KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Andrius Armonas

KONCEPTUALIŲJŲ APRIBOJIMŲ TRANSFORMACIJA Į SQL KODĄ

Magistro darbas

Vadovė

doc. dr. L. Nemuraitė

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

TVIRTINU

Katedros vedėjas

doc. dr. R. Butleris

KONCEPTUALIŲJŲ APRIBOJIMŲ TRANSFORMACIJA Į SQL KODĄ

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė

Lietuvių k. kat. lekt.

dr. J. Mikelionienė

Vadovė

doc. dr. L. Nemuraitė

Recenzentas

doc. dr. E. Bareiša

Atliko

IFM 9/1 gr. stud.

A. Armonas

KAUNAS, 2005

SUMMARY

In this paper, the method is proposed for transforming *UML* class diagrams with *OCL* constraints to relational database schemas, having advantages over "*UML Profile for Databases*" models. The proposed method consists of two phases supplementing each other: metamodel based transformations and pattern-based transformations. All transformations are based on *OMG* official standards or *RFPs* (*Request for Proposals*) and are prepared for use in *MDA* (*Model Driven Architecture*) context. This means, that resulting models, created using the described method, are long-lasting, independent from platform and abstract enough to be independent from technology.

This work covers analysis of the field of object – relational transformations, analysis of support of *OCL* in currently used *UML* tools, transformation rule sets of metamodel transformations and pattern-based transformations, a lot of examples illustrating every transformation. Metamodel transformations are designed to transform object models written in *UML* language to corresponding relational models: types, classes, attributes, association classes, associations to tables, columns, foreign keys and other concepts. Pattern-based transformations are designed for generating static relational concepts such as *check constraints, views, stored procedures* and *triggers*. Using the two described transformation types and the whole method described, one can generate full-fledged relational database schemas, have precise *UML* models and keep much more consistency between different project parts.

TURINYS

ĮVADAS **7**

1. MODELIŲ VALDOMOS ARCHITEKTŪROS SĄVOKOS **10**

2. OCL – SQL TRANSFORMACIJŲ SRITIS IR PROBLEMATIKA **12**

2.1. RDB SCHEMŲ GENERAVIMAS ŠIANDIEN **12**
2.2. OCL IR RDB APRIBOJIMŲ KLASIFIKACIJA IR PALYGINIMAS **13**
2.3. SUNKUMAI, IŠKYLANTYS TRANSFORMUOJANT UML/OCL MODELIUS Į SQL KODĄ **15**
2.4. TRANSFORMACIJOMS NAUDOJAMI PROGRAMINIAI PAKETAI IR SRITIES ESMINĖS PROBLEMOS **18**

3. KONCEPTUALIŲJŲ APRIBOJIMŲ TRANSFORMACIJŲ METODAI **20**

3.1. KONCEPTUALIEJI APRIBOJIMAI METAMODELIŲ TRANSFORMACIJŲ METU **20**
3.1.1. OMG METAMODELIŲ LYGMENYS 20
3.1.2. DARBE NAUDOJAMI METAMODELIŲ TRANSFORMACIJŲ TIPAI IR KALBOS 22
3.1.3. UML/OCL MODELIŲ PAPRASTŲJŲ TIPŲ TRANSFORMACIJOS Į RELIACINIŲ MODELIŲ PAPRASTUOSIUS TIPUS 25
3.1.4. UML MODELIŲ TRANSFORMACIJOS Į RELIACINIUS MODELIUS 31
3.1.5. UML MODELIŲ TRANSFORMACIJOS Į RELIACINIUS MODELIUS (TRL VARIANTAS) 35
3.2. TIPINIAI TRANSFORMACIJŲ ŠABLONAI **37**
3.2.1. UML OCL TRANSFORMACIJŲ Į VAIZDUS METODAI 37
3.2.2. UML OCL TRANSFORMAVIMAS Į VAIZDUS NAUDOJANT IŠVESTINES KLASES 38
3.2.3. UML OCL TRANSFORMAVIMAS Į VAIZDUS NAUDOJANT OCL „DERIVE“ IŠRAIŠKAS 43
3.2.4. RELIACINĖ ALGEBRA IR OCL 45
3.2.5. TRANSFORMACIJŲ MEDŽIŲ PANAUDOJIMAS SQL KODO GENERAVIMUI IŠ OCL IŠRAIŠKŲ 46
3.2.6. SERVERIO PROCEDŪRŲ IR TRIGERIŲ GENERAVIMO PRINCIPAI 50

4. APIBENDRINTAS ALGORITMAS UML MODELIŲ TRANSFORMACIJAI Į PILNAVERČIUS RELIACINIUS MODELIUS **53**

5. METODO TAIKYMO PAVYZDYS **54**

IŠVADOS **58**

LITERATŪRA **59**

TERMINŲ IR SANTRUMPŲ ŽODYNAS **61**

PRIEDAS. PUBLIKUOTI STRAIPSNIAI **62**

Lentelių sąrašas

2.1 lentelė OCL ir RDBVS apribojimų palyginimas 14

2.2 lentelė Numanomųjų, išreikštinių ir būdingųjų apribojimų reliaciniuose modeliuose ir *SQL*:2003 standarte semantikos palyginimas 14

3.1 lentelė OCL ir SQL standartų paprastųjų tipų palyginimas 27

5.1 lentelė Gamybos informacijos sistemos dalykinės srities klasių diagramos OCL apribojimai 55

5.2 lentelė Gamybos informacijos sistemos SQL kodo fragmentai, gauti transformacijų metu 56

Paveikslėlių sąrašas

1.1 pav. Modeliais valdomos architektūros koncepcija 10

1.2 pav. Verslo, informacijos ir programų sistemų sąryšis su MDA 11

2.1 pav. OCL-SQL transformacijų klasifikacija 18

3.1 pav. Modelių, kalbų, metamodelių ir metakalbų sąryšis 20

3.2 pav. OMG modelių lygmenys M0, M1, M2, M3 21

3.3 pav. MDA, papildytas metamodelių lygmenimis 22

3.4 pav. OCL tipų metamodelis. Abstrakti sintaksė 25

3.5 pav. OCL Standard Library bibliotekoje aprašyti tipai 26

3.6 pav. Reliacinis metamodelis: lentelės, stulpeliai ir duomenų tipai (CWM 1.1 standartas, ad/03-03-02) 27

3.7 pav. Grafinė transformacijos *StringToCHAR* interpretacija 29

3.8 pav. Transformacijos *StringToCHAR* demonstravimui skirta klasių diagrama ir OCL apribojimai 29

3.9 pav. Supaprastintas UML metamodelis 31

3.10 pav. Supaprastintas SQL metamodelis 32

3.11 pav. Supaprastintas RDBMS metamodelis (TRL variantui) 36

3.12 pav. Supaprastintas UML metamodelis (TRL variantui) 36

3.13 pav. Klasė *Emp1* 38

3.14 pav. Emp1, Emp2 klasių ryšys su Database klase 38

3.15 pav. Filmų pavyzdinė schema 40

3.16 pav. Klasė /MovieProducer 40

3.17 pav. Metodu *convertToMP()* papildyta *Movie* klasė 40

3.18 pav. Tarpusavyje nesusietos asociacijomis klasės *Emp1* ir *Emp2* 41

3.19 pav. Klasių diagrama, papildyta *Emp12*, *Emp21* bei *Database* klasėmis 41

3.20 pav. Klasių diagrama, papildyta */Emp* klase 42

3.21 pav. Klasė NewMovies, kurios egzempliorių reikšmės gautos operacijų su Movies klasės egzemplioriais metu 43

3.22 pav. Klasė Cars ir išvestinė klasė CarsEur 43

3.23 pav. Klasės Cars, Manufacturers ir išvestinė klasė CarManufacturers 44

3.24 pav. Klasės CarsOld, CarsNew ir jų dekarto sandauga OldXNew 44

3.25 pav. Dekarto sandaugos demonstracija 45

3.26 pav. Gamybinės sistemos dalykinės srities priklausomo nuo platformos lygmens klasių diagramos dalis 47

3.27 pav. Metodo getAllProducts() transformacijos medis 48

3.28 pav. Metodo getProductDescriptions() transformacijos medis 48

3.29 pav. Metodo getTotalPrice() transformacijos medis 49

3.30 pav. Metodo getPlannedAmount() transformacijos medis 50

3.31 pav. Klasė Product ir jos metodas produktų kainų sumos skaičiuoti 50

3.32 pav. Metodo getTotalPrice(), panaudojant iterate operaciją, transformacijos medis 51

5.1 pav. Gamybos informacijos sistemos PIM lygmens dalykinės srities klasių diagrama. 54

5.2 pav. Gamybos informacijos sistemos PSM lygmens dalykinės srities klasių diagrama. 54

ĮVADAS

Didžioji dalis praktikoje naudojamų duomenų bazių valdymo sistemų – reliacinės, tačiau vis labiau įsigali objektinės orientacijos projektavimo procesai, taigi jie turėtų leisti kurti bei palaikyti programų sistemas, veikiančias kartu su reliacinėmis duomenų bazėmis.

Nemažai autorių aprašo objektinių modelių klasių diagramų transformavimą į reliacinių duomenų bazių schemas. Dažniausiai šios transformacijos apsiriboja paprastu atveju *klasė → lentelė*, į transformavimo procesą įtraukiant tik atributus ir ryšius. Tokios transformacijos rezultatas labai ribotai panaudoja reliacinių duomenų bazių technologijos teikiamas galimybes. Domenų bazės integralumo užtikrinimo priemonės – trigeriai, ryšių vientisumo priemonės (angl. *referential integrity*), automatinis įvairių apribojimų tikrinimas paliekami nuošalyje.

Vienas iš praktikoje naudojamų metodų generuoti reliacinių duomenų bazių (*RDB*) schemas iš *UML* diagramų – *Rational* bendrovės sukurtas „*UML profile for databases*“ modelis [3]. Dirbant pagal šią metodiką, iš *UML* modelių automatiškai nesugeneruojami nei trigeriai, nei serverio procedūros, nei sąlygų tikrinimo funkcijos. Taigi minėtoji metodika (ji nėra standartas), naudojama *Rational* produktuose, netinka automatiniam pilnų *RDB* schemų generavimui. Kitos bendrovės taip pat kuria panašių galimybių modelius ir integruoja juos į savo įrankius.

Taigi, galima išskirti keletą esminių problemų srityje:

- Sukurti modeliai, iš kurių generuojamos *RDB* schemos, neužtikrina suderinamumo tarp skirtingų projekto artefaktų (*RDB*, *XML* ar operacijų schemų)
- Patys *RDB* generavimo mechanizmai nebūna standartizuoti (kiekviena bendrovė kuria savo mechanizmus bei modelius *RDB* generavimui)
- Didžioji dalis tokių konceptų, kaip serverio procedūros, trigeriai, aprašomi konkrečiu *SQL* dialektu, taip užkertant kelią pakartotiniam modelių panaudojimui. Sukurti modeliai, iš kurių generuojamos *RDB* schemos, turi būti koreguojami rankiniu būdu, norint migruoti tarp skirtingų bendrovių *SQL* serverių, ir perprojektuojami iš naujo, pasikeitus technologijoms

MDA (Model Driven Architecture) [14] siūloma architektūra skirta greitai reaguoti į besikeičiančias informacinių sistemų kūrimo technologijas bei išvengti su tokiais pokyčiais susijusio sistemos perprojektavimo. *MDA* modelius gali naudoti įvairūs įrankiai, skirti generuoti duomenų bazių schemas, veikiantį programinį kodą, testavimo šablonus, įdiegimo scenarijus ir pan. Ši architektūra aprašo bendrus modelių kūrimo principus, tačiau nepateikia konkrečių transformacijų į šių dienų technologijų programinį kodą. Viena aktualiausių yra transformacija iš objektinio modelio į

reliacinį. Nors ji naudojama jau seniai, tačiau aukščiau minėtos *RDB* generavimo problemos nėra išspręstos iki šiol.

Šio darbo tikslas – panagrinėti, kaip *MDA PIM (Platform Independent Model)* ir *PSM (Platform Specific Model)* lygmenų modelius galima papildyti *OCL* išraiškomis, kad jiems būtų galima pritaikyti metamodelių transformacijas, o vėliau, naudojant transformacijų šablonus, būtų galima generuoti pilnavertį *SQL* kodą.

Taigi, darbe siekiama:

- Sudaryti metodą, kurio pagrindu kuriami aukštesnio abstrakcijos lygmens informacinių sistemų modeliai, tam panaudojant *MDA* architektūrą
- Pasiekti pilnaverčių *RDB* schemų generavimą iš konceptualiojo lygmens modelių ir aprašyti principus, kaip iš šio lygmens modelių generuoti reliacinių duomenų bazių integralumo apribojimus, vaizdus, trigerius ir serverio procedūras

Darbe remiamasi tik oficialiais *OMG* standartais bei *RFP* dokumentais, taip pat *SQL:2003* [21] standartu. Aprašomu metodu sukurti modeliai veikia *MDA* [12] architektūros principais. Pirmiausia sudaromi nuo platformos nepriklausomi (*PIM*) *UML* [2] [17] [18] [19] modeliai. Jie papildomi *OCL* [19] apribojimais, taip pilnavertiškai nusakant dalykinę sritį. Iš šio lygmens modelių galima sugeneruoti ne tik *RDB* schemas, tačiau ir kitos technologijos programinį kodą ar schemas. Kadangi dalykinės srities modeliai aprašyti *OCL* išraiškomis, jie yra tikslūs, ir net revoliuciškai pasikeitus technologijoms, lieka vertingi. Vėliau metamodelių lygmens transformacijų metu iš *PIM* lygmens modelių gaunami priklausomi nuo platformos modeliai (*PSM*). Metamodelių lygmenyje naudojamos *egzomorfinės* arba kitaip tariant *horizontalios* transformacijos – jų metu veikiama dviejuose metamodeliuose, t. y. šaltinio ir tikslo modeliai aprašomi naudojant skirtingus metamodelius. Šaltinio metamodelis šiame darbe – *UML*. O tikslo metamodeliu pasirinkta *OMG CWM* (angl. C*ommon Warehouse Metamodel*) [20] standarto reliacinė dalis. Šio metamodelio pagrindu galima sudaryti reliacines schemas atitinkančius objektinius modelius. Paskutiniame etape iš *PSM* lygmens modelių generuojamas *SQL* kodas. Tam naudojami tipiniai transformacijų šablonai. Jie taikomi ten, kur negalima pritaikyti transformacijų metamodelių lygmenyje. Tokios situacijos dažniausiai susidaro tada, kai *UML* modelių elementams specifikuojami *OCL* apribojimai. Jų transformuoti metamodelių lygmenyje negalima. Pavyzdžiui, vaizdų generavimui darbe naudotos *OCL* „*derive*“ išraiškos. Reikia atkreipti dėmesį, jog pagal darbe aprašytą metodą sudaryti *PSM* lygmens modeliai, kaip ir *PIM* lygmens modeliai yra tikslūs, todėl tinka įvairių *SQL* dialektų kodo generavimui.

1 skyrelyje „Modelių valdomos architektūros sąvokos“ aprašomi *MDA* architektūros principai bei *OCL* kalbos panaudojimo galimybės šioje architektūroje. 2 skyrelyje „OCL – SQL transformacijų

sritis ir problematika“ atliekama literatūros analizė, analizuojamas reliacinių duomenų bazių schemų generavimas šiandien, šių schemų generavimo įrankiai, metodikų trūkumai, palyginami objektinio ir reliacinio modelių apribojimai, pateikiamos susistemintos problemos srityje. 3 skyrelyje „Konceptualiųjų apribojimų transformacijų metodai“ siūlomas dviejų fazių generavimo iš *UML*/*OCL* modelių į *RDB* schemas metodas: metamodelių lygmens transformacijų ir tipinių transformacijų šablonų. Šiame skyrelyje pateikiama *OMG* metamodelių lygmenų analizė; metamodelių transformacijų klasifikacija ir esminiai principai; metamodelių transformacijų kalbų analizė ir darbe naudojamos kalbos aprašymas; klasių, atributų, asociacijų, asociacijų klasių, tipų transformacijos į reliacinių duomenų bazių modelių elementus. Taip pat pateikiami tipinių transformacijų šablonų sudarymo principai, demonstruojami duomenų bazių integralumo apribojimų, vaizdų generavimo, trigerių ir serverio procedūrų generavimo pavyzdžiai. 4 skyrelyje pateikiamas apibendrintas algoritmas reliacinių duomenų bazių schemų generavimui. 5 skyrelyje demonstruojamas metodo taikymas konkrečiai dalykinės srities klasių diagramai.

Šiame darbe pateiktas metodas pilnaverčių reliacinių duomenų bazių schemų generavimui. Skirtingai nuo esamos praktikos, aprašytame metode reliaciniai modeliai generuojami iš konceptualių modelių, nepriklausančių nuo konkrečios technologijos (nėra orientuoti išskirtinai į reliacinių duomenų bazių schemų generavimą), metodas naudoja tik *OMG* standartus ir leidžia generuoti pilnavertį *SQL* kodą.

Darbe pateiktos metamodelių lygmens transformacijų taisyklės, transformuojančios *UML* metamodelio elementus į *CWM* metamodelio elementus bei reliacinių lentelių, atributų, domenų, pirminių ir išorinių raktų, išvedimo taisyklių, vaizdų, tikrinimo funkcijų, trigerių, procedūrų generavimo pavyzdžiai.

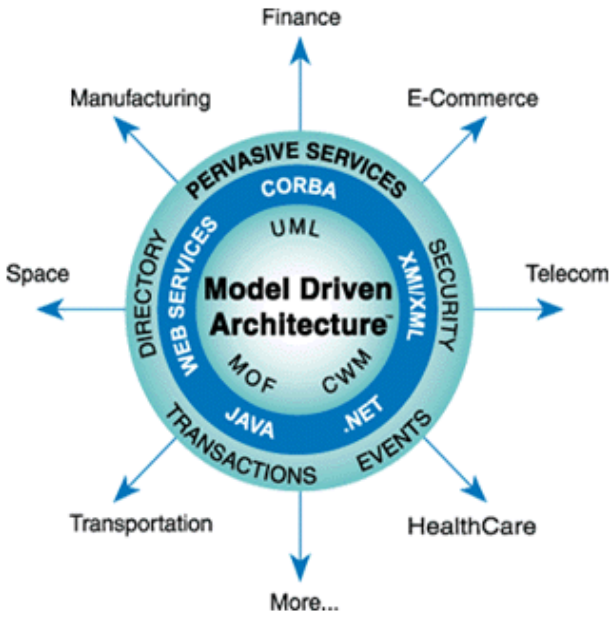
Dalyvauta konferencijose (straipsnių medžiaga pateikta priede):

- Informacinės technologijos 2005 (X tarpuniversitetinė magistrantų ir doktorantų konferencija), KTU, skaitytas pranešimas „Konceptualių apribojimų transformavimo į SQL kodą principai“
- Informacinės technologijos verslui 2005, VUKHF, priimtas spausdinti konferencijos leidinyje straipsnis „Pilnaverčių reliacinių duomenų bazių schemų generavimas naudojant UML ir OCL“

1. MODELIŲ VALDOMOS ARCHITEKTŪROS SĄVOKOS

Šiandien vis dar daug programų sistemų kuriama neatsižvelgiant į faktą, jog nuolat kinta infrastuktūra, kuriai kuriama sistema, reikalavimai ir, svarbiausia, – technologijos. Todėl, siekiant panaudoti pažangias technologijas, dažnai būtina kurti naują sistemą. Taip greitai keičiantis technologijoms, dar nesukurta sistema neretai būna pasenusi. Šiems pokyčiams valdyti *OMG* grupė pasiūlė *MDA* (angl. *model driven architecture*) (1.1 pav.) architektūrą.

Svarbu pažymėti, jog keičiantis technologijoms, verslo procesai, verslo taisyklės ir verslo aplinka daugeliu atveju lieka nepasikeitę arba keičiasi nežymiai [6] [14] [15]. Todėl ankščiau sukurti *MDA* modeliai lieka ir juos galima pakartotinai panaudoti, nors jie ir susiję su pasenusiomis technologijomis.



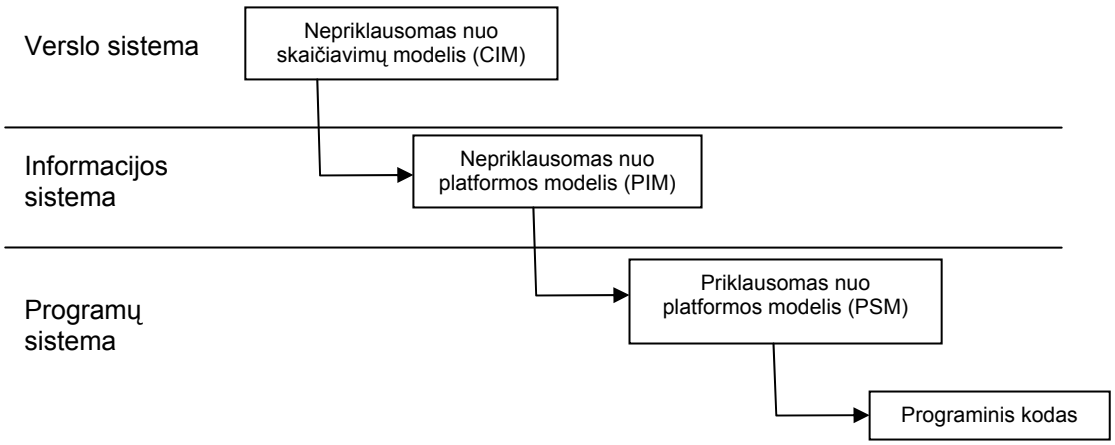
1.1 pav. Modeliais valdomos architektūros koncepcija

MDA siūlo spręsti modelių pakartotinio panaudojimo problemą naudojant trijų tipų modelius:

- nepriklausomą nuo skaičiavimų *(*angl. *computation independent model, CIM)*
- nepriklausomą nuo platformos *(*angl. *platform independent model, PIM)*
- priklausomą nuo platformos *(*angl. *platform specific model, PSM)*

CIM modelius kuria verslo analitikai tam, kad būtų galima aprašyti verslą, jame vykstančius procesus, verslo taisykles. Kuriamų sistemų architektūra aprašoma naudojant *PIM* lygmens modelius. Ją aprašo programinės įrangos projektuotojai. *PSM* savo ruožtu kuriamas projektuotojo, programuotojo arba testuotojo tam, kad būtų galima įgyvendinti konkrečią sistemos architektūrą. *OMG* siūloma architektūra nėra nauja. Tai kitų autorių pasiūlytų geriausių praktikų bei teorinių schemų realizacija.

Tarp *MDA* lygmenų modelių galima atlikti transformacijas (1.2 pav.). Tam naudojami modelių metamodeliai (gali būti naudojami net metametamodeliai) ir pasirinktos transformacijų kalbos. Pasikeitus verslui, *CIM* modeliai transformuojami į *PIM* lygmens modelius. Įvykus revoliuciniam technologijos pasikeitimui (paprastai kas dešimt metų), atliekamas atitinkamas *PIM* lygmens modelių keitimas ir tolimesnis pusiau automatinis transformavimas į *PSM* modelius. Smulkūs technologijų pasikeitimai neturi įtakos *PIM* lygmens modeliams. Taip *PIM* lygmuo apsaugomas nuo dažnos technologijų kaitos.



1.2 pav. Verslo, informacijos ir programų sistemų sąryšis su MDA

UML galima laikyti pusiau formalia modeliavimo kalba, todėl sukurtiems modeliams dažniausiai trūksta detalumo, jie neretai būna nepilnaverčiai ir tiksliai nespecifikuoja analizuojamų sričių, todėl iš jų dažniausiai nėra galimybių generuoti pilnavertį programinį kodą. Visuose *MDA* lygmenyse modelių apribojimams bei verslo taisyklėms aprašyti galima taikyti *OCL* kalbą. Visuose lygmenyse *OCL* išraiškomis papildyti modeliai tampa tikslesni ir formalesni, taigi ir tinkamesni kodo generavimui.

Šiame darbe nagrinėjamos *UML* modelių, papildant juos *OCL* išraiškomis, transformacijos į *SQL* kodą. Programinis kodas (taip pat ir *SQL* kodas), gali būti generuojamas ne tik iš *PSM* lygmens, bet iškart iš *PIM* lygmens modelių. Darbe bus nagrinėjama, kaip *OCL* išraiškos iš *PSM* lygmens modelių transformuojamos į programinį kodą. Vienas pagrindinių darbo tikslų – sudaryti tokius modelius, kurie galėtų būti kiek įmanoma geriau pakartotinai panaudojami.

2. OCL – SQL TRANSFORMACIJŲ SRITIS IR PROBLEMATIKA

2.1. RDB schemų generavimas šiandien

Paprastai didžioji dalis praktikoje naudojamų *DBVS* – reliacinės, tačiau vis labiau įsigali objektinės orientacijos projektavimo procesai [7], [8], [12], taigi jie turėtų leisti kurti bei palaikyti programų sistemas, veikiančias kartu su reliacinėmis duomenų bazėmis.

Nemažai autorių ([1], [5], [9], [10], [16]) aprašo objektinių modelių klasių diagramų transformavimą į reliacinių duomenų bazių schemas. Dažniausiai šios transformacijos apsiriboja paprastu atveju *klasė → lentelė*, į transformavimo procesą įtraukiant tik atributus ir ryšius. Tokios transformacijos rezultatas labai ribotai panaudoja reliacinių duomenų bazių technologijos teikiamas galimybes. Domenų bazės integralumo užtikrinimo priemonės – trigeriai, ryšių vientisumo priemonės (angl. *referential integrity*), automatinis įvairių apribojimų tikrinimas paliekami nuošalyje.

Vienas iš praktikoje naudojamų metodų generuoti *RDB* schemas iš *UML* diagramų – *Rational* bendrovės sukurtas „*UML profile for databases*“ modelis. Naudojant „*UML profile for databases*“ metodiką, *UML* modeliai transformuojami į tarpinius modelius, pataisomi projektuotojo ir tada generuojamas *SQL* kodas *RDB* schemoms kurti ar modifikuoti. Dirbant pagal šią metodiką, iš *UML* modelių automatiškai nesugeneruojami nei trigeriai, nei serverio procedūros, nei sąlygų tikrinimo funkcijos. Taigi minėtoji metodika (ji nėra standartas), naudojama *Rational* produktuose, netinka automatiniam pilnų *RDB* schemų generavimui. Be to pagal ją sukurti modeliai skirti tik *RDB* schemoms generuoti, todėl nėra pakartotinai panaudojami.

Gerai žinoma, jog veiklos taisyklių aprašymas duomenų bazių schemoje (panaudojant trigerius ir sąlygų tikrinimą) sumažina programinio kodo kiekį bei garantuoja, jog visa programinė įranga, dirbanti su konkrečia duomenų baze veiks pagal tas pačias veiklos taisykles, t. y. integralumą išlaikančius apribojimus.

Vienas iš tokių apribojimų generavimo metodų – *UML* panaudojimas kartu su *OCL* kalba.

