijos lygmens santykio su kitu lygmeniu, pvz. pagal 3 pav., jeigu kalbame apie UML – tai UML šiuo metu vadinama „kalba“, UML kalbos metamodelis – aprašytas „metakalba“ MOF, o MOF metamodelis aprašytas – „metametakalba“, kuri irgi yra MOF.



3 pav. -- Keturi OMG metaduomenų architektūros lygmenys

Kiekvienas OMG metaduomenų architektūros lygmuo atitinka tam tikrą modelių abstrakcijos lygį (žr. 2 lent.).

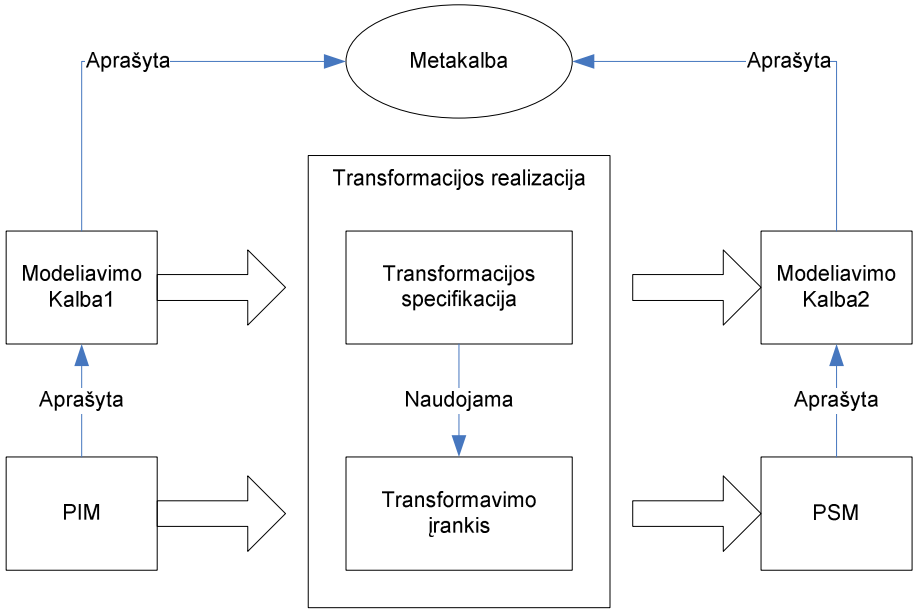
2 lent. -- Keturi metaduomenų architektūros lygmenys

Abstraktumo lygmuo	**Aprašymas**
M3	Metametamodeliai, kurių elementai susieti statiniais ryšiais. MOF modelio lygmuo.
M2	Metamodeliai, kurių elementai – tai M3 lygmens modelio elementų egzemplioriai (*instances*) ir M2 lygmens metaklasės. Šitas lygmuo atitinka UML kalbos gramatiką – UML metamodelį.
M1	Modeliai, kuriu elementai – M2 lygmens modelio egzemplioriai. Tai UML modelių lygmuo.
M0	Modelių panaudojimas. Šito lygmens elementai yra M1 lygmens elementų egzemplioriai, pvz. M1 lygmenyje – apibrėžiama klasė `Customer`, tai M0 lygmenyje klasės egzemplioriai bus `Customer` tipo objektai.

Literatūroje [3, 90p.] minimos dvi priežastys dėl kurių metamodeliavimimas labai svarbus modelių transformavimui ir MDA:

1. Tai modeliavimo kalbų specifikavimo priemonė MDA architektūroje. Metakalbos pagrindu apibrėžę modeliavimo kalbos metamodelį galime specifikuoti PIM ar PSM kalbą.
2. Transformacijų taisyklės (*rules*) aprašo kokie pradinio modelio elementai turi būti transformuoti į atitinkamus tikslinio modelio elementus.

4 pav. demonstruoja kaip metamodeliavimas siejamas su PIM – PSM modelių transformacijomis.



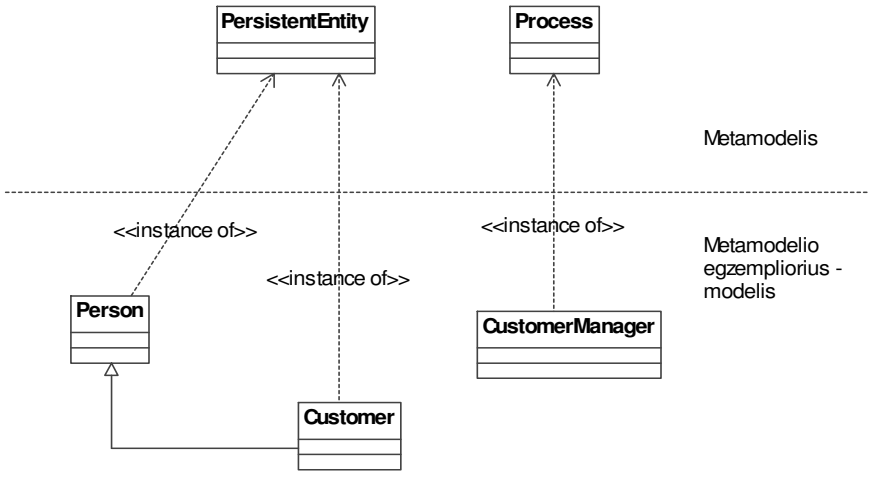
4 pav. -- Metamodeliavimo ir modelių transformacijų sąryšis

Pradiniame transformacijų realizavimo etape, naudojant metakalbą, mums tenka specifikuoti formalią PIM modeliavimo kalbą. Tai pasiekiama aprašant PIM metamodelį. PIM metamodelio pagrindu sukuriamas PIM modelio egzempliorius, kuris atitinka mūsų modeliuojamą programinę sistemą. Norint gauti tam tikrai programavimo platformai specifinį PSM modelį, mes taip pat turime specifikuoti PSM modelio formalizmą (*formalism*) aprašydami PSM metamodelį. Galutinis aplinkos kūrimo etapas – tai transformavimo įrankio realizavimas. Transformavimo įrankis, naudodamas transformacijos specifikaciją, transformuos mūsų PIM į PSM. Šiuo požiūriu transformacijų realizacija yra neatskiriama nuo metamodeliavimo. Tokiu būdu, norėdami realizuoti transformaciją, pradžioje turime apsibrėžti išeities PIM ir tikslo PSM modeliavimo kalbas.

2.2 Modelių susiejimo strategijos

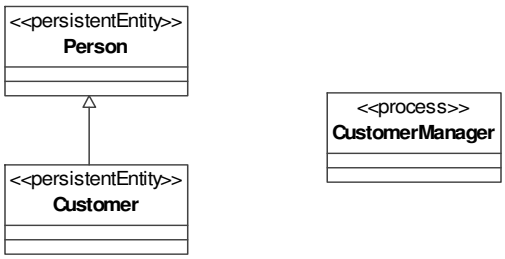
Norint įvykdyti išeities modelio transformaciją į tikslo modelį – turime aprašyti tų modelių elementų loginius sąryšius. Toks modelių elementų sąryšių specifikavimas vadinamas modelių susiejimu (*model mapping*). OMG mini tris [4, 3-2p.] išeities ir tikslo modelių susiejimo strategijas:

1. Modelių tipų (metamodelių) susiejimas – tai toks modelių susiejimas, kai susiejami PIM modeliavimo kalbos aprašomi tipai su PSM modeliavimo kalbos aprašomais tipais. Būtent šitam modelių susiejimui svarbus metamodeliavimas, kurio svarba buvo nagrinėta praeitame skyriuje, nes susiejami vienos modeliavimo kalbos metamodelio elementai su kitos modeliavimo kalbos metamodelio elementais. 5 pav. pateiktas pavyzdys kaip susiję skirtingo tipo modelio elementai su metamodelio klasėmis.



5 pav. -- Modelio elementai - kaip metamodelio klasių egzemplioriai

2. Modelių egzempliorių susiejimas – tai susiejimas modelių egzempliorių lygyje, kai apibrėžiama kokias sąlygas atitinkantis vieno modelio elementas susiejamas su kitu antrojo modelio elementu. Elementų sąlygų nustatymui naudojamos žymės (*marks*), pvz. UML klasei `Customer` uždedama žymė panaudojant stereotipą `<<persistentEntity>>` ir transformacijos apibrėžime nurodoma, kad PIM modelio klasės su šita žyme bus transformuojamos į esybės EJB komponentą (*entity bean EJB*). Tokio modelio pavyzdys pateiktas 6 pav..



6 pav. -- Skirtingų tipų modelio elementų atvaizdavimas naudojant kaip žymes stereotipus

Modelių tipų (metamodelių) susiejimas yra griežtai formalus, nes susiejami elementai nurodomi metamodelio lygyje ir tokių būdu apibrėžia modeliavimo kalbos formalizmą. Toks griežtai formalus susiejimas leidžia apriboti modeliavimo kalbos vartotoją nuo nekorektiškų modelių sukūrimo.

Modelių egzempliorių susiejimas – nėra formalus ir griežtas naudojamos modeliavimo kalbos požiūriu, nes susiejami ne metamodelio elementai, kurie yra modeliavimo kalbos formalizmo dalis, o jų egzemplioriai. Tokio susiejimo metu – labai sudėtinga automatizuotai patikrinti sudaryto modelio korektiškumą, pvz. UML kalba leidžia klasėm uždėti stereotipus, tačiau šitos kalbos formalizmas nenustato, kokios, mūsų sudaryto modelio, klasės gali tūrėti stereotipą `<<entity>>`, o kokios negali.

Tokiu pagrindu eksperimentinėms transformacijoms pasirinkau metamodeliavimo strategiją ir metamodelių susiejimą, nes jis leidžia sukurti griežtai apibrėžtus, formalius modelius ir modelių transformacijas.

2.3 Modelių transformacijų specifikavimo strategijos

Transformacijos realizacijoje galime išskirti dvi pagrindines dalis: transformavimo įrankį ir transformacijos specifikaciją (*transformation definition*) (žr. 4 pav.). Šios dalys priklausomai nuo pasirinktos realizacijos architektūros gali būti atskiros arba apjungtos kaip vienas įrankis su įprogramuotu transformacijos šablonu (*transformation pattern*). Realizacijos architektūros pasirinkimas priklauso nuo transformacijų specifikavimo strategijos. Galima tris transformacijų specifikavimo strategijas:

1. Algoritminis transformacijų realizavimas – tai tokios modelių susiejimo ir jų transformacijų realizacijos, kurios panaudoja jau egzistuojančias algoritmines programavimo ar transformavimo kalbas. Galima būtų išskiri:

a. Algoritminių kalbų panaudojimą (pvz. Java) – čia transformacijos realizacija fiksuotai įprogramuojam į transformavimo įrankį arba transformuojanti dalis užkraunama dinamiškai (*plug-in*);
b. Interpretuojamų kalbų panaudojimas pačių transformavimo taisyklių aprašymui (pvz. Java);
c. Šablonų kalbų panaudojimas (pvz. XSL, Velocity).

2. Metamodeliavimo – tai modelių susiejimo ir transformacijų specifikavimo būdas, kuris pagrįstas metamodelių ir jų programavimo sąsajų (*API*) panaudojimu. OMG konsorciumas paskelbė užklausą pasiūlymams RFP (*Request for Proposal*) dėl MOF 2.0 užklausų, atvaizdų ir transformacijų (*MOF-QVT – MOF Model / Views /Transformations*) [7] ir tikisi sukurti naują standartizuotą transformacijų modeliavimo kalbą. Transformacijų modeliavimo kalba leistų metamodelių lygyje ir naudojant modeliavimo priemones susieti modelius ir specifikuoti transformaciją iš vieno modelio į kitą. OMG jau gavo atsakymų į MOF-QVT RFP, tačiau šita MOF metakalbos dalis dar nestandartizuota iki šio momento.

Vienas šio darbo tikslų – realizuoti eksperimentinę modelių transformaciją. 2.2 skyriuje pasirinkome metamodelių susiejimo strategiją mūsų eksperimentinės transformacijos realizacijai. Šiai realizacijai laikantis labiausiai tinkantis transformacijų specifikavimo metodas būtų transformacijų modeliavimo kalba, nes jos priemonėmis galima būtų formaliai susieti modelius ir tuo pagrindu realizuoti vaizdinio modeliavimo įrankius. Kadangi metamodeliavimo principu besiremiančio transformacijų modeliavimo standarto nėra, tai pasirenkame algoritminį transformacijų realizavimo būdą. Programinių realizavimo priemonių pasirinkimas bus analizuojamas 2.4 šio darbo skyriuje.

2.4 Transformacijų realizavimo priemonės

Transformacijoms realizuoti pasirinkau Java programavimo kalbą. Ši kalba gana populiari, jai tiekiama daug programavimo aplinkų ir bibliotekų. Taip pat šios programavimo kalbos pagrindu sukurta J2EE (*Java2 Enterprise Edition*) platforma vis dar pati populiariausia įmonės lygmens (*enterprise*) interneto programų kūrimo platforma, o būtent įmonės lygmens programinei įrangai yra svarbūs tikslai kurių siekiama MDA.

Atlikdamas tyrimą užsibrėžiau sukurti išeities ir tikslo modelius ir realizuoti jų transformaciją. Tokiu būdu mums reikia:

1) Specifikuoti išeities metamodelį;

2) Specifikuoti tikslo metamodelį;
3) Sukonstruoti išeities modelį;
4) Realizuoti transformaciją:
    a. Išeities modelio nuskaitymą;
    b. Išeities modelio elementų manipuliavimą ir jų susiejimą su tikslo modeliu;
    c. Gauto tikslo modelio išsaugojimą.

OMG apibrėžė XMI, kaip standartinį modelių ir metamodelių saugojimo būdą [9]. Pasirinkus realizacijos kalbą Java, šiuo metu galima aptikti dvi programavimo aplinkas (*frameworks*), kurių pagalbą galima būtų realizuoti modelių nuskaitymą ir išsaugojimą į XMI formato bylas – tai NSUML (http://nsuml.sourceforge.net/) ir EMF (http://www.eclipse.org/emf). Šių programavimo aplinkų analizės rezultatai pateikti 3 lent..

3 lent. -- NSUML ir EMF programavimo aplinkų palyginimas

Funkcionalumas	**NSUML**	**EMF**
Palaikoma XMI versija	UNISYS XMI 1.1	XMI 2.0
Suteikiamos metamodelių kūrimo priemonės	Nėra	• Java interfeisai; • Ecore editorius; • XML schemos (XSD);
Palaikomos išorinės metamodelių kūrimo priemonės.	MagicDraw	Rational Rose
Integracija su darbo aplinkom	Nėra	Eclipse platforma
Kurią MOF standarto versiją atitinka galimybės	MOF 1.4	EMOF (*Essential MOF*) 2.0[2]
Gauto modelio manipuliavimo programavimo aplinka (*API*)	Generatorius, kuris sugeneruoja MOF programavimo sąsają.	Tiekiamas generatorius, kuris sugeneruoja reikiamas EMF.Edit aplinkos klases.
Galimybė išsaugoti rankomis pakoreguotas kodo dalis vykdant kitas modelių	Nėra	Automatiškai apjungia naujai generuojamus ar pergeneruojamus senus

[2] Paminėtina, kad EMF aplinka teikia Ecore modeliavimo kalbą. Ecore pagal funkcionalumą atitinka EMOF 2.0 ir galima ją laikyti EMOF realizacija, tačiau vengiant painiavos šita kalba vadinama Ecore [10, 37p.].

programavimo sąsajų generacijas.		elementus ir rankomis realizuotas dalis.
Galimybė sukurti gauto metamodelio egzempliorius.	Programiniu būdu.	• Programiniu būdu; • Galima sukurti grafinį redaktorių.

