KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Edvinas Šinkevičius

SBVR veiklos žodynų transformacijų į UML sudarymas ir tyrimas

Magistro darbas

Darbo vadovas

prof. dr. Rimantas Butleris

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Edvinas Šinkevičius

SBVR veiklos žodynų transformacijų į UML sudarymas ir tyrimas

Magistro darbas

Recenzentas
doc. dr. Vytautas Pilkauskas
2010-05-31

Darbo vadovas
prof. dr. Rimantas Butleris
2010-05-31

Atliko
IFM-4/4 gr. stud.
Edvinas Šinkevičius
2010-05-31

Kaunas, 2010

SUMMARY

This work was initiated aiming to improve the quality of business model-based development of information systems (IS) and the quality of information systems themselves by providing the novel business rules (BR) specification method and engineering solutions of this method. The end result of the work has been made available to everybody interested and is ready for immediate application in practice. Not many SBVR implementations were developed yet, but the popularity of SBVR is growing as different interest-groups are finding the variety of ways for applying the SBVR standard. The goal of the work is to show how to transform SBVR Business Vocabularies into UML class models.

The individual transformation method and SBVR-based information systems development process using transformation of business vocabularies into the UML class model, was developed and Magic Draw plugin was made.

TURINYS

1. Įvadas 8

2. Veiklos žodynų transformacijų analizė 10

2.1. Veiklos žodynų transformacijų analizės tikslas 10
2.2. Veiklos žodynų ir taisyklių išgavimo ir įvedimo problema 10
2.3. Informacinių sistemų kūrimo, remiantis veiklos žodynais ir taisyklėmis, procesų analizė 11
2.4. Veiklos taisyklių vartotojų analizė 15
2.4.1. Veiklos taisyklių vartotojų aibė, tipai ir savybės 15
2.4.2. Veiklos taisyklių vartotojų tikslai ir problemos 15
2.5. Veiklos žodynų ir taisyklių standartų analizė 16
2.5.1. Modeliais grindžiama architektūra (MDA) 17
2.5.2. Veiklos žodyno ir taisyklių standartas (SBVR) 20
2.5.3. Produkcinių taisyklių atvaizdavimas (PRR) 24
2.5.4. OMG veiklos taisyklių standartų analizės apibendrinimas 27
2.6. Veiklos žodynų transformavimo kalbų analizė 28
2.6.1. ATL 29
2.6.2. QVT 30
2.6.3. XSLT 32
2.6.4. Veiklos žodynų transformavimo kalbos analizės apibendrinimas 33
2.7. Veiklos žodynų transformavimo siekiamas sprendimas 33
2.8. Veiklos žodynų transformavimo darbo tikslas ir uždaviniai 34
2.9. Nefunkciniai reikalavimai ir apribojimai 34
2.10. Įrankio kūrimo rizikos faktorių analizė 34
2.11. Veiklos žodynų transformacijos kokybės kriterijai 38
2.12. Analizės išvados 38

3. Veiklos žodynų transformacijos įrankio reikalavimų specifikacija ir analizė 39

3.1. Veiklos žodynų transformacijos metodo specifikacija 39
3.1.1. Įrankio kūrimo tikslų modelis 39
3.1.2. Kompiuterizuojamų panaudojimo atvejų diagrama 39
3.2. Veiklos žodynų transformacijų modeliai 42
3.2.1. Veiklos žodyno transformacijų SBVR modelis 42
3.2.2. Veiklos žodynų transformacijų UML modelis 43
3.3. Veiklos žodynų transformacijos metodo reikalavimų analizės apibendrinimas 44

4. Veiklos žodynų transformacijos įskiepio projektas 45

4.1. Įrankio architektūros projektas 45
4.1.1. Įrankio loginė architektūra 45
4.1.2. Vartotojo paslaugos 45
4.1.3. Veiklos paslaugos 46
4.2. Sukurto įrankio detalus projektas 47
4.3. Įrankio elgsenos modelis 50

5. Veiklos žodynų transformacijos įskiepio realizacija 52

5.1. Prototipo veikimo aprašymas 52
5.2. Sukurto metodo testavimas 57

6. Veiklos žodynų transformacijos metodo eksperimentinis tyrimas 59

6.1. Termino transformavimas 59
6.2. Faktų tipo transformavimas 60
6.3. isPropertyOf faktų tipo transformavimas 62
6.4. Veiklos žodynų transformavimo eksperimento apibendrinimas 64

7. Išvados 65

8. Literatūra 66

Terminų ir santrumpų žodynėlis 69

9. Priedai 70

1 priedas. Mokslinis straipsnis tarptautinėje konferencijoje „Information Technologies'2010" .. 70
2 priedas. Mokslinis straipsnis konferencijoje „Informacinė visuomenė ir universitetinės studijos (IVUS 2010)" 78
3 priedas. Mokslinis straipsnis konferencijoje „Informacinė visuomenė ir universitetinės studijos (IVUS 2010)" 84
4 priedas. Mokslinis straipsnis konferencijoje „Informacinės technologijos 2009" 90
5 priedas. Veiklos žodynų transformacijos SBVR2UML specifikacija ATL kalboje 94
6 priedas. VeTIS įrankio mokomoji medžiaga, demonstracija 112
7 priedas. Tyrimo planas 2008-2010 metais 126

PAVEIKSLŲ SĄRAŠAS

2.1 pav. SBVR į UML transformacijos esmės paaiškinimas 11
2.2 pav. Dabartinis veiklos procesas 12
2.3 pav. Veiklos modeliais grindžiamas kompiuterizuotas informacinių sistemų kūrimo procesas, jungiantis veiklos taisyklių koncepciją ir MDA 14
2.4 pav. MDA transformacijos „PIM → PSM → Kodas" 18
2.5 pav. MDA transformacijos „CIM → PIM → PSM → Kodas" 19
2.6 pav. Penki pagrindiniai SBVR aspektai [20] 21
2.7 pav. RuleML specifikacijoje apibrėžta taisyklių klasifikacija 25
2.8 pav. ATL transformacijos schema 30
2.9 pav. QVT UML klasių transformacija į duomenų bazių modelį 31
2.10 pav. XSLT panaudojimas 32
2.11 pav. SWOT schema 35
2.12 pav. SWOT schema po vienerių metų 37
3.1 pav. SBVR veiklos žodynų transformacijų į UML metodo kūrimo tikslai 39
3.2 pav. Kompiuterizuojamų panaudojimo atvejų diagrama 39
3.3 pav. PA „Business rules specification" scenarijus 42
3.4 pav. Supaprastintas SBVR metamodelio fragmentas 43
3.5 pav. SBVR modelis kuris transformuojamas į UML modelį 43
3.6 pav. Supaprastintas UML metamodelio fragmentas 44
3.7 pav. UML modelis 44
4.1 pav. IS loginė architektūra 45
4.2 pav. IS vartotojo sąsajos planas 46
4.3 pav. IS veiklos paslaugų valdikliai 46
4.4 pav. Sistemos klasių diagrama 47
4.5 pav. Veiklos taisyklų specifikavimo sekų diagrama 50
4.6 pav. UML klasių diagramos būsenų diagrama 51
5.1 pav. Sistemos diegimo diagrama 52
5.2 pav. Sistemos realizavimo diagrama 54
5.3 pav. Ryšys tarp SBVR metamodelių 55
5.4 pav. SBVR metamodelių sujungimo schema 56
5.5 pav. Pagrindinės SBVR metamodelio klasės 57

LENTELIŲ SĄRAŠAS

2.1 lentelė. Veiklos taisyklių apibrėžtys 16
2.2 lentelė. SSGG analizė (SWOT) 35
2.3 lentelė. SWOT analizės įverčiai 36
2.4 lentelė. SWOT analizės po vienerių metų įverčiai 37
3.1 lentelė. PA “Business rules specification“ specifikacija 40
3.2 lentelė. PA “Specificate business rules vocabulary“ specifikacija 40
3.3 lentelė. PA “Transform to UML“ specifikacija 40
3.4 lentelė. PA „Generate code“ specifikacija 41
5.1 lentelė. Eclipse programoje naudojami įskiepiai 53
9.1 lentelė. Tyrimo planas 2008-2010 metais 126

1. Įvadas

Tiriamojo darbo tikslas yra pagerinti informacinių sistemų ir veiklos modeliais grindžiamo kompiuterizuoto jų kūrimo procesų kokybę, pasiūlant veiklos žodynų ir taisyklių specifikavimo metodą ir jį realizuojančius inžinerinius sprendimus.

Magistrinis darbas turi išspręsti problemas dėl veiklos žodynų ir taisyklių išgavimo ir įvedimo į automatizuoto projektavimo įrankius informacinių sistemų kūrimo metu, jų transformavimo į projektines specifikacijas ir programos kodą bei reagavimo į pokyčius sistemos funkcionavimo metu. Tam tikslinga sukurti veiklos taisyklių užrašymo artima natūraliajai kalba šablonus, kuriuos naudodami analitikai ir dalykinės srities ekspertai galėtų vienareikšmiškai vieni kitus suprasti ir sukurti semantiškai vertingesnius formalius taisyklių aprašus. Šiuos aprašus būtų galima naudoti tolesniuose projektavimo etapuose, transformuojant taisykles į projektines specifikacijas ir programos kodą.

Metodas būtų pritaikomas didelei informacinių sistemų aibei ir įvairiems realizavimo metodams – vykdymui veiklos taisyklių procesoriuose, verslo būsenų mašinose, duomenų bazių trigeriais ar įvairių platformų programų kodu. Didžiausias esamų veiklos taisyklių koncepcijos metodus įgyvendinančių įrankių trūkumas yra tame, kad taisykles į juos reikia įvesti specifinėmis formaliomis kalbomis, kurios nėra lengvai suprantamos probleminės srities atstovams, be to, esami metodai paprastai tinka tik tam tikro tipo informacinėms sistemoms, kurios turi dideles taisyklių aibes. Darbo metu pasiektas tyrimų lygis sudarytu prielaidas sukurti vartotojui draugiškas taisyklių specifikavimo priemones, kurios leistų vienareikšmiškai suprasti, tikrinti, modifikuoti ir įgyvendinti veiklos taisykles įvairiose informacinėse sistemose nuo veiklos koncepcijų iki kodo.

Atlikus veiklos žodynų ir taisyklių standartų analizę buvo pasirinktas SBVR standartas. SBVR yra pirmasis OMG grupės standartas, skirtas išimtinai veiklos žodynams ir veiklos taisyklėms specifikuoti. Nors standartas yra jau keletas metų kaip patvirtintas, tačiau įrankių, kurie jį taikytų yra labai nedaug – galima teigti, kad standartas kol kas dar nesusilaukė didelio pripažinimo. Veiklos žodyno ir veiklos taisyklių semantikos standartas SBVR yra OMG specifikacija, kuri išreiškia veiklos žinias veiklos dalyviams suprantama kalba ir leidžia užrašyti jas notacija, nereikalaujančia specialių IT žinių. Veiklos žodynai ir veiklos taisyklės paprastai specifikuojamos tekstu. SBVR vartojama ribota natūrali kalba (angl. *Controlled Natural Language*), pavyzdžiui, anglų.

Veiklos žodynas leidžia sukurti natūralią aktyvią sąsają tarp žmonių, dalyvaujančių veikloje, ir tą veiklą kompiuterizuojančių programų sistemų. Kai veiklos taisyklėms nustatyti yra taikomi kuo

įvairiausi metodai, tolesnis veiklos ir veiklos taisyklių žodynų taikymas yra labai aktualus šiuolaikinių informacinių sistemų kūrimui, duomenų bazių integravimui, interneto paslaugų, Pasaulio semantinio tinklo ir IS projektavimo metodų plėtrai.

Darbo eigoje buvo sukurtas šio metodo projektas, realizuotas ir ištestuotas Magic Draw įskiepio prototipas. Šio metodo sukūrimas yra Lietuvos projekto „Veiklos taisyklių sprendimai informacinių sistemų kūrimui (VeTIS)“ dalis. Sukurtas įrankis realizuoja transformaciją iš veiklos žodynų semantinių aprašų į UML nuo platformos nepriklausomus (PIM) klasių modelius. Tai pirma tokios apimties šios transformacijos realizacija, apimanti beveik visus veiklos žodyno metamodelio elementus.

Be įvado magistro darbą sudaro turinys, santrauka anglų kalba, sutrumpinimų žodynas, penki skyriai, paskelbtų publikacijų ir cituotų šaltinių sąrašai, priedai.

Antrajame darbo skyriuje pristatoma esamos problemos ir vartotojų analizė, OMG konsorciumo veiklos žodynų ir taisyklių, UML ir MDA standartų, metodo esamų sprendimų analizės rezultatai, taip pat ir transformavimo procesorių analizė, rizikos faktoriai ir analizės išvados. Trečiajame skyriuje pateikiama veiklos žodynų transformacijos įrankio reikalavimų specifikacija ir analizė, tikslų modelis, kompiuterizuojamų panaudojimo atvejų modelis ir pristatomi SBVR bei UML modeliai. Ketvirtajame skyriuje aprašomas įskiepio projektas: loginė architektūra, vartotojo paslaugos, detalusis projektas ir elgsenos modelis. Penktajame skyriuje aprašoma metodo realizacija, aptariama, kaip veikia prototipas ir kaip jis buvo testuojamas. Šeštajame skyriuje atliekamas eksperimentas iškeltai hipotezei patvirtinti, kad pagrindinius UML klasių elementus (klases, klasių atributus ir ryšius) galima gauti transformavus iš ribota natūralia kalba specifikuotų konceptų. Toliau pateikiamos išvados, literatūros sąrašas, terminų ir santrumpų žodynėlis, bei priedai, kuriuose pateikiami darbo tema parašyti straipsniai ir sukurta VeTIS įrankio mokomoji medžiaga, tyrimo planas pateikiamas 7 priede.

2. Veiklos žodynų transformacijų analizė

Sparčiai tobulėjant informacinėms technologijoms ir didėjant kuriamų sistemų sudėtingumui bei skaičiui, atsirado poreikis kuo daugiau automatizuoti IS kūrimo procesą. Automatizavimo problemai spręsti naudojami modeliais grindžiamo kūrimo (angl. Model Driven Development) metodai, kurių esmė – didesnis naudojamų abstrakcijų lygis, kai vietoj programavimo kalbų naudojamos modeliavimo kalbos (UML). Šiandien, unifikuota modeliavimo kalba (UML) yra viena plačiausiai naudojamų modeliavimo kalbų specifikuojant konceptualias schemas. UML schemos gali būti papildomos tekstinėmis išraiškomis, kurios aprašytu veiklos taisykles ir kurios negalėjo būti išreikštos grafiškai [21]. Šios tekstinės išraiškos aprašomos objektinių reiškinių kalba (OCL).

2.1. Veiklos žodynų transformacijų analizės tikslas

Šios analizės tikslas yra įsitikinti ar toks metodas yra reikalingas ir ištirti ar egzistuoja jau sukurtų šio metodo analogų kuriuos būtų galima naudoti tiesiogiai arba pritaikyti. Taip pat išsiaiškinti kaip pagerėtų informacinių sistemų ir veiklos modeliais grindžiamo jų kompiuterizuoto kūrimo procesų kokybė. Išanalizuoti veiklos žodynų ir taisyklių standartus, vartotojus, įvertinti kuriamo įrankio riziką, transformacijų kalbas. Apibrėžti sistemai keliamus reikalavimus. Analizei atlikti buvo pasirinkti šie analizės metodai:

- Literatūros šaltinių studijavimas
- Esamų, jau sukurtų metodų ir jų realizacijos analizavimas
- Veiklos žodynų ir taisyklių aprašymų studijavimas

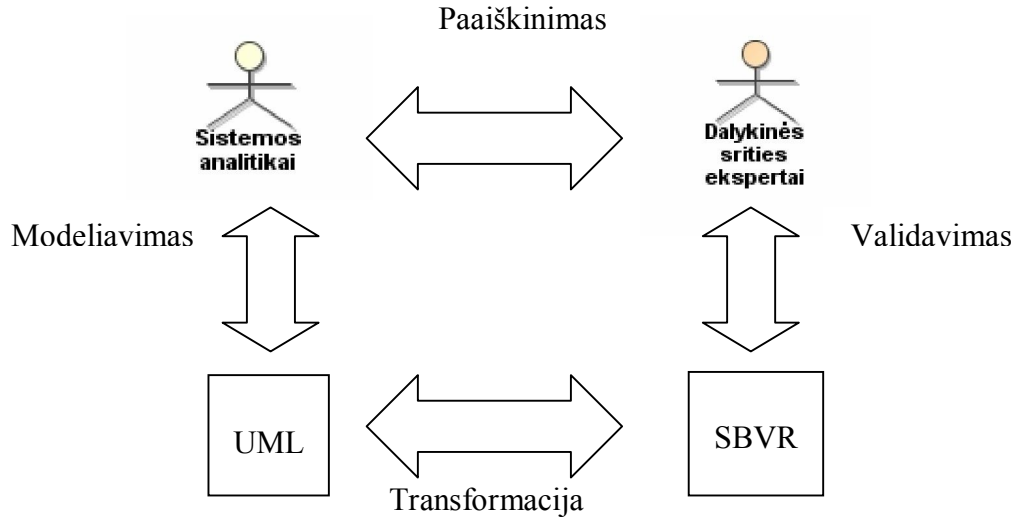
2.2. Veiklos žodynų ir taisyklių išgavimo ir įvedimo problema

Tyrimo sritis - veiklos taisyklių koncepcijos bruožai, žinomi taisyklių specifikavimo ir modeliavimo sprendimai bei standartai, specializuoti programiniai įrankiai ir jų charakteristikos.

Tyrimų objektas – veiklos taisyklių išraiškos ir specifikavimo modelių bei metodų sudarymas ir jų taikymas kuriant informacines sistemas.

Tyrimas turi išspręsti problemas dėl veiklos taisyklių išgavimo ir įvedimo į automatizuoto projektavimo įrankius informacinių sistemų kūrimo metu, jų transformavimo į projektines specifikacijas ir programos kodą bei reagavimo į pokyčius sistemos funkcionavimo metu. Tam tikslinga sukurti veiklos taisyklių užrašymo artima natūraliajai kalba šablonus [13], kuriuos naudodami analitikai ir dalykinės srities ekspertai galėtų vienareikšmiškai vieni kitus suprasti ir sukurti semantiškai vertingesnius formalius taisyklių aprašus. Šiuos aprašus būtų galima naudoti

tolesniuose projektavimo etapuose, transformuojant taisykles į projektines specifikacijas ir programos kodą.



2.1 pav. SBVR į UML transformacijos esmės paaiškinimas

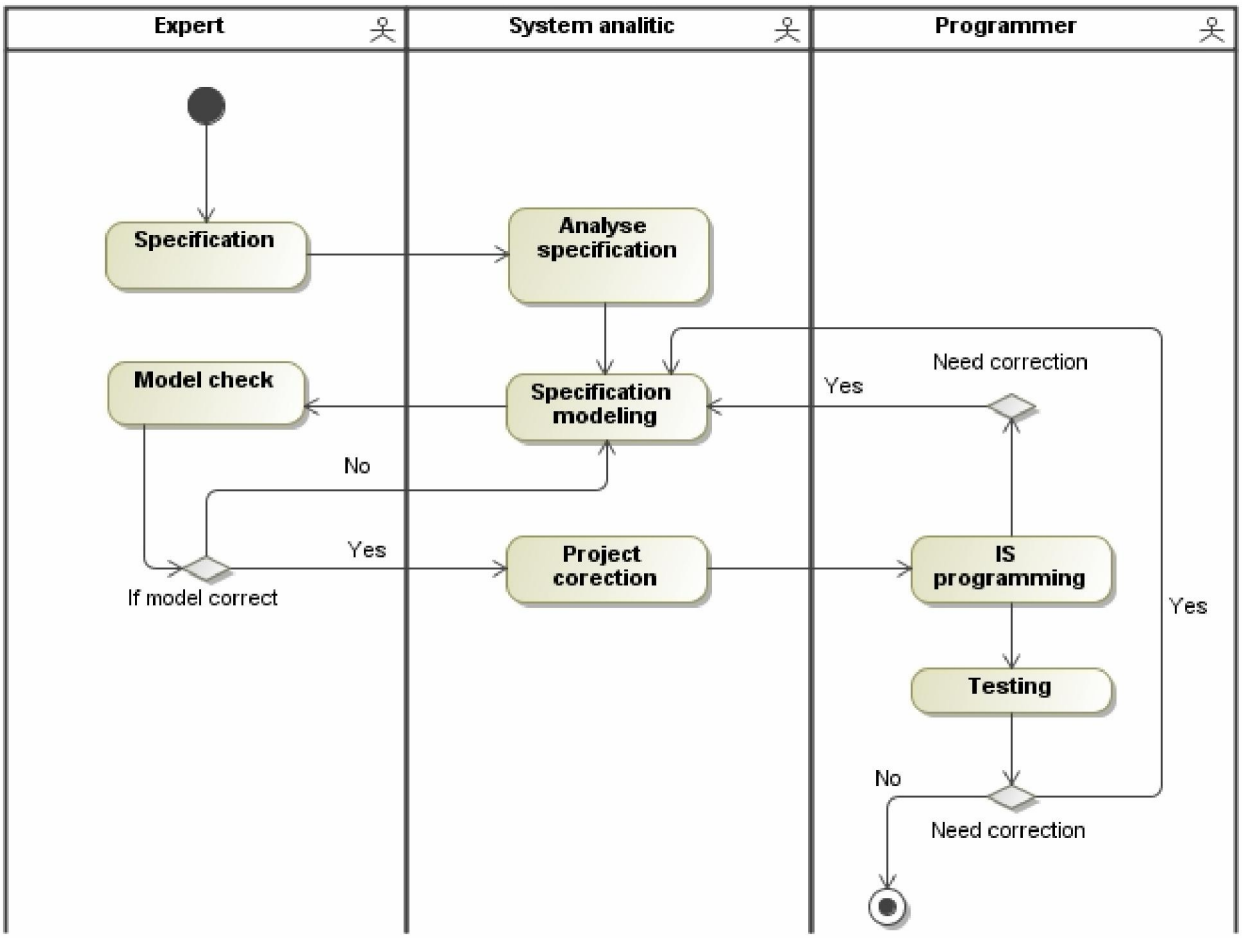
Žinoma, unifikuota modeliavimo kalba (UML) ir SBVR nėra susietos, todėl SBVR reikia transformuoti į UML, kad probleminės srities ekspertas lengvai galėtu suprasti projektuotojo sumodeliuotą modelį (supaprastintas paaiškinimas parodytas 2.1 pav.)[12].

2.3. Informacinių sistemų kūrimo, remiantis veiklos žodynais ir taisyklėmis, procesų analizė

Šiuo metu pradeda įsitvirtinti naujos kartos automatizuoto programinės įrangos kūrimo metodai – taip vadinamas modeliais grindžiamas kūrimas (MDD) [2], kai pagrindinės kūrybinės pastangos dedamos modelių lygmenyje, o didžioji dalis schemų ir programų kodo generuojama automatiškai. „Programavimas“ naudojant nuo realizacijos platformos nepriklausomus modelius (PIM) išlaisvina kūrėją nuo būtinybės žinoti visas realizacijos detales ir tampa vienintele išeitimi, kai nuolat didėjanti informacinių technologijų įvairovė ir sudėtingumas reikalauja įvaldyti ištisus kalbų ir technologijų rinkinius. Stambiausios pasaulio IT korporacijos, standartų organizacija OMG ir kitos mokslinės-pramoninės bendruomenės nuosekliai plėtoja MDD ir MDA metodus, ypatingai baigiamųjų kūrimo etapų automatizavimą: specifinių platformų modelių PSM ir programinio kodo generavimą iš PIM modelių [6].

Modeliais grindžiamo kūrimo metoduose modelių kokybei keliami griežti kokybiniai semantinio aiškumo ir vienareikšmiškumo reikalavimai. Užtikrinti šią kokybę be įrankių sunku, todėl tampa vis labiau aktualu automatizuoti ankstesnius kūrimo etapus. Probleminės srities ekspertui sunku patikrinti sistemų analitiko sudarytas specifikacijas (2.2 pav.) dėl nesuprantamos

formalios specifikavimo kalbos (pvz., UML). Iš kitos pusės, analitikui sudėtinga patikrinti probleminės srities eksperto pateiktos informacijos pilnumą ir teisingumą dėl šiam tikslui dažniausiai naudojamos natūraliosios kalbos ypatumų (natūraliąja kalba pateikti teiginiai gali būti daugiareikšmiai, prieštaringi, nestruktūrizuoti ir pan.). Padėti jiems susikalbėti gali veiklos taisyklių specifikavimo kalba, kuri iš vienos pusės išreiškiama natūraliajai kalbai artimais sakiniais, iš kitos pusės, yra formali (tiksli ir vienareikšmiška). Šia kalba turi būti galima aprašyti visus kuriamai sistemai keliamus reikalavimus ir apribojimus, apibrėžiant juos probleminės srities žodyno sąvokomis ir neprarandant semantikos.



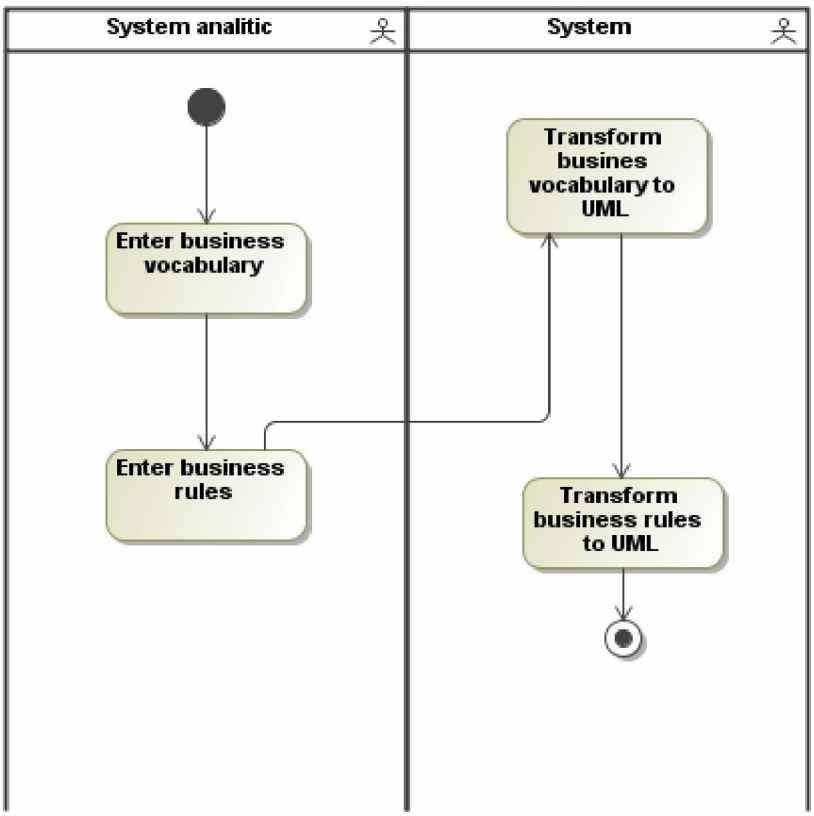
2.2 pav. Dabartinis veiklos procesas

Veiklos taisyklių koncepcija vis labiau skverbiasi į įvairiausias programinės įrangos kūrimo sritis. Norint kompiuterizuoti bent kiek sudėtingesnį uždavinį ilgesniam vartojimui, padaryti jį lankstesnį pokyčiams atsiranda veiklos taisyklių užrašymo ir atsekimo problema. Veiklos taisyklių specifikacijos iš veiklos aprašo keliauja į sistemos reikalavimų, vėliau – projektinių sprendimų specifikacijas ir įgyvendinamos programos kode. Tam galimi įvairūs požiūriai ir metodai, tačiau visų jų esminis reikalavimas – palaikyti ryšį tarp dalykinės srities semantikos ir jos atvaizdavimo

kode tam, kad probleminės srities pokyčius būtų galima greitai atsekti ir automatiškai atvaizduoti programinėje realizacijoje. Praradus šį ryšį, dubliuojamas taisyklių rinkimas ir specifikavimas, didėja klaidų tikimybė, darosi nebeįmanoma surasti pokyčių paveiktas vietas, nes jos išsisklaido po visą sistemą.

Taisyklių išsibarstymas po įvairias sistemos vietas ir sistemų nelankstumas pokyčiams yra dvi tarpusavyje susijusios problemos, kurios neišvengiamai apsunkina tiek programinės įrangos specifikacijų valdymą (kūrimą, keitimą, šalinimą) ir analizavimą IS kūrimo metu, tiek pakartotiną šių specifikacijų naudojimą. Sukurtoji sistema, pasikeitus probleminei sričiai (organizacijos veiklos logikai), taip pat turėtų keistis; kitu atveju kompiuterizuota sistema nebeatitiks esamos situacijos. Esmė tame, kad, kuriant sistemą tradiciniais metodais, veiklos logika yra įsiuvama programiniame kode ir duomenų bazėse, todėl, įvykus pasikeitimams organizacijos veiklos logikoje, modifikuoti tokią informacinę sistemą yra itin sudėtinga.

Minėtos problemos gali būti sprendžiamos pritaikant IS kūrimo procese veiklos taisyklių koncepciją, kuri mažina nesusikalbėjimo barjerą tarp IS kūrėjų ir užsakovų (leidžia tiksliai, nedviprasmiškai ir visiems suprantama forma specifikuoti vartotojo reikalavimus) bei sudaro prielaidas kurti lankstesnes informacines sistemas (2.3 pav.). Veiklos taisyklėmis grindžiamų IS kūrimo metodų kūrimo sritis šiuo metu yra labai sparčiai vystoma, tačiau didžiausi pasiekimai yra įgyvendinti tik taip vadinamuose veiklos taisyklių procesoriuose (angl. Business Rule Engines), kurie skirti produkcinių taisyklių vykdymui ir tinka tam tikro tipo informacinėms sistemoms. Minėtieji taisyklių procesoriai yra tik viena galima veiklos taisyklių realizacija. Dažniau reikalingas veiklos taisyklių pokyčių palaikymas duomenų bazėse, tinklo programinės įrangos paslaugų ir komponentų sistemose. Tiek tradiciniuose, tiek naujausiuose modeliais grindžiamo kūrimo metoduose veiklos taisyklių koncepcija taikoma ribotai, esami sprendimai yra daugiau eksperimentinio lygmens. Veiklos taisykles šiuo metu galima užrašyti tik nedaugelyje CASE įrankių (natūraliąja kalba arba OCL UML CASE įrankiuose), kai kuriuose įrankiuose galima jas dalinai patikrinti, tačiau gyvavimo ciklas nuo įvedimo iki kodo nėra užtikrinamas. Ten, kur galima įvesti taisykles natūraliąja kalba, jos nėra formalios, o formaliam užrašymui naudojama OCL kalba nėra draugiška dalykinės srities ekspertams. Taigi vis labiau aktualus taisyklių palaikymas nuo veiklos koncepcijų iki kodo visame informacinės sistemos gyvavimo cikle nėra išspręstas, dalinai jis sprendžiamas tam tikrų tipų sistemų realizavimo etapuose.



2.3 pav. Veiklos modeliais grindžiamas kompiuterizuotas informacinių sistemų kūrimo procesas, jungiantis veiklos taisyklių koncepciją ir MDA

Pastaruoju metu smarkiai plečiasi semantinio pasaulio tinklo (angl. Semantic Web) galimybės. Semantinio tinklo metodai, intensyviai naudojantys ontologijas ir taisykles, skverbiasi ir į esamas informacines sistemas. Ontologinė analizė gali padėti išgauti tekstiniuose dokumentuose užrašytus reikalavimus – įmonėse naudojami didžiuliai tokių dokumentų kiekiai. Tai dar viena galimybė pagerinti veiklos taisyklių koncepcija besivadovaujančius metodus, sukuriant priemones veiklos taisyklių išgavimui iš natūraliosios kalbos tekstų bei projektuojant taisyklių kalbą suderinamą su semantinio tinklo taisyklėmis. Pastarasis reikalavimas svarbus įvairios paskirties sistemų sąveikai (angl. Interoperability).

Apibendrinant galima teigti, kad sukurtasis metodas pagerintų informacinių sistemų kokybę ir pagerintų kūrimo procesą (padidintų jo našumą dėl ankstesnių etapų automatizavimo, taisyklių išgavimo iš tekstinių dokumentų, pakartotino taisyklių naudojimo, sumažintų dubliavimą ir pan.).

Inžineriniai darbo rezultatai turėtų įtakos plėtojant smulkių ir vidutinių IT įmonių verslą ne tik Lietuvos, bet ir pasaulio kontekste.

2.4. Veiklos taisyklių vartotojų analizė

Sukurtu metodu naudosis trijų tipų vartotojai – informacinių technologijų bendrovės, kurie kuria programinę įrangą, universitetai savo moksliniuose tyrimuose ir vyriausybinės organizacijos, turinčios didelę aibę veiklos taisyklių.

2.4.1. Veiklos taisyklių vartotojų aibė, tipai ir savybės

Pagrindinė suinteresuotų vartotojų grupė (potencialūs vartotojai) yra informacinių technologijų (IT) bendrovės, kurių veiklos produktai yra programinės įrangos projektai ir programinė įranga, ypač jei ji gaminama plačiu mastu, nes tuomet daugkartinis taisyklių ir modelių panaudojimas įgauna dar didesnę prasmę.

Tiesiogiai su IT bendrovėmis yra susijusios užsakovų organizacijos, kurioms kuriama programinė įranga – tai gali būti bet kuri Lietuvos ar užsienio šalies įmonė, ketinanti kompiuterizuoti savo veiklos procesus.

Dar viena suinteresuotų organizacijų grupė yra universitetai, plėtojantys naujus mokslinius-technologinius pasiekimus savo moksliniuose tyrimuose ir taikantys juos praktikoje. Veiklos taisyklių surinkimas, specifikavimas ir automatizavimas tam pritaikytais įrankiais, prasmingos informacijos išgavimas taisyklių pavidale iš tekstinių dokumentų aktualus ir pritaikomas daugelyje sričių: teisėje, medicinoje, finansų analizėje, transporto valdyme ir kt.

Galiausiai, sukurtasis veiklos taisyklių surinkimo ir struktūrizavimo metodas pravartus vyriausybės, viešųjų ir visuomeninių institucijų veiklos kompiuterizavimui, nes čia itin sunku prižiūrėti didelę dažnai perteklinių ir prieštaringų, iš įvairių šaltinių ateinančių taisyklių aibę.

2.4.2. Veiklos taisyklių vartotojų tikslai ir problemos

Metodas ir jo programinė realizacija įgalintų IT kompanijų sistemų analitikus ir projektuotojus:

- tiksliai specifikuoti kuriamas sistemas ir suderinti šias specifikacijas su probleminės srities ekspertais (organizacijų, kurioms kuriamos sistemos, atstovais);
- kurti informacines sistemas, lanksčiai pritaikomas prie nuolat dažnėjančių organizacijų ir jų aplinkos pokyčių.

Programiškai realizavus metodą, probleminės srities ekspertai (įmonių atstovai) gautų priemonę, leidžiančią dalykiškai ir nedviprasmiškai komunikuoti su IT bendrovėmis – visų pirma, tiksliai ir nedviprasmiškai išreikšti ir specifikuoti savo reikalavimus būsimajai programinei įrangai.

Veiklos taisyklių automatizavimas padidina galimybes pritaikyti eksploatuojamas informacines sistemas naujiems poreikiams ir greitai reaguoti į pokyčius be programuotojų pagalbos.

2.5. Veiklos žodynų ir taisyklių standartų analizė

Iki šiol nėra vieningos nuomonės klausimais, susijusiais su VT klasifikavimu, identifikavimu ir atvaizdavimu [4]. Tuo pačiu nėra ir vieningos VT apibrėžties (2.1 lentelė).

2.1 lentelė. Veiklos taisyklių apibrėžtys

Metai, šaltinis	Apibrėžtis
1984 m., Daniel S. Appleton (Appleton, 1984)	Aiškus apribojimo teiginys, kuris egzistuoja lygiagrečiai su verslo ontologija.
1987 m., Ronald G. Ross "Entity Modeling: Techniques and Application" (Ross, 1987)	Specialios taisyklės (ar verslo politika), kurios valdo organizacijos elgseną ir išskiria ją iš kitų. Šios taisyklės valdo organizacijos būsenos pasikeitimus.
1990 m., J. Bell, D. Brooks ir kt. „Re-Engineering Case Study Analysis of Business Rules and Recommendations for Treatment of Rules in a Relational Database Environment", (apibrėžtis percituota iš (Knolmeyer ir kt., 1994)).	Teiginiai apie tai, kaip vykdoma veikla, t.y. apie gaires ir apribojimus organizacijos būsenų ir joje veikiančių procesų atžvilgiu.
1997 m., GUIDE veiklos taisyklių projektas (vėliau tapo žinomas kaip Veiklos taisyklių grupė (Business rules Group), ver. 1.2 (Hay ir kt., 1997). Aktualioje redakcijoje (Business, 2000a) apibrėžtis išliko nepakitus.	Tam tikrus veiklos aspektus apibrėžiantis arba apribojantis teiginys, kuris reikalingas norint įvertinti veiklos struktūrą arba valdyti/įtakoti veiklą. Jokia VT negali būti suskaidyta į detalesnes VT neprarandant svarbios informacijos apie veiklą.
2000 m., Veiklos taisyklių grupė, Veiklos taisyklių motyvacijos modelis (Business, 2000b).	Direktyva, skirta daryti įtaką ar valdyti verslo elgseną. Tokios direktyvos pagrindžia verslo politiką, kuri yra suformuluota kaip atsakas į galimybes, grėsmes, stiprybes ir silpnybes.
2001 m., Managing Reference Data in Enterprise Databases, Malcolm Chisholm (Ross, 2003)	Atskiras teiginys, apdorojantis duomenis ar informaciją, kurią organizacija valdo, ir išvedantis kitus duomenis ar informaciją iš šių duomenų, arba naudojantis juos tam tikro veiksmo sužadinimui.
2002 m., Barbara von Halle, "Business Rules Applied: Building Better Systems Using the Business Rule Approach" (Von Halle, 2001)	Sąlygos, kuriomis vadovaujantis veiklos įvykiai valdomi taip, kaip yra priimtina verslui.
2004 m., Veiklos taisyklių komanda, Veiklos taisyklių veiklos semantika (Business, 2004).	Patarimas (nuoroda), liečianti elgseną, veiksmą, praktiką (nusistovėjusią tvarką) ar procedūrą konkrečioje veikloje ar aplinkoje (sferoje).
2005 m., Veiklos taisyklių grupė, Veiklos motyvacijos modelis,	Direktyva, nereikalaujanti papildomo interpretavimo (t.y. vienareikšmiškai suprantama) bei garantuojanti

Metai, šaltinis	Apibrėžtis
versija 1.1 (Business, 2005b).	numatytų strategijų ir taktikų laikymąsi.

Nežiūrint į VT apibrėžčių gausą, dauguma jų gali būti padengiami apibrėžties, suformuluotos GUIDE projekte 1997 m.[11] Pagal šią apibrėžtį veiklos taisykle laikytini ne tik teiginiai ir apribojimai, reikalingi norint valdyti ar įtakoti veiklą, bet ir teiginiai, reikalingi norint apibrėžti ar įvertinti veiklos struktūrą. Tokie struktūriniai teiginiai – tai terminai (angl.: term) ir ryšius tarp terminų nusakantys faktai (angl.: fact). Tokiu būdu GUIDE VT samprata traktuotina kaip visapusiška ir universali, todėl ja bus remiamasi šiame darbe.

Veiklos taisyklių specifikavimo modeliai (sutrumpintai gali būti vadinami VT modeliais) visuomet apima VT klasifikavimo schemą, t.y. apibrėžia VT sąvoką, jos kilmę bei tipus. Detaliuose VT specifikavimo modeliuose gali būti nagrinėjamas ir VT nustatymo bei fiksavimo procesas. Griežtas VT specifikavimo modelis yra būtinas, norint kokybiškai analizuoti vartotojo poreikius bei vienareikšmiai užrašyti VT. Paprastai šiuolaikiniuose VT modeliuose išskiriami 2-6 pagrindiniai VT tipai, kurie toliau detalizuojami į potipius.

Neabejotinai ryškiausias VT modelis iš iki šiol pristatytųjų yra pateiktas grupės mokslininkų pasiūlytame GUIDE projekte. GUIDE VT modelis yra vertinamas kaip pamatinė konceptualaus pobūdžio veiklos taisyklių studija, su juo siejasi visos žinomos VT klasifikacijos. Verta pažymėti, jog VT modeliai vystosi, intensyviai kuriami ir standartai, tačiau GUIDE projekte suformuluotos tezės neturėtų prarasti savo aktualumo ir ateityje.

Šiame skyriuje pateikiami OMG grupės parengtų arba rengiamų standartų, vienaip ar kitaip susijusių su veiklos modeliavimu ir veiklos taisyklėmis, analizės rezultatai. Analizuoti šie OMG standartai ir iniciatyvos:

- Model-driven Architecture (MDA),
- Semantics of Business Vocabulary & Rules (SBVR),
- Production Rules Representation (PRR).

2.5.1. Modeliais grindžiama architektūra (MDA)

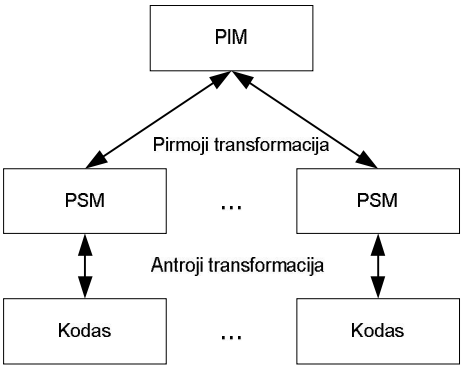
OMG (angl., *Object Management Group, www.omg.org*) grupė pasiūlė MDA (angl., *Model-driven architecture*) principą modeliais grindžiamam sistemų kūrimui (angl., *Model-driven Development - MDD*). MDA principo esmė yra kompiuterizuojamos sistemos funkcionalumo specifikacijos atskyrimas nuo specifikacijos, aprašančios to funkcionalumo realizavimo aspektus konkrečioje technologinėje platformoje.

MDA procesas yra padalintas į tokius žingsnius (2.4 pav.):

1. Sukurti aukšto abstrakcijos lygio modelį, kuris yra nepriklausomas nuo bet kokios realizacinės platformos. Šis modelis yra vadinamas nuo platformos nepriklausomu modeliu (angl., *Platform Independent Model* – PIM);

2. Transformuoti PIM į vieną ar daugiau modelių, kuriuose yra įvertintas konkrečios realizacinės platformos aspektas. Šie modeliai yra vadinami nuo platformos priklausomi modeliai (angl., *Platform Specific Model* – PSM);

3. Transformuoti PSM modelius į kodą.



2.4 pav. MDA transformacijos „PIM → PSM → Kodas"

Trumpai apžvelkime PIM ir PSM modelius. PIM modelis, kartais dar vadinamas domeno modeliu (angl., *Domain Model*), išreiškia išimtinai kompiuterizuojamos veiklos (angl., *business*) funkcionalumą ir elgseną kuriamos sistemos požiūriu. Šis modelis kuriamas bendradarbiaujant probleminės srities ekspertams ir IS kūrėjams. PIM modelis neturi būti įtakojamas jokios technologinės platformos, o modeliavimui pasirinktoji notacija turi būti suprantama abiem bendradarbiaujančioms pusėm – tokiu būdu probleminės srities ekspertai galės užtikrinti, kad tas funkcionalumas, kuris yra atvaizduotas PIM modelyje yra pilnas ir teisingas. Kitas privalumas yra tas, kad nuo platformos nepriklausantis modelis išliks nekintantis ir išlaikys savo vertę ilgą laiką, net ir vykstant technologijų kaitai (modelis pakis tik tuo atveju, jei pakis pati probleminė sritis, kurią išreiškia PIM modelis).