Šio metodo privalumai:

- *UML* – plačiai paplitęs standartas, naudojamas objektinės orientacijos projektavimo procesuose. Todėl *UML* palaiko didelė dalis *CASE* įrankių
- *OCL* – abstrakti kalba, kuria galima tiksliai apibrėžti integralumo taisykles. Tai atliekama aprašant modelių objektų invariantus. Navigavimo (angl. *navigation – oriented*) metodologija iš dalies sutampa su reliacinių užklausų koncepcija
- Dauguma šiandieninių *RDBVS* naudoja savus dialektus trigeriams ir sąlygoms (nors *SQL:2003* standartas tam apibrėžia standartinę kalbą). Naudojant *OCL*, galima

specifikuoti apribojimus duomenų bazių schemoms nepriklausomai nuo konkrečios *RDBVS*

- Žiūrint iš metodologinės pusės, visada rekomenduojama specifikuoti apribojimus objektams *OCL* kalba. Praktiškai – tai ilgas procesas. Todėl šio proceso metu atliktą darbą tikslinga panaudoti, automatiškai generuojant *SQL* kodą
- *post-* ir *pre-* sąlygų specifikavimas *OCL* kalba gali būti geras atspirties taškas *SQL* užklausoms generuoti (*update*, *select* sakiniai, serverio procedūros, trigeriai)

2.2. OCL ir RDB apribojimų klasifikacija ir palyginimas

Objektinėje metodologijoje apribojimas traktuojamas kaip apribojimas objektinio modelio ar sistemos reikšmėms, jų poaibiui, t. y. jis visada susietas su modeliu.

Apskritai apribojimai reikalingi veiklos taisyklių apibrėžimui, kai veiklos taisyklės negali būti apibrėžtos per patį modelį. Dabartinėje *OCL* specifikacijoje nėra aiškios ribos tarp objektinės modeliavimo kalbos ir apribojimų kalbos. Kaip pavyzdys galėtų būti kardinalumo specifikavimas: jį galima specifikuoti panaudojant *UML* klasių diagramų elementus arba *OCL* apribojimus. Tokiu būdu *UML* klasių diagramų elementus galima laikyti paprastesne notacija analogiškos paskirties *OCL* blokams.

Reliacinių duomenų bazių valdymo sistemų apribojimai – tai integralumo apribojimai. Juos būtų galima klasifikuoti taip:

- Numanomieji (angl. *implicit)* – tai integralumo apribojimai, kurie yra duomenų modelio dalis ir turi būti nusakyti konkretiems santykiams
- Išreikštiniai (angl. *explicit)* – tai dažniausiai įvairios veiklos taisyklės
- Būdingieji (angl. *inherent*) – šie apribojimai nenusakomi schemoje, jie yra reliacinio modelio dalis. Juos galima palyginti su *UML* metamodelio apribojimais

Visus apribojimus galima skirstyti:

- Pagal sistemos pjūvius:
    - *Statinis pjūvis* – šiuo atveju apribojimas yra apibrėžiamas kaip invariantas, t. y. sąlyga, kuri turi būti visada teisinga sistemai (ar daliai jos) esant bet kurioje būsenoje
    - *Dinaminis pjūvis* – apribojimas yra sąlyga, leidžianti pereiti sistemai iš vienos būsenos į kitą

- *Funkcinis pjūvis* – apribojimai nusakomi kaip išėjimo reikšmių priklausomybė nuo įėjimo reikšmių būsenos pasikeitimo metu. *OCL* kalboje tokie apribojimai nusakomi *post-* ir *pre-* sąlygomis

- Pagal elgesį pažeidus apribojimus:
  - *Veiksmo* (angl. *operational*) *apribojimai* – automatiškai atstatoma sistemos būsena į prieš tai buvusią
  - *Deklaratyvieji (*angl. *declarative) apribojimai* – konstatuojamas faktas, jog apribojimas pažeistas, nesiimant jokių tolimesnių veiksmų

Taip klasifikavus apribojimus, galima palyginti *OCL* apribojimus ir *RDBVS* integralumo išlaikymo mechanizmus (2.1 lentelė).

2.1 lentelė

OCL ir RDBVS apribojimų palyginimas

	OCL	**RDBVS**
Sistemos pjūvių požiūris	Statiniai, dinaminiai, funkciniai	Statiniai
Apribojimų pažeidimo požiūris	Deklaratyvūs	Deklaratyvūs, veiksmo

Matyti, jog *OCL* ir *RDBVS* apribojimai persidengia. Konkrečiai – statinių ir deklaratyviųjų atveju. Pagrindinis skirtumas tarp *OCL* ir *RDBVS* apribojimų – *OCL* apribojimai laikomi aukštesnio lygio, tuo tarpu *RDBVS* apribojimai laikomi žemesnio lygio, konkretesniais ir nuo konkrečios *RDBVS* realizacijos priklausančiais apribojimais.

2.2 lentelėje bus apibendrintas numanomųjų, išreikštinių ir būdingųjų apribojimų reliaciniuose modeliuose ir *SQL*:2003 standarte semantikos palyginimas.

2.2 lentelė

Numanomųjų, išreikštinių ir būdingųjų apribojimų reliaciniuose modeliuose ir *SQL*:2003 standarte semantikos palyginimas

Integralumo apribojimų tipas	**Reliacinis modelis**	**Specifikacija SQL kalboje**
Numanomieji	Pirminis raktas	PRIMARY KEY (NOT NULL)

	Išorinis raktas	FOREIGN KEY REFERENCES
Išreikštiniai	Atributų arba lentelių apribojimai	CHECK, NOT NULL, UNIQUE, ...
	Sąlygų tikrinimai	CONSTRAINT ... CHECK, CREATE ASSERTION
	Domenai, tipai	CREATE DOMAIN
	Trigeriai	CREATE TRIGGER
Būdingieji	Atominės atributų reikšmės	besąlyginė

Tik trigeriai ir ryšių vientisumo (angl. *referential integrity*) apribojimai su įvykiais (*SET DEFAULT*, *SET NULL*, *CASCADE*) yra veiksmo apribojimai. Visi kiti (įskaitant ir *FOREIGN KEY REFERENCES ... NO ACTION*) yra deklaratyvieji apribojimai.

Dauguma autorių aprašo *OCL-SQL* transformacijas deklaratyviesiems apribojimams. Tuo tarpu veiksmo apribojimų transformacijos yra paliečiamos retai. Galima nagrinėti veiksmo apribojimų transformacijos į *SQL* kodą, naudojantis *OCL pre-* ir *post-* sąlygomis bei *body* operatoriais.

2.3. Sunkumai, iškylantys transformuojant UML/OCL modelius į SQL kodą

Nagrinėjant *UML*/*OCL* apribojimus reliacinėms duomenų bazėms, labai svarbu atkreipti dėmesį į objektinio ir reliacinio modelio skirtumus. Nustačius taisykles, kaip elgtis šių skirtumų atveju, galima pilnai išnaudoti reliacinės algebros galimybes.

Transformavimą iš *OCL* į *SQL* galima aprašyti:

1. *UML*/*OCL* ir reliacinių metamodelių tarpusavio transformacijomis.
2. Transformacijų šablonais skirstant juos į tipinius atvejus.

Pirmojo tipo transformacijos gana ribotos, nes transformacijų rezultatas – deklaratyvūs apribojimai arba tokie apribojimai, kurie iš esmės tiesiogiai turi atitikmenis abiejuose metamodeliuose.

Antrojo tipo transformacijoms literatūroje aprašomi standartiniai transformacijų atvejai, tačiau dauguma jų vienaip ar kitaip turi tam tikrų apribojimų ar problemų:

1. ***Paprastųjų OCL tipų transformacijos***. Paprastieji *OCL* tipai (*real*, *integer*, ir t.t.) tiesiogiai transformuojami į *SQL* tipus. Problemos gali atsirasti tada, kai nėra *SQL* tipų tiesioginių atitikmenų (pvz. *INTERVAL*)..

2. ***OCL invariantų transformacijos***. *OCL* invariantų transformavimas į *SQL* apribojimus gana gerai išnagrinėtas. Jie dažniausiai transformuojami į sąlygų tikrinimą (angl. *assertions*). Reikia pastebėti, jog *SQL* išraiškos gali būti įvertinamos net jei gražinama *NULL* reikšmė. Tuo tarpu *OCL* nenusako, kaip dirbama su *OCL* atitikmeniu *Undefined*.

3. ***Klasių ir atributų transformacijos***. Transformacija, kai klasė su atributais tiesiogiai transformuojama į lentelę ir jos atributus, yra gana gerai išnagrinėta. Tačiau čia iškyla dvi rimtos problemos, susijusios su objektinio ir reliacinio modelių nesutapimu:

    a. *Objektų identifikacija*. Objekto identifikatoriaus sąvoka skiriasi nuo dirbtinio lentelės pirminio rakto sąvokos. T. y. objekto identifikatorių gali atitikti aibė lentelės pirminio rakto reikšmių. Be to, ši aibė gali kisti. Todėl nagrinėjant įvairius transformacijų atvejus, labai svarbu įvertinti šį neatitikimą.

    b. *Objektų palyginimo problema*. Čia reiktų išskirti objektų reikšmių palyginimą ir pačių objektų palyginimą. T. y. objektai nebūtinai yra lygūs, jei jų reikšmės lygios. Todėl jei turime *OCL* išraišką `object = (object2 : OCLAny): Boolean`, tai reiškia, jog po transformacijos reliacinėje duomenų bazėje turės būti lyginamos objektus atitinkančių pirminių raktų reikšmės.

4. ***OCL navigavimo problemos***. Dažnai apribojimai nesibaigia vieno objekto ribomis. T. y. tenka kreiptis per asociacijas į kitus objektus, kad būtų galima pilnai nusakyti norimą apribojimą. Šios transformacijos gali būti realizuojamos subužklausų pagalba. Tačiau kai kurias situacijas galima spręsti tik tokiomis sudėtingomis užklausomis, kurias palaiko labai nedidelė dalis šiandieninių *RDBMS*.

5. ***OCL Collection tipai***. *Collection* objektus išgauti galima nusakant juos kaip išraiškas, naudojant navigaciją arba *allInstances* operaciją. *SQL* palaiko ir tam tikras konstrukcijas kaip *distinct*, kuriomis galima konstruoti *bag* ir *sequence* tipo objektus.

Visi paminėti atvejai transformuojasi į įprastus *SQL* sakinius. Lieka *OCL* konstrukcija *iterate*, kuri yra per daug bendra ir kur kas sudėtingesnė nei paminėti atvejai/operacijos. Todėl apribojimams, kuriuose naudojama *iterate* konstrukcija, išreikšti reikalinga pilnavertė kalba. *RDBMS* atveju, tokios kalbos galėtų būti serverio procedūroms programuoti naudojamos kalbos. *SQL:2003* standartas numato standartinę kalbą serverio procedūroms ir vienas iš šio darbo tikslų būtų panagrinėti serverio procedūrų generavimą minėtam atvejui.

Apibendrinus *OCL* apribojimų transformacijas į *SQL* apribojimus ir užklausas, visiems transformacijų variantams yra būdingos šios problemos:

1. *Sugeneruotų SQL sakinių sudėtingumo problema*: norint išvengti serverio procedūrų generavimo, kartais būtina generuoti tokio sudėtingumo *SQL* sakinius, kuriuos palaiko labai nedaug *RDBMS* serverių.

2. *Kai kurių OCL išraiškų užrašymo sudėtingumas*. Užrašytos *OCL* išraiškos gali būti labai sudėtingos ir tai labai apsunkina jų tipinių atvejų atpažinimą bei generavimą į *SQL* kodą. Sakykime, unikalaus atributo nurodymas *SQL* sakiniu atrodytų taip:

```
create table ACCOUNT (..,ACCOUNTNUMBER varchar not null unique,..)
```

Tuo tarpu *OCL* kalba tai aprašoma tokia išraiška:

```
[context Account]
Account.allInstances -> forAll ( a1, a2 |
a1<>a2 implies a1.accountNumber<>a2.accountNumber )
```

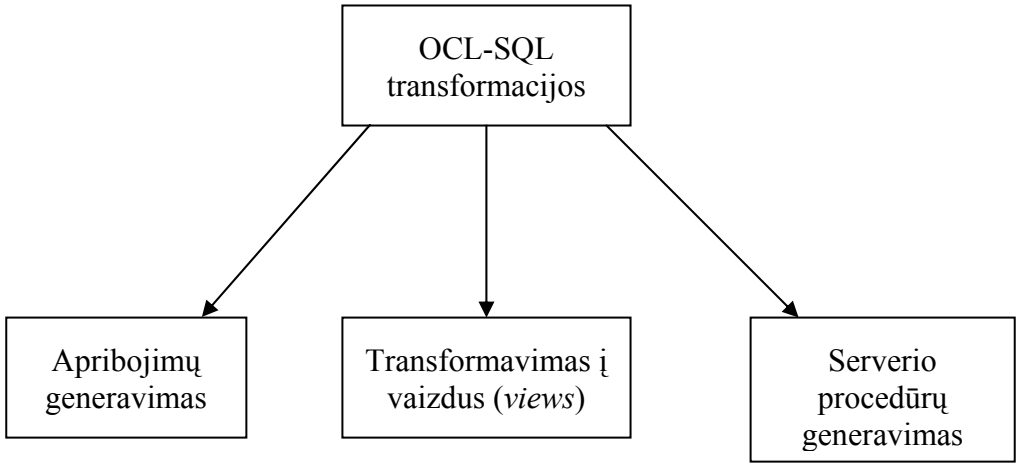
Dalis šių problemų jau sprendžiamos, išleidžiant naujas *OCL* versijas. Pvz., *OCL* 2.0 standarte įvestas operatorius *isUnique*:

```
[context Account]
Account.allInstances -> isUnique (accountNumber)
```

Kai negalima išspręsti išraiškų sudėtingumo problemų minėtu būdu, reiktų panagrinėti galimybes išplėsti *UML* stereotipų aibę stereotipais konceptualiam apribojimų modeliavimui. Tuomet, generuojant duomenų modelį, tie apribojimai atsirastų ir fizinėje duomenų bazėje. Reikia atkreipti dėmesį, jog tokiu atveju apeinama *OCL* kaip apribojimų kalba, nors tai ir išsisprendžia aprašytų atvejų sudėtingumo problemą.

2.4. Transformacijoms naudojami programiniai paketai ir srities esminės problemos

Galima išskirti esmines problemas, generuojant *OCL* apribojimus ir išraiškas į *SQL* apribojimus bei sakinius. Sritys, kuriose bus dirbama magistro darbo metu, pavaizduotos 2.1 pav.



2.1 pav. OCL-SQL transformacijų klasifikacija

Transformuojant *OCL* apribojimus į *SQL* apribojimus išskirtinos šios problemos:

- *OCL* invariantų transformavimo į sąlygų tikrinimą problemos
- Klasių ir atributų transformacijos – objektų identifikacijos ir objektų palyginimo problemos
- *OCL* navigavimo problemos
- *OCL* konstrukcijų, tokių kaip *iterate* transformacijos į ekvivalentų *SQL* kodą
- Sugeneruotų *SQL* sakinių sudėtingumo problemos bei *OCL* išraiškų užrašymo sudėtingumo problemos

Transformuojant *OCL* išraiškų sekas į vaizdus (*views*) esminės problemos:

- Vaizdų aprašymo modeliai sudėtingi ir labai priklausantys nuo konkrečios situacijos (apibendrinimo trūkumas)
- Nėra apibrėžtas darbas su agregatais ir grupavimo konstrukcijomis
- Nėra apibrėžtas darbas su subužklausomis
- Nėra apibrėžtas darbas su lentelių rezultatų apjungimu (*SQL* sakinys *UNION*)

Išanalizavus ir išsprendus bent dalį minėtų problemų, būtų galima nagrinėti serverio procedūrų generavimą iš *OCL*. Vienas iš pagrindinių darbo tikslų – pasiekti bent dalinį serverio procedūrų generavimą. Tam reikėtų ne tik išspręsti bent dalį minėtų problemų, bet ir parodyti, jog *OCL* turi ne mažesnes išraiškos galimybes nei reliacinė algebra.

Išbandžius nemažą dalį *UML* palaikančių projektavimo įrankių, išsiaiškinta, kad *OCL* apribojimų generavimo į *SQL* kodą kol kas pilnai nepalaiko nei vienas įrankis. Iš dalies *OCL* apribojimų generavimą ir tikrinimą palaiko *MagicDraw* paketas (http://www.magicdraw.com) bei *ArgoUML* paketas (http://www.argouml.org). Abu šiuos paketus galima išplėsti. Dresdeno universitetas išleido paketų rinkinį „*Dresden OCL toolkit*“, skirtą *OCL* apribojimams tikrinti ir generuoti į *SQL* kodą. Jis integruotas į *ArgoUML* paketą ir yra laisvai prieinamas modifikavimui.

Techniškai generavimas iš *OCL* į *SQL* apribojimus, vaizdus ir serverio procedūras vienaip ar kitaip išsprendžiamas. Tačiau esminis dalykas, ko trūksta srityje – bendra metodika, kaip elgtis vienais ar kitais atvejais, kai susiduriama su dviprasmybėmis arba *OCL*-*SQL* nesuderinamumais. Atlikus darbą, tokia metodika tūrėtų būti sudaryta.

Šiame darbe transformacijos bus nagrinėjamos derinant reliacinių ir *UML*/*OCL* metamodelių transformacijas bei tipinių transformacijų šablonus.

3. KONCEPTUALIŲJŲ APRIBOJIMŲ TRANSFORMACIJŲ METODAI

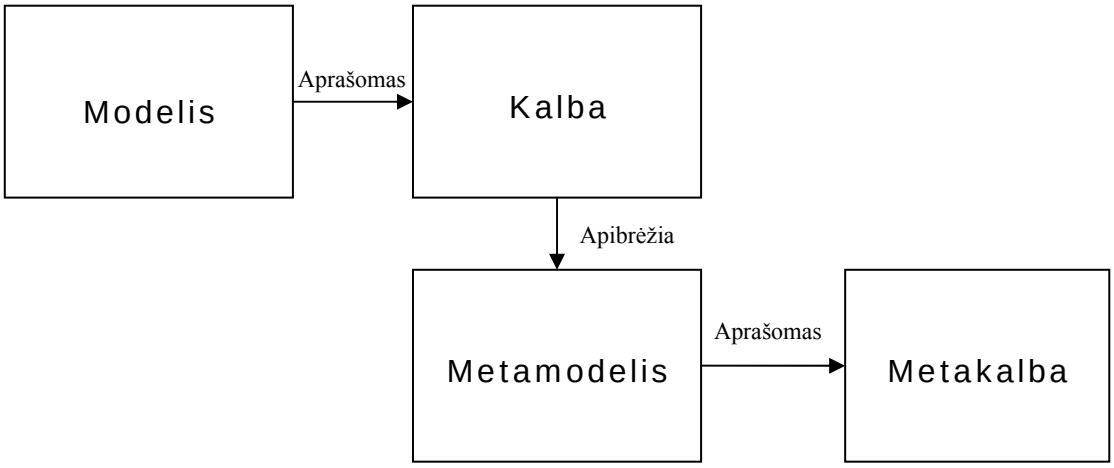
3.1. Konceptualieji apribojimai metamodelių transformacijų metu

3.1.1. OMG metamodelių lygmenys

Modelis aprašo, kokie elementai gali egzistuoti sistemoje. Jei sistemos modelyje aprašome klasę, tai jos egzemplioriai gali būti sukurti ir egzistuoti modeliuojamoje sistemoje. Kalba taip pat aprašo, kokie elementai gali būti naudojami modelyje (pvz. *UML* modeliavimo kalba aprašo konceptus „*Class*“, „*State*“, „*package*“ ir t.t.). Taigi, kalbą galima aprašyti modeliu: jis aprašo elementus, kurie gali būti naudojami kalboje.

Kiekvienas elementas, kurį naudoja projektuotojas, yra apibrėžtas modeliavimo kalbos metamodelyje. *UML* kalboje galima naudoti klases, atributus, asociacijas, būsenas, operacijas ir t.t., nes jie aprašyti *UML* metamodelyje.

Metamodelis yra taip pat modelis, todėl jis taip pat turi būti aprašytas tam tikra kalba. Tokia kalba vadinama metakalba, kuri vėl gali būti aprašyta metametamodeliu (3.1 pav.). Teoriškai toks lygmenų skaičius neribotas, tačiau *OMG* siūlo naudoti 4 lygmenis: *M0*, *M1*, *M2*, *M3*.



3.1 pav. Modelių, kalbų, metamodelių ir metakalbų sąryšis

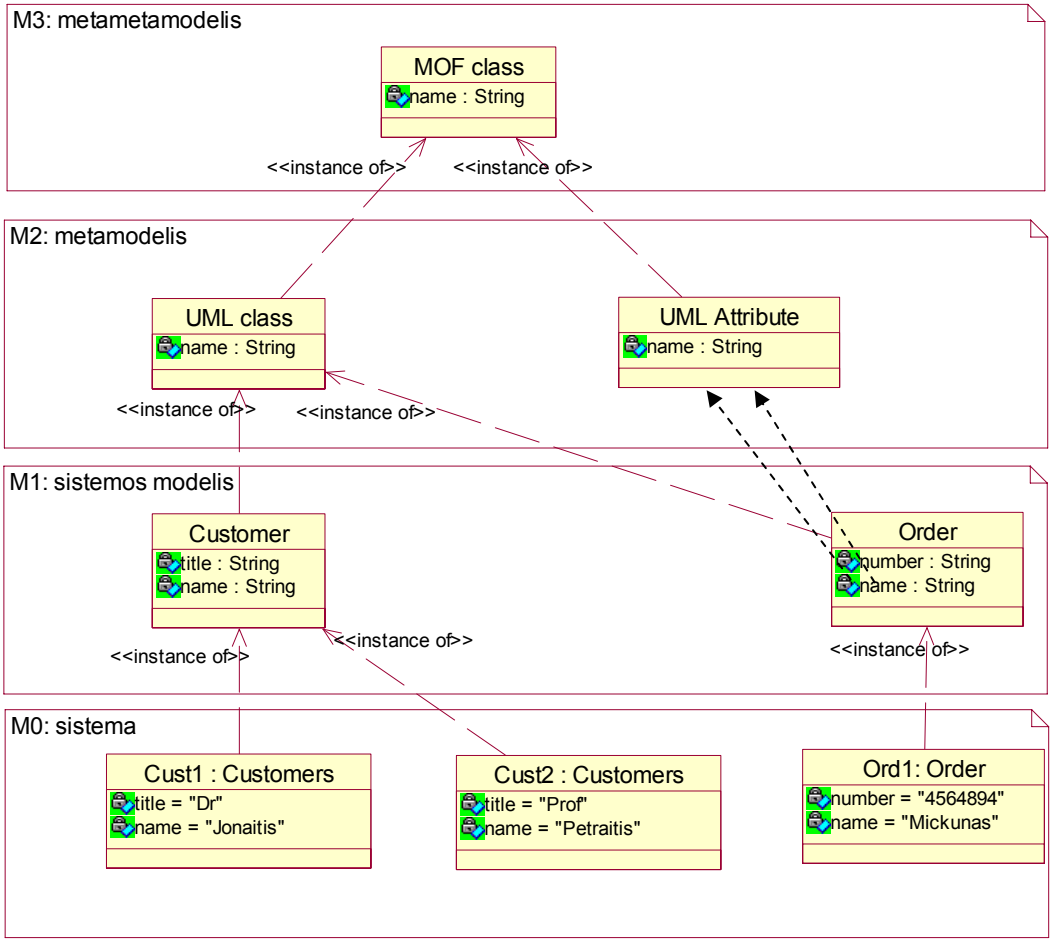
M0 lygmenyje veikia sumodeliuota sistema, t. y. šiame lygmenyje sąveikauja elementų egzemplioriai. Taigi *M0* – egzempliorių lygmuo.

M1 lygmenyje aprašomi modeliai, pvz. kuriamos sistemos *UML* modeliai. *M0* ir *M1* lygmenys susiję. T. y. kiekvienas *M0* lygio elementas yra *M1* elemento egzempliorius.

Elementai, apibrėžiami *M1* lygmenyje yra *M2* lygmens egzemplioriai. Šis ryšys panašus į ryšį tarp *M0* ir *M1* lygmens modelių. *M2* lygmens modelio elementai apibrėžia tokias sąvokas kaip *UML* klasė, atributas, asociacija ir t.t. *M2* lygmens modelis vadinamas metamodeliu.

Dar aukštesnio abstrakcijos lygmens yra *M3* lygmuo. Jame aprašomi elementai, kurių egzemplioriai yra *M2* modelio elementai. M3 modelis apibendrina *M2* modelio elementus. *OMG MOF* kalba yra *M3* lygmens kalba.

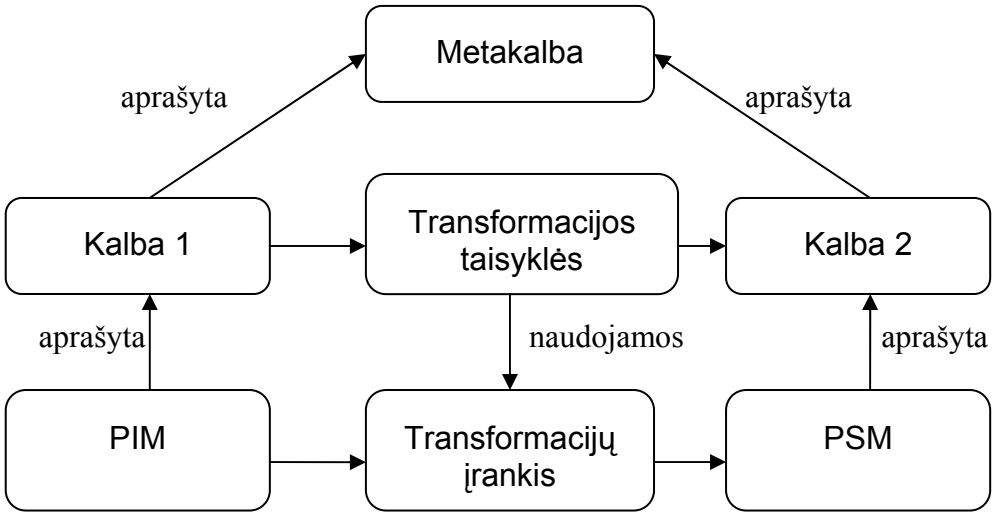
3.2 pav. pavaizduotas *M0, M1, M2, M3* modelių tarpusavio ryšių pavyzdys.



3.2 pav. OMG modelių lygmenys M0, M1, M2, M3

Reikia pastebėti, kad lygmenų skirstymas į *M0*, *M1*, *M2* ir *M3* yra sąlyginis: daugelis sistemų saugo ir metaduomenis, o tai reiškia, kad *M1* lygmens modeliai fiziškai saugomi toje pačioje sistemoje kaip ir *M0* lygmens duomenys. Daugelis sistemų be šių lygmenų saugo ir *M2* lygmens metamodelį. Taigi, beveik visada galima tiesiogiai prieiti prie visų sistemos modelių ir metamodelių duomenų.

MDA (1.1 pav.), papildytas metamodelių lygmeniu, pavaizduotas 3.3 pav.



3.3 pav. MDA, papildytas metamodelių lygmenimis

Modelių transformacijos įrankis gali skaityti, kurti ir suprasti modelius. Transformacijų įrankis naudoja transformacijų taisykles, kurios apibrėžia, kaip modelis vienoje kalboje transformuojamas į modelius kitoje kalboje. Transformacijos atliekamos tarp tų modelių, kurių kalbos apibrėžtos tų pačių lygmenų metamodeliais (šiame darbe nagrinėjamu atveju – *MOF*).