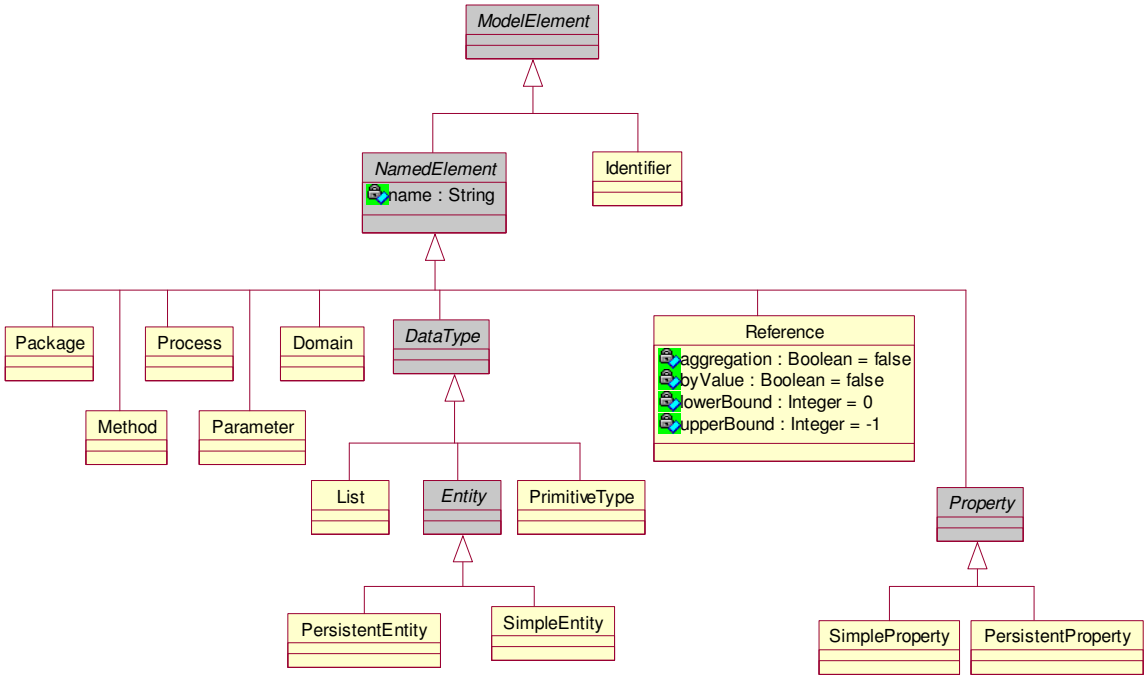
Matome, kad EMF programavimo aplinka palaiko modernesnius standartus ir suteikia daugiau galimybių kuriant metamodelius ir manipuliuojant jais. Taip pat EMF integruojasi su patogia ir šiuo metu greitai besivystančia programavimo platforma Eclipse (http://www.eclipse.org). Todėl EMF programavimo aplinką pasirinkau eksperimentiniams metamodeliams sukurti ir jų transformacijoms realizuoti.

3 PIM „ESYBĖ – PROCESAS“ METAMODELIO SUDARYMAS

Norint realizuoti eksperimentinę transformaciją, kaip minėjau 2.1 skyriuje, reikia sukurti išeities ir tikslo modeliavimo kalbas. Pradinėje tyrimo stadijoje buvo sukurta PIM modeliavimo kalbą. Ši kalba buvo pavadinta „esybės – proceso“ modeliavimo kalba. PIM kalbos sudarymui buvo nustatyti sekantys reikalavimai:

1. Kalba turi būti paprasta, ir tūrėti mažai techninių detalių;
2. Kalboje turi išsiskirti du pagrindiniai verslo probleminių sričių elementai – tai esybės ir procesai:
    a. Esybė – tai duomenų struktūravimo vienetas, kuris saugo informaciją apie probleminės srities objektą ir jo būseną. Galima išskirti du esybių tipus: tai išsaugančios savo būseną tik vykdymo metu (*runtime*) ir nuolat išsaugančios savo būseną (*persistent*).
    b. Procesas – tai būsenos neturintis probleminės srities elementas, kuris savyje paslepia (*encapsulates*) vykdymo logiką. Elementarus proceso žingsnis – metodas, kuriam galima apibrėžti parametrus ir gražinamą rezultatą. Procesai negali saugoti savo būsenos (*stateless*).
3. Kalboje galima apsibrėžti duomenų tipus, kurie bus naudojami atominiam atributam sukurti;
4. Tarp esybių galima nustatyti sąryšius, palaikomas ryšių kardinalumas 1:1, 1:N;
5. Turi būti galimybė apibrėžti tipizuotus objektų sąrašus.

Sukurtos PIM modeliavimos kalbos metamodelių paveldėjimo hierarchija parodyta 7 pav..



7 pav.– „Esybė – procesas“ metamodelio paveldėjimo hierarchija

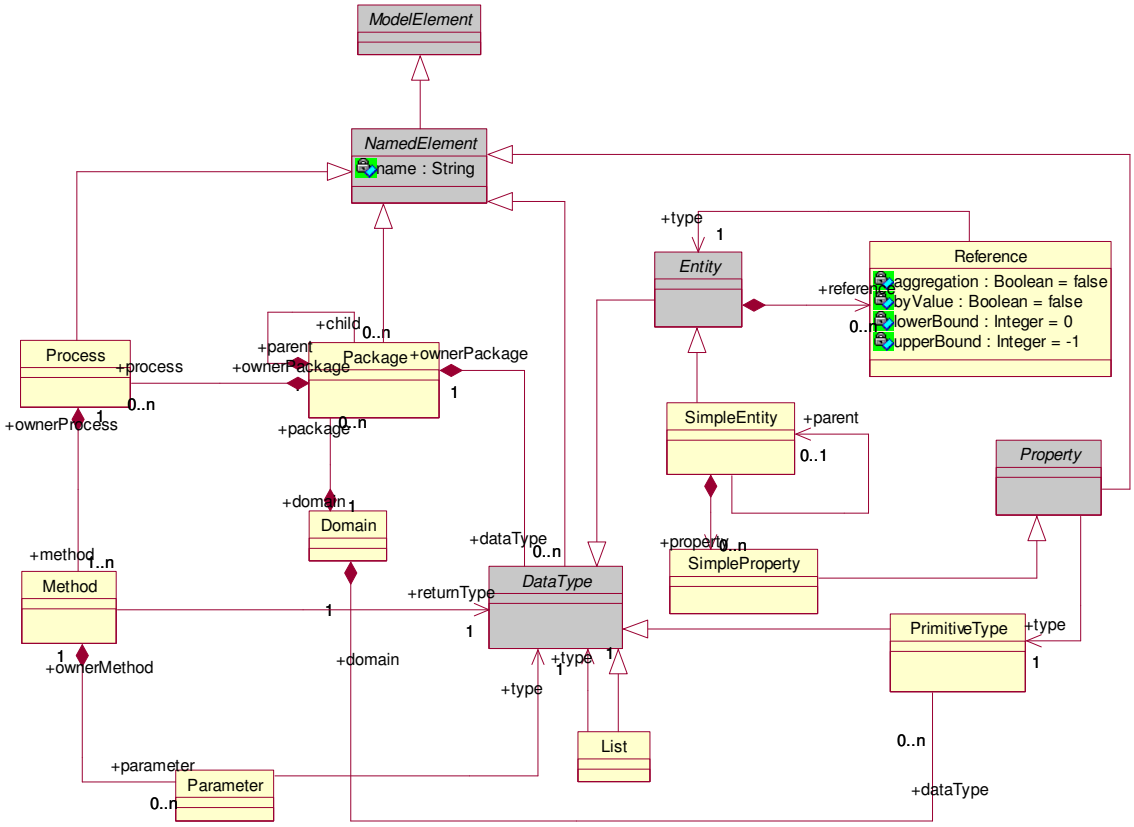
Abstraktūs metamodelio elementai pažymėti tamsesne spalva „Esybė-procesas“ modeliavimo kalbos elementų aprašymai pateikti 4 lent..

4 lent. – „Esybė – procesas“ PIM kalbos metamodelio elementai

Metamodelio klasė	**Aprašymas**
DataType	Bazinė visų duomenų tipų metaklasė; abstrakti ir negali būti sukurtas jos egzempliorius.
Domain	Tai šakninis PIM modelio elementų konteineris. Sukūrus nauja modelio failą – pirma sukuriamas šis elementas.
Entity	Bazinė visų tipų esybių metaklasė. Tarp esybių galima nurodyti paveldėjimo sąryšius.
Identifier	Identifikatoriaus žymeklis, kuris nurodo, koks esybės atributas yra unikalus jos identifikatorius.
List	Tai sąrašo metaklasė.
Method	Metodas – proceso elementas. Gauna duomenis per parametrus ir gražina tam tikrą vykdymo rezultatą.
ModelElement	Bazinė visų metamodelio elementų (metaklasių) klasė. Abstrakti ir negali būti sukurtas jos egzempliorius.

NamedElement	Abstraktus metamodelio elementas. Jo egzempliorius negali būti sukurtas, tačiau jis apibrėžia bendrą visiems metamodelio elementams (išskyrus identifikatorių) savybę – vardą.
Package	Modelio elementų grupavimo mechanizmas. `Package` elementas iš principo atitinka UML kalboje naudojama elementą su tuo pačiu pavadinimu.
Parameter	Tai metodo parametro metaklasė. Privalomas nurodyti parametro duomenų tipas (DataType).
PersistentEntity	Nuolat savo būsena išsauganti (*persistent*) esybė. Tokia esybė būtinai turi turėti identifikatorių, jeigu jo neturi tėvinė esybė. `PersistentEntity` metaklasės egzempliorius (klasė) gali būti susieta paveldėjimu tik su tokia klase, kurios metaklasė taip pat `PersistentEntity`.
PersistentProperty	Tai atributų nuolat išsaugančių savo būseną (*persistent*) metaklasė.
PrimitiveType	Taip primityviem tipam kurti skirta metaklasė.
Process	Verslo procesą modeliuojanti metaklasė. Verslo procesai negali tūrėti būsenos (*stateless*) ir nepalaiko paveldėjimo sąryšio.
Property	Abstrakti esybės atributo metaklasė. Negalima sukurti nė vieno jos egzemplioriaus.
SimpleEntity	Būseną tik vykdimo metu išlaikanti esybė (*non-persistent*).
SimpleProperty	Būseną tik vykdymo metu išlaikantis esybės atributas (*non-persistent*).

„Esybės – proceso“ metamodelio elementų ryšiai pavaizduoti 8 pav.. Matome, kad kiekvieno modelio (metamodelio egzemplioriaus) klasių pagrindinis konteineris yra `Domain` metaklasės egzempliorius. `Domain` metaklasė kompozicijos ryšiais susieta su `Package` ir `PrimitiveType` metaklasėmis, o visi likę duomenų tipai ir `Process` egzemplioriai – gali būti talpinami tik į `Package` elementus. Paketų kompozicijos ryšys – rekurentinis, t.y. paketas gali turėti kitus paketus.



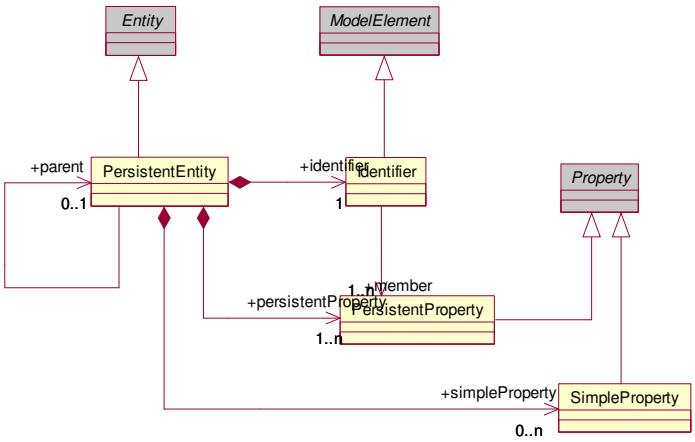
8 pav. – „Esybė-procesas“ PIM metamodelio elementų ryšiai (2 dalis)

Paketai – tai modelio elementų grupavimo mechanizmas. Paketo sudėtinės dalys gali būti esybės, ir duomenų tipai (esybės, paprasti duomenų tipai ir sąrašai).

Tarp nuorodų gali būti sukuriamos nuorodos (*references*). Nuorodom galima nurodyti jų tipą, pavadinimą, agregavimo būdą (kompozicija ar agregavimas) bei kardinalumą.

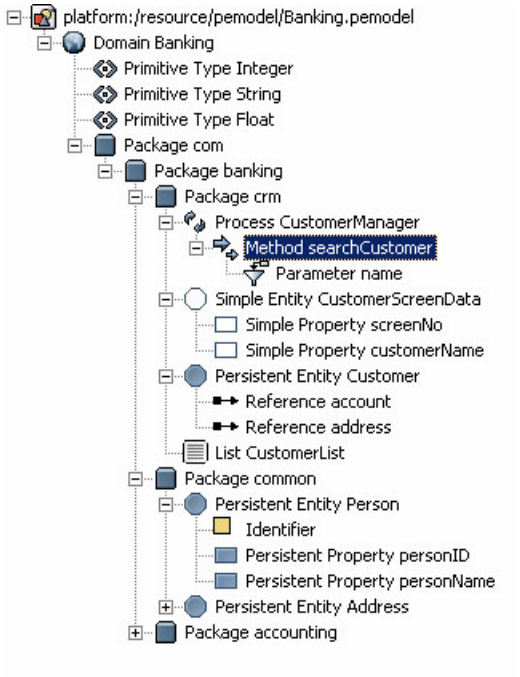
9 pav. vaizduoja būseną išsaugančių esybių ir jų atributų sąryšius. Matome, kad būseną išsauganti esybė, gali turėti ne tik būseną išsaugančius laukus, bet ir tik vykdymo metu saugomus laukus. Paminėsime, kad būseną nuolat išsaugančių ir būseną tik vykdymo metu išsaugančių objektų paveldėjimo medžiai yra atskirti.

„Esybė – procesas“ metamodelio formali specifikacija XMI 2.0 formate pateikta priede 10.1.



9 pav. – „Esybė-procesas“ PIM metamodelio būsenas išlaikančios esybės (3 dalis)

Sukurto PIM metamodelio pagrindu, naudojant EMF priemones, buvo sukurtas modelio redaktorius. Pavyzdinio metamodelio egzemplioriaus atvaizdavimas parodytas 10 pav.. Naudojant „Esybė – procesas“ modelio redaktorių galima sukurti PIM modelio egzempliorių, kurį vėliau naudosime eksperimentinėms modelių transformacijoms.



10 pav. -- Pavyzdinis „esybė-procesas“ metamodelio egzempliorius

Redaktorius atvaizduoja visą XMI formate saugomą modelio egzempliorių kaip medžio struktūrą. Kiekviena medžio viršūnė atitinka tam tikros metamodelio klasės egzempliorių. Modelio redaktoriaus piktogramų aprašymai pateikiami 5 lent..