PSM modelis yra kuriamas transformuojant PIM modelį (iš vieno PIM modelio galima sukurti vieną ir daugiau PSM modelių, pritaikytų skirtingoms realizacinėms platformoms). Vykdant „PIM → PSM" transformaciją svarbų vaidmenį atlieka vadinamieji profiliai (arba bibliotekos, angl., *Profile*), kuriuose saugomi standartizuoti UML išplėtimų (stereotipų, iš anksto nustatytų reikšmių, apribojimų) rinkiniai, skirti konkrečios platformos specifikai įvertinti. Profilis yra UML mechanizmas, skirtas metamodeliams apibrėžti ir išplėsti.

OMG pateikia keturis būdus, kaip galima atlikti transformacijas tarp PIM ir PSM modelių:

1. Transformacija atliekama rankiniu būdu, transformacijos taisykles sukuriant kiekvienam konkrečiam atvejui. Atlikus transformaciją, taisyklės, kaip tai buvo padaryta, nėra išsaugomos.

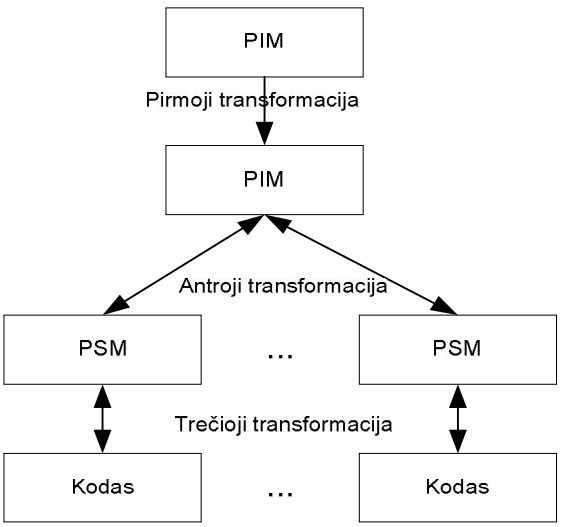
2. Transformacija atliekama rankiniu būdu, tačiau procesas vykdomas naudojant sukurtus šablonus, kurie pritaikyti PIM transformavimui į konkretų PSM.

3. Sukurtieji šablonai yra taikomi naudojant MDA įrankyje integruotą algoritmą, kuris sukuria PSM modelio skeletą. Šis pirminis modelio variantas vėliau turi būti papildytas rankiniu būdu.

4. Gerai išvystytas MDA įrankis atlieka pilną transformaciją iš PIM į PSM modelį.

CIM nuo skaičiavimų nepriklausomas modelis (angl., *Computation Independent Model* – CIM) arba – veiklos modelis (angl., *Business Model*). Dar prieš keletą metų OMG laikėsi nuomonės, kad CIM modelis nėra būtinas ir yra naudojamas tik kaip gairė kuriant PIM modelį. Tam buvo keletas priežasčių (OMG-Rules, 2003):

- OMG vizijoje MDA principas yra realizuojamas naudojant eilę OMG pasiūlytų standartų, vienas iš kurių yra UML. Tačiau pati OMG pripažįsta, kad „grynasis“ UML nėra visiškai tinkama kalba CIM modeliavimui. Dėl šios priežasties OMG negalėjo pasiūlyti jokio formalaus metodo „CIM → PIM“ transformacijai atlikti – tebuvo numatyta galimybė iš CIM gauti PIM. Šios transformacijos ilgą laiką buvo tyrinėjamos tik pavienių mokslininkų [10], mokslinių darbų šioje srityje yra labai nedaug.
- OMG tikėjosi sulaukti veiklos modeliavimui skirtų, su UML ir kitais OMG standartais suderinamų standartų, kurių pagrindu sukurto CIM modelio elementus būtų galima transformuoti į PIM modelio elementus.



2.5 pav. MDA transformacijos „CIM → PIM → PSM → Kodas”

Šiuo metu CIM modelio pozicijos MDA erdvėje labai sustiprėjo (2.5 pav.), kadangi OMG šiuo metu jau turi pasiūliusi eilę standartų (BPMN, OSM, BMM, SBVR), kurie gali būti tiesiogiai priskirti CIM lygmeniui ir tokiu būdu integruoti su kitais OMG standartais (visų pirma, su UML); dar keletas su veiklos modeliavimu susijusių iniciatyvų yra pasiūlymų teikimo arba vystymo stadijose [25].

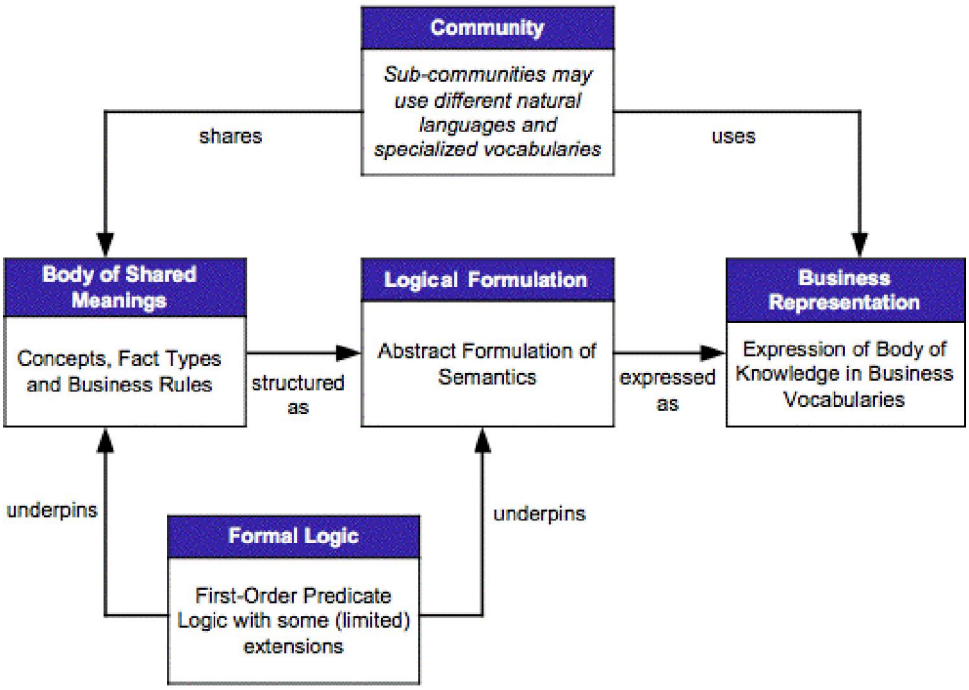
Atlikus MDA analizę galima įžvelgti tokius pagrindinius šio principo taikymo IS gyvavimo cikle privalumus:

- MDA papildžius naujaisiais OMG standartais (BMM, SBVR ir kt.), skirtais veiklai modeliuoti, atsiranda galimybė glaudžiai ir vienareikšmiškai susieti realaus pasaulio fragmentus (probleminę veiklos sritį) su informacinėmis sistemomis, tokiu būdu užtikrinant kuriamų sistemų lankstumą (adaptyvumą), skaidrumą ir kitas kokybines charakteristikas.
- Galimybė atlikti automatizuotas transformacijas tarp modelių [10] bei kodo generavimą sukurtųjų modelių pagrindu sąlygoja trumpesnį sistemos kūrimo laiką bei kaštus;
- CIM ir PIM modelių naudojimas skatina modelių daugkartinį panaudojimą (angl., *model reusability*);
- Visi projektiniai modeliai yra kuriami naudojant vieningą modeliavimo kalbą (UML) arba kalbas, sukurtas bendro metamodelio (MOF) pagrindu, o tai dalinai išsprendžia modelių suderinamumo problemą;
- Lengviau priderinti sukurtąją IS prie besikeičiančių technologinių bei veiklos reikalavimų;
- Taikant MDA principą kartu su XMI sistemos migracija į kitas technologines platformas gali būti atliekama greičiau ir paprasčiau.

Toliau trumpai apžvelgsime kitus šiam darbui svarbius OMG standartus.

2.5.2. Veiklos žodyno ir taisyklių standartas (SBVR)

SBVR specifikacija [20] apibrėžia veiklos žodyno (*business vocabulary*), veiklos faktų (*business facts*) ir veiklos taisyklių (*business rules*) semantiką bei XMI schemą veiklos žodynų ir veiklos taisyklių apsikeitimui tarp organizacijų ar kompiuterizuotų sistemų. MDA architektūroje SBVR yra Veiklos modelio (CIM) sudėtinė dalis – šioje pozicijoje SBVR gali būti naudojamas kaip viso veiklos modelio žodynas. Labai svarbus SBVR akcentas yra tai, kad tiek veiklos terminų žodynai, tiek ir pačios veiklos taisyklės yra sudaromos naudojant natūralios kalbos konstrukcijas – tai yra ypač svarbu veiklos modeliavimo lygmenyje.



2.6 pav. Penki pagrindiniai SBVR aspektai [20]

SBVR galima išskirti 5 aspektus (2.6 pav.): Bendruomenė (*Community*), Bendrų prasmių branduolys (*Body of Shared Meanings*), Loginė formuluotė (*Logical Formulation*), Veiklos atvaizdavimas (*Business Representation*), Formali logika (*Formal Logic*):

- Veiklos žodyno (*Business vocabulary*) pagrindas yra *bendruomenė*. Veiklos lygmenyje pirmos svarbos bendruomenės yra pačios organizacijos, vykdančios tam tikrus veiklos procesus (prižiūrimus ir valdomus veiklos taisyklių). Tačiau ir tokios bendruomenės, kaip industrijos šaka, kurioje veikia organizacija, arba organizacijos partneriai, standartų grupės ir kitos panašios organizacijos taip pat gali būti įvertintos. Kitas svarbus bendruomenių momentas yra sub-bendruomenės pačioje organizacijoje (bendruomenėje) , kurios gali turėti savus bendrų prasmių branduolius, išreikštus kituose žodynuose (tai gali būti įvairūs žargonai ar netgi kitos kalbos). SBVR standarte šios sub-bendruomenės yra vadinamos kalbinėmis bendruomenėmis (*speech communities*). SBVR standartas turi apibrėžęs ir dar vieną bendruomenės tipą – tai semantinė bendruomenė (*semantic community*). Semantinės bendruomenės yra bendruomenės, kurios veiklos konceptus supranta vienodai. Tokiu būdu vieną semantinę bendruomenę gali sudaryti kelios kalbinės bendruomenės;
- Kiekviena bendruomenė turi savą *bendrų prasmių branduolį*, kurį sudaro konceptai (įskaitant ir faktų tipus) bei veiklos taisyklės. SBVR specifikacijoje *prasmė* yra apibūdinama kaip „tai, kas norima pasakyti žodžiu, ženklu, teiginiu ar aprašu; tai, ką kažkas nori išreikšti“. Bendrų prasmių branduolyje yra nustatomos bendros dalykų prasmės, bet ne tų dalykų išraiškos formos.

Natūralu, kad norint šiomis bendromis prasmėmis keistis, apie jas diskutuoti ir jas validuoti, šios prasmės turi būti kažkokiu būdu išreikštos; tačiau SBVR ypatumas yra tame, kad šis standartas atskiria veiklos dalyko prasmę nuo bet kokios konkrečios to dalyko išreiškimo formos. Bendrų prasmių branduolio struktūra (t.y. kuris konceptas kokią rolę vaidina viename ar kitame fakte, kuris faktas naudojamas kurioje taisyklėje ir pan.) yra apibrėžiama, asocijuojant *abstrakčius* konceptus, faktų tipus (fakto tipas yra asociacija tarp dviejų ar daugiau konceptų) ir VT, o ne asocijuojant išraiškas konkrečioje natūralioje kalboje;

- *Loginė formuluotė* suteikia formalią, abstrakčią, nuo kalbos nepriklausomą sintaksę bendrų prasmių branduolio semantikai užrašyti. Ji palaiko įvairias pateikimo formas, pavyzdžiui, daiktavardžių ir veiksmažodžių išreiškimo formas, galimybę skaityti asociacijas abiem kryptim. Loginė formuluotė palaiko dvi svarbias SBVR savybes: pirma, bendrų prasmių branduolio susiejimą (atvaizdavimą) su bendruomenių naudojamais žodynais, ir antra, atvaizdavimą į XMI, kas leidžia keistis konceptais, faktais ir veiklos taisyklėmis tarp įvairių įrankių, palaikančių SBVR;
- *Atvaizdavimas*. Konceptai ir veiklos taisyklės, sudarantys bendrų prasmių branduolį, turi būti atvaizduoti žodynuose kalbinėms bendruomenėms priimtina ir suprantama forma. Šios pateikimo formos gali būti įvairios natūralios kalbos, modeliavimo kalbos, tokios kaip UML, arba specializuoti natūralios kalbos poaibiai, naudojami, pavyzdžiui, inžinierių ar teisininkų. SBVR palaiko veiklos prasmės atvaizdavimą į konkrečią kalbą, asocijuodama bendrų prasmių branduolio elementus su tam tikrais žymekliais (*signifiers*), pavyzdžiui, konceptus suriša su tokiais terminais kaip „vartotojas“, „mašina“ ir pan., o faktų tipus – su tokiais simboliais, veiksmažodžiais ar frazėmis kaip „nuomojasi“, „yra saugomas“ ir pan. Tokiu būdu loginės formuluotės užtikrina struktūrą, o žymekliai yra įterpiami į logines formuluotes išreiškimo (tam tikroje kalboje) tikslais. SBVR naudoja skirtingas spalvas skirtingiems elementams atvaizduoti: oranžinė – SBVR sintaksė, žalia – žodynas (terminai, individualūs konceptai (vardai, tikriniai daiktavardžiai), skaičiai), mėlyna – faktų tipai (asociacijos tarp konceptų);
- SBVR turi tvirtą teorinį *formalios logikos* pagrindą, pagrindžiantį tiek SBVR logines formuluotes, tiek ir bendrų prasmių branduolio struktūrą. Pagrindas yra pirmos eilės predikatų logika (su tam tikrais apribotais aukštesnės eilės logikos išplėtimais), o taip pat modalinės logikos (*modal logic*) konceptai, tokie kaip būtinybė, galimybė, įsipareigojimas ir leidimas.

SBVR specifikacijoje galima išskirti dvi vieną su kitu susijusius žodynus (ir juos palaikančius metamodelius):

- SBVR žodynas, skirtas aprašyti veiklos žodynams (*Vocabulary for Describing Business Vocabularies*), kurio branduolį sudaro veiklos žodyno metamodelis. Kiekvienas veiklos žodynas,

sudarytas naudojant SBVR, turėtų saugoti savyje visus specializuotus terminus ir apibrėžimus konceptų, kuriuos konkreti organizacija ar bendruomenė naudoja savo veikloje. Šis žodynas yra nepriklausomas nuo naudojamų technologijų, kurios naudojamos organizacijoje veiklos procesams palaikyti. Veiklos konceptai yra apibrėžiami nepriklausomai nuo terminų ar kalbos, naudojamos šiems konceptams išreikšti – tai leidžia viena kalba išreikštus veiklos konceptus išversti į kitą kalbą (t.y. SBVR palaiko dialektus). Specifikacija taip pat apibrėžia konstruktus, kurie leidžia atlikti tekstinių dokumentų lingvistinę analizę, o tai savo ruožtu sudaro prielaidas išgauti veiklos žodyną ir veiklos taisykles iš tekstinių dokumentų automatizuotomis priemonėmis.

- SBVR žodynas, skirtas veiklos taisyklėms aprašyti (*Vocabulary for Describing Business Rules*), kurio branduolį sudaro veiklos taisyklių metamodelis. SBVR specifikacijoje veiklos taisyklė yra apibrėžiama kaip „taisyklė, kuri yra veiklos jurisdikcijoje“ („*a rule that is under business jurisdiction*“). Veiklos taisyklės yra sudaromos naudojant veiklos žodyną ir specialią SBVR sintaksę. SBVR atskiria taisyklės prasmę (*Meaning*) nuo potencialiai galimų būdų išreikšti šią taisyklę natūralia kalba. Tokiu būdu tampa įmanoma aptikti skirtingais žodžiais (netgi skirtingomis kalbomis) užrašytas veiklos taisykles, kurios turi vieną ir tą pačią prasmę (t.y. yra sinonimai). Ši daugiakalbystės savybė teoriškai įgalina veiklos taisykles automatiškai versti iš vienos kalbos į kitą, iš vieno dialekto į kitą.

Žodyno, skirto veiklos taisyklėms aprašyti, metamodelis susideda iš dviejų dalių: pirmoji dalis skirta veiklos taisyklių konceptams, o antroji – formuluotėms, kurios skirtos toms veiklos taisyklėms išreikšti.

Naudojant SBVR, veiklos taisyklės yra užrašomos formaliai ir vėliau interpretuojamos naudojant formalią logiką – tai reiškia, kad atitinkami įrankiai tokias VT gali patikrinti įvairių formalių kriterijų atžvilgiu (VT pilnumas, suderinamumas ir pan.). Formalizuotos natūralios kalbos konstrukcijos taip pat leidžia sumažinti natūralios kalbos dviprasmiškumo lygį. Be kita ko, SBVR palaiko ir multidimencines, hierarchines kategorijas, kas leidžia užrašinėti organizacijos konceptus nuo bendriausių iki labiausiai specifinių (taksonomijos, kategorijų schemos). SBVR taip pat palaiko tezaurų mechanizmus (sinonimai, trumpiniai, „see also“).

Reikia pastebėti, kad naudojant SBVR galima sudarinėti tiktai deklaratyvias veiklos taisykles. Šiuo metu OMG dar nėra pasiūliusi taisyklių, kaip deklaratyvias SBVR taisykles atvaizduoti į tokius gerai žinomus ir pripažintus taisyklių formatus kaip „If Then“, ECA (event-condition-action), sprendimų lenteles ar medžius ir pan. Dėl šių priežasčių kai kurie mokslininkai ir praktikai į SBVR perspektyvą žiūri gana rezervuotai. Dar vienas niuansas yra tas, kad ne visos SBVR veiklos taisyklės, net ir vykdomosios, gali būti automatizuotos (kompiuterizuotos)

informacijos sistemos lygmenyje. Tai nėra labai svarbu SBVR specifikacijos, kuri skirta veiklos lygmens taisyklėms užrašyti, lygmenyje, tačiau tai svarbu žiūrint iš MDA pozicijos, kadangi veiklos modelis (kartu su veiklos taisyklėmis) turi būti transformuojamas į PIM modelį [14]. Pereinant į PIM lygmenį tokios neautomatizuojamos veiklos taisyklės turėtų būti fiksuojamos pareigybinėse instrukcijose ar taisyklių rinkiniuose, kurie reglamentuoja vartotojo veiksmus. Šios vietos OMG kol kas dar neturi išsprendusi.

SBVR sąsają su kitais OMG standartais taip pat yra gana dviprasmiška. Iš vienos pusės SBVR turi MOF modelį (OMG *Meta Object Facility*), ir tai reiškia, kad SBVR modeliai gali būti saugoti MOF saugyklose, jais galima keistis, jie gali būti susieti su kitais MOF grindžiamais modeliais (tame tarpe ir UML); per MOF standartas yra pilnai integruotas į MDA architektūrą. Iš kitos pusės, šiuo metu dar nėra nei vieno metodo ar aiškių taisyklių, kaip konkrečiai SBVR turėtų sietis su kitais OMG standartais.

SBVR specifikacijoje [20] yra skyrius, kuriame bendrais bruožais pristatomas metodas, kaip SBVR veiklos žodyną grafiškai atvaizduoti per UML klasių diagramą. Reikia pastebėti, kad šis metodas įvertina mažiau nei 20 iš daugiau nei 100 SBVR veiklos žodyno metamodelio konceptų. Specifikacijoje nėra keliamas klausimas, ar toks ženklus konstruktų (kalbos ekspresyvumo) skaičiaus sumažinimas nesukelia kokių nors sunkumų, ar tai neįtakoja specifikacijos pilnumo ir kitų savybių.

2.5.3. Produkcinių taisyklių atvaizdavimas (PRR)

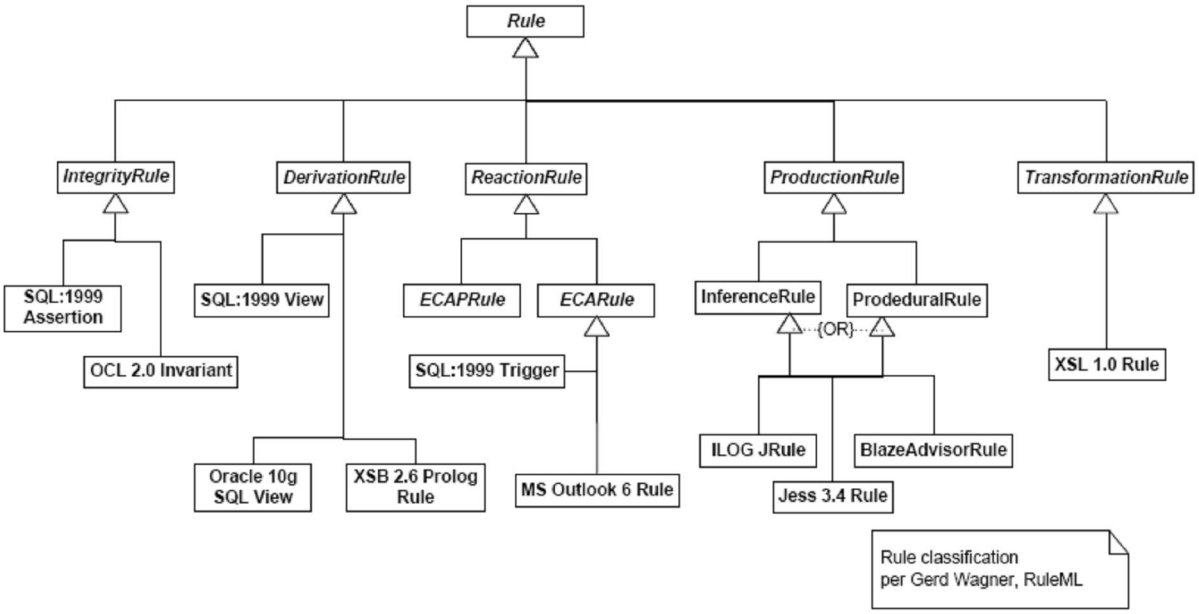
PRR iniciatyva šiuo metu dar tiktai praėjo pasiūlymų pateikimo (*Request for Proposals*) stadiją, todėl jokios oficialios, viešai prieinamos PRR specifikacijos šiuo metu dar nėra.

OMG jau turi standartą, skirtą formaliam veiklos taisyklių užrašymui veiklos modeliavimo (CIM) lygmenyje – tai aukščiau apžvelgtasis SBVR. Tuo tarpu PRR iniciatyvos tikslas yra pasiūlyti standartą, kuris leistų specifikuoti veiklos taisykles (o tiksliau, taisyklių poaibį – produkcines taisykles (*production rules*), išreiškiančias sistemos elgseną) sistemos modeliavimo lygmenyje (PIM) [9]. Reikia pabrėžti, kad PRR specifikacijose ypač akcentuojama būtinybė pagerinti produkcinių taisyklių modeliavimą UML kalboje, ir MDA apskritai – kadangi būtent tai yra įvardinama kaip viena iš prioritetinių PRR taikymo sričių. PRR standartu bus siekiama standartizuoti šio tipo veiklos taisyklių modeliavimą šiame lygmenyje ir užtikrinti šių taisyklių apsikeitimo galimybę (per standartizuotą apsikeitimo formatą (*interchange format*) – XMI) sistemos modeliavimo lygmenyje tarp įvairių veiklos taisyklių modeliavimą palaikančių įrankių.

Dar vienas siektinas PRR standarto tikslas yra užtikrinti produkcinių taisyklių atsekamumo (*traceability*) UML modeliuose (ir kituose MDA standartuose) palaikymą.

Prognozuojama, kad PRR standarto taikymas praktikoje paskatins greitesnį produkcinių taisyklių (ar apskritai, veiklos taisyklių) požiūrio taikymą kompiuterizuotose sistemos (tame tarpe ir IS) – to bus siekiama pagrinde per PRR ir UML metamodelių susiejimą; produkcinės taisyklės taptų UML išplėtimu, kurį sistemų kūrėjai galėtų panaudoti savo kuriamuose OO modeliuose. Taip pat tikimasi, kad tai padidintų pasitikėjimą produkcinių taisyklių (ar apskritai, veiklos taisyklių) interpretavimo/vykdymo mechanizmais, tokiais kaip taisyklių varikliai (*rule engines*), ir paskatintų dažnesnį jų taikymą. PRR šiuo metu vis dar yra kūrimo stadijoje.

Produkcinės taisyklės yra programinės logikos (*programming logic*) fragmentas [24], specifikuojantis vieno ar daugiau veiksmų vykdymą prie tam tikrų apibrėžtų patenkintų sąlygų. Produkcinės taisyklės turi operacinę sintaksę (formalizuojančią būsenų pasikeitimą). Produkcinių taisyklių įtaka sistemai gali priklausyti nuo jų vykdymo eilės tvarkos. Visos produkcinės taisyklės turi bendrą struktūrą: „Jei (sąlyga) tai (veiksmų sąrašas)“ (*if (condition) then (action-list)*). Kai kuriuose taikymuose produkcinės taisyklės struktūra gali būti išplėsta konstruktu „kitu atveju“ („*else*“): „Jei (sąlyga) tai (veiksmų sąrašas) kitu atveju (alternatyvus veiksmų sąrašas)“ (*if (condition) then (action-list) else (alternative-action-list)*); tačiau reikia pastebėti, kad pastaroji produkcinės taisyklės konstrukcija PRR specifikacijoje nebus taikoma.



2.7 pav. RuleML specifikacijoje apibrėžta taisyklių klasifikacija

Produkcines taisykles ir jų potipius galima rasti RuleML specifikacijoje pateiktoje taisyklių klasifikacijoje (2.7 pav.); tiesa, PRR specifikacijos kūrėjai pastebi, kad PRR Produkcinė taisyklė yra

ne Taisyklės (*Rule*), o Vykdomosios taisyklės (*Computer Executable Rule*) poklasis – taip padaryta, siekiant išvengti nesusipratimų metamodelio klasę Taisyklė (*Rule*) naudojant kitoms reikmėms.

PRR semantiką apibrėžia PRR metamodelis, kurį galima padalinti į dvi sritis (fragmentus):

- PRR branduolys sudaro PRR metamodelio pagrindą. PRR Core modeliu apibrėžtomis produkcinėmis taisyklėmis bus galima keistis su kitais PRR Core modeliais. Tiesa, tam reikės susikurti atitinkamas transformacijos taisykles.

Kitas PRR metamodelio fragmentas yra PRR OCL. PRR OCL yra išplėsta UML OCL versija, skirta sąlygoms (*conditions*) ir veiksmams (*actions*) apibrėžti. PRR OCL apibrėžia nenormatyvinę, abstrakčią sintaksę (kurios pagrindą sudaro OCL kalba), skirtą specifikuoti PRR išraiškoms.

Kaip jau minėta, PRR gali būti glaudžiai integruojama su *OCL* kalba – tai atliekama per *PRR OCL* metamodelį, kuris yra išplėsta UML OCL versija, skirta sąlygoms (*conditions*) ir veiksmams (*actions*) apibrėžti. PRR OCL apibrėžia nenormatyvinę, abstrakčią sintaksę (kurios pagrindą sudaro OCL kalba), skirtą specifikuoti PRR išraiškoms.

PRR OCL metamodelyje naudojamos *EssentialOCL* paketo, kuris yra pilno OCL metamodelio poaibis, klasės.

PRR iniciatyvoje dalyvauja didelis skaičius su veiklos taisyklėmis dirbančių kompanijų (pvz., Fair Isaac, ILOG, LibRT, Pega, Corticon, TIBCO), taip pat akademinės visuomenės atstovai (RuleML.org) bei kitos didelės IT kompanijos (Fujitsu, IBM). Tokio didelio konsorciumo subūrimas leidžia daryti prielaidas, kad PRR standartas susilauks ne tik akademinės bendruomenės, bet ir didžiųjų IT kompanijų dėmesio ir palaikymo.

PRR metamodelis bus sukurtas taip, kad jį būtų nesudėtinga išplėsti.

Apibendrinant galima pasakyti, kad PRR standartu OMG sieks išspręsti visą eilę klausimų, susijusių su veiklos taisyklių taikymu informacijos sistemose (ir kitose programų sistemose) ir kituose OMG standartuose:

- PRR užtikrina standartizuotą produkcinių taisyklių atvaizdavimo būdą, kuris yra suderinamas su taisyklių procesorių taikoma produkcinių taisyklių koncepcija. Dėl šios priežasties PRR gali būti taikomas veiklos taisyklių apsikeitime tarp įvairių taisykles modeliuojančių įrankių (ir kitų įrankių, kurie vienu ar kitu aspektu palaiko taisyklių modeliavimą);
- PRR gali būti taikomas MDA PIM modelių kūrime; yra visos prielaidos manyti, kad PRR bus palaikomas ir PSM lygmenyje, tačiau tai didžiąja dalimi priklausys nuo taisyklių interpretavimo sistemų (ir kitų, pavyzdžiui, CASE įrankių) gamintojų;

- PRR parodo, kaip produkcinių taisyklių specifikavimas gali išplėsti OMG UML kalbos panaudojamumą;
- PRR yra produkcinių taisyklių atvaizdavimo standartas, kuris gali būti panaudotas kaip pagrindas kitose taisyklių standartizavimo iniciatyvose, tokiose kaip W3C RIF (*Rule Interchange Format*) ar produkcinėms taisyklėms skirtoje RuleML versijoje.

2.5.4. OMG veiklos taisyklių standartų analizės apibendrinimas

Išanalizavus visus OMG standartus, kurie vienu ar kitu aspektu yra susiję su veiklos taisyklių principu, galima teigti, kad didžiausią įtaką turi SBVR, kadangi tiek veiklos terminų žodynai, tiek ir pačios veiklos taisyklės yra sudaromos naudojant natūralios kalbos konstrukcijas – tai yra ypač svarbu veiklos modeliavimo lygmenyje.

MDA architektūra integruoja tarpusavyje visus kitus OMG standartus (tai pasiekiama per bendrą OMG grupės meta-metamodelį – MOF). Dauguma šiuolaikinių CASE įrankių, tame tarpe ir *UML MagicDraw*, palaiko vieną ar kitą MDA aspektą, kas leidžia daryti prielaidas, kad tokie įrankiai per MDA architektūra palaikys ir kuriamą metodą, kuris bus grindžiamas būtent OMG standartais. Reikia pastebėti, kad ne visi MDA principai (pvz., CIM modeliavimas, CIM-PIM transformacijos ar kokybiško, pilno kodo generavimas [1]) šiuo metu yra sėkmingai realizuoti ar pakankamai gerai išplėtoti.

SBVR yra pirmasis OMG grupės standartas, skirtas išimtinai veiklos žodynams ir veiklos taisyklėms specifikuoti. Standartas yra taikytinas CIM (veiklos modeliavimo) lygmenyje. Nors standartas yra jau keletas metų kaip patvirtintas, tačiau įrankių, kurie jį taikytų yra labai nedaug – galima teigti, kad standartas kol kas dar nesusilaukė didelio pripažinimo. Viena iš realiausių priežasčių, kodėl taip yra – standartui OMG grupė kol kas neturi užtikrinusi tolimesnio palaikymo MDA gyvavimo cikle, t.y. kol kas dar nėra patvirtintų standartų (kalbų), į kuriuos būtų galima transformuoti SBVR veiklos taisykles, tokiu būdu užtikrinant veiklos taisyklių palaikymą viso taisyklių ir kompiuterizuotų sistemų gyvavimo ciklo metu.

PRR yra OMG grupės iniciatyva, kuri greitu laiku turėtų duoti rezultatą – standartą, skirtą vieno iš veiklos taisyklių tipų, t.y. produkcinių taisyklių, modeliavimui. Su PRR standarto atsiradimu SBVR taip pat įgautų naują pagreitį, kadangi PRR palaikytų taisyklių modeliavimą jau ne veiklos modeliavimo (kaip kad SBVR), o nuo platformos nepriklausomame sistemos projektavimo etape. Tai būtų didelis ir svarbus žingsnis plėtojant MDA koncepciją veiklos taisyklų aspektu.

PRR specifikacijose ypač akcentuojama būtinybė pagerinti (produkcinių) taisyklių modeliavimą UML kalboje, ir MDA apskritai – kadangi būtent tai yra įvardinama kaip viena iš prioritetinių PRR taikymo sričių. PRR standartu bus siekiama standartizuoti šio tipo veiklos taisyklių modeliavimą PIM lygmenyje ir užtikrinti šių taisyklių apsikeitimo galimybę (per standartizuotą apsikeitimo formatą – XMI) sistemos modeliavimo metu tarp įvairių veiklos taisyklių modeliavimą palaikančių įrankių. Dar vienas siektinas PRR standarto tikslas yra užtikrinti produkcinių taisyklių atsekamumo (*traceability*) UML modeliuose (ir kituose MDA standartuose) palaikymą. Prognozuojama, kad PRR standarto taikymas praktikoje paskatins greitesnį produkcinių taisyklių (ar apskritai, veiklos taisyklių) požiūrio taikymą kompiuterizuotose sistemos (tame tarpe ir IS) – to bus siekiama pagrinde per PRR ir UML metamodelių susiejimą; produkcinės taisyklės taptų UML išplėtimu, kurį sistemų kūrėjai galėtų panaudoti savo kuriamuose OO modeliuose. Taip pat tikimasi, kad tai padidintų pasitikėjimą vykdomųjų taisyklių interpretavimo/vykdymo mechanizmais, tokiais kaip taisyklių varikliai (*rule engines*), ir paskatintų dažnesnį jų taikymą.

2.6. Veiklos žodynų transformavimo kalbų analizė

Informacinėse technologijose transformavimo kalbos yra ypač svarbios.

Labiausiai paplitusios modeliu transformacijos kalbos:

- ATL - populiari metamodeliais paremta transformacijų kalba,
- QVT - OMG standartas modelių transformavimui MOF/QVT trumpai: QVT,
- Beanbag - operacijomis paremta kalba užtikrinanti inkrementinio duomenų augimo vientisumą,
- MOLA - grafinė aukšto lygmens transformavimo kalba įgyvendinta Lx pagrindu,
- Tefkat - transformavimo kalba ir modelių transformavimo procesorius.

Naudojant modeliavimo paketą MagicDraw UML, transformacija suteikia modelio ir duomenų migravimo tarp skirtingų UML profilių ir tarp skirtingo profilio versijų transformacijos galimybę. Transformacija yra aprašoma deklaratyviai, grafiškai, modelio pagalba. Transformacija leidžia pakeisti elemento tipą, požymius, požymių tipus. Transformacija yra vienakryptė, skirta vieno pirminio modelio transformavimui į vieną galutini.

Tam, jog pasirikti tinkamiausia modelių transformavimo kalbą ir įrankį reikia įvertinti argumentus. Gerai apibendrinus pasirinkimo kriterijus, renkantis reikia atsakyti į klausimus (Kas bus transformuojama? Koks transformacijos tikslas? Ar pakartotinė transformacija bus reikalinga? ir t.t.) ir įvertinti kriterijus.

Be jokių abejonių tinkamiausias sprendimas yra modeliu paremtos transformacijos kalbos leidžiančios atlikti transformacijas metamodelių pagalba. Transformacijos turi būti gerai palaikančios horizontalias transformacijas ir pakankamai populiarios, jog turėtų išvystytus transformavimo įrankius.

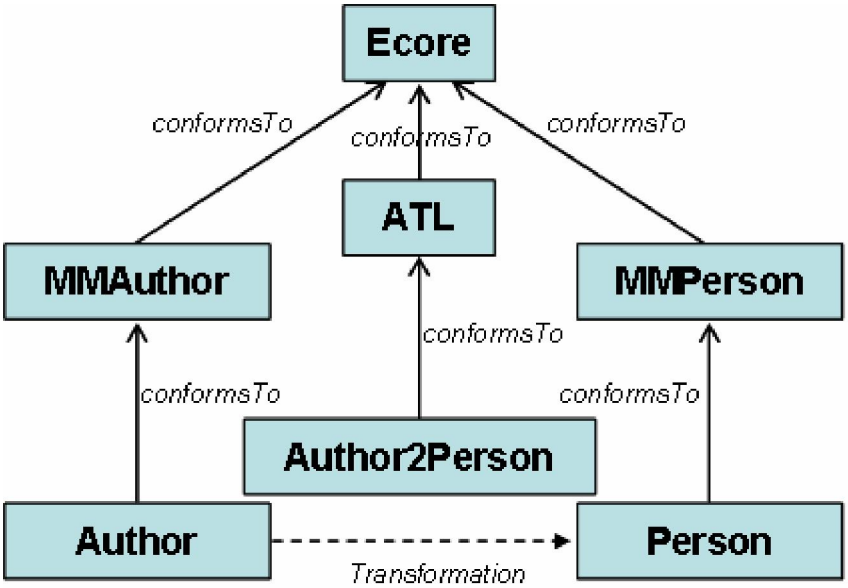
Tokios transformavimo kalbos ir technologijos yra ATL [3] ir QVT [17]. Taip pat verta paminėti XSLT [26] dėl glaudžios sąsajos tarp XML ir modelių, kuri yra dažniausia jų saugojimo kalba XSLT yra ypač paplitusi modelių transformavime. Dėl šio derinio modeliai išeksportuoti į XML dažnai transformuojami XSLT pagalba, o vėliau vėl paverčiami modeliais.

2.6.1. ATL

ATL (angl. ATLAS Transformation Language) yra modelių transformavimo kalba ir įrankių rinkinys. ATL naudojimas modeliais paremtoje inžinerijoje (angl. *Model Driven Engineering* – MDE) tikslas pagaminti daugybę galutinių modelių remiantis pirminiu modeliu.
Pagal MDE principą transformacija taip pat aprašoma modeliu. ATL įgyvendinta populiarioje Eclipse platformoje, tai suteikia papildomas galimybes, sintaksės validavimą ir kitas galimybes bei transformacijų bibliotekas. Šiuo metu yra keltas metamodeliavimo technologijų. ATL palaiko MOF (angl. *Meta Object Facilities*) apibrėžiama OMG ir Ecore metamodelį. Tai reiškia, jog ATL gali transformuoti metamodelius specifikuotus MOF ir Ecore (2.8 pav.).

ATL pavyzdys:

```
rule Author {
        from
            a : Author!Author
        to
            p : Person!Person (
                name <- a.name,
                surname <- a.surname)
}
```



2.8 pav. ATL transformacijos schema

Ši schema apžvelgia ATL transformaciją (Author2Person) kuri leidžia generuoti Person modelį atitinkanti MMPerson metamodelį iš Author modelio kuris atitinka MMAuthor metamodelį. ATL aprašyta transformacija atitinka ATL metamodelį. Transformacija atliekama tarp skirtingų metamodelių modelių.

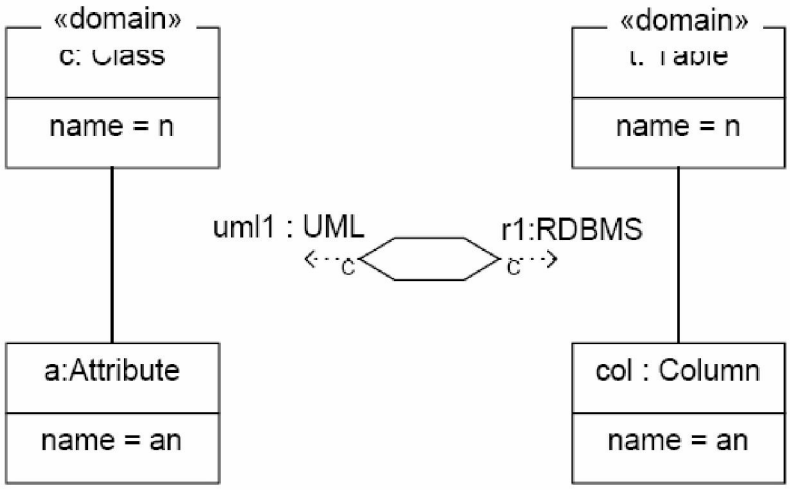
2.6.2. QVT

QVT (angl. *Query/Views/Transformation*) palaiko mišrią transformacijos užrašymo specifikacija: deklaratyvia ir imperatyve. Pagal QVT yra OMG standartas – kalba transformacijų aprašymui MDA kontekste. Tai yra vienas iš svarbiausių modelių transformacijų standartų.

Užklausa – tai išraiška modelio išrinkimui. OCL yra viena iš žymiausiu kalbų skirta užklausoms. Modelis pilnai kilęs iš kito modelio yra vadinamas Vaizdu. Dažniausiai vaizdas nėra saugomas ir negali būti keičiamas nepriklausomai nuo pradinio modelio. Programa kuri iš pradinio modelio sukuria galutini vadinama Transformacija.

QVT pavyzdys:

```
Relation UML2Rel {
      checkonly domain uml1
                c:Class {name = n, attribute = a:Attribute{name = an}}
      checkonly domain r1
                t:Table {name = n, column = col:Column{name = an}}
}
```



2.9 pav. QVT UML klasių transformacija į duomenų bazių modelį

2.9 pav. atvaizduojama klasių diagramos transformacijos į reliacinę duomenų schemą, transformacijos tarp klasės ir lentelės aprašas. QVT palyginus su ATL yra ganėtinai jauna kalba.

QVT principai:

- pirminis ir galutinis modeliai gali atitikti MOF;
- transformacijos aprašas yra modelis, taigi taip pat atitinka MOF metamodelį, t.y. abstrakti QVT sintakse atitinka MOF versijos 2.0 metamodelį.

  Detaliau QVT integruoja OCL versijos 2.0 standartą ir išplečia jį iki imperatyvaus. QVT nusako net tris DSL kalbas (angl. *Domain Specific Languages*):
- Ryšių (angl. *Relations*),
- Centrinę (angl. *Core*) ir
- Operacine (angl. *Operational*) sąsajų.

Šios kalbos yra suskirstytos į lygmenis. Ryšių ir Centrinė yra deklaratyvios kalbos skirtinguose abstrakcijos lygmenyse su tarpusavio sąsajomis. Ryšių kalba turi tekstinę ir grafinę sintaksę. Operacinė sąsajų kalba yra imperatyvinė kalba praplečianti Ryšių ir Centrinę kalbas. Operacinės kalbos sintaksės konstrukcijos randamos imperatyviose kalbose (ciklai, sąlygos, ir kt.).

Taip pat svarbi specifikacijos dalis yra mechanizmas vadinamas QVT/Juoda dėžė (angl. *QVT/BlackBox*), skirtas iškviesti transformacijas aprašytas kitomis kalbomis (XSLT, XQuery). Labai svarbi specifikacijos leidžianti integruoti esamas ne QVT paremtas transformacijas.