Transformacijos gali būti suskirstytos į dvi stambias grupes:

- *Endomorfinės* arba kitaip tariant *vertikalios* transformacijos – šios transformacijos metu veikiama vieno metamodelio lygmenyje, sukuriami nauji elementai. Tokios transformacijos naudojamos tikslinant modelius
- *Egzomorfinės* arba kitaip tariant *horizontalios* transformacijos – šios transformacijos metu veikiama mažiausiai dviejuose metamodeliuose: šaltinio ir tikslo modeliai aprašyti naudojant skirtingus metamodelius. Tokios transformacijos naudojamos transformuojant *PIM* modelius į *PSM* modelius, arba generuojant programinį kodą iš *PSM* modelių

Būtent horizontalios transformacijos bus naudojamos tolimesniame darbe tiriant *UML/OCL* metamodelių transformacijas į *CWM* reliacinį metamodelį. Transformacijos bus atliekamos *M2* lygmenyje, t. y. metamodelių lygmenyje.

3.1.2. Darbe naudojami metamodelių transformacijų tipai ir kalbos

Transformacijų metu reikia susieti kiekvieną šaltinio modelio elementą su generuojamo modelio elementais. Paprasčiausias būdas tai padaryti – susieti šaltinio modelio elemento metaklasę su generuojamo modelio elemento metaklase. Tokioms sąsajoms specifikuoti naudojamos modelių

transformacijų kalbos. Šioje srityje kol kas tėra tokių kalbų pasiūlymai. Iš esmės metamodelių transformacijas galima specifikuoti tokiais būdais:

- Neformaliai – pvz. naudojant šnekamąją kalbą. Šis būdas nors ir labai patogus vartotojui, juo užrašytų specifikacijų negali apdoroti programiniai paketai
- Formaliai – naudojant transformacijų specifikavimo kalbas. Šiuo būdu aprašytas transformacijas gali naudoti modelių transformavimo įrankiai

OMG yra pateikusi užklausimą pasiūlymams tokios kalbos standartui (MOF 2.0 Query/Views/Transformations RFP, ad/2004-04-10, [13]). Viena iš siūlomų yra *TRL* (angl. *Transformation Rule Language, OMG* dokumentas ad/2003-08-05) kalba. Ji skirta atlikti modelių užklausas bei specifikuoti metamodelių transformacijas *MOF* 2.0 bazėje. Ši kalba – *OCL* 2.0 išplėtimas, todėl visos operacijos, veikiančios *OCL* kalboje, veikia ir *TRL* kalboje. *TRL* kalba gana sudėtinga ir sunkiai skaitoma, todėl darbe naudosime paprastesnę transformacijų kalbą.

Darbe naudojamos transformacijų kalbos taisyklė turės elementus:

- Nuorodą į šaltinio kalbą
- Nuorodą į tikslo kalbą
- Neprivalomus transformacijos parametrus, pvz. konstantas
- Aibę elementų iš šaltinio metamodelio, kurie dalyvaus transformacijoje (aibė S)
- Aibę elementų iš tikslo metamodelio, kurie dalyvaus transformacijoje (aibė T)
- Krypties indikatorių, nusakantį ar šaltinio modelis gali būti sugeneruotas ir tikslo modelio
- Šaltinio kalbos sąlygą: invariantą, rodantį, kada galima taikyti transformaciją šaltinio modelio elementams
- Tikslo kalbos sąlygą: invariantą, rodantį, kada galima taikyti transformaciją tikslo modelio elementams
- Susiejimo taisyklių aibę, aprašančią, kaip aibės S elementai siejasi su aibės T elementais

Aprašysime šios kalbos transformacijos taisyklių notaciją.

Kiekviena transformacijos taisyklė prasideda žodžiu *Transformation* ir po jo einančiu transformacijos pavadinimu. Šaltinio ir tikslo kalbos nurodomos tarp skliaustų poros, einančios po transformacijos pavadinimo. Pirmoji kalba yra šaltinio kalba, o antroji – tikslo kalba. Šiame darbe naudosime sutrumpintus kalbų pavadinimus (*UML*, *SQL*). Pvz.:

```
Transformation ClassToTable( UML, SQL ) {

...

}
```

Konkrečios transformacijos parametrai užrašomi bloke, kuris prasideda žodžiu *params*. Parametrų tipai privalo būti iš šaltinio arba tikslo kalbų. Pvz.:

```
params

    setterprefix: String = ‘get’;
    getterprefix: String = ‘set’;
```

Šaltinio ir tikslo kalbos elementai užrašomi atitinkamai po žodžių *source* ir *target*. Modelių elementų tipai privalo būti iš šaltinio ir tikslo kalbų. Pvz.:

```
target

    c: SQL::Column;
    f: SQL::ForeignKey;
```

Transformacijos kryptį aprašo raktiniai žodžiai *bidirectional* arba *unidirecitonal*. *bidirectional* reiškia, kad transformacija galima iš šaltinio kalbos į tikslo kalbą ir atvirkščiai, o *unidirectional* reiškia, kad transformacija galima tik viena kryptimi. Pvz.:

```
bidirectional;
```

Šaltinio ir tikslo kalbos sąlygos užrašomos kaip *OCL* išraiškos, gražinančios *Boolean* tipo reiškmę. Šios išraiškos užrašomos blokuose, prasidedančiuose žodžiais *source condition* ir *target condition.* Išraiškose naudojami elementai turi būti iš šaltinio ir tikslo kalbų, ir jose naudojami turi būti tik tie elementai, kurie apibrėžti *source* ir *target* sekcijose. Pvz.:

```
target condition

    f.value = c and c.type = f.refersTo.value.type;
```

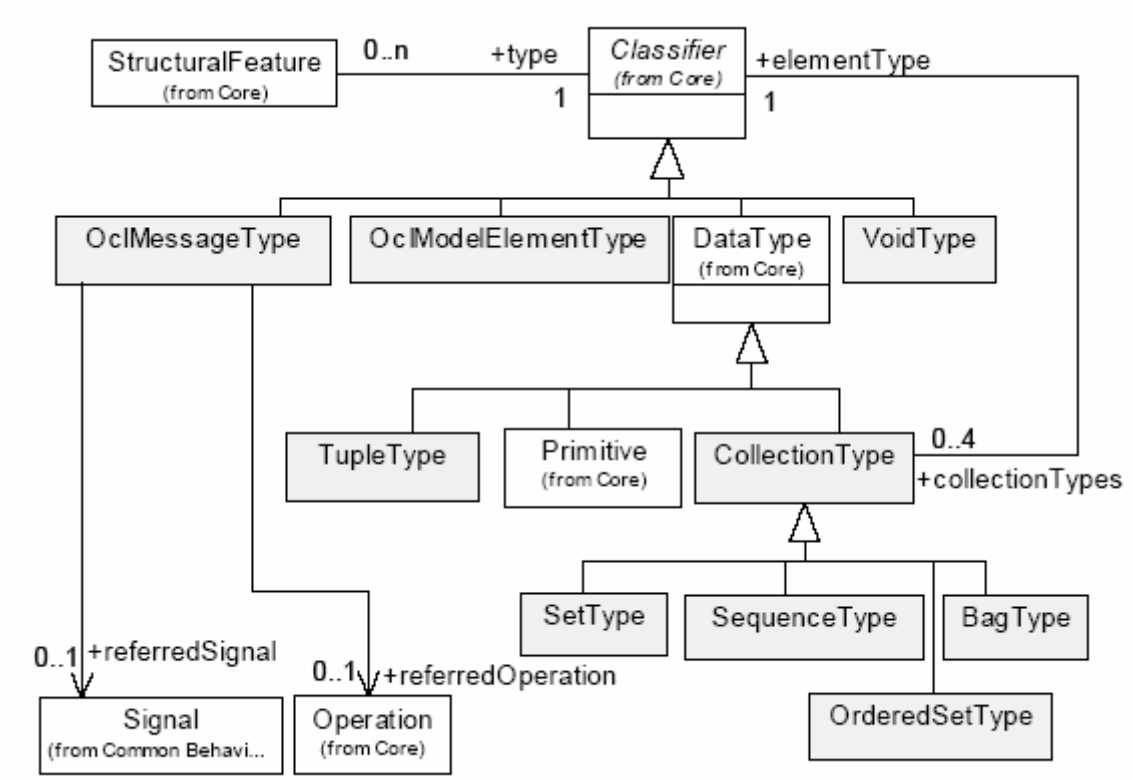
Susiejimo taisyklės aprašomos bloke, prasidedančiame žodžiu *mapping*. Šiame bloke naudojamas <~> simbolis. Neformaliai jis reiškia: „Tam tikra transformacijos taisyklė šaltinio ir tikslo kalbose transformuoja kairėje operatoriaus esantį operandą į dešinėje operatoriaus pusėje esantį operandą“. Jei transformacija - dvikryptė, neformalus paaiškinimas tinka ir skaitant iš dešinės į kairę. Pvz.:

```
mapping

    c.name <~> t.name;
```

mapping operatorius naudojamas ir tais atvejais, kai jo operandai yra aibės.

3.1.3. UML/OCL modelių paprastųjų tipų transformacijos į reliacinių modelių paprastuosius tipus

OCL yra griežtai tipizuota kalba. Kiekviena išraiška gražina tam tikro tipo elementus. Tipai apibrėžiami tiesiogiai, arba nustatomi išraiškos įvertinimo metu. *OCL* metamodelyje apibrėžti tipai pavaizduoti 3.4 pav. Reikia atkreipti dėmesį, kad pavaizduotų klasių egzemplioriai yra patys tipai (pvz. *Integer*), o ne tipų egzemplioriai (pvz. -1, 0, 1, 2, 3, ...).



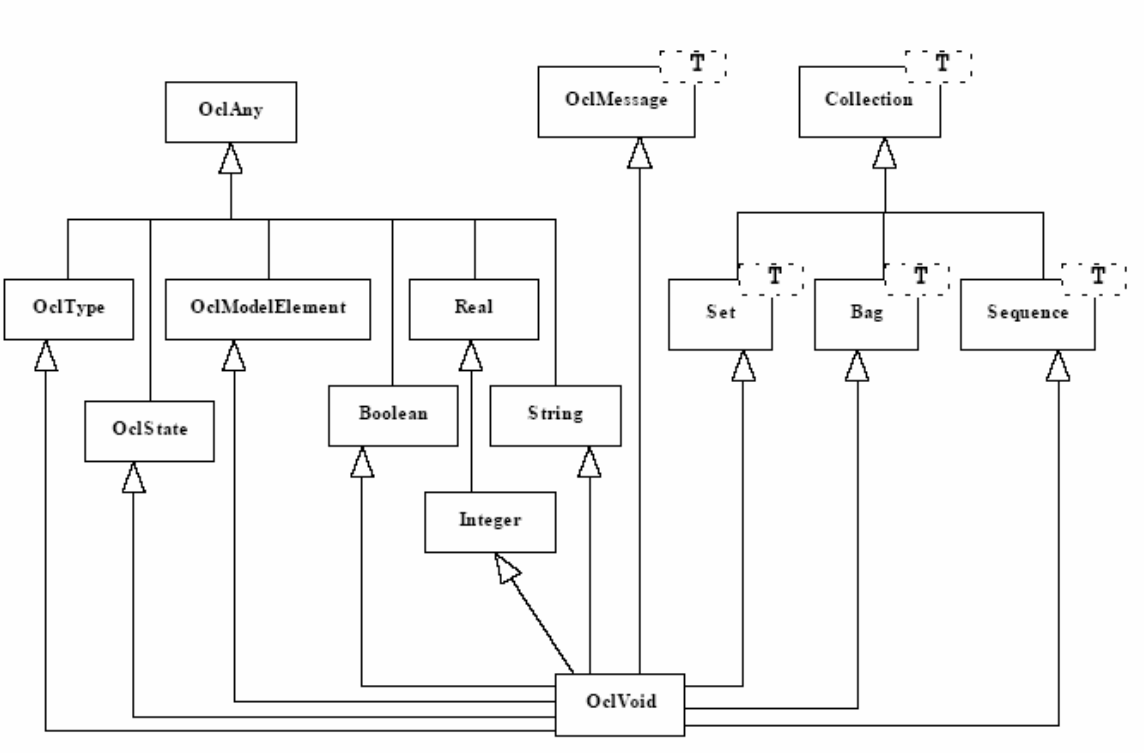
3.4 pav. OCL tipų metamodelis. Abstrakti sintaksė

Bazinis tipas – *UML Classifier*, kuris apibendrina visus tipus, paimtus iš *UML* infrastruktūros.

Abstrakti *OCL* sintaksė neapibrėžia, kokie konkretūs tipai turėtų būti naudojami konkrečioje *OCL* realizacijoje. Konkretūs tipai (3.5 pav.) bei operacijos su jais apibrėžtos *OCL* standartinėje bibliotekoje (angl. *OCL standard library*), kuri įtraukiama į visas *OCL* realizacijas. Šioje bibliotekoje apibrėžti keli paprastieji tipai: *Boolean*, *Real*, *String* ir *Integer*. Standartinė biblioteka apibrėžia ir aibės tipus: *Bag, Set, Sequence* ir *Collection*. Čia *Collection* – abstraktus tipas. *OCL standard library* tipai naudojami *M1* lygmenyje, o abstrakti šių tipų sintaksė apibrėžiama *M2* lygmenyje.

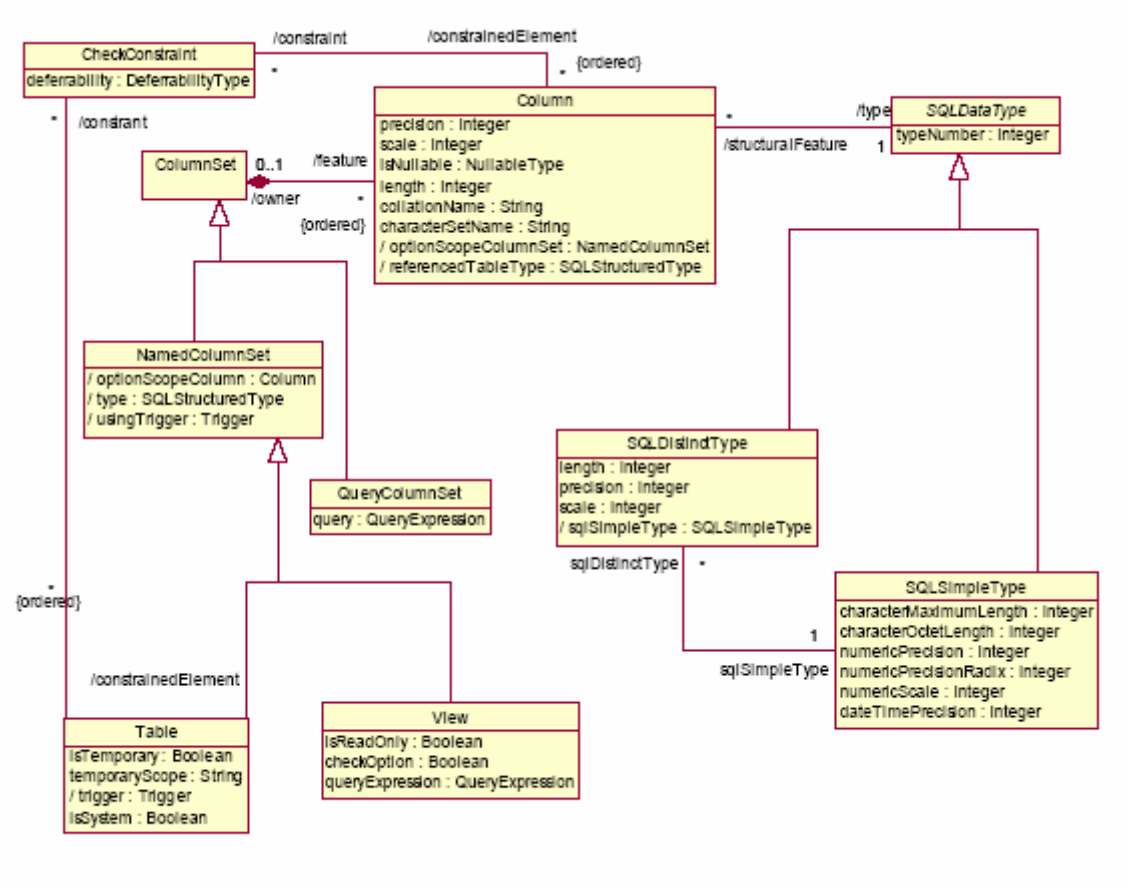
Paprastieji tipai (*Boolean*, *Real*, *String* ir *Integer*) yra UML klasės *Primitive* egzemplioriai. *Real* tipo skaičiai atitinka matematinės realiųjų skaičių aibės elementus. Reikia pastebėti, kad *Integer* klasė yra *Real* subklasė, todėl visur, kur galima naudoti *Real*, ten galima naudoti ir *Integer* egzempliorius. Pati *Real* klasė yra metatipo *Primitive* egzempliorius. *Integer* klasės egzemplioriai atitinka

matematinės sveikųjų skaičių aibės elementus. *String* klasės egzemplioriai yra eilutės, kurių simbolius sudaro *ASCII* arba *Unicode* simboliai. *Boolean* tipo egzemplioriai – tai *True/False* reikšmės.



3.5 pav. OCL Standard Library bibliotekoje aprašyti tipai

Paprastųjų tipų transformacijai naudosime transformacijų kalbą, aprašytą skyrelyje 3.1.2. Paprastuosius tipus atitinkančios klasės *OCL* metamodelyje bus transformuojamos į *OMG Common Warehouse Metamodel* standarto (*CWM 1.1, ad/03-03-02*) pateikiamo reliacinio metamodelio paprastuosius tipus atitinkančias klases.



3.6 pav. Reliacinis metamodelis: lentelės, stulpeliai ir duomenų tipai (CWM 1.1 standartas, ad/03-03-02)

CWM standarte paprastuosius *SQL* tipus atitinka reliacinio metamodelio (3.6 pav.) *SQLDataType* ir *SQLSimpleDataType* klasės. Šios klasės tiesiogiai neapibrėžia konkrečių *SQL* duomenų tipų – jos apibrėžia tik tų tipų savybes, o šių klasių egzemplioriai atitinka konkrečius tipus konkrečioje *SQL* kalbos realizacijoje. Šiame skyrelyje bus pademonstruotos paprastųjų tipų transformacijos į *SQL:2003* standarto paprastuosius tipus.

3.1 lentelėje pavaizduoti *OCL 2.0* ir *SQL:2003* atitinkami paprastieji tipai.

3.1 lentelė

OCL ir SQL standartų paprastųjų tipų palyginimas

OCL	SQL:2003
Boolean	BIT
Real	{ NUMERIC [(n [, m])] } | { DECIMAL [(n [, m])] } | { DEC [(n [, m])] } | { FLOAT (n) } | REAL | { DOUBLE PRECISION }

String	{ CHARACTER [(n)] } | { CHAR [(n)] } | { CHARACTER VARYING [(n)] } | { CHAR VARYING [(n)] } | { VARCHAR [(n)] }
Integer	INTEGER | INT | SMALLINT

BIT tipo reikšmių aibė yra 0 ir 1. Todėl *OCL* tipas *Boolean* su *True/False* reikšmėmis tiesiogiai atitinka *BIT* tipą *SQL* standarte.

OCL Real tipą *SQL* standarte gali atitikti daugiau nei vienas tipas (parametras *n* reiškia, kiek skaitmenų iš viso saugoma, o *m* reiškia, kiek skaitmenų saugoma po kablelio; jei *n* nenurodytas – imama reikšmė pagal nutylėjimą kiekvienai realizacijai; jei *m* nenurodytas – imama reikšmė „0“):

- *NUMERIC* tipui galima nenurodyti nei *n*, nei *m* parametrų
- *DECIMAL* ir *DEC* tipai yra tapatūs. Nuo *NUMERIC* šis tipas skiriasi tuo, kad *NUMERIC* apibrėžia realų tikslumą, o *DECIMAL* – minimalų tikslumą
- *FLOAT, REAL ir DOUBLE PRECISION* tipų duomenys saugomi formatu $a10^b$
- *CHARACTER ir CHAR* tipai tapatūs. Nenurodžius eilutės ilgio parametro *n*, imama reikšmė 1. Šie tipai skirti fiksuoto dydžio eilutėms
- *CHARACTER VARYING, CHAR VARYING, VARCHAR* yra tapatūs. Tokių tipų eilutės – kintamo dydžio. Parametras *n* – maksimalus eilutės ilgis. Jo nenurodžius, imama maksimali reikšmė „1“
- *INTEGER ir INT* yra tapatūs tipai, skirti sveikiesiems skaičiams
- *SMALLINT* taip pat skirtas sveikiesiems skaičiams, tačiau saugojimų sveikų skaičių rėžis šiam tipui siauresnis už *INT* tipo rėžį

Transformaciją, kuri metamodelių lygmenyje *UML*/*OCL String* tipo elementus transformuos į *SQL CHAR* tipo elementus:

```
Tranformation StringToCHAR (UML, SQL) {

    params

        length = N
        rdbmsStringTypeNumber = S

    source umlDataType : UML::DataType;

    target
```

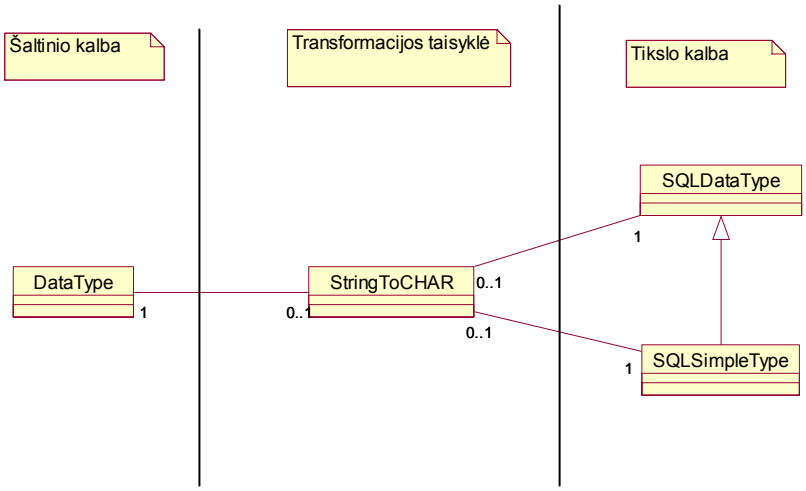
```
        sqlDataType   : SQL::SQLDataType;
        sqlSimpleType : SQL::SQLSimpleType;

    source condition

        umlDataType.oclIsTypeOf( UML::String );

    target condition

        sqlDataType.typeNumber = rdbmsStringTypeNumber and
        sqlSimpleType.characterMaximumLength = length;

    unidirectional;

mapping

    umlDataType <~> sqlDataType;
    umlDataType <~> sqlSimpleType;

}
```

Šios transformacijos metu *UML/OCL* tipas *String* transformuojamas į *SQL* tipą *CHAR*, naudojant 2 parametrus:

1. *length* – tai maksimalus eilutės ilgis. Šis parametras turės būti nurodomas konkrečiame modelyje kaip *OCL* apribojimas.

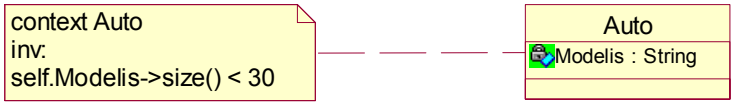
2. *rdbmsStringTypeNumber* – konkrečios *RDBMS String* tipo identifikatorius.

Grafiškai ši transformacija pavaizduota 3.7 pav.



3.7 pav. Grafinė transformacijos *StringToCHAR* interpretacija

Transformacijos *StringToCHAR* taikymo pavyzdys – 3.8 pav.



3.8 pav. Transformacijos *StringToCHAR* demonstravimui skirta klasių diagrama ir OCL apribojimai

3.8 pav. pavaizduota klasių diagrama su klase *Auto*, kurios atributas *Modelis* turi būti ne ilgesnis nei 30 simbolių. Po transformacijų gautas *SQL* kodas atrodytų taip (paryškintu šriftu atvaizduotas transformacijos *StringToCHAR* rezultatas):

```
CREATE TABLE Auto (
  AutoID INT PRIMARY KEY,
  Modelis CHAR(30)
)
```

Galima užrašyti transformaciją, kuri metamodelių lygmenyje *UML*/*OCL Real* tipo elementus transformuos į *SQL DECIMAL* tipo elementus.

```
Tranformation RealToDEC (UML, SQL) {

    params

        numericPrecision = N
        numericScale = M
        rdbmsDecimalTypeNumber = S

    source umlDataType : UML::DataType;

    target

        sqlDataType   : SQL::SQLDataType;
        sqlSimpleType : SQL::SQLSimpleType;

    source condition

        umlDataType.oclIsTypeOf( UML::Real );

    target condition

        sqlDataType.typeNumber         = rdbmsRealTypeNumber and
        sqlSimpleType.numericPrecision = numericPrecision and
        sqlSimpleType.numericScale     = numericScale;

    unidirectional;

mapping

    umlDataType <~> sqlDataType;
    umlDataType <~> sqlSimpleType;

}
```

Šios transformacijos parametrai *numericPrecision* ir *numericScale* tiesiogiai atitinka *SQL DECIMAL* tipo parametrus *n* ir *m* bei priklauso nuo konkretaus paketo realizacijos. Modeliuose šie parametrai tiesiogiai nenurodomi.