5 lent. -- PIM modelio redaktoriaus piktogramos ir jų reikšmės

Piktograma	Paaiškinimas
	Atitinka `Domain` klasės egzempliorių. Tai išorinis viso modelio konteineris.
	Atitinka `Package` klasės egzempliorių. Tai paketo objektas.
	Atitinka `PrimitiveType` klasės egzempliorių. Šituo elementu vaizduojami abstraktūs atominiai tipai, pvz. sveikieji skaičiai, eilutės, slankaus kablelio skaičiai.
	Atitinka `List` klasės egzempliorių. Tai tipizuoto sąrašo duomenų tipą modeliuojantis elementas.
	Atitinka `SimpleEntity` klasės egzempliorių. Neišliekančią (*non-persistent*) esybę modeliuojantis elementas.
	Atitinka `SimpleProperty` klasės egzempliorių. Neišliekantį atributą modeliuojantis elementas.
	Atitinka `PersistentEntity` klasės egzempliorių. Išliekanti esybė (*persistent entity*).
	Atitinka `PersistentProperty` klasės egzempliorių. Išliekantis atributas.
	Atitinka `Identifier` klasės egzempliorių. Elementas, kurio pagrindinė funkcija – nurodyti, kurie esybės atributai – yra esybės identifikatoriai.
	Atitinka `Reference` klasės egzempliorių. Nuorodą į kitą objektą modeliuojantis elementas.
	Atitinka `Process` klasės egzempliorių. Proceso metaklasės egzempliorių atvaizduojantis elementas.
	Atitinka `Method` klasės egzempliorių. Proceso metodą modeliuojantis elementas.
	Atitinka `Parameter` klasės egzempliorių. Metodo parametrą modeliuojantis elementas.

4 PSM METAMODELIŲ SUDARYMAS

Antroje tyrimo stadijoje buvo sukurtos dvi platformom specifinės modeliavimo kalbos. Kadangi PIM modelis apibrėžia būsenas išsaugančias esybes tai buvo pasirinkta sukurti reliacinį PSM. Šio PSM modelis sąlyginai pavadintas SQL modeliu. Visos PIM išsaugančios būseną esybės ir jų sąryšiai transformuojami į reliacinį modelį. Taikomosiose realizacijose gautas reliacinis modelis būtų naudojamas DDL (*Data Definition Language*) resursam generuoti. Reliacinis metamodelis detalizuotas 4.1 skyriuje.

Kadangi transformacijų tikslas yra gauti detalizuotą PSM modelį iš kurio galima būtų vėliau gauti programinį kodą, tai kaip vykdymo platformą pasirinkta Java kalba. Java kalbai modeliuoti sukurtas Java PSM. Java metamodelio sandara detaliau aprašyta 4.2 skyriuje.

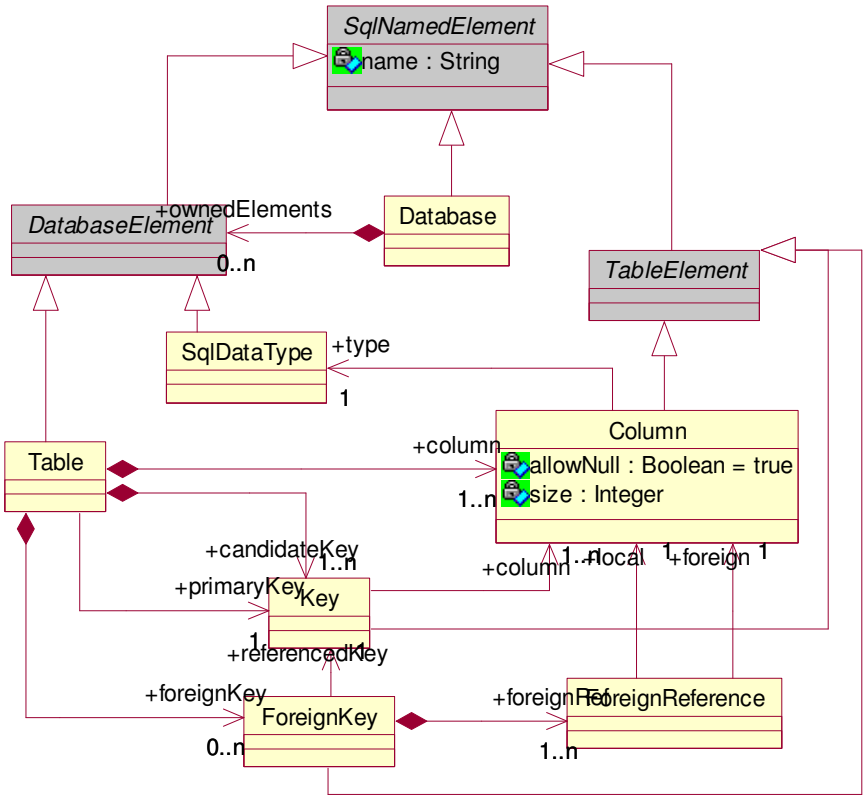
4.1 Reliacinis metamodelis

Kuriant reliacinio modeliavimo PSM kalbą, buvo sukurta reliacinio modeliavimo kalba, kuri suteikia galimybę modeliuoti duomenų bazių lenteles, duomenų tipus, lentelių stulpelius, išorinius ir pirminius raktus. Sukurtas reliacinis metamodelis modeliuoja esmines reliacinių duomenų bazių bruožus ir neskirtas jokiai konkrečiai DBVS. Kuriant reliacinį tikslo metamodelį buvo apsibrėžti tokie reikalavimai:

1. Modeliavimo kalba turi suteikti galimybę apibrėžti esybes atitinkančias lenteles ir tų lentelių stulpelius;
2. Stulpeliams galima nurodyti netik tipą, bet to tipo dydį (pvz. VARCHAR(32));
3. Turi būti galimybė sukurti reikiamą duomenų tipą;
4. Kalba turi leisti nurodyti kurie lentelės stulpeliai yra pirminiai raktai;
5. Reliacinio modeliavimo kalba turi leisti nurodyti kurie lentelės stulpeliai yra išoriniai raktai ir kokius išorinės lentelės stulpelius jie atitinka;
6. Turi būti galimybė nurodyti kokį išorinės lentelės unikalų raktą atitinka pasirinktos lentelės išorinis raktas.

Bazinė visų reliacinio metamodelio elementų klasė – `SqlNamedElement`. Šita klasė turi vienintelį atributą `name`. `Database` klasės egzempliorius yra išorinis viso modelio egzemplioriaus konteineris. `Database` klasės egzemplioriaus sudėtinės dalys – tai lentelės `Table` ir paprasto duomenų tipo `SqlDataType` egzempliorių rinkiniai. `Table` klasės egzemplioriai modeliuose gali turėti stulpelio (`Column`), rakto (`Key`) ir išorinio rakto

(`Foreign`) elementus. Išorinio rakto klasės egzempliorius privalo turėti nuorodą į išorinio rakto objektą, taip pat `ForeignKey` klases egzempliorių sąrašą, kur kiekvienas objektas nurodo koks vietinis stulpelis surištas su atitinkamu išoriniu stulpeliu.



11 pav. -- Reliacinis PSM metamodelis

Buvo sukurtas reliacinio modeliavimo kalbos metamodelio klasės ir jų sąryšiai pavaizduoti 11 pav..

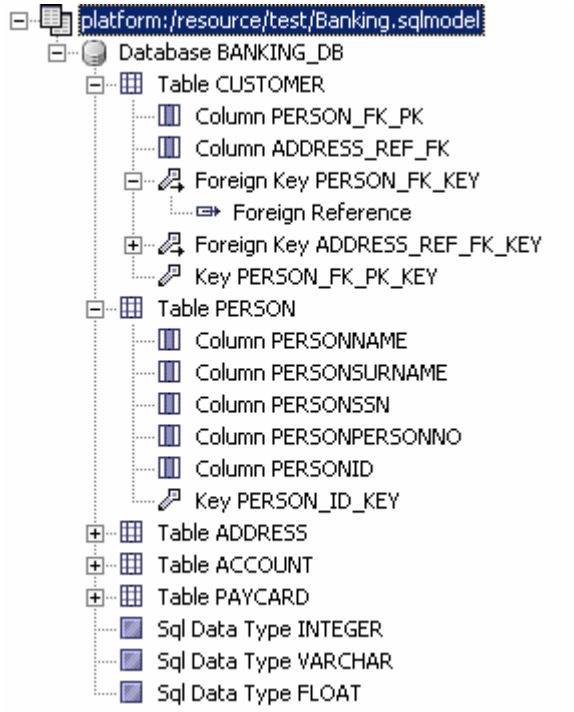
6 lent. – Reliacinio PSM metamodelio klasės

Metamodelio klasė	**Aprašymas**
Column	Stulpelį modeliuojanti metaklasė. Stulpelis turi priskirtą duomenų tipą, taip nurodomas duomenų tipo reikšmės dydis.
Database	Tai duomenų bazę modeliuojanti metaklasė.
DatabaseElement	Tai abstrakti metaklasė – negali būti sukurtas nė vienas jos egzempliorius.
ForeignKey	Išorinį raktą modeliuojanti metaklasė.

ForeignReference	Metaklasė, kurios egzempliorius apibrėžia koks vienos lentelės išorinio rakto stulpelis susijęs su kitos lentelės stulpeliu.
Key	Metaklasė, kurios egzempliorius nurodo, kokie lentelės stulpeliai yra tos lentelės raktai.
SqlDataType	Duomenų tipą modeliuojanti metaklasė.
SqlNamedElement	Abstrakti metaklasė – bazinė visų Reliacinio metamodelio elementų klasė.
Table	Lentelę modeliuojantis metamodelio elementas.
TableElement	Abstrakti lentelės sudedamąją dalį modeliuojanti metaklasė.

Aprašyto reliacinio metamodelio formali specifikacija XMI formate pateikta 10.2 priede. Metamodelio specifikacija buvo sukurta naudojant EMF aplinkos priemones.

Naudojant sudarytąjį metamodelį ir EMF aplinkos įrankius, buvo sukurtas reliacinio PSM modelio redaktorius. Redaktoriaus priemonėmis, galima sudaryti reliacinius modelius. Pavyzdinis reliacinio modelio egzempliorius redaktoriaus lange pavaizduotas 12 pav..



12 pav. -- Pavyzdinis reliacinio metamodelio egzempliorius

Mūsų tyrime, reliacinio modelio redaktorius bus naudojamas tik vaizdiniam modelių demonstravimui, nes reliacinį modelį iš PIM modelio sukurs transformacijos realizacija. Reliacinio modelio redaktorius atvaizduoja reliacinį modelį saugomą XMI formato byloje.

Kiekvienas atvaizduojamo modelio elementas parodomas kaip medžio viršūnė su tam tikra piktograma. Piktogramų aprašymai pateikti 7 lent..

7 lent. -- Reliacinio PSM modelio piktogramos ir jų reikšmės

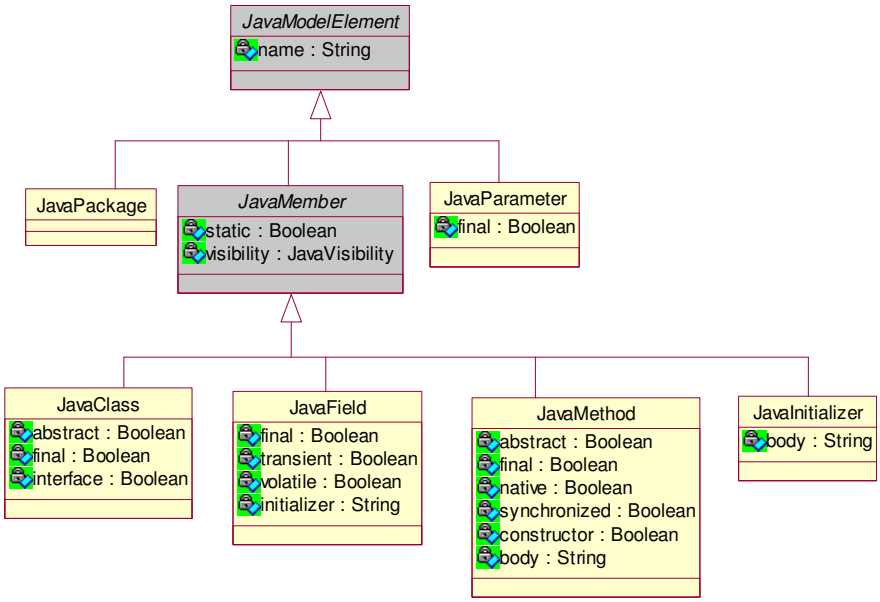
Piktograma	**Paaiškinimas**
	Duomenų bazės modelio elementas. Atitinka `Database` klasės egzempliorių.
	Primityvaus duomenų tipo elementas. Atitinka SqlDataType klasės egzempliorių.
	Lentelės tipo elementas. Atitinka `Table` klasės egzempliorių.
	Stulpelio tipo elementas. Atitinka `Column` klasės egzempliorių.
	Rakto tipo objektas. Atitinka `Key` klasės egzempliorių.
	Išorinio rakto tipo objektas. Atitinka `ForeignKey` klasės egzempliorių.
	Saito tarp lokalaus stulpelio ir išorinio rakto stulpelio egzempliorius. Atitinka `ForeignReference` klasės egzempliorių.

4.2 Java kalbos metamodelis

Tyrimo metu sudarytas Java kalbos metamodelis. Sudarant metamodelį buvo remiamasi Java kalbos specifikacija [11]. Modeliui nustatyti tokie bendrieji reikalavimai:

1. Java kalbos PSM turi modeliuoti Java kalbos struktūrą [11, 13p.];
2. Turi būti galimybė modeliuoti Java kalbos duomenų tipus [11, 31p.];
3. Galimybė modeliuoti Java klases ir sąsajas [11, 135p., 199p.];
4. Turi būti galimybė modeliuoti klasių išplėtimo (*extends*) ir realizacijos (*implements*) sąryšius [11, 142-144p.];
5. Java modelio klasės gali tūrėti sudėtinius elementus [11, 140p.]: atributus, metodus ir vidines klases;
6. Turi būti galimybė modeliuoti klasių, laukų ir metodų modifikatorius (*modifiers*) ir jų pagrindines savybes.

Sudarytojo Java PSM metamodelio paveldėjimo hierarchija pateiktas diagramoje 13 pav., o metamodelio elementų sąryšiai atvaizduoti 14 pav..



13 pav. -- Java kalbos PSM metamodelio paveldėjimo hierarchija

Bazinis visų Java kalbos metamodelio elementas – tai `JavaModelElement` klasė. Šita klasė turi vienintelį atributą `name`, kurį paveldi visi kiti metamodelio elementai. `JavaApplication` klasė – tai išorinis viso metamodelio egzemplioriaus konteineris, kurio sudėtinės dalys gali būti tik `JavaPackage` egzemplioriai. `JavaPackage` klasė atitinka Java programavimo kalbos paketą.