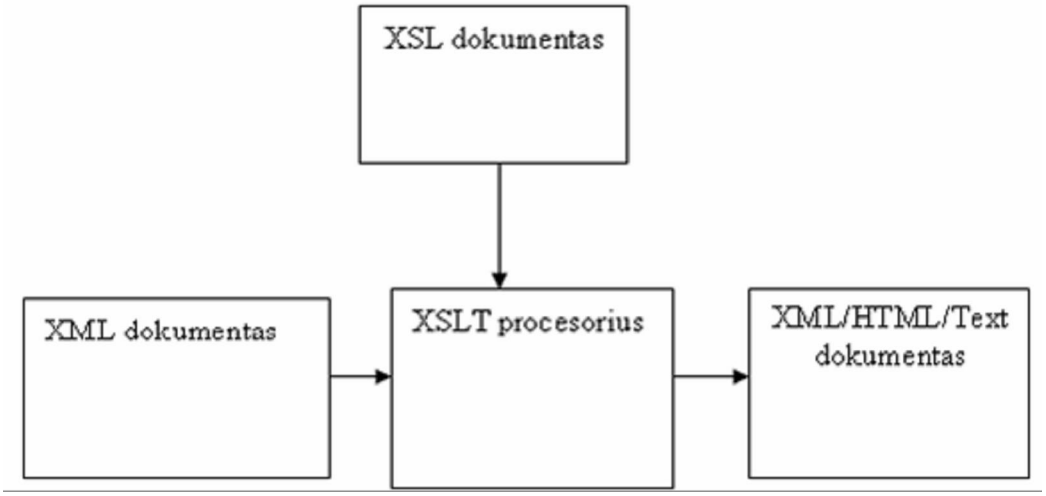
Šiuo metu QVT palaiko modelio – modelio transformacijas. Modelio į tekstą transformacijos yra už QVT specifikacijos ribų. Detalus QVT principai aprašomi OMG standarte [17].

2.6.3. XSLT

XSLT (angl. *eXtensible Stylesheet Language Transformations*) – kalba, aprašanti XML dokumento transformaciją į HTML (angl. *Hyper text Markup Language*) dokumentą arba į kitokios struktūros XML dokumentą. Ši kalba neretai naudojama tinklalapių turiniui tiesiogiai iš XML duomenų generuoti, tačiau tai tik vienas iš daugelio jos pritaikymų. XSLT naudoja XML sintaksę. Nėra iki galo sutarta, ar šią kalbą galima laikyti pilnaverte programavimo kalba, tačiau ja įmanoma realizuoti sudėtingus algoritmus. XSLT yra sukurta naudojimui kaip XSL (angl. Extensible Stylesheet Language) kalbos dalis. 2.2.3 paveiksle parodytas bendras XSLT naudojimo atvejis.

XSLT pavyzdys:

```
    <?xml version="1.0" ?>
    <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
          <xsl:template match="teacher">
                    <p><u><xsl:value-of select="."/></u></p>
          </xsl:template>
          <xsl:template match="student">
                    <p><b><xsl:value-of select="."/></b></p>
          </xsl:template>
          <xsl:template match="/">
                    <html>
                    <body>
                    <xsl:apply-templates/>
                    </body>
                    </html>
          </xsl:template>
    </xsl:stylesheet>
```



2.10 pav. XSLT panaudojimas

2.6.4. Veiklos žodynų transformavimo kalbos analizės apibendrinimas

Tarp ATL ir QVT yra daug bendro: abi naudoja OCL, abi nusakomos MOF metamodeliu, abi palaiko XMI kaip pirminio modelio ir galutinio rezultato formatus. Tačiau, šiuo metu ATL yra viena labiausiai QVT standartą atitinkančiu atviro kodo transformacijos kalbų. Bet dėl savo komponentiškumo ATL turi potencialą būti lengviau plečiama už QVT. Praplečiantys ATL atvirojo kodo įrankiai: AMMA, AMW, AM3, TCS, ir t.t.

ATL galima butų traktuoti kaip skirta platesniam problemų sprendimo ratui, nei QVT kuri orientuojasi dažniausiai į PIM ir PSM. Pavyzdžiui, AMMA platformoje turėtų būti įmanoma transformuoti binarinius failus, kuomet QVT apsiriboja „XMI į XMI transformacijomis“ su apribotu praplėtimu.

Iš architektūrinės pusės QVT ir ATL lyginamos pagal keletą kategorijų: abstraktumas, paradigma, kryptingumas, kardinalumas, ir t.t.. Rezultatai rodo, jog ATL ir QVT yra suderinami ir galimos tarpusavio transformacijos. Tad pasirinkus viena iš šių transformavimo sprendimų būtų prieinama įrankių bazė, bibliotekos, sprendimų analizės, standartų palaikymas, ir t.t. Tačiau analizės metu [22] buvo išanalizuoti J. Cabot ATL pagalba sudarytos transformacijos pavyzdžiai *UML2SBVR*[1] ([7], [8]), kurie lėmė apsisprendimą naudoti ATL transformavimo kalbą šiam metodui realizuoti.

2.7. Veiklos žodynų transformavimo siekiamas sprendimas

Svarbiausias mokslinis rezultatas – tai veiklos taisyklėmis išplėstas veiklos modeliais grindžiamas informacinių sistemų kūrimo metodas, palaikomas standartinių UML 2.x CASE įrankių ir apimantis:

- veiklos taisyklių specifikavimo metodą natūraliajai kalbai artimais šablonais, naudojant probleminės srities žodyną;
- veiklos taisyklių palaikymą informacinių sistemų gyvavimo ciklo etapuose „nuo veiklos koncepcijų iki kodo“ ir transformavimą iš vieno etapo modelių į kito etapo modelius bei programinę realizaciją;
- veiklos taisyklių vaizdavimo UML modeliuose kalbą (UML veiklos taisyklių profilį) – grafinius simbolius, stereotipus, žymes bei apribojimus, kuriais galima atvaizduoti veiklos taisykles UML diagramose;
- metodo iliustraciją pavyzdžiais, metodinius nurodymus ir rekomendacijas.

[1] http://jordicabot.com/research/SBVR/index.html

2.8. Veiklos žodynų transformavimo darbo tikslas ir uždaviniai

Tyrimo tikslas - pagerinti informacinių sistemų ir veiklos modeliais grindžiamo jų kompiuterizuoto kūrimo procesų kokybę, pasiūlant veiklos taisyklių specifikavimo metodą ir jį realizuojančius inžinerinius sprendimus.

Magistriniame darbe bus sprendžiami šie pagrindiniai uždaviniai:

1. Pirmajame etape bus analizuojama ir vertinama esama naujausių mokslinių tyrimų ir inžinerinių sprendimų padėtis veiklos taisyklių, ir modeliais grindžiamo (MDD) informacinių sistemų kūrimo srityje.

2. Veiklos taisyklių specifikavimo metodo sukūrimas. Šiame etape bus sukurti esminiai metodo elementai: atitinkama taisyklių klasifikacija; šablonų rinkinys taisyklių įvedimui natūraliajai kalbai artima forma; su UML metamodeliu integruoti veiklos taisyklių metamodeliai ir jų transformacijos; grafiniai elementai taisyklių vaizdavimui.

3. Sukurtojo metodo inžinerinė realizacija ir jo įvertinimas. Šiame etape bus atliekama metodą realizuojančio įskiepio tipiniam UML CASE įrankiui realizacija (tam numatytas MagicDraw UML CASE įrankis, kurio kūrėjai yra UAB „Baltijos programinė įranga“). Sukurtasis prototipas bus testuojamas, atliekami eksperimentai metodui įvertinti [5].

2.9. Nefunkciniai reikalavimai ir apribojimai

Įrankiui keliami tokie nefunkciniai reikalavimai:

- Grafinė vartotojo sąsaja turi būti realizuota JAVA kalba;
- Galimybė paleisti programą paleidus Magic Draw paketą;
- Apribojimai turi būti įvedami struktūrizuota kalba, artima natūraliai;
- Apribojimai turi būti matomi Magic Draw pakete;
- Sistemos sąsaja turi būti nesudėtinga ir suprantama analitikui ir projektuotojui.

2.10. Įrankio kūrimo rizikos faktorių analizė

Trumpinys **SWOT** (**S**trengths, **W**eaknesses, **O**pportunities, and **T**hreats) lietuviškai rašomas SSGG (stiprybės, silpnybės, galimybės, grėsmės).

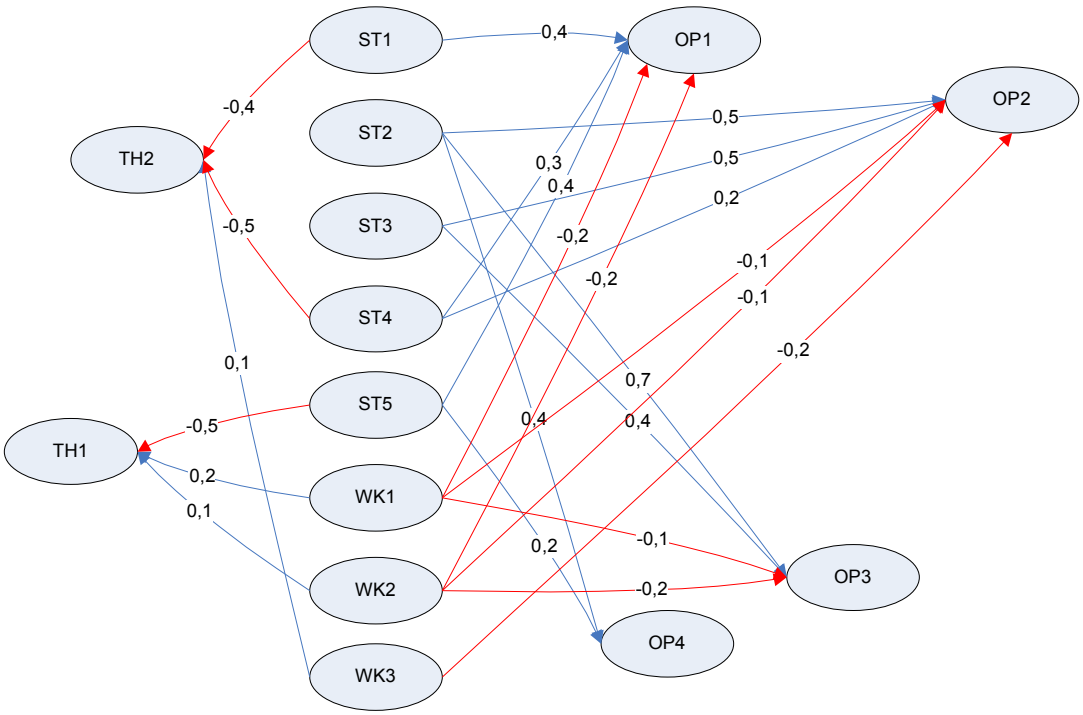
Sistemos projektui 2.2 lentelėje išvardijamos stiprybės, silpnybės (t.y. vidiniai aplinkos veiksniai) ir galimybės bei grėsmės (t.y. išoriniai aplinkos veiksniai).

2.2 lentelė. SSGG analizė (SWOT)

STIPRYBĖS	SILPNYBĖS
1. Geras projekto vykdytojas; 2. Detalus veiklos taisyklių modelis; 3. VT tipų apibrėžimai bei pavyzdžiai; 4. Kvalifikuoti darbuotojai; 5. Naujausios technologijos.	1. Nėra darytų analogų; 2. Nauja sritis; 3. Sunku kontroliuoti projekto eigą.
GALIMYBĖS	**GRĖSMĖS**
1. ES struktūrinių fondų parama, diegiant įskiepį; 2. Mažesnės IS projektavimo ir kūrimo laiko sąnaudos; 3. Mažesni IS projektavimo ir kūrimo kaštai; 4. Lengvesnis reagavimas į IS funkcionavimo pokyčius.	1. Programinio kodo generavimo nesuderinamumas; 2. Aplinkos nestabilumas.

Dar nepradėjus kurti sistemos pradžioje nustatome pradines Stiprybių, Silpnybių, Galimybių ir Grėsmių įverčius, kurie yra labiausiai tikėtini prieš sukuriant įskiepį.

Pateikiamas MPP grafas 2.11 pav., sudarytas pagal SWOT analizės parametrus. Mėlyna spalva nurodo stiprinančią įtaką, o raudona - silpninančią.



2.11 pav. SWOT schema

2.3 lentelėje pateikiama SWOT analizės Stiprybių , Silpnybių, Galimybių ir Grėsmių įverčiai:

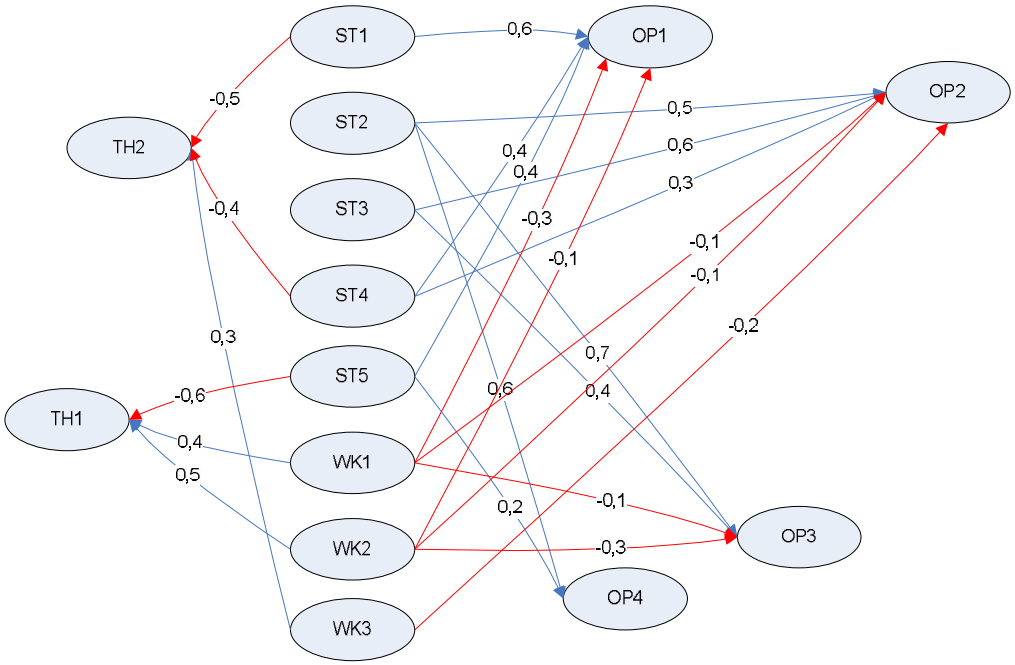
2.3 lentelė. SWOT analizės įverčiai

		Tikėtinumo funkcijos reikšmė	Įtakos stiprumas	Stiprybės					Silpnybės			$\sum$
				ST1	ST2	ST3	ST4	ST5	WK1	WK2	WK3	
Galimybės	OP1	0,6	0,2	0,3			0,3	0,4	-0,5	-0,3		0,16
	OP2	0,4	0,2		0,5	0,5	0,2		-0,2	-0,1	-0,2	0,22
	OP3	0,5	0,3		0,7	0,7			-0,1	-0,3		0,45
	OP4	0,6	0,1		0,6			0,2				0,14
											Bendroji suma:	**0,97**
Grėsmės	TH1	0,8	0,9	-0,5					0,2	0,3		0,72
	TH2	0,7	0,7	-0,2			-0,2				0,1	0,28
											Bendroji suma:	**1**

Gautas įvertis parodo, jog kiekybinis galimybių įvertis yra mažesnis už kiekybinį grėsmių įvertį. Tai rodo, jog neverta atlikti darbo, tačiau visa tai svyruoja ant ribos.

Išanalizuosime įtaką, jeigu mes veiklos taisyklių automatizuoto projektavimo įrankio sukūrimą atidėtume metams. Tarkime, kad pradėtume kurti sistemą po vienerių metų, tuomet atsirastu daugiau veiklos taisyklių aprašų ir pavyzdžių, taip pat pagerėtų ir projekto vadovo įgūdžiai vadovauti projektui, galbūt atsirastu didesnės kvalifikacijos specialistų, taip pat palengvėtų programinio kodo suderinamumas.

Pateikiamas MPP grafas 2.12 pav., sudarytas pagal SWOT analizės parametrus po vienerių metų. Mėlyna spalva nurodo stiprinančią įtaką, o raudona silpninančią.



2.12 pav. SWOT schema po vienerių metų

2.4 lentelėje pateikiama SWOT analizės Stiprybių , Silpnybių, Galimybių ir Grėsmių po vienerių metų įverčiai:

2.4 lentelė. SWOT analizės po vienerių metų įverčiai

		Tikėtinumo funkcijos reikšmė	Įtakos stiprumas	Stiprybės					Silpnybės			∑
				ST1	ST2	ST3	ST4	ST5	WK1	WK2	WK3	
Galimybės	**OP1**	0,2	0,2	0,6			0,4	0,4	-0,3	-0,1		0,24
	OP2	0,7	0,1		0,5	0,6	0,3		-0,1	-0,1	-0,2	0,17
	OP3	0,8	0,2		0,7	0,4			-0,1	-0,3		0,3
	OP4	0,9	0,1		0,6			0,2				0,17
											Bendroji suma:	**0,88**
Grėsmės	**TH1**	0,5	0,9	-0,5					0,1	0,5		0,54
	TH2	0,7	0,8	-0,4			-0,6				0,3	0
											Bendroji suma:	**0,54**

Gautas įvertis parodo, jog kiekybinis galimybių įvertis yra didesnis už kiekybinį grėsmių įvertį. Tai rodo, jog verta atlikti automatizuoto projektavimo įrankio sukūrimą.

Atlikus projekto SWOT analizę gautas bendras galimybių įvertis yra didesnis už bendrą grėsmių įvertį po vienerių metų, tačiau ir esamu laikotarpiu nebūtų nuostolinga vykdyti projektą. Apibendrinus galima sakyti, kad šį projektą realizuoti verta, nes yra abipusė nauda tiek automatizuoto projektavimo įrankio sukūrime, tiek vartotojui, nes būtent vartotojui bus palengvinamas darbas IS projektavimo ir realizavimo srityje.

2.11. Veiklos žodynų transformacijos kokybės kriterijai

Tikslas bus pasiektas, jeigu bus įgyvendinti pagrindiniai projekto reikalavimai:

- VT specifikavimas natūraliajai kalbai artimais šablonais;
- Transformavimą iš vieno etapo modelių į kito etapo modelius bei programinę realizaciją;
- Veiklos taisyklių vaizdavimo UML modeliuose kalbą;
- Realizuotas įskiepis Magic Draw įrankiui

2.12. Analizės išvados

1. Išanalizavus informacinių sistemų projektavimo procesą buvo nustatyta, kad UML kalba sunkiai suprantama dalykinės srities ekspertams, todėl siekiama supaprastinti bendradarbiavimą tarp IS kūrimo proceso dalyvių, pritaikant veiklos taisyklėmis paremtus specifikavimo metodus.
2. Šioje darbo dalyje išnagrinėti analizės priemonių bei metodų pasirinkimo kriterijai, nustatyta, kad veiklos taisyklės turėtų būti specifikuojamos natūralios kalbos šablonais.
3. Analizės metu išanalizuota esamų veiklos žodynų ir taisyklių (VT) surinkimo ir specifikavimo metodai, iš jų buvo pasirinktas SBVR, kadangi tiek veiklos terminų žodynai, tiek ir pačios veiklos taisyklės yra sudaromos naudojant ribotos natūralios kalbos konstrukcijas.
4. Buvo pasirinkta ATL veiklos žodynų transformavimo kalba, nes transformacijos vykdomos metamodelių pagalbą naudojant OCL konstrukcijas ir ji yra labiausiai išvystyta.
5. Analizės metu išsiaiškinta, kad nėra reikiamos žodyno transformacijos, todėl turės būti kuriama nauja SBVR veiklos žodynų transformacija į UML klasių diagramas.
6. Šio darbo tikslas:
   - Gerinti IS kūrimo kokybę;
   - Palengvinti projektų dalyvių sąveikavimą tarpusavyje;
   - Įvesti veiklos žodyną kalba artima natūraliai.
7. Kadangi potencialūs vartotojai projektavimui naudoja Magic Draw, veiklos žodyno įvedimo galimybes tikslinga kurti kaip šio įrankio išplėtimą (įskiepį).

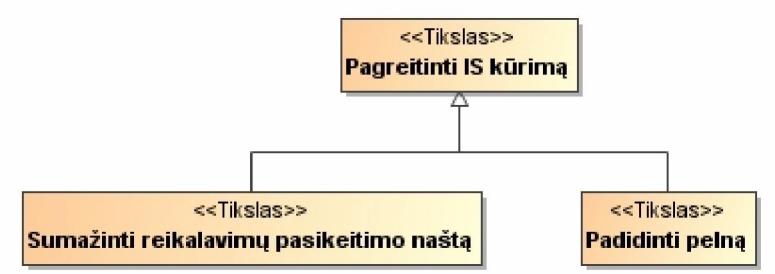
3. Veiklos žodynų transformacijos įrankio reikalavimų specifikacija ir analizė

Šiame skyriuje yra pateikiama kuriamo prototipo projektavimo metodika, aprašoma kokiu principu yra surenkamos taisyklės ir žodynas. Plačiau nagrinėjamas projektavimo etapas: kokia seka yra aprašomos veiklos taisyklės ir kaip generuojama klasių diagrama.

3.1. Veiklos žodynų transformacijos metodo specifikacija

3.1.1. Įrankio kūrimo tikslų modelis

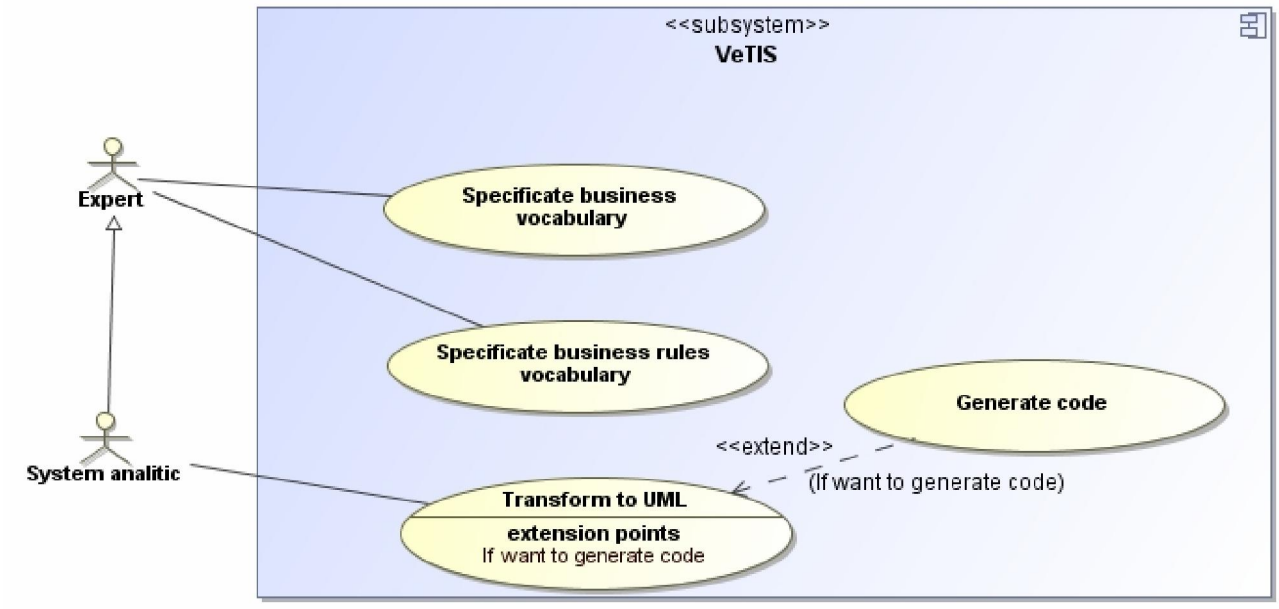
Pagrindiniai kuriamo prototipo tikslai pavaizduoti 3.1 pav.



3.1 pav. SBVR veiklos žodynų transformacijų į UML metodo kūrimo tikslai

3.1.2. Kompiuterizuojamų panaudojimo atvejų diagrama

Kompiuterizuojamų panaudojimo atvejų modelyje išskiriamos vartotojų grupės, bei kuriamos informacinės sistemos paslaugos, kuriomis naudosis šių grupių vartotojai. 3.2 pav. atvaizduota panaudojimo atvejų diagrama, panaudojimo atvejų specifikacija (3.1 - 3.4 lentelėje), o 3.3 pav. scenarijus.



3.2 pav. Kompiuterizuojamų panaudojimo atvejų diagrama

3.1 lentelė. PA "Business rules specification" specifikacija

PA „Specificate business vocabulary"		
Tikslas. Specifikuoti veiklos žodyną		
Aprašymas.		
Aktorius		System analitic, expert
Sužadinimo sąlyga		Vartotojas įvedė veiklos žodyną
Susiję panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas įveda veiklos žodyną		1.1. Sistema išsaugo veiklos žodyną ir atitinkamai jį apdoroja.
2. Vartotojas baigia PA		
Po sąlyga:		
Alternatyvūs scenarijai		

3.2 lentelė. PA "Specificate business rules vocabulary" specifikacija

PA „Specificate business rules vocabulary"		
Tikslas. Specifikuoti veiklos taisykles		
Aprašymas.		
Aktorius		System analitic, expert
Sužadinimo sąlyga		Vartotojas įvedė veiklos taisykles
Susiję panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas įveda veiklos taisykles		1.1. Sistema išsaugo veiklos taisykles ir atitinkamai jas apdoroja tolimesniems veiksmams.
2. Vartotojas baigia PA		
Po sąlyga:		
Alternatyvūs scenarijai		

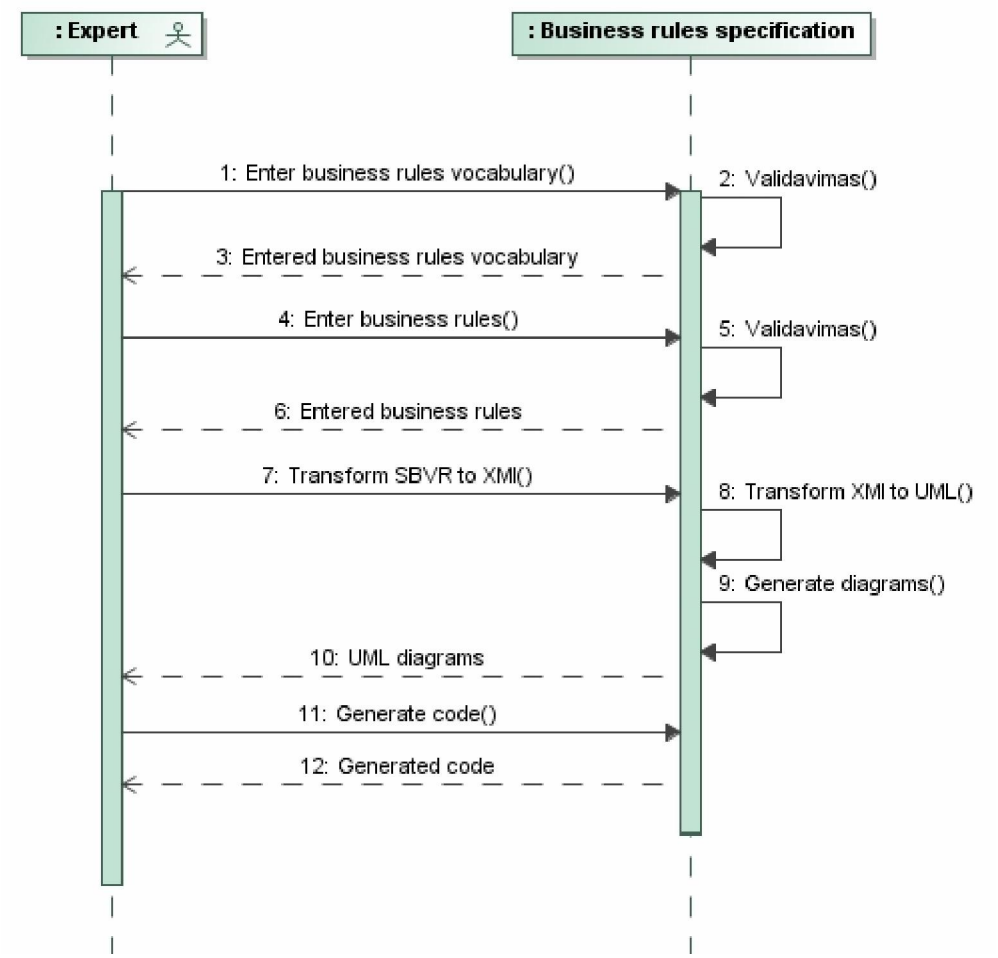
3.3 lentelė. PA "Transform to UML" specifikacija

PA „Transform to UML"		
Tikslas. Transformuoti veiklos taisykles ir žodyną ir gauti UML atitinkančią diagramą		
Aprašymas.		
Aktorius		System analitic
Sužadinimo sąlyga		Vartotojas inicijavo veiklos taisyklių ir žodyno transformavimą
Susiję panaudojimo	**Išplečia PA**	
	Apima PA	

	Specializuoja PA	
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas inicijuoja transformavimą		1.1. Sistema transformuoja veiklos žodyną ir taisykles į UML.
2. Vartotojas baigia PA		
Po sąlyga:		
Alternatyvūs scenarijai		

3.4 lentelė. PA „Generate code“ specifikacija

PA „Generate code“		
Tikslas. Sugeneruoti programinį išeities kodą		
Aprašymas.		
Aktorius		System analitic
Sužadinimo sąlyga		Vartotojas inicijavo kodo generavimą
Susiję panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas inicijuoja kodo generavimą		1.1. Sistema generuoja programinį kodą iš UML diagramų.
2. Vartotojas baigia PA		
Po sąlyga:		
Alternatyvūs scenarijai		



3.3 pav. PA „Business rules specification“ scenarijus

Prototipui keliami tokie nefunkciniai reikalavimai:

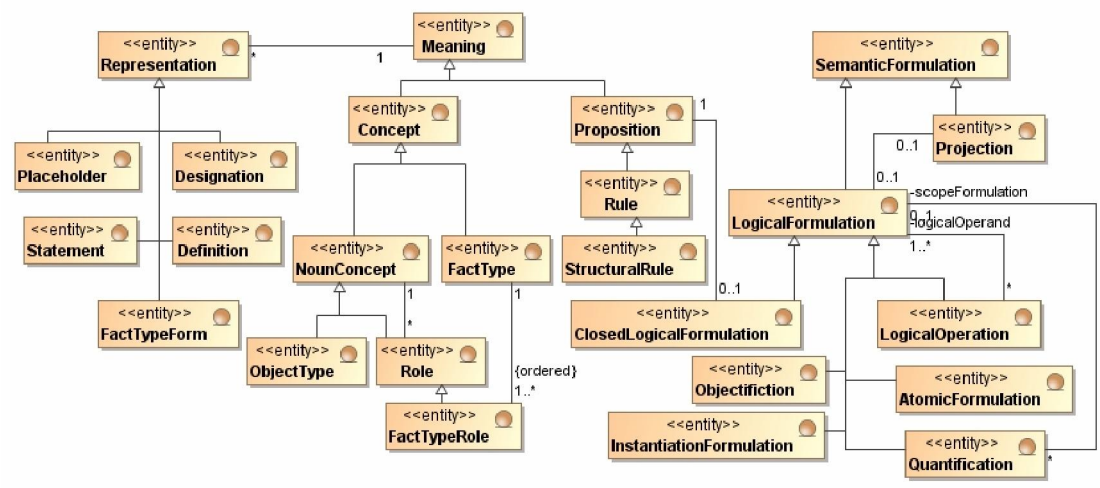
- Sistema turi veikti kartu *su MagicDraw UML* paketu;
- Veiklos konceptai ir taisyklės turi būti įvedamos ribota natūralia kalba;
- Veiklos konceptų ir taisyklių įvedimo sąsaja turi būti intuityviai suprantama veiklos dalyviams;
- Veiklos konceptų ir taisyklių transformavimo sąsaja turi būti patogi analitikams ir projektuotojams

3.2. Veiklos žodynų transformacijų modeliai

3.2.1. Veiklos žodyno transformacijų SBVR modelis

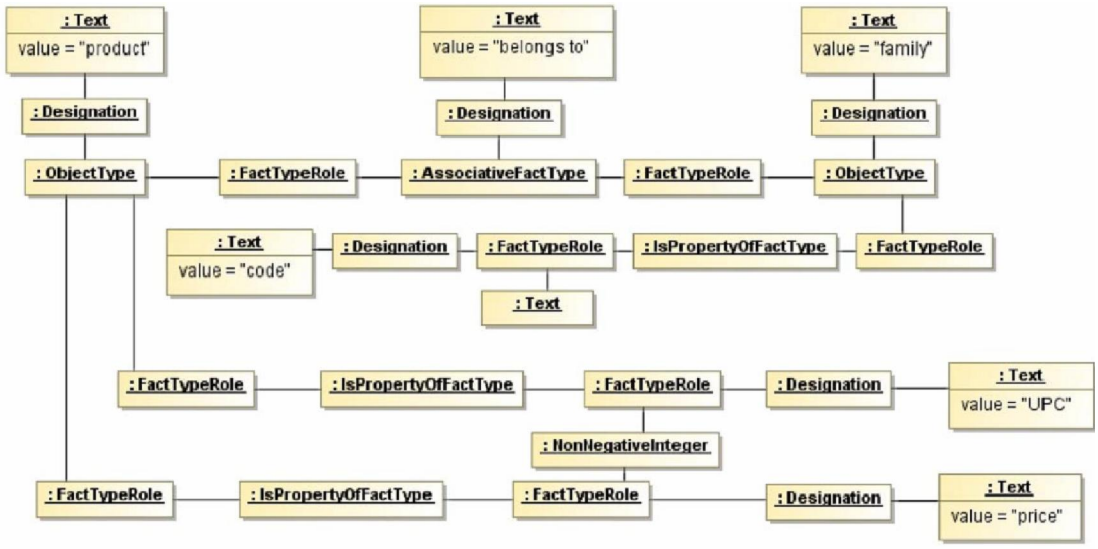
SBVR specifikacija (OMG-SBVR, 2008)[20] apibrėžia veiklos žodyno (*business vocabulary*), veiklos faktų (*business facts*) ir veiklos taisyklių (*business rules*) semantiką bei XMI schemą veiklos žodynų ir veiklos taisyklių apsikeitimui tarp organizacijų ar kompiuterizuotų sistemų. Labai svarbus SBVR akcentas yra tai, kad tiek veiklos terminų žodynai, tiek ir pačios veiklos taisyklės yra sudaromos naudojant natūralios kalbos konstrukcijas – tai yra ypač svarbu

veiklos modeliavimo lygmenyje [16]. Kaip pavyzdys, supaprastintas SBVR metamodelis pateiktas 3.4 pav.



3.4 pav. Supaprastintas SBVR metamodelio fragmentas

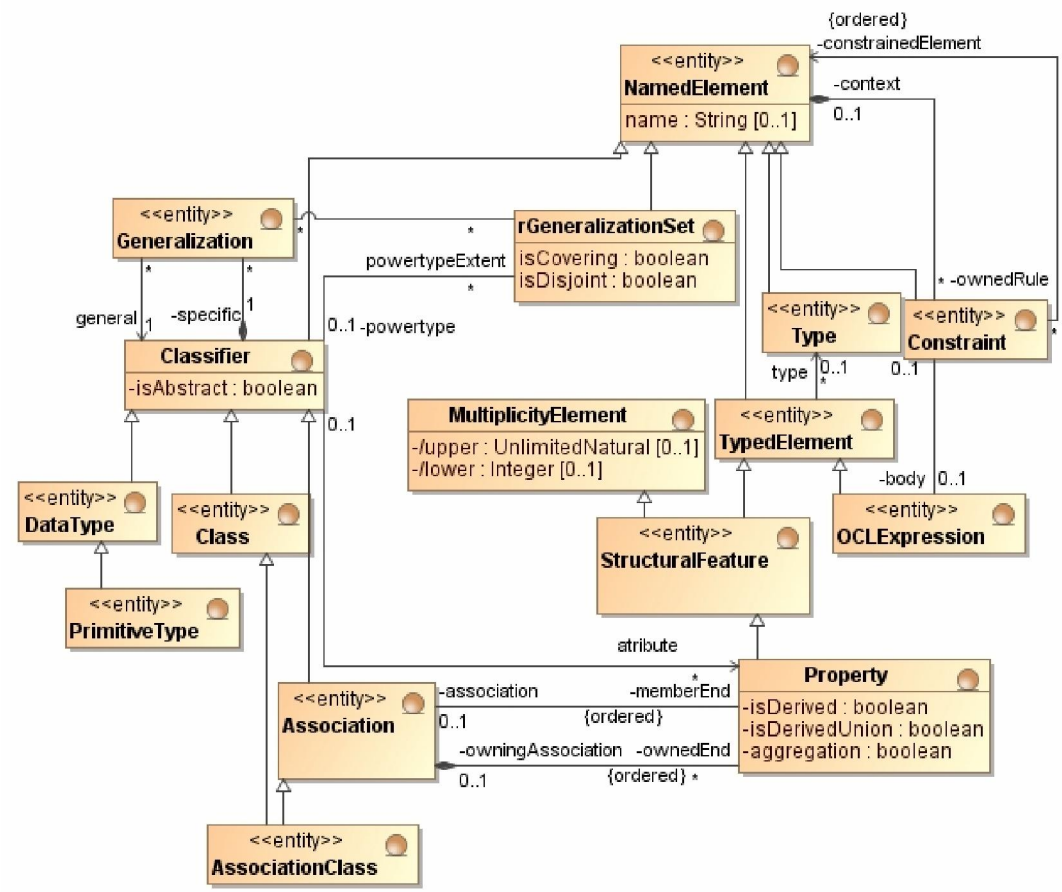
Iš pateikto SBVR modelio 3.5 pav. sudarytas UML modelis (3.7 pav.), kuris remiasi UML metamodeliu (3.6 pav.). Transformacijos metu UML klasės buvo transformuotos iš SBVR objektų tipų (ObjectType) *family, product*. [7]



3.5 pav. SBVR modelis kuris transformuojamas į UML modelį

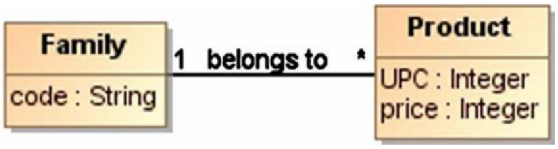
3.2.2. Veiklos žodynų transformacijų UML modelis

UML kalba formaliai remiasi UML metamodeliu kuris nusako abstrakčia kalbos sintaksę. Metamodelis nusako galimus elementus, ryšius, kurie gali būti atvaizduoti UML modelyje. Kaip pavyzdys supaprastintas UML metamodelis pateiktas 3.5 pav., kuris nusako pagrindinius elementus: klases, atributus, asociacijas, apibendrinimą.



3.6 pav. Supaprastintas UML metamodelio fragmentas

UML modelis, remiantis šiuo metamodeliu (3.6 pav.) pavaizduotas paprastoje schemoje 3.7 pav., kuriame pavaizduota *Product* klasė (kiekvienas produktas turi unikalų kodą, kainą) ir *Family* klasė (kuri turi tik kodą).



3.7 pav. UML modelis

3.3. Veiklos žodynų transformacijos metodo reikalavimų analizės apibendrinimas

Reikalavimų specifikavimo metu buvo nustatyta, kad pagrindinė būsimosios informacijos sistemos funkcija bus veiklos taisyklių ir žodyno specifikavimas žmogiškąja kalba ir transformavimas į UML klases naudojantis Eclipse ATL įskiepiu, kuris naudos SBVR ir UML metamodelius. Pagrindinis uždavinys bus suderinti SBVR ir UML modelius. Tam buvo sugalvotas algoritmas, kuris pagal ATL kalbos aprašą transformuoja vieną modelį į kitą, kuris aprašytas 5 skyriuje.

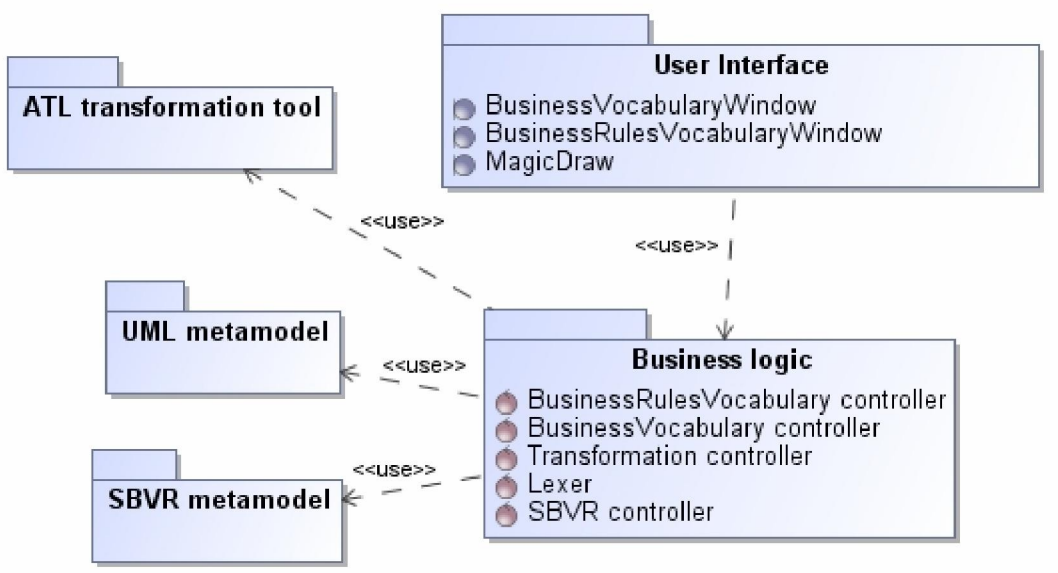
4. Veiklos žodynų transformacijos įskiepio projektas

Šiame skyrelyje aprašomas metodas skirtas transformuoti ribota natūralia kalba specifikuotą žodyną į UML klasių diagramą. Transformacijos vykdomos dviem etapais: pirmiausia žodynas transformuojamas į SBVR XMI modelį, kuris kitame etape, naudojant ATL transformavimo procesorių, transformuojamas į UML modelį, kitame etape gautas modelis importuojamas į Magic Draw ir sugeneruojamas klasių diagramos modelis.

4.1. Įrankio architektūros projektas

4.1.1. Įrankio loginė architektūra

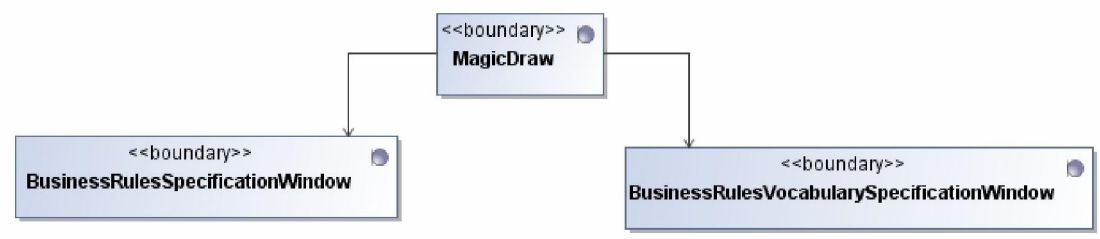
Sistemos loginė architektūra (4.1 pav.) parodo jos pagrindinius architektūrinius komponentus. Pagrindinis komponentas yra „Business Logic“, kuriame apdorojamos natūralios kalbos taisyklės ir vykdomos transformacijos panaudojant metamodelius transformavimo įrankiu ATL.