Transformaciją, kuri metamodelių lygmenyje *UML*/*OCL Integer* tipo elementus transformuoja į *SQL INT* tipo elementus:

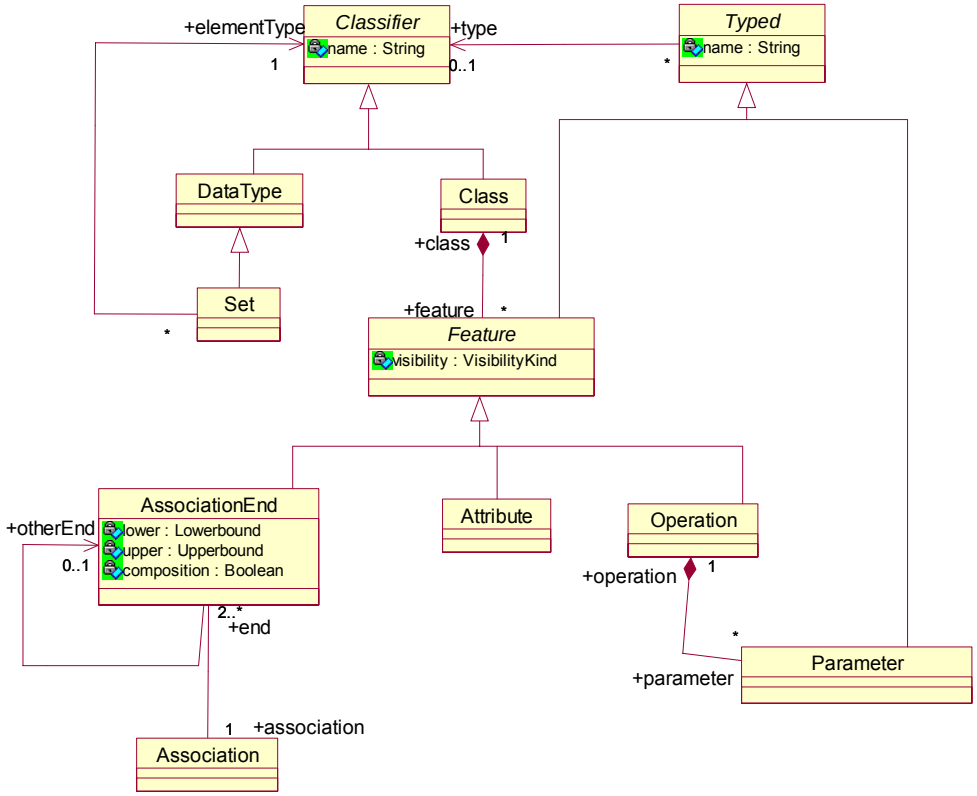
```
Tranformation IntegerToINT (UML, SQL) {

    params

        numericPrecision = N
        rdbmsIntegerTypeNumber = S

    source umlDataType : UML::DataType;

    target

        sqlDataType   : SQL::SQLDataType;
        sqlSimpleType : SQL::SQLSimpleType;

    source condition

        umlDataType.oclIsTypeOf( UML::Integer );

    target condition

        sqlDataType.typeNumber         = rdbmsIntegerTypeNumber and
        sqlSimpleType.numericPrecision = numericPrecision and
```

```
        sqlSimpleType.numericScale     = 0;

    unidirectional;

mapping

    umlDataType <~> sqlDataType;
    umlDataType <~> sqlSimpleType;

}
```

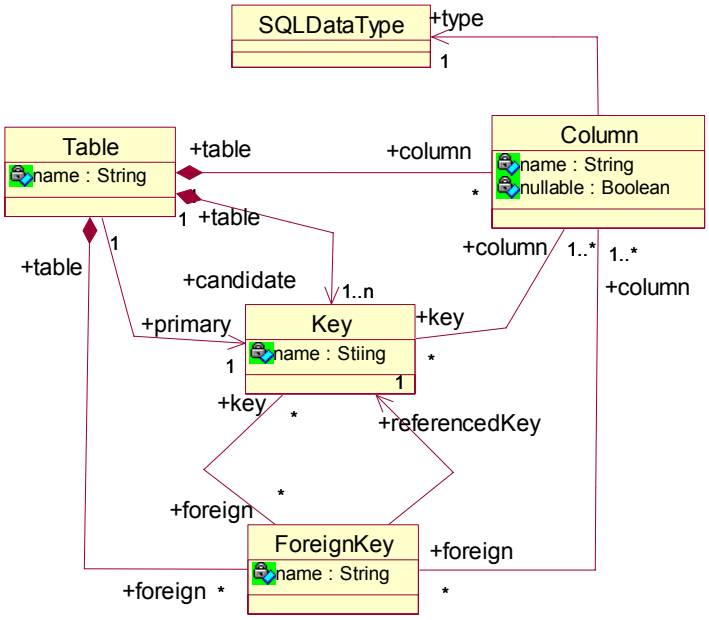
Transformacija, skirta *UML/OCL Boolean* tipo transformacijai į *SQL BIT* tipą yra analogiška ankščiau užrašytoms transformacijoms.

3.1.4. UML modelių transformacijos į reliacinius modelius

Šiame skyrelyje aprašomos *UML* modelių transformacijos į reliacinius modelius metamodelių lygmenyje. Čia bus naudojami, kaip ir 3.1.3 skyrelyje, *CWM* 1.1 standarte pateikiami reliaciniai modeliai, tačiau nebus nepateiktos detalios transformacijų specifikacijos, todėl *UML* metamodeliai bei *CWM* reliaciniai metamodeliai bus supaprastinti (3.9 pav. ir 3.10 pav.).



3.9 pav. Supaprastintas UML metamodelis



3.10 pav. Supaprastintas SQL metamodelis

UML metamodelio transformacijos į reliacinį *CWM* metamodelį seka:

1. Pirmoji transformacija aprašo klasės *Class* (*UML*) transformaciją į klasę *Table* (*SQL*). Šios transformacijos metu klasėje *Table* sukuriamas pirminis raktas, kurio pavadinimas sudaromas iš klasės *Class* pavadinimo ir eilutės „ID“.

```
Transformation ClassToTable (UML, SQL) {

params

    tidName : String = “ID”;
    tidType : SQLDataType = INTEGER;

source

    class : UML::Class;

target

    table   : SQL::Table;
    primary : SQL::Key;
    tid     : SQL::Column;

target condition

    table.primary = primary and
    tid.type      = tidType and
    tid.table     = table and
    tid.key       = primary and
    tid.nullable  = false;

unidirectional;

mapping

    class.name + tidName    <~> tid.name;
    class.name              <~> table.name;
    class.attributes()      <~> table.column;
    class.associationEnds() <~> table.foreign;

}
```

2. Ši transformacijos taisyklė aprašo asociacijų klasės transformavimą į *SQL* lentelę. Ji atliekama kartu su kitomis taisyklėmis, skirtomis transformuoti atributus į stulpelius ir asociacijų galus į išorinius raktus (angl. *foreign keys*).

```
Transformation AssociationClassToTable (UML, SQL) {

source

    assocClass : UML::AssociationClass;

target

    table   : SQL::Table;
    primary : SQL::Key;

target condition

    table.primary = primary;

unidirectional;

mapping

    assocClass.name              <~> table.name;
    assocClass.attributes        <~> table.column;
    assocClass.associationEnds() <~> table.foreign;
    assocClass.end               <~> table.foreign;

}
```

3. Asociacijos galas *UML* kalboje transformuojamas į išorinį raktą *SQL* kalboje. Transformacijos taisyklėje nurodyta sąlyga teigia, jog ši transformacija naudojama tik paprastoms asociacijoms, o ne asociacijų klasėms.

```
Transformation AssociationEndToForeignKey (UML, SQL) {

source

    assocEnd : UML::AssociationEnd;

target

    foreign   : SQL::ForeignKey;

source condition:

    assocEnd.upper = 1 and
    assocEnd.association.oclIsTypeOf( UML::Association )

unidirectional;

mapping

    assocEnd.name <~> foreign.name;
    assocEnd.type <~> foreign.referencedKey;

}
```

4. Asociacijos galo, priklausančio asociacijai su asociacijos klase, transformacija į išorinį raktą.

```
Transformation AssociationClassEndToForeignKey (UML, SQL) {

source

    assocEnd : UML::AssociationEnd;

target
```

```
    foreign   : SQL::ForeignKey;

source condition:

    assocEnd.upper = 1 and
    assocEnd.association.oclIsTypeOf( UML::AssociationClass )

target condition:

    foreign.table.primary.foreign->includes(foreign);

unidirectional;

mapping

    assocEnd.name <~> foreign.name;
    assocEnd.type <~> foreign.referencedKey;

}
```

5. UML atributas tampa SQL stulpeliu.

```
Transformation AttributeToColumn (UML, SQL) {

source

    attr   : UML::Attribute;

target

    column : SQL::Column;

target condition:

    column.nullable = true;

unidirectional;

mapping

    attr.name <~> column.name;
    attr.type <~> column.type;

}
```

6. Paprastųjų tipų bei 1-5 transformacijomis gavome nepilną reliacinį modelį. 6 ir 7 transformacijos užbaigs transformacijos procesą, sugeneruodamos stulpelius išoriniams raktams. Šios transformacijos iš reliacinio modelio sugeneruos išplėstą reliacinį modelį. 6-oji transformacijos taisyklė apibrėžia stulpelių generavimą išoriniams raktams, kurie nėra pirminio rakto dalis. Reikia atkreipti dėmesį, kad užrašas „*column.table* <~> *foreign.table*“ reiškia, jog taisyklės šaltinio ir tikslo lentelės sutampa.

```
Transformation CompleteForeignColumns (SQL, SQL) {

source

    foreign          : SQL::ForeignKey;
    referencedColumn : SQL::Column;
    key              : SQL::Key;

target

    column : SQL::Column;

source condition:

    key.table = foreign.table and
    foreign.referencedKey.column->includes(referencedColumn) and
    not key.foreign->includes(foreign)

target condition:
```

```
    column.nullable = true;

unidirectional;

mapping

    column.name             <~> referencedColumn.name;
    column.table            <~> foreign.table;
    column.type             <~> referencedColumn.type;
    column.foreign->first() <~> foreign;

}
```

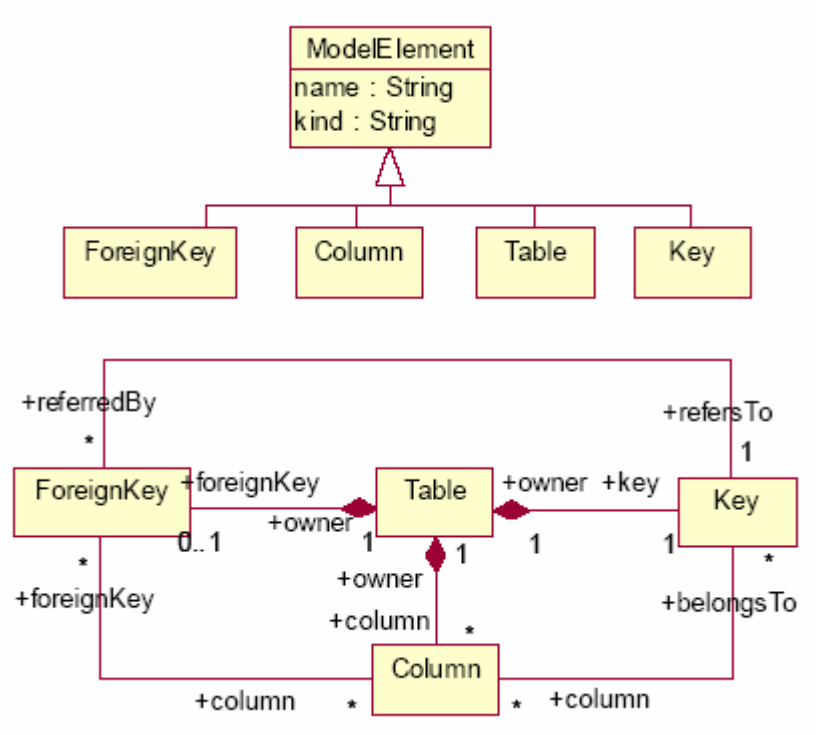
7. Paskutinė transformacija generuoja stulpelius išoriniams raktams, kurie yra pirminio rakto dalis.

```
Transformation CompleteKeyColumns (SQL, SQL) {

source

    foreign          : SQL::ForeignKey;
    referencedColumn : SQL::Column;
    key              : SQL::Key;

target

    column : SQL::Column;

source condition:

    key.table = foreign.table and
    foreign.referencedKey.column->includes(referencedColumn) and
    key.foreign->includes(foreign)

target condition:

    column.nullable = true;

unidirectional;

mapping

    column.name             <~> referencedColumn.name;
    column.table            <~> foreign.table;
    column.type             <~> referencedColumn.type;
    column.key              <~> key;
    column.foreign->first() <~> foreign;

}
```

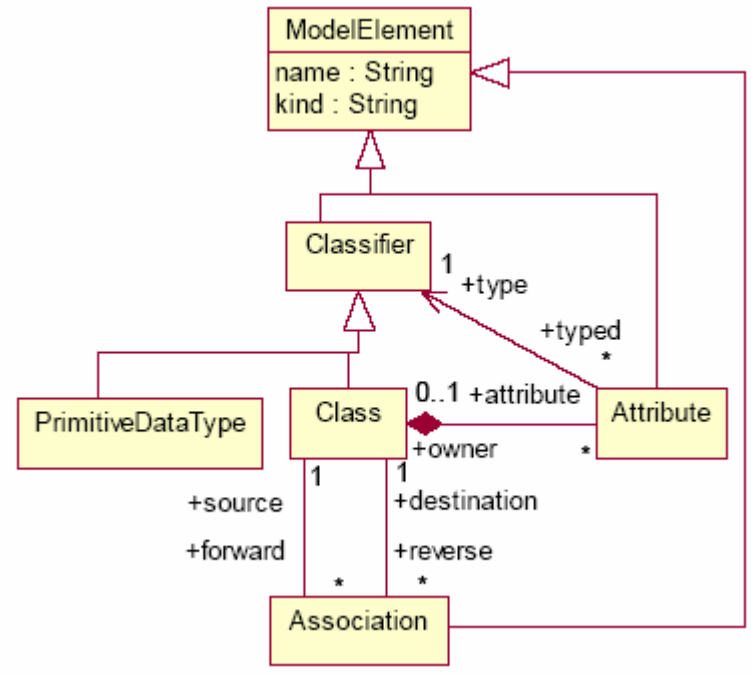
3.1.5. UML modelių transformacijos į reliacinius modelius (TRL variantas)

Šiame skyrelyje pateiktame pavyzdyje bus pademonstruota *TRL* kalba atlikta transformacija analogiška 3.1.4 skyrelyje pateiktai. Reikalavimai šiai transformacijai:

- Klasės atvaizduojamos į lenteles
- Paprastieji klasės atributai atvaizduojami į lentelių stulpelius
- Sudėtiniai atributai išskaidomi į paprastuosius ir atvaizduojami į lentelių stulpelius
- Asociacijos atvaizduojamos į išorinius raktus (angl. *foreign keys*) asociacijų pradžiose
- Išoriniai raktai siejami su tų lentelių pirminiais raktais, kurios atitinka asociacijų galus



3.11 pav. Supaprastintas RDBMS metamodelis (TRL variantui)



3.12 pav. Supaprastintas UML metamodelis (TRL variantui)

TRL specifikacija:

```
transformation Uml2Rdbms;
use modeltype SimpleRDBMS, SimpleUML;
purpose create dbmsmodel:SimpleRDBMS from umlmodel:SimpleUML;

configure modeltype SimpleUML {
  intermediate class LeafAttribute {name:String;kind:String;}
  derived property Class::leafAttributes : Sequence(LeafAttribute);
}

activator go () {
```

```
  umlmodel.objects()[Class and kind='persistent'].create Table();
  umlmodel.objects()[Association].update Table();
}

rule R1 create Table from Class [kind='persistent'] () {
  init {leafAttributes := attribute.create Sequence(LeafAttribute)("","");}
  name := 't_'+name;
  column := leafAttributes.create „ordinary“ Column(„“);
  key := self.create Key();
}

rule R2 create Sequence(LeafAttribute) from Attribute
  (prefix:String="",outerkind:String="") {
  init {
    var k := if outerkind="" then self.kind else outerkind endif;
    result := if type.isKindOf(PrimitiveDataType)
    then self.create LeafAttribute(prefix+name,k).asSequence()
    else type.attribute.create Sequence(LeafAttribute)(name+"_",k)
    endif;
  }
}

rule R3 create LeafAttribute from Attribute (n:String,k:String) {
  attr := self; name := n; kind := k;
}

rule R4 create Key from Class () {
  name := 'k_'+name;
  column := leafAttributes[kind='primary'].resolve(Column);
}

rule R5 create „ordinary“ Column from LeafAttribute (prefix:String) {
  name := prefix+name;
  kind := kind;
  type := if attr.type.name='int' then 'NUMBER' else 'VARCHAR' endif;
}

rule R6 create Table from Association () {
  pre {self.source.kind = 'persistent'
       and self.destination.kind = 'persistent';}
  init { result := self.destination.resolveone(Table); }
  foreignKey := self.create ForeignKey();
  column += result.foreignKey.column;
}

rule R7 create ForeignKey create Association () {
  name := 'f_' + name;
  refersTo := source.resolveone(Table).key;
  column := source.leafAttributes[kind='primary']
  .create „foreign“ Column(source.name+'_');
}

rule R8 create „foreign“ Column from LeafAttribute (prefix:String)
  inherits „ordinary“ Column {
  kind := „foreign“;
}
```

3.2. Tipiniai transformacijų šablonai

3.2.1. UML OCL transformacijų į vaizdus metodai

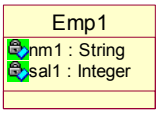
Duomenų bazių vaizdai turi labai glaudžias sąsajas su užklausomis. Juos galima laikyti užklausomis su priskirtu joms pavadinimu. Vaizdams generuoti iš *OCL* siūlomi keli metodai, kuriuos bendrai galima būtų suskirstyti taip:

1. *OCL* kalbos išplėtimai. Šie metodai labai sudėtingi ir turi labai ribotą praktinę reikšmę.
2. Jei lygintume objektinį ir reliacinį modelius, tai duomenų bazių vaizdus objektiniame modelyje galima atvaizduoti paveldėtomis klasėmis. Šiam metodui nereikalingi *OCL* išplėtimai.

Šiame darbe aprašyti du metodai vaizdų generavimui: naudojant išvestines klases bei naudojant *OCL „derive“* išraiškas. Jiems nereikia *OCL* išplėtimų.

3.2.2. UML OCL transformavimas į vaizdus naudojant išvestines klases

Sakykime, jog apibrėžta klasė *Emp1* (3.13 pav.), kurios atributai *nm1* ir *sal1* atitinka darbuotojo vardą ir jo atlyginimą.



3.13 pav. Klasė *Emp1*

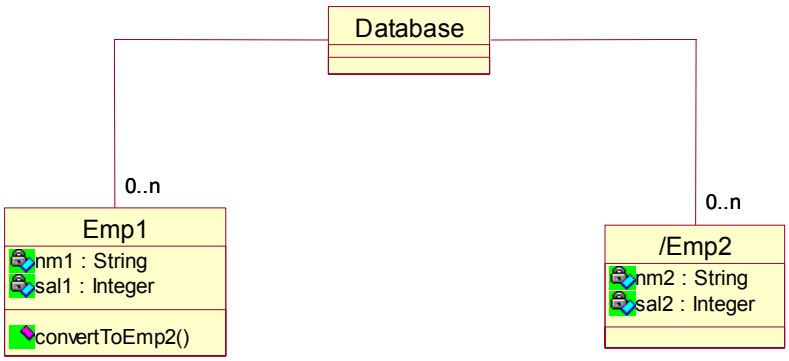
3.13 pav. diagramą galima papildyti klase *Emp2*, kurios objektai pilnai paveldimi iš klasės *Emp1* objektų. Klasės *Emp2* atributai – *nm2* ir *sal2*. Teigsime, jog kiekvieną objektą *e1:Emp1* atitinka toks objektas *e2:Emp2*, kuriam:

*e2.nm2 = e1.nm1 ir e2.sal2 = e1.sal1*2.*

Iš esmės *Emp2* egzempliorių aibė yra tapati klasės *Emp1* egzempliorių aibei, kuriai pritaikyti tam tikri skaičiavimai arba operacijos. Todėl galima teigti, jog *Emp2* yra *Emp1* vaizdas (reliacinių duomenų bazių terminais kalbant). Kalbant *UML* terminais, *Emp2* yra paveldėta iš *Emp1* klasė.

Parodysime, kaip galima aprašyti paveldėtą klasę *Emp2 UML/OCL* kalboje, jog šis aprašas tenkintų reliacinių duomenų bazių vaizdų reikalavimus. Pirmiausia reikia įvykdyti reikalavimą, jog *Emp2* egzempliorių aibė yra tapati *Emp1* egzempliorių aibei, kuriai atlikti tam tikri skaičiavimai. Tam reikia įvesti klasę *Database*, kuri jungiama per asociacijas tiek su *Emp1*, tiek su *Emp2* klasėmis.

Database objektas atspindės esamą duomenų bazės būseną, o klasė *Database* visuomet turės tik vieną objektą. Kintamasis *self Database* klasės kontekste žymės esamą nagrinėjamos duomenų bazės būseną. *Database* klasės kontekste galime apibrėžti skaičiavimus, kuriais gaunama *Emp2* egzempliorių aibė iš *Emp1* egzempliorių aibės (*Emp1* egzempliorius imant kaip įėjimo duomenis).



3.14 pav. Emp1, Emp2 klasių ryšys su Database klase

3.14 pav. *Emp2* pavadinimas pažymėtas simboliu „/“, kuris reiškia, jog tai - paveldėta klasė. Klasėje *Emp1* apibrėžtas metodas *convertToEmp2(),* kuris paverčia pasirinktą *Emp1* objektą *Emp2* objektu. Ši operacija gali būti užrašyta *OCL* kalba:

```
context Emp1::convertToEmp2( ): Emp2
post: self.convertToEmp2.nm2 = self.nm1 and
self.convertToEmp2.sal2 = (2*self.sal1)
```

Galima apibrėžti *Emp2* klasės egzempliorių sąsają su *Emp1* klasės egzemplioriais. Tam reikia klasės *Database* kontekste apibrėžti invariantą, kuris nusakys, kaip gauti *Emp2* egzempliorių aibę iš *Emp1* egzempliorių aibės:

```
context Database inv:
self.Emp2 = self.Emp1–> collect(e:Emp1 | e.convertToEmp2) and
Emp1.allInstances = self.Emp1 and
Emp2.allInstances = self.Emp2
```

Tokiu būdu galima užrašyti *Emp2* kaip *Emp1*, su kuriuo buvo atlikti tam tikri veiksmai, tuo pačiu iškeliant sąlygą, jog vieninteliai *Emp1* ir *Emp2* objektai yra tie, kurie gaunami iš asociacijų, susietų su objektu *Database*.

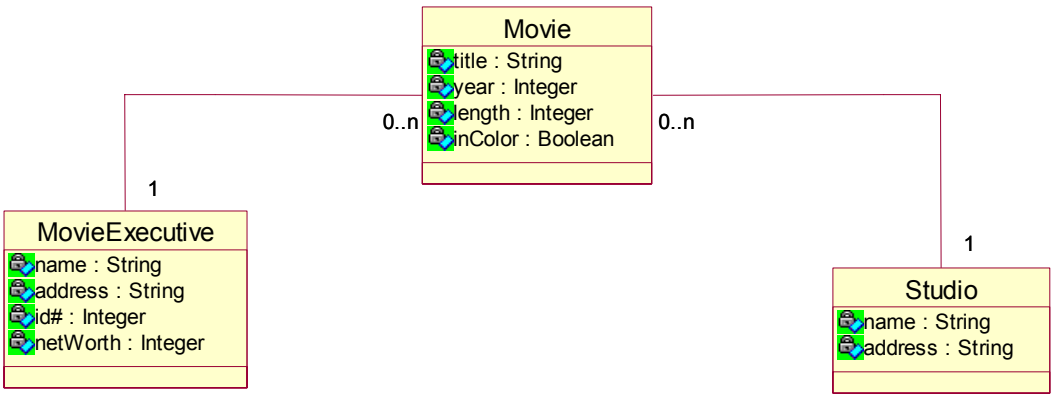
Kai kurie autoriai siūlo *Emp2* aprašyti tiesiogiai, nenaudojant tarpinės klasės *Database*. Šis požiūris klaidingas, nes tokiu būdu aprašyti vaizdai jau nebeatitinka vaizdo sąvokos ir savybių. Sakykime *Emp2* yra duomenų bazės vaizdas, paveldėtas iš *Emp1* klasės. Galima manyti, jog vaizdui apibrėžti pakaks teisingai užrašyti *OCL* apribojimus. Pvz. būtų galima apibrėžti tokius *OCL* apribojimus:

```
context Emp2 inv:
Emp1.allInstances –>
exists(e1 | e1.nm1 = self.nm2 and 2*e1.sal1 = self.sal2)
context Emp1 inv:
Emp2.allInstances –>
Exists(e2 | e2.nm2 = self.nm1 and e2.sal2 = 2*self.sal1)
```

Taip tariamai apibrėžiama, kokį turinį norima gauti *Emp2* klasėje. Šis apibrėžimas nebesiderina su vaizdo apibrėžimu: pademonstruoti apribojimai nusako dvi bazines klases, kurios vieną kitą apriboja. Taigi, *Emp2* šiuo atveju nėra abstrakti klasė, kurios turinys yra paveldėtas iš klasės *Emp1,* atliekant tam tikrus skaičiavimus. Šioje situacijoje *Emp1* gali keisti savo duomenis visiškai nepriklausomai nuo *Emp2* klasės ir atvirkščiai. Taigi, vaizdai turėtų būti realizuojami per abstrakčias klases.

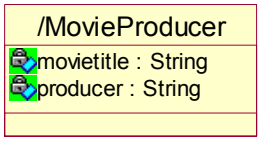
Aukščiau aprašytu principu galima pavaizduoti ir kur kas sudėtingesnį pavyzdį, kuomet vaizdas sudaromas iš kelių klasių.

3.15 pav. pavaizduota filmų klasių diagrama, kuria fiksuojami filmai, juos kūrę prodiuseriai ir studija, kurioje filmai išleisti. Reikia aprašyti vaizdą, *MovieProducer*, iš kurio galima gauti informaciją apie filmų pavadinimus ir jų prodiuserius.



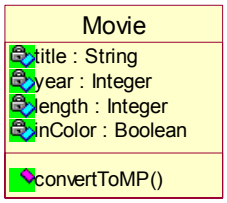
3.15 pav. Filmų pavyzdinė schema

Tam reikia apibrėžti modelio klasę */MovieProducer* (3.16 pav.)



3.16 pav. Klasė /MovieProducer

Kaip ir praeitame pavyzdyje, šią klasę ir *Database* klasę sies asociacija. Įėjimo duomenimis *MovieProducer* klasei bus *Movie* ir *MovieExecutive* klasės. Asociacija tarp *Movie* ir *MovieExecutive* klasių jau yra, jos papildomai aprašyti nereikia. Tokiu būdu pakanka *Movie* objekto tam, kad gauti informaciją apie filmo pavadinimą ir prodiuserį. Klasę *Movie* reikia papildyti operacija *convertToMP()*, kuri paverčia *Movie* objektą į atitinkamą objektą su informacija apie prodiuserį (3.17 pav.).



3.17 pav. Metodu *convertToMP()* papildyta *Movie* klasė

OCL specifikacija šiai operacijai:

```
context Movie::convertToMP( ): MovieProducer
post: self.convertToMP.movietitle = self.title and
self.convertToMP.producer = self.MovieExecutive.name
```

Database klasės invariantas, apibrėžiantis, jog išvestinės klasės *MovieProducer* objektai yra klasės *Movie* objektai, kuriems pritaikytos tam tikros operacijos:

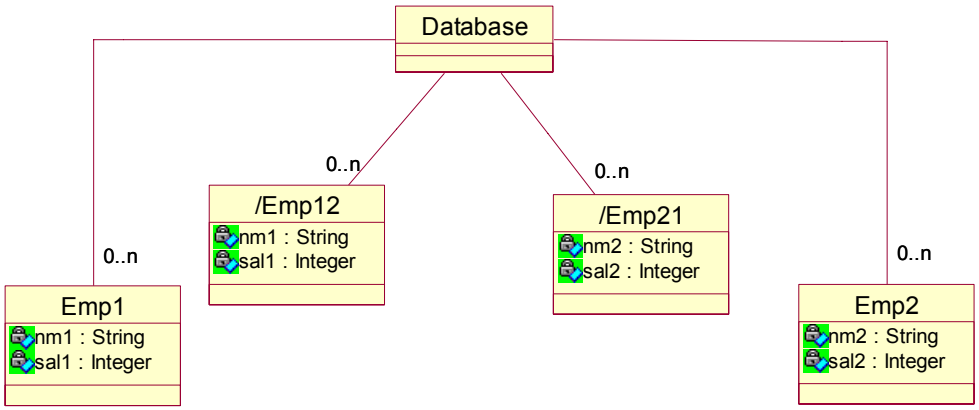
```
context Database inv:
self.MovieProducer = self.Movie->collect(m:Movie | m.convertToMP)
```

Šiame pavyzdyje *Movie* ir *MovieExecutive* klasės tarpusavyje susijusios per asociaciją. Sudėtingesnis atvejis – kai klasės tarpusavyje iš esmės nesusijusios (t. y. kai viena klasė negali būti

pasiekta iš kitos klasės per asociacijų aibę), tačiau reikia sukurti vaizdą iš jų duomenų. Sakykime, modelyje apibrėžtos dvi klasės *Emp1* ir *Emp2*, kurių atributai *nm1* ir *sal1* bei *nm2* ir *sal2* atitinkamai. Reikia sukurti vaizdą (išvestinę klasę), kuri turėtų tuos *Emp1* ir *Emp2* objektus (t. y. tuos darbuotojus), kurių atributų *nm1* ir *nm2* reikšmės vienodos, o vaizde būtų atvaizduojama atributų *sal1* ir *sal2* suma kiekvienam išrinktam darbuotojui. Laikysime, jog *nm1* ir *nm2* nesikartoja *Emp1* ir *Emp2* klasių egzempliorių aibėje. Nagrinėjama klasių diagrama pavaizduota 3.18 pav.