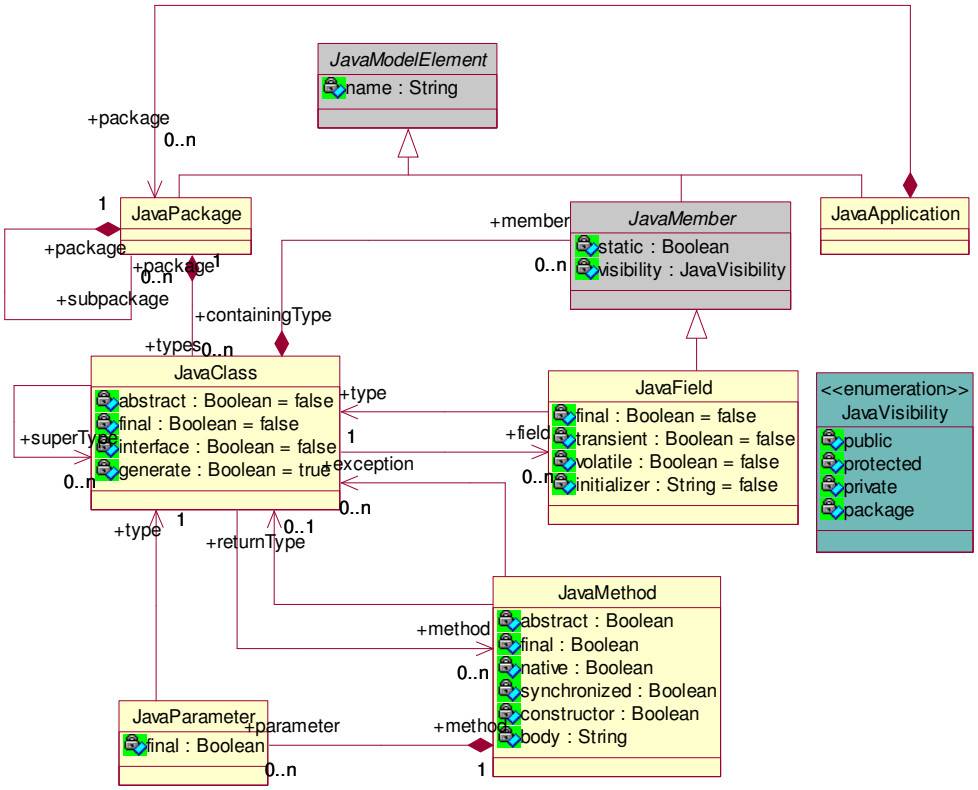
`JavaMember` klasės egzemplioriai modeliuoja sudėtinius Java kalbos klasės elementus: vidines klases, metodus ir laukus. Vidines ir paprastas Java kalbos klases atitinka `JavaClass` modelio elementas. Šitam modelio elementui apibrėžtos papildomos savybės: abstraktumas (laukas `abstract`), paveldėjimo draudimas (laukas `final`) ir laukas nusakantis ar klasė yra sąsaja.

Vienas iš galimų `JavaClass` elementų `JavaField`, turi papildomus, Java kalbos sintaksės elementus, atitinkančius laukus: `final`, `transient`, `volatile` ir inicializatoriaus išraišką.

Kitas galimas Java klasės elementas – metodas (JavaMethod) apibrėžia tokius papildomus atributus:

1. `final` – šitas atributas nurodo ar metodas gali būti perrašytas paveldėjusioje klasėje;
2. `native` – nurodo ar metodas nėra išorinio dinaminės bibliotekos iškvietimas pagal C konvenciją;

3. `synchronized` – nurodo ar metodas gali būti vykdomas vienos ar kelių lygiagrečių vykdymo gijų;
4. `constructor` – laukas nurodo ar metodas yra klasės inicializatorius.



14 pav. -- Java kalbos PSM metamodelio elementų sąryšiai

Java kalbos metamodelio klasių aprašymai pateikti 8 lent..

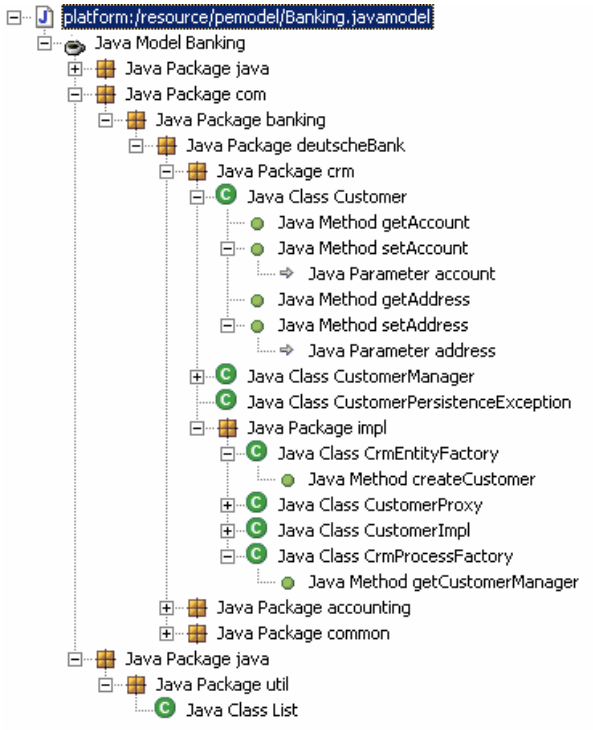
8 lent. -- Java PSM metamodelio klasės

Metamodelio klasė	**Aprašymas**
JavaApplication	Java kalba parašyto programos kodo visumą modeliuojanti metaklasė.
JavaClass	Java duomenų tipą modeliuojanti klasė. Gali atitikti paprastąją ar abstrakčiąją klasę, sąsają arba duomenų tipą.
JavaField	Java klasės atributą modeliuojanti metaklasė.
JavaInitializer	Lauko inicializatorių modeliuojanti klasė.
JavaMember	Klasės sudėtinį komponentą modeliuojanti abstrakti metaklasė.
JavaMethod	Java funkciją modeliuojanti metaklasė.

JavaModelElement	Abstrakti bazinė metaklasė.
JavaPackage	Java kalbos paketą modeliuojanti metaklasė.
JavaParameter	Java funkcijos parametrą modeliuojanti metaklasė.
JavaVisibility	Java klasių, sąsajų, metodų, atributų priėjimo (*access*) modifikatorių sąrašas.

Su formalia Java kalbos metamodelio specifikacija XMI formate galima susipažinti 10.3 priede.

Naudojant EMF programavimo aplinkos įrankius, sukurtas Java modelių redaktorius. Pavyzdinio Java modelio egzempliorius modelio redaktoriaus lange – pateiktas 15 pav.. Sukurtąja Java PSM modeliavimo kalba galima modeliuoti pagrindinius Java kalbos elementus bei jų ryšius. Java PSM metamodelio egzempliorius gali būti naudojamas vėlesnei Java kalbos kodo generacijai.



15 pav. -- Pavyzdinis Java kalbos PSM metamodelio egzempliorius

Sukurto Java modelio redaktoriaus piktogramų reikšmės surašytos 9 lent..

9 lent. -- Java kalbos modelio redaktoriaus piktogramos

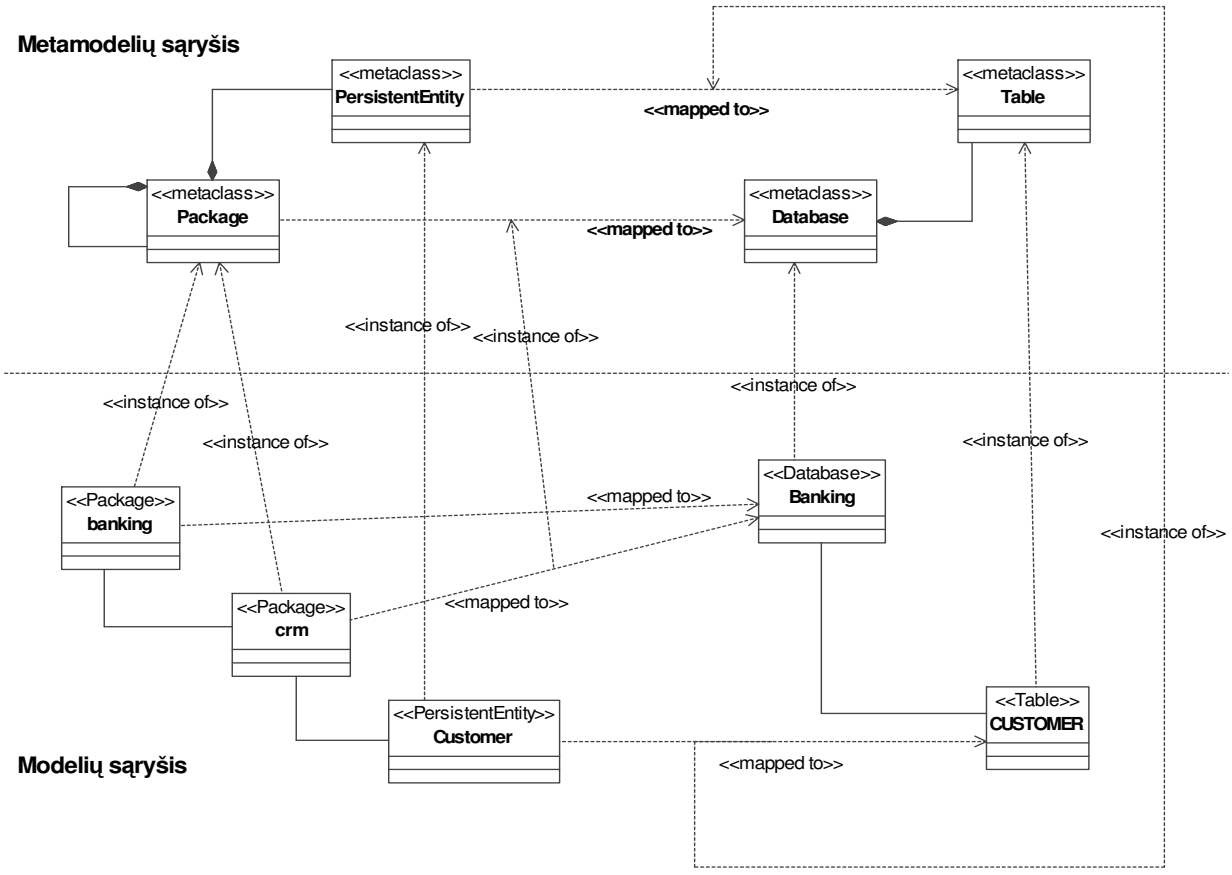
Piktograma	**Paaiškinimas**
	Išorinis visų modelio elementų konteineris. Atitinka `JavaApplication` klasės egzempliorių.
	Java kalbos paketo sąvoką atitinkantis modelio elementas. Atitinka `JavaPackage` klasės egzempliorių.
	Java kalbos klasę modeliuojantis elementas. Atitinka `JavaClass` klasės egzempliorių..
	Klasės atributą modeliuojantis objektas. Atitinka `JavaField` egzempliorių.
	Atitinka `JavaMethod` klasės egzempliorių. Modeliuoja Java kalbos klasę.
	Metodo parametro elementas.

5 PIM – PSM TRANSFORMACIJŲ REALIZAVIMAS

Šitame skyriuje analizuojamas PIM modelio transformavimo į sukurtus PSM modelius procesas. Aprašoma pasirinkta transformacijų realizavimo architektūra.

Turėdami PIM ir PSM metamodelius, mes galime nusistatyti sąryšius tarp transformuojamų modelių metaklasių. Kadangi 2.2 skyriuje mes pasirinkome modelių tipų susiejimą – tai mes galime aprašyti modelių elementų sąryšius metamodelio lygyje. Tokiu būdu mes galime aprašyti atskirų PSM modelio elementų transformavimo į PIM modelio elementus taisykles, tačiau aprašydami manipuliuosime metamodelio terminais . 16 pav. pateikta diagrama parodanti kaip modelių sąryšis aprašomas modelio ir metamodelio lygmenyje.

Kita modelių transformavimo problema – tai transformacijos dekompozicija. Kad transformacija būtų lengva realizuoti ir modifikuoti atskiri transformavimo etapai, taip pat skirtingo tipo modelio elementų transformavimas turi būti atskirti.



16 pav. -- Modelių sąryšiai aprašyti modelio ir metamodelio lygmenyje

Tyrimo metu nustatyti keli transformavimo proceso dekompozicijos variantai:

1. Modelio apėjimas pagal kompozicijos ryšius su „atidėjimu“. Modelio elementų medis apeinamas „į plotį“ pagal kompozicijos ryšius apibrėžtus metamodelyje. Kiekvienas modelio elementas apdorojamas atskirai. Jeigu modelio elementas dar negali būti apdorotas – jis talpinamas į laukimo eilę ir tokiu būdu atidedamas jo apdorojimas. Po apdorojimo elementai išimami iš apdorojimo eilės. Pastebėta, kad algoritmui su atidėjimu reikalingos dvi transformavimo stadijos:
    a. Pagrindinė transformavimo stadija – jos metu kiekvienas modelio elementas transformuojamas į atitinkamą kito modelio elementą. Stadijos trukmė – vienas pilnas modelio apėjimas. Jos metu registruojami išeities ir tikslo modelių elementų transformavimo sąryšiai, taip pat išeities modelio elementų ryšiai talpinami į antrinio apdorojimo eilę ir bus apdoroti antrinėje stadijoje.
    b. Antrinė transformavimo stadija. Antrinės stadijos metu sukuriami ryšiai tarp tikslo modelio elementų iš ryšių patalpintų į antrinio apdorojimo eilę.
2. Modelio apėjimas pagal kompozijos ryšius su apdorojimu pagal poreikį (*on demand*). Modelyje kiekvienas elementas apdorojamas atskirai. Jeigu modelio elementui apdorojimas priklauso nuo kito elemento apdorojimo – tai iškviečiamas reikiamo elemento apdorojimas. Apdoroti elementai registruojami ir apėjimo metu jie praleidžiami – nes jau buvo apdoroti. Apdorojimo ciklo trukmė – vienas modelio apėjimas.
3. Laisvas modelio elementų apėjimas panaudojant filtrų architektūrinį šabloną (*Pipes and Filters Architecture*) [12, 427p.]. Realizuojant transformaciją taikant filtrų architektūrą transformacijos stadijos buvo dekomponuotos į komponentus vadinamus filtrais. Kiekvienas toks filtras realizuoja vieno tipo elementų transformaciją. Kiekvienas filtras atitinka tam tikrą transformavimo stadiją. Apdorojimo metu registruojami tiesioginiai ir atvirkštiniai transformavimo ryšiai tarp išeities ir tikslo modelių. Vieno transformavimo ciklo trukmė: $\tau = N \cdot S$, kur N – modelio elementų skaičius, S – transformavimo stadijų skaičius.

Transformacijų realizavimui buvo pasirinkta filtrų architektūra, kuri leido žymiai paprasčiau atskirti modelio transformavimo stadijas. Kiekvienas filtras realizuoja modelio elementui taikomą sąlygą, jeigu ta sąlyga tenkinama – tai vykdomas elemento transformavimas. Reliacinio modelio transformavimo taisyklės pateiktos 5.1 šio darbo skyriuje. 5.2 skyriuje

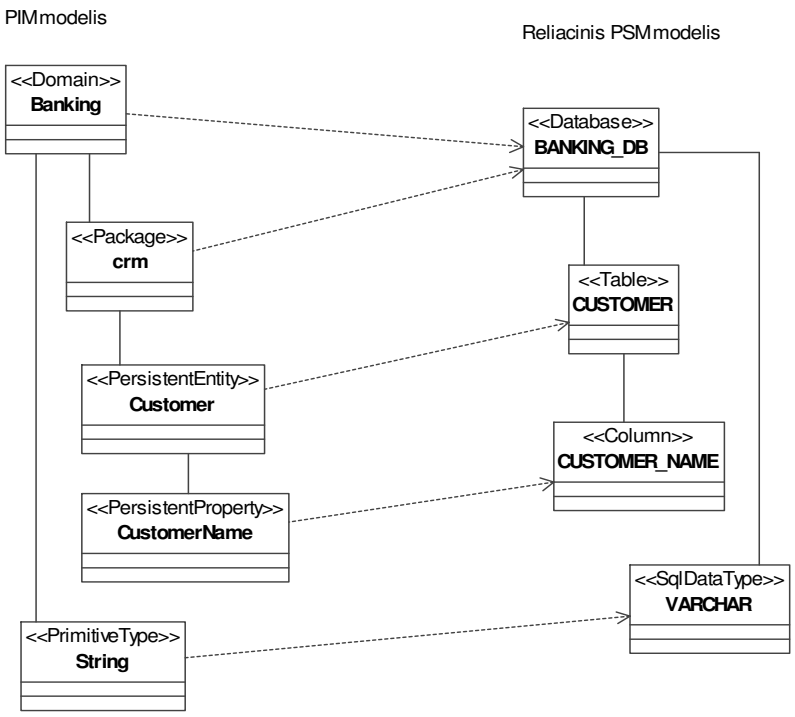
5.1 „Esybė – procesas" modelio transformavimas į reliacinį modelį

Realizuojant „esybė – procesas" PIM modelio transformavimą į PSM reliacinį modelį buvo apsibrėžtos ir realizuotos 10 lent. pateiktos transformavimo taisyklės. Transformacijų taisyklės programiškai realizuojamos filtruose.