4.1 pav. IS loginė architektūra

4.1.2. Vartotojo paslaugos

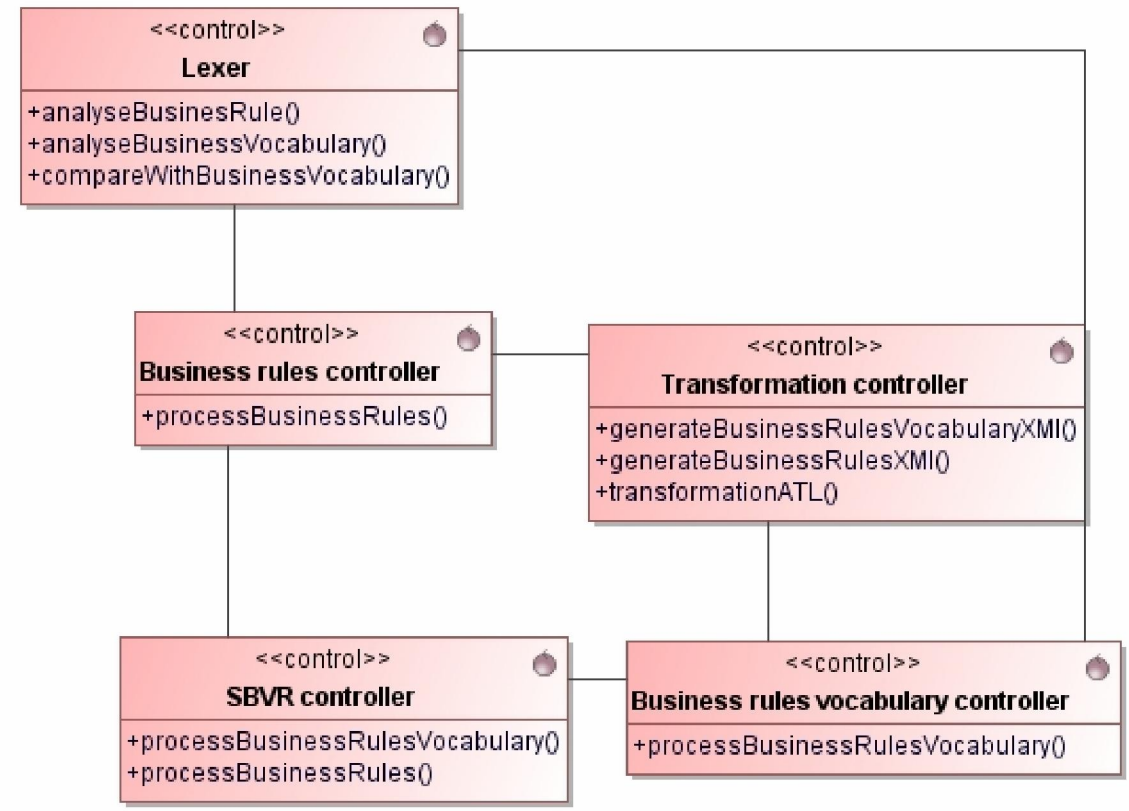
Vartotojo sąsają sudaro MagicDraw langas iš kurio yra iškviečiami kiti kuriamo įskiepio langai, tokie kaip taisyklių ir žodyno įvedimo ir redagavimo, tai pavaizduoja 4.2 pav.



4.2 pav. IS vartotojo sąsajos planas

4.1.3. Veiklos paslaugos

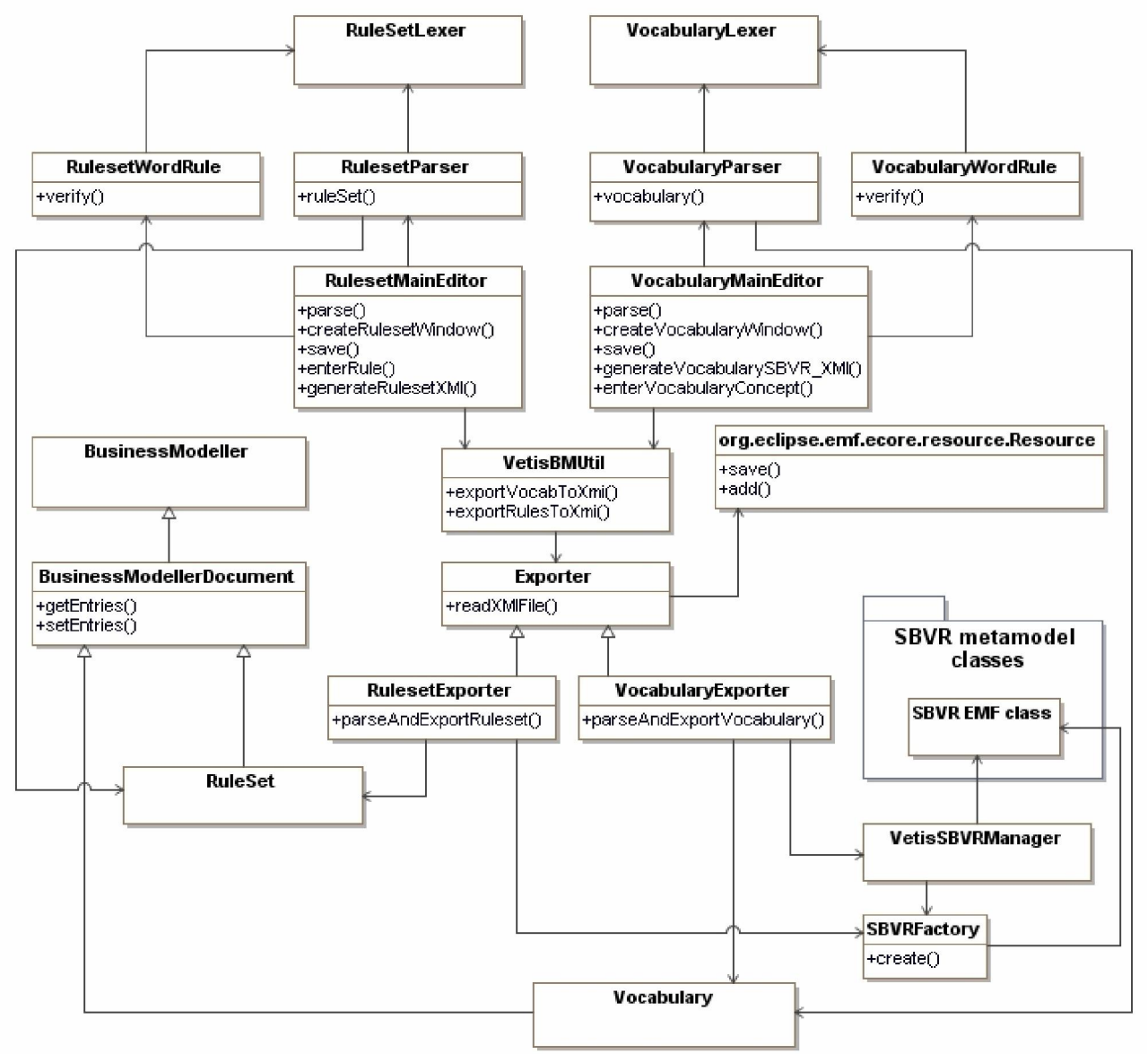
Veiklos paslaugas sudaro penkios klasės: Lexer, Business rules controller, SBVR controller, Transformation controller, Business rules vocabulary controller. Lexer – gramatiškai apdoroja gautas veiklos taisykles ir žodyną ir išskaido jas į atskirus objektų tipus, Business rules controller – apdoroja veiklos taisykles, SBVR controller – šioje klasėje saugoma visa veiksmų logika, Transformation controller – ši klasė atlieka visus transformavimo veiksmus, Business rules vocabulary controller – apdoroja veiklos žodyno objektus.



4.3 pav. IS veiklos paslaugų valdikliai

4.2. Sukurto įrankio detalus projektas

Pateiktoje klasių diagramoje (4.4 pav.) matyti projektuojamos IS klasės, kurios skirtos veiksmams su veiklos taisyklėmis ir UML klasių diagramomis atlikti.



4.4 pav. Sistemos klasių diagrama

4.1 lentelė. Pagrindinių įrankio klasių aprašymas

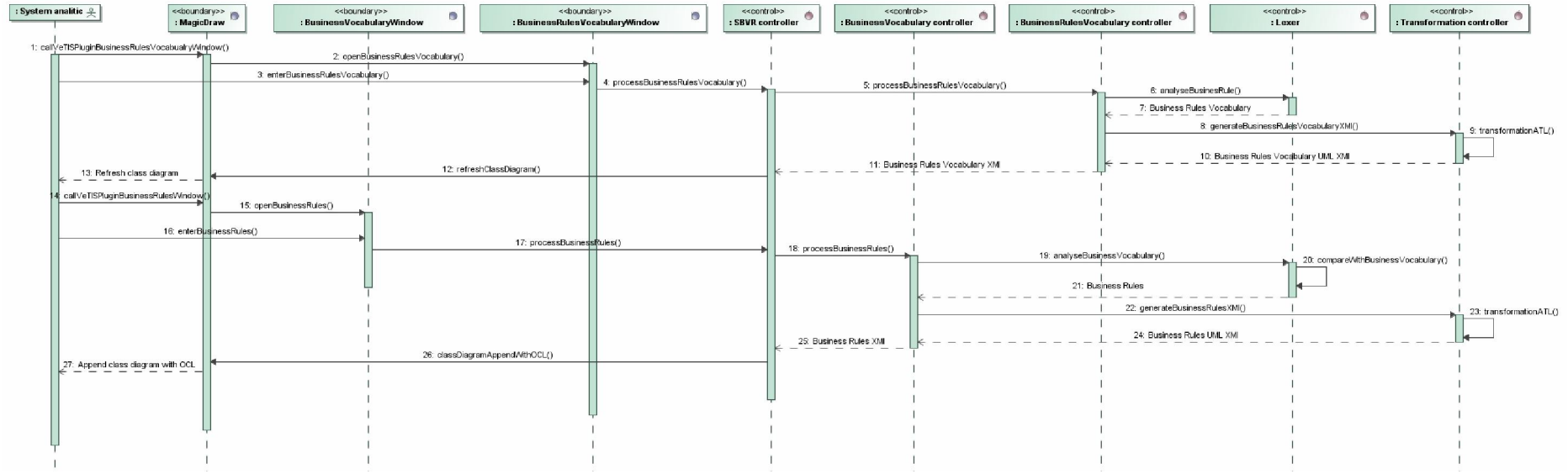
Klasės pavadinimas	Aprašymas
RuleSetLexer	Taisyklių gramatikos klasė, kurioje aprašomi veiklos taisykllių simboliai, žodyno atpažinimo funkcijos
	Pagrindiniai metodai
	mNUMBER() – skaičiaus atpažinimas *mSIGNIFIER()* – žodžio iš mažųjų raidžių atpažinimas mICSIGNIFIER() – žodžio iš didžiosios raidės atpažinimas *mWS()* – tarpo atpažinimas
VocabularyLexer	Žodyno gramatikos klasė, kurioje aprašomi veiklos žodyno simboliai.
	Pagrindiniai metodai
	mSIGNIFIER() – žodžio iš mažųjų raidžių atpažinimas mICSIGNIFIER() – žodžio iš didžiosios raidės atpažinimas *mWS()* – tarpo atpažinimas
RulesetWordRule	Taisyklių gramatikos analizės klasė, skirta įvestų veiklos taisyklių žodžiams nuspalvinti.
	Pagrindiniai metodai
	verify() – veiklos taisyklių validavimas
VocabularyWordRule	Žodyno gramatikos analizės klasė, skirta įvestiems veiklos konceptams nuspalvinti.
	Pagrindiniai metodai
	verify() – veiklos žodyno validavimo metodas
RulesetParser	Taisyklių gramatikos analizės klasė, skirta veiklos taisyklių dokumentui sudaryti iš vartotojo įvesto teksto.
	Pagrindiniai metodai
	ruleSet() – veiklos taisyklių analizė, taisyklių dokumento sudarymas
V*ocabularyParser*	Žodyno gramatikos analizės klasė, skirta veiklos žodyno dokumentui iš vartotojo įvesto teksto sudaryti
	Pagrindiniai metodai
	vocabulary() – veiklos žodyno dokumento sudarymas
BusinessModeller	Redaktoriaus nustatymų klasė
BusinessModellerDocument	Apibendrintas VeTIS redaktoriaus dokumentas.
	Pagrindiniai metodai
	getEntries() – dokumento įrašų gavimas *setEntries()* – dokumento įrašų nustatymas
RuleSet	VeTIS redaktoriaus taisyklių saugojimo dokumentas.
Vocabulary	VeTIS redaktoriaus žodyno saugojimo dokumentas.
VetisBMUtil	VeTIS redaktoriaus eksportavimo į *XMI* sąsaja. Ši sąsaja naudojama programos viduje, tačiau per ją prie eksportavimo funkcijų prisijungia ir *VeTIS MagicDraw* įskiepis (*VeTIS MD UML plugin)*.
	Pagrindiniai metodai
	exportVocabToXmi() – veiklos žodyno dokumento eksportavimas į *SBVR XMI* schemą *exportRulesToXmi()* – veiklos taisyklių dokumento eksportavimas į *SBVR XMI* schemą
Exporter	Apibendrinta eksportavimo į *XMI* klasė.

	Pagrindiniai metodai
	readXMIFile() – sugeneruoto XMI SBVR failo nuskaitymas ir grąžinimas tekstiniu formatu
org.eclipse.emf.ecore.resource.Resource	*Eclipse Modeling Framework (EMF)* klasė, skirta saugoti sukurtus *SBVR* elementus.
	Pagrindiniai metodai
	add() – naujo SBVR elemento priskyrimas resursui *save()* – resurse saugojamų SBVR elementų išsaugojimas *SBVR XMI* schemoje
RulesetExporter	Ši klasė nagrinėja sukurtą veiklos taisyklių dokumentą ir juo remiantis sukuria taisyklių *SBVR* elementus, kuriuos išsaugo *XMI* faile.
	Pagrindiniai metodai
	parseAndExportRuleset() – veiklos taisyklių analizė ir eksportavimas į SBVR XMI schemą
VocabularyExporter	Ši klasė nagrinėja sukurtą veiklos žodyno dokumentą ir pagal jį sukuria žodyno *SBVR* elementus, kuriuos išsaugo *XMI* faile.
	Pagrindiniai metodai
	parseAndExportVocabulary() – veiklos žodyno analizė ir eksportavimas į SBVR XMI schemą
SBVRFactory	Klasė, skirta sukurti vieną *SBVR* elementą (*ObjectType*, *Designation*, *Text* ir kt.).
	Pagrindiniai metodai
	createObjectType() – objekto tipo elemento kūrimas *createText()* – teksto elemento kūrimas *createFactType()* – fakto tipo elemento kūrimas ...
VetisSBVRManager	Klasė, skirta sukurti ir sujungti *SBVR* elementus pagal *SBVR* standartą, pvz. norint transformuoti veiklos terminą į *SBVR*, sukuriamas ne tik *ObjectType* elementas, bet ir *Text*, *Designation*, *Term*, taip pat jie sujungiami tarpusavyje.
	Pagrindiniai metodai
	createVetisText() – tekstinio elemento kūrimas bei patikrinimas, ar schemoje nėra tokio paties sukurto anksčiau *createVetisPositiveInteger()* – teigiamo sveikojo skaičiaus kūrimas bei patikrinimas, ar schemoje nėra tokio paties sukurto anksčiau ...
SBVR EMF class	Apibendrinta *SBVR* elemento klasė. Jų yra daug (*ObjectType*, *FactType*, *Text*,...), todėl dėl vietos stokos jos nerodomos. Šios klasės sugeneruojamos naudojant *EMF* technologiją, t.y. iš sukurto *SBVR* metamodelio.

Remiantis klasių diagrama (4.4 pav.) pateikiami klasių aprašymai 4.1 lentelėje.

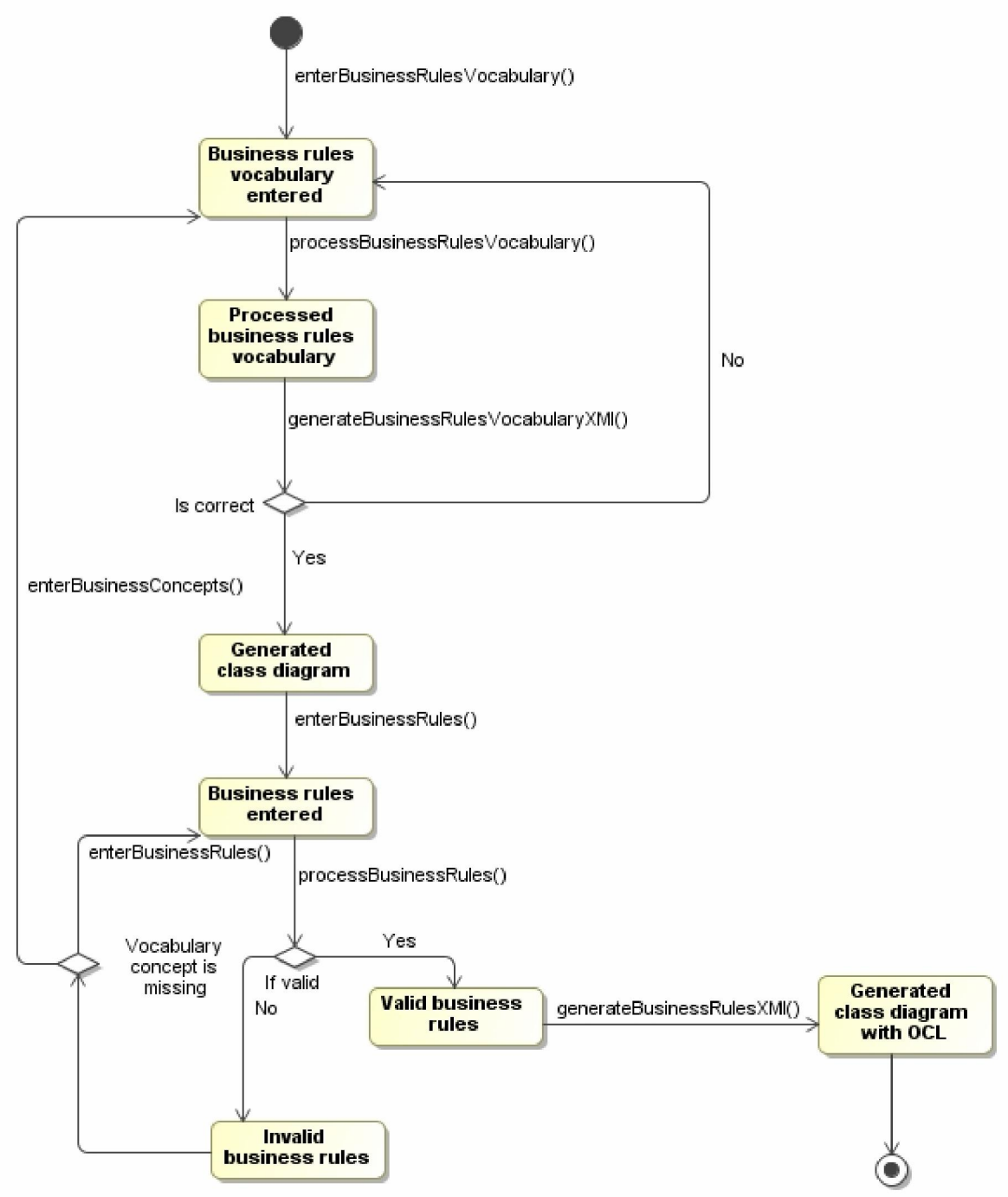
4.3. Įrankio elgsenos modelis

Sekų diagramoje (4.5 pav.) yra pateikiamas sistemos veikimo algoritmas. Aprašomas transformavimo algoritmas, kuris iš vartotojo natūralia kalba įvestu taisyklių ir žodyno sudaro klasių diagramą su OCL apribojimais.



4.5 pav. Veiklos taisyklų specifikavimo sekų diagrama

Ši diagrama aprašo sudaromas klasių diagramos būsenas ir jų kitimus, taip pat joje atsispindi visa modelio sukūrimo logika (4.6 pav.).



4.6 pav. UML klasių diagramos būsenų diagrama

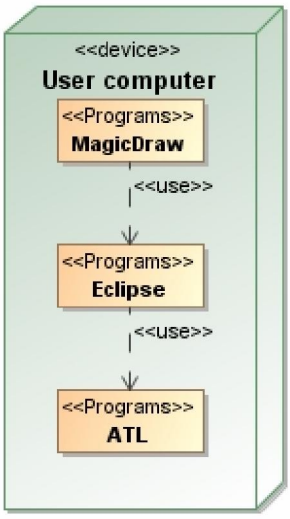
5. Veiklos žodynų transformacijos įskiepio realizacija

Šis darbas yra Lietuvos universitetų vykdomo projekto „Veiklos taisyklių sprendimai informacinių sistemų kūrimui (VeTIS)“ dalis. Projekte sukurto VeTIS[2] įrankio tikslas – leisti specifikuoti veiklos žodynus ir veiklos taisykles pagal OMG SBVR standartą ir vykdyti transformaciją SBVR → UML / OCL. Jis realizuoja OMG veiklos žodyno ir veiklos taisyklių standarto „Semantics of Business Vocabulary and Business Rules“ (SBVR) viziją – išreikšti veiklos žinias ribota natūralia kalba, kurią vienareikšmiškai supranta žmonės ir kompiuterinės programos. VeTIS įrankis taip pat atitinka OMG MDA viziją – transformuoja formaliai aprašytas veiklos žinias į programinės įrangos modelius, išreiškiamus grafine UML ir tekstine OCL kalba.

Įrankis realizuotas *Eclipse* 3.4.1 platformoje taikant *SBVR 1.0* ir *UML 2.1.2* metamodelius (*XMI* formatu), *ATL* transformacijų kalbą 3.0.0 ir *ATL* transformavimo procesorių 3.0.0 yra SBVR standarto pagrindu parengtas MagicDraw UML CASE įrankio įskiepis, tačiau VeTIS redaktorių galima naudoti ir kaip savarankišką įrankį (Eclipse įskiepį). VeTIS redaktoriaus vartotojo sąsaja buvo sukurta adaptuojant SBeaVeR Eclipse įskiepį, kuris yra Digital Business Ecosystem projekto dalis [18].

5.1. Prototipo veikimo aprašymas

Pateiktoje diagramoje (5.1 pav.) matyti projekto programinės realizacijos diegimo principinė schema ir papildomi jo komponentai. Visa sistema bus diegiama viename kompiuteryje, kuris turės JAVA palaikymą, bei įdiegtą „MagicDraw“ programą. O pats realizuojamas projektas veiks kaip Eclipse paremtas „MagicDraw“ įskiepis.



5.1 pav. Sistemos diegimo diagrama

[2] http://www.magicdraw.com/main.php?ts=navig&cmd_show=1&menu=plugins#free

Įskiepis skirtas sugeneruoti *SBVR XMI* schemą iš ribota natūralia kalba įvesto veiklos žodyno ir veiklos taisyklių. Redaktorių reikia patalpinti *Eclipse plugins* kataloge. Jo veikimui reikalingi šie *Eclipse* įskiepiai:

5.1 lentelė. Eclipse programoje naudojami įskiepiai

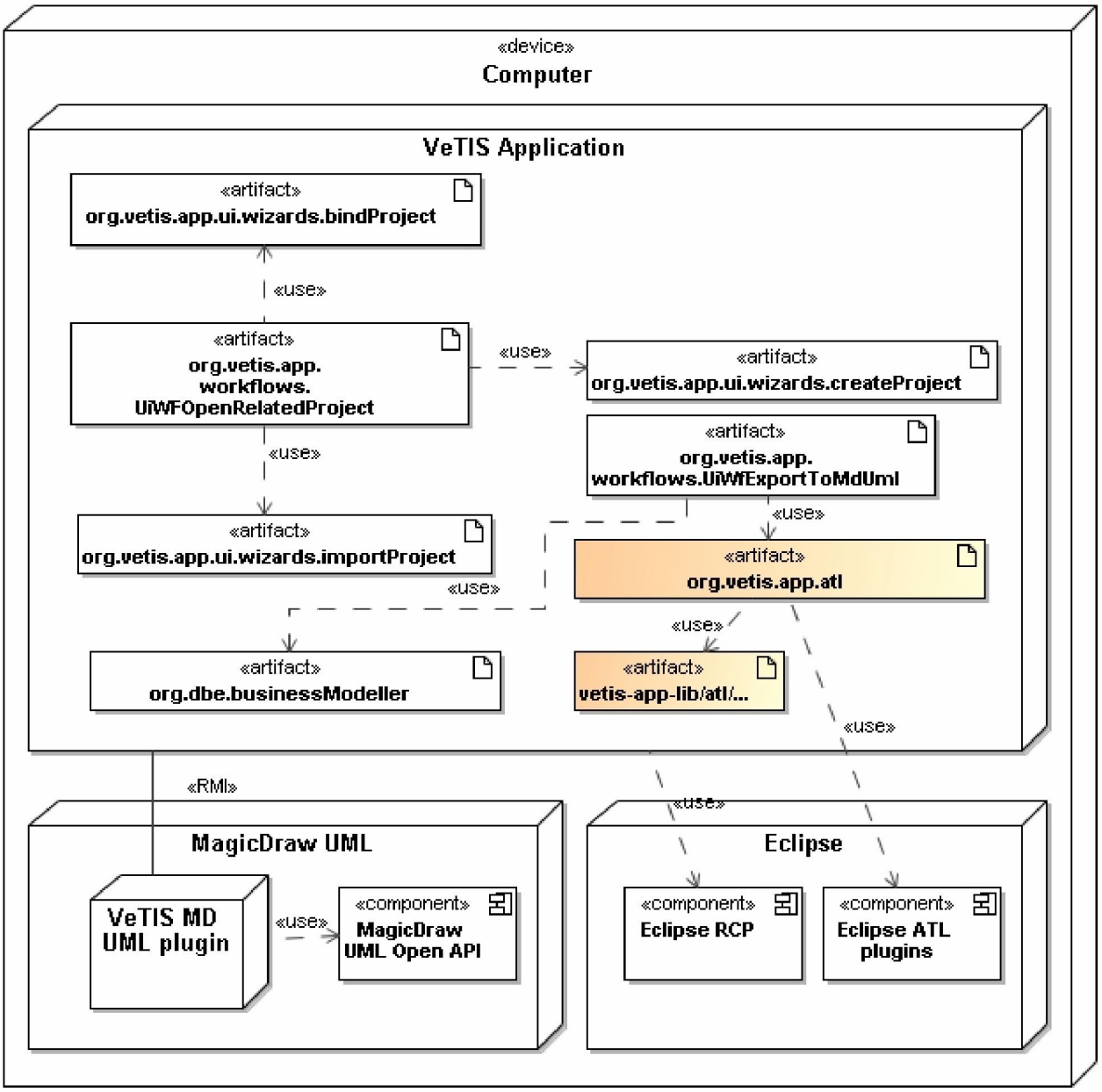
Eclipse įskiepis	Aprašymas
org.eclipse.ui	Programų kūrimo sąsajos, komunikavimui su *Eclipse* platformos vartotojo sąsaja.
org.eclipse.core.runtime	Palaiko *Eclipse* vykdymo platformą.
org.eclipse.jface.text	Manipuliavimo su tekstiniais dokumentais sistema.
org.eclipse.core.resources	Darbo srities (workspace) ir jo ištekliais palaikymo ir manipuliavimo įskiepis.
org.eclipse.ui.editors	Pagrindinė *Eclipse* redaktoriaus klasė.
org.eclipse.ui.ide	Pagrindinis *Eclipse* platformos vartotojo sąsajos elementas.
org.eclipse.ui.workbench.texteditor	Tekstinio redaktoriaus komponentas.
org.eclipse.ui.views	Darbastalio dalis kuri leidžia naviguoti tarp hierarchinių objektų ir jų savybių.
org.eclipse.ui.forms	Formos kurias naudoja vedliai, redaktoriai ir hierarchiniai sąrašai.
org.eclipse.emf	*Eclipse* modeliavimo karkasas.
org.eclipse.xsd	Teikia programų kūrimo sąsaja XML schemoms.
org.eclipse.emf.ecore.xmi	Naudojamas skaityti ir rašyti į XML failus.
org.dbe.businessModeller.lib	Pagalbinės VeTIS redaktoriaus bibliotekos

Projekto prototipo realizavimo diagrama pavaizduota (5.2 pav.). Norint naudoti prototipą kaip MagicDraw įskiepį, kompiuteryje turi būti įdiegta MagicDraw UML 16.0 arba aukštesnė versija. Į MagicDraw plugins katalogą nukopijuojamas MD UML įskiepis, kuris su MagicDraw įrankiu bendrauja (importuoja UML schemas, jas modifikuoja, kuria diagramas) per MagicDraw UML Open API komponentą.

Kita įskiepio dalis yra VeTIS programa (VeTIS Application) [15], kuri yra Eclipse redaktorius.

Ši programa turi projekto kūrimo, importavimo komponentus, VeTIS redaktorių, kurio sugeneruotas SBVR XMI schemas transformuoja į UML&OCL naudojant ATL. Sugeneruotas UML&OCL schemas VeTIS programa importuoja į MagicDraw per VeTIS MD UML įskiepį, su kuriuo bendrauja per RMI technologiją.

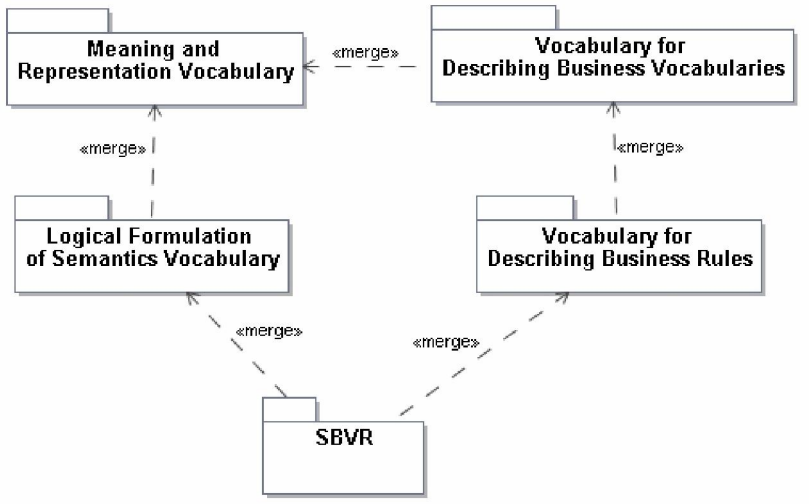
VeTIS projektas buvo komandinis darbas, tačiau šio magistrinio darbo autoriaus pagrindinė dalis – ATL transformacijos, kurių realizacija pateikta kita spalva (5.2 pav.). Ji skirta transformuoti SBVR XMI schemai į UML XMI schemą.



5.2 pav. Sistemos realizavimo diagrama

Norint pritaikyti SBVR metamodelį siekiamam redaktoriui, iškilo problemų dėl to, kad SBVR standarte pateikiamas metamodelis aprašytas atskirais fragmentais ir nėra tinkamas veiklos žodynų ir taisyklių redaktoriams realizuoti. Darbe naudojama *Jordi Cabot* realizuoto įrankio UML2SBVR keturi metamodeliai (5.3 pav):

- *LFSV* (Logical Formulation of Semantics Vocabulary);
- *VDBR* (Vocabulary for Describing Business Rules);
- *VDBV* (Vocabulary for Describing Business Vocabularies);
- *MRV* (Meaning and Representation Vocabulary).

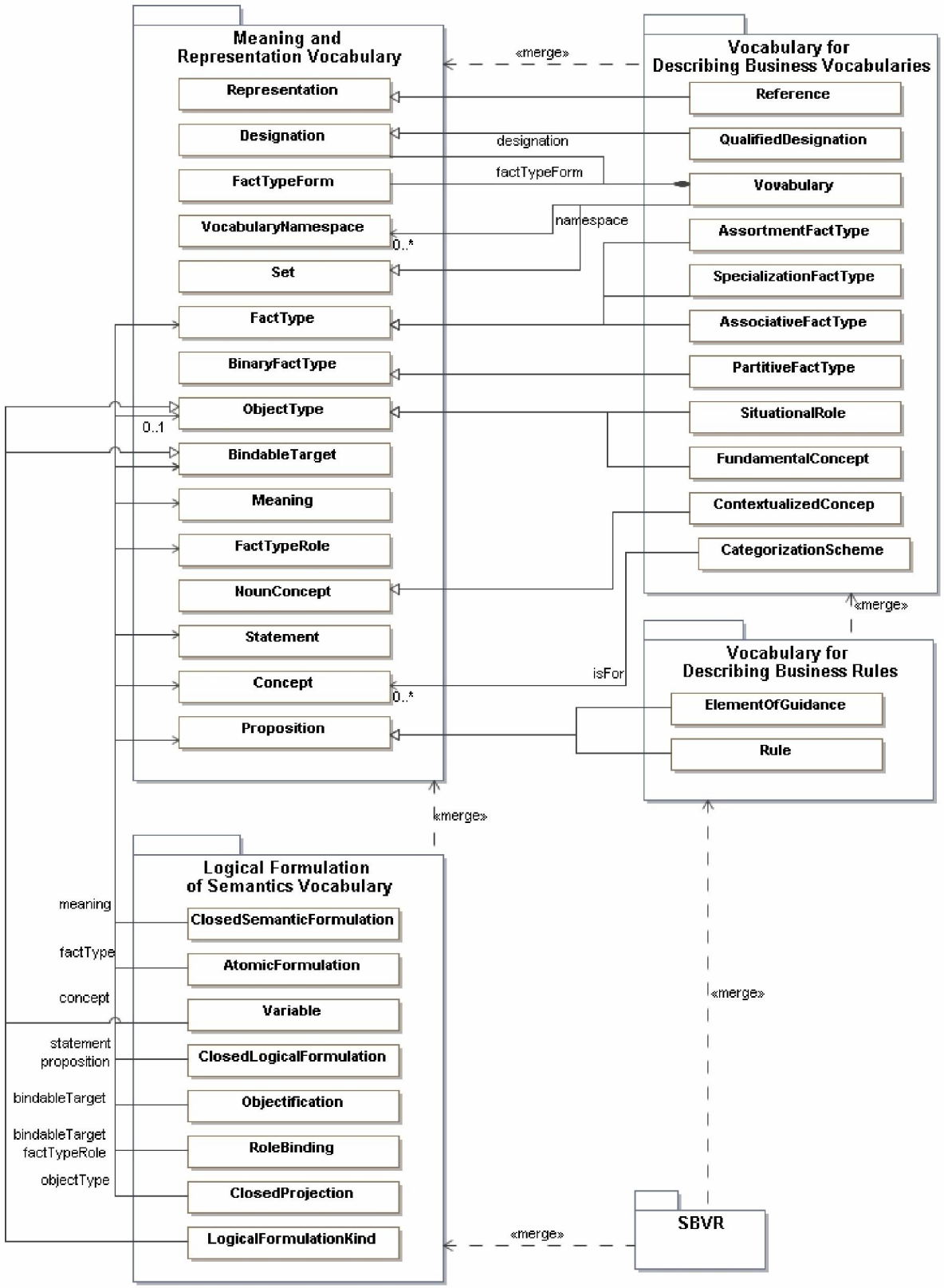


5.3 pav. Ryšys tarp SBVR metamodelių

MRV skirtas aprašyti pagrindinius veiklos terminus, jų sąryšius. Svarbiausios šio metamodelio dalys yra šios:

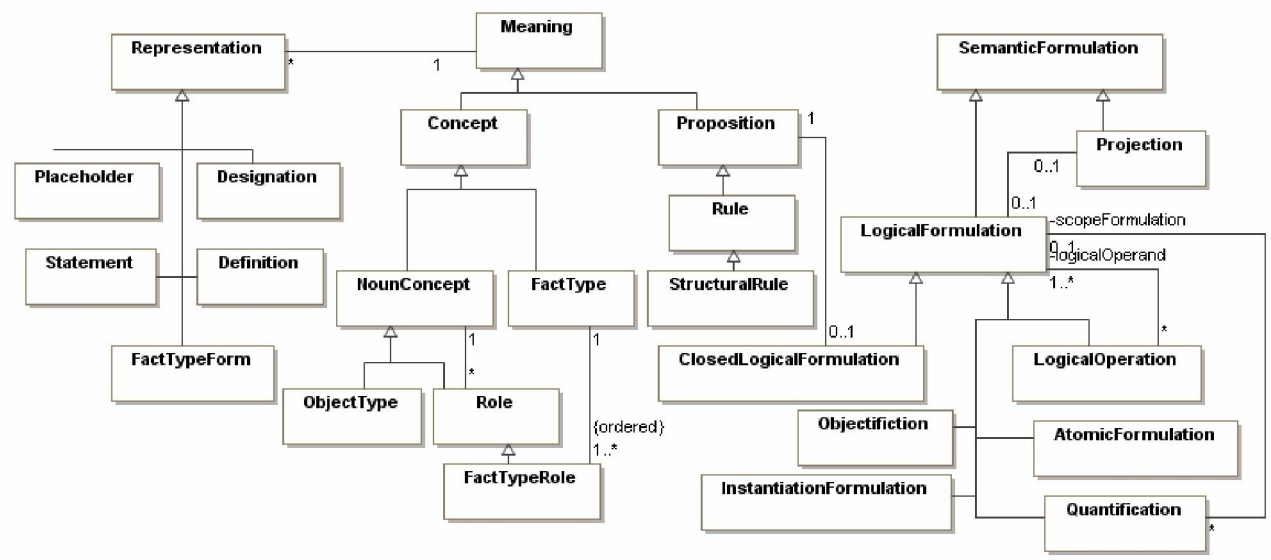
- Reiškinys (angl. *expression*) – veiklos koncepto tekstinis aprašas;
- Atvaizdis (angl. *representation*) – koncepto ir jo tekstinės išraiškos sujungimo elementas;
- Prasmė (angl. *meaning*) – veiklos terminas;
- Ekstensija (angl. *extension*) – realaus pasaulio egzempliorių aibė.
- VDBV yra pagrindinis žodyno aprašymo metamodelis, aprašo veiklos konceptus ir faktų tipus, papildomą informaciją apie žodyną.
- VDBR - veiklos taisyklių aprašymo metamodelis, kuris skirsto veiklos taisykles į operacines ir struktūrines.
- LFSV skirtas aprašyti veiklos taisyklių, išreikštų natūralia kalba, struktūrą semantinėmis formuluotėmis, kurios būna dviejų tipų:
    - loginės formuluotės – skirtos aprašyti loginėms operacijoms, kvantoriams, atominėms formuluotėms, kurios paremtos faktų tipais ir kitos formuluotėms;
    - projekcijos – formuluoja aprašus, agregacijas bei klausimus.

Šie keturi metamodeliai buvo sujungti į vieną SBVR metamodelį tam, kad būtų paprasčiau vykdyti ATL transformacijas. Metamodeliai buvo jungiami per bendrus elementus, naudojant grafinį Eclipse diagramų redaktorių [23]. MRV metamodelis buvo naudojamas kaip pagrindinis. Prie jo buvo prijungtas VDBV metamodelis, kuris naudoja MRV elementus. Vėliau buvo prijungti VDBR ir LFSV metamodeliai. 5.4 paveiksle pavaizduota metamodelių jungimo schema. Schemoje pavaizduoti tik pagrindiniai metamodelių elementai.



5.4 pav. SBVR metamodelių sujungimo schema

5.5 paveiksle pateiktas sujungto SBVR metamodelio fragmentas.



5.5 pav. Pagrindinės SBVR metamodelio klasės

5.2. Sukurto metodo testavimas

Programos testavimas buvo vykdomas keletą etapų. Kiekviena programos dalis buvo testuojama atskirai prieš apjungiant į vieningą sistemą (integravimo testavimo metodas). Šis metodas buvo naudojamas kartu su testavimo strategija "iš apačios į viršų" (angl. "bottom-up") t.y. prieš apjungiant žemesnio lygio komponentus į vieningą sistemą, yra įsitikinama, kad jie individualiai dirba teisingai, o tik tada testuojama integruotai.

Sukurtasis programinis prototipas (redaktorius) ir transformacija buvo testuojami sugeneruojant kiekvienam veiklos žodyno koncepto ir veiklos taisyklės sakinio tipui XMI schemą ir transformuojant ATL pagalba ją į atitinkamus UML XMI schemos elementus, kurie buvo importuojami į MagicDraw UML įrankį ir pavaizduojami klasių diagrama. Iš pavaizduotos klasių diagramos buvo sprendžiama ar natūrali kalba buvo tinkamai konvertuota į grafinius elementus. Kadangi sudėtingiems elementams reikėjo didelių fragmentų, čia pateikiami principiniai fragmentai, atspindintys atitinkamų elementų metamodelius ir transformacijas testavimo metu.

Veiklos žodyno konceptui Vocabulary sugeneruojamas individualus konceptas, jo vardas, nurodoma kalba ir unikalus identifikatorius URI:

Konceptas	Vocabulary
Sugeneruotos SBVR XMI schemos fragmentas	
<?xml version="1.0" encoding="cp1257"?> <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:SBVR="http://www.eclipse.org/sbvr/1.0.0/SBVR">	

```
<SBVR:Vocabulary xmi:id="_7J5fYKHdEd6PwLziY7l4iA"/>
<SBVR:Name xmi:id="_7J5fYaHdEd6PwLziY7l4iA" meaning="_7J5fY6HdEd6PwLziY7l4iA"
 expression="_7J5fYqHdEd6PwLziY7l4iA" preferred="true"
    refersTo="_7J5fYKHdEd6PwLziY7l4iA"/>
<SBVR:Text xmi:id="_7J5fYqHdEd6PwLziY7l4iA" value="Loan Contracts"/>
<SBVR:IndividualConcept xmi:id="_7J5fY6HdEd6PwLziY7l4iA"
 instance="_7J5fYKHdEd6PwLziY7l4iA"/>
<SBVR:Language xmi:id="_7J5fZKHdEd6PwLziY7l4iA"/>
<SBVR:Name xmi:id="_7J5fZaHdEd6PwLziY7l4iA" meaning="_7J5fZqHdEd6PwLziY7l4iA"
 expression="_7J5fZ6HdEd6PwLziY7l4iA" preferred="true"
    refersTo="_7J5fZKHdEd6PwLziY7l4iA"/>
<SBVR:IndividualConcept xmi:id="_7J5fZqHdEd6PwLziY7l4iA"
 instance="_7J5fZKHdEd6PwLziY7l4iA"/>
<SBVR:Text xmi:id="_7J5fZ6HdEd6PwLziY7l4iA" value="English"/>
<SBVR:URI xmi:id="_7J5faKHdEd6PwLziY7l4iA"
 value="http://www.loc.gov/standards/iso639-2/php/English_list.php"/>
```

Žodynas transformuojamas į *UML* paketą, kuris įdedamas į aukštesnio lygio paketą – modelį. :

Transformuotos *UML XMI* schemos elementas Package

```
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
 xmlns:uml="http://www.eclipse.org/uml2/2.1.0/UML"
    xmi:id="_E0WAsKt8Ed68LcNaTXhLcw" name="Data" viewpoint="">
    <ownedComment xmi:id="_E0o7oKt8Ed68LcNaTXhLcw">
       <body>Author:Created by VeTIS</body>
    </ownedComment>
<packagedElement xmi:type="uml:Package" xmi:id="_E0o7oat8Ed68LcNaTXhLcw"
    name="Loan Contracts">
```

Transformuotos UML XMI schemos elementas Package ir importuotas į MagicDraw

Kiekvienas elementas testuojamas atskirai. Ribota natūralia kalba užrašyti sakiniai transformuojami į SBVR XMI schemą, vėliau ATL transformacijos pagalba schema transformuojama į UML schemą, kuri importuojama į MagicDraw. MagicDraw programoje sugeneruota klasių diagrama lyginama su ribota natūralia kalba užrašytais sakiniais ir vertinama ar gautas rezultatas teisingas.

6. Veiklos žodynų transformacijos metodo eksperimentinis tyrimas

Realizuotas įrankis skirtas transformuoti ribota natūralia kalba specifikuotą žodyną į UML klasių diagramą. Vykdant eksperimentą buvo iškelta hipotezė, kad pagrindinius UML klasių elementus (klases, klasių atributus ir ryšius) galima gauti transformavus iš ribota natūralia kalba specifikuotų konceptų.