3.18 pav. Tarpusavyje nesusietos asociacijomis klasės *Emp1* ir *Emp2*

Klasių diagramoje reikia apibrėžti dvi pagalbines klases: klasės *Emp12* egzemplioriai bus tie, kurie gali būti susieti su klasės *Emp2* egzemplioriais; klasės *Emp21* egzemplioriai bus tie, kurie gali būti susieti su klasės *Emp1* egzemplioriais. Taip pat reikia apibrėžti klasę *Database*, kuri turės asociacijas su *Emp1*, *Emp12*, Emp2 bei *Emp21* klasėmis (3.19 pav.).



3.19 pav. Klasių diagrama, papildyta *Emp12*, *Emp21* bei *Database* klasėmis

Klasių */Emp12* bei */Emp21 OCL* specifikacijos:

```
context Database inv:
self.Emp12 = self.Emp1 ->
             select(e1:Emp1| self.Emp2 ->
                             exists(e2:Emp2| e1.nm1=e2.nm2))
and
self.Emp21 = self.Emp2 ->
             select(e2:Emp2| self.Emp1 ->
                             exists(e1:Emp1| e1.nm1=e2.nm2))
```

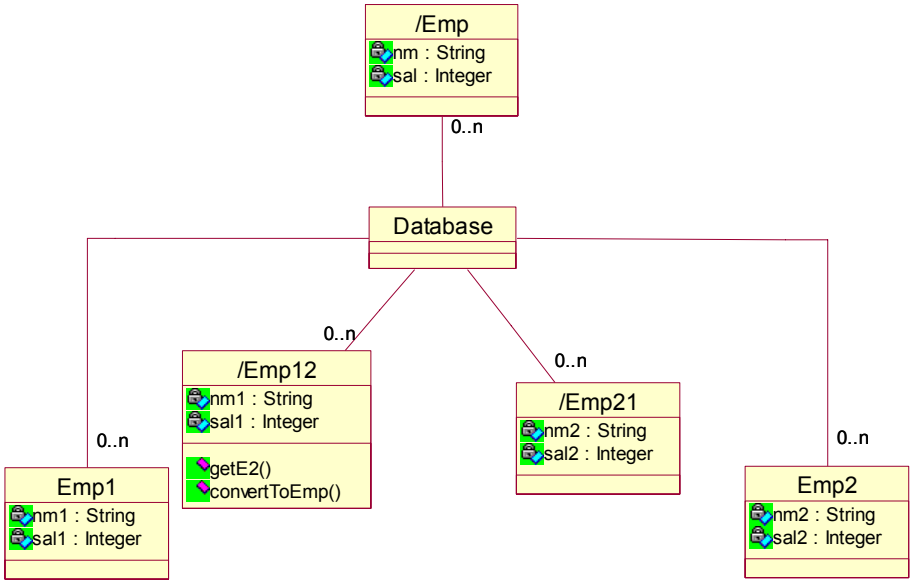
Emp1 ir *Emp2* objektus galima susieti, apibrėžiant operaciją *getE2 Emp12* kontekste:

```
context Emp12::getE2:Emp21
post: self.getE2.nm2=self.nm1
```

/Emp12 kontekste apibrėžus operaciją *convertToEmp*, galima susieti */Emp12* su */Emp*:

```
context Emp12::convertToEmp:Emp
post: self.convertToEmp.nm = self.nm1 and
self.convertToEmp.sal = self.sal1 + self.getE2.sal2
```

Papildyta klasių diagrama pavaizduota 3.20 pav.



3.20 pav. Klasių diagrama, papildyta *Emp* klase

Database klasei priskirtas *OCL* invariantas:

```
context Database inv:
self.Emp = (self.Emp12 ->
                collect(e1:Emp12| e1.convertToEmp))
                        ->asSet
```

Taigi, minėtais būdais pademonstruota, jog naudojant *OCL*, galima aprašyti vaizdus. T. y. buvo pademonstruotos tokios darbo su vaizdais operacijos:

1. Atributų pervardinimas.
2. Skaičiavimai su atributų reikšmėmis.
3. Apribojimai atributams.
4. Vaizdų formavimas iš kelių klasių, kurios tarpusavyje susijusios asociacijų aibe.
5. Vaizdų formavimas iš kelių klasių, kurios tarpusavyje nesusijusios.

Minėtoje metodikoje yra trūkumų:

- Visi pavyzdžiai gana sudėtingi ir labai priklausantys nuo konkrečios situacijos (apibendrinimo trūkumas)
- Nėra apibrėžtas darbas su agregatais, grupavimo konstrukcijomis ir subužklausomis
- Nėra apibrėžtas darbas su lentelių rezultatų sujungimu (*SQL* sakinys *UNION*)

Turint omenyje tai, jog vaizdas gali būti tiesiogiai siejamas su konkrečia *SQL* užklausa, galima teigti, jog išsprendus vaizdų generavimo problemas, būtų galima priartėti prie *SQL* užklausų bei serverio procedūrų generavimo.

3.2.3. UML OCL transformavimas į vaizdus naudojant OCL „derive“ išraiškas

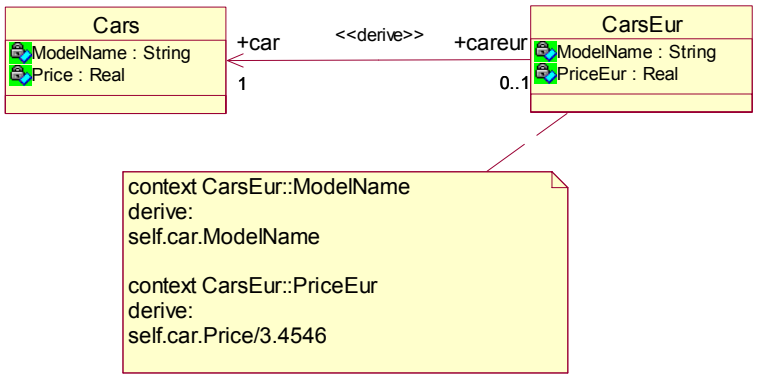
UML 2.0, skirtingai nuo UML 1.x, apibrėžia išvestinio elemento sąvoką – tai elementas, gautas iš kito elemento atliekant kokias nors operacijas su juo. Tokių elementų asociacijoms žymėti *UML* 2.0 siūlo <<derive>> stereotipą (pvz. 3.21 pav.).

3.21 pav. Klasė NewMovies, kurios egzempliorių reikšmės gautos operacijų su Movies klasės egzemplioriais metu

OCL 2.0 taip pat aprašo išraišką išvestinėms reikšmėms aprašyti – *„derive“*.

Šiame skyrelyje bus pademonstruota, kaip naudojant *„derive“* išraiškas, aprašyti vaizdus.

Sakykime, turime klasę *Cars*, kuri aprašo automobilių markes ir jų kainas. *CarsEur* klasė aprašo tų pačių automobilių markes, tačiau kaina apskaičiuota eurais. *CarsEur* visi egzemplioriai išvedami iš klasės *Cars* egzempliorių (3.22 pav.):



3.22 pav. Klasė Cars ir išvestinė klasė CarsEur

Sugeneruotas *SQL* kodas atrodys taip:

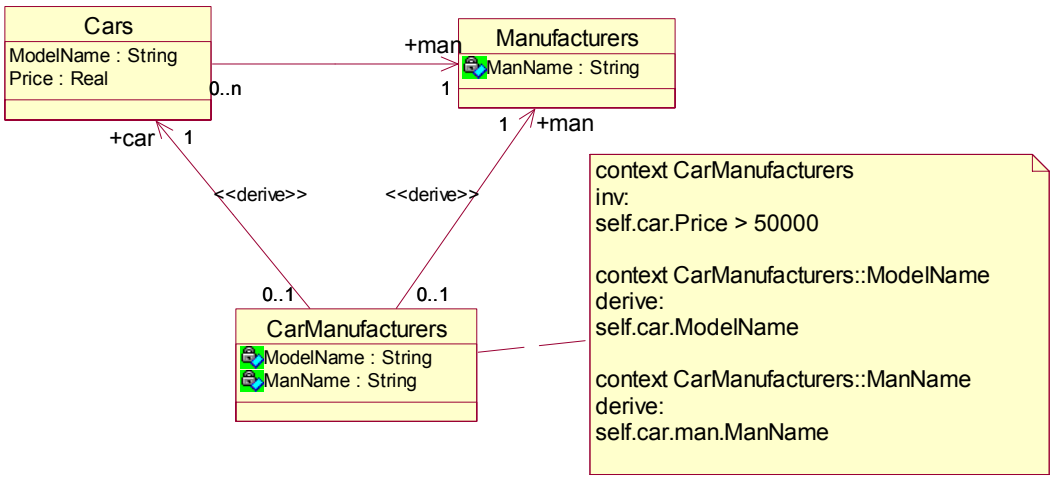
```
CREATE VIEW CarsEur AS
  SELECT
    ModelName AS ModelName,
    Price/3.4546 AS PriceEur
  FROM
    Cars
```

Šiuo pavyzdžiu pademonstruota:

- Atributų pervardijimas
- Skaičiavimas su atributų reikšmėmis

Sekančiame pavyzdyje bus pademonstruotas vaizdų formavimas iš kelių klasių, kurios tarpusavyje susijusios asociacijų aibe. Jame taip pat bus pademonstruoti atributų apribojimai.

3.23 pav. klasė *CarManufacturers* aprašo automobilius, kurių kaina viršija 50000 bei jų gamintojus.

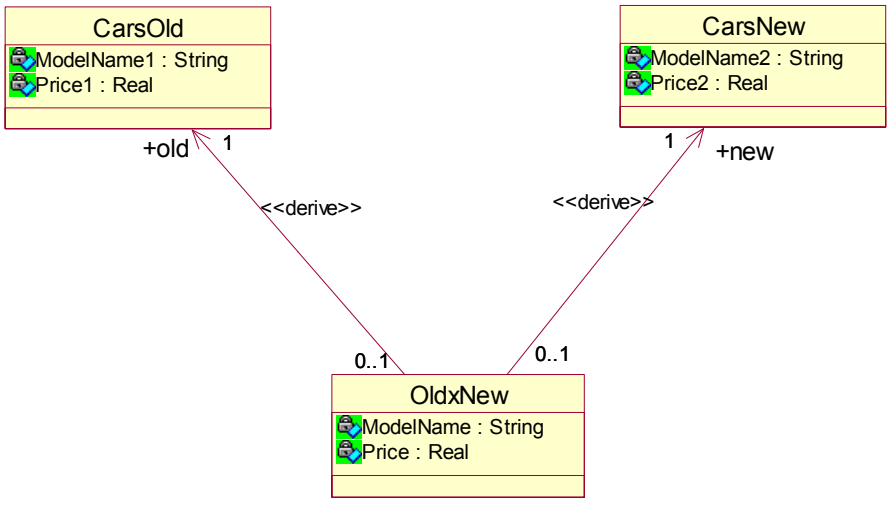


3.23 pav. Klasės Cars, Manufacturers ir išvestinė klasė CarManufacturers

Šios klasių diagramos transformacijos į *SQL* kodą rezultatas:

```
CREATE VIEW CarManufacturers AS
  SELECT
    ModelName AS ModelName,
    ManName AS ManName
  FROM
    Cars
  INNER JOIN
    Manufacturers
  ON Cars.ManufacturersID = Manufacturers.ManufacturersID
  WHERE
    Cars.Price > 50000
```

Sekančiame pavyzdyje bus pademonstruotas vaizdų generavimas iš klasių, kurios tarpusavyje nesusijusios.



3.24 pav. Klasės CarsOld, CarsNew ir jų dekarto sandauga OldXNew

```
context OldxNew::ModelName
derive:
self.old.ModelName1
```

```
context OldxNew::Price
derive:
self.old.Price1 + self.new.Price2

context OldxNew::product(c2: Collection(T2)) :
        Set( Tuple( first: T, second: T2) )
post:
result = self->iterate (e1; acc: Set(Tuple(first: T, second: T2)) = Set{} |
c2->iterate (e2; acc2: Set(Tuple(first: T, second: T2)) = acc |
acc2->including (Tuple{first = e1, second = e2}) ) )

context OldxNew
inv:
self = self.old->product(self->new)
```

Sugeneruoto vaizdo *SQL* kodas:

```
CREATE VIEW OldxNew AS
  SELECT
    CarsOld.ModelName1 AS ModelName,
    CarsOld.Price1 + CarsNew.Price2 AS Price
  FROM
    CarsOld, CarsNew
```

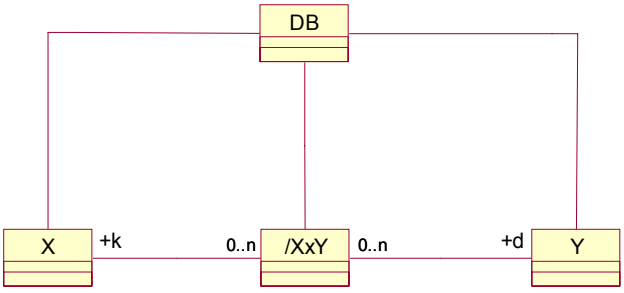
Taigi, šiame skyrelyje pademonstruotos tas pačias vaizdų generavimo galimybes kaip ir 3.2.2 skyrelyje, tik čia nenaudojama išvestinės klasė, o naudojamasi galimybėmis, atsiradusiomis *OCL* 2.0 kalboje. Reikia atkreipti dėmesį, jog naudojant *„derive“* išraiškas, *OCL* invariantai užrašomi paprastesnėmis ir aiškesnėmis konstrukcijomis, todėl paprastesnis ir su šiomis išraiškomis susietų transformacijų medžių sudarymas SQL kodo generavimui (3.2.5 skyrelis).

3.2.4. Reliacinė algebra ir OCL

Kad būtų galima realizuoti vaizdų transformaciją (aprašytą 3.2.2 bei 3.2.3 skyreliuose) iš *OCL* į *SQL* bazę, reikia apibrėžti pagrindines reliacinės algebros operacijas ir jų atitikmenis modeliuose.

Viena svarbiausių reliacinės algebros operacijų – *Dekarto sandauga*. Jei turime aibę X ir aibę Y, tai šių aibių Dekarto sandauga bus aibė, sudaryta iš visų X elementų ir visų Y elementų kiekvienam elementui iš X aibės. Ši operacija žymima XxY. Atvaizduosime Dekarto sandaugą *UML*/*OCL* atveju.

Sakykime, turime klases X ir Y, kurioms reikia apibrėžti Dekarto sandaugą. Dekarto sandaugos rezultatui reikia sukurti išvestinę klasę /XxY, taip pat klasę DB, turinčią asociacijas į klases X, Y bei /XxY (3.25 pav.).



3.25 pav. Dekarto sandaugos demonstracija

Klasei DB aprašytos dvi *OCL* operacijos: *prod* ir *PROD*. *prod* sukuria vieną kortežą iš dviejų kortežų. *PROD* gražina X ir Y klasių Dekarto sandaugos rezultatą.

```
context DB::prod(e:X, e':Y): XxY
post : prod(e,e').k = e and
       prod(e,e').d = e'

context DB::PROD( ): Set(XxY)
post : result = (self.X ->
                 collect(e:X | self.Y ->
                               collect(e':Y| prod(e,e')))
                 ->asSet
```

Tam, kad užtikrinti, jog visi klasių X ir Y kortežai įtraukiami į Dekarto sandaugą, klasei DB reikia aprašyti invariantą:

```
context DB inv:
self.X = X.allInstances and
self.Y = Y.allInstances and
self.XxY = XxY.allInstances and
self.XxY = self.PROD
```

Tokiu būdu parodyta, kaip realizuoti Dekarto sandaugą *UML OCL* atveju. Dekarto sandaugos dalinis atvejis – *NATURAL JOIN* operacija *SQL* kalboje. Taigi, pasirėmus aukščiau esančiu aprašymu, jį galima praplėsti šia *OCL* išraiška (kiti *JOIN* operacijos tipai gali būti aprašyti analogiškai):

```
X-JOIN-Y =
  XxY ->
  select(t : XxY| (t.k.attributes intersect(t.d.attributes)) ->
                   forall(d :String| t.k.d=t.d.d))
```

Apskritai, kad kalba turėtų tokias pačias išraiškos galimybes kaip reliacinė algebra, reikia jog ji palaikytų šias operacijas:

1. Operacijas su aibėmis: sąjunga (*union*), sankirta (*intersection*), skirtumas (*difference*). *OCL* kalboje yra visos šios operacijos.

2. Išrinkimo (*selection*) ir projekcijos (*projection*) operacijos. Išrinkimas *OCL* kalboje ir reliacinėje algebroje yra analogiški pagal savo prasmę. *OCL* kalboje nėra numatyta projekcijos operacijos, tačiau ją galima aprašyti kitų operacijų seka.

3. Dekarto sandauga ir *join* operacija. Šios operacijos aprašytos 4.2.4 skyrelyje.

4. Pervardinimo operacija. Šios operacijos *OCL* kalboje nėra, tačiau ją galima aprašyti netiesiogiai.

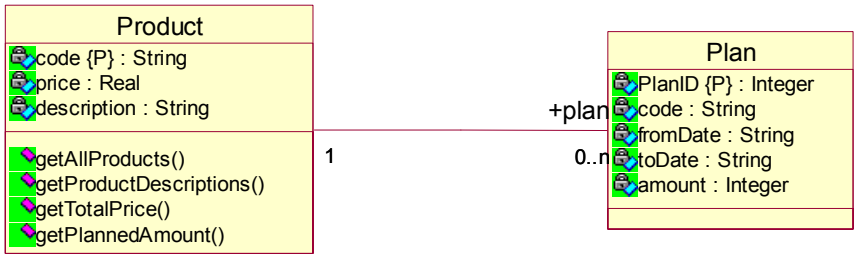
Galima teigti, jog *OCL* kalba užrašytas išraiškas galima pilnai transformuoti į reliacinės algebros išraiškas. Taigi, galima generuoti ir pilnavertes *SQL* užklausas.

3.2.5. Transformacijų medžių panaudojimas SQL kodo generavimui iš OCL išraiškų

Konceptualiųjų apribojimų išraiškų transformavimo į *SQL* kodą scenarijai aprašyti ankstesniuose skyreliuose, tačiau nebuvo aprašytas metodas pačiam kodo generavimui atlikti. Šiam tikslui

pasiūlysime naudoti transformacijų medžius. T. y. pirmiausiai konceptualieji apribojimai bus užrašomi transformacijų medžių pavidalu ir tada pagal sudarytus šablonus medžio atskiroms dalims sugeneruojamas *SQL* kodas.

3.26 pav. pavaizduota gamybinės sistemos dalykinės srities priklausomo nuo platformos lygmens klasių diagramos dalis. Šio lygmens klasių diagrama gaunama iš nepriklausomų nuo platformos lygmens modelių, paliekant tame lygmenyje užrašytas konceptualiųjų apribojimų išraiškas.



3.26 pav. Gamybinės sistemos dalykinės srities priklausomo nuo platformos lygmens klasių diagramos dalis

Klasė *„Product“* turi 4 metodus:

1. *getAllProducts()* – metodas, gražinantis visus gaminamus produktus einamuoju laiko momentu.
2. *getProductDescriptions()* – metodas, gražinantis visų gaminamų produktų aprašymus.
3. *getTotalPrice()* – metodas, gražinantis visų produktų kainų sumą.
4. *getPlannedAmount()* – metodas, gražinantis, kiek planuojama gaminti konkretaus produkto vienetų.

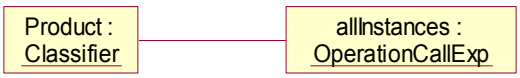
Šių metodų turinys apibrėžiamas naudojant *OCL* išraiškas:

```
context Product::getAllProducts() : Set(Product)
post:
result = Product.allInstances()

context Product::getProductDescriptions() : Bag(String)
post:
result = (Product.allInstances())->collect(description)

context Product::getTotalPrice() : Integer
post:
result = (Product.allInstances())->collect(price)->sum()

context Product::getPlannedAmount() : Integer
post:
result = self.plan->collect(amount)->sum()
```

Tam, kad būtų galima sugeneruoti kiekvienam iš šių metodų *SQL* išraiškas, reikia visų pirma šiuos metodus užrašyti transformacijų medžio pavidalu. Šio medžio viršūnės – *OCL* metamodelio klasių egzemplioriai, atitinkantys *OCL* išskaidytas išraiškas. Lankai, jungiantys medžio viršūnes

atitinka ryšius tarp atskirų išraiškos dalių. Transformacijų medis – tai objektų diagrama, sudaryta *OCL* metamodelio pagrindu ir atvaizduojanti *OCL* išraiškas hierarchiškai.

3.27 pav. pavaizduotas paprasčiausio metodo *getAllProducts()* transformacijos medis.



3.27 pav. Metodo getAllProducts() transformacijos medis

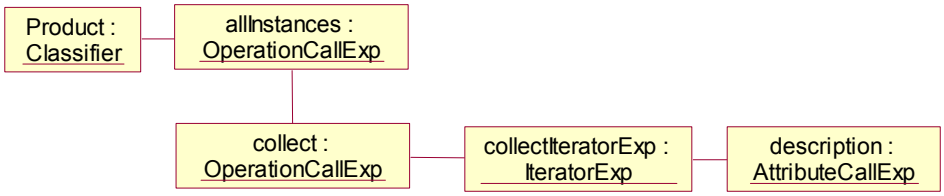
3.27 pav. matyti, jog iš išraiškos *Products.allInstances()* buvo sukurtas transformacijos medis su dviem objektais ir ryšiu tarp jų: *UML* metamodelio *Classifier* klasės objektas *Product* per ryšį sujungtas su *OCL* metamodelio *OperationCallExp* klasės objektu *allInstances*. Šį transformacijos medį galima susieti su tokiu *SQL* sakinio šablonu:

```
select * from $(Source)
```

Demonstruojamam pavyzdžiui šio šablono elementas *Source* bus pakeistas į *„Product“*. Reikia atkreipti dėmesį, jog objektas *Product* atitinka klasę, apibrėžtą projektavimo metu ir naudojamą dalykinei sričiai aprašyti, tuo tarpu *allInstances()* yra *OCL* kalbos išraiška, kuri susiejama su *SQL* kalbos išraiška *„ *“*. Pagal minėtą šabloną bus sugeneruotas toks SQL sakinys:

```
select * from Product
```

Metodas *getProductDescriptions()* ir jau pademonstruotas *getAllProducts()* turi panašumų: abu jie naudoja *allInstances()* operaciją. Tačiau pirmasis dar naudoja ir *collect()* operaciją, kuri reliacinėje algebroje apibrėžiama kaip projekcija. Kadangi abu minėti metodai naudoja *allInstances()* operaciją, su kuria jau susietas *SQL* sakinio šablonas, galima šį šabloną panaudoti ir generuojant kodą *getProductDescriptions()* metodui.



3.28 pav. Metodo getProductDescriptions() transformacijos medis

3.28 pav. pavaizduotas transformacijos medis *getProductDescriptions()* metodui. Objektai *Product* ir *allInstances* atitinka 3.27 pav. objektus. Tuo tarpu objektus *collect*, *collectExp*, *collectIteratorExp* ir *description* galima susieti su tokiu pačiu *SQL* šablonu kaip ir praeitame pavyzdyje:

```
select $(ColExp) from $(Source)
```

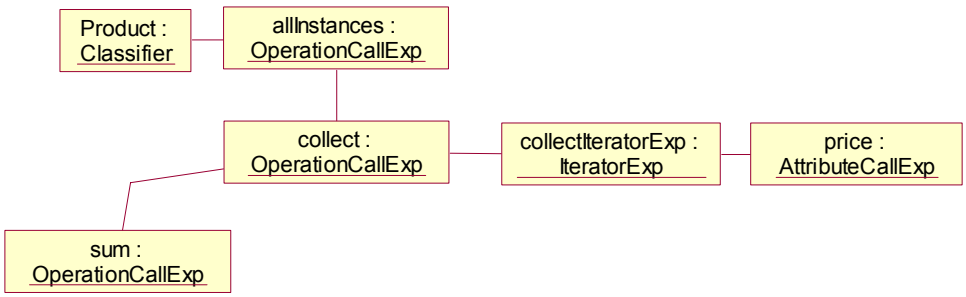
Reikia atkreipti dėmesį, jog šablono *Source* elementas gaunamas transformuojant *Product* ir *allInstances* objektus į *SQL* kodą.

Demonstruojamo pavyzdžio *getProductDescriptions()* metodo *OCL* kodas bus transformuotas į tokį *SQL* kodą:

```
select description from
(select * from Product)
```

Šiame darbe labiau akcentuojamos generavimo idėjos ir galimybės, todėl šios transformacijos rezultatas nėra optimalus (galima gauti ekvivalentų rezultatą, kai sugeneruojama viena užklausa).

Panagrinėsime pavyzdį, kuomet panaudojama *OCL* sumos operacija *sum()*. Ji panaudota metode *getTotalPrice()*. Reikia pastebėti, jog šio metodo *OCL* išraiška turi tuos pačius elementus kaip ir praeitų pavyzdžių išraiškos, tačiau čia dar iškviečiama *OCL* operacija *sum()*. Transformacijos medis šiai išraiškai užrašytas 3.29 pav.



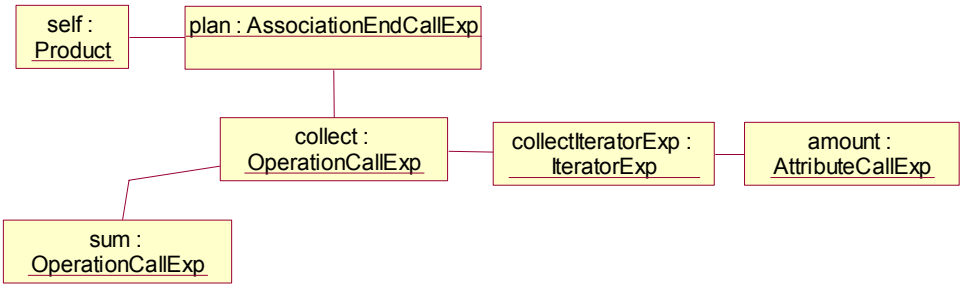
3.29 pav. Metodo getTotalPrice() transformacijos medis

Transformacijos rezultatas 3.29 pav. užrašytam transformacijos medžiui:

```
select sum(price) from
(select * from Product)
```

Svarbu atkreipti dėmesį, jog pagrindinė generavimo šaka yra *Product*, *allInstances*, *collect* ir *sum* objektai. Tuo tarpu objektai *collectIteratorExp* ir *price* yra įtraukiami į transformacijos procesą tuo pat metu, kaip ir *collect* objektas ir tik atlikus objekto *collect* transformaciją į *SQL* kodą, atliekama *sum* objekto transformacija.