10 lent. -- PIM transformavimo į reliacinį modelį taisyklės (pavyzdys pateiktas 17 pav.)

Transformacijos sąlyga	**Transformacijos taisyklė**
Tipas `Domain`	Sukurti `Database` tipo elementą.
Tipas `Package`	Susieti su `Database` tipo elementu
Tipas `PersistentEntity`	Sukurti `Table` tipo elementą ir susieti su `Database` tipo elementu.
Tipas `PersistentProperty`	Sukurti `Column` tipo elementą ir susieti su atitinkamu `Table` elementu
Tipas `Identifier` ir savininkas nėra paveldėtas	Sukurti `Key` elementą ir susieti su `Column` elementais, kurie buvo transformuoti iš `PersistentProperty` elementų, kurie yra susieti su `Indentifier` elementu.
Tipas PersistentEntity ir yra paveldėjimo ryšys su kitu `PersistentEntity`	Sukurti ForeignKey elementą Table elemente gautame iš dabar apdorojamo `PersistentEntity`. Tėvą atitinkančio `Table` elemento raktas susiejamas su išoriniu raktu vaiko klasę atitinkančiame `Table` elemente.
Tipas `Reference` ir kardinalumas 1:1	Sukurti `ForeignKey` elementą `Table` elemente gautame iš dabar apdorojamo `PersistentEntity`. Kaimyną atitinkančio `Table` elemento raktas susiejamas su išoriniu raktu vaiko klasę atitinkančiame Table elemente.
Tipas `Reference` ir kardinalumas 1:N	Sukurti `ForeignKey` elementą Table elemente atitinkančiame dabar apdorojamo `PersistentEntity` kaimyną. Dabar apdorojamo `PersistentEntity` atitinkančio `Table` elemento raktas susiejamas su išoriniu raktu kaimyno klasę atitinkančiame `Table` elemente.

Vykdant PIM modelio transformavimą į reliacinį PSM modelį – būseną išsaugančios esybės transformuojamos į atitinkamas lenteles, o jų sąryšiai realizuojami per išorinius raktus.



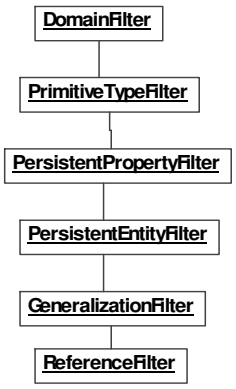
17 pav. -- PIM modelio susiejimas su reliaciniu PSM modeliu

Gautame po transformacijos reliaciniame modelyje, PIM modelio būseną neišsaugančios esybės Transformuojant PIM į reliacinį PSM panaudota filtrų architektūra. Filtrų sujungimo grandinė pavaizduota 18 pav.. Žemiau pateikiama lentelė su filtrų aprašymais.

11 lent. -- PIM į reliacinį PSM modelį transformuojantys filtrai

Filtras	**Aprašymas**
DomainFilter	Transformuoja Domain elementą į Database.
GeneralizationFilter	Transformuoja paveldėjimo ryšius į ryšius per išorinį raktą.
PersistentEntityFilter	Transformuoja PersistentEntity į Table elementus.
PersistentPropertyFilter	Transformuoja PersistentEntity attributus į Column elementus.
PrimityveTypeFilter	Transformuoja primityvius PIM duomenų tipus į reliacinio modelio duomenų tipus.
ReferenceFilter	Transformuoja 1:1 ir 1:N ryšius į reliacinius ryšius per išorinius raktus.

18 pav. pavaizduota filtrų aktyvavimosi grandinė. Kiekvienas filtras naudoja jam tiekiamus duomenis ir įvykdo tam tikrą dalį transformacijos proceso.



18 pav. -- PIM į reliacinį PSM transformuojančių filtrų struktūra (duomenys pereina nuo viršaus į apačią)

Matome, kad transformacijos į reliacinį modelį atveju, filtrų grandinėje nėra išsišakojimų. Kiekviena sekanti stadija vykdoma po to kai baigta ankstesnė.

Eksperimentiniai transformacijos vykdymai pateikiami 6 skyriuje.

5.2 „Esybė – procesas" modelio transformavimas į Java kalbos modelį

Esybė – procesas PIM transformacijos į Java PSM realizavimui taip pat buvo taikoma filtrų architektūrinis šablonas, tačiau filtrų išsidėstymas ir jų realizacijos skiriasi. Tyrimo tikslais nuspręsta padidinti transformacijos detalumą ir pritaikyti Proxy šabloną [13, 207p.] visiems `Entity` ir `Process` tipo elementams. Transformavimo taisyklės pateikiamos 12 lent..

12 lent. -- PIM į Java PSM transformacijos taisyklės

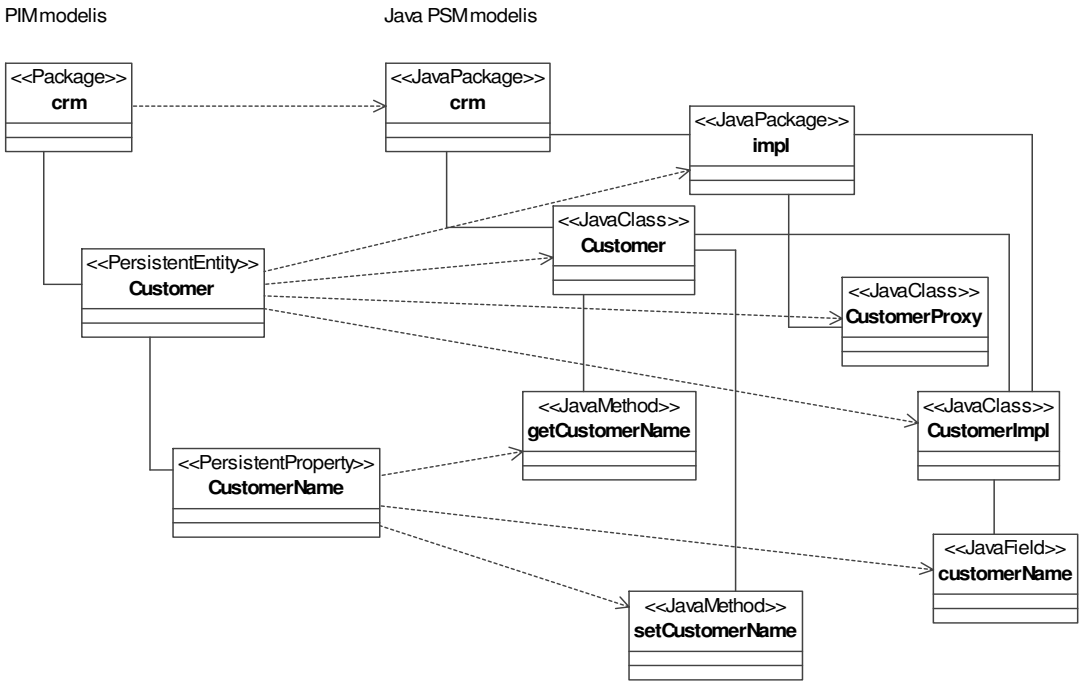
Transformacijos sąlyga	**Transformacijos taisyklė**
Tipas `Domain`	Sukuriamas JavaApplication elementas (pvz. 19 pav.).
Tipas `Package`	Sukuriamas JavaPackage elementas.
Tipas `Reference`	Sukuriamas JavaField elementas pagal sąryšį atitinkamame JavaClass elemente.
Tipas `PersistentEntity`	Transformuoja į JavaClass elementą atitinkantį sąsają, klasę su vardo priesaga „Impl" ir klasę su vardo priesaga „Proxy" sukuria JavaPackage elementą su vardu „impl" ir sujungia elementus pagal Proxy šabloną (pvz. 20 pav.).
Tipas `SimpleEntity`	
Tipas `SimpleProperty`	Transformuoja į JavaMethod elementus vieną su priedaga „set", kita – su priesaga „get", priskiria atitinkamo tipo parametrus ir gražinamus tipus.
Tipas `PersistentProperty`	

Tipas `Process`	Transformuojamas į JavaClass elementų rinkinį pagal Proxy šabloną (pvz. 21 pav.).

Kadangi `Domain` tipo elementas yra tik išorinis modelio elementų konteineris, tai jis transformuojamas į Java kalbos modelio analogišką modelių elementų konteinerį `JavaApplication`.

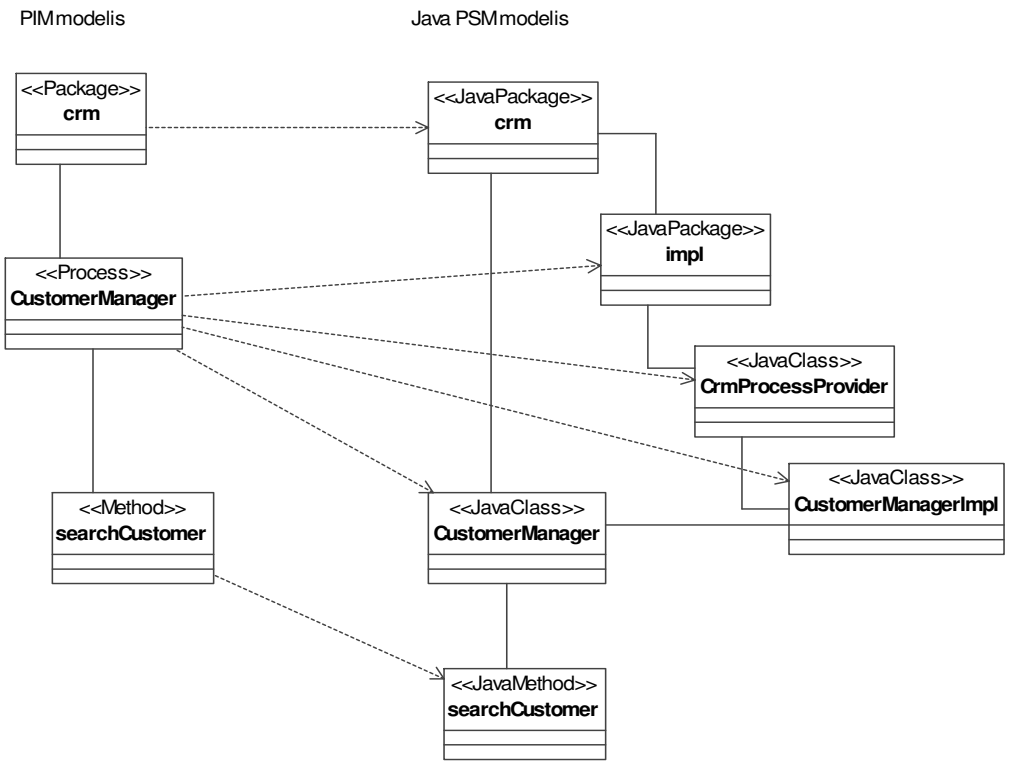
19 pav. -- PIM modelio `Domain` tipo elemento transformavimas į `JavaApplication` tipo elementą

PIM esybės transformavimas į Java modelio elementus daug sudėtingesnis. PIM esybės atitikmuo Java modelyje bus klasių ir sąsajų kompleksas atitinkantis *Proxy* šabloną. Kiekviena esybė transformuojama į sąsajos klasė, realizacijos klasę atstovaujanti klasė (*proxy class*) ir gamyklos klasė (*factory class*). Dalinis transformacijos rezultatas schematiškai atvaizduotas 20 pav..



20 pav. -- PIM esybės transformavimas į Java PSM klases pritaikant Proxy šabloną

PIM modelio proceso elementui transformuojant taip pat pritaikomas *Proxy* šablonas. Taigi procesas išskaidomas Java modelyje į sąsajos, realizacijos atstovavimo ir gamyklos klasę.



21 pav. -- PIM proceso transformavimas į Java PSM klases pritaikant Proxy šabloną

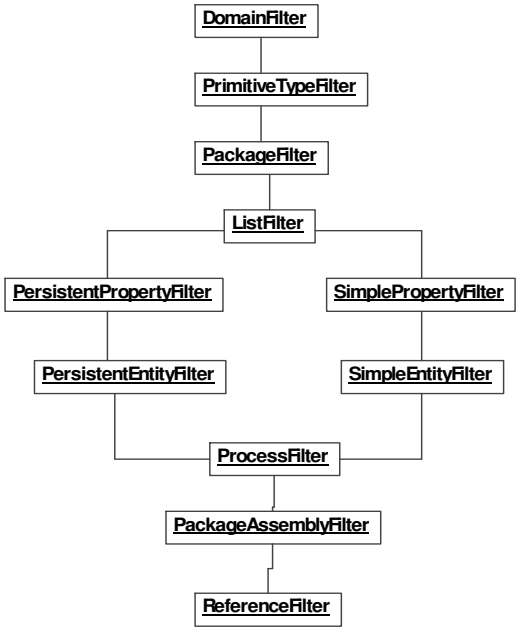
Transformacijai realizuoti panaudota filtrų struktūra pavaizduota 22 pav.. Filtrų aprašymai pateikti 13 lent..

13 lent. -- PIM į Java PSM filtrai

Filtras	**Aprašymas**
DomainFilter	Transformuoja `Domain` elementą į `JavaApplication` elementą.
ListFilter	Transformuoja PIM List duomenų tipus į java.util.List duomenų tipą.
PackageAssemblyFilter	Paketų surinkimo filtras. Sudeda paketus pagal kompozicijos ryšius tarp atitinkamų išeities modelio elementų.
PackageFilter	Paketų filtras. Transformuoja `Package` į `JavaPackage`.
PersistentEntityFilter	Transformuoja `PersistentEntity` į `JavaClass` su atitinkamu pavadinimu.
PersistentPropertyFilter	Transformuoja `PersistentProperty` į

	`JavaField` elementą.
PrimitiveTypeFilter	Transformuoja `PrimitieType` elementus į atitinkamus `JavaClass` elementus, atitinkančius standartinius Java kalbos duomenų tipus.
ProcessFilter	`Process` tipo elementų filtras.
ReferenceFilter	Sudaro ryšius tikslo modelyje pagal išeities modelio elementų ryšius.
SimpleEntityFilter	Transformuoja `SimpeEntity` elementus į `JavaClass` elementų rinkinį.
SimplePropertyFilter	Transformuoja `SimpeProperty` elementus į `JavaField` elementus.

PIM transformavimo į Java modelį filtrų struktūra sudėtingesnė negu transformacijos į reliacinį modelį. Pradiniai filtrai gali būti vykdomi tik tam tikra eilės tvarka, nes toliau grandinėje veikiantys filtras naudoja prieš tai einančių filtrų rezultatus.



22 pav. -- PIM modelio filtravimo stadijos transformuojant į Java PSM (duomenys pereina nuo viršaus į apačią)

Kiekvienas filtras savo veikimo rezultatą užregistruoja registre. Registras – tai sąrašas kuriame surenkami visų PIM elementų atvaizdavimai į PSM elementus. Naudojant tokį registrą, kiekvienas filtras gali gauti prieš tai buvusių filtrų rezultatus.