Kiekvieno elemento transformavimas buvo vykdomas atskirai. Žodyno elementai buvo transformuojami į SBVR XMI CIM modelį. Gauta SBVR XMI schema transformuojama į UML XMI PIM schemą ir importuojama į Magic Draw. Iš Magic Draw sugeneruotos klasių diagramos buvo sprendžiama ar žodyno elementas transformuotas korektiškai.

6.1. Termino transformavimas

Veiklos žodyno konceptui Object_type sugeneruojamas koncepto tipas, kuris įdedamas į koncepcinę schemą. Koncepcinė schema priklauso faktų modeliui. Sukuriamas terminas, žymėjimas ir tekstas su objekto tipo vardu, į žodyno vardų erdvę įtraukiama objekto tipo žymėjimo designation nuoroda:

Konceptas	bank

Sugeneruotos SBVR XMI schemos fragmentas

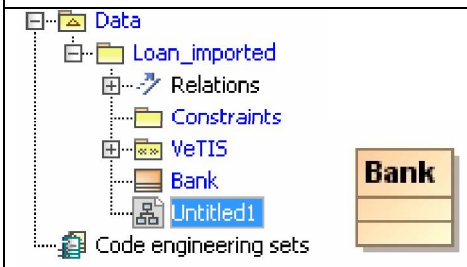
```
<SBVR:ConceptualSchema xmi:id="_YPMBSeY_Ed6uQNG2vQOEZw">
 <concept xmi:type="SBVR:ObjectType" xmi:id="_YPMBSuY_Ed6uQNG2vQOEZw"/
    </SBVR:ConceptualSchema>
<SBVR:FactModel xmi:id="_YPMBS-Y_Ed6uQNG2vQOEZw"
 conceptualSchema="_YPMBSeY_Ed6uQNG2vQOEZw"/>
<SBVR:VocabularyNamespacexmi:id="_YPMBTOY_Ed6uQNG2vQOEZw"
 designation="_YPMBQeY_Ed6uQNG2vQOEZw _YPMBReY_Ed6uQNG2vQOEZw
    _YPMBTuY_Ed6uQNG2vQOEZw _YPMBT-Y_Ed6uQNG2vQOEZw"
       uri="_YPMBSOY_Ed6uQNG2vQOEZw"
          language="_YPMBROY_Ed6uQNG2vQOEZw"/>
<SBVR:Text xmi:id="_YPMBTeY_Ed6uQNG2vQOEZw" value="bank"/>
<SBVR:Designation xmi:id="_YPMBTuY_Ed6uQNG2vQOEZw"
 meaning="_YPMBSuY_Ed6uQNG2vQOEZw" expression="_YPMBTeY_Ed6uQNG2vQOEZw"/>
<SBVR:Term xmi:id="_YPMBT-Y_Ed6uQNG2vQOEZw" meaning="_YPMBSuY_Ed6uQNG2vQOEZw"
 expression="_YPMBTeY_Ed6uQNG2vQOEZw" preferred="true"/>
```

Object_type transformuojamas į klasę:

Transformuotos *UML XMI* schemos elementas

```
<packagedElement xmi:type="uml:Class" xmi:id="_FE4XEKt8Ed68LcNaTXhLcw" name="Bank">
</packagedElement>
```

Transformuotos UML XMI schemos elementas importuotas į MagicDraw ir atvaizduotas grafiškai



6.2. Faktų tipo transformavimas

Transformuojant žodyno faktų tipo konceptą, pagrindinis sukuriamas elementas yra *fact type,* kuris per *designation* elementą jungiasi su dviem terminais. Taip pat faktų tipas turi dvi roles. Toliau pateikiamas sugeneruotas asociacijos ryšio SBVR schemos fragmentas. Kitų faktų tipų (agregavimas, apibendrinimas) schemos yra tokios pačios tik skiriasi konceptų tipas.

Konceptas	bank *gives* loan

Sugeneruotos SBVR XMI schemos fragmentas

```
<SBVR:ConceptualSchema xmi:id="_O17L2eY_Ed6uQNG2vQOEZw">
<concept xmi:type="SBVR:ObjectType" xmi:id="_O17L2uY_Ed6uQNG2vQOEZw"/>
<concept xmi:type="SBVR:ObjectType" xmi:id="_O17L2-Y_Ed6uQNG2vQOEZw"/>
<concept xmi:type="SBVR:AssociativeFactType" xmi:id="_O17L3OY_Ed6uQNG2vQOEZw">
  <role xmi:id="_O17L3eY_Ed6uQNG2vQOEZw" objectType="_O17L2uY_Ed6uQNG2vQOEZw"/>
  <role xmi:id="_O17L3uY_Ed6uQNG2vQOEZw" objectType="_O17L2-Y_Ed6uQNG2vQOEZw"/>
</concept>
</SBVR:ConceptualSchema>
<SBVR:FactModel xmi:id="_O17L3-Y_Ed6uQNG2vQOEZw"
conceptualSchema="_O17L2eY_Ed6uQNG2vQOEZw"/>
<SBVR:VocabularyNamespace xmi:id="_O17L4OY_Ed6uQNG2vQOEZw"
designation="_O17L0eY_Ed6uQNG2vQOEZw _O17L1eY_Ed6uQNG2vQOEZw
   _O17L4uY_Ed6uQNG2vQOEZw _O17L4-Y_Ed6uQNG2vQOEZw _O17L5eY_Ed6uQNG2vQOEZw
      _O17L5uY_Ed6uQNG2vQOEZw _O17L5-Y_Ed6uQNG2vQOEZw _O17L6-Y_Ed6uQNG2vQOEZw
   _O17L7uY_Ed6uQNG2vQOEZw _O17L7-Y_Ed6uQNG2vQOEZw" uri="_O17L2OY_Ed6uQNG2vQOEZw"
language="_O17L1OY_Ed6uQNG2vQOEZw"/>
<SBVR:Text xmi:id="_O17L4eY_Ed6uQNG2vQOEZw" value="bank"/>
<SBVR:Designation xmi:id="_O17L4uY_Ed6uQNG2vQOEZw"
meaning="_O17L2uY_Ed6uQNG2vQOEZw"
expression="_O17L4eY_Ed6uQNG2vQOEZw"/>
<SBVR:Term xmi:id="_O17L4-Y_Ed6uQNG2vQOEZw" meaning="_O17L2uY_Ed6uQNG2vQOEZw"
expression="_O17L4eY_Ed6uQNG2vQOEZw" preferred="true"/>
<SBVR:Text xmi:id="_O17L5OY_Ed6uQNG2vQOEZw" value="loan"/>
<SBVR:Designation xmi:id="_O17L5eY_Ed6uQNG2vQOEZw" meaning="_O17L2Y_Ed6uQNG2vQOEZw"
```

```
   expression="_O17L5OY_Ed6uQNG2vQOEZw"/>
   <SBVR:Term xmi:id="_O17L5uY_Ed6uQNG2vQOEZw" meaning="_O17L2-Y_Ed6uQNG2vQOEZw"
   expression="_O17L5OY_Ed6uQNG2vQOEZw" preferred="true"/>
   <SBVR:Designation xmi:id="_O17L5-Y_Ed6uQNG2vQOEZw"
   meaning="_O17L3OY_Ed6uQNG2vQOEZw"
   expression="_O17L6uY_Ed6uQNG2vQOEZw"/>
   <SBVR:SententialForm xmi:id="_O17L6OY_Ed6uQNG2vQOEZw"
   meaning="_O17L3OY_Ed6uQNG2vQOEZw" expression="_O17L7eY_Ed6uQNG2vQOEZw"
      designation="_O17L6-Y_Ed6uQNG2vQOEZw _O17L7uY_Ed6uQNG2vQOEZw"/>
   <SBVR:PositiveInteger xmi:id="_O17L6eY_Ed6uQNG2vQOEZw" value="1"/>
   <SBVR:Text xmi:id="_O17L6uY_Ed6uQNG2vQOEZw" value="gives"/>
   <SBVR:Placeholder xmi:id="_O17L6-Y_Ed6uQNG2vQOEZw"
   meaning="_O17L3eY_Ed6uQNG2vQOEZw"
   expression="_O17L4eY_Ed6uQNG2vQOEZw"
      isAtStartingCharacterPosition="_O17L6eY_Ed6uQNG2vQOEZw"/>
   <SBVR:PositiveInteger xmi:id="_O17L7OY_Ed6uQNG2vQOEZw" value="12"/>
   <SBVR:Text xmi:id="_O17L7eY_Ed6uQNG2vQOEZw" value="bank gives loan"/>
   <SBVR:Placeholder xmi:id="_O17L7uY_Ed6uQNG2vQOEZw"
   meaning="_O17L3uY_Ed6uQNG2vQOEZw"
   expression="_O17L5OY_Ed6uQNG2vQOEZw"
      isAtStartingCharacterPosition="_O17L7OY_Ed6uQNG2vQOEZw"/>
   <SBVR:FactSymbol xmi:id="_O17L7-Y_Ed6uQNG2vQOEZw" meaning="_O17L3OY_Ed6uQNG2vQOEZw"
   expression="_O17L6uY_Ed6uQNG2vQOEZw" preferred="true"/>
```

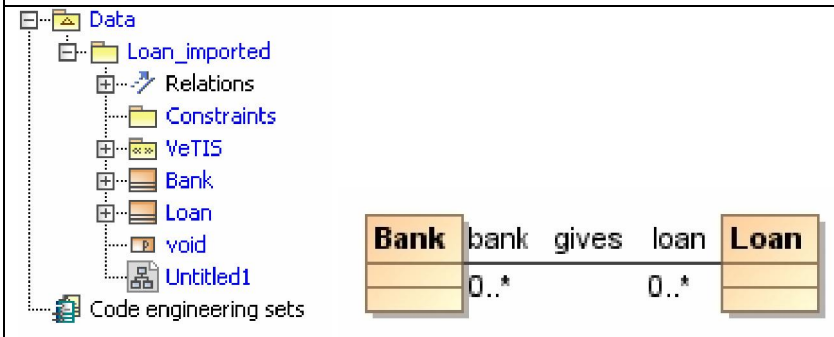
Veiklos žodyno konceptas associative_fact_type transformuojamas į asociacija

Transformuotos *UML XMI* schemos elementas

```
   <packagedElement xmi:type="uml:Class" xmi:id="_F3Hu8Ay5Ed-GTMgM16I5Vw" name="Bank">
         <ownedAttribute xmi:id="_F3Hu8Qy5Ed-GTMgM16I5Vw" name="loan"
visibility="private" type="_F3Hu9Ay5Ed-GTMgM16I5Vw" association="_F3Hu-Qy5Ed-
GTMgM16I5Vw">
            <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_F3Hu8gy5Ed-
GTMgM16I5Vw" name="" value="*"/>
            <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_F3Hu8wy5Ed-GTMgM16I5Vw"
name=""/>
         </ownedAttribute>
        </packagedElement>
        <packagedElement xmi:type="uml:Class" xmi:id="_F3Hu9Ay5Ed-GTMgM16I5Vw"
name="Loan">
         <ownedAttribute xmi:id="_F3Hu9Qy5Ed-GTMgM16I5Vw" name="bank"
visibility="private" type="_F3Hu8Ay5Ed-GTMgM16I5Vw" association="_F3Hu-Qy5Ed-
GTMgM16I5Vw">
            <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_F3Hu9gy5Ed-
GTMgM16I5Vw" name="" value="*"/>
            <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_F3Hu9wy5Ed-GTMgM16I5Vw"
name=""/>
```

```
            </ownedAttribute>
          </packagedElement>
          <packagedElement xmi:type="uml:Association" xmi:id="_F3Hu-Qy5Ed-GTMgM16I5Vw"
name="gives" memberEnd="_F3Hu9Qy5Ed-GTMgM16I5Vw _F3Hu8Qy5Ed-GTMgM16I5Vw"/>
```

Transformuotos UML XMI schemos elementas importuotas į MagicDraw ir atvaizduotas grafiškai



6.3. isPropertyOf faktų tipo transformavimas

Aprašomi du fakto tipai ir *isPropertyOf* fakto tipas. Jis atpažįstamas pagal veiksmažodį *has*. Nuo asociacijos fakto tipo jis skiriasi tuo, kad pirmoji fakto tipo rolė naudojama aprašyti atributui bei atsiranda papildomas objekto tipas aprašyti atributo duomenų tipui (nagrinėjamame pavyzdyje *text*), jis naudojamas kaip pirmos rolės *objectType*.

Konceptas	loan amount loan *has* amount

Sugeneruotos SBVR XMI schemos fragmentas

```
<SBVR:ConceptualSchema xmi:id="_lNXSeeZAEd6uQNG2vQOEZw">
<concept xmi:type="SBVR:ObjectType" xmi:id="_lNXSeuZAEd6uQNG2vQOEZw"/>
<concept xmi:type="SBVR:Role" xmi:id="_lNXSe-ZAEd6uQNG2vQOEZw"/>
<concept xmi:type="SBVR:IsPropertyOfFactType" xmi:id="_lNXSfOZAEd6uQNG2vQOEZw">
  <role xmi:id="_lNXSfeZAEd6uQNG2vQOEZw" objectType="_lNXSf-ZAEd6uQNG2vQOEZw"/>
  <role xmi:id="_lNXSfuZAEd6uQNG2vQOEZw" objectType="_lNXSeuZAEd6uQNG2vQOEZw"/>
</concept>
<concept xmi:type="SBVR:ObjectType" xmi:id="_lNXSf-ZAEd6uQNG2vQOEZw"/>
</SBVR:ConceptualSchema>
<SBVR:FactModel xmi:id="_lNXSgOZAEd6uQNG2vQOEZw"
 conceptualSchema="_lNXSeeZAEd6uQNG2vQOEZw"/>
<SBVR:VocabularyNamespace xmi:id="_lNXSgeZAEd6uQNG2vQOEZw"
 designation="_lNXSceZAEd6uQNG2vQOEZw _lNXSdeZAEd6uQNG2vQOEZw _lNXSg
 ZAEd6uQNG2vQOEZw _lNXShOZAEd6uQNG2vQOEZw _lNXShuZAEd6uQNG2vQOEZw _lNXSh
ZAEd6uQNG2vQOEZw _lNXSiOZAEd6uQNG2vQOEZw _lNXSjOZAEd6uQNG2vQOEZw _lNXSj
```

```
ZAEd6uQNG2vQOEZw _lNXSkOZAEd6uQNG2vQOEZw _lNXSkuZAEd6uQNG2vQOEZw _lNXSk
ZAEd6uQNG2vQOEZw" uri="_lNXSeOZAEd6uQNG2vQOEZw"
language="_lNXSdOZAEd6uQNG2vQOEZw"/>
<SBVR:Text xmi:id="_lNXSguZAEd6uQNG2vQOEZw" value="loan"/>
<SBVR:Designation xmi:id="_lNXSg-ZAEd6uQNG2vQOEZw"
 meaning="_lNXSeuZAEd6uQNG2vQOEZw" expression="_lNXSguZAEd6uQNG2vQOEZw"/>
<SBVR:Term xmi:id="_lNXShOZAEd6uQNG2vQOEZw" meaning="_lNXSeuZAEd6uQNG2vQOEZw"
expression="_lNXSguZAEd6uQNG2vQOEZw" preferred="true"/>
<SBVR:Text xmi:id="_lNXSheZAEd6uQNG2vQOEZw" value="amount"/>
<SBVR:Designation xmi:id="_lNXShuZAEd6uQNG2vQOEZw" meaning="_lNXSe
ZAEd6uQNG2vQOEZw" expression="_lNXSheZAEd6uQNG2vQOEZw"/>
<SBVR:Term xmi:id="_lNXSh-ZAEd6uQNG2vQOEZw" meaning="_lNXSe-ZAEd6uQNG2vQOEZw"
expression="_lNXSheZAEd6uQNG2vQOEZw"/>
<SBVR:Designation xmi:id="_lNXSiOZAEd6uQNG2vQOEZw"
meaning="_lNXSfOZAEd6uQNG2vQOEZw" expression="_lNXSi-ZAEd6uQNG2vQOEZw"/>
<SBVR:SententialForm xmi:id="_lNXSieZAEd6uQNG2vQOEZw"
meaning="_lNXSfOZAEd6uQNG2vQOEZw" expression="_lNXSjuZAEd6uQNG2vQOEZw"
designation="_lNXSjOZAEd6uQNG2vQOEZw _lNXSj-ZAEd6uQNG2vQOEZw"/>
<SBVR:PositiveInteger xmi:id="_lNXSiuZAEd6uQNG2vQOEZw" value="10"/>
<SBVR:Text xmi:id="_lNXSi-ZAEd6uQNG2vQOEZw" value="has"/>
<SBVR:Placeholder xmi:id="_lNXSjOZAEd6uQNG2vQOEZw"
meaning="_lNXSfeZAEd6uQNG2vQOEZw" expression="_lNXSheZAEd6uQNG2vQOEZw"
isAtStartingCharacterPosition="_lNXSiuZAEd6uQNG2vQOEZw"/>
<SBVR:PositiveInteger xmi:id="_lNXSjeZAEd6uQNG2vQOEZw" value="1"/>
<SBVR:Text xmi:id="_lNXSjuZAEd6uQNG2vQOEZw" value="loan has amount"/>
<SBVR:Placeholder xmi:id="_lNXSj-ZAEd6uQNG2vQOEZw"
meaning="_lNXSfuZAEd6uQNG2vQOEZw" expression="_lNXSguZAEd6uQNG2vQOEZw"
isAtStartingCharacterPosition="_lNXSjeZAEd6uQNG2vQOEZw"/>
<SBVR:FactSymbol xmi:id="_lNXSkOZAEd6uQNG2vQOEZw" meaning="_lNXSfOZAEd6uQNG2vQOEZw"
expression="_lNXSi-ZAEd6uQNG2vQOEZw" preferred="true"/>
<SBVR:Text xmi:id="_lNXSkeZAEd6uQNG2vQOEZw" value="text"/>
<SBVR:Designation xmi:id="_lNXSkuZAEd6uQNG2vQOEZw" meaning="_lNXSf
ZAEd6uQNG2vQOEZw" expression="_lNXSkeZAEd6uQNG2vQOEZw"/>
<SBVR:Term xmi:id="_lNXSk-ZAEd6uQNG2vQOEZw" meaning="_lNXSf-ZAEd6uQNG2vQOEZw"
expression="_lNXSkeZAEd6uQNG2vQOEZw" preferred="true"/>
```

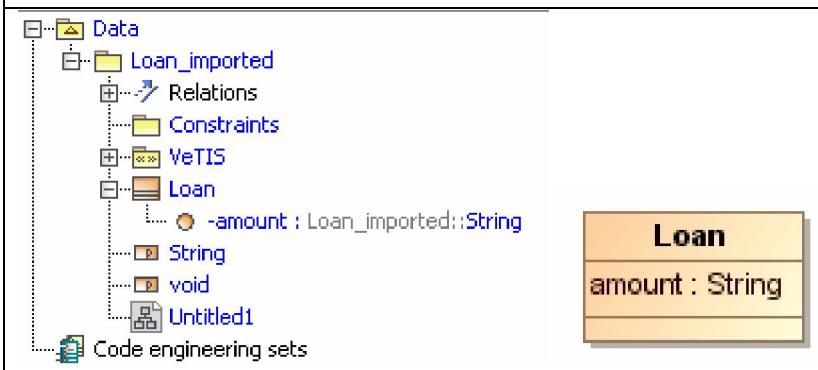
Du fakto tipai ir *isPropertyOf* fakto tipas transformuojamas į klasę ir atributą.

Transformuotos *UML XMI* schemos elementas

```
<packagedElement xmi:type="uml:Class" xmi:id="_kiZUAwy6Ed-GTMgM16I5Vw" name="Loan">
        <ownedAttribute xmi:id="_kiZUBAy6Ed-GTMgM16I5Vw" name="amount"
visibility="private" type="_kiZUBQy6Ed-GTMgM16I5Vw"/>
```

```
        </packagedElement>
        <packagedElement xmi:type="uml:PrimitiveType" xmi:id="_kiZUBQy6Ed-GTMgM16I5Vw"
name="String"/>
```

Transformuotos UML XMI schemos elementas importuotas į MagicDraw ir atvaizduotas grafiškai



6.4. Veiklos žodynų transformavimo eksperimento apibendrinimas

Vykdant eksperimentą buvo atliekamos šių SBVR konceptų transformacijos:

- Termino,
- Asociacijos,
- isPropertyOf fakto tipo.

Iš šių konceptų buvo sugeneruoti tokie elementai:

- UML klasė,
- Dvi klasės su asociacijos ryšiu,
- Klasė su atributu.

Eksperimentas patvirtino kad sukurtas įrankis leidžia transformuoti visus pagrindinius klasių diagramų elementus.

7. Išvados

1) Literatūros šaltinių ir egzistuojančių veiklos žodynų transformacijos analizės metu nustatyta, kad natūralia kalba modeliuoti veiklos taisykles gali tiek IT ekspertai, tiek verslo atstovai. Tačiau dalykinės srities ekspertui rašyti specifikaciją raštu yra lengviau, negu naudoti grafinę notaciją.

2) Įvertinus sistemos reikalavimus, metodui realizuoti buvo pasirinkta JAVA technologija ir Eclipse aplinka, kadangi tik šia kalba realizuota sistema gali dirbti su ALT transformacijų varikliuku ir taip pat ji lengvai komunikuoja su MagicDraw produktu kaip įskiepis, nereikalaujant kitos papildomos programinės įrangos.

3) Transformacijoms realizuoti buvo pritaikytas Cabot (2009) sudarytas SBVR metamodelis (susidedantis iš keturių metamodelių), kurio savybės buvo papildytos SBVR standarto atžvilgiu tam, kad jį būtų galima pritaikyti realizacijai. Cabot SBVR metamodeliai buvo sujungti ir papildyti kai kuriais ryšiais, kadangi jų reikėjo pilnai išpildyti transformaciją; kai kurių Cabot SBVR metamodelio savybių buvo atsisakyta, siekiant išlaikyti jį kuo artimesnį oficialiam standartui.

4) Realizacijos metu sudarytas SBVR metamodelis leidžia aprašyti veiklos žodyną ir taisykles, juo remiantis galima daryti transformacijas ne tik į UML modelius, bet ir aprašinėti tam tikros veiklos ontologijas.

5) Sukurto metodo realizacijos testavimas patvirtino, kad programinė įranga veikia ir įgyvendina visus funkcinius reikalavimus. Taip pat transformuoja SBVR veiklos žodyno konceptus į UML klasių diagramas be klaidų.

6) Siūlomo metodo efektyvumui nustatyti, atliktas eksperimentas, kurio metu buvo vykdoma ribotos natūralios kalbos konceptų transformacija į UML klasių elementus. Nustatyta, kad taikant šį metodą aprašytų ir transformuotų į UML diagramą konceptų prasmė nepasikeičia ir transformuoja visus pagrindinius UML klasių diagramos elementus. Tokiu būdu yra išlaikomas veiklos žodynų ir taisyklių vientisumas.

7) VeTIS projekte sukurto Magic Draw įskiepio rezultatai parodė, kad veiklos žodynų ir taisyklių specifikavimo ribota natūralia kalba galimybių plėtra turi dideles perspektyvas. Atskirų veiklos sričių ribotos natūralios kalbos žodynai leidžia sukurti kompiuteriams ir žmonėms suprantamas sąsajas tarp programų ir realiame pasaulyje vartojamos žmonių kalbos.

8) Magistro darbo tematika buvo paskelbti keturi straipsniai ir perskaityti pranešimai konferencijose.

8. Literatūra

[1] Ablonskis, L. (2007). A suggestion for improvement of program code generators // In J. Nummenmaa, E. Soderstrom (Eds.), Proc. Of the 6th International Conference on Perspectives in Business Information Research – BIR'2007, Tampere, Finland, September 2007.

[2] AI Wiki [žiūrėta 2008 11 10]. Prieiga per internetą: https://ai.ia.agh.edu.pl/wiki/pl:miw:miw08_bizrulesvocabularies

[3] ATL Documentation. [žiūrėta 2009-04-20]. Prieiga per internetą: http://www.eclipse.org/m2m/atl/doc/

[4] Avdejenkov, V., Vasilecas, O. (2006). Priemonių veiklos taisyklėms(VT) informacinėse sistemose realizuoti analizė. Informacinės Technologijos 2006, Kaunas, Technologija.

[5] Bartkus, A., Bisikirskas, J. (2007). Veiklos taisyklių įskiepis CASE įrankiui "MagicDraw UML". Bakalauro darbas. KTU.

[6] Business Rules Team. (2004). Business Semantics of Business Rules (version 2.1). Initial Submission to BEI RFP br/2003-06-03. [Interaktyvus], [žiūrėta 2008-10-20]. Prieiga per internetą: www.omg.org/cgi-bin/doc?bei/04-01-04.

[7] Cabot J., Pau R., Ravento R. From UML/OCL to SBVR specifications: A challenging transformation, 2008

[8] Cabot J., Pau R., Ravento R. UML-to-SBVR and SBVR-to-HTML transformations. Prieiga internete: www.lsi.upc.edu/~jcabsearch/SBVR.

[9] Fair Isaac-PRR. (2007). Production Rule Representation. Fair Isaac Corporation, ILOG SA, Submission to Production Rule Representation. Ver. 1.03, 2007, [žiūrėta 2008-10-20] Prieiga per internetą: www.omg.org.

[10] Gudas S., Skersys T. Lopata A. Framework for knowledge–based IS engineering. In: Proceedings of third international conference „Advances in Information systems (ADVIS2004)", Izmir, Turkey, October 20-22, 2004, LNCS 3261, 2004, 512 – 522.

[11] Kapocius K., Butleris R. Repository for business rules based IS requirements. Informatica, 2006, Vol. 17, Nr. 4, 503–518.

[12] Kleiner, M., Albert, P., Bézivin, J. Parsing SBVR-Based Controlled Languages. In Model Driven Engineering Languages and Systems, LNCS, Vol. 5795, 2009, 122–136.

[13] Lebedys, E., Vasilecas, O. (2004). Analysis of business rules modelling languages. In Proc. of the IT'2004, Lithuania, Technologija.

[14] Linehan, M. H. Semantics in Model-Driven Business Design. In Proc. 2nd Int. Semantic Web Policy Workshop (SWPW'06), 2006.

[15] Nemuraitė L., Skersys T., Šukys A., Šinkevičius E., Ablonskis L. VETIS tool for editing and transforming SBVR business vocabularies and business rules into UML&OCL models. // Information Technologies' 2010 : proceedings of the 16th international conference on Information and Software Technologies, IT 2010, Kaunas, Lithuania, April 21-23, 2010 / Edited by A. Targamadze, R. Butleris, R. Butkiene Kaunas University of Technology. Kaunas : Technologija. ISSN 2029-0020. 2010, p. 377-384., 2010

[16] Raj A., Prabhakar T. V., Hendryx S. Transformation of SBVR business design to UML models. In: ISEC '08: Proceedings of the 1st conference on India software engineering conference, ACM, Hyderabad, India, 2008.

[17] QVT Specification version 1.0. [žiūrėta 2009-04-20]. Prieiga per internetą: http://www.omg.org/spec/QVT/1.0/

[18] SBVR an Eclipse based plugin to create fact-oriented business models and rules [žiūrėta 2008 11 08]. Prieiga per internetą: http://sbeaver.sourceforge.net.

[19] Schacher M. Business Rules from an SBVR and an xUML Perspective (Parts 1–3). Veiklos taisyklių žurnalas, 2006, vol. 7, Nr. 6–8.

[20] Semantics of Business Vocabulary and Business Rules (SBVR), v1.0. OMG formali specifikacija/2008-01-02, 2008.

[21] OMG Unified Modeling Language, Superstructure, V2.1.2. OMG formali specifikacija/2007-11-02, 2007.

[22] Šinkevičius E., Butleris R. Transformacija iš UML į SBVR naudojant Atlas kalbą Eclipse aplinkoje. // 14-osios Tarpuniversitetinės magistrantų ir doktorantų mokslinės konferencijos "Informacinės technologijos" pranešimų medžiaga / Vilniaus universitetas, Vytauto Didžiojo universitetas, Kauno technologijos universitetas. [Kaunas] : [Vilniaus universiteto Kauno Humanitarinis fakultetas]. ISSN 2029-249X., p. 187-190., 2009

[23] Šinkevičius E., Tutkutė L. SBVR metamodelio sudarymas ir panaudojimas veiklos žodynų transformacijoms į UML taikant ATLAS transformavimo kalbą // Informacinė visuomenė ir universitetinės studijos (IVUS) 2010: konferencijos pranešimo medžiaga / Vilniaus universitetas, Kauno technologijos universitetas, Vytauto Didžiojo universitetas., 2010

[24] Tutkutė L., Ručinskaitė Agnė Veiklos taisyklių, grindžiamų šablonais, valdymas informacinėse sistemose. // 14-osios Tarpuniversitetinės magistrantų ir doktorantų mokslinės konferencijos "Informacinės technologijos" pranešimų medžiaga / Vilniaus universitetas, Vytauto Didžiojo universitetas, Kauno technologijos universitetas. [Kaunas] : [Vilniaus universiteto Kauno Humanitarinis fakultetas]. ISSN 2029-249X, p. 177-181., 2009

[25] Tutkutė L., Šinkevičius E. Veiklos procesų modeliavimo praplėtimas veiklos taisyklėmis // Informacinė visuomenė ir universitetinės studijos (IVUS) 2010: konferencijos pranešimo medžiaga / Vilniaus universitetas, Kauno technologijos universitetas, Vytauto Didžiojo universitetas., 2010

[26] W3C Recommendations for XSL Transformations (XSLT) version 1.0. [žiūrėta 2009-04-20]. Prieiga per internetą: http://www.w3.org/tr/xslt/

Terminų ir santrumpų žodynėlis

ATL – yra Eclipse.org Model-To-Model (M2M) projekto dalis. Tai yra modelių transformavimo kalba apibrėžta metamodeliais ir tekstinę sintakse (angl. *Atlas transformation language*).

CASE – automatizuotas programų kūrimas (kūrimo sistema) (angl. *Computer Aided Software Engineering*)

IS – informacinė sistema

MDA – modeliais grindžiama architektūra, OMG plėtojamas programų sistemų kūrimo būdas ir standartų aibė (angl. *Model Driven Architecture*)

MDD – modeliais grindžiamas sistemų kūrimas, bendras šiuolaikinis požiūris į programų sistemų kūrimą (angl. *Model Driven Development*)

OCL – formali išraiškų kalba, UML standarto dalis (angl. *Object Constraint Language*)

OMG – mokslinis konsorciumas, kurio pradinis tikslas buvo objektinių metodų standartizavimas, tačiau vėliau jo veikla išplito ir į kitas kryptis (angl. *Object Management Group*)

OSM – organizacinės struktūros metamodelis, vienas iš OMG standartų veiklos procesų modeliavimui (angl. *Organization Structure Metamodel*)

SBVR – veiklos žodyno ir veiklos taisyklių semantikos apibrėžimas, duodantis abstrakčius formalius pagrindus veiklos taisyklių modeliavimui (angl. *Semantics of Business Vocabulary and Business Rules*)

UML – unifikuota modeliavimo kalba programų sistemų modeliavimui, konstravimui ir dokumentavimui; turi išplėtimo mechanizmus ir pritaikyta daugeliui sričių, tame tarpe veiklos modeliavimui (angl. *Unified Modeling Language*)

VT – veiklos taisyklė

XMI – XML schema apsikeitimui metaduomenimis, naudojama UML CASE įrankiuose UML modelių metaduomenų aprašymui (angl. *XML Metadata Interchange*)

XML - bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba (angl. *eXtensible Markup Language*)

9. Priedai

1 priedas. Mokslinis straipsnis tarptautinėje konferencijoje „Information Technologies‘2010“

VETIS TOOL FOR EDITING AND TRANSFORMING SBVR BUSINESS VOCABULARIES AND BUSINESS RULES INTO UML&OCL MODELS

Lina Nemuraite[1], Tomas Skersys[1], Algirdas Sukys[1], Edvinas Sinkevicius[1], Linas Ablonskis[1]

[1]Kaunas University of Technology, Department of Information Systems, Studentu 50-313a, Kaunas, Lithuania, lina.nemuraite@ktu.lt

Abstract. The OMG SBVR standard is the most matured abstract representation for Business semantics; however, the complexity of SBVR metamodel prevents it of a broad and rapid usage in business communities. There are not a lot of SBVR implementations yet, but the popularity of SBVR is growing as different interest-groups are finding the variety of ways for applying the SBVR standard. The goal of the paper is to present the VeTIS tool capable for editing SBVR Business Vocabularies and Business Rules and for transforming them into UML class models supplemented with OCL constraints. It is integrated into MagicDraw UML CASE tool and organically combines with simple development process, which pursues defining requirements via Use Cases and modelling business processes via Activity diagrams.

Keywords. SBVR, UML, OCL, Business Vocabulary, Business Rules, modelling, transformation, development process.

1 Introduction

The vision of OMG “Semantics of Business Vocabulary and Business Rules” (SBVR) standard [20] is expressing business knowledge in a controlled natural language unambiguously understandable by human and computer systems. Such knowledge is captured by business experts or information system analysts who need tools that would allow storing SBVR specifications in metamodel repositories (e.g. based on MOF[1] or EMF[2]) for interchanging and linking them with other models (e.g. UML&OCL models). SBVR is fully integrated into the OMG’s Model-Driven Architecture (MDA) [4].

The precise meaning of SBVR vocabularies allows transforming them to software models of information systems without violating business semantics. SBVR vocabularies and business rules are formulated in logical formulations that are based on Common Logic standard [7] but also they appreciate the grammar of the natural language. Currently, the SBVR structured English is the concrete language for SBVR metamodel though other languages are possible. The peculiarity of SBVR metamodel is the explicit separation of meaning, representation and symbolization: the same meaning can have many representations and the same signifier or expression can represent different meanings. For this reason, conceptual models presented in other representation languages and having the same interpretation will have many-to-one relationships with the corresponding SBVR model. SBVR semantic formulations have formal interpretation and are able to express complex definitions and statements that are related with SBVR concepts and fact types. As concepts and fact types are basic elements of both natural and software modelling languages, the foundations of SBVR are able to serve as means for better alignment of business and information technologies.The another great worth of SBVR is capability for reflecting different contexts – each SBVR Vocabulary represents meaning shared in some Semantic Community. As the shared meaning can be represented in multiple languages, the Speech Communities can exist inside the Semantic Communities. In the future, there are perspectives for having a rich multilingual notation for representing the shared semantics defined by SBVR. The complexity of SBVR metamodel prevents it of a broad and rapid usage in practise. There are not a lot of SBVR implementations yet. The first version of SBVR standard was completed in 2008, but preliminary versions existed some years before. Currently, the popularity in SBVR is growing as different interest groups are finding the variety of ways for applying the SBVR standard in Semantic Web and for engineering software. As there still is a lack of SBVR tools, we are offering the VeTIS[3],[4] – SBVR editor integrated into MagicDraw UML CASE tool capable for generating UML class diagrams with OCL constraints from SBVR vocabularies and rules.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 is devoted to editing SBVR vocabularies and rules, section 4 – for transforming them into UML&OCL models. Section 5 presents the SBVR based development process. Section 6 draws conclusions and highlights the future work.

---

[1] Meta Object Facility is the OMG standard – foundation of OMG Model-driven Architecture, both language and environment for generating application code from models

[2] Eclipse Metamodeling Framework, the modeling framework and code generation facility similar to MOF

[3] http://www.magicdraw.com/main.php?ts=navig&cmd_show=1&menu=plugins

[4] http://www.magicdraw.com/main.php?ts=navig&cmd_show=1&menu=online_demo&#plugins

2 Related Work

The goal of SBVR standard is specifying business semantics without any considerations about its implementation. It can serve for many purposes, for example, ensuring semantic interoperability between different Semantic and Speech Communities or distributed information systems on the Web [22]; or engineering the software; often it is related with service oriented systems [9][14]. The approaches of business rule automation are endeavouring to build software systems directly from vocabularies and rules. In the context of Model-driven Architecture, the straightforward way for doing this is to transform SBVR into UML&OCL models. The early proposals for such transformations are limited [12][17]. The most sophisticated matching between SBVR and UML&OCL is the reverse transformation between UML schemas and SBVR vocabularies and rules in order to verbalize and validate UML schemas [1]. However, it also has many drawbacks, for example, it does not considers UML operations and conceals the actual difficulties of verbalizing conceptual languages as UML. Another important development concerns SBVR transformation to UML with advanced parser for SBVR structured language [10].

The [6] research is related with transforming SBVR specifications into Web Ontology Language OWL and Semantic Web rules R2ML. It is worth to mention that whereas the correspondence between OWL and SBVR is declared in SBVR specification, currently there are no established tools for transforming SBVR vocabularies and rules to ontologies as well as all SBVR transformations to UML&OCL are incomplete.

The Model Driven Enterprise Engineering (MDEE) methodology created by KnowGravity is one of the first efforts to apply OMG SBVR in the holistic IS development process [21]. MDEE supports the smooth going from SBVR structural and operative rules to PIM Constraints, ECA and CA (Condition-Action) rules. [11] integrates SBVR rules into IBM Model-driven Business Transformation process integrating interaction among rules, processes, and ontologies. In [13][14] , SBVR metamodel is used as a modeling language for building the so called *generative information systems*, built on the declarative technologies. The architecture proposed by Marinos and Krause is based on REST[5] architectural style that focuses on resources identified by names, a fixed number of methods with known semantics to manipulate these resources and stateless interactions between client and supplier. As SBVR does not explicitly specify business processes, the resulting architecture is capable for implementing atomic operations on resources which are necessary but not sufficient for the operation of an information system. For this purpose authors extend SBVR specification for declarative representation of business processes and compose them with business rules [18].

The prerequisites of SBVR implementation presented in the current paper were described in [15][2], [3]. In [15], the possibilities of representing different types of business rules in UML models supplemented with OCL expressions were identified. The paper [2] outlined the methodological points for modelling business processes and business rules: how to separate and relate these concerns for obtaining simple and manageable software systems. The [3] presents the SBVR based software development process devoted for service-oriented information systems. It should be noted that our current implementation is more universal and simpler. We have endeavoured to keep to the SBVR standard as close as possible and have made only those extensions that were necessary for implementation. Therefore, we propose the VeTIS tool capable for editing SBVR business vocabularies and business rules, and for transforming them into UML class model with OCL constraints. It is integrated into MagicDraw UML CASE tool and organically combines with the incident development process with defining requirements via Use Cases and modelling business processes via Activity diagrams.

3 Editing SBVR Business Vocabularies and Business rules in the VeTIS tool

The VeTIS editor working as MagicDraw UML plug-in was implemented on the Eclipse 3.4.1 platform on the basis of SBVR 1.0 metamodel, whereas the user interface was adapted from SBeaVeR tool [5], [19] that was extended for generating SBVR 1.0 XMI schema, specifying rule sets, validating them etc. Also, we have used Cabot's [1] EMF SBVR XMI schema but extended it in some points and merged the five metamodels into a whole. There are five basic SBVR concept types that can be presented in a Business vocabulary using VeTIS tool: *object types* (or general concepts), *individual concepts*, *fact types* (or verb concepts), and *roles*. All these concept types are represented in different styles allowing recognizing them from the entered text: the "term" font is devoted for object types and roles, the "*verb*" font renders fact types, the "Name" – individual concepts (slightly differently from original SBVR style), and "each" font is for keywords; individual concepts start with the capital letter for distinguishing them from object types. Also, it is possible to explicitly specify the desirable concept types in particular cases when it is required (e.g. for fact type roles, categorization schemes etc) or if you want to change the meaning of a predefined verb. The Business Vocabulary and Business Rule views are presented in the Fig. 1.

Following the SBVR, fact types are defined using the existing noun concepts (object types or individuals), which have already been defined in the business vocabulary. *Verbs* (or fact symbols) represent fact

[5] REST is a competing paradigm for Web Service technologies as the latter suffer from theirs complexity and lack of adoption for distributed applications on the World Wide Web

types creating relationships between these nouns or specifying their characteristics, for example, "bank *gives* loan" presents an *active sentential form* of the fact type. Also, fact types can have *passive* sentential forms (e.g. "loan *is_given by* bank") or *noun forms* (e.g. "loan *of_the* bank"). All these forms are *synonymous forms* of one and the same fact type (similarly, noun concepts can have *synonyms*), while the verb in the expression "bank *gives* loan" is accepted for the *primary designation* of that fact type. Normally, the primary designation of a concept corresponds to its *preferred designation*. In reverse cases, the description of a concept includes a reference to the preferred representation under the caption "See". Mostly, synonymous forms of fact types are used in business rules statements. Only primary designations of the meaning are assessed during transformation of SBVR specifications into UML models.

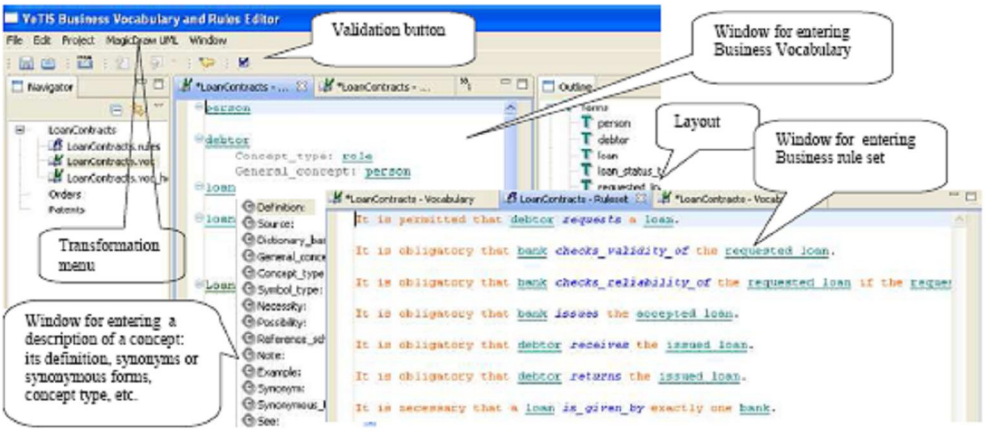


Fig. 1. Business Vocabulary and Business Rule views in the VeTIS tool

Presented examples are of the most common *associative fact type* that has two or more roles involved. A *role* is a noun concept that corresponds to things based on their playing a part, assuming a function or being used in some particular situation, for example, in the fact type "debtor *requests* loan" the "debtor" is the role played by a person. A role is identified by the vocabulary entry, which has the field "Concept_type:" set to "role", and the "General_concept:" field set to the object type, which plays that role in the corresponding fact type. Role is represented by a *placeholder* that indicates a place for expression of what fills a role in the fact type form.

VeTIS supports only two roles for associative fact types. Though SBVR allows fact types with many roles, interpretation of n-ary fact types is more complicated; such concepts are not well supported in other languages as OWL or tools as MagicDraw UML. Besides, it is always possible to represent fact type with several roles by several fact types with two roles. Usually, there are two noun concepts playing specific roles in the relationship defined by the associative fact type. Only in the case of a reflexive relationship, there is one noun concept playing both roles. A *partitive* fact type is the binary fact type stating that one noun concept (all of its instances) is in the composition of a given whole, i.e. another noun concept. Partitive fact types are identified by the verb phrases "*includes*" and "*is_included_in*" (for active and passive forms respectively).