Metodo *getPlannedAmount()* transformacijos į *SQL* kodą metu bus panaudota navigacija: šiuo atveju turi būti išrenkami tie klasės *Plan* egzemplioriai, kurie susiję su tam tikru *Product* klasės egzemplioriumi. Transformacijos medis pavaizduotas 3.30 pav.



3.30 pav. Metodo getPlannedAmount() transformacijos medis

3.30 pav. transformacijos medžio objektus *self* ir *plan* galime susieti su tokiu *SQL* šablonu:

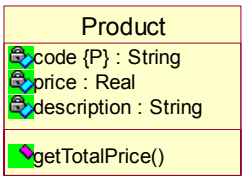
```
select * from $(Left)
inner join $(Right) on $(Left).$(LPrimary) = $(Right).$(LPrimary)
where $(Left).$(LPrimary) = $(LPValue)
```

Kaip transformuoti likusią transformacijos medžio dalį (*collect*, *collectExp*, *collectIteratorExp*, *amount* ir *sum* objektus) pademonstruota anksčiau. Transformavę visą medį į SQL kodą, gauname:

```
select sum(amount) from
(
select * from Product
inner join Plan on Product.code = Plan.code
where Product.code = $(LPValue)
)
```

3.2.6. Serverio procedūrų ir trigerių generavimo principai

Serverio procedūros bei trigeriai generuojami tais pačiais principais, kurie išdėstyti 3.2.5 skyrelyje: *OCL* išraiškos užrašomos transformacijų medžiais, kurie, naudojant šablonus, transformuojami į *SQL* kodą. Šiame skyrelyje bus pademonstruota *OCL* operacijos *iterate* transformacija į serverio procedūrą.



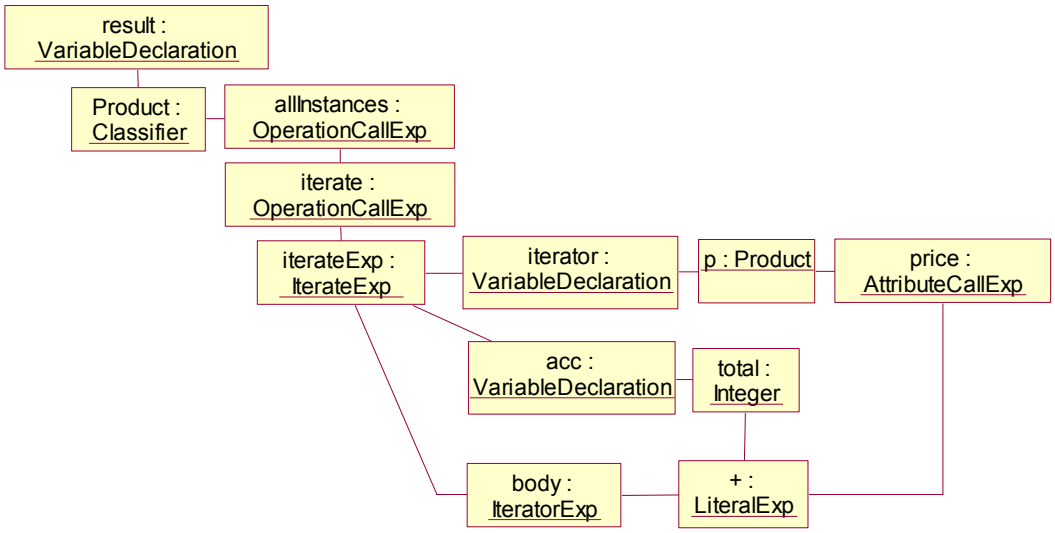
3.31 pav. Klasė Product ir jos metodas produktų kainų sumos skaičiuoti

3.31 pav. pavaizduotos klasės metodas skaičiuoja visų gaminamų produktų kainų sumą, t.y. jo paskirtis tokia pati, kaip ir 3.2.5 skyrelio *Product* klasės atitinkamo metodo. Tačiau jo OCL apribojimas užrašytas panaudojant nebe *collect* operaciją, o *iterate* operaciją:

```
context Product::getTotalPrice() : Integer
post:
result = (Product.allInstances())->iterate( p: Product;
total: Integer = 0 |
total + p.price )
```

collect operaciją buvo galima transformuoti į *SQL select* sakinį, tačiau *iterate* operacijos transformacijai reikalingas ciklo sakinys, taigi reikalinga pilnavertė programavimo kalba, tokia kaip *PL/SQL* (*Oracle*) arba *T-SQL* (*Microsoft SQL Server*).

3.32 pav. pavaizduotas transformacijos medis *getTotalPrice()* operacijai.



3.32 pav. Metodo getTotalPrice(), panaudojant iterate operaciją, transformacijos medis

Objektai *Product*, *allInstances* ir *iterate* atitinka 3.2.5 skyrelyje nagrinėtus objektus bei sugeneruotus šablonus. Objektas *result* skirtas saugoti *iterate* operacijos rezultatą. Medžio šaka, prasidedanti *iterateExp* objektu, siejasi su objektais *iterator*, *acc*, ir *body*, kurie atitinka *OCL iterate* išraiškos operandus. Šią šaką galime sieti su *SQL* šablonu:

```
DECLARE
   $(Variables)
   CURSOR C IS
      $(Source);
BEGIN

   FOR rec IN C LOOP
      $(Actions)
   END LOOP;

END;
```

Objektas *acc* transformuojamas į *$(Variables)* dalį, objektas *iterator* – į *$(Source)* dalį, o objektas *body* – į *$(Actions)* dalį. Demonstruojamu atveju transformacijos rezultatas (serverio procedūra), naudojant šį šabloną, gaunamas toks:

```
DECLARE
   Total NUMBER;
   CURSOR C IS
      SELECT price FROM PRODUCTS;
BEGIN

   FOR rec IN C LOOP
      Total := Total + rec.price;
   END LOOP;

END;
```

Svarbu atkreipti dėmesį, jog transformacijos medžio objekto *iterator* šakos sugeneruotas *SQL* kodas *$(Source)* priklauso nuo *body* objekto šakos sudėtingumo. Pvz. jei *body* šakoje būtų naudojama *OCL* navigacija, tai *$(Source)* dalyje jau reikėtų naudoti Dekarto sandaugos operacijas, kad vėliau būtų galima sugeneruoti ekvivalentų kodą užrašytoms išraiškoms.

4. APIBENDRINTAS ALGORITMAS UML MODELIŲ TRANSFORMACIJAI Į PILNAVERČIUS RELIACINIUS MODELIUS

Vienas iš *MDA* tikslų – kuo labiau pakartotinai panaudoti sudarytus sistemų modelius, keičiantis technologijoms. T. y. modeliai sudaromi tokiu principu, kad tam tikra jų dalis nepasikeistų arba pasikeistų nedaug net radikaliai pasikeitus technologijoms.

Šiame darbe pademonstruota, kaip iš klasių diagramų (naudojant *OCL* apribojimus) generuoti reliacinius modelius su reliaciniais apribojimais, vaizdais, serverio procedūromis. Darbe apeinamas reliaciniam modeliui generuoti sukurtas „*UML Profile for Databases*“ modelis. Remiamasi tik *MOF* pagrindu veikiančiais metamodeliais (*UML, OCL, CWM*), todėl vadovaujantis darbe aprašytu modeliavimo metodu, sukurti modeliai tinka ne tik reliacinei paradigmai.

Galime užrašyti apibendrintą reliacinių modelių bei *SQL* kodo generavimo iš *UML*/*OCL* modelių algoritmą:

1. *UML* modelių transformacijos į *CWM* reliacinius modelius metamodelių pagrindu:

a) Klasių (taip pat ir asociacijų klasių) transformavimas į lenteles
b) Asociacijų (taip pat ir priklausančių asociacijų klasėms) transformacijos į išorinius raktus
c) *UML* atributų transformacijos į *RDBVS* stulpelius
d) Išorinių raktų stulpelių generavimas
e) *UML/OCL* paprastųjų ir sudėtinių tipų transformacijos į *SQL* tipus

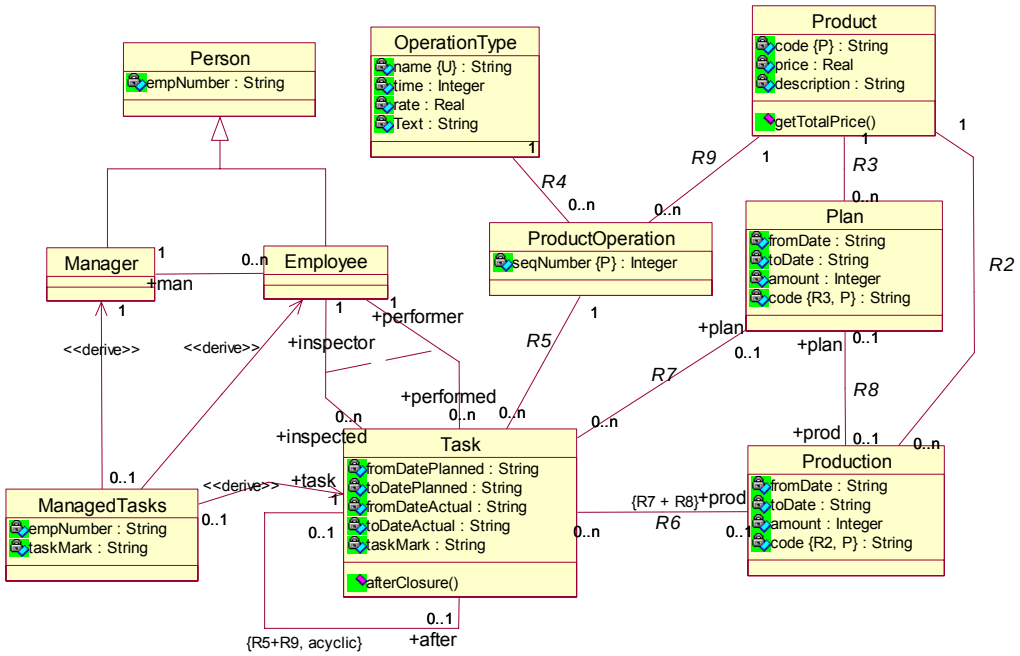
2. Apribojimų *RDBVS* lentelėms generavimas iš *OCL* invariantų
3. *RDBVS* vaizdų generavimas iš *UML*/*OCL* modelių
4. Serverio procedūrų bei trigerių generavimas iš *OCL* išraiškų

Šiame darbe aprašytas nepilnas generavimo iš *UML/OCL* į reliacinius modelius procesas. Todėl jis galėtų būti papildytas šiomis dalimis:

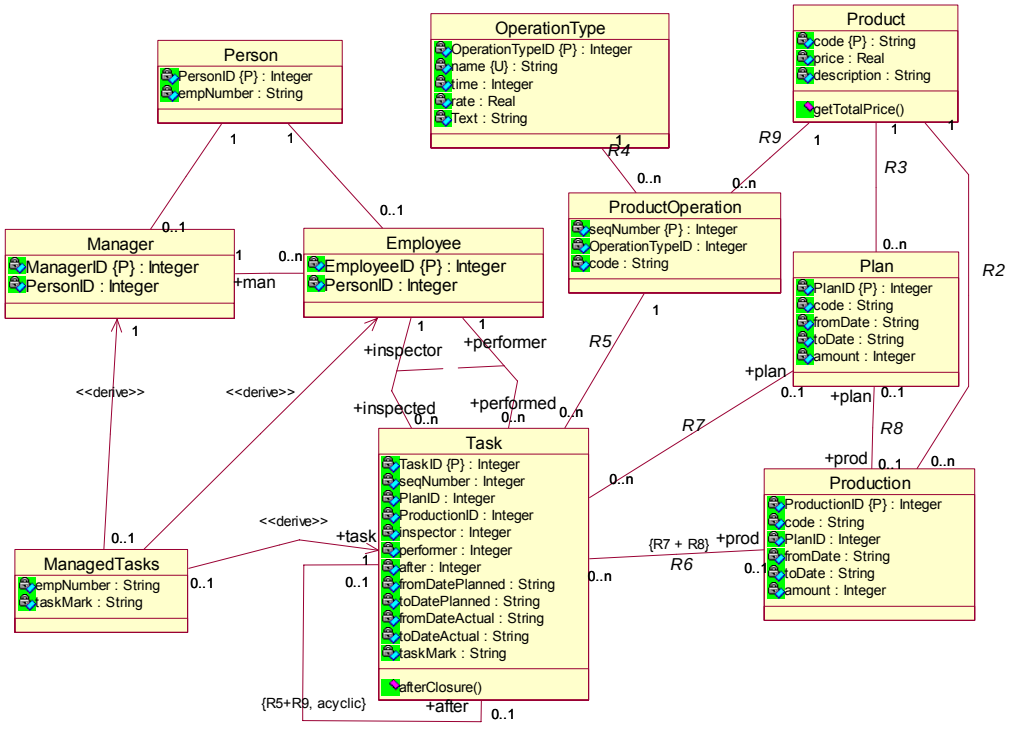
1. Sudėtinių tipų transformacijomis iš *UML/OCL* modelių į reliacinius modelius metamodelių lygmenyje.
2. Darbo su agregatais, grupavimo konstrukcijomis, subužklausomis, rezultatų sujungimo („*union*“) tipiniais atvejais analize.
3. Pirminio rakto, unikalumo ir kitų stereotipų (*„{P}“, „{U}“, „{N}“*) panaudojimo *UML/OCL* modeliuose analize.
4. Transformacijų taisyklių grafinėmis interpretacijomis.

5. METODO TAIKYMO PAVYZDYS

Transformacijai iš *UML/OCL* į *SQL* kodą pademonstruoti naudosime gamybos informacijos sistemos modelį, pritaikytą iš [4] bei dalį [4] aprašytų *OCL* apribojimų. Šios sistemos *PIM* ir *PSM* lygmenų modeliai pateikti 5.1 pav. ir 5.2 pav.



5.1 pav. Gamybos informacijos sistemos PIM lygmens dalykinės srities klasių diagrama.



5.2 pav. Gamybos informacijos sistemos PSM lygmens dalykinės srities klasių diagrama.

5.1 pav. pavaizduotas dalykinės srities *PIM* modelis. Šis modelis nepriklausomas nuo realizacijos, todėl gali būti naudojamas ne vien reliacinėms schemos generuoti. Tuo tarpu 5.2 pav. pavaizduotas *PSM* lygmens modelis skirtas reliacinių duomenų bazių schemos generavimui. Jis gaunamas transformuojant *PIM* lygmens *UML* modelius į *CWM* reliacinius modelius metamodelių lygmenyje. Tokios transformacijos metu atliekamas: klasių (taip pat ir asociacijų klasių) transformavimas į lenteles; asociacijų (taip pat ir priklausančių asociacijų klasėms) transformacijos į išorinius raktus; *UML* atributų transformacijos į *RDBVS* stulpelius; išorinių raktų stulpelių generavimas; *UML/OCL* paprastųjų ir sudėtinių tipų transformacijos į *SQL* tipus. Po transformacijos metamodelių lygmenyje, atliekamos šabloninės transformacijos: apribojimų *RDBVS* lentelėms generavimas iš *OCL* invariantų; *RDBVS* vaizdų generavimas iš *UML*/*OCL* modelių; serverio procedūrų bei trigerių generavimas iš *OCL* išraiškų.

Šiame pavyzdyje pademonstruosime transformacijas metamodelių lygmenyje bei keletą šabloninės transformacijos taikymo pavyzdžių: apribojimų *RDBVS* lentelėms generavimą iš *OCL* invariantų, vaizdų bei serverio procedūros generavimą iš *UML/OCL*. Generuojant vaizdą, bus atliktas atributų pervardijimas, pademonstruotas *JOIN* sakinio, rezultatų apjungimo konstrukcijos *UNION* bei sąlygos sakinio *WHERE* generavimas.

Aprašysime keletą *OCL* apribojimų šioms klasių diagramoms (5.1 lentelė).

5.1 lentelė

Gamybos informacijos sistemos dalykinės srities klasių diagramos OCL apribojimai

Aprašymas	Apribojimo tekstas OCL kalboje

Klasėje *Task* bent vienas iš atributų *fromDatePlanned, toDatePlanned, fromDateActual, toDateActual* turi būti apibrėžtas	`context Task inv datesDefined:`
	`self.fromDatePlanned.oclIsKindOf(OclVoid) = false or`
	`self.toDatePlanned.oclIsKindOf(OclVoid) = false or`
	`self.fromDateActual.oclIsKindOf(OclVoid) = false or`
	`self.toDateActual.oclIsKindOf(OclVoid) = false`

Jei produkcijai pagaminti reikalingos užduotys suplanuotos, tai klasės *Task* egzemplioriai, turintys asociacijas R7 ir R8, turi ir asociaciją R6	`context Task`
	`inv certainlyPlanned:`
	`if self.prod->notEmpty() and self.prod.plan -> notEmpty()`
	`then self.plan = self.prod.plan endif`

Produkto gamybos užduočių sekoje užduotis atliekama tik vieną kartą ir negali kartotis	`context Task::afterClosure( t: Task ) : Set(Task)`
	`post: result = t.after->iterate (`
	`p : Task;`
	`acc : Set(Task) = t.after |`
	`acc ->`
	`if t.after->notEmpty() then`
	`if acc.includes(t) then acc`

	else acc.union(t.afterClosure()) endif endif) context Task inv: not self.afterClosure(self)->includes(self)
Užduotį vykdo atlikėjas, o tikrina inspektorius. Jie negali būti vienas ir tas pats asmuo	context Task inv diffPersons: not(self.performer = self.inspector)
ManagedTasks klasė yra išvestinė. Šios klasės egzemplioriai nusako visus gamybos vadovus, atsakingus už užduočių atlikimą 2005.01.01.	context ManagedTasks::taskMark : String derive: self.task.taskMark context ManagedTasks::empNumber : String derive:self.task.inspector.man.empNumber context ManagedTasks::empNumber : String derive: self.task.performer.man.empNumber context ManagedTasks inv Salyga: self.task.fromDatePlanned = '2005-01-01'
Metodas *getTotalPrice()* gražina visų gaminamų produktų kainų sumą	context Product::getTotalPrice() : Integer post: result = (Product.allInstances())->iterate(p: Product; total: Integer = 0 | total + p.price)

Šiems apribojimams pritaikysime transformacijas, aprašytas darbe ir sugeneruosime atitinkamą *SQL* kodą. Jis pateiktas 5.2 lentelėje.

5.2 lentelė

Gamybos informacijos sistemos SQL kodo fragmentai, gauti transformacijų metu

OCL apribojimo pavadinimas	SQL kodas
datesDefined	CHECK fromDatePlanned IS NOT NULL OR toDatePlanned IS NOT NULL OR fromDateActual IS NOT NULL OR toDateActual IS NOT NULL
diffPersons	CHECK NOT performer = inspector
Salyga, „derived“ iraiškos	CREATE VIEW ManagedTasks AS SELECT task.taskMark, person.empNumber FROM Tasks task INNER JOIN Employee on task.inspector = Employee.EmployeeID INNER JOIN Manager man on Employee.ManagerID = Manager.ManagerID INNER JOIN Person person on Manager.PersonID = Person.PersonID

	WHERE Task.fromDatePlanned = ‘2005-01-01‘ UNION ALL SELECT task.taskMark, person.empNumber FROM Tasks task INNER JOIN Employee on task.performer = Employee.EmployeeID INNER JOIN Manager man on Employee.ManagerID = Manager.ManagerID INNER JOIN Person person on Manager.PersonID = Person.PersonID WHERE Task.fromDatePlanned = ‘2005-01-01‘

getTotalPrice() apribojimas	DECLARE
	Total NUMBER;
	CURSOR C IS
	SELECT price FROM PRODUCTS;
	BEGIN
	
	FOR rec IN C LOOP
	Total := Total + rec.price;
	END LOOP;
	END;

IŠVADOS

- Atlikta metodų, skirtų objektiniams modeliams transformuoti į reliacinius, analizė
- Išbandytos esamų *CASE* įrankių galimybės analizuotoms transformacijoms atlikti
- Pastebėta, jog šiai praktiškai labai svarbiai problemai spręsti trūksta sistemiško požiūrio: modeliai, pagal kuriuos veikia dauguma šių dienų reliacinių duomenų bazių generavimo įrankių, nėra sudaryti pagal standartus (tai dažniausiai konkrečių bendrovių sukurti modeliai, naudojami jų produktuose)
- Minėtieji modeliai neturi galimybių automatiškai generuoti pilnavertes reliacinių duomenų bazių schemas (automatiškai negeneruojamos nei integralumo apribojimus palaikančios *DBVS* funkcijos, nei vaizdai, nei trigeriai, nei serverio procedūros)
- Generavimas atliekamas iš tarpinių modelių, tinkamų vaizduoti tik reliacinėms schemoms
- Pasiūlytas metodas, kuris pagal *MDA* principus leidžia generuoti reliacines schemas iš konceptualiojo modelio ir užtikrina aukštesnį informacijos sistemų kūrimo abstrakcijos lygmenį
- Siūlomas metodas leidžia pasiekti didesnį projekto dalių vientisumą ir gyvavimo laiką, kadangi konceptualusis modelis tiksliai aprašomas *UML* ir *OCL* išraiškomis; iš jo galima generuoti ir kitus projekto rezultatus
- Metodas remiasi tik *OMG* standartais arba *RFP* dokumentais ir leidžia automatiškai paversti konceptualųjį modelį su apribojimais ir išvedimo taisyklėmis į *DBVS* palaikomas funkcijas, vaizdus, trigerius ir serverio procedūras
- Pilnavertės reliacinės duomenų bazių schemos generuojamos dviem etapais:
  - pirmame etape remiantis metamodelių transformacijomis *UML* modeliai verčiami į rekliacinius;
  - antrame etape remiantis tipinėmis transformacijomis atliekamas *DBVS* palaikomų funkcijų, vaizdų, trigerių ir serverio procedūrų generavimas
- Darbe pateikiamas eksperimentinio pavyzdžio aprašymas, pademonstruotos pilnaverčių reliacinių schemų generavimo galimybės, kurios galėtų papildyti esamų *CASE* įrankių funkcionalumą, tuo palengvindamos projektuotojų darbą, pagreitindamos kūrimą ir užtikrindamos geresnę informacinių sistemų projektų kokybę

LITERATŪRA

1. Gogolla M., Richters M. Expressing UML class diagrams properties with OCL. Lecture Notes in Computer Science, Vol. 2263, 2002, p.85-114.
2. Smith J., Kokar M., Baclawski K. Formal Verification of UML Diagrams: A First Step Towards Code Generation [žiūrėta – 2004-01-21]. Prieiga per internetą: <http://www1.coe.neu.edu/~kokar/>
3. Database design models – the UML profile for database design [žiūrėta 2004-03-06]. Prieiga per internetą: <http://www.agiledata.org/essays/umlDataModelingProfile.html>
4. Miliauskaitė, E., Nemuraitė L. Taxonomy Of Integrity Constraints In Conceptual Models. In: Proceedings of IADIS Database Systems 2005 (to be published).
5. Demuth B. and Hussmann H. The Unified Modeling Language, Proc. 2nd International Conference, Springer LNCS 1723, 1999, p. 598-613.
6. Sosunovas S., Vasilecas O. Verslo taisyklių modelių transformacija meta-modelių pagrindu. Informacinės Technologijos'2004, Kainas, Technologija, p.585-590.
7. Jacobson I., Booch G., Rumbaugh J. The Unified Modeling Language User Guide. Boston: Addison Wesley, 2000.
8. Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process. Boston: Addison Wesley, 1999.
9. Akehurst D.H., Bordbar B. On Querying UML data models with OCL, in: UML 2001: The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001 (LNSC 2185).
10. Balsters H. Derived Classes as a Basis for Views in UML/OCL Data Models, University of Groningen, Report N 02A47, 2003.
11. Casanave C. OMG Enterprise Collaboration Architecture (ECA) [žiūrėta – 2004-02-03]. Prieiga per internetą: <http://www.omg.org>
12. Miller J., Mukerji J. MDA Guide Version 1.0.1 [žiūrėta – 2004-04-27]. Prieiga per internetą: <http://www.omg.org>
13. Response to the MOF 2.0 Query/views/transformations RFP (ad/2002-04-10). OMG document ad/2003-08-05 [žiūrėta – 2004-02-12]. Prieiga per internetą: <http://www.omg.org>

14. Frankel D. Model Driven Architecture: reality and implementation [žiūrėta – 2004-01-25]. Prieiga per internetą: <http://www.omg.org>

15. Malia T. OMG Model Driven Architecture in the Application Life Cycle [žiūrėta – 2004-06-02]. Prieiga per internetą: <http://www.omg.org>

16. Alagic S., Bernstein P.A. A Model Theory for Generic Schema Management. In: Lecture Notes in Computer Science, Vol. 2397, Springer Verlag, 2002, p. 228-246

17. UML 2.0 Superstructure Specification (ptc/03-08-02). OMG document ptc/03-08-02 [žiūrėta – 2005-02-21]. Prieiga per internetą: <http://www.omg.org>

18. UML 2.0 Infrastructure Specification (ptc/03-09-15). OMG document ptc/03-09-15 [žiūrėta – 2005-02-24]. Prieiga per internetą: <http://www.omg.org>

19. UML 2.0 OCL Specification (ptc/03-10-14). OMG document ptc/03-10-14 [žiūrėta – 2005-03-10]. Prieiga per internetą: <http://www.omg.org>

20. Common Warehouse Metamodel (CWM) specification (formal/03-03-02). OMG document formal/03-03-02 [žiūrėta – 2005-02-17]. Prieiga per internetą: <http://www.omg.org>

21. (ISO-ANSI Working Draft) SQL/Foundation (ANSI TC NCITS H2, ISO/IEC JTC 1/SC 32/WG 3), 2003.