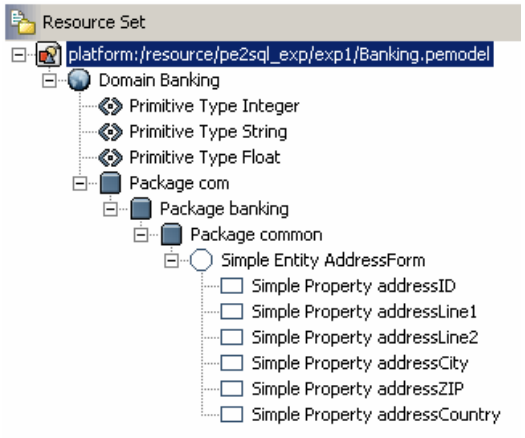
Transformacijų realizacijos eksperimentinio vykdymo rezultatai pateikiami 6 skyriuje.

6 EKSPERIMENTINIS TRANSFORMACIJŲ VYKDYMAS

Šeštame skyriuje pateikiami eksperimentinių transformacijų pradinių duomenų ir rezultatų grafiniai atvaizdavimai. Dėl didelių gaunamų PSM modelių apimčių, modelių medžiai suskleisti, išskleistos tik eksperimento esmę ir naują rezultatą parodančios dalys.

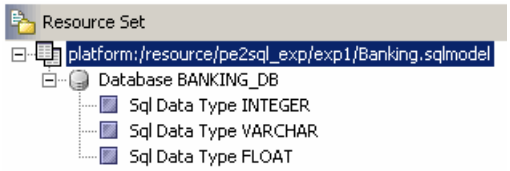
6.1 Būsenos neišsaugančių esybių transformavimas

Pirmasis eksperimentas demonstruoja, kaip realizuotoji transformacija detalizuoja būsenos neišsaugančią esybę. Kadangi būsenos neišsaugančios esybės neturi prasmės reliaciniam modeliui – atitinkamą metaklasę (`SimleEntity`) turinčių elementų filtras nebuvo realizuotas.



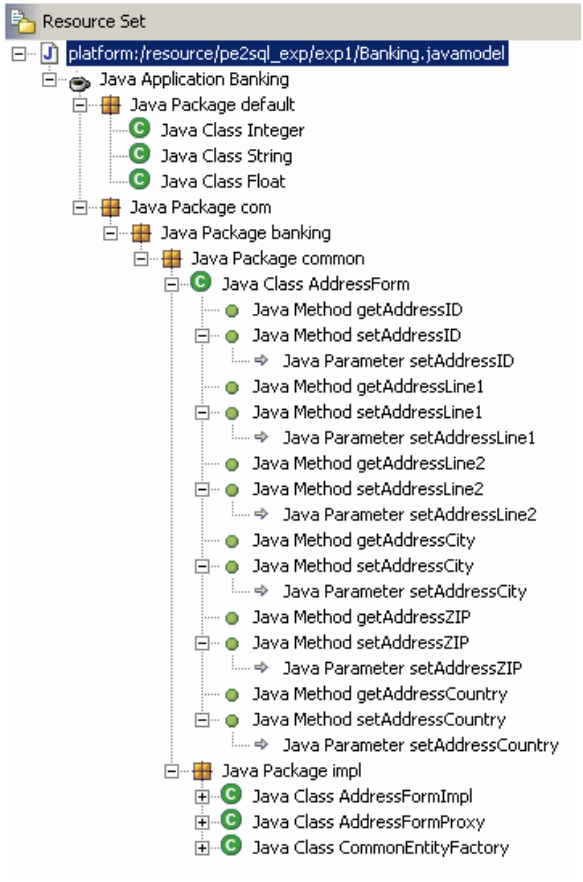
23 pav. -- Išeities PIM modelio egzempliorius

Matome, kad iš PIM modelio pavaizduoto 23 pav. Į reliacinį modelį buvo atvaizduoti tik primityvūs duomenų tipai (žr. 24 pav.).

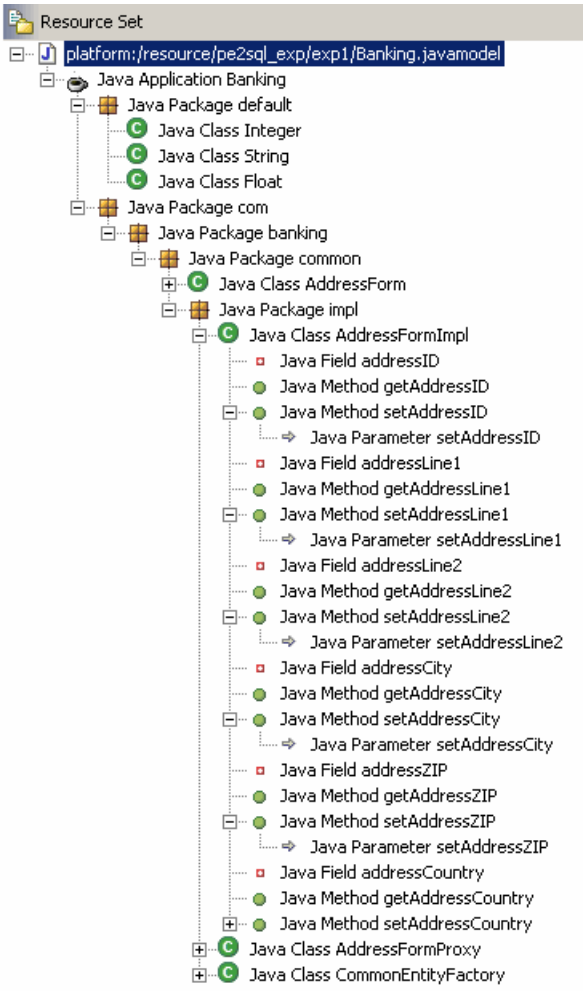


24 pav. -- Gautas reliacinis PSM modelio egzempliorius

Visiškai kitokį rezultatą matome Java PSM modelyje: eksperimentinės transformacijos metu transformuojamiems objektams pritaikoma *Proxy* šablonas. Taikant Proxy šabloną kiekvienai esybei sukuriamos sąsajos, realizacijos, realizacijos atstovavimo ir gamyklos klasės (žr.). Gaunamo Java PSM apimtis žymiai didesnė negu išeities modelio.



25 pav. -- Esybę atitinkančios sąsajos klasės sandara

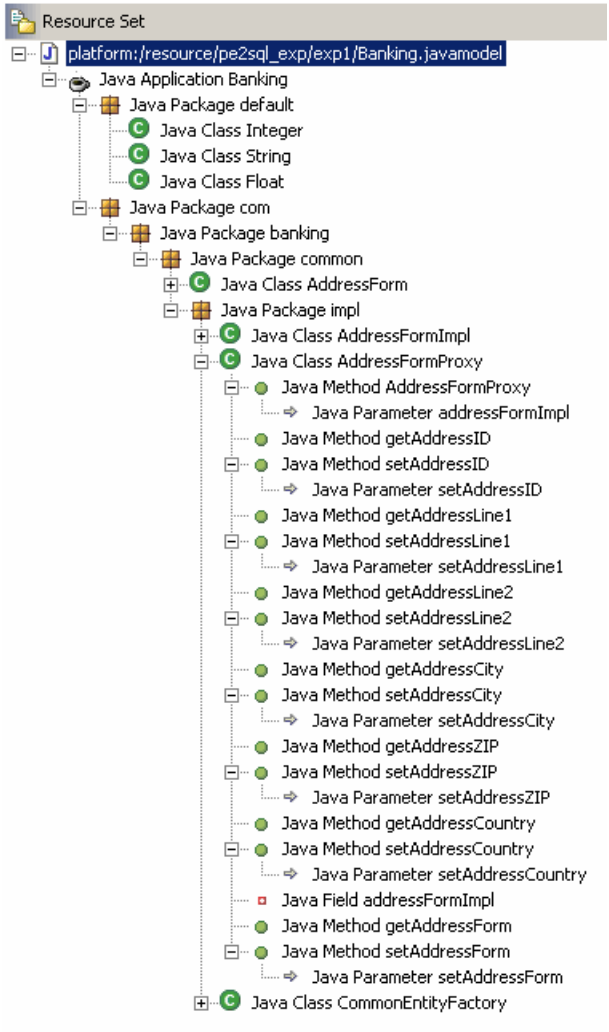


26 pav. -- Esybę atitinkančios realizacijos klasės sandara

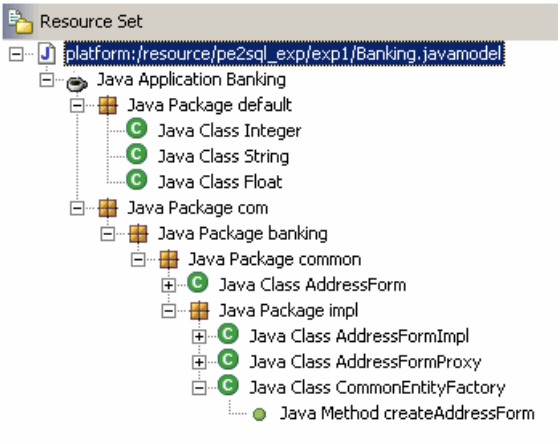
Kaip matome iš išskleistos sąsajos grafinio atvaizdavimo, kiekvienam esybės atributui buvo sukurti priėjimo metodai, kurių realizacijos kontroliuos esybės atributų reikšmių gavima ir keitimą. Esybės atributai transformuoti į atitinkamo pavadinimo laukus realizacijos klasę atitinkančiame elemente (žr. 26 pav.).

Realizaciją atstovaujančios klasės medis išskleistas 27 pav.. Kadangi atstovaujančios klasės pagrindinis tikslas perdavinėti iškvietimus realizacijos metodams, tai ji turi vieną privalomą lauką – nurodą į realizaciją. Taip pat atstovaujanti klasė turi konstruktoriaus funkciją, kurios argumentas – atstovaujama realizacija.

28 pav. pavaizduota gamyklos klasė sugeneruota transformacijos metu. Šitos klasės pagrindinė funkcija – atstovaujančių ir realizacijos klasių kūrimas ir gražinimas kaip nuoroda į sąsaja.



27 pav. -- Sąsajos realizaciją atstovaujanti klasė

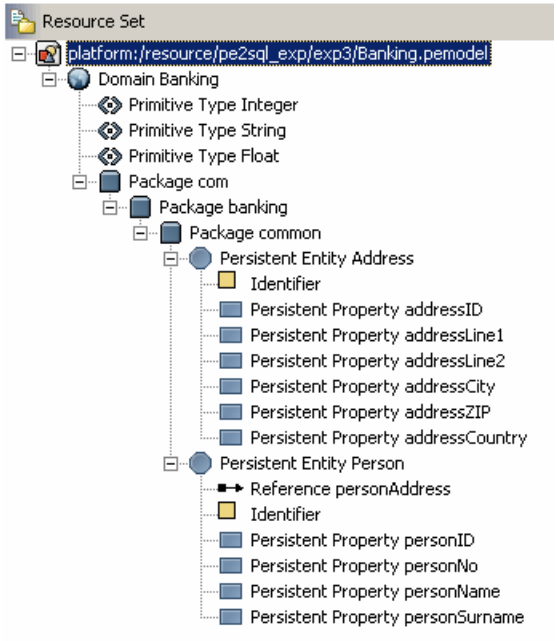


28 pav. -- Gamyklos klasės atvaizdavimas

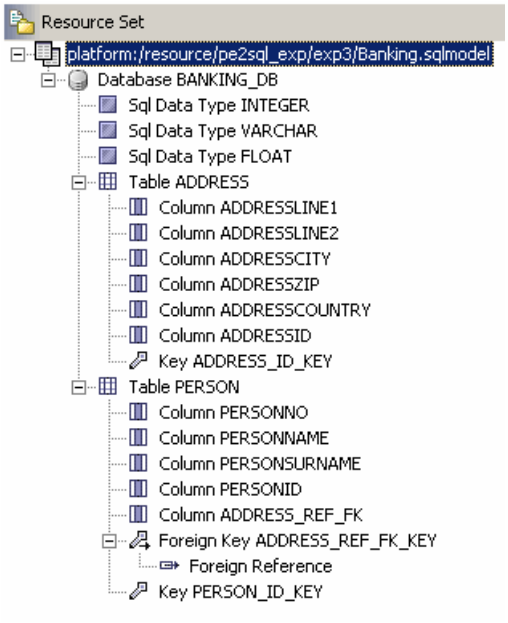
6.2 Būseną išsaugančių esybių su vienpusiu sąryšiu transformavimas

Antrasis eksperimentas skirtas pademonstruoti būseną išsaugančių esybių transformavimą į reliacinį modelį bei nuorodų transformavimas.

Kaip matome visos esybės specifikuotos išeities PIM modelyje 29 pav. buvo transformuotos į atitinkamas lenteles PSM modelyje 30 pav..



29 pav. -- Išeities PIM modelis

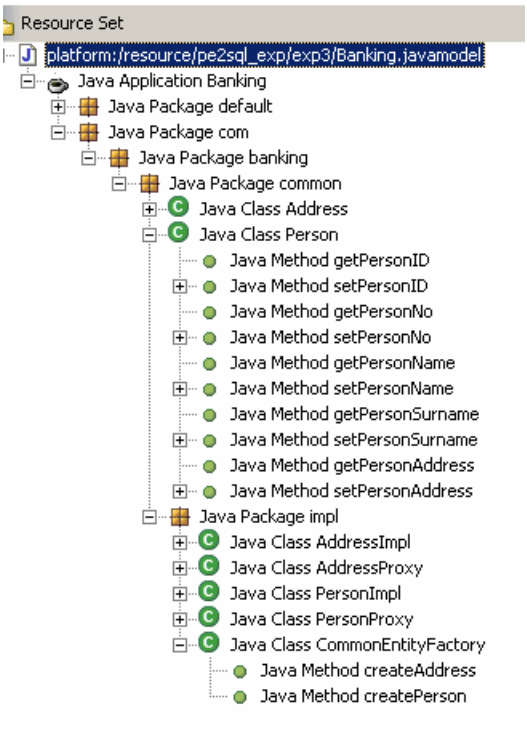


30 pav. -- Gautas reliacinis PSM modelis

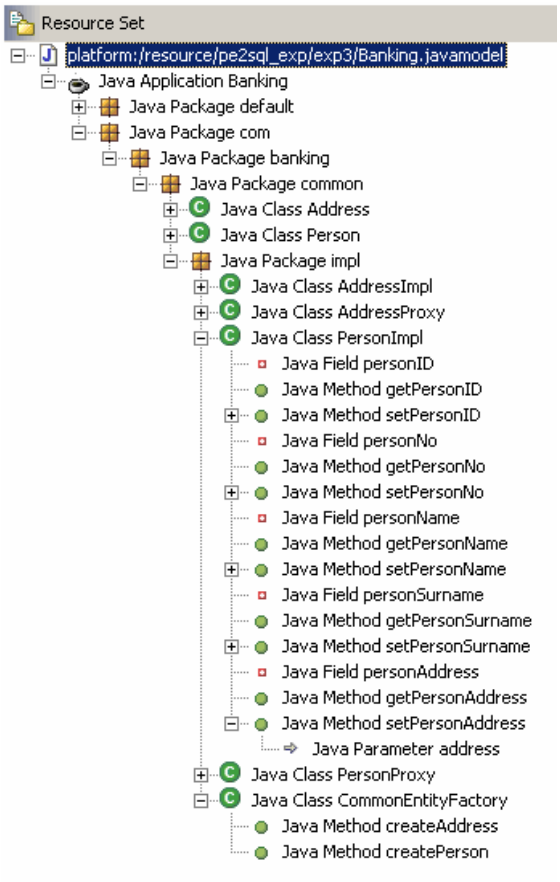
Išeities modelio stulpeliai, kurie PIM modelyje buvo nurodyti kaip identifikatoriai – reliaciniame modelyje tampa pirminiais unikaliais raktais.