Is_property_of fact types define the essential qualities of a given noun concepts. They are identified by the verb phrases "*has*" and "*is_property_of*" (for active and passive forms respectively), e.g. "loan *has* amount" or "amount *is_property_of* loan". It is unnecessary to define "Concept type: role" for roles that are used in *is_property_of* fact types. Roles of *is_property_of* fact types have predefined *elementary object types* text, integer and number as their general concepts, e.g. role "amount" will have the "number" in the "General_concept:" field. *Characteristic* is a fact type having only one role, e.g. "loan *is_returned*".

Categorization fact type represents relationship between the more general noun concept and more specific noun concept, which is a category of the first concept. VeTIS tool allows specifying simple categorization fact types, e.g. "accepted_loan *is_a* loan" by using the following pairs of verbs and verb phrases: "*specializes*" and "*generalizes*", "*is_category_of*" and "*is_of_category*", "*is_a*" and "*is_a*" (VeTIS interprets *is_a* as a relationship between a specific concept and the general concept, and creates the synonymous form with the verb phrase *is_a* between the general concept and the specific concept by default). Simple categorization fact types can also be represented in noun concept entries with additionally specified "General_concept:" fields.

More complex structures involving *categorization types* and *categorization schemes* are also supported by VeTIS. Categorization scheme is a set of categories that subdivides instances of a general concept into subsets specialized by some feature (categorization type) (Fig. 2) (such a structure is mandatory, if you want to

specify the complete and correct information about a categorization scheme). Similarly, *segmentation* is a categorization scheme whose contained categories comprise complete set of categories (with respect to the general concept), and sets of objects belonging to these categories are disjoint.

```
requested loan
    General_concept: loan
loan status type
    Concept_type: categorization type
    Necessity: is_for general concept requested loan
Loans by loan status type
    Necessity: categorization scheme for general concept requested loan that subdivides requested loan by loan status type
accepted loan
    General_concept: requested loan
    Necessity: is_included_in Loans by loan status type
rejected loan
    General_concept: requested loan
    Necessity: is_included_in Loans by loan status type
```

Fig. 2. An example of the categorization scheme

According SBVR, *Business rules* are rules under business jurisdiction. Business rules are constructed as *closed modal formulations* that have recursively embedded logical formulations and are based on fact types. Modal formulations may be *alethic* modal formulations, i.e. necessities, possibilities or impossibilities used in Structural Business Rule statements expressing Structural Business Rules, or *deontic* modal formulations, i.e. obligations, permissions or prohibitions used in Operative Business Rule statements expressing Operative Business Rules. *Structural Business Rules* are rules about how the business chooses to organize (i.e., "structure") the things it deals with. Structural Rules are constraints and derivations, and supplement definitions. *Operative Business Rules* are rules that govern the conduct of business activity (dynamic or action rules). In contrast to Structural Rules, Operative Rules can be *directly* violated by people involved in the affairs of the business. VeTIS supports four types of business rules (you can define impossibilities and prohibitions using remaining types of rules):

Necessities: "It is necessary that a loan *owns* exactly one bail."

Possibilities: "It is possible that a debtor *gets* at most 3 loan."

Permissions: "It is permitted that a debtor *requests* a loan."

Obligations: "It is obligatory that a bank *gives* a loan if the loan *is_a* valid loan and the loan *is_a* reliable loan.

4 Transforming SBVR specifications into UML&OCL models

One of the core features of the VeTIS tool is the transformation of the SBVR specification (i.e. Business Vocabulary and Business Rules) in SBVR Structured English into SBVR 1.0 XMI format and later – into the UML class model with OCL constraints (in EMF UML 2.1.2 XMI). For that purpose ATL transformation language [8] and ATL transformation engine 3.0.0 were used. In the last step UML model is visualized in the MagicDraw UML tool using API. The transformation process is presented in Fig. 3.

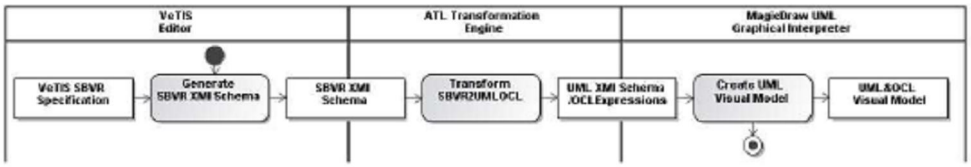


Fig. 3. SBVR2UML&OCL transformation steps

4.1 Transforming Business Vocabularies

Business Vocabulary (along with Business Rules) is transformed into a **UML package**, while the vocabulary name is transformed into the package name. Two elements are included in that newly created package: "VeTIS" profile that supports visualization of constraints in UML class diagrams, and "Constraints" package for holding OCL constraints obtained from SBVR. "VeTIS" profile includes single stereotype `<<constrained>>`, base metaclass of which is "`Element`", so it is applicable for classes, operations and other UML elements.

Object type (represented by term) is transformed into a **UML class** holding the same name as the object type. **Primitive object types** as numbers, texts, integers etc are transformed into **UML data types**. **Role** (represented by a placeholder) is transformed into UML property having the same name as its placeholder and

the type of the property is set to a class obtained from the object type playing the corresponding role in that fact type.

Associative fact type is transformed into an **association relationship** between two classes. The names of properties corresponding to the ends of that association are obtained from names of roles of that fact type. A verb (or a verb phrase) used by the fact type becomes a name of that association. If there are no business rules constraining the number of occurrences of instances of a fact type, multiplicity bounds of the association are set to "0..*" by default. **Is_property_of fact type** is transformed into a property presenting an **attribute** of a class similarly as the associative fact type. Roles in SBVR are ordered, so a class that owns that attribute is obtained from object type playing the second role in the corresponding is_property_of fact type. The type of an attribute is set to a data type obtained from the "General_concept:" that in this case specifies a primitive object type.

Characteristic (fact type having only one role) is transformed into an **attribute of the Boolean type**. **Partitive fact type** is transformed into a **composition relationship** between two classes. **Categorization fact type** is transformed into a **generalization relationship** between two classes. **Categorization scheme** is transformed to a **generalization set** and **categorization type** is transformed to a class, which is set as a **powertype** of that generalization set. The name of categorization scheme becomes the name of the generalization set and the generalization set is referenced in generalization relationships obtained from categorization fact types included in that categorization scheme. **Segmentation** is transformed analogously to a categorization scheme except the difference that generalization relationships are generated with the {complete, disjoint} constraints instead of the {incomplete, disjoint} constraints used in the case of the categorization scheme.

4.2 Transforming Business Rules into UML

Some types of business rules are transformed into elements of UML class diagrams. **Structural Business rules** formulated by alethic modal formulations having directly embedded quantifiers (different from universal quantifier)[6] are transformed into multiplicity bounds of the corresponding associations or compositions. In VeTIS, it is possible to define the overall variety of multiplicity bounds specified by SBVR structural business rules.

Fig. 4. Transforming alethic modal formulations

Operative business rules (obligations and permissions) formulated as closed deontic formulations are transformed into **UML operations**. Verbs denoting fact types on which these formulations are based are transformed into operation names, roles – into parameter names, object types playing these roles – into parameter types. For example, the business rule: "It is permitted that a debtor *requests* a loan." is based on the fact type "debtor *requests* loan" and it is transformed into the following UML operation: „`Loan::requests(debtor:Person, loan:Loan):void`" (the operation of the class Loan in Fig. 8).

4.3 Transforming Business Rules into OCL

Structural Business rules formulated by alethic modal formulations that do not have directly embedded implications are transformed into **OCL class invariants expressing integrity constraints**. For example, structural business rule "It is necessary that amount *of_the* loan *is_not_greater_than* 10000." is transformed into class invariant `invLoan1` (see the corresponding constraints in Fig. 8):

```
context Loan inv invLoan1: self.amount<=1000
```

All constraints are created in the package "`Constraints`". Constraint names are unique. Stereotype `<<constrained>>` marks classes or other elements having related OCL constraints. Constraints may be visualized in notes by clicking the corresponding button on the right-click class or operation menu.

Structural business rules (necessities), formulated as alethic formulations with directly embedded implications, are transformed into **OCL class invariants expressing derivation rules.** It is necessary to describe derivation rules for all specialized concepts that cannot be categorized only by playing roles in fact types.

"It is necessary that the requested_loan *is_a* reliable_loan if each issued_loan *that_is_of_the* debtor *that_receives_the* issued_loan *is_returned*." is transformed to:

```
context RequestedLoan
   inv invRequestedLoan3: self.oclIsTypeOf(ReliableLoan) implies
```

[6] We will use „directly embedded" for the sake of simplicity in the cases like this. Factually, these "quantifiers different from universal quantifier" are embedded into a universal quantifier that is embedded into a corresponding modal formulation

```
self.debtor.issuedLoan→forAll(it1|it1.isReturned=true)
```

Operative business rules (obligations and permissions), formulated as deontic formulations with directly embedded implications, are transformed into **operations and operation preconditions**. Implication consequent is transformed into operation, antecedent – into precondition:

"It is obligatory that bank *checks_reliability_of* the requested loan if the requested loan *is_a* valid_loan." Is transformed to

```
Context RequestedLoan::checksReliabilityOf(bank:Bank,requestedLoan:RequestedLoan):
   OclVoid pre preChecksReliabilityOf1: self.oclIsTypeOf (ValidLoan)
```

There are further possibilities to automatically obtain attribute default values, operation results and post conditions from SBVR definitions. Though mathematical calculations are difficult to express in SBVR, the standard is extensible. Therefore, such improvements depend on the willingness of developers.

5 Development process based on SBVR

The process of defining business vocabularies and business rules is not easy. It is difficult to formulate consistent and complete sets of concepts and propositions governing business. SBVR specifications are of declarative nature and they define just business constraints – constraints on structure and on activities, but not on control flows of these activities, i.e. business processes. Though it is possible to predefine sequences of activities by declarative constraints or even extend SBVR for specifying business processes, such a practice is not recommended by business rule methodologists. In contrast, they propose "separation of concerns" – keeping models of business processes and specifications of business rules separately, not intertwining them, because they are changing independently [2]. Also, visual modelling is better suited for definition of business processes. Additionally, modelling of business processes helps define right and consistent business rules as well. Let's take a look at how all this integrates into the software development process (Fig. 5.).

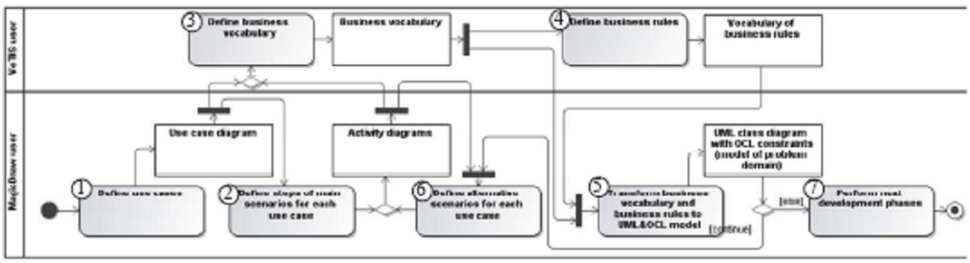


Fig. 5. SBVR based development process

In the presented process, we recommend writing software requirements (i.e. use case specifications) in alignment with defining business processes (e.g. in the form of BPMN or UML activity diagrams), business vocabularies and business rules. We will demonstrate the process with the simplified example of a Bank Information system (Fig. 6). In the first step „Define use cases" (Fig. 5.) we choose the use cases of the system and in the second step we define straightforward processes representing steps of main success scenarios. We draw a UML activity diagram for each use case and represent our desired business processes (Fig. 6).

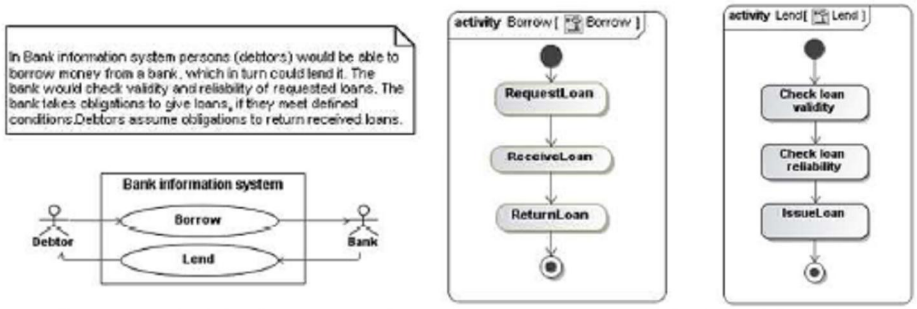


Fig. 6. Uses cases and activity diagrams for main success scenarios for the example of Bank information system

In the 3rd and 4th steps we define business vocabulary and business rules. Now we can try to transform the vocabulary and rules to UML&OCL model though the model would not be complete. In the 6th step "Define alternative scenarios" we consider how the process will look in the cases when requested loan is not valid or unreliable. Refined activities representing use cases include specific concept categories corresponding to different states of object types: requested loan, issued loan, rejected loan, etc. We supplement business

vocabulary by adding these categories and specify categorization schemes if needed. Then we can augment the business vocabulary with fact types corresponding to activities of alternative scenarios (Fig. 7).

Now we can add or refine operative business rules regarding alternative scenarios providing when needed the corresponding synonymous forms. Also, we describe additional structural rules for derivation of categorized object types and all required constraints. After the next export to MagicDraw UML tool, we obtain the class diagram as shown in Fig. 8. Really, the process would not be so straightforward. We would make many refinements and iterations till moving to the next phase(s) of Model-driven software development.

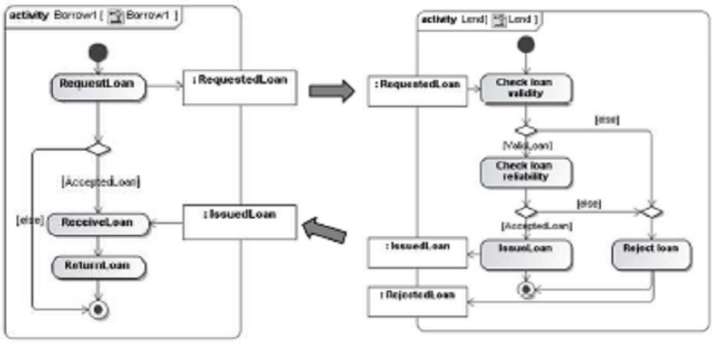


Fig. 7. Activity diagrams representing alternative scenarios of Bank information system

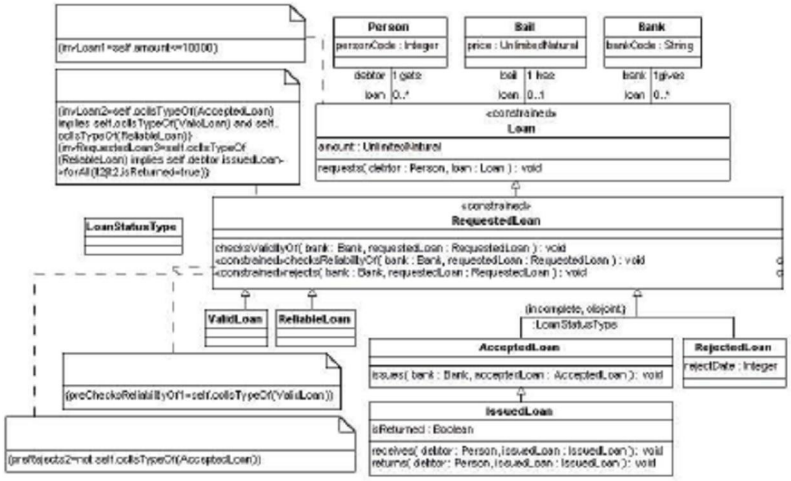


Fig. 8. UML class diagram with OCL constraints obtained after transformation

6 Conclusions and Future Work

The implementation of VeTIS has shown the possibility of transforming SBVR vocabularies and rules into non-trivial conceptual models. We have assured that UML language is capable for representing all concepts of SBVR vocabularies (we have explained earlier why we have abandoned fact types with many fact type roles, though their representation in UML is feasible) and some kinds of business rules: alethic formulations having directly embedded quantifiers expressing multiplicity bounds, and simple deontic rules (having no directly embedded implications) expressing unconstrained UML operations. However, our transformations from SBVR rules into OCL are yet incomplete for the following reasons:

– It is difficult to recognize the right formulations of complex rules in SBVR structured language entered using VeTIS editor interface. For generating SBVR XMI schema for complex rules, we are now considering several possible approaches: creating the advanced parser, which could present several possible variants of meaning for confirming to user; proposing rule wizards; implementing visual editor for rules, etc.

– SBVR metamodel is not fully comprehensive with regards to specifying rules: it does not allow to represent simple computations, iterations, if-then-else constructs etc. The extensibility of SBVR makes it

possible to supplement it, however, it is difficult to anticipate encompassing all possibilities of OCL. From the other side, OCL is not capable to expressing SBVR modalities; however, modalities are not very important for implementing software.

Our future work is directed to enhancing the interface for entering complex rules, and to implementing multilingual vocabulary (first including the Lithuanian language). For this purpose, SBVR terms, names and other designations should be extended with attributes for identifying the corresponding language, and with synonymous forms for representing cases. Furthermore, there are a lot of problems to research for managing large vocabularies, reusing them, enhancing transformations etc.

References

[1] **Cabot, J., Pau, R., Raventos, R.** From UML /OCL to SBVR specifications: A challenging transformation. *Information Systems*, 2008, 1–24.

[2] **Ceponiene, L., Nemuraite, L., Vedrickas, G.** Separation of event and constraint rules in UML&OCL models of service oriented information systems. *Information technology and control, Vol. 38(1)*, 2009, 29–37.

[3] **Ceponiene, L., Nemuraite, L., Vedrickas, G..** Semantic business rules in service oriented development of information systems. *In Information Technologies' 2009 : proceedings of the 15th International Conference on Information and Software Technologies, IT 2009, Kaunas, Lithuania, April 23-24, 2009, Kaunas University of Technology, Kaunas, Technologija*, 2009, 404–416.

[4] **Ceravolo, P., Fugazza, C., Leida, M.** Modeling Semantics of Business Rules. *2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007)*, 2007, 171–176.

[5] **De Tommasi, M., Corallo, A.** SBEAVER: A Tool for Modeling Business Vocabularies and Business Rules. *In Proc. 10th Int. Conf. on Knowledge-Based Intelligent Information and Engineering Systems (KES'06), LNCS Vol. 4253*, 2006, 1083–1091.

[6] **Demuth, B., Liebau, H. B.** An Approach for Bridging the Gap Between Business Rules and the Semantic Web. *Advances in Rule Interchange and Applications, LNCS, Vol. 4824*, 2009, 119–133.

[7] Information technology - Common Logic (CL): a framework for a family of logic-based languages. *ISO/IEC 24707*, 2007.

[8] **Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.** ATL: a QVT-like transformation language. *In Proc. OOPSLA Companion*, 2006, 719–720.

[9] **Kamada, A., Mendes, M.** Business Rules in a Service Development and Execution Environment. *Eleventh International IEEE EDOC Conference Workshop (EDOCW'07)*, 2007, 1366–1371.

[10] **Kleiner, M., Albert, P., Bézivin, J.** Parsing SBVR-Based Controlled Languages. *In Model Driven Engineering Languages and Systems, LNCS, Vol. 5795*, 2009, 122–136.

[11] **Linehan, M. H.** Ontologies and Rules in Business Models. *In Proc. 11th Int. EDOC Conference Workshop (EDOC '07)*, 2007, 149–156.

[12] **Linehan, M. H.** Semantics in Model-Driven Business Design. *In Proc. 2nd Int. Semantic Web Policy Workshop (SWPW'06)*, 2006.

[13] **Marinos, A., Krause, P.** An SBVR Framework for RESTful Web Applications. *In Rule Interchange and Applications, LNCS, Vol. 5858*, 2009, 144–158.

[14] **Marinos, A., Krause, P.** Using SBVR, REST and Relational Databases to develop Information Systems native to Digital Ecosystems. *In 3rd IEEE International Conference on Digital Ecosystems and Technologies*, 2009, 109–114.

[15] **Nemuraite, L., Ceponiene, L., Vedrickas, G.** Representation of business rules in UML&OCL models for developing information systems. *In Stirna, J., Persson, A. (Eds.). The Practice of Enterprice Modeling : First IFIP WG 8.1 Working Conference, PoEM 2008, Stockholm, Sweden, November 12-13, 2008 : proceedings, Lecture Notes in Business Information Processing, Vol. 15, Berlin, Heidelberg, New York : Springer*, 2008, 182–196.

[16] **Pau, R., Cabot, J.** Paraphrasing OCL Expressions with SBVR. *In: NLDB, LNCS, Vol. 5039*, 2008, 311–316.

[17] **Raj, A., Prabhakar, T. V., Hendryx, S.** Transformation of SBVR business design to UML models. *In: ISEC '08: Proceedings of the 1st conference on India software engineering conference, ACM, Hyderabad, India*, 2008, 29–38.

[18] **Razavi, A., Marinos, A., Moschoyiannis, S., Krause, P.** RESTful Transactions Supported by the Isolation Theorems. *In Web Engineering, LNCS, Vol. 5858*, 2009, 394–409.

[19] SBeaVeR – Business Modeller. *Eclipse plug-in, available at http://sbeaver.sourceforge.net*, 2006.

[20] Semantics of Business Vocabulary and Business Rules (SBVR). *Version 1.0. December, 2008, available at* http://www.omg.org/docs/formal/08-01-02.pdf.

[21] **Schacher, M.** Business Rules from an SBVR and an xUML Perspective (Parts 1–3). *Business Rules Journal*, 2006, Vol. 7(6–8).

[22] **Spreeuwenberg, S., Anderson, H. K.** SBVR's approach to controlled natural languages. *Workshop on Controlled Natural Language (CNL 2009)*, 2009.

2 priedas. Mokslinis straipsnis konferencijoje „Informacinė visuomenė ir universitetinės studijos (IVUS 2010)“

SBVR metamodelio sudarymas ir panaudojimas veiklos žodynų transformacijoms į UML taikant ATLAS transformavimo kalbą

Edvinas Šinkevičius
Informacijos sistemų katedra
Kauno technologijos universitetas
Kaunas, Lietuva
edvinas.sinkevicius@ktu.lt

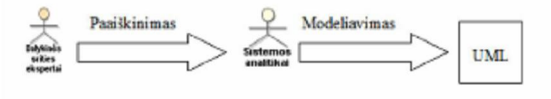
Lina Tutkutė
Informacijos sistemų katedra
Kauno technologijos universitetas
Kaunas, Lietuva
lina.tutkute@ktu.lt

Santrauka— UML kalba yra viena plačiausiai naudojamų modeliavimo kalbų specifikuojant informacines sistemas (IS). Tačiau ne visi dalykinės srities ekspertai (veiklos atstovai) UML modeliavimo kalbą supranta ir moka. Visą informaciją pateiktą klasių diagramomis galima užrašyti natūralia kalba. Siam tikslui galima panaudoti veiklos žodyno ir veiklos taisyklių semantiką (SBVR). OMG SBVR standartas labiausiai atitinkantis aprašas veiklos semantikai, kol kas jis nėra plačiai paplitęs, bet jo populiarumas vis didėja skirtingose panaudojimo grupėse. Žinoma sistemos analitikui unifikuota modeliavimo kalba yra daug priimtinesnė projektuojant IS, tačiau UML ir SBVR nėra susietos, jas reikia suderinti – transformuoti. Šiame straipsnyje yra apjungiamas SBVR metamodelis ir parodoma automatizuotą transformaciją iš SBVR į UML pasinaudojant sudarytu metamodeliu, kad modeliuotojai ir veiklos atstovai lengvai sąveikautu tarpusavyje remiantis suprojektuotu IS modeliu.

Raktiniai žodžiai— UML, SBVR, OCL, ATL, modelių transformacija, modeliavimas, veiklos žodynas, veiklos taisyklės.

I. Įvadas

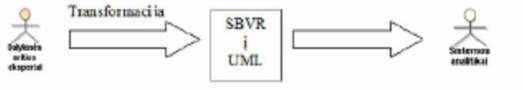
Sparčiai tobulėjant informacinėms technologijoms ir didėjant kuriamų sistemų sudėtingumui bei skaičiui, atsirado poreikis kuo daugiau automatizuoti IS kūrimo procesą. Automatizavimo problemai spręsti naudojami modeliais grindžiamo kūrimo (angl. Model Driven Development) metodai, kurių esmė – didesnis naudojamų abstrakcijų lygis, kai vietoj programavimo kalbų naudojamos modeliavimo kalbos (UML)[6]. Šiandien, unifikuota modeliavimo kalba (UML) yra viena plačiausiai naudojamų modeliavimo kalbų specifikuojant konceptualias schemas. UML schemos gali būti papildomos tekstinėmis išraiškomis, kurios aprašytu veiklos taisykles ir kurios negalėjo būti išreikštos grafiškai. Šios tekstinės išraiškos aprašomos objektinių reiškinių kalba (OCL).



Paveikslas 1. Dalykinės srities eksperto ir sistemos analitiko dabartinė sąveika

Išleistas veiklos žodyno ir veiklos taisyklių (SBVR) standartas 2008[9] turėjo didelį postūmį veiklos taisyklėms. Šis standartas apibrėžia metamodelį, semantinėms veiklos taisyklėms, veiklos faktams ir veiklos žodynui dokumentuoti. SBVR yra pilnai integruota į OMG Model-Driven Architecture (MDA) [3]. SBVR tikslas apjungti veiklos konceptus taip pat veiklos taisykles ir užrašyti juos paprasta žmogiškąja kalba, kad probleminės srities ekspertai(įmonės atstovai) galėtų lengvai jas suprasti. Probleminės srities ekspertui sunku patikrinti sistemų analitiko sudarytas specifikacijas dėl nesuprantamos formalios specifikavimo kalbos (pvz., UML). Iš kitos pusės, analitikui sudėtinga patikrinti probleminės srities eksperto pateiktos informacijos pilnumą ir teisingumą dėl šiam tikslui dažniausiai naudojamos natūraliosios kalbos ypatumų (natūraliaja kalba pateikti teiginiai gali būti daugiareikšmiai, prieštaringi, nestruktūrizuoti ir pan.). Padėti jiems susikalbėti gali veiklos taisyklių specifikavimo kalba, kuri iš vienos pusės išreiškiama natūraliajai kalbai artimais sakiniais, iš kitos pusės, yra formali (tiksli ir vienareikšmiška), kad būtų lengvesnis informacijos apsikeitimas tarp organizacijų ir programinių įrankių. Veiklos taisyklės aprašytos remiantis SBVR standartu gali būti išreiškiamos daugelyje kalbų. Pirmoji SBVR standarto versija atsirado 2008, bet preliminarios versijos jau buvo sukurtos keleriais metais anksčiau. Dabar SBVR populiarumas ir susidomėjimas didėja skirtingose grupėse, bandoma rasti įvairių būdų pritaikant SBVR standartą semantiniuose tinkluose ir programinėse įrangose. Kol kas yra nedaug SBVR įrankių, vienas iš jų yra VeTIS[1,2] - SBVR redaktorius integruotas į MagicDraw programą, gebantis generuoti UML klasių diagramas su OCL apribojimais iš SBVR žodyno ir taisyklių. Yra ir daugiau įrankių vykdančių panašias transformacijas, bet daugelis iš jų yra pilnai neišpildyti [7].

Žinoma, unifikuota modeliavimo kalba (UML) ir SBVR nėra susietos, todėl natūralią kalbą reikia transformuoti į UML, kad sistemos analitikas lengvai galėtų suprasti probleminės srities eksperto, remiantis SBVR standartu, sudaryta modelį [10]. Todėl pagrindinė šio straipsnio mintis – aprašyti automatinės transformacijos iš SBVR į UML idėją (kuri aiškiau paaiškinta „Paveikslas 2.“).



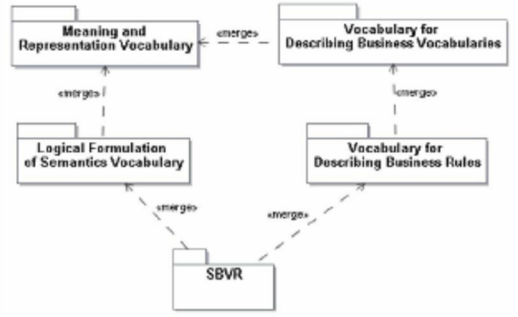
Paveikslas 2. Rezultatas pasiektas panaudojant SBVR transformacija į UML

Kaip matoma („Paveikslas 1."), jei reikėtų sumodeliuoti IS tuomet sistemos analitikui privalu atlikti jos analizę, konsultuojantis su užsakovu, vėliau pagal surinktus reikalavimus modeliuojama IS. Šio ilgo ir varginančio proceso galima išvengti leidžiant pačiam veiklos atstovui modeliuoti IS, kompiuterio pagalba įvedant atitinkamas reikšmes natūralia kalba.

II. SBVR METAMODELIO SUDARYMAS

Norint pritaikyti SBVR metamodelį siekiamam rezultatui, iškilo problemų dėl to, kad SBVR standarte pateikiamas metamodelis aprašytas atskirais fragmentais ir nėra tinkamas veiklos žodynų ir taisyklių redaktoriams realizuoti. Analizės metu buvo pritaikyti keturi metamodeliai, kuriuos pagal SBVR sudarė ir savo UML2SBVR įrankyje taikė Cabot [2], [1] („Paveikslas 3."):

- MRV (angl. Meaning and Representation Vocabulary);
- LFSV (angl. Logical Formulation of Semantics Vocabulary);
- VDBV (angl. Vocabulary for Describing Business Vocabularies);
- VDBR (angl. Vocabulary for Describing Business Rules).



Paveikslas 3. SBVR metamodelio dalių ryšiai

MRV aprašo pagrindinių veiklos konceptų prasmę, vaizdavimą ir ryšius. Svarbiausios šio metamodelio dalys yra šios:

- Reiškinys (angl. expression) – veiklos koncepto tekstinis aprašas;
- Atvaizdis (angl. representation) – konceptą ir jo tekstinį reiškinį siejantis elementas;
- Prasmė (angl. meaning) – veiklos konceptų prasmė;
- Ekstensija (angl. extension) – realaus pasaulio daiktų egzempliorių aibė.

VDBV yra pagrindinis žodyno aprašymo metamodelis, aprašo veiklos konceptus ir faktų tipus, papildomą informaciją apie žodyną.

VDBR – veiklos taisyklių aprašymo metamodelis;

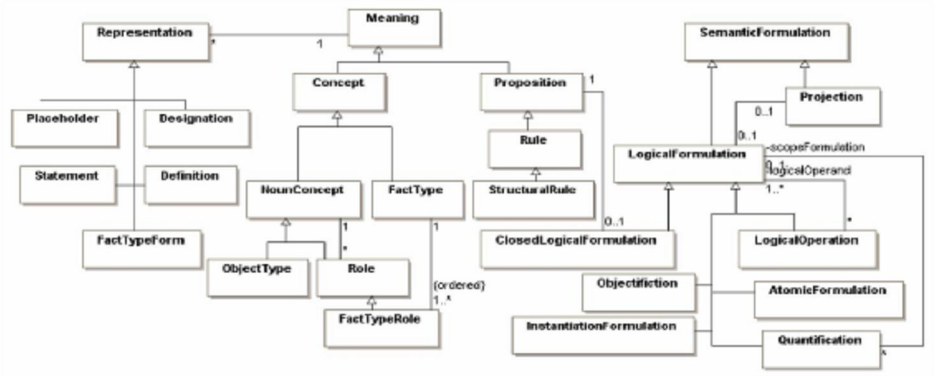
LFSV skirtas aprašyti veiklos taisyklių, išreikštų natūralia kalba, struktūrą semantinėmis formuluotėmis, kurios būna dviejų tipų:

- loginės formuluotės – skirtos aprašyti loginėms operacijoms, kvantoriams, atominėms formuluotėms, kurios paremtos faktų tipais ir kitos formuluotėms;
- projekcijos – formuluoja aprašus, agregacijas bei klausimus.

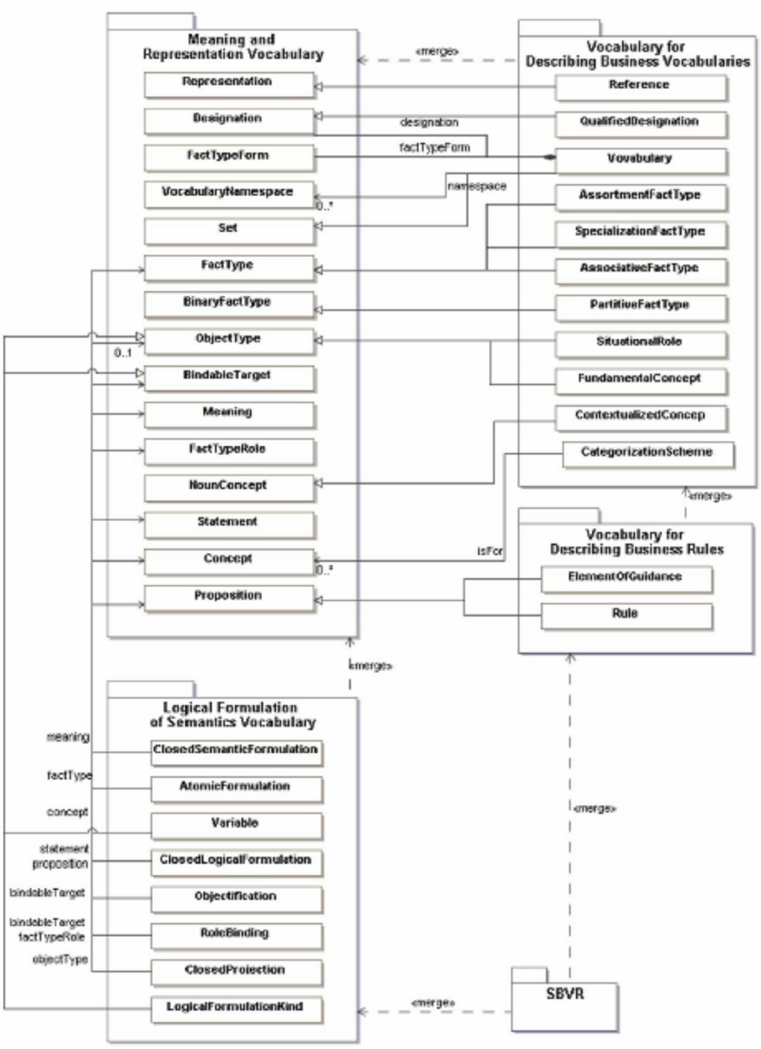
Šie keturi metamodeliai analizės metu buvo sujungti į vieną SBVR metamodelį tam, kad būtų paprasčiau vykdyti ATL transformacijas. Metamodeliai buvo jungiami per bendrus elementus, naudojant grafinį Eclipse diagramų redaktorių. „Paveikslas 5." pavaizduota metamodelių jungimo schema.

MRV metamodelis buvo naudojamas kaip pagrindinis. Prie jo buvo prijungtas VDBV metamodelis, kuris naudoja MRV elementus. Vėliau buvo prijungti VDBR ir LFSV metamodeliai. Schemoje pavaizduoti tik pagrindiniai metamodelių elementai.

„Paveikslas 4." pateiktas sujungto SBVR metamodelio fragmentas [1].



Paveikslas 4. Pagrindinės SBVR metamodelio klasės



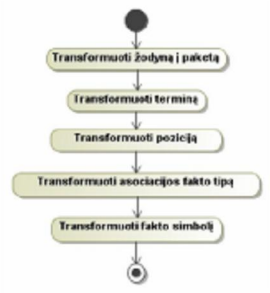
Paveikslas 5. SBVR metamodelių sujungimo schema

III. ATL TRANSFORMACIJA REMIANTIS SUDARYTU SBVR METAMODELIU

SBVR2UML transformacija realizuojama ATL procesoriumi, kuris veikia Eclipse platformoje. Kadangi šis tyrimas daugiausiai susijęs su projektavimu ir SBVR metamodelio sudarymu, transformacijų greičiui nebuvo keliami specialūs reikalavimai, kurie gali būti aktualūs taikant transformacijas kitokio pobūdžio uždaviniuose. Tuomet reikėtų atskirų tyrimų dėl transformacijų ir kodo generavimo greičio užtikrinimo. Sudėtingo modelio transformacijos laiko sąnaudos yra didelės, lyginant su mažiau elementų turinčiais modeliais.

A. ATL transformacija

ATL buvo realizuotas Eclipse 3.4.1 platformoje taikant SBVR 1.0 ir UML 2.1.2 metamodelius (XMI formatu), ATL transformacijų kalbą 3.0.0 ir ATL transformavimo procesorių 3.0.0. ATL transformacijos vykdymas SBVR žodynui pavaizduotas apibendrinta veiklos diagrama („Paveikslas 6."), Pirmiausia transformuojami terminai, reiškiantys primityvius objektų tipus, tada terminai, reiškiantys objektus, transformuojami į klases. Tada faktų tipų vaidmenų pozicijos transformuojamos į savybes, atitinkančias nuosavus klasių atributus ir į savybes, atitinkančias ryšių atributus. Faktų simboliai transformuojami į asociacijas ir apibendrinimus.



Paveikslas 6. SBVR2UML transformacijos apibendrinta veiklos diagrama

Norint gauti UML XMI kurią būtų galima importuoti ir gauti klasių diagramas reikia apsirašyti ATL transformacija, kurios kodo fragmentas pateiktas „Paveikslas 7."

```
rule ObjectType2Class {

    from
        s : MRV!ObjectType
    to
        t : UML!Class {

            name <- s.representation->first().text.value
        }

}
```

Paveikslas 7. ATL kalbos fragmentas panaudotas transformacijai iš SBVR į UML

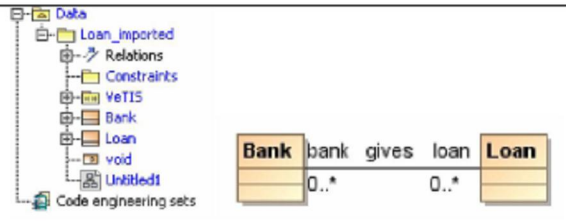
Lentelė 1. SUGENERUOTAS UML XMI SCHEMOS FRAGMENTAS, KURIS IMPORTUOJAMAS Į MAGICDRAW IR GAUNAMA KLASIŲ DIAGRAMA

Konceptas	bank *gives* loan

Transformuotos *UML XMI* schemos elementas

```
<packagedElement xmi:type="uml:Class" xmi:id="_F3Hu8Ay5Ed-GTMgM16I5Vw" name="Bank">
        <ownedAttribute xmi:id="_F3Hu8Qy5Ed-GTMgM16I5Vw" name="loan"
visibility="private" type="_F3Hu9Ay5Ed-GTMgM16I5Vw" association="_F3Hu-Qy5Ed-
GTMgM16I5Vw">
          <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_F3Hu8gy5Ed-
GTMgM16I5Vw" name="" value="*"/>
          <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_F3Hu8wy5Ed-GTMgM16I5Vw"
name=""/>
        </ownedAttribute>
      </packagedElement>
      <packagedElement xmi:type="uml:Class" xmi:id="_F3Hu9Ay5Ed-GTMgM16I5Vw"
name="Loan">
        <ownedAttribute xmi:id="_F3Hu9Qy5Ed-GTMgM16I5Vw" name="bank"
visibility="private" type="_F3Hu8Ay5Ed-GTMgM16I5Vw" association="_F3Hu-Qy5Ed-
GTMgM16I5Vw">
          <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_F3Hu9gy5Ed-
GTMgM16I5Vw" name="" value="*"/>
          <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_F3Hu9wy5Ed-GTMgM16I5Vw"
name=""/>
        </ownedAttribute>
      </packagedElement>
      <packagedElement xmi:type="uml:Association" xmi:id="_F3Hu-Qy5Ed-GTMgM16I5Vw"
name="gives" memberEnd="_F3Hu9Qy5Ed-GTMgM16I5Vw _F3Hu8Qy5Ed-GTMgM16I5Vw"/>
```

Transformuotos UML XMI schemos elementas importuotas į MagicDraw ir atvaizduotas grafikškai



B. *Sugeneruoti UML XMI schemos elementai panaudojant ATL*

Tyrime transformacija buvo testuojama sugeneruojant kiekvienam veiklos žodyno konceptui XMI schemą ir transformuojant ją į atitinkamus UML XMI schemos elementus [6], kurie buvo importuojami į MagicDraw UML įrankį ir pavaizduojami klasių diagrama. Kadangi sudėtingiems elementams reikėjo didelių fragmentų, čia pateikiami principiniai fragmentai, atspindintys atitinkamų elementų metamodelius ir transformacijas („Lentelė 1“).

Veiklos žodyno konceptams specifikuoti vartojami trys elementai:

- term - Terminas „term" skirtas vaizduoti objektų tipus (angl. object types arba general concepts) ir vaidmenis (angl. roles). Terminai užrašomi vienaskaita. Pavyzdžiui: sauna, hotel, loan, bank ir t.t.
- Name - Vardas „Name" skirtas vaizduoti individualius konceptus, kurie paprastai yra tikriniai daiktavardžiai. Viena iš išimčių - skaičių reikšmės, kurios taip pat vaizduojamos šiuo tipu (pvz., 25). Pavyzdžiui: KTU, Kaunas, katedra, 10 ir t. t.
- verb - Faktų simboliai „verb" vaizduoja veiksmažodį, prielinksnį arba jų kombinaciją. Veiksmažodžiai rašomi vienaskaita, aktyvia ar pasyvia formomis, kurios dažnai būna viena kitos sinonimai. Pavyzdžiui, aktyvi asociacijos fakto tipo „bank gives loan" forma turi pasyvią sinonimine formą „loan is given by bank".

Šie elementai pasinaudojant ATL transformacijos taisykles transformuojami į atitinkamus UML elementus.

IV. IŠVADOS IR TOLIMESNI TYRIMAI

Šiuo metu egzistuoja tik keli bandymai transformuoti tarpusavyje SBVR ir UML: Cabot ir bendraautorių transformacija iš UML į SBVR [2], [1]; labai supaprastinta transformacija iš SBVR į UML [4]. Bendriniai SBVR transformavimo principai nagrinėjami [5]. VeTIS SBVR2UMLOCL transformacija yra kur kas išsamesnė, ji apima (beveik) visų SBVR veiklos žodyno konceptų transformavimą į UML klasių diagramas ir pagrindinių SBVR veiklos taisyklių transformavimą į UML klasių diagramų elementus bei OCL invariantus ir operacijų prieš sąlygas. Be to, sukurtas IS projektavimo metodas, kuriame veiklos žodynų ir veiklos taisyklių aprašų sudarymas integruotas į modeliais grindžiamą kūrimo procesą.

Transformacijoms realizuoti buvo pritaikytas Cabot (2009) sudarytas SBVR metamodelis (susidedantis iš keturių metamodelių), kurio savybės buvo papildytos SBVR standarto atžvilgiu tam, kad jį būtų galima pritaikyti realizacijai. Cabot SBVR metamodeliai buvo sujungti ir papildyti kai kuriais ryšiais, kadangi jų reikėjo pilnai išpildyti transformaciją; kai kurių Cabot SBVR metamodelio savybių buvo atsisakyta, siekiant išlaikyti jį kuo artimesnį oficialiam standartui.

Veiklos žodynų ir taisyklių specifikavimo ribota natūralia kalba galimybių plėtra turi dideles perspektyvas. Atskirų veiklos sričių ribotos natūralios kalbos žodynai leidžia sukurti kompiuteriams ir žmonėms suprantamas sąsajas tarp duomenų bazių / programinių paslaugų ir realiame pasaulyje vartojamos žmonių kalbos. Jie vienareikšmiškai aprašo veiklos žinias ir leidžia jas taikyti informacinių sistemų projektavimo procesuose.