TERMINŲ IR SANTRUMPŲ ŽODYNAS

CASE (angl. *Computer Aided Software Engineering*)	Kompiuterizuotas programinės įrangos projektavimas
OMG (angl. *Object Management Group*)	Objektų valdymo grupė
MDA (angl. *Model Driven Architecture*)	Modeliais grįsta architektūra
UML (angl. *Unified Modeling language*)	Universali modeliavimo kalba
OCL (angl. *Object Constraint Language*)	Objektų apribojimo kalba
TRL (angl. *Transformation Rule Language*)	Metamodelių transformacijos kalba
CWM (angl. *Common Warehouse Metaodel*)	Vieningos saugyklos metamodelis
PIM (angl. *Platform Independent Model*)	Nuo platformos nepriklausomas modelis
PSM (angl. *Platform Specific Model*)	Nuo platformos priklausomas modelis
RFP (angl. *Request for Proposals*)	Užklausa pateikti pasiūlymus
SQL (angl. *Structured Query Language*)	Struktūrinių užklausų kalba
DBVS	Duomenų bazių valdymo sistema
RDBVS	Reliacinių duomenų bazių valdymo sistema
RDB	Reliacinė duomenų bazė

PRIEDAS. PUBLIKUOTI STRAIPSNIAI

Pilnaverčių reliacinių duomenų bazių schemų generavimas naudojant uml ir ocl[1]

Andrius Armonas, Lina Nemuraitė
Kauno technologijos universitetas, Informacijos sistemų katedra

Straipsnyje apžvelgiamos šių dienų reliacinių duomenų bazių schemų generavimo technologijos, atskleistos jų problemos, kurioms spęsti pateikiami susisteminti pilnaverčių reliacinių schemų generavimo principai. Šias schemas siūloma generuoti iš konceptualiojo, nuo platformos nepriklausomo modelio, laikantis objektinių standartų. Toks požiūris leistų greičiau reaguoti į verslo pokyčius, pagerinti pačių duomenų, bazių projektavimo ir priežiūros proceso kokybę, palengvinti specialistų veiklą.

1. ĮVADAS

Duomenys yra organizacijos turtas, o reliacinės duomenų bazių valdymo sistemos – vis dar dažniausiai naudojamas programinės įrangos tipas duomenims valdyti. Nors pačios duomenų bazių valdymo technologijos palyginti gerai išvystytos, tačiau jų projektavimo metodai dažnai nederinami su bendru programinės įrangos projektavimo procesu. Projektuoti ar generuoti pilnavertes reliacinių duomenų bazių schemas paprastai galima tik specializuotais konkrečių duomenų bazių valdymo sistemų įrankiais ir *SQL* dialektais. Todėl migravimas tarp skirtingų reliacinių platformų, jau nekalbant apie perėjimą nuo vienos technologijos prie kitos, tampa problematiškas arba iš vis neįmanomas, nekeičiant visos sistemos ir projektavimo įrankių.

Situaciją galima pagerinti naudojant *OMG* modeliais grindžiamą architektūrą *MDA* (*Model Driven Architecture*). Ji skirta greitai reaguoti į besikeičiančias informacinių sistemų kūrimo technologijas bei išvengti su tokiais pokyčiais susijusio sistemos perprojektavimo. Straipsnyje aprašoma, kaip *MDA* nuo platformos nepriklausomus modelius *PIM (Platform Independent Model)* su apribojimais, aprašytais *OCL* išraiškomis, galima transformuoti į platformai būdingus modelius *PSM (Platform Specific Model),* o po to į kodą, tam naudojant metamodelių transformacijas ir transformacijų šablonus. Iš vieno nuo platformos nepriklausomo modelio galima generuoti įvairių platformų programines realizacijas, tame tarpe ir *SQL* išraiškas.

Siūlomo metodo taikymas duoda naudą verslui daugeliu aspektų: leidžia pagerinti duomenų kokybę, palengvinti jų projektavimą ir priežiūrą, greitai reaguoti į verslo pokyčius; suderinti duomenų bazių projektavimą su bendriniu projektavimo procesu, tai reiškia – pagerinti informacinių sistemų projektavimu užsiimančių organizacijų verslo procesus, analitikų ir projektuotojų veiklą, sumažinti rutininių programavimo darbų kiekį.

2. RELIACINŲ DUOMENŲ BAZIŲ SCHEMŲ GENERAVIMO IŠ UML/OCL MODELIŲ PRIVALUMAI

Praktikoje vis labiau įsigali objektinės orientacijos projektavimo procesai (Jacobson, 2000; MDA, 2003), tačiau didžioji dalis naudojamų duomenų bazių yra reliacinės, taigi tokie procesai turėtų leisti kurti bei palaikyti programų sistemas, veikiančias kartu su reliacinėmis duomenų bazėmis.

Nemažai autorių (Demuth, 1999; Akehurst, 2001; Gogolla, 2002; Alagic, 2002; Balsters, 2003) aprašo objektinių modelių klasių diagramų transformavimą į reliacinių duomenų bazių schemas. Dažnai šios transformacijos apsiriboja paprastu atveju *klasė* → *lentelė*, į transformavimo procesą įtraukiant tik atributus ir ryšius. Tokios transformacijos rezultatas labai ribotai naudoja reliacinių duomenų bazių technologijos teikiamas galimybes. Domenų bazės integralumo užtikrinimo priemonės – trigeriai, ryšių vientisumo priemonės (*angl. referential integrity*), automatinis įvairių apribojimų tikrinimas paliekami nuošalyje.

Vienas iš praktikoje naudojamų metodų generuoti reliacines schemas iš *UML* (UML 2.0 Superstructure, 2003; UML 2.0 Infrastructure, 2003) diagramų – *Rational* bendrovės sukurtas „*UML profile for databases*“ modelis. Naudojant „*UML profile for databases*“ metodiką, *UML* modeliai transformuojami į tarpinius modelius, pataisomi projektuotojo ir tada generuojamas *SQL* kodas. Tačiau pagal šį modelį (jis nėra standartas) sugeneruojami tik pagrindiniai schemos elementai (lentelės, atributai, pirminiai ir išoriniai raktai), į kurių tarpą nepatenka nei trigeriai, nei serverio procedūros, nei sąlygų tikrinimo funkcijos.

[1] Šį darbą remia Lietuvos valstybinis mokslo ir studijų fondas pagal Eureka programos projektą „IT-Europe” (Reg. Nr. 3473)

Veiklos taisyklių aprašymas duomenų bazių schemoje (naudojant trigerius ir sąlygų tikrinimą) sumažina programinio kodo kiekį bei garantuoja, jog visa programinė įranga, dirbanti su konkrečia duomenų baze, veiks pagal tas pačias veiklos taisykles (Sosunovas, 2004), t.y. integralumą išlaikančius apribojimus. Vienas iš metodų generuoti tokius apribojimus – *UML* naudojimas kartu su *OCL* (UML 2.0 OCL, 2003) kalba. *UML* – plačiai paplitęs standartas, naudojamas objektinės orientacijos projektavimo procesuose. Todėl *UML* palaiko didelė dalis *CASE* įrankių. *OCL* (*Object Constrasint Language*), kuri yra *UML* dalis, leidžia tiksliai apibrėžti integralumo taisykles. Tai atliekama aprašant modelių objektų invariantus. *OCL* navigavimo (*angl. navigation – oriented*) metodologija iš dalies sutampa su reliacinių užklausų koncepcija. Dauguma šiandieninių reliacinių duomenų bazių turi nuosavus dialektus trigeriams ir sąlygoms (nors *SQL:2003* (SQL, 2003) standartas tam apibrėžia standartinę kalbą). Naudojant *OCL*, galima specifikuoti apribojimus duomenų bazių schemoms nepriklausomai nuo konkrečios reliacinės duomenų bazių valdymo sistemos.

Apribojimų specifikavimas *OCL* kalba ir automatinis generavimas iš *OCL* į *SQL* žymiai sumažintų kūrimo laiko sąnaudas, o *post-* ir *pre-* sąlygų specifikavimas *OCL* kalba gali būti geras atspirties taškas *SQL* užklausų generavimui (update, select sakiniai, trigeriai).

3. ESMINĖS TRANSFORMAVIMO IŠ UML/OCL Į SQL PROBLEMOS

Atlikus literatūros analizę, galima išskirti esmines problemas, generuojant OCL apribojimus bei išraiškas į SQL apribojimus bei sakinius. Tai – apribojimų generavimo, transformavimo į vaizdus bei serverio procedūrų generavimo problemos.

Transformuojant *OCL* apribojimus į *SQL* apribojimus išskirtinos šios problemos:

● Paprastųjų *OCL* tipų transformacijos. Paprastieji *OCL* tipai (*real*, *integer*, ir t.t.) tiesiogiai transformuojami į *SQL* tipus. Problemos gali atsirasti tada, kai nėra tiesioginių *SQL* tipų atitikmenų (pvz., *INTERVAL*).

● *OCL* invariantų transformacijos. *OCL* invariantų transformavimas į *SQL* apribojimus gana gerai išnagrinėtas. Jie dažniausiai transformuojami į sąlygų tikrinimą (*angl. Assertions*). Tačiau reikia pastebėti, jog *SQL* išraiškas galima įvertinti net jei gražinama *NULL* reikšmė. Tuo tarpu *OCL* nenusako kaip dirbama su *OCL* atitikmeniu *unknown*.

● Klasių ir atributų transformacijos. Transformacija, kuomet klasė su atributais tiesiogiai transformuojama į lentelę ir jos atributus, yra gana gerai išnagrinėta. Tačiau čia iškyla dvi rimtos problemos, susijusios su objektinio ir reliacinio modelių nesutapimu:

  ○ *Objektų identifikacija*. Objekto identifikatoriaus sąvoka skiriasi nuo dirbtinio lentelės pirminio rakto sąvokos. Objekto identifikatorių gali atitikti aibė lentelės pirminio rakto reikšmių. Negana to, ši aibė gali kisti. Todėl nagrinėjant įvairius transformacijų atvejus, labai svarbu įvertinti šį neatitikimą.

  ○ *Objektų palyginimo problema*. Čia reiktų išskirti objektų reikšmių palyginimą ir pačių objektų palyginimą. Objektai nebūtinai yra lygūs, jei jų reikšmės lygios. Todėl jei turime *OCL* išraišką `object = (object2 : OCLAny): Boolean`, tai reiškia, jog po transformacijos reliacinėje duomenų bazėje turės būti lyginamos objektus atitinkančių pirminių raktų reikšmės.

● *OCL* navigavimo problemos. Dažnai apribojimai nesibaigia vieno objekto ribomis. Per asociacijas tenka kreiptis į kitus objektus, kad pilnai nusakyti norimą apribojimą. Šios transformacijos gali būti realizuojamos subužklausų pagalba. Tačiau kai kurias situacijas galima aprašytii tik labai sudėtingomis užklausomis, kurias palaiko nedidelė dalis šiandieninių reliacinių duomenų bazių valdymo sistemų.

● Kai kurių *OCL* konstrukcijų, pavyzdžiui, *iterate* transformavimas į ekvivalentų *SQL* kodą.

● *OCL* išraiškų užrašymo ir *SQL* sakinių generavimo sudėtingumo problemos.

Esminės *OCL* išraiškų sekų transformavimo į vaizdus (*views*) problemos:

● Vaizdų aprašymo modeliai gana sudėtingi ir labai priklausantys nuo konkrečios situacijos (apibendrinimo trūkumas)

● Neapibrėžtas darbas su agregatais, grupavimo konstrukcijomis, subužklausomis, lentelių rezultatų jungimu (*SQL* sakinys *UNION*)

Bendrai paėmus *OCL* apribojimų transformacijas į *SQL* apribojimus ir užklausas, visiems transformacijų variantams būdingos problemos yra šios:

● *Sugeneruotų SQL sakinių sudėtingumo problema*: norint išvengti serverio procedūrų generavimo, kartais būtina generuoti tokio sudėtingumo *SQL* sakinius, kuriuos palaiko labai nedaug *RDBMS* serverių.

● *Kai kurių OCL išraiškų užrašymo sudėtingumas*. Užrašytos *OCL* išraiškos gali būti labai sudėtingos ir tai labai apsunkina jų tipinių atvejų atpažinimą bei generavimą į *SQL* kodą.

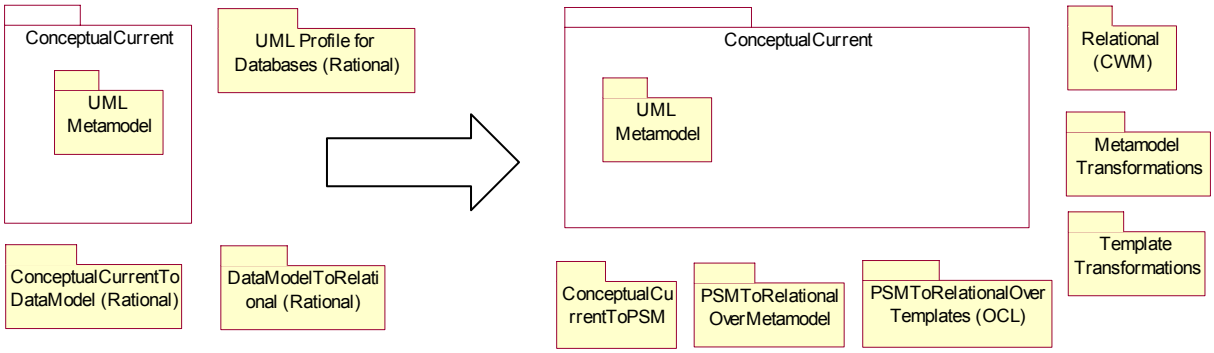
OCL transformavimas į *SQL* apribojimus, vaizdus ir serverio procedūras techniškai vienaip ar kitaip išsprendžiamas. Tačiau esminis dalykas, ko trūksta šioje srityje – bendra metodika, kaip elgtis vienais ar kitais atvejais, kuomet susiduriama su dviprasmybėmis arba *OCL-SQL* nesuderinamumais.

Šiame darbe transformacijos nagrinėjamos kombinuojant *UML/OCL* ir reliacinių metamodelių transformacijas bei tipinių transformacijų šablonus. Svarbus akcentas tenka *MDA* principams – konceptualieji *UML/OCL* modeliai yra nepriklausomi nuo konkrečios platformos ir gali būti pakartotinai panaudojami įvairių technologijų kodui generuoti.

4. RELIACINIŲ SCHEMŲ GENERAVIMAS, NAUDOJANT UML BEI OCL

Šiuo metu praktikoje naudojamas ir šiame darbe siūlomas metodas grafiškai pavaizduoti 1 paveiksle.

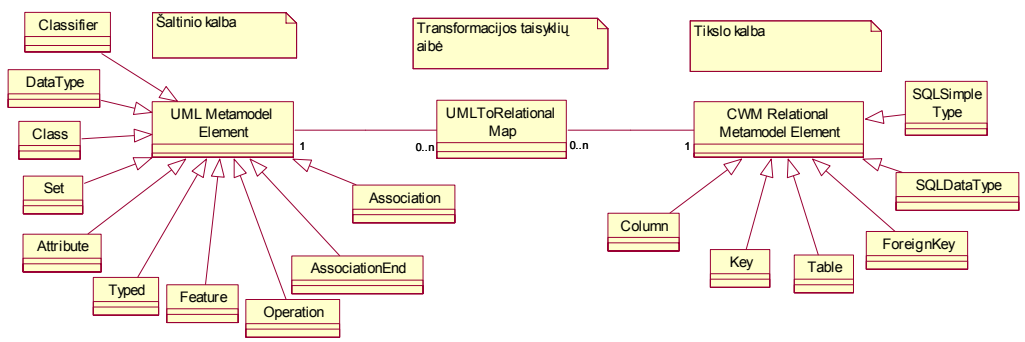
Kairėje 1 paveikslo pusėje pavaizduotas *Rational* bendrovės siūlomas metodas, naudojantis „*UML Profile for Database*“ modelius. Dirbant pagal šį metodą, pirmiausia sudaromos *UML* klasių diagramos, kurios transformuojamos į specializuotus duomenų modelius („*ConceptualCurrentToDataModel*“ paketas). Tada sistemos projektuotojas papildo šiuos modelius serverio procedūrų bei trigerių tekstais ir generuoja *RDB* schemas („*DataModelToRelational*“). Reikia atkreipti dėmesį, kad „*UML Profile for Database*“ modeliai skirti generuoti *SQL* kodui, todėl negali būti traktuojami kaip nuo platformos nepriklausomi modeliai, be to, jie naudojami tik *Rational* produktuose ir nesiremia *OMG* standartais.



1 pav. Esami ir siūlomi UML/OCL transformacijų į reliacines schemas principai

Dešinėje 1 paveikslo pusėje pavaizduotas siūlomas metodas. Čia taip pat sudaromos *UML* klasių diagramos, tačiau atsisakoma „*UML Profile for Databases*“ modelių. Vietoje jų naudojami *CWM* standarto (Common Warehouse Metamodel, 2003) reliaciniai modeliai, metamodelių transformacijų aibė bei tipinių transformacijų šablonų aibė. Metamodelių lygmenyje naudosime *egzomorfines* arba kitaip tariant *horizontalias* transformacijas – jų metu veikiama dviejuose metamodeliuose, t.y. šaltinio ir tikslo modeliai aprašomi naudojant skirtingus metamodelius. Šios transformacijos bus naudojamos transformuojant nuo platformos nepriklausomus į platformai būdingus modelius.

Metamodelių transformacijų (Kleppe, 2003) metu reikia susieti kiekvieną šaltinio modelio elementą su generuojamo modelio elementais. Paprasčiausias būdas tai padaryti – susieti šaltinio modelio elemento metaklasę su generuojamo modelio elemento metaklase. Tokioms sąsajoms specifikuoti naudojamos modelių transformacijų kalbos. *OMG* yra pateikusi užklausimą pasiūlymams tokios kalbos standartui (*MOF 2.0* Query/Views/Transformations RFP, ad/2004-04-10). Viena iš siūlomų kalbų yra *TRL* (*angl. Transformation Rule Language, OMG* dokumentas ad/2003-08-05). Ji skirta atlikti modelių užklausoms bei specifikuoti metamodelių transformacijoms *MOF 2.0* bazėje. Ši kalba – *OCL 2.0* išplėtimas, todėl visos operacijos, veikiančios *OCL* kalboje, veikia ir *TRL* kalboje. *TRL* kalba gana sudėtinga ir sunkiai skaitoma, todėl darbe naudojama paprastesnė transformacijų kalba (jos sintaksė yra detaliai aprašyta, tačiau čia nepateikiama dėl vietos stokos). 2 paveiksle grafiškai iliustruotos metamodelių transformacijos naudojant transformacijų taisyklių aibę (ši taisyklių aibė taip pat yra detaliai aprašyta).

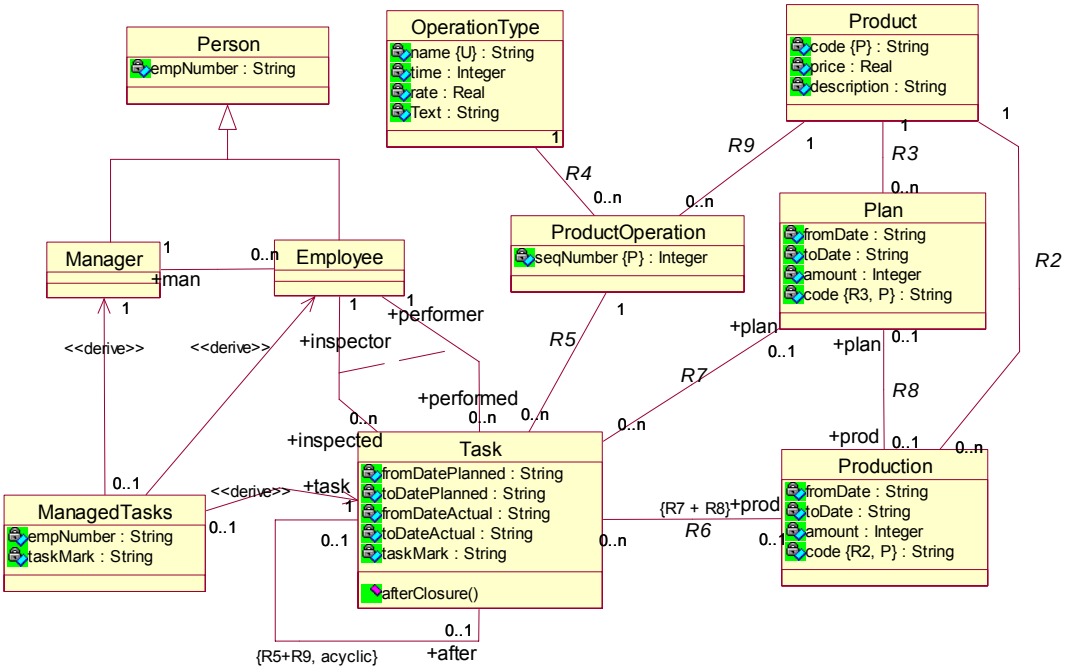


2 pav. Metamodelių transformacijos naudojant transformacijų taisykles

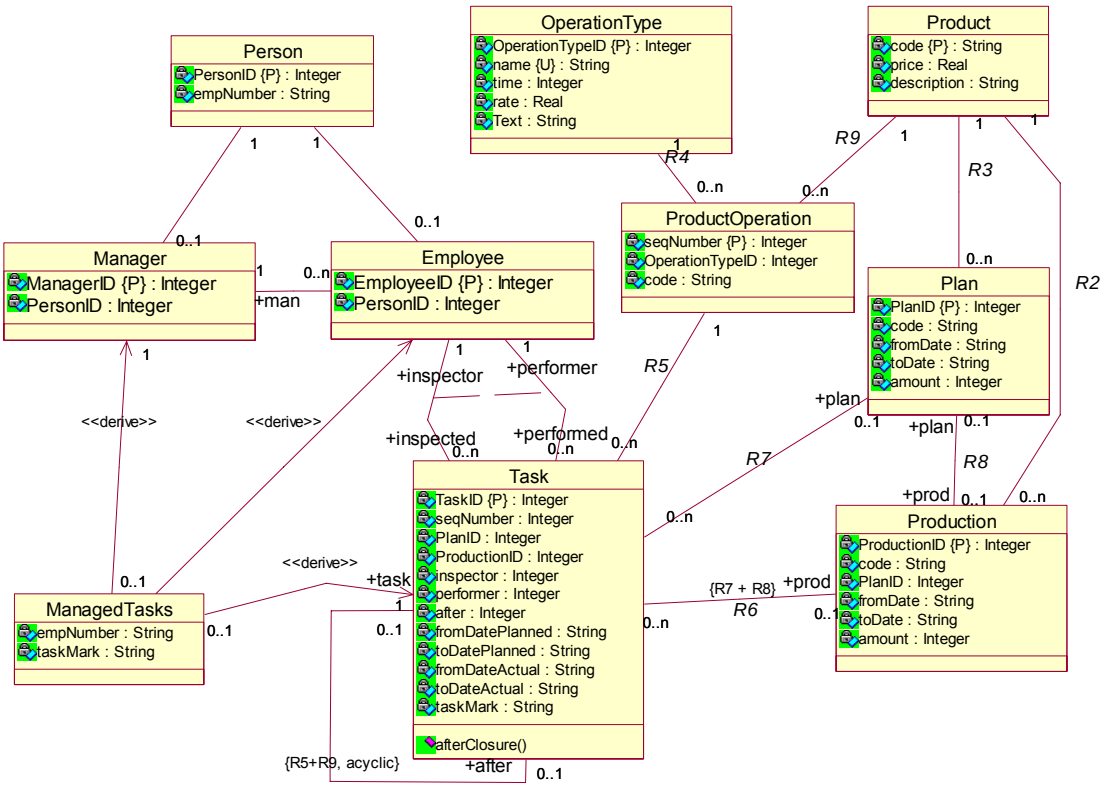
Tipiniai transformacijų šablonai taikomi jau atlikus transformacijas metamodelių lygmenyje. Ten kur negalima pritaikyti transformacijų metamodelių lygmenyje, taikomi tipiniai transformacijų šablonai. Tokios situacijos dažniausiai susidaro tada, kai *UML* modelių elementams specifikuojami *OCL* apribojimai. Jų transformuoti metamodelių lygmenyje negalima. Pavyzdžiui, vaizdų generavimui darbe naudotos *OCL* „*derive*“ išraiškos. 5 skyrelyje pademonstruotas vaizdų generavimas naudojant vieną iš sukurtų transformacijų.

5. TRANSFORMACIJOS PAVYZDYS

Transformacijai iš *UML/OCL* į *SQL* kodą pademonstruoti naudosime gamybos informacijos sistemos modelį, pritaikytą iš (Miliauskaitė, 2005) bei dalį ten aprašytų *OCL* apribojimų. Šios sistemos *PIM* ir *PSM* modeliai pateikti 3 ir 4 paveiksluose.



3 pav. Gamybos informacijos sistemos PIM lygmens dalykinės srities klasių diagrama



4 pav. Gamybos informacijos sistemos PSM lygmens dalykinės srities klasių diagrama

3 paveiksle pavaizduotas dalykinės srities *PIM* modelis. Šis modelis nepriklausomas nuo realizacijos, todėl gali būti naudojamas ne vien reliacinėms schemos generuoti. Tuo tarpu 4 paveiksle pavaizduotas *PSM* lygmens modelis skirtas reliacinių duomenų bazių schemos generavimui. Jis gaunamas transformuojant *PIM* lygmens *UML* modelius į *CWM* reliacinius modelius metamodelių lygmenyje. Tokios transformacijos metu atliekamas: klasių (taip pat ir asociacijų klasių) transformavimas į lenteles; asociacijų (taip pat ir priklausančių asociacijų klasėms) transformacijos į išorinius raktus; *UML* atributų transformacijos į duomenų bazės lentelių stulpelius; išorinių raktų stulpelių generavimas; *UML/OCL* paprastųjų ir sudėtinių tipų transformacijos į *SQL* tipus. Po transformacijos metamodelių lygmenyje, atliekamos šabloninės transformacijos: apribojimų lentelėms generavimas iš *OCL* invariantų; vaizdų generavimas iš *UML*/*OCL* modelių; serverio procedūrų generavimas iš *OCL* išraiškų.

Šiame pavyzdyje pademonstruosime transformacijas metamodelių lygmenyje bei keletą šabloninės transformacijos taikymo pavyzdžių: apribojimų lentelėms generavimą iš *OCL* invariantų bei vaizdų generavimą iš *UML/OCL*. Generuojant vaizdą, bus atliktas atributų pervardijimas, pademonstruotas *JOIN* sakinio, rezultatų apjungimo konstrukcijos *UNION* bei sąlygos sakinio *WHERE* generavimas.

Aprašysime keletą *OCL* apribojimų šioms klasių diagramoms (1 lentelė).

1 lentelė. Dalykinės srities klasių diagramos OCL apribojimai

Aprašymas	Apribojimo tekstas OCL kalboje

Klasėje *Task* bent vienas iš atributų *fromDatePlanned, toDatePlanned, fromDateActual, toDateActual* turi būti apibrėžtas (turėti reikšmę)	`context Task inv datesDefined:`
	`self.fromDatePlanned.oclIsKindOf(OclVoid) = false or`
	`self.toDatePlanned.oclIsKindOf(OclVoid) = false or`
	`self.fromDateActual.oclIsKindOf(OclVoid) = false or`
	`self.toDateActual.oclIsKindOf(OclVoid) = false`

Jei produkcijai pagaminti reikalingos užduotys suplanuotos, tai klasės Task egzemplioriai, turintys asociacijas R7 ir R8, turi ir asociaciją R6	`context Task`
	`inv certainlyPlanned:`
	`if self.prod->notEmpty() and self.prod.plan -> notEmpty()`
	`then self.plan = self.prod.plan endif`

Produkto gamybos užduočių sekoje užduotis atliekama tik vieną kartą ir negali kartotis	`context Task::afterClosure( t: Task ) : Set(Task)`
	`post: result = t.after->iterate (p : Task;`
	`acc : Set(Task) = t.after |acc ->`
	`if t.after->notEmpty() then`
	`if acc.includes(t) then acc`
	`else acc.union(t.afterClosure())endif endif)`
	`context Task inv:`
	`not self.afterClosure(self)->includes(self)`

Užduotį vykdo atlikėjas, o tikrina inspektorius. Jie negali būti vienas ir tas pats asmuo	`context Task`
	`inv diffPersons:`
	`not(self.performer = self.inspector)`

ManagedTasks klasė yra išvestinė. Šios klasės egzemplioriai nusako visus gamybos vadovus, atsakingus už užduočių atlikimą 2005.01.01.	`context ManagedTasks::taskMark : String`
	`derive: self.task.taskMark`
	`context ManagedTasks::empNumber : String`
	`derive:self.task.inspector.man.empNumber`
	`context ManagedTasks::empNumber : String`
	`derive: self.task.performer.man.empNumber`
	`context ManagedTasks inv Salyga:`
	`self.task.fromDatePlanned = '2005-01-01'`

Šiems apribojimams pritaikysime transformacijas, aprašytas darbe ir sugeneruosime atitinkamą *SQL* kodą. Jis pateiktas 2 lentelėje.