Galima pastebėti, kad išeities modelio 1:1 kardinalumo nuoroda (`Reference` elementas medyje), rezultato modelyje transformavosi į išorinio rakto elementą.

31 pav., 32 pav. matome, kad transformacija elgiasi kitaip – nuoroda tampa realizacijos klasės lauku, kurio tipas yra nuorodos rodoma klasė. Esybėms taip pat pastebimas (žr. 33 pav.) *Proxy* šablono taikymas (analogiškai kaip aprašyta 6.1 skyriuje).



31 pav. -- Sugeneruota esybės `Person` sąsaja



32 pav. -- Išskleista `Person` esybės realizacijas

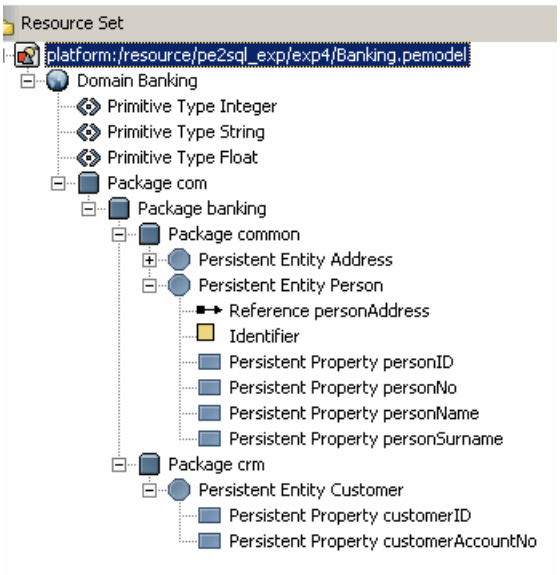
33 pav. -- Realizaciją atstovaujanti klasė

6.3 Būseną išsaugančių esybių su paveldėjimo sąryšiu transformavimas

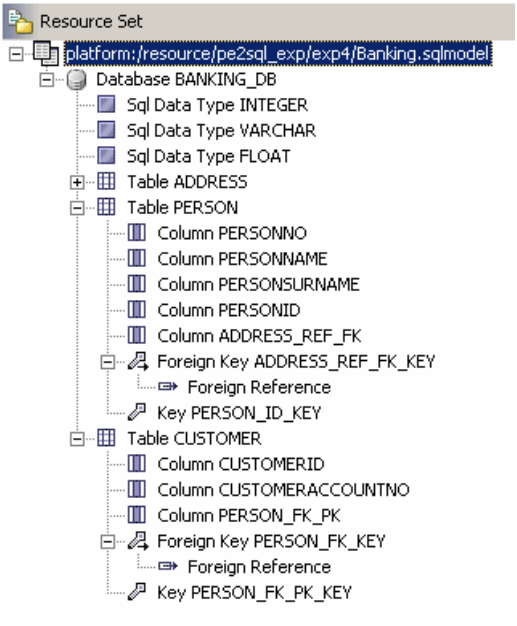
Trečio bandymo tikslas buvo pademonstruoti PIM paveldėjimo ryšio transformaciją į reliacinį ir Java modelį.

Reliaciniame modelyje matome sudėtingesnius pasikeitimus negu prieš tai buvusiuose bandymuose. Išeities modelyje `Customer` esybė paveldima iš `Person` esybės (žr. 34 pav.), todėl nustatomas ryšys iš CUSTOMER į PERSON per išorinį raktą PERSON_FK_KEY, kuris suriša lokalaus unikalaus rakto stulpelio reikšmę su tėvo unikalaus stulpelio-rakto reikšme (žr. 35 pav.).

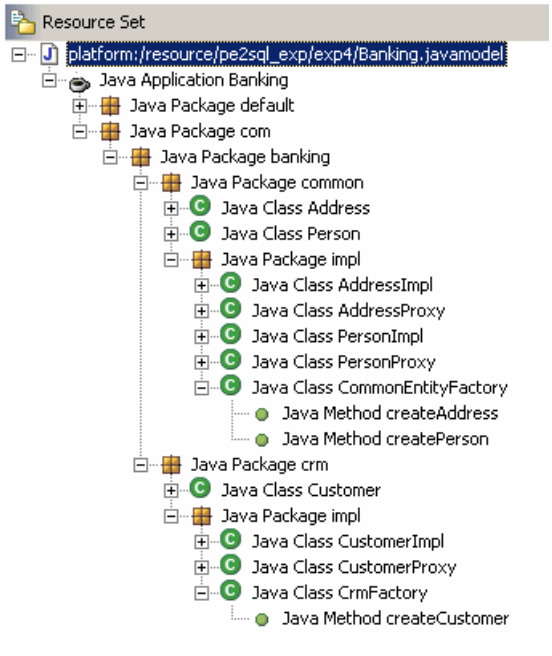
Gautame Java kalbos modelyje, taip pat kaip ir visur galime stebėti ta patį Proxy šablono taikymą, tačiau `Person` ir `Customer` šablonų Java klasės ir sąsajos susiję paveldėjimo ryšiais.



34 pav. -- Išeities PIM modelis paveldėjimo transformacijai stebėti



35 pav. -- Reliacinis PSM gautas trečiojo bandymo metu

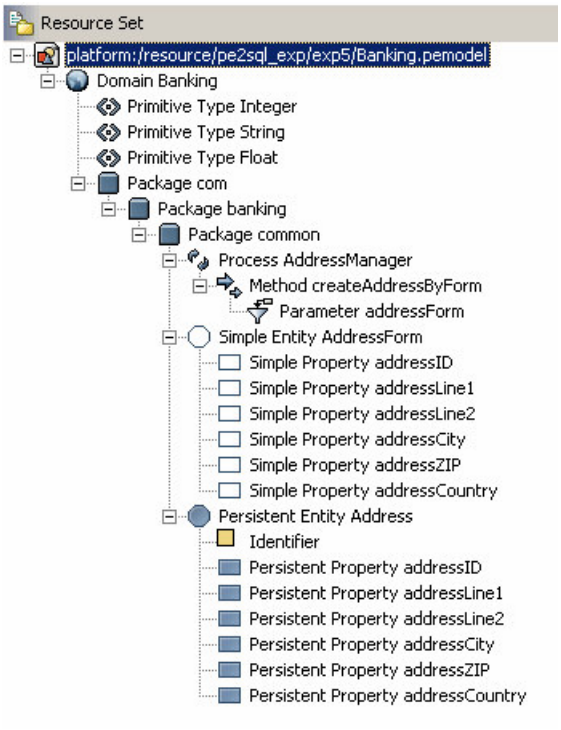


36 pav. -- Java PSM gautas trečiajame bandyme

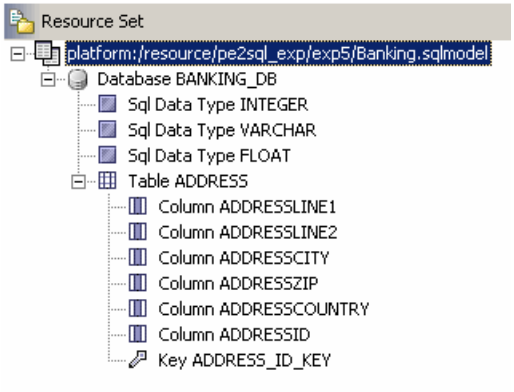
6.4 Procesų transformavimas

Ketvirto eksperimento metu buvo stebima proceso PIM modelio elemento, vienos išsaugančios būseną ir vienos neišsaugančios būsenos esybių.

37 pav. pateiktas išeities modelio grafinis atvaizdas. PIM modelyje pavaizduotas atvejis, kai modeliuojamas vienos esybės gavimas iš kitos naudojant proceso metodą.



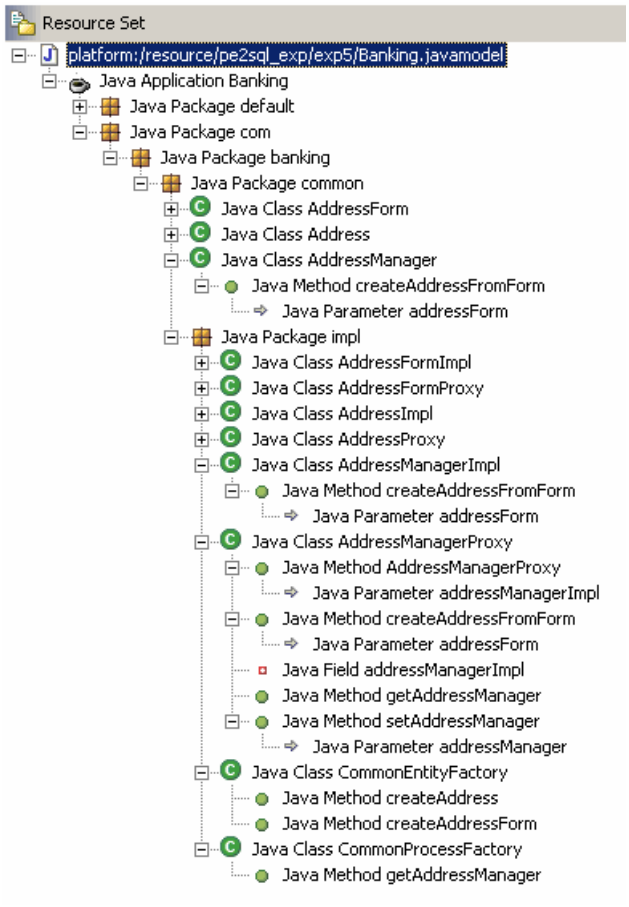
37 pav. -- PIM išeities modelis proceso ir esybių transformavimo bandymui



38 pav. -- Gautas reliacinis PSM

38 pav. Pateiktas gauto reliacinio modelio grafinis vaizdas, kuriame matome, vienos būseną išsaugančios esybės transformacijos rezultatą – lentelę ADDRESS su pirminiu raktu stulpelyje ADDRESSID.

PIM proceso objekto transformacijos kokybinis rezultatas parodytas 39 pav..



39 pav. -- Proceso PIM modelio elemento transformavimo rezultatas

6.5 Eksperimentų kiekybiniai stebėjimų rezultatai

Atlikus bandymus buvo stebimi kokybiniai rezultatai (grafinis jų atvaizdavimas), kurie pateikti ankstesniuose skyriuose. Atsižvelgus į tai, kad transformacijų metu formuojamo XMI struktūra panaši nuspręsta padaryti apytikslį kiekybinį įvertinimą.

14 lent. -- Kiekybiniai eksperimentų rezultatai (modelių dydžiai nurodyti eilučių skaičiumi)

Bandomosios transformacijos numeris	Išeities modelio dydis	Gauto reliacinio modelio dydis	Gauto Java modelio dydis	Gautų modelių dydžių suma	Generuoto padidėjimo dalis
1	20	6	111	117	82,91%
2	29	26	191	217	86,64%
3	35	35	238	273	87,18%
4	34	15	230	245	86,12%
Vidurkiai:	**29,5**	**20,5**	**192,5**	**213**	**85,71%**

Eilučių skaičius buvo pasirinktas kaip kiekybinio matavimo priemonė. Eksperimentų kiekybinių rezultatų suvestinė pateikta 14 lent..

Analizuodami kiekybinių rezultatų suvestinę pastebime, kad reliacinis modelis gali būti įvertintas kaip nedetalizuojantis, t.y. mes negauname detalesnio rezultate modelio.

Visai kitokią išvadą galima padaryti apie transformaciją į Java PSM. Rezultatai akivaizdžiai rodo didelį modelio elementų generacijos laipsnį. Nežiūrint to, kad mūsų pritaikytas *Proxy* šablonas gana paprastas, galima pastebėti labai artimas rezultatų reikšmę paskelbtiems [5] dokumente (t.p. žiūrėkite 1 lent.).

7 IŠVADOS

1. Tiriamojo darbo metu sukurti:
    a. Metamodelis „Esybė – procesas“ aprašantis nuo platformos nepriklausomų modelių (PIM) kūrimo kalba.
    b. Metamodelis Java programavimo kalba parašytam programiniam kodui modeliuoti.
    c. Metamodelis reliaciniam modeliam kurti.
    d. EMF programavimo aplinkos priemonėmis sukurti visų modelių redaktoriai ir programavimo sąsajos modelių nuskaitymui, manipuliavimui ir saugojimui XMI formate.
2. Išanalizuotas PIM į PSM transformavimo procesai ir tris galimos realizavimo architektūros.
3. Pasirinkta ir pritaikyta transformacijos realizavimo architektūra pagrįsta filtrų architektūriniu šablonu, tokios architektūros pranašumai:
    a. Realizacijos lankstumas pakeitimams – galima įjungti papildomus ar išjungti naudojamus nepriklausomus filtrų blokus, tokiu būdu keičiant transformacijos rezultatą;
    b. Pritaikius filtrų architektūros taikymas transformacijos dekompozicijos procesas tampa paprastesnis;
    c. Galimos lygiagrečios tokios architektūros optimizuotos realizacijos, tokiu būdu galima padidinti transformavimo įrankių našumą.
4. Sukurta eksperimentinė transformacijos realizacija ir įvykdytos sėkmingos bandomosios transformacijos.
5. Bandymų metu įvertinti kiekybiniai kriterijai parodė, kad taikant net paprasčiausią Proxy šabloną transformacijos metu sugeneruoto modelio dalis yra vidutiniškai 85,71%.
6. Pastebėta, kad gauta generuojamo modelio dalis savo reikšme labai artima kodo generavimo efektyvumo statistinei reikšmei nurodytai literatūroje [5].

8 LITERATŪRA

1. OMG. „Meta Object Facility (MOF) Specification“. 2002 [žiūrėta 2003-04-20]. Prieiga per internetą: http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf.
2. OMG. „Model Driven Architekture (MDA)“. 2001 [žiūrėta 2003-02-04]. Prieiga per internetą: http://www.omg.org/docs/ormsc/01-07-01.pdf.
3. Kleppe A., Warmer J., Bast W. „MDA Explained. The Model Driven Architecture: Practice and Promise“. Boston: Addison-Wesley, 2003.
4. OMG. „MDA Guide Version 1.0.1“. 2003 [žiūrėta 2003-10-11]. Prieiga per internetą: http://www.omg.org/docs/omg/03-06-01.pdf.
5. Bettin J. „Model-Driven Architecture – Implementation and Metrics. Version 1.1“. 2003 [žiūrėta 2004-01-29]. Prieiga per internetą: http://www.softmetaware.com/mda-implementationandmetrics.pdf.
6. OMG. „Meta Object Facility (MOF) 2.0 Core Proposal“. 2003 [žiūrėta 2003-05-05]. Prieiga per internetą: http://www.omg.org/docs/ad/03-04-07.pdf.
7. OMG. „Request for Proposal: MOF 2.0 Query / Views / Transformations RFP“. 2002 [žiūrėta 2003-08-01]. Prieiga per internetą: http://www.omg.org/docs/ad/02-04-10.pdf.
8. Judson S. R., France R. B., Carver D. L.. „Specifying Model Transformations at the Metamodel Level“. [žiūrėta 2003-04-07] Prieiga per internetą: http://www.metamodel.com/wisme-2003/19.pdf.
9. OMG. „XML Metadata Interchange (XMI) Specification“. 2003 [žiūrėta 2003-10-16]. Prieiga per internetą: http://www.omg.org/cgi-bin/apps/doc?formal/03-05-02.pdf.
10. Budinsky F., Steinberg D., Merks E., Ellersick R., Grose T. „Eclipse modeling framework“. Boston: Addison-Wesley, 2004.
11. Gosling J., Joy B., Steele G., Bracha G. „The Java™ Language Specification. Second Edition“. Boston: Addison-Wesley, 2000.
12. Coplien J. O., Schmidt D. C.. „Pattern Languages of Program Desing“. Boston: Addison-Wesley, 1995.
13. Gamma E., Helm R. Johnson R., Vlissides J. „Design patterns“. Boston: Addison-Wesley, 1994.