SBVR metamodelis leidžia aprašyti tam tikros veiklos srities ontologiją, taigi veiklos žodynų ir veiklos taisyklių redaktorių galima pritaikyti ontologijoms kurti ribota natūralia kalba. KTU Informacijos sistemų katedroje jau pradėti tyrimai šia tema. Praktikoje plačiai taikomų tokios paskirties įrankių kol kas nėra, tačiau šioje srityje vykdomi moksliniai darbai rodo šių įrankių aktualumą (Eder 2008), (Linehan 2008), (Lukichev 2007), (Wagner 2004, 2006). Veiklos žodynai, pavaizduoti ontologijomis, turi daug žadančias potencialias galimybes susieti Semantinio tinklo ir įmonių programinės įrangos paslaugų technologijas, kurti semantines interneto paslaugas, integruoti paskirstytas duomenų bazes, vystyti natūralios kalbos technologijas ir kt.

VeTIS projekto rezultatai parodė, kad veiklos žodynų ir taisyklių specifikavimo ribota natūralia kalba galimybių plėtra turi dideles perspektyvas. Atskirų veiklos sričių ribotos natūralios kalbos žodynai leidžia sukurti kompiuteriams ir žmonėms suprantamas sąsajas tarp duomenų bazių / programinių paslaugų ir realiame pasaulyje vartojamos žmonių kalbos. Jie vienareikšmiškai aprašo veiklos žinias ir leidžia jas taikyti informacinių sistemų projektavimo procesuose.

Tolimesni tyrimai tikslingi šiais požiūriais:

- Sukurti galimybes aprašyti veiklos žodynus ir taisykles keliomis kalbomis, pirmiausia lietuvių kalba. Šiam tikslui veiklos konceptų terminų, vardų ir faktų tipų formų specifikacijas reikėtų išplėsti kalbų atributais; lietuvių kalbai šias specifikacijas reikėtų papildyti terminų formomis, kurios leistų nurodyti terminų linksnius;
- Sukurti atvirkštinę transformaciją iš UML / OCL į SBVR projektinei dokumentacijai generuoti [10];
- Plėtoti tyrimus, kurie leistų taikyti SBVR veiklos žodynus ir veiklos taisykles Pasaulio semantinio tinklo sprendimuose, susiejant specialių sričių ontologijas su platesnėmis ontologijomis, integruojant informacinius išteklius ir plėtojant semantines tinklo paslaugas.
- Plėtoti tyrimus, kurie leistų rasti būdą pagreitinti transformacijas dideliems modeliams.

LITERATŪRA

[1] Cabot J., Pau R., Ravento R. From UML/OCL to SBVR specifications: A challenging transformation, [interaktyvus, žiūrėta 2010-04-01]. Prieiga internete: http://www.lsi.upc.es/~jcabot/papers/IS09.pdf.

[2] Cabot J., Pau R., Ravento R. UML-to-SBVR and SBVR-to-HTML transformations, [interaktyvus, žiūrėta 2010-03-10]. Prieiga internete: http://jordicabot.com/research/SBVR/index.html

[3] Ceravolo, P., Fugazza, C., Leida, M. Modeling Semantics of Business Rules. 2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007), 2007, 171–176.

[4] Kleiner, M., Albert, P., Bézivin, J. Parsing SBVR-Based Controlled Languages. In Model Driven Engineering Languages and Systems, LNCS, Vol. 5795, 2009, 122–136.

[5] Linehan, M. H. Semantics in Model-Driven Business Design. In Proc. 2nd Int. Semantic Web Policy Workshop (SWPW'06), 2006.

[6] OMG Unified Modeling Language, Superstructure, V2.1.2. OMG formali specifikacija/2009-02-02, 2009.

[7] Raj A., Prabhakar T. V., Hendryx S. Transformation of SBVR business design to UML models. In: ISEC '08: Proceedings of the 1st conference on India software engineering conference, ACM, Hyderabad, India, 2008, 29–38.

[8] Schacher M. Business Rules from an SBVR and an xUML Perspective (Parts 1–3). Veiklos taisyklių žurnalas, 2006, vol. 7, Nr. 6–8.

[9] Semantics of Business Vocabulary and Business Rules (SBVR). Version 1.0. December, 2008, Prieiga internete: http://www.omg.org/docs/formal/08-01- 02.pdf.

[10] Šinkevičius E., Butleris R. Transformacija iš UML į SBVR naudojant Atlas kalbą Eclipse aplinkoje. 14-osios Tarpuniversitetinės magistrantų ir doktorantų mokslinės konferencijos "Informacinės technologijos" pranešimų medžiaga / Vilniaus universitetas, Vytauto Didžiojo universitetas, Kauno technologijos universitetas. [Kaunas] : [Vilniaus universiteto Kauno Humanitarinis fakultetas]. ISSN 2029-249X., p. 187-190., 2009

3 priedas. Mokslinis straipsnis konferencijoje „Informacinė visuomenė ir universitetinės studijos (IVUS 2010)“

Veiklos procesų modeliavimo praplėtimas veiklos taisyklėmis

Lina Tutkutė, Edvinas Šinkevičius
Informacijos sistemų katedra
Kauno technologijos universitetas
Kaunas, Lietuva
lina.tutkute@ktu.lt, edvinas.sinkevicius@ktu.lt

Santrauka— **straipsnyje aptarta galimybė veiklos procesų modeliavimą praplėsti veiklos taisyklėmis. Išanalizuoti veiklos procesai, jų struktūra, modeliavimo standartai. Nustatyti veiklos taisyklių abstrakcijos lygmenys, veiklos taisyklių klasifikacija bei integracijos į informacinę sistemą galimybės. Pateiktas VT integracijos į veiklos procesus modeliavimo standartą planas. Sudaryta preliminari veiklos procesų modeliavimo naudojant VT metodika. Pagrįstas veiklos procesų modeliavimo praplėsto veiklos taisyklėmis naudingumas ir pritaikomumas.**

Raktiniai žodžiai – veiklos procesai; veiklos taisyklės; veiklos procesų modeliavimas; BPMN; BPDM

I. Įvadas

Organizacijos, siekiančios adaptuotis prie sparčiai besikeičiančių aplinkos sąlygų, privalo optimizuoti savo procesus. Veiklos procesų (VP) modeliavimas suteikia galimybę ne tik grafiškai atvaizduoti organizacijoje vykdomus procesus, bet ir vykdyti jų simuliavimą, tokiu būdu nustatant silpnąsias vietas. Vystant veiklos procesų modeliavimo metodikas ir įrankius, sukurtas OMG grupės standartas – BPMN (angl. Business Process Modeling Notation), kuris tinkamai atvaizduoja realaus pasaulio objektus bei eliminuoja atotrūkį tarp veiklos procesų ir kompiuterizuojamų procesų modeliavimo. Vis dėlto šis standartas modeliuoja proceso dinamiką, tačiau nepilnai įvertina galimus apribojimus, t.y. specifikuojami apribojimai nėra pakankamai išsamūs ir nepilnai atvaizduoja realaus pasaulio procesų elgseną. Dėl šios priežasties veiklos procesų modeliavimą būtina papildyti apribojimų valdymo elementais, t.y. veiklos taisyklėmis (VT). Jos užtikrina įvairaus sudėtingumo apribojimų/iniciavimo/stabdymo sąlygų įvertinimą veiklos procese. Veiklos taisyklė nusako kaip turi būti elgiamasi tam tikroje situacijoje, esant tam tikroms sąlygoms. Veiklos taisyklių klasifikacijų yra įvairių, dėl to būtina atrinkti tinkamiausią veiklos proceso specifikavimui. Veiklos taisyklės ne tik įvertina esamus apribojimus, tačiau taip pat suvienodina organizacijos veiklos žodyną, kurio pagrindu jos formuojamos.

Pirmame skyriuje aptarta veiklos procesų modeliavimo sąvoka, išnagrinėtos modeliavimo vystymosi tendencijos ir dabartinė situacija. Atlikta VP metodikų analizė, įvertinant organizacijos atstovo ir IT eksperto indėlį modeliuojant procesus bei nustatant modeliavimo rezultato panaudojimą, kuriant informacinę sistemą (IS) ar optimizuojant įmonės veiklą. Aprašytas veiklos procesų modeliavimo metamodelis (BPDM – angl. Business Process Definition Metamodel), jo struktūra, privalumai modeliuojant VP skirtinguose įrankiuose ir realizuojant juos skirtingose platformose. Išanalizuota veiklos proceso struktūra pagal BPMN, jo galimybės atvaizduoti realaus pasaulio objektus, jų tarpusavio ryšius, taip pat nustatyti esami trūkumai, susiję su apribojimų veiklos procesams aprašymu. Antrame skyriuje analizuojamos veiklos taisyklių abstrakcijų lygiai bei galimos klasifikacijos. Trečiame skyriuje pasiūlyta schema informacinės sistemos praplėtimui veiklos taisyklėmis, integruojant jas veiklos procesų modeliavime. Taip pat aprašomi veiklos procesų modeliavimo metamodelio praplėtimo veiklos taisyklėmis žingsniai bei detalizuojami VP modeliavimo etapai formuojant veiklos taisykles. Paskutiniame skyriuje pateikiamos išvados bei planuojami tolimesni tyrimai.

II. Veiklos procesų modeliavimas

Veiklos procesų modeliavimas (angl. Business Process Modeling) – tai organizacijos procesų atvaizdavimas ir simuliavimas programinės įrangos pagalba. Procesų modeliavimas organizacijoje atliekamas norint įdiegti naują programinę/techninę įrangą, atnaujinti įrangą, optimizuoti veiklos procesus, restruktūrizuoti/plėsti veiklą, įsidiegti sertifikatus, apjungti/išskaidyti organizaciją [10].

A. Veiklos procesų vystymosi tendencijos

Pirmoji procesų modeliavimo karta (1900-1950m.) buvo srautų diagramos, funkcinių srautų blokinės diagramos, duomenų srautų diagramos, kontrolinių srautų diagramos, Ganto diagramos, Petri tinklai ir IDE. Šie modeliai buvo paremti funkcijų ir procedūrų identifikavimu bei valdymu. Pagrindinė šios metodikos paskirtis buvo inžinerinių procesų dokumentavimas. Antrojoje kartoje (1950-1990m.) pradėti modeliuoti ne tik inžineriniai, bet ir veiklos procesai [15], [18].

Trečioje procesų modeliavimo kartoje [15] funkcijas ir procedūras pakeitė procesai. Augant organizacijų vykdomos veiklos sudėtingumui, funkcijomis grindžiamos metodikos neužteko identifikuoti vykdomų procesų, jų tarpusavio sąveikos bei apribojimų. Proceso sąvoka leido sumodeliuoti sudėtingas veiklas, išlaikant procesų tarpusavio nepriklausomumą, tačiau tuo pat metu formaliai aprašant jų tarpusavio sąveiką bei reikiamus apribojimus [15]. Ši procesų modeliavimo metodika įvedė naują sąvoka procesų reinžinerija. Tokiu būdu buvo sudarytas uždaras proceso gyvavimo ciklas (identifikavimas, modeliavimas, vykdymas, optimizavimas) užtikrinantis tinkamą procesų valdymą.

Pradėjus modeliuoti procesus programinės įrangos pagalba, veiklos procesų ir procesų modeliavimo sąvokos buvo atskirtos. Pagrindinis dėmesys buvo akcentuojamas kompiuterizuojamiems organizacijos procesams. Dėl šios priežasties organizacija prarado tinkamą šių procesų valdymą, nes modeliavimą atlikdavo IT analitikai/projektuotojai. Modeliuojant procesus nebūdavo įvertinami organizacijos verslumo aspektai. Siekiant diagramas padaryti aiškias tiek IT analitikams, tiek organizacijos atstovams, buvo pradėtos vystyti procesų modeliavimo grafinės notacijos (UML – angl. Unified Modeling Language).

Šiuo metu, ketvirtojoje procesų modeliavimo kartoje sudaryti modeliai ne tik leidžia grafiškai identifikuoti procesus, tačiau taip pat leidžia sumodeliuoti visą informacinę sistemą ir tokiu būdu gauti ją realizuojantį programinį kodą (MDA – angl. Model Driven Architecture) [13]. Siekiant pilnai sumodeliuoti sistemą, būtina tinkamai identifikuoti organizacijoje vykdomus veiklos procesus. Tokiu būdu užtikrinama pilna sukurtos informacinės sistemos integracija [14]. Šis procesų modeliavimas apima ir esybių, duomenų srautų, procesų srautų, vartotojo sąsajos modeliavimą. Vis dėlto, kaip jau minėta, organizacijos identifikuoja procesus ne tik kurdamos informacines sistemas, bet ir pvz. optimizuodamos savo veiklą. Vystant procesų grafinę notaciją, organizacijos atstovams tapo įmanoma patiems identifikuoti ir apsibrėžti savo veiklos procesus. Naujausi standartai (BPMN) leidžia apibrėžti procesą, jo vykdymo aplinkybes, apribojimus, atsakingus asmenis bei sekti procesų vykdymą ar atlikti jų simuliavimą. Atotrūkis tarp veiklos procesų ir inžinerinių procesų yra eliminuotas. Šiuo metu veiklos procesų modeliavimas dažnai sutapatinamas su veiklos procesų valdymu (angl. Business Process Management), kadangi apima visą funkcionalumą.

Lygiagrečiai BPMN [12] (OMG grupės standartas) yra vystomi ir kiti veiklos procesų modeliavimo metodai tokie kaip Darbų sekų modelis (angl. Workflow), IDEF (angl. Integration Definition Function Modeling), UML (angl. Unified Modeling Language) [11]. Darbų sekų modelyje [9], [19] veiklos procesas yra išskaidomas į darbų sekas, identifikuojant atsakingus asmenis bei ryšį su kitais procesais. UML skirtas modeliuoti kompiuterizuojamus veiklos procesus [7] ir atstovauja PIM (angl. Platform Independent Model) lygmenį MDA hierarchijoje. BPMN tiksliausiai atvaizduoja realaus pasaulio objektus ir savo notacija yra suprantamas organizacijų atstovams [12].

B. Veiklos procecų aprašymo metamodelis (BPDM)

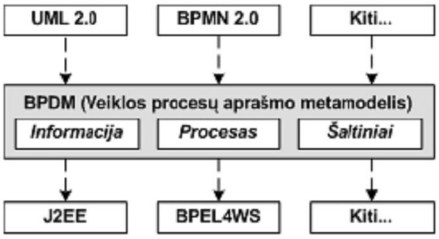
Veiklos procesas yra rinkinys susietų struktūrizuotų veiklų ar užduočių, vykdančių tam tikrą paslaugą ar gaminančių tam tikrą produktą. Veiklos procesą galima dekomponuoti į keletą subprocesų. Dekomponavimas vykdomas naudojant „iš viršaus – į apačią“ (angl. top-down) principą, kol procesas yra išskaidomas į nedalomas veiklas (angl. activity).

Veiklos procesai, priklausomai nuo pasirinktos modeliavimo metodikos, turi skirtingas struktūras. Veiklos procesų modeliavimo pagrindinis tikslas yra pilnai ir vienareikšmiškai atvaizduoti realaus pasaulio procesus. Taip pat, remiantis MDA metodika, turi būti ne tik galimybė simuliuoti veiklos procesus, bet juos sumodeliavus transformuoti į programinį kodą.

Siekiant įgyvendinti šiuos abu uždavinius, OMG (angl. Object Management Group) grupė sudarė formalų metamodelį BPDM skirtą vykdomų veiklos procesų (angl. Executable Business Process) sintaksei apibrėžti [2]. Metamodelis vienareikšmiškai identifikuoja elementus ir ryšius tarp jų, kurie nusako veiklos procesą. BPDM neaprašo veiklos proceso atvaizdavimo, t.y. nėra notacija. Jis yra nuo platformos nepriklausomas. Sumodeliuotas veiklos procesas gali būti transformuotas į pasirinktą technologinę platformą. Kadangi proceso struktūrą yra vienareikšmiškai apibrėžta, sudarius transformaciją, ją galima atlikti pakartotinai ir automatizuotai (1 pav.). Tokiu būdu BPDM užtikrina sumodeliuotų veiklos procesų suderinamumą tarp skirtingų notacijų (BPDM, UML ir kitos) ir technologinių platformų (J2EE, BPEL4WS [1], XPDL ir pan.), įgalina veiklos procesų migraciją tarp modeliavimo įrankių, palaikančių šį standartą.

BPDM metamodelio pagrindas yra PSL (angl. Process Specific Language) [1] .

BPDM palaiko tiek procesų orkestruotę, tiek choreografiją. Orkestruotė – tai proceso ir jį detalizuojančių elementų (veiklų, ryšių, informacinių srautų, apribojimų) valdymas. Choreografija – tai abstraktesnė sąvoka, nusakanti pačių procesų valdymą tarp bendradarbiaujančių esybių, pvz. tarp skirtingų organizaciją atstovaujančių vaidmenų (angl. actor), tarp vidinių ir išorinių esybių (klientų, tiekėjų, padalinių ir pan.).



1 pav. Veiklos procesų modelių transformacijos naudojant BPDM

BPDM metamodelis sudarytas iš 6 paketų sugrupuotų į tris grupes:

- Bendros elgsenos modelis (angl. *Common Behavior Model*) aprašo dinaminės elgsenos aspektus tarp orkestruotės ir choreografijos bendruoju atveju (Elgsenos modelis ir Sąveikos elgsenos modelis), apima visus elementus skirtus procesų elgsenai aprašyti;
- Veiklos modelis (angl. *Activity Model*) su BPMN praplėtimu skirtas orkestruotės valdymui, t.y. procesų vidinių elementų tarpusavio sąveikai ir vykdymui Jis palengvina procesų pokyčių valdymą pasikeitus veiklos sąlygoms.
- Sąveikų protokolo modelis (angl. *Interaction Protocol Model*) yra skirtas choreografijų valdymui, t.y. jų grupavimui, pakartotiniam panaudojimui.

BPDM naudodamas elementus iš bendrų modelių bibliotekos, užtikrina suderinamumą su kitais MOF (angl. Meta Object Facility) grupės standartais.

BPDM yra veiklos procesų modeliavimo kalbos BPMN metamodelis. Taigi BPMN yra nuo platformos nepriklausoma kalba, skirta veiklos procesų orkestruotėms ir choreografijai modeliuoti. BPMN standartas yra grafinė notacija, kurios elementai skirti modeliuoti verslo procesus ar transakcijas. Šios notacijos paskirtis - palengvinti organizacijų bendradarbiavimą bei verslo transakcijų supratimą [7], [3]. Elementų aibė apima procesų pradžią, vidinį judėjimą, procesus, srautus, transakcijas bei jų užbaigimą. Notacija orientuota į procesų sekas arba transakcijas. Elementų aibės plėtiniai leidžia detaliai projektuoti procesus ir transakcijas, o detalūs modeliai yra informatyvūs realizuojant sistemą tiek rankiniu, tiek automatiniu būdu.

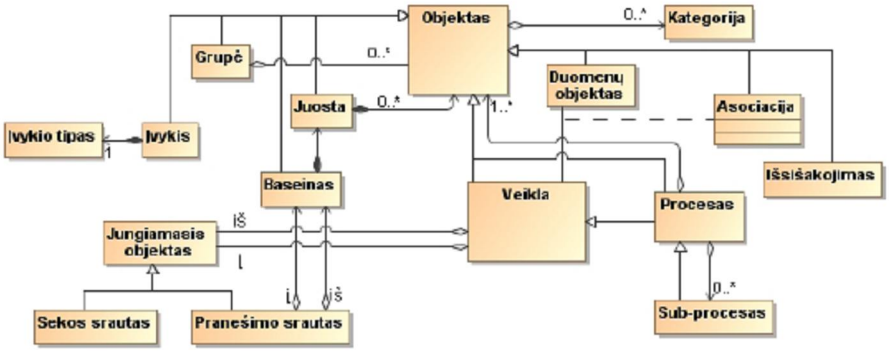
BPMN taip pat perteikia tradicinio verslo procesų galimybes tvarkant B2B procesų sąvokas, tokias kaip viešus bei privačius procesus ir choreografiją, taip pat pažangias modeliavimo sąvokas, tokias kaip prieštaravimo tvarkymas, operacijų ir kompensavimo valdymas.

C. *Proceso struktūra*

Pagrindinis BPMN konceptas yra Procesas (angl. Process), kurį gali sudaryti keletas sub-procesų (angl. subprocess) (3 pav.), tai leidžia išskaidyti modelio elgseną ir pakartotinai panaudoti proceso komponentus ir veiklas (angl. activity). Veiklos gali turėti transakcijų semantiką (angl. transactional semantics), tai reiškia, kad veiklose saugoma elgsena turi būti maksimaliai atominė. Veiklas gali atlikti žmonės, organizacijos arba automatizuotos sistemos

Veiklos, sudarančios procesą, jungiamos jungiamaisiais objektais (angl. connecting object), tai gali būti sekos ar pranešimo srautas. Esant poreikiui, jungiamieji objektai praplečiami išsišakojimais (angl. gateways), esant proceso vykdymo išsišakojimui ir įvykiais (angl. event), kurie gali inicijuoti/stabdyti proceso vykdymą. BPDM elementai yra grupuojami į juostas (angl. lane), jos į baseinus (angl. pool). Juostos skirtos specifikuoti procesą, t.y aprašyti orkestruotę. Pranešimų srautai tarp baseinų realizuoja procesų choreografiją.

Apribojimams išskirtinis dėmesys nėra skiriamas, juos galima dalinai realizuoti naudojant išsišakojimus arba pranešimo srautus, tačiau tolimesniame procesų modeliavime ar simuliavime jie neatsispindi. Vis dėlto neužtenka integruoti sudėtingesnius apribojimus į veiklos procesų modeliavimą, būtina užtikrinti ir jų valdymą. Dėl šios priežasties veiklos procesų modeliavimą būtina papildyti nauja sąvoka - veiklos taisyklė.



2 pav. Veiklos proceso metamodelis (pagal BPDM)

III. Veiklos taisyklių klasifikacija

A. *Veiklos taisyklių abstrakcijos lygmenys*

Veiklos taisyklė – tai loginis teiginys, nusakantis kaip turi būti elgiamasi, t.y. kokių veiksmų turi būti imtasi, konkrečioje situacijoje. Veiklos taisyklė gali būti keturių skirtingų abstrakcijos lygių (4 pav.) [8], kiekvienas iš jų skirtas skirtingai darbo grupei dirbančiai su VT [16]. Pirmajame lygyje veiklos taisyklę suformuoja įmonės atstovai, išreikšdami idėją, kad tam tikras įmonės vykdomas procesas turėtų būti kontroliuojamas. Antrame lygyje VT idėja užfiksuojama neformaliu VT aprašu, nusakančiu jos pagrindinę paskirtį, tačiau nedetalizuojant jos veikimo. Šiame lygmenyje siekiama VT poreikį išgryninti, t.y. nors ir neformaliai veiklos taisyklė yra interpretuojama kaip nedalomas vienetas susijęs su konkrečiais verslo procesais. Šio lygio veiklos taisykles leidžia modeliuoti SBVR (angl. Semantics Business Vocabulary and Rules) standartas. Trečiame lygyje veiklos taisyklė yra modeliuojama pasirinkus taisyklių specifikavimo kalbą. Ši užduotis yra skirta IT ekspertui (analitikui/projektuotojui). VT specifikacija atsako į klausimą „ką veiklos taisyklė turi daryti“, bet ne „kaip“.

3 pav. Veiklos taisyklių abstrakcijos lygiai

Šiame lygyje veiklos taisyklė įgyja visas deklaratyviai apibrėžtai taisyklei reikalingas savybes, tokias kaip užbaigtumas, tikslumas, autonomiškumas, deklaratyvumas, aktualumas.

Idealiu atveju, tinkamai sumodeliavus veiklos taisykles, remiantis MDA (angl. *Model Driven Architecture, Modeliais grindžiama architektūrą*) principu, iš jų galima sugeneruoti programinį kodą. Kadangi nėra vieningos VT modeliavimo kalbos, taigi skirtingi veiklos taisyklių valdymo įrankiai tas pačias VT realizuoja skirtingai. Dėl to, priklausomai nuo VT sudėtingumo, tam tikrą dalį sumodeliuotų veiklos taisyklių VT valdymo įrankis pats transformuoja į programinį kodą, likusioms (sudėtingą logiką turinčioms VT) reikia parašyti vykdymo kodą. Tai yra ketvirtas abstrakcijos lygis, kai veiklos taisyklės yra išreikštos programiniame kode. Su šiomis VT dirba programuotojas.

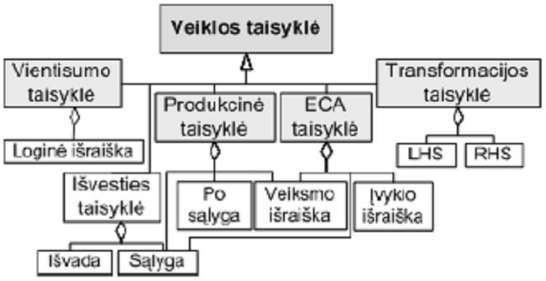
Ilgą laiką įmonės atstovams veiklos taisyklių valdymas buvo neprieinamas, kadangi jos buvo programuojamos ir talpinamos programiniame kode ar duomenų bazių trigeriuose. Siekiant VT atskirti nuo programinio kodo, buvo sukurtos VTVS (veiklos taisyklių valdymo sistemos). Jų pagrindinė paskirtis [6], [17], turint IS duomenų struktūrą, grafinio įrankio pagalba sukurti veiklos taisykles ir integruoti jas į informacinę sistemą. Vis dėlto šiame etape veiklos taisyklės būdavo aprašomos formaliomis struktūromis, t.y. būdavo trečio abstrakcijos lygmens. Organizacijos atstovams ne visada būdavo suprantami VT aprašymo būdai ir susiejimai su duomenų struktūromis. Taip pat visiškai nebuvo susiejimo su veiklos procesais (organizacijos atstovas galėjo tik vizualiai VT grupuoti). Siekiant tinkamo veiklos taisyklių valdymo, jos turi būti integruotos į veiklos procesų modeliavimą ir būti antro/trečio abstrakcijos lygmenų.

B. Veiklos taisyklių klasifikacija

Veiklos taisyklės gali turėti ne tik skirtingus abstrakcijos lygmenis, bet ir skirtingą struktūrą. Siekiant tinkamai sumodeliuoti veiklos taisykles, reikia įvertinti VT galimybes atsižvelgiant į jų tipus ir struktūrą. Kadangi nėra vieningos VT modeliavimo kalbos, dėl to galimų VT klasifikacijų yra labai daug. Veiklos taisyklės pagal jų veiksmus informacinėje sistemoje gali būti grupuojamos į šias tris kategorijas:

- apribojančios informaciją priklausomai nuo veiklos įvykio (apima apribojimus ir rekomendacijas);
- leidžiančios vykdyti veiksmus, įvykus tam tikram veiklos įvykiui (apima leidimus vykdyti);
- sukuriančios naujus duomenis, įvykus tam tikram veiklos įvykiui (apima skaičiavimus ir išvadas).

Pagal detalesnę veiklos taisyklių klasifikaciją (5 pav.), VT skirstomos į vientisumo, išvesties, produkcines, ECA (įvykiais grindžiamas), transformacijos taisykles [6]. Taisyklė gali: apibrėžti terminą, sujungti terminus į faktus, pateikti skaičiavimų rezultatus, patikrinti sąlygas naujo fakto sukūrimui, patikrinti sąlygas veiksmui inicijuoti.



4 pav. Veiklos taisyklių klasifikacija pagal atliekamas funkcijas informacinėje sistemoje

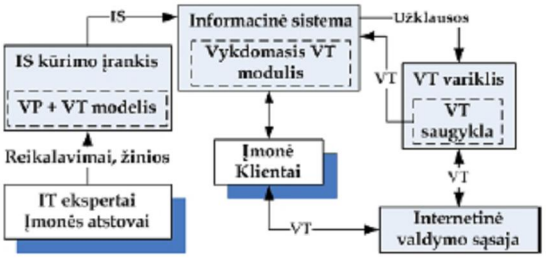
VT modeliavimo įrankiuose dažniausiai naudojama struktūra yra *if* <sąlyga> *then* <veiksmas1> *else* <*veiksmas2*> (produkcinė taisyklė). *Sąlyga* nurodo kas turi būti tenkinama, kad būtų galima vykdyti *veiksmas1*, priešingu atveju vykdomas *veiksmas2*. Veiksmai gali būti įvairūs: sprendimo priėmimas, informacijos išvedimas, duomenų įrašymas/keitimas, veiksmo inicijavimas (pvz. *if* studento vidurkis yra 10, *then* stipendija yra 200Lt).

IV. Veiklos procesų modeliavimo metodo praplėtimas veiklos taisyklėmis

A. Informacinės sistemos praplėtimas veiklos taisyklių moduliu

Šiuo metu veiklos procesų modeliavimas apima veiklos dinamikos specifikavimą, įvertinant tam tikrus bazinius apribojimus (pvz. išsišakojimus, inicijavimo/stabdymo/nutraukimo sąlygas ir pan.). Visus kitus papildomus apribojimus ar sudėtingesnes sąlygas reikia specifikuoti atskirai ir jos nėra įvertinamos simuliuojant sumodeliuotus procesus.

Taigi, modeliuojant naują informacinę sistemą ar rekonstruojant esamą (6 pav.), būtina veiklos procesų modeliavimo (*VP*) įrankį praplėsti veiklos taisyklių modeliavimu (*VT*). Tokiu būdu būtų užtikrinta, kad *įmonės atstovai* identifikuos veiklą ribojančios sąlygos pirmojoje IS kūrimo stadijoje ir susies su konkrečiais veiklos procesais.



5 pav. Informacinės sistemos kūrimas grindžiamas veiklos procesų modeliamu bei veiklos taisyklių valdymu

Sumodeliuotos VT taisyklės yra transformuojamos į vykdomą kodą ir saugomos *VT saugykloje*. Kadangi *VT saugykla* yra autonomiškas elementas, jis gali būti prieinamas skirtingoms duomenų saugykloms, taikomosioms programoms.

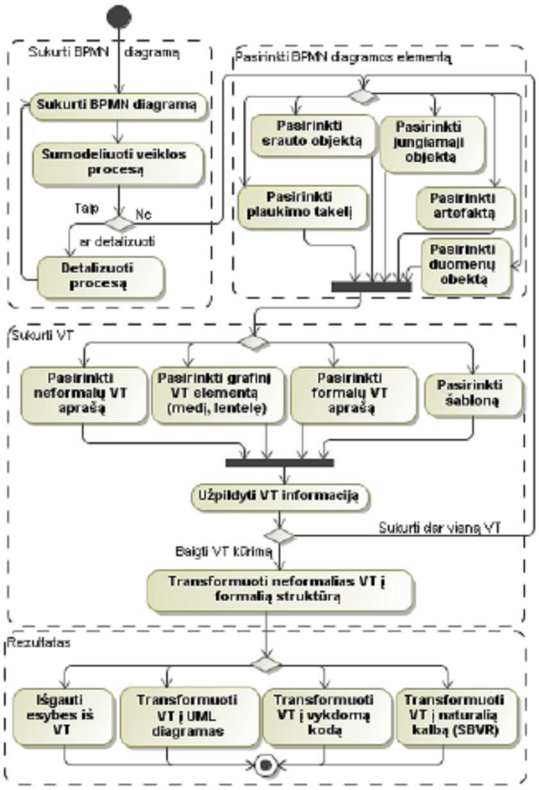
Sukūrus veiklos taisykles, *vykdomasis VT modulis* yra integruojamas į *informacinę sistemą*. Dirbant su informacine sistema, ji reikiamu momentu kreipiasi į *VT* variklį, kuris atrenka reikiamas VT iš *VT saugyklos* ir per *vykdomąjį VT modulį* grąžina. Įmonės atstovas esant poreikiui gali modifikuoti VT per *internetinę valdymo sąsają*.

B. *Veiklos taisyklių integracija į veiklos procesų modeliavimo metamodelį*

Veiklos procesų modeliavimo metodo (BPMN) praplėtimas veiklos taisyklių modeliavimu vykdomas šiais etapais:

- identifikuoti BPMN elementus;
- identifikuoti BPMN diagramos vietas ir elementus, kurie gali būti praplėsti VT;
- sudaryti veiklos taisyklių metamodelį;
- papildyti BPMN metamodelį VT metamodeliu;
- nustatyti VT formalizavimo laipsnio priklausomybę nuo BPMN diagramos detalizavimo lygmens.

Praplėtus veiklos procesų modelį veiklos taisyklėmis, veiklos procesų diagramos sudarymas yra pateikiamas 7 pav.



6 pav. Veiklos procesų modeliavimas integruojant veiklos taisykles

Pirmas etapas yra *sudaryti BPMN diagramą*. Antras etapas – *pasirinkti reikiamą diagramos elementą*, kuriam norima sukurti veiklos taisyklę. Diagramos elementai gali būti penkių tipų srauto objektas (įvykis, veikla, išsišakojimas), duomenų objektas (duomenų objektas, duomenų įeiga, duomenų išeiga, duomenų saugykla, savybės), jungiamasis objektas (sekos srautas, pranešimo srautas, asociacija, duomenų asociacija), plaukimo takelis (juosta, baseinas) ir artefaktai (grupės, teksto anotacijos). Trečias etapas yra *sukurti veiklos taisyklę*. Veiklos taisyklės gali būti skirtingų tipų, atsižvelgiant į detalumo lygmenį (neformalus/formalus aprašas), atvaizdavimo formą (medis, lentelė), pakartotinis panaudojimas (šablonas).

Veiklos taisyklių integravimo į veiklos procesų modeliavimą, privalumai:

- modeliuojant veiklos taisykles (statiką), yra sudaromas veiklos žodynas, kuris suvienodina organizacijoje naudojamus terminus;
- veiklos taisyklės kartu su veiklos procesais gali būti transformuojami į atitinkamas UML diagramas, t.y. vykdomas perėjimas nuo CIM prie PSM modelių pagal MDA metodiką;
- veiklos taisyklės kartu su veiklos procesais gali būti transformuojamos į vykdomą kodą (priklausomai nuo įrankio galimybių);
- veiklos taisykles galima transformuoti į veiklos proceso dokumentaciją [5], t.y. organizacijos atstovas gali perskaityti VT užrašytas natūralios kalbos sakinias (SBVR standartas), jas patvirtinti arba atmesti. Tokiu būdu validuojamas veiklos procesas.

V. IŠVADOS IR TOLIMESNI TYRIMAI

Organizacijoms, siekiančios optimizuoti savo kaštus, būtina tinkamai identifikuoti vykdomus procesus. Šiuo metu egzistuojančios modeliavimo priemonės, tokios kaip BPMN, leidžia organizacijų atstovams naudotis lengvai įsisavinama notacija ir jos pagalba modeliuoti bei simuliuoti savo veiklos procesus. Vis dėto, norint identifikuoti tam tikrus veiklos apribojimus, kurie dažniausiai būna sudėtingi, esamos notacijos neužtenka, ją būtina praplėsti papildomomis struktūromis - veiklos taisyklėmis. Tokiu būdu būtų išvengta papildomų dokumentų, aprašančių sudėtingus apribojimus, bei užtikrinta, kad į juos bus atsižvelgta vykdant procesų simuliavimą. Tokiu būdu galima užtikrinti pilną VP kontrolę. Veiklos taisyklių identifikavimas pirminiame procesų modeliavimo etape (CIM), užtikrina tinkamą jų įgyvendinimą vykdomame kode. Panaudojant veiklos taisykles, galima sudaryti/suvienodinti organizacijoje naudojamą veiklos žodyną.

Tolimesni tyrimai bus susieti su veiklos taisyklių metamodelio sudarymu ir jo integraciją į veiklos procesų metamodelį. Šiame etapus bus nustatyti galimi veiklos taisyklių tipai, jų pritaikymo galimybės skirtingiems BPMN standarto elementams. Veiklos taisyklės turi užtikrinti ne tik grafinį informacijos (apribojimų) atvaizdavimą, bet ir tolimesnį jos panaudojimą (generuojant kodą, modeliuojant IS, dokumentuojant procesus ir pan.).

LITERATŪRA

[1] C. Bock and M. Gruninger, "PSL: A Semantic Domain for Flow Models," Software and Systems Modeling Journal, 2005

[2] BPDM specification, OMG group, 2008, [interaktyvus, žiūrėta 2010-03-20], prieiga per internetą: http://www.omg.org/spec/BMM/1.1/Beta2/PDF

[3] BPMN specification, OMG group, 2008, [interaktyvus, žiūrėta 2010-03-20], prieiga per internetą: http://www.bpmn.org/

[4] Business Process with BPEL4WS: Understanding BPEL4WS, [interaktyvus, žiūrėta 2008-02-21]. Prieiga per internetą: www.ibm.com/developerworks/library/ws-bpelcol1/

[5] J. Cabot, R. Pau and R. Ravents, "From uml/ocl to sbvr specifications: a challenging Transformation", Information Systems Elsevier, 2009

[6] M. Chisholm, „How to Build a Business Rules Engine", Morgan Kaufman Publishers, 2004

[7] O. Glassey, „A case study on process modelling - Three questions and three techniques", Decision Support Systems, v.44 n.4, pp. 842-853, March, 2008

[8] B. Von Halle, „Business Rules Applied: Building Better Systems Using the Business Rules Approach", John Wiley & Sons, Inc., Announces Quarterly Dividend, ISBN: 978-0-471-41293-9, 2001

[9] D. Hollingsworth, „The Workflow Reference Model 10 Years On" Fujitsu Services, United Kingdom Chair, Technical Committee, WfMC [interaktyvus, žiūrėta 2009-12-15] Prieiga per internetą: http://www.wfmc.org/Specifications-Working-Documents/General/View-category.html?limitstart=10

[10] A. Lindsay, D. Downs and K. Lunn, „Business processes—attempts to find a definition", Information and Software Technology n. 45, pp. 1015–1019, 2003

[11] G. Mentzas, C. Halaris and S. Kavadias, „ Modelling business processes with workflow systems: an evaluation of alternative approaches", International Journal of Information Management, vol. 21, pp. 123-135, 2001

[12] M. Muehlen, M. Indulska and G. Kamp, „Business process and business rule modeling languages for compliance management: a representational analysis", ACM International Conference Proceeding Series, vol. 334, 2007, ISBN ~ ISSN:1445-1336 , 978-1-920682-64-4, pp. 127-132, 2007

[13] OMG Model-Driven Architecture Home Page [interaktyvus, žiūrėta 2010 03 10]. Prieiga per internetą: www.omg.org/mda/index.htm.

[14] R. Paiano, A. L. Guido and A. Pandurino, "Designing Complex Web Information Systems: Integrating Evolutionary Process Engineering", ISBN 978-1-60566-300-5, 2009

[15] A. Rolstadås, „Business process modeling and reengineering", in Performance Management: A Business Process Benchmarking Approach, pp. 148-150, 1995

[16] T. Skersys, V. Pečiulis and R. Simutis, „Business rules specification using natural language-based templates: approach and implementation", Information Technologies' 2008 : proceedings of the 14th International Conference on Information and Software Technologies, Kaunas, Lithuania, ISSN 2029-0020, pp. 353-360, April, 2008

[17] L. Tutkutė, „Possibilities of business rules modeling and integration into the information systems", Research Reports on Business Informatics, Proceedings of the 8th MINE Workshop Methodologies for Interactive Networked Enterprises, Gdansk, pp. 29-34, September, 2008

[18] S. Williams, "Business Process Modeling Improves Administrative Control" in Automation, pp. 44 – 50, December, 1967

[19] Workflow Management Coalition, Glossary—A Workflow Management Coalition Specification, Brussels Belgium, 1994

4 priedas. Mokslinis straipsnis konferencijoje „Informacinės technologijos 2009“

TRANSFORMACIJA IŠ UML Į SBVR NAUDOJANT ATLAS KALBĄ ECLIPSE APLINKOJE

Edvinas Šinkevičius, Rimantas Butleris

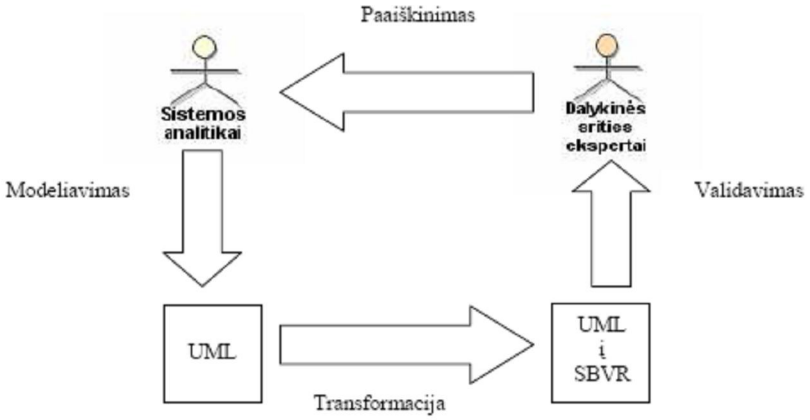
Kauno technologijos universitetas, Informacijos sistemų katedra, Studentų 50-308, Kaunas

Santrauka. UML kalba yra viena plačiausiai naudojamų modeliavimo kalbų specifikuojant informacines sistemas (IS). Tačiau ši modeliavimo kalba netinkama dalykinės srities ekspertams (veiklos atstovams) norintiems suprasti sumodeliuota informacija. Šiam tikslui galima panaudoti veiklos žodyno ir veiklos taisyklių semantiką (SBVR). Žinoma, unifikuota modeliavimo kalba (UML) ir SBVR nėra susietos, jas reikia suderinti – transformuoti. Šio straipsnio pagrindinis tikslas ir yra parodyti automatizuotą transformaciją iš UML į SBVR, kad modeliuotojai ir veiklos atstovai sąveikautu tarpusavyje remiantis suprojektuotu IS modeliu.

Raktiniai žodžiai: UML, SBVR, OCL, modelių transformacija

1 Įvadas

Sparčiai tobulėjant informacinėms technologijoms ir didėjant kuriamų sistemų sudėtingumui bei skaičiui, atsirado poreikis kuo daugiau automatizuoti IS kūrimo procesą. Automatizavimo problemai spręsti naudojami modeliais grindžiamo kūrimo (angl. Model Driven Development) metodai, kurių esmė – didesnis naudojamų abstrakcijų lygis, kai vietoj programavimo kalbų naudojamos modeliavimo kalbos (UML). Šiandien, unifikuota modeliavimo kalba (UML) yra viena plačiausiai naudojamų modeliavimo kalbų specifikuojant konceptualias schemas. UML schemos gali būti papildomos tekstinėmis išraiškomis, kurios aprašytu veiklos taisykles ir kurios negalėjo būti išreikštos grafiškai. Šios tekstinės išraiškos aprašomos objektinių reiškinių kalba (OCL).