2 lentelė. SQL kodo fragmentai, gauti transformacijų met.

OCL apribojimo pavadinimas	**SQL kodas**
datesDefined	`CHECK fromDatePlanned IS NOT NULL OR toDatePlanned IS NOT NULL OR fromDateActual IS NOT NULL OR toDateActual IS NOT NULL`
diffPersons	`CHECK NOT performer = inspector`

Salyga, „derived" išraiškos	`CREATE VIEW ManagedTasks AS SELECT task.taskMark, person.empNumber`
	`FROM Tasks task`
	`INNER JOIN Employee on task.inspector = Employee.EmployeeID`
	`INNER JOIN Manager man on Employee.ManagerID = Manager.ManagerID`
	`INNER JOIN Person person on Manager.PersonID = Person.PersonID`
	`WHERE Task.fromDatePlanned = '2005-01-01'`
	`UNION ALL`
	`SELECT task.taskMark, person.empNumber FROM Tasks task`
	`INNER JOIN Employee on task.performer = Employee.EmployeeID`
	`INNER JOIN Manager man on Employee.ManagerID = Manager.ManagerID`
	`INNER JOIN Person person on Manager.PersonID = Person.PersonID`
	`WHERE Task.fromDatePlanned = '2005-01-01'`

6. IŠVADOS

Nors reliacinių duomenų bazių schemos jau seniai generuojamos iš objektinio modelio, tačiau sugeneruojama tik dalis galimo SQL kodo; konceptualieji apribojimai specifikuojami detalaus projektavimo metu realizacijos lygmens išraiškomis išraiškomis ir prieinami tik reliacinėms technologijoms. Šiame darbe susisteminti pilnaverčių reliacinių duomenų bazių schemų generavimo principai. Skirtingai nuo praktikoje naudojamo „*UML Profile for Databases*", schemos generavimas grindžiamas konceptualiuoju, nuo platformos nepriklausomu modeliu (PIM) ir OMG standartais. Objektinis modelis su apribojimais, nepriklausomas nuo konkrečios technologijos, nėra apribotas vien tik schemos generavimu ir leidžia gauti pilnavertį *SQL* kodą. Schemoms generuoti pagal MDA principus naudojamos UML metamodelių transformacijos ir tipiniai šablonai; pateikti tokių transformacijų pavyzdžiai.

7. LITERATŪROS SĄRAŠAS

Jacobson I., Booch G., Rumbaugh J. The Unified Modeling Language User Guide. Boston: Addison Wesley, 2000.

Demuth, B., Hussmann, H. Using UML/OCL Constraints for Relational Database Design. UML 1999, The Unified Modeling Language, Proc. 2nd International Conference, Springer LNCS 1723, 1999: pp. 598-613.

Akehurst D.H., Bordbar B., 2001. On Querying UML data models with OCL, in: UML 2001: The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, LNSC 2185, 2001

Gogolla M., Richters M. Expressing UML class diagrams properties with OCL. Lecture Notes in Computer Science, Vol. 2263, 2002: pp.85-114.

Sosunovas, S., Vasilecas, O. Verslo taisyklių modelių transformacija meta-modelių pagrindu. Informacinės Technologijos'2004, Technologija, Kaunas, 2004: pp.585-590.

Kleppe A., Warmer J., Bast W. The Model Driven Architecture: Practice and Promise. Addison Wesley, Boston, 2003.

Miliauskaite, E., Nemuraite L. Taxonomy Of Integrity Constraints In Conceptual Models. In: Proceedings of IADIS Database Systems 2005, http://www.iadis.org/multi2005/program_multi2005.htm.

Alagic S., Bernstein P.A. A Model Theory for Generic Schema Management. In: Lecture Notes in Computer Science, Vol. 2397. Springer Verlag, 2002: pp. 228-246.

Balsters H. Derived Classes as a Basis for Views in UML/OCL Data Models, University of Groningen, Report N 02A47, 2003.

MDA Guide Version 1.0.1. OMG document omg/03-06-01, 2003 [žiūrėta – 2005-04-17] Prieiga per internetą: <http://www.omg.org>

UML 2.0 Superstructure Specification. OMG document ptc/03-08-02, 2003 [žiūrėta – 2005-02-21] Prieiga per internetą: <http://www.omg.org>

UML 2.0 Infrastructure Specification. OMG document ptc/03-09-15, 2003 [žiūrėta – 2005-02-24]. Prieiga per internetą: <http://www.omg.org>

UML 2.0 OCL Specification. OMG document ptc/03-10-14, 2003 [žiūrėta – 2005-03-10]. Prieiga per internetą: <http://www.omg.org>

Common Warehouse Metamodel (CWM) specification. OMG document formal/03-03-02, 2003 [žiūrėta – 2005-02-17]. Prieiga per internetą: <http://www.omg.org>

SQL/Foundation (ISO-ANSI Working Draft) (ANSI TC NCITS H2, ISO/IEC JTC 1/SC 32/WG 3), 2003.

Generating full-fledged relational schemas using precise UML

In this paper, we briefly describe method for transforming *UML* class diagrams with *OCL* constraints to relational database schemas, thus showing the method to overtake "*UML Profile for Databases*" models. This article proposes two methods supplementing each other for generation of *RDB* schemas: metamodel-based transformations and pattern-based transformations. All transformations are based on *OMG* official standards or *RFPs* (*Request for Proposals*) and are prepared for use in *MDA* (*Model Driven Architecture*) context.

Konceptualių apribojimų transformavimo į SQL kodą principai

Andrius Armonas, vadovė Lina Nemuraitė

Kauno Technologijos Universitetas, Informacijos sistemų katedra

Straipsnyje pateikiami susisteminti UML/OCL transformacijų į reliacinius modelius principai MDA pagrindu. Nors transformacijos iš objektinių modelių į reliacines naudojamos jau seniai, jos yra supaprastintos ir daugelis problemų neišspręsta iki šiol. Pilnavertis objektinio modelio transformavimas į reliacinį galimas, naudojant viena kitą papildančias transformacijas metamodelių ir šablonų pagrindu. Straipsnyje pagrindžiami tokio transformavimo principai, pateikiami konceptualių apribojimų transformavimo į SQL kodą pavyzdžiai.

ĮŽANGA

MDA (Model Driven Architecture) [6] siūloma architektūra skirta greitai reaguoti į besikeičiančias informacinių sistemų kūrimo technologijas bei išvengti su tokiais pokyčiais susijusio sistemos perprojektavimo. *MDA* modelius gali naudoti įvairūs įrankiai, skirti generuoti duomenų bazių schemas, veikiantį programinį kodą, testavimo šablonus, įdiegimo scenarijus ir pan. Ši architektūra aprašo bendrus modelių kūrimo principus, tačiau nepateikia konkrečių transformacijų į šių dienų technologijų programinį kodą.

Viena aktualiausių yra transformacija iš objektinio modelio į reliacinį. Nors ji naudojama jau seniai, tačiau daug problemų nėra išspręsta iki šiol. Straipsnyje aprašoma, kaip *MDA PIM (Platform Independent Model)* ir *PSM (Platform Specific Model)* lygmenų modelius galima papildyti *OCL* išraiškomis, kad jiems būtų galima pritaikyti metamodelių transformacijas, o vėliau, naudojant transformacijų šablonus, būtų galima generuoti pilnavertį *SQL* kodą.

ESAMOS GALIMYBĖS GENERUOTI RELIACINIŲ DUOMENŲ BAZIŲ SCHEMAS IŠ UML/OCL MODELIŲ

Paprastai didžioji dalis praktikoje naudojamų *DBVS* – reliacinės, tačiau vis labiau įsigali objektinės orientacijos projektavimo procesai [1], taigi jie turėtų leisti kurti bei palaikyti programų sistemas, veikiančias kartu su reliacinėmis duomenų bazėmis.

Nemažai autorių [2], [3], [4] aprašo objektinių modelių klasių diagramų transformavimą į reliacinių duomenų bazių schemas. Dažniausiai šios transformacijos apsiriboja paprastu atveju *klasė → lentelė*, į transformavimo procesą įtraukiant tik atributus ir ryšius. Tokios transformacijos rezultatas labai ribotai panaudoja reliacinių duomenų bazių technologijos teikiamas galimybes. Domenų bazės integralumo užtikrinimo priemonės – trigeriai, ryšių vientisumo priemonės (*angl. referential integrity*), automatinis įvairių apribojimų tikrinimas paliekami nuošalyje.

Vienas iš praktikoje naudojamų metodų generuoti *RDB* schemas iš *UML* diagramų – *Rational* bendrovės sukurtas „*UML profile for databases*“ modelis. Naudojant „*UML profile for databases*“ metodiką, *UML* modeliai transformuojami į tarpinius modelius, pataisomi projektuotojo ir tada generuojamas *SQL* kodas *RDB* schemoms kurti ar modifikuoti. Dirbant pagal šią metodiką, iš *UML* modelių automatiškai nesugeneruojami nei trigeriai, nei serverio procedūros, nei sąlygų tikrinimo funkcijos. Taigi minėtoji metodika (ji nėra standartas), naudojama *Rational* produktuose, netinka automatiniam pilnų *RDB* schemų generavimui.

Gerai žinoma, jog veiklos taisyklių aprašymas duomenų bazių schemoje (panaudojant trigerius ir sąlygų tikrinimą) sumažina programinio kodo kiekį bei garantuoja, jog visa programinė įranga, dirbanti su konkrečia duomenų baze, veiks pagal tas pačias veiklos taisykles [5], t.y. integralumą išlaikančius apribojimus. Vienas iš metodų generuoti tokius apribojimus – *UML* naudojimas kartu su *OCL* kalba. Šio metodo privalumai:

- *UML* – plačiai paplitęs standartas, naudojamas objektinės orientacijos projektavimo procesuose. Todėl *UML* palaiko didelė dalis *CASE* įrankių.
- *OCL* – abstrakti kalba, kuria galima tiksliai apibrėžti integralumo taisykles. Tai atliekama aprašant modelių objektų invariantus. Navigavimo (*angl. navigation – oriented*) metodologija iš dalies sutampa su reliacinių užklausų koncepcija.
- Dauguma šiandieninių *RDBVS* naudoja nuosavus dialektus trigeriams ir sąlygoms (nors *SQL:2003* standartas tam apibrėžia standartinę kalbą). Naudojant *OCL*, galima specifikuoti apribojimus duomenų bazių schemoms nepriklausomai nuo konkrečios *RDBVS*.
- Apribojimų specifikavimas *OCL* kalba ir automatinis generavimas iš *OCL* į *SQL* žymiai sumažintų kūrimo laiko sąnaudas.
- *post*- ir *pre*- sąlygų specifikavimas *OCL* kalba gali būti geras atspirties taškas *SQL* užklausų generavimui (update, select sakiniai, trigeriai).

ESMINĖS OCL TRANSFORMAVIMO Į SQL PROBLEMOS

Atlikus literatūros analizę, galima išskirti esmines problemas, generuojant OCL apribojimus bei išraiškas į SQL apribojimus bei sakinius. Tai – apribojimų generavimo, transformavimo į vaizdus bei serverio procedūrų generavimo problemos.

Transformuojant *OCL* apribojimus į *SQL* apribojimus išskirtinos šios problemos:

- *OCL* invariantų transformavimas į sąlygų tikrinimą
- Klasių ir atributų transformacijos – objektų identifikavimo ir palyginimo problemos
- *OCL* navigavimo problemos
- Kai kurių *OCL* konstrukcijų, pavyzdžiui, *iterate* transformavimas į ekvivalentų *SQL* kodą
- *OCL* išraiškų užrašymo ir *SQL* sakinių generavimo sudėtingumo problemos

Esminės *OCL* išraiškų sekų transformavimo į vaizdus (*views*) problemos:

- Vaizdų aprašymo modeliai gana sudėtingi ir labai priklausantys nuo konkrečios situacijos (apibendrinimo trūkumas)
- Nėra apibrėžtas darbas su agregatais, grupavimo konstrukcijomis, subužklausomis, lentelių rezultatų jungimu (*SQL* sakinys *UNION*)

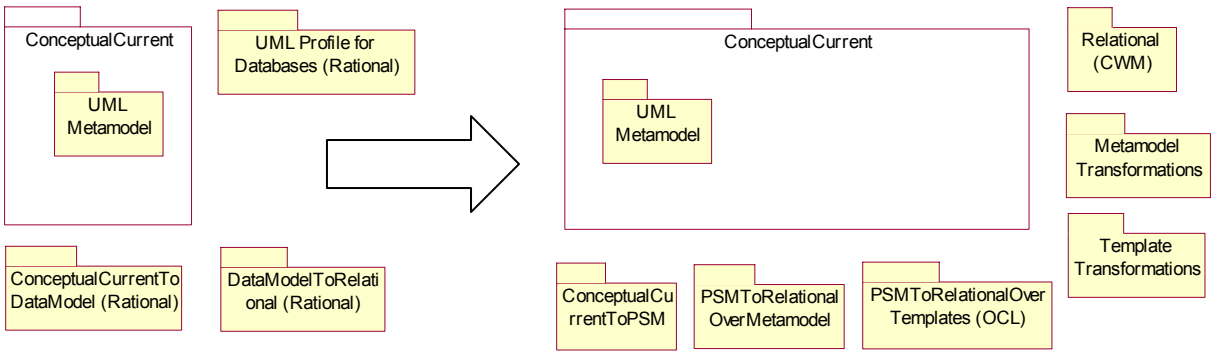
OCL transformavimas į *SQL* apribojimus, vaizdus ir serverio procedūras techniškai vienaip ar kitaip išsprendžiamas. Tačiau esminis dalykas, ko trūksta šioje srityje – bendra metodika, kaip elgtis vienais ar kitais atvejais, kuomet susiduriama su dviprasmybėmis arba *OCL-SQL* nesuderinamumais.

Šiame darbe transformacijos nagrinėjamos kombinuojant *UML*/*OCL* ir reliacinių metamodelių transformacijas bei tipinių transformacijų šablonus. Svarbus akcentas tenka *MDA* principams – konceptualieji *UML/OCL* modeliai yra nepriklausomi nuo konkrečios platformos ir gali būti pakartotinai naudojami įvairių technologijų kodui generuoti.

KONCEPTUALIŲ APRIBOJIMŲ TRANSFORMAVIMO METODŲ TOBULINIMAS

Šiuo metu praktikoje naudojamas ir šiame darbe siūloma metodas grafiškai pavaizduoti 1 paveiksle.

Kairėje 1 paveikslo pusėje pavaizduotas *Rational* bendrovės siūlomas metodas, naudojantis „*UML Profile for Database*“ modelius. Dirbant pagal šį metodą, pirmiausia sudaromos *UML* klasių diagramos, kurios transformuojamos į specializuotus duomenų modelius („*ConceptualCurrentToDataModel*“ paketas). Tada sistemos projektuotojas papildo šiuos modelius serverio procedūrų bei trigerių tekstais ir generuoja *RDB* schemas („*DataModelToRelational*“). Reikia atkreipti dėmesį, kad „*UML Profile for Database*“ modeliai skirti generuoti *SQL* kodui, todėl negali būti traktuojami kaip PIM modeliai, be to, jie naudojami tik *Rational* produktuose ir nesiremia *OMG* standartais.

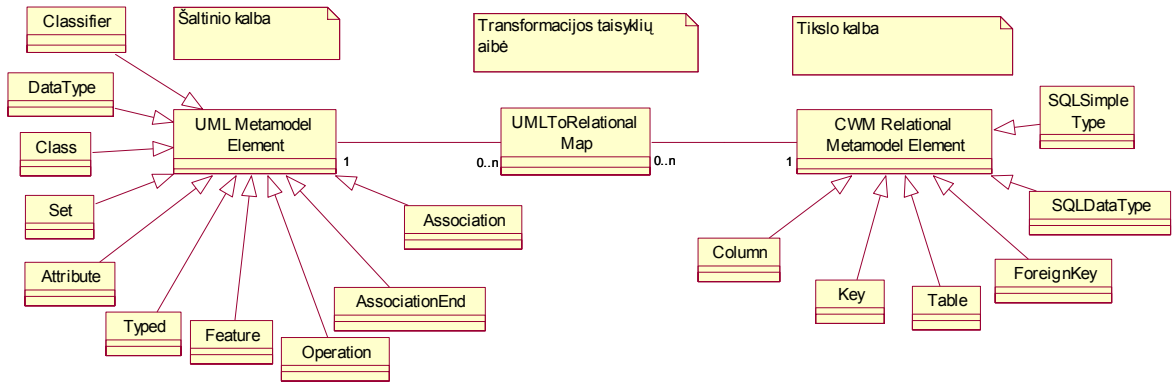


1 pav. Esama ir siūloma UML/OCL transformacijų į reliacines schemas metodika.

Dešinėje 1 paveikslo pusėje pavaizduotas siūlomas metodas. Čia taip pat sudaromos *UML* klasių diagramos, tačiau atsisakoma „*UML Profile for Databases*“ modelių. Vietoje jų naudojami reliaciniai modeliai (iš *CWM* standarto, *CWM 1.1, ad/03-03-02*), metamodelių transformacijų aibė bei tipinių transformacijų šablonų aibė.

Metamodelių transformacijų [6] metu reikia susieti kiekvieną šaltinio modelio elementą su generuojamo modelio elementais. Paprasčiausias būdas tai padaryti – susieti šaltinio modelio elemento metaklasę su generuojamo modelio elemento metaklase. Tokioms sąsajoms specifikuoti naudojamos modelių transformacijų kalbos. *OMG* yra pateikusi užklausimą pasiūlymams tokios kalbos standartui (*MOF 2.0* Query/Views/Transformations RFP, ad/2004-04-10). Viena iš

siūlomų kalbų yra *TRL* (*angl. Transformation Rule Language, OMG* dokumentas ad/2003-08-05). Ji skirta atlikti modelių užklausoms bei specifikuoti metamodelių transformacijoms *MOF 2.0* bazėje. Ši kalba – *OCL 2.0* išplėtimas, todėl visos operacijos, veikiančios *OCL* kalboje, veikia ir *TRL* kalboje. *TRL* kalba gana sudėtinga ir sunkiai skaitoma, todėl darbe naudojama paprastesnė transformacijų kalba (jos sintaksė yra detaliai aprašyta, tačiau čia nepateikiama dėl vietos stokos). 2 paveiksle grafiškai iliustruotos metamodelių transformacijos naudojant transformacijų taisyklių aibę (ši taisyklių aibė taip pat yra detaliai aprašyta).

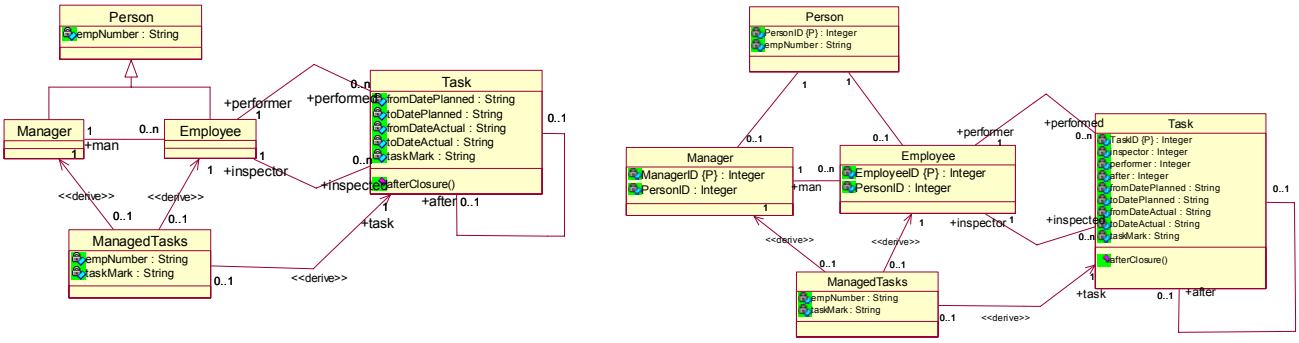


2 pav. Metamodelių transformacijos naudojant transformacijų taisykles.

Tipiniai transformacijų šablonai taikomi jau atlikus transformacijas metamodelių lygmenyje. T.y., ten kur negalima pritaikyti transformacijų metamodelių lygmenyje, taikomi tipiniai transformacijų šablonai. Tokios situacijos dažniausiai susidaro tada, kai *UML* modelių elementams specifikuojami *OCL* apribojimai. Jų transformuoti metamodelių lygmenyje negalima. Pavyzdžiui, vaizdų generavimui darbe panaudotos *OCL* „*derive*“ išraiškos. 5 skyrelyje pademonstruotas vaizdų generavimas naudojant vieną iš sukurtų transformacijų.

TRANSFORMACIJOS PAVYZDYS

Transformacijai iš *UML/OCL* į *SQL* kodą pademonstruoti pateikiama gamybos informacijos sistemos modelio dalis: *PIM* ir *PSM* (3 pav.) lygmenų modeliai.



3 pav. Gamybos informacijos sistemos PIM ir PSM lygmens dalykinės srities klasių diagramos.

3 paveiksle kairėje pavaizduotas dalykinės srities *PIM* modelis. Šis modelis nepriklausomas nuo realizacijos, todėl gali būti panaudotas ne vien reliacinėms schemos generuoti. Tuo tarpu 3 paveiksle dešinėje pavaizduotas *PSM* lygmens modelis, kuris gaunamas iš *PIM* lygmens modelio metamodelių transformacijų metu. *PSM* lygmens modelis skirtas reliacinės schemos generavimui. Aprašysime keletą *OCL* apribojimų šioms klasių diagramoms (1 lentelė).

1 lentelė. Dalykinės srities klasių diagramos OCL apribojimai.

Aprašymas	Apribojimo tekstas OCL kalboje

Klasėje *Task* bent vienas iš atributų *fromDatePlanned, toDatePlanned, fromDateActual, toDateActual* turi būti apibrėžtas	`Context Task inv datesDefined:`
	`self.fromDatePlanned.oclIsKindOf(OclVoid) = false or`
	`self.toDatePlanned.oclIsKindOf(OclVoid) = false or`
	`self.fromDateActual.oclIsKindOf(OclVoid) = false or`
	`self.toDateActual.oclIsKindOf(OclVoid) = false`

Užduotį vykdo atlikėjas, o tikrina inspektorius. Jie negali būti vienas ir tas pats asmuo	`Context Task inv diffPersons:` `not(self.performer = self.inspector)`

ManagedTasks klasė yra išvestinė. Šios klasės egzemplioriai nusako visus gamybos vadovus, atsakingus už užduočių atlikimą 2005.01.01.	`context ManagedTasks::taskMark : String`
	`derive: self.task.taskMark`
	`context ManagedTasks::empNumber : String`
	`derive:self.task.inspector.man.empNumber`
	
	`context ManagedTasks::empNumber : String`
	`derive: self.task.performer.man.empNumber`
	
	`context ManagedTasks inv Salyga:`
	`self.task.fromDatePlanned = '2005-01-01'`

Šiems apribojimams pritaikysime transformacijas, aprašytas darbe ir sugeneruosime atitinkamą *SQL* kodą. Jis pateiktas 2 lentelėje.

2 lentelė. SQL kodo fragmentai, gauti transformacijų metu.

OCL apribojimo pavadinimas	SQL kodas
datesDefined	`CHECK fromDatePlanned IS NOT NULL OR toDatePlanned IS NOT NULL OR fromDateActual IS NOT NULL OR toDateActual IS NOT NULL`
diffPersons	`CHECK NOT performer = inspector`

Salyga, „derived“ išraiškos	`CREATE VIEW ManagedTasks AS SELECT task.taskMark, person.empNumber`
	`FROM Tasks task`
	`INNER JOIN Employee on task.inspector = Employee.EmployeeID`
	`INNER JOIN Manager man on Employee.ManagerID = Manager.ManagerID`
	`INNER JOIN Person person on Manager.PersonID = Person.PersonID`
	`WHERE Task.fromDatePlanned = ‘2005-01-01‘`
	`UNION ALL`
	`SELECT task.taskMark, person.empNumber FROM Tasks task`
	`INNER JOIN Employee on task.performer = Employee.EmployeeID`
	`INNER JOIN Manager man on Employee.ManagerID = Manager.ManagerID`
	`INNER JOIN Person person on Manager.PersonID = Person.PersonID`
	`WHERE Task.fromDatePlanned = ‘2005-01-01‘`

IŠVADOS

Šiame darbe atlikta metodų, skirtų reliacinių duomenų bazių schemų generavimui iš objektinių modelių, analizė bei pasiūlytas metodas, kurį naudojant ir išplėtojant galima pašalinti esamus reliacinių schemų generavimo trūkumus: siūlomu metodu sudaryti modeliai nepriklauso nuo konkrečios technologijos (nėra orientuoti išskirtinai į *RDB* schemų generavimą), yra kuriami naudojantis tik *OMG* standartais ir leidžia generuoti pilnavertį *SQL* kodą.

LITERATŪROS SĄRAŠAS

[1] **Jacobson I., Booch G., Rumbaugh J.** The Unified Modeling Language User Guide. Boston: Addison Wesley, 2000.

[2] **Demuth, B., Hussmann, H.** Using UML/OCL Constraints for Relational Database Design UML 1999 The Unified Modeling Language, Proc. 2nd International Conference, Springer LNCS 1723, pp. 598-613, 1999.

[3] **Akehurst D.H., Bordbar B.,** 2001. On Querying UML data models with OCL, in: UML 2001: The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001 (LNSC 2185).

[4] **Gogolla M., Richters M.** Expressing UML class diagrams properties with OCL. Lecture Notes in Computer Science, Vol. 2263, pp.85-114, 2002.

[5] **Sosunovas, S., Vasilecas, O**. Verslo taisyklių modelių transformacija meta-modelių pagrindu. Informacinės Technologijos'2004, Technologija, Kaunas, p.585-590.

[6] **Kleppe A., Warmer J., Bast W.** The Model Driven Architecture: Practice and Promise. Boston: Addison Wesley, 2003.

Principles of transformation of conceptual constraints to SQL code

In this paper, the method is proposed for transformation of *UML* class diagrams with *OCL* constraints to relational database schemas. All transformations are based on *OMG* official standards or *RFPs* (*Request for Proposals*) and are prepared for use in *MDA* (*Model Driven Architecture*) context. This article describes two methods supplementing each other for generation of *RDB* schemas: metamodel based transformations and pattern-based transformations.