9 TERMINŲ IR SANTRUMPŲ ŽODYNAS

15 lent. -- Terminai

Lietuviškai	**Angliškai**	**Paaiškinimas**
Abstrakcija	Abstraction	Esybės ar proceso aprašymas atmetant detales kurios nagrinėjimo metu laikomos neesminėmis.
Detalizavimas	Refinement	Detalizavimas – tai detalesnis sistemos aprašymas kuris vienareikšmiškai atitinka abstraktesnį sistemos aprašymą.
Infrastruktūra	Infrastructure	Infrastruktūra (taip pat vadinama skaičiavimo (*computing*) infrastruktūra) yra programinės ar aparatinės sistemos dalys kurios laikomos duotomis taikomosios programos realizavimo metu.
Modelis	Model	Modelis yra sistemos dalies (funkcijos, struktūros ir/arba elgesio) atvaizdavimas kokioje nors sintaksiškai formalioje kalboje (pvz. UML, MOF).
Nuo platformos nepriklausantys modeliai	Platform independent models (PIM)	PIM pateikia formalią sistemos struktūros ir funkcijų specifikaciją abstrahuotą nuo techninių realizacijos detalių.
Nuo skaičiavimų nepriklausomas modelis	Computationally Independent Model	CIM yra sistemos atvaizdavimas iš nuo skaičiavimų nepriklausomo požiūrio. CIM neparodo programos struktūros detalių. CIM kartais vadinamas probleminės srities modeliu.
Objektinė programinė įranga	Object Oriented Software	Programinė įranga sukurta pagal objektinio programavimo principus naudojant pasirinktą objektinio programavimo kalbą.
Platforma	Platform	Platforma yra programinė infrastruktūra

		realizuota tam tikra technologija (pvz.: Unix platforma, CORBA platforma, Windows platforma, .NET platforma, J2EE platforma)
Platformai specifiniai modeliai	Platform specific models (PSM)	PSM yra išreikštas tikslinės platformos modelio terminais ir pateikia formalią sistemos specifikaciją tam tikroje tikslinėje platformoje.
Požiūrio taškas	Viewpoint	Susitarimų (*convention*) specifikacija apibrėžianti požiūrių konstravimą ir naudojimą.
Požiūris	View	Visos sistemos reprezentacija iš susijusių interesų perspektyvos. Kiekvienas atvaizdas papildo sistemos aprašymą tam tikros perspektyvos požiūriu.
Projekcija	Mapping	Taisyklių ir metodų aibė naudojama modelio modifikavimui norint gauti kitą modelį. Sukuriant modelių projekcijas į tam tikras platformas vėliau galima atlikti automatinį ar rankinį modelio transformavimą į tikslinę platformą.
Vykdymo aplinka	Execution environment	Vykdymo aplinka priklauso nuo programinės ir aparatinės infrastruktūros ir yra realizuota vienos ar kelių platformų.

16 lent. -- Santrumpos

Santrumpa	**Pilnas pavadinimas**	**Paaiškinimas/Vertimas**
IT	Informacinės technologijos	-
IS	Informacinė sistema	-
CIM	Computationally independent Model	Nuo skaičiavimų nepriklausomas modelis.
CWM	Common Warehouse	Duomenų saugyklų metamodelio standartas.

	Metamodel	
DTD	Document Type Definition	Kalba naudojama dokumento struktūrai apibrėžti.
EJB2.0	Enterprise Java Beans 2.0	J2EE paskirstytų komponentų architektūra sukurta kompanijos Sun Microsystems Inc.
J2EE	Java 2 Enterprise Edition	Java 2 verslo sistemų programavimo platforma.
M0	Metalevel-0 (Runtime)	0-is metalygis – atitinka konkretų taikomosios programos vykdymą.
M1	Metalevel-1 (Application Model)	1-as metalygis – atitinka konkrečios programos modelį (pvz.: objektinės programos specifikacija UML, IDL interfeisų aprašai, Java programos tekstas).
M2	Metalevel-2 (Application Metamodel)	2-as metalygis – atitinka programos modelio gramatiką.
M3	Metalevel-3 (Application Meta-Metamodel)	3-ias metalygis – atitinka gramatiką leidžiančią apibrėžti naujas modeliavimo ir gramatikų apibrėžimo gramatikas (pvz.: MOF –gali apibrėžti save ir kitas modeliavimo kalbas kaip UML).
MDA	Model Driven Architecture	Modeliais pagrįsta architektūra.
MOF	Meta-Object Facility	Metaobjektų infrastruktūra – metamodeliavimo ir metaduomenų saugyklų standartas.
MT	Model Transformation	Modelių transformacija MDA architektūros kontekste.
OCL	Object Constraint Language	OMG objektų apribojimų specifikavimo kalba. Šiuo metu kūrimo stadijoje.
OMG	Object Management Group	Kompanijos pavadinimas kuri sukūrė UML, MOF, CWM, XMI, CORBA ir daugybe kitų specifikacijų.
PIM	Platform Independent Model	Nuo platformos nepriklausomas modelis.
PSM	Platform Specific Model	Platformai specifinis modelis.
UML	Unified Modeling Language	Universali, OMG konsorciumo sukurta,

		modeliavimo kalba skirta aprašyti bei specifikuoti objektinės programinės įrangos sudėtiniams elementams ir sistemoms.
XMI	XML Metadata Interchange	OMG metaduomenų apsikeitimo standartas pagrįstas XML dokumentais.
XML	Extensible Markup Language	W3C konsorciumo standartas, išplečiama kalba, naudojama duomenų struktūros sužymėjimui, saugojimui ir perdavimui.

10 PRIEDAI

10.1 Esybė-Procesas metamodelio aprašymas Ecore, XMI 2.0 formate

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="pemodel"
    nsURI="http:///pemodel.ecore" nsPrefix="pemodel">
  <eClassifiers xsi:type="ecore:EClass" name="Process" eSuperTypes="#//NamedElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="method" lowerBound="1"
        upperBound="-1" eType="#//Method" containment="true"
eOpposite="#//Method/ownerProcess"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ownerPackage"
eType="#//Package"
        transient="true" eOpposite="#//Package/process"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Entity" abstract="true"
eSuperTypes="#//DataType">
    <eStructuralFeatures xsi:type="ecore:EReference" name="reference" upperBound="-1"
        eType="#//Reference" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="SimpleProperty"
eSuperTypes="#//Property"/>
  <eClassifiers xsi:type="ecore:EClass" name="Reference" eSuperTypes="#//NamedElement">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="aggregation"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="byValue"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="lowerBound"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EIntegerObject"
        defaultValueLiteral="0"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="upperBound"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EIntegerObject"
        defaultValueLiteral="-1"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="type" lowerBound="1"
eType="#//Entity"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Parameter" eSuperTypes="#//NamedElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="ownerMethod" lowerBound="1"
        eType="#//Method" transient="true" eOpposite="#//Method/parameter"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="type" lowerBound="1"
eType="#//DataType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="NamedElement" abstract="true"
eSuperTypes="#//ModelElement">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Method" eSuperTypes="#//NamedElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="ownerProcess" lowerBound="1"
        eType="#//Process" transient="true" eOpposite="#//Process/method"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="parameter" upperBound="-1"
        eType="#//Parameter" containment="true" eOpposite="#//Parameter/ownerMethod"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="returnType" lowerBound="1"
        eType="#//DataType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Package" eSuperTypes="#//NamedElement">
```

```
    <eStructuralFeatures xsi:type="ecore:EReference" name="child" upperBound="-1"
        eType="#//Package" containment="true" eOpposite="#//Package/parent"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="parent" lowerBound="1"
        eType="#//Package" transient="true" eOpposite="#//Package/child"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="process" upperBound="-1"
        eType="#//Process" containment="true" eOpposite="#//Process/ownerPackage"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="domain" lowerBound="1"
        eType="#//Domain" transient="true" eOpposite="#//Domain/package"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="dataType" upperBound="-1"
        eType="#//DataType" containment="true" eOpposite="#//DataType/ownerPackage"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DataType" abstract="true"
eSuperTypes="#//NamedElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="ownerPackage" lowerBound="1"
        eType="#//Package" transient="true" eOpposite="#//Package/dataType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="PrimitiveType" eSuperTypes="#//DataType">
    <eStructuralFeatures xsi:type="ecore:EReference" name="domain" lowerBound="1"
        eType="#//Domain" transient="true" eOpposite="#//Domain/dataType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Identifier"
eSuperTypes="#//ModelElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="member" lowerBound="1"
        upperBound="-1" eType="#//PersistentProperty"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="PersistentEntity"
eSuperTypes="#//Entity">
    <eStructuralFeatures xsi:type="ecore:EReference" name="identifier" lowerBound="1"
        eType="#//Identifier" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="persistentProperty"
lowerBound="1"
        upperBound="-1" eType="#//PersistentProperty" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="parent"
eType="#//PersistentEntity"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="simpleProperty"
upperBound="-1"
        eType="#//SimpleProperty" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="PersistentProperty"
eSuperTypes="#//Property"/>
  <eClassifiers xsi:type="ecore:EClass" name="Property" abstract="true"
eSuperTypes="#//NamedElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="type" lowerBound="1"
eType="#//PrimitiveType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ModelElement" abstract="true"/>
  <eClassifiers xsi:type="ecore:EClass" name="SimpleEntity" eSuperTypes="#//Entity">
    <eStructuralFeatures xsi:type="ecore:EReference" name="property" upperBound="-1"
        eType="#//SimpleProperty" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="parent"
eType="#//SimpleEntity"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="List" eSuperTypes="#//DataType">
    <eStructuralFeatures xsi:type="ecore:EReference" name="type" lowerBound="1"
eType="#//DataType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Domain" eSuperTypes="#//NamedElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="dataType" upperBound="-1"
        eType="#//PrimitiveType" containment="true"
eOpposite="#//PrimitiveType/domain"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="package" upperBound="-1"
        eType="#//Package" containment="true" eOpposite="#//Package/domain"/>
  </eClassifiers>
</ecore:EPackage>
```

10.2 Reliacinio metamodelio aprašymas Ecore, XMI 2.0 formate

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="sqlmodel"
    nsURI="http:///sqlmodel.ecore" nsPrefix="sqlmodel">
  <eClassifiers xsi:type="ecore:EClass" name="SqlDataType"
eSuperTypes="#//DatabaseElement"/>
  <eClassifiers xsi:type="ecore:EClass" name="Table" eSuperTypes="#//DatabaseElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="column" lowerBound="1"
        upperBound="-1" eType="#//Column" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="primaryKey" lowerBound="1"
        eType="#//Key"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="foreignKey" upperBound="-1"
        eType="#//ForeignKey" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="candidateKey" lowerBound="1"
        upperBound="-1" eType="#//Key" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Column" eSuperTypes="#//TableElement">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="allowNull"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="size" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EIntegerObject"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="type" lowerBound="1"
eType="#//SqlDataType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Key" eSuperTypes="#//SqlNamedElement
#//TableElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="column" lowerBound="1"
        upperBound="-1" eType="#//Column"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ForeignKey"
eSuperTypes="#//SqlNamedElement #//TableElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="referencedKey"
lowerBound="1"
        eType="#//Key"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="foreignRef" lowerBound="1"
        upperBound="-1" eType="#//ForeignReference" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="SqlNamedElement" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Database"
eSuperTypes="#//SqlNamedElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="ownedElements" upperBound="-
1"
        eType="#//DatabaseElement" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DatabaseElement" abstract="true"
eSuperTypes="#//SqlNamedElement"/>
  <eClassifiers xsi:type="ecore:EClass" name="TableElement" abstract="true"
eSuperTypes="#//SqlNamedElement"/>
  <eClassifiers xsi:type="ecore:EClass" name="ForeignReference">
    <eStructuralFeatures xsi:type="ecore:EReference" name="local" lowerBound="1"
eType="#//Column"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="foreign" lowerBound="1"
        eType="#//Column"/>
  </eClassifiers>
</ecore:EPackage>
```

10.3 Java metamodelio aprašymas Ecore, XMI 2.0 formate

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="javamodel"
    nsURI="http:///javamodel.ecore" nsPrefix="javamodel">
  <eClassifiers xsi:type="ecore:EClass" name="JavaPackage"
eSuperTypes="#//JavaModelElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="types" upperBound="-1"
        eType="#//JavaClass" containment="true" eOpposite="#//JavaClass/package"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="subpackage" upperBound="-1"
        eType="#//JavaPackage" containment="true" eOpposite="#//JavaPackage/package"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="package" lowerBound="1"
        eType="#//JavaPackage" transient="true" eOpposite="#//JavaPackage/subpackage"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="JavaModelElement" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="JavaMember" abstract="true"
eSuperTypes="#//JavaModelElement">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="static"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="visibility"
eType="#//JavaVisibility"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="containingType"
eType="#//JavaClass"
        transient="true" eOpposite="#//JavaClass/member"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="JavaParameter"
eSuperTypes="#//JavaModelElement">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="final"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="type" lowerBound="1"
eType="#//JavaClass"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="method" lowerBound="1"
        eType="#//JavaMethod" transient="true" eOpposite="#//JavaMethod/parameter"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="JavaClass" eSuperTypes="#//JavaMember">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="abstract"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="final"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="interface"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="generate"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="package" lowerBound="1"
        eType="#//JavaPackage" transient="true" eOpposite="#//JavaPackage/types"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="superType" upperBound="-1"
        eType="#//JavaClass"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="member" upperBound="-1"
        eType="#//JavaMember" containment="true"
eOpposite="#//JavaMember/containingType"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="method" upperBound="-1"
        eType="#//JavaMethod"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="field" upperBound="-1"
        eType="#//JavaField"/>
  </eClassifiers>
```

```
  <eClassifiers xsi:type="ecore:EClass" name="JavaField" eSuperTypes="#//JavaMember">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="final"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="transient"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="volatile"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="initializer"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
        defaultValueLiteral="false"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="type" lowerBound="1"
eType="#//JavaClass"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="JavaMethod" eSuperTypes="#//JavaMember">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="abstract"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="final"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="native"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="synchronized"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="constructor"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="body" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="parameter" upperBound="-1"
        eType="#//JavaParameter" containment="true"
eOpposite="#//JavaParameter/method"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="returnType"
eType="#//JavaClass"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="exception" upperBound="-1"
        eType="#//JavaClass"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="JavaVisibility">
    <eLiterals name="public"/>
    <eLiterals name="protected" value="1"/>
    <eLiterals name="private" value="2"/>
    <eLiterals name="package" value="3"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="JavaInitializer"
eSuperTypes="#//JavaMember">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="body" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="JavaApplication"
eSuperTypes="#//JavaModelElement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="package" upperBound="-1"
        eType="#//JavaPackage" containment="true"/>
  </eClassifiers>
</ecore:EPackage>
```