1 pav. UML į SBVR transformacijos esmės paaiškinimas

Išleistas veiklos žodyno ir veiklos taisyklių (SBVR) standartas 2008[8] turėjo didelį postūmį veiklos taisyklėms. Šis standartas apibrėžia metamodelį, semantinėms veiklos taisyklėms, veiklos faktams ir veiklos žodynui dokumentuoti. SBVR tikslas apjungti veiklos konceptus taip pat veiklos taisykles ir užrašyti juos paprasta žmogiškąja kalba, kad probleminės srities ekspertai(įmonės atstovai) galėtu lengvai jas suprasti. Probleminės srities ekspertui sunku patikrinti sistemų analitiko sudarytas specifikacijas dėl nesuprantamos formalios specifikavimo kalbos (pvz., UML). Iš kitos pusės, analitikui sudėtinga patikrinti probleminės srities eksperto pateiktos informacijos pilnumą ir teisingumą dėl šiam tikslui dažniausiai naudojamos natūraliosios kalbos ypatumų (natūraliaja kalba pateikti teiginiai gali būti daugiareikšmiai, prieštaringi, nestruktūrizuoti ir pan.). Padėti jiems susikalbėti gali veiklos taisyklių specifikavimo kalba, kuri iš vienos pusės išreiškiama natūraliajai kalbai artimais sakiniais, iš kitos pusės, yra formali (tiksli ir vienareikšmiška), kad būtu lengvesnis informacijos apsikeitimas tarp organizacijų ir programinių įrankių. Veiklos taisyklės aprašytos remiantis SBVR standartu gali būti išreiškiamos daugelyje kalbų. Dabar yra tik keletas projektų kurie remiasi SBVR, vienas iš jų SbeaVeR – atviro kodo programa, kuri remiasi ankstesne SBVR standarto versija, išleista 2006 ir daugiau

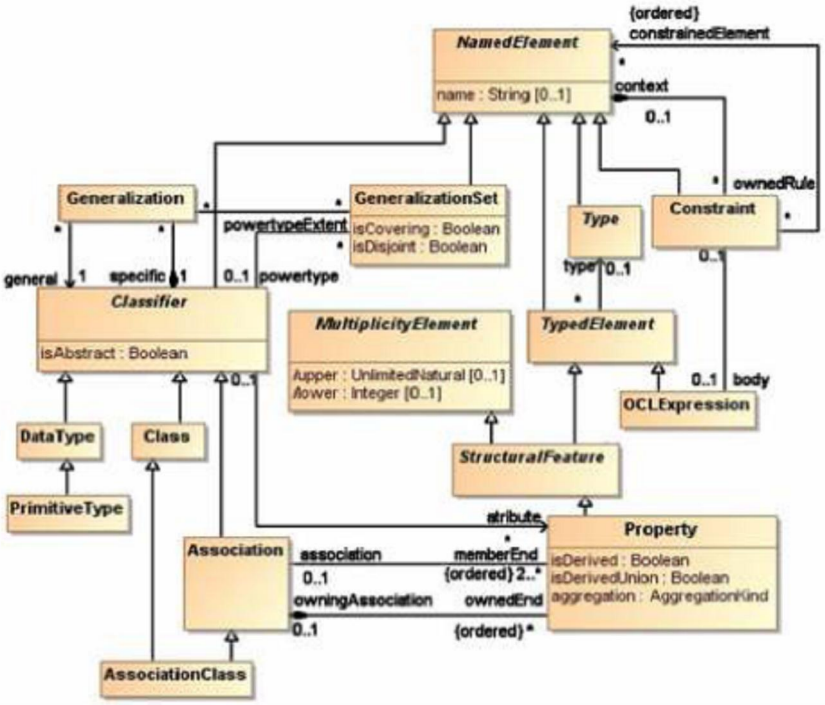
nebepalaikoma. Lietuvos projekte „Veiklos taisyklių sprendimai informacinių sistemų kūrimui (VeTIS)“[1] taip pat siekiam susieti veiklos žodyno ir veiklos taisyklių semantinį standartą.

Žinoma, unifikuota modeliavimo kalba (UML) ir SBVR nėra susietos, todėl UML reikia transformuoti į SBVR, kad probleminės srities ekspertas lengvai galėtų suprasti projektuotojo sumodeliuotą modelį. Todėl pagrindinė šio straipsnio mintis – aprašyti automatinės transformacijos iš UML į SBVR idėją (kuri aiškiau paaiškinta 1 pav.).

2 Transformavimo aprašymas

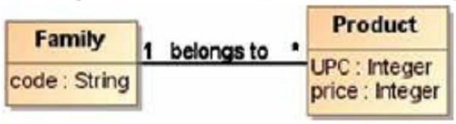
2.1 UML kalba

UML kalba formaliai remiasi UML metamodeliu kuris nusako abstrakčia kalbos sintaksę. Metamodelis nusako galimus elementus, ryšius, kurie gali būti atvaizduoti UML modelyje. Kaip pavyzdys supaprastintas UML metamodelis pateiktas 2 pav., kuris nusako pagrindinius elementus: klases, atributus, asociacijas, apibendrinimą.



2 pav. Ištrauka iš UML metamodelio

UML modelis, remiantis šiuo metamodeliu (2 pav.) pavaizduotas paprastoje schemoje 3 pav., kuriame pavaizduota *Product* klasė (kiekvienas produktas turi unikalų kodą, kainą) ir *Family* klasė (kuri turi tik kodą).

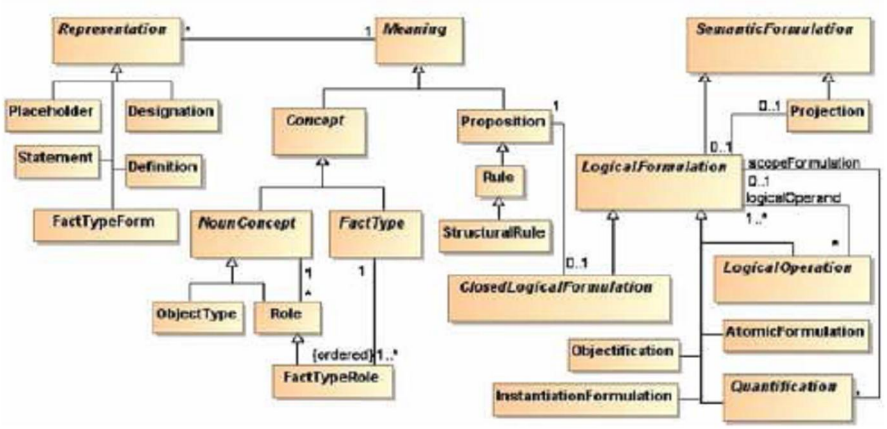


3 pav. UML modelis

[1] Projektas remiamas Lietuvos valstybinio mokslo ir studijų fondo vykdant „Aukštųjų technologijų plėtros“ programos projektą "VeTIS" (Reg.Nr. B-07042)

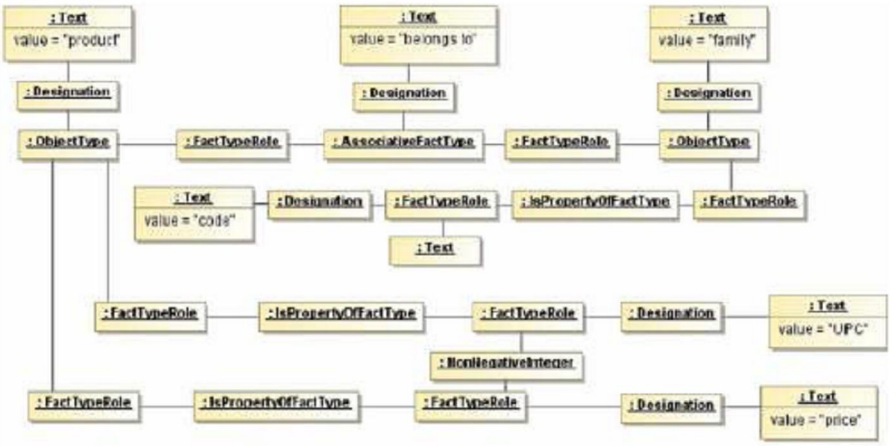
2.2 SBVR kalba

SBVR specifikacija (OMG-SBVR, 2008)[8] apibrėžia veiklos žodyno (*business vocabulary*), veiklos faktų (*business facts*) ir veiklos taisyklių (*business rules*) semantiką bei XMI schemą veiklos žodynų ir veiklos taisyklių apsikeitimui tarp organizacijų ar kompiuterizuotų sistemų. Labai svarbus SBVR akcentas yra tai, kad tiek veiklos terminų žodynai, tiek ir pačios veiklos taisyklės yra sudaromos naudojant natūralios kalbos konstrukcijas – tai yra ypač svarbu veiklos modeliavimo lygmenyje. Kaip pavyzdys, supaprastintas SBVR metamodelis pateiktas 4 pav.



4 pav. Ištrauka iš SBVR metamodelio

Iš aukščiau pateikto UML modeliu (3 pav.) buvo sudarytas SBVR modelis (5 pav.), kuris remiasi SBVR metamodeliu (4 pav.). Transformacijos metu UML klasės buvo transformuotos į SBVR objektų tipus (ObjectType) *family, product*.



5 pav. SBVR modelis transformuotas is UML modelio pateikto 3 pav.

2.3 Transformacija

Aukščiau pateiktos transformacijos (iš 3 pav. į 5 pav.) pavyzdys buvo vykdomas ATLAS (ATL) kalbos pagalba. ATL yra Eclipse.org Model-To-Model (M2M) projekto dalis. Tai yra modelių transformavimo kalba apibrėžta metamodeliais ir tekstine sintakse. Žemiau pateiktas ATL kalbos sintaksės fragmentas panaudotas automatizuotai transformacijai iš UML į SBVR (6 pav.).

```
rule ObjectType2Class {

    from
        s : MRV!ObjectType
    to
        t : UML!Class (

            name <- s.representation->first().text.value
        )

}
```

6 pav. ATL kalbos fragmentas panaudotas transformacijai iš UML į SBVR

Iš gautos transformacijos galima išreikšti taisykles artimas natūraliajai kalbai (7 pav.).

```
1 product
2 family
3 upc
4 code
5 product has upc
6 family has code
7 family has product
```

7 pav. SBVR taisyklės išreikštos natūralia kalba

3 Išvados

Šiame straipsnyje buvo aprašyta paprasta transformacija iš UML modelio į SBVR modelį, kuris supaprastina probleminės srities ekspertui, sistemų analitiko sudarytas specifikacijas dėl nesuprantamos formalios specifikavimo kalbos (UML). Iš kitos pusės, analitikui palengvėja probleminės srities eksperto pateiktos informacijos pilnumo ir teisingumo patikrinimas.

Literatūros sąrašas

[1] **Cabot J., Pau R., Ravento R.** UML-to-SBVR and SBVR-to-HTML transformations. Prieiga internete: www.lsi.upc.edu/~jcabsearch/SBVR.

[2] **Cabot J., Pau R., Ravento R.** From UML/OCL to SBVR specifications: A challenging transformation.

[3] **Gudas S., Skersys T. Lopata A.** Framework for knowledge–based IS engineering. In: Proceedings of third international conference „Advances in Information systems (ADVIS2004)", Izmir, Turkey, October 20-22, 2004, LNCS 3261, 2004, 512 – 522.

[4] **Kapocius K., Butleris R.** Repository for business rules based IS requirements. Informatica, 2006, Vol. 17, Nr. 4, 503–518.

[5] OMG Unified Modeling Language, Superstructure, V2.1.2. OMG formali specifikacija/2007-11-02, 2007.

[6] **Raj A., Prabhakar T. V., Hendryx S. Transformation of SBVR business design to UML models.** In: ISEC '08: Proceedings of the 1st conference on India software engineering conference, ACM, Hyderabad, India, 2008.

[7] **Schacher M. Business Rules from an SBVR and an xUML Perspective (Parts 1–3).** Veiklos taisyklių žurnalas, 2006, vol. 7, Nr. 6–8.

[8] Semantics of Business Vocabulary and Business Rules (SBVR), v1.0. OMG formali specifikacija/2008-01-02, 2008.

5 priedas. Veiklos žodynų transformacijos SBVR2UML specifikacija ATL kalboje

```
module VeTISSBVR2UML2; -- Module Template
create OUT : UML from IN : SBVR;


helper def: privateVisibilityValue:UML!VisibilityKind = #private;

helper def: publicVisibilityValue:UML!VisibilityKind = #public;

helper def: compositeAggregation:UML!AggregationKind = #composite;

helper def: returnDirection:UML!ParameterDirectionKind = #return;

helper def: trueValue:Boolean = true;

helper def: falseValue:Boolean = false;

helper def: voidValue:String = 'void';

helper def: zvaigzde:String='-1';

helper def: invCounter:Integer = 1;

helper def: preCounter:Integer = 1;

helper def:iteratorCounter:Integer=1;

helper def: unlimitedVariable:UML!UnlimitedNatural=
       thisModule.zvaigzde.toInteger();

-- This helper removes "_" signs from the sentence and capitalize each word

helper def: getNameWithNoSpace(s:SBVR!Text):String = s.split('_')->
       iterate(i; n: String = '' | n + i.substring(1, 1).toUpper() + i.substring(2,
i.size()));

-- This helper removes "_" signs from the sentence and capitalize each word, except first
word

helper def: getNameWithNoSpaceAtr(s:SBVR!Text):String =
       let text:SBVR!Text = thisModule.getNameWithNoSpace(s) in
            text.substring(1, 1).toLower() + text.substring(2, text.size());

helper def: factSymbolAllInstances:Set(SBVR!FactSymbol)=SBVR!FactSymbol.allInstances();

helper def: termAllInstances:Set(SBVR!Term)=SBVR!Term.allInstances();

helper def:
placeholderAllInstances:Set(SBVR!Placeholder)=SBVR!Placeholder.allInstances();

--SBVR2UML helpers

helper def: getMultiplicityValue (o:SBVR!Placeholder):SBVR!Quantification =
       SBVR!Quantification.allInstances()->select(q|
            not q.oclIsTypeOf(SBVR!UniversalQuantification)
                        and SBVR!RoleBinding.allInstances()
                                    ->exists(rb|rb.bindableTarget=q.variable
                        and q.scopeFormulation=rb.atomicFormulation
                                    and rb.factTypeRole=o.meaning))->asSequence()-
>first();

helper def: getLowerValue (o:SBVR!Placeholder):SBVR!Quantification =
       SBVR!AtLeastNQuantification.allInstances()->select(q|
            SBVR!RoleBinding.allInstances()->exists(rb|rb.bindableTarget=q.variable
                        and q.scopeFormulation=rb.atomicFormulation
                                    and rb.factTypeRole=o.meaning))->asSequence()-
>first();
```

```
helper def: getNumericValue (o:SBVR!Placeholder):SBVR!Quantification =
       SBVR!NumericRangeQuantification.allInstances()->select(q|
            SBVR!RoleBinding.allInstances()->exists(rb|rb.bindableTarget=q.variable
                        and q.scopeFormulation=rb.atomicFormulation
                                    and rb.factTypeRole=o.meaning))->asSequence()-
>first();

helper def: getUpperValue (o:SBVR!Placeholder):SBVR!Quantification =
       SBVR!AtMostNQuantification.allInstances()->select(q|
            SBVR!RoleBinding.allInstances()->exists(rb|rb.bindableTarget=q.variable
                        and q.scopeFormulation=rb.atomicFormulation
                                    and rb.factTypeRole=o.meaning))->asSequence()-
>first();

helper def: getUpperOrLowerValue(o:SBVR!Placeholder):SBVR!Quantification =
       SBVR!ExactlyNQuantification.allInstances()->select(q|
            SBVR!RoleBinding.allInstances()->exists(rb|rb.bindableTarget=q.variable
                        and q.scopeFormulation=rb.atomicFormulation
                                    and rb.factTypeRole=o.meaning))->asSequence()-
>first();

helper def: getFactSymbolValue(s:String):String = s.split(' ')->
       subSequence(2,2)->first();

helper def: isPlaceholderPreferred(pl:SBVR!Placeholder):Boolean=
       let sententialForm1:SBVR!SententialForm=
            SBVR!SententialForm.allInstances()->select(sf|
                        sf.designation->includes(pl))->asSequence()->first()in
       let sententialFormValue:String=sententialForm1.expression.value.toString()in
       let factSymbolValue:String=thisModule.getFactSymbolValue(sententialFormValue) in
            thisModule.factSymbolAllInstances->exists(fs|fs.preferred=true
                        and fs.meaning=sententialForm1.meaning
                                    and fs.expression.value.toString()=factSymbolValue);

helper def: getGeneralizations(o:SBVR!Term):Sequence(SBVR!Designation)=
       SBVR!Designation.allInstances()->
            select(c|c.meaning.oclIsTypeOf(SBVR!CategorizationFactType))->
                        select(cc|cc.meaning.role->first().objectType->
                  asSequence()->first()=o.meaning
                                    and o.preferred=true)->asSequence();

helper def : getPackagedElementsRef(): Sequence(SBVR!Designation)=
       thisModule.termAllInstances->select(e |e.preferred=true and
              (e.meaning.oclIsTypeOf(SBVR!ObjectType)
                        or e.meaning.oclIsTypeOf(SBVR!CategorizationType)))->union
                                    (thisModule.factSymbolAllInstances-
>select(e|e.preferred=true
                                                and
(e.meaning.oclIsTypeOf(SBVR!AssociativeFactType)
                         or e.meaning.oclIsTypeOf(SBVR!PartitiveFactType))))->
                                    union(SBVR!Name->allInstances()->select(ic|
                                    --          ic.preferred=true and
                        ic.refersTo->asSequence()->first().oclIsKindOf
                           (SBVR!CategorizationScheme)))->asSequence();

helper def: getOwnedRules():Sequence(SBVR!Designation)=
       thisModule.getInvariants()->union(thisModule.getPreconditions());

helper def: getInvariants():Sequence(SBVR!Designation)=
       SBVR!Statement.allInstances()->select(st|thisModule.isStructuralRule(st)or
            thisModule.isDerivationRule(st))->asSequence();

helper def: getOwnedAttributes(o:SBVR!Term):Sequence(SBVR!Placeholder)=
       let b:Sequence(SBVR!Placeholder)=
            SBVR!AssociativeFactType.allInstances()->select(pft|
                                    thisModule.factTypeIsAssociation(pft) and
                        pft.role->last().objectType->asSequence()->first()=o.meaning
```

```
                        and o.preferred=true )->
            collect(pf|pf.role->first())->asSequence()->
                        iterate(p;acc:Sequence(SBVR!Placeholder)=Sequence{}|
                                                    let
      a:Sequence(SBVR!Placeholder)=thisModule.placeholderAllInstances->
                        select(pl|pl.meaning=p)in acc->append(a))in

      let c:Sequence(SBVR!Placeholder)=
            SBVR!AssociativeFactType.allInstances()->select(pft|
                        thisModule.factTypeIsAssociation(pft) and
            pft.role->first().objectType->asSequence()->first()=o.meaning)->
                        collect(pf|pf.role->last())->asSequence()->
                                    iterate(p;acc:Sequence(SBVR!Placeholder)=Sequence{}|
                        let
a:Sequence(SBVR!Placeholder)=thisModule.placeholderAllInstances
                           ->select(pl|pl.meaning=p)in
            acc->append(a))in

      let d:Sequence(SBVR!Placeholder)=
            SBVR!IsPropertyOfFactType.allInstances()->select(pft|
                        pft.role->last().objectType->asSequence()->first()=o.meaning)->
                                    collect(pf|pf.role->first())->asSequence()->

      iterate(p;acc:Sequence(SBVR!Placeholder)=Sequence{}|
                        let
a:Sequence(SBVR!Placeholder)=thisModule.placeholderAllInstances
                           ->select(pl|pl.meaning=p)in
            acc->append(a))in

      let e:Sequence(SBVR!Placeholder)=
            SBVR!PartitiveFactType.allInstances()->select(pft|
                        pft.role->last().objectType->asSequence()->first()=o.meaning)->
                                    collect(pf|pf.role->first())->asSequence()->
                                    iterate(p;acc:Sequence(SBVR!Placeholder)=Sequence{}|
                        let
a:Sequence(SBVR!Placeholder)=thisModule.placeholderAllInstances
                           ->select(pl|pl.meaning=p)in
            acc->append(a))in

      let f:Sequence(SBVR!Placeholder)=
            SBVR!PartitiveFactType.allInstances()->select(pft|
                        pft.role->first().objectType->asSequence()->first()=o.meaning)->
                                    collect(pf|pf.role->last())->asSequence()->
                        iterate(p;acc:Sequence(SBVR!Placeholder)=Sequence{}|
            let a:Sequence(SBVR!Placeholder)=thisModule.placeholderAllInstances
                           ->select(pl|pl.meaning=p)in
            acc->append(a))in

      let g:Sequence(SBVR!Placeholder)=SBVR!Characteristic.allInstances()
            ->select(pft|
                        pft.role->last().objectType->asSequence()->first()=o.meaning)->
                                    collect(pf|pf.role->first())->asSequence()->

      iterate(p;acc:Sequence(SBVR!Placeholder)=Sequence{}|
                                                    let
      a:Sequence(SBVR!Placeholder)=thisModule.placeholderAllInstances
            ->select(pl|pl.meaning=p)in
                        acc->append(a))

            in d->union(b)->union(c)->union(e)->union(f)->union(g);

helper def: factTypeIsOperation(o:SBVR!AssociativeFactType):Boolean=
      SBVR!AtomicFormulation.allInstances()->exists(af|af.factType=o and
            SBVR!ModalFormulation.allInstances()->exists(mf|
                        (mf.oclIsTypeOf(SBVR!ObligationFormulation)or
                                    mf.oclIsTypeOf(SBVR!PermissibilityFormulation))and
                                                    if
mf.logicalFormulation.oclIsTypeOf(SBVR!UniversalQuantification)then
```

```
                                                    if
mf.logicalFormulation.scopeFormulation.oclIsTypeOf(SBVR!AtomicFormulation)then
            mf.logicalFormulation.scopeFormulation=af
                                                    else if
mf.logicalFormulation.scopeFormulation.oclIsTypeOf(SBVR!Implication)then
            mf.logicalFormulation.scopeFormulation.consequent=af
                        else false endif endif else false endif));

helper def: factTypeIsAssociation(o:SBVR!AssociativeFactType):Boolean=
      not thisModule.factTypeIsOperation(o) or
            SBVR!AtomicFormulation.allInstances()->exists(af|af.factType=o and
                        SBVR!ModalFormulation.allInstances()->exists(mf|
                                    (mf.oclIsTypeOf(SBVR!NecessityFormulation) or

      mf.oclIsTypeOf(SBVR!PossibilityFormulation)) and
                                                          if
mf.logicalFormulation.oclIsTypeOf(SBVR!UniversalQuantification)then
--if mf.logicalFormulation.scopeFormulation=af then true else
                                    if
mf.logicalFormulation.scopeFormulation.oclIsKindOf(SBVR!Quantification) then
                        mf.logicalFormulation.scopeFormulation.scopeFormulation=af else
false endif else false endif));

helper def: getOwnedOperations(o:SBVR!Term):Sequence(SBVR!AssociativeFactType)=
      SBVR!AssociativeFactType.allInstances()->select(ft|
            ft.role->last().objectType->asSequence()->first()=o.meaning)->
                        asSequence()->
                  iterate(p;acc:Sequence(SBVR!AssociativeFactType)=Sequence{}|
                                    if thisModule.factTypeIsOperation(p)then acc-
>append(p)
                                                    else acc endif)->asSequence();

helper def: getMemberEnds(s:SBVR!FactSymbol):Sequence(SBVR!Placeholder)=
      thisModule.placeholderAllInstances->select(r|
            r.meaning=s.meaning.role->first())->asSequence()->
      union(thisModule.placeholderAllInstances->select(r|r.meaning=s.meaning.role
                  ->last())->asSequence()
                                    );

helper def:
categorizationSchemeNameOfCategorizationFactSymbol(fs:SBVR!FactSymbol):SBVR!Name=
      let category1:SBVR!ObjectType=fs.meaning.role->first().objectType->asSequence()
            ->first() in
                        SBVR!Name.allInstances()->select(na|
                                    na.refersTo->asSequence()->
                        first().oclIsKindOf(SBVR!CategorizationScheme))->
                                    select(na|na.refersTo->asSequence()-
>first().containsCategory
                           ->includes(category1))->asSequence()->first();

helper def:
isCategorizationFactTypeWithCategorizationScheme(cf:SBVR!FactSymbol):Boolean=
            not
thisModule.categorizationSchemeNameOfCategorizationFactSymbol(cf).oclIsUndefined();


--SBVR2UML rules


rule Vocabulary2Package {

from

s : SBVR!Name(s.meaning.oclIsTypeOf(SBVR!IndividualConcept)
      and s.refersTo->asSequence()->first().oclIsTypeOf(SBVR!Vocabulary))

to
```

```
t : UML!Model (

name <- 'Data'
,
viewpoint <- ''
,
ownedComment <- Sequence {author}
,
packagedElement <- Sequence {pack}
)
,
author: UML!Comment (
body <- 'Author:Created by VeTIS'
)
,
pack: UML!Package (

name <- thisModule.getNameWithNoSpace(s.expression.value)
,
packagedElement <- thisModule.getPackagedElementsRef()->including(constraintPackage)

)
,
constraintPackage:UML!Package(
       name<-'Constraints'
       ,
       ownedRule<-thisModule.getOwnedRules()
)
}

rule Term2PrimitiveType {

       from
            s:SBVR!Term(s.meaning.oclIsTypeOf(SBVR!ObjectType)
                        and s.preferred=true
                                    and (s.expression.value='integer'
                                                or s.expression.value='text'
                                                            or
s.expression.value='number'
                                                                        or
s.expression.value='null'
                                                                                    or
s.expression.value='boolean'))
                        to
            t : UML!PrimitiveType (
                        name <- if s.expression.value='integer'then 'Integer'
                                    else if s.expression.value='text'then 'String'
                                                else if s.expression.value='number' then
                                                            'UnlimitedNatural'
                                                else if s.expression.value='null' then
'void'
                                    else 'Boolean' endif endif endif endif

                        )
            }


rule Placeholder2OwnedAttribute {

       from

       s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
            SBVR!IsPropertyOfFactType.allInstances()->exists(a|a.role
                        ->first()=s.meaning)
       )

       to
```

```
        t : UML!Property (
        name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
        ,
        visibility <- thisModule.privateVisibilityValue
        ,
        type <- thisModule.termAllInstances->
             select(tt|tt.preferred=true and
                         tt.meaning=s.meaning.objectType->asSequence()->first())
                              ->asSequence()->first()
             )
        }


rule Placeholder2OwnedAssociation {

        from

        s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)
             and thisModule.isPlaceholderPreferred(s)and
                         thisModule.getMultiplicityValue(s).oclIsUndefined()
                              and (thisModule.factSymbolAllInstances->select
                                        (a|a.preferred=true and

        a.meaning.oclIsTypeOf(SBVR!AssociativeFactType)
                                                            and
        thisModule.factTypeIsAssociation(a.meaning))->
             exists(b|b.meaning.role->asSequence()
                         ->includes(s.meaning))or
                              (thisModule.factSymbolAllInstances->select
                                        (a|a.preferred=true and
                              a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                         exists(b|b.meaning.role->last()=s.meaning)))
                         )

        to

        t : UML!Property (

        name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
        ,
        visibility <- thisModule.privateVisibilityValue
        ,
        type <-
             thisModule.termAllInstances->select(tt|tt.preferred=true and
                         tt.meaning=s.meaning.objectType->asSequence()->
                              first())->asSequence()->first()
        ,
        association <- thisModule.factSymbolAllInstances->select
             (a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!AssociativeFactType)
                         and thisModule.factTypeIsAssociation(a.meaning))->
                              select(b|b.meaning.role->asSequence()-
>includes(s.meaning))
                                                  ->asSequence()->first()
        ,
        upperValue <- upperValueDefault
        ,
        lowerValue <- lowerValueDefault
        )
        ,

        upperValueDefault:UML!LiteralUnlimitedNatural
             (name<-'',
                         value<-thisModule.unlimitedVariable
             )
             ,
        lowerValueDefault:UML!LiteralInteger (
             name<-'',
             value<-0
```

```
            )
      }

rule Placeholder2OwnedAssociationWithCardinalities {

      from

      s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)
            and thisModule.isPlaceholderPreferred(s)
                        and not thisModule.getLowerValue(s).oclIsUndefined()
                              and not thisModule.getUpperValue(s).oclIsUndefined()
and
                        (thisModule.factSymbolAllInstances->select
                              (a|a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                        exists(b|b.meaning.role->asSequence()->includes(s.meaning))or
            (thisModule.factSymbolAllInstances->select
                        (a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                              exists(b|b.meaning.role->last()=s.meaning)))
                              )

to

t : UML!Property (

      name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
            ,
      visibility <- thisModule.privateVisibilityValue
            ,
      type <-
            thisModule.termAllInstances->select(tt|tt.preferred=true and
                        tt.meaning=s.meaning.objectType->asSequence()->first())
                              ->asSequence()->first()
            ,
      association <- thisModule.factSymbolAllInstances->select
            (a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                        select(b|b.meaning.role->includes(s.meaning))->asSequence()-
>first()
            ,
      upperValue <- upperValue1
      ,
      lowerValue <- lowerValue1
            )
            ,
      upperValue1:UML!LiteralUnlimitedNatural (
            name <-'',
            value <-
thisModule.getUpperValue(s).maximumCardinality.value.toString().toInteger()
            )
            ,
      lowerValue1:UML!LiteralInteger (
      name <- '',
            value <-
            thisModule.getLowerValue(s).minimumCardinality.value.toString().toInteger()
            )
}

rule Placeholder2ownedAssociationWithExactCardinalities {

      from

      s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
            thisModule.isPlaceholderPreferred(s) and
                        not thisModule.getUpperOrLowerValue(s).oclIsUndefined()
                              and (thisModule.factSymbolAllInstances->select

      (a|a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                              exists(b|b.meaning.role->asSequence()-
>includes(s.meaning))or
```

```
                        (thisModule.factSymbolAllInstances->select
                                    (a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                                                exists(b|b.meaning.role-
>last()=s.meaning)))
                        )

        to

        t : UML!Property (

        name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
        ,
        visibility <- thisModule.privateVisibilityValue
        ,
        type <-
             thisModule.termAllInstances->select(tt|tt.preferred=true and
                        tt.meaning=s.meaning.objectType->asSequence()->first())
                                    ->asSequence()->first()
        ,
        association <- thisModule.factSymbolAllInstances->select
             (a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                        select(b|b.meaning.role->asSequence()->includes(s.meaning))
                                    ->asSequence()->first()
        ,
        upperValue <- upperValue1
        ,
        lowerValue <- lowerValue1)
        ,
        upperValue1:UML!LiteralUnlimitedNatural (
             name <-'',
             value <-
        thisModule.getUpperOrLowerValue(s).cardinality.value.toString().toInteger()
             )
        ,
        lowerValue1:UML!LiteralInteger (
        name <- '',
             value <-
        thisModule.getUpperOrLowerValue(s).cardinality.value.toString().toInteger()
        )
}

rule Placeholder2OwnedAssociationWithNumericCardinalities {

        from

        s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
             thisModule.isPlaceholderPreferred(s) and
                        not thisModule.getNumericValue(s).oclIsUndefined()
                                    and (thisModule.factSymbolAllInstances->select

        (a|a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                                    exists(b|b.meaning.role->asSequence()-
>includes(s.meaning))or
                        (thisModule.factSymbolAllInstances->select
                                    (a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                                                exists(b|b.meaning.role-
>last()=s.meaning)))
                        )

        to

        t : UML!Property (

        name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
        ,
        visibility <- thisModule.privateVisibilityValue
        ,
        type <-
```

```
            thisModule.termAllInstances->select(tt|tt.preferred=true and
                        tt.meaning=s.meaning.objectType->asSequence()->
                              first())->asSequence()->first()
      ,
      association <- thisModule.factSymbolAllInstances->select
            (a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                        select(b|b.meaning.role->asSequence()->includes(s.meaning))
                              ->asSequence()->first()
      ,
      upperValue <- upperValue1
      ,
      lowerValue <- lowerValue1)
      ,
      upperValue1:UML!LiteralUnlimitedNatural (
            name <-'',
            value <-
      thisModule.getNumericValue(s).maximumCardinality.value.toString().toInteger()
            )
            ,
      lowerValue1:UML!LiteralInteger (
      name <- '',
            value <-
      thisModule.getNumericValue(s).minimumCardinality.value.toString().toInteger()
      )
}

rule Placeholder2OwnedAssociationWithLowerCardinalities {

      from

      s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
            thisModule.isPlaceholderPreferred(s) and
                        not thisModule.getLowerValue(s).oclIsUndefined()
                              and thisModule.getUpperValue(s).oclIsUndefined()
                        and (thisModule.factSymbolAllInstances->select
                              (a|a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                                    exists(b|b.meaning.role->asSequence()
                                          ->includes(s.meaning))or
                                    (thisModule.factSymbolAllInstances-
>select
                              (a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                        exists(b|b.meaning.role->includes(s.meaning))))
            )

      to

      t : UML!Property (

      name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
      ,
      visibility <- thisModule.privateVisibilityValue
      ,
      type <-
            thisModule.termAllInstances->select(tt|tt.preferred=true and
      tt.meaning=s.meaning.objectType->asSequence()->first())->asSequence()->first()
      ,
      association <- thisModule.factSymbolAllInstances->select
            (a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                        select(b|b.meaning.role->asSequence()->includes(s.meaning))
                              ->asSequence()->first()
      ,
      upperValue <- upperValueDefault
      ,
      lowerValue <- lowerValue1
            )
      ,
      upperValueDefault:UML!LiteralUnlimitedNatural
            (           name<-'',
```

```
                        value<-thisModule.unlimitedVariable
            )
      ,
      lowerValue1:UML!LiteralInteger (
            name <- '',
            value <-
      thisModule.getLowerValue(s).minimumCardinality.value.toString().toInteger()
            )
      }

rule Placeholder2OwnedAssociationWithUpperCardinalities {

      from

      s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
            thisModule.isPlaceholderPreferred(s) and
                        not thisModule.getUpperValue(s).oclIsUndefined()
                              and thisModule.getLowerValue(s).oclIsUndefined()
                        and (thisModule.factSymbolAllInstances->select
                              (a|a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                                    exists(b|b.meaning.role->asSequence()
                                          ->includes(s.meaning))or
                                    (thisModule.factSymbolAllInstances-
>select
                              (a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                        exists(b|b.meaning.role->last()=s.meaning)))
            )

      to

      t : UML!Property (

      name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
      ,
      visibility <- thisModule.privateVisibilityValue
      ,
      type <-
            thisModule.termAllInstances->select(tt|tt.preferred=true and
                        tt.meaning=s.meaning.objectType->asSequence()->first())
                              ->asSequence()->first()
      ,
      association <- thisModule.factSymbolAllInstances->select
            (a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!AssociativeFactType))->
                        select(b|b.meaning.role->asSequence()->includes(s.meaning))
                              ->asSequence()->first()
      ,
      upperValue <- upperValue1
      ,
      lowerValue <- lowerValueDefault)
      ,
      upperValue1:UML!LiteralUnlimitedNatural (
            name <-'',
            value <-
      thisModule.getUpperValue(s).maximumCardinality.value.toString().toInteger()
            )
            ,
      lowerValueDefault:UML!LiteralInteger (
            name<-'',
            value<-0
            )
      }

rule Placeholder2Characteristic {

      from

      s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
            thisModule.factSymbolAllInstances->select
```

```
                        (a|a.preferred=true and
                                    a.meaning.oclIsTypeOf(SBVR!Characteristic))->
                        exists(b|b.meaning.role->asSequence()->first()=s.meaning))

        to

        t : UML!Property (


        name <- let ftr:SBVR!FactSymbol=thisModule.factSymbolAllInstances->
              select(ch|ch.preferred=true and
                                    ch.meaning.oclIsTypeOf(SBVR!Characteristic))->
                                          select(bc|bc.meaning.role-
>first()=s.meaning)
                                    ->asSequence() in
                        thisModule.getNameWithNoSpaceAtr(ftr->first().expression.value)
        ,
        visibility <- thisModule.privateVisibilityValue
        ,
        type <- thisModule.termAllInstances->
              select(tt|tt.preferred=true and
                        tt.expression.value='boolean')->asSequence()->first()
              )
        }

rule Placeholder2ownedCompositeAssociation {

        from

        s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
              thisModule.isPlaceholderPreferred(s)and
                        thisModule.getMultiplicityValue(s).oclIsUndefined()
                                    and thisModule.factSymbolAllInstances->select

        (a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                                                            exists(b|b.meaning.role-
>first()=s.meaning)
                                    )

        to

        t : UML!Property (

        name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
        ,
        visibility <- thisModule.privateVisibilityValue
        ,
        type <-
              thisModule.termAllInstances->select(tt|tt.preferred=true and
        tt.meaning=s.meaning.objectType->asSequence()->first())->asSequence()->first()
        ,
        aggregation <- thisModule.compositeAggregation
        ,
        association <- thisModule.factSymbolAllInstances->select
              (a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                        select(b|b.meaning.role->asSequence()->includes(s.meaning))
                                    ->asSequence()->first()
        ,
        upperValue <- upperValueDefault
        ,
        lowerValue <- lowerValueDefault
              )
        ,

        upperValueDefault:UML!LiteralUnlimitedNatural
              (name<-'',
                        value<-thisModule.unlimitedVariable
              )
```

```
	,
	lowerValueDefault:UML!LiteralInteger (
		name<-'',
		value<-0
	)
}

rule Placeholder2ownedCompositeAssociationWithCard {

	from

	s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
		thisModule.isPlaceholderPreferred(s) and
			not thisModule.getLowerValue(s).oclIsUndefined()
				and not thisModule.getUpperValue(s).oclIsUndefined()
					and thisModule.factSymbolAllInstances-
>select
				(a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
					exists(b|b.meaning.role-
>first()=s.meaning)
					)

	to

	t : UML!Property (

	name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
	,
	visibility <- thisModule.privateVisibilityValue
	,
	type <-
		thisModule.termAllInstances->select(tt|tt.preferred=true and
			tt.meaning=s.meaning.objectType->asSequence()->first())
				->asSequence()->first()
	,
	aggregation <- thisModule.compositeAggregation
	,
	association <- thisModule.factSymbolAllInstances->select
		(a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
			select(b|b.meaning.role->asSequence()->includes(s.meaning))
				->asSequence()->first()
	,
	upperValue <- upperValue1
	,
	lowerValue <- lowerValue1
		)
	,
	upperValue1:UML!LiteralUnlimitedNatural (
		name <-'',
		value <-
	thisModule.getUpperValue(s).maximumCardinality.value.toString().toInteger()
		)
	,
	lowerValue1:UML!LiteralInteger (
	name <- '',
		value <-
thisModule.getLowerValue(s).minimumCardinality.value.toString().toInteger()
	)
}

rule Placeholder2ownedCompositeAssociationWithExactCard {

	from

	s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
		thisModule.isPlaceholderPreferred(s) and
			not thisModule.getUpperOrLowerValue(s).oclIsUndefined()
				and thisModule.factSymbolAllInstances->select
```

```
	(a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                                                        exists(b|b.meaning.role-
>first()=s.meaning)
                                    )

	to

	t : UML!Property (

	name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
	,
	visibility <- thisModule.privateVisibilityValue
	,
	type <-
		thisModule.termAllInstances->select(tt|tt.preferred=true and
				tt.meaning=s.meaning.objectType->asSequence()->first())
		->asSequence()->first()
	,
	aggregation <- thisModule.compositeAggregation
	,
	association <- thisModule.factSymbolAllInstances->select
		(a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
				select(b|b.meaning.role->asSequence()->includes(s.meaning))
					->asSequence()->first()
	,
	upperValue <- upperValue1
	,
	lowerValue <- lowerValue1)
	,
	upperValue1:UML!LiteralUnlimitedNatural (
		name <-'',
				value <-
	thisModule.getUpperOrLowerValue(s).cardinality.value.toString().toInteger()
		)
	,
	lowerValue1:UML!LiteralInteger (
	name <- '',
		value <-
	thisModule.getUpperOrLowerValue(s).cardinality.value.toString().toInteger()
	)
}

rule Placeholder2ownedCompositeAssociationWithNumericCard {

	from

	s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)and
		thisModule.isPlaceholderPreferred(s) and
				not thisModule.getNumericValue(s).oclIsUndefined()
					and thisModule.factSymbolAllInstances->select

	(a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
                                                        exists(b|b.meaning.role-
>first()=s.meaning)
                                    )

	to
	t : UML!Property (

	name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
	,
	visibility <- thisModule.privateVisibilityValue
	,
	type <-
		thisModule.termAllInstances->select(tt|tt.preferred=true and
				tt.meaning=s.meaning.objectType->asSequence()->first())
		->asSequence()->first()
```

```
	,
	aggregation <- thisModule.compositeAggregation
	,
	association <- thisModule.factSymbolAllInstances->select
		(a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
				select(b|b.meaning.role->asSequence()->includes(s.meaning))
					->asSequence()->first()
	,
	upperValue <- upperValue1
	,
	lowerValue <- lowerValue1)
	,
	upperValue1:UML!LiteralUnlimitedNatural (
		name <-'',
		value <-
	thisModule.getNumericValue(s).maximumCardinality.value.toString().toInteger()
		)
	,
	lowerValue1:UML!LiteralInteger (
		name <- '',
		value <-
	thisModule.getNumericValue(s).minimumCardinality.value.toString().toInteger()
	)
}

rule Placeholder2ownedCompositeAssociationWithUpperCard {

	from

	s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole) and
		thisModule.isPlaceholderPreferred(s) and
				thisModule.getLowerValue(s).oclIsUndefined()
					and not thisModule.getUpperValue(s).oclIsUndefined()
				and thisModule.factSymbolAllInstances->select
					(a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
						exists(b|b.meaning.role-
>first()=s.meaning)
				)

	to

	t : UML!Property (

	name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
	,
	visibility <- thisModule.privateVisibilityValue
	,
	type <-
		thisModule.termAllInstances->select(tt|tt.preferred=true and
	tt.meaning=s.meaning.objectType->asSequence()->first())->asSequence()->first()
	,
		aggregation <- thisModule.compositeAggregation
	,
	association <- thisModule.factSymbolAllInstances->select
		(a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
				select(b|b.meaning.role->asSequence()->includes(s.meaning))
					->asSequence()->first()
	,
	upperValue <- upperValue1
	,
	lowerValue <- lowerValueDefault
		)
	,
	upperValue1:UML!LiteralUnlimitedNatural (
		name <-'',
		value <-
	thisModule.getUpperValue(s).maximumCardinality.value.toString().toInteger()
		)
```

```
	,
	lowerValueDefault:UML!LiteralInteger (
		name<-'',
		value<-0
	)
}

rule Placeholder2ownedCompositeAssociationWithLowerCard {

	from

	s : SBVR!Placeholder(s.meaning.oclIsTypeOf(SBVR!FactTypeRole)
		and thisModule.isPlaceholderPreferred(s)
				and not thisModule.getLowerValue(s).oclIsUndefined()
					and thisModule.getUpperValue(s).oclIsUndefined()
						and thisModule.factSymbolAllInstances-
>select
					(a|a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
				exists(b|b.meaning.role->first()=s.meaning)

					)

	to

	t : UML!Property (

	name <- thisModule.getNameWithNoSpaceAtr(s.expression.value)
	,
	visibility <- thisModule.privateVisibilityValue
	,
	type <-
		thisModule.termAllInstances->select(tt|tt.preferred=true and
	tt.meaning=s.meaning.objectType->asSequence()->first())->asSequence()->first()
	,
		aggregation <- thisModule.compositeAggregation
	,
	association <- thisModule.factSymbolAllInstances->select
		(a|a.preferred=true and a.meaning.oclIsTypeOf(SBVR!PartitiveFactType))->
				select(b|b.meaning.role->asSequence()->includes(s.meaning))
					->asSequence()->first()
	,
	upperValue <- upperValueDefault
	,
	lowerValue <- lowerValue1
		)
	,
	upperValueDefault:UML!LiteralUnlimitedNatural
		(		name<-'',
				value<-thisModule.unlimitedVariable
				)
				,
	lowerValue1:UML!LiteralInteger (
		name <- '',
		value <-
	thisModule.getNumericValue(s).minimumCardinality.value.toString().toInteger()
	)
}


rule FactSymbol2Association {

	from
		s : SBVR!FactSymbol (s.preferred=true and
				(s.meaning.oclIsTypeOf(SBVR!AssociativeFactType)
					and thisModule.factTypeIsAssociation(s.meaning)or

	s.meaning.oclIsTypeOf(SBVR!PartitiveFactType)))
	to
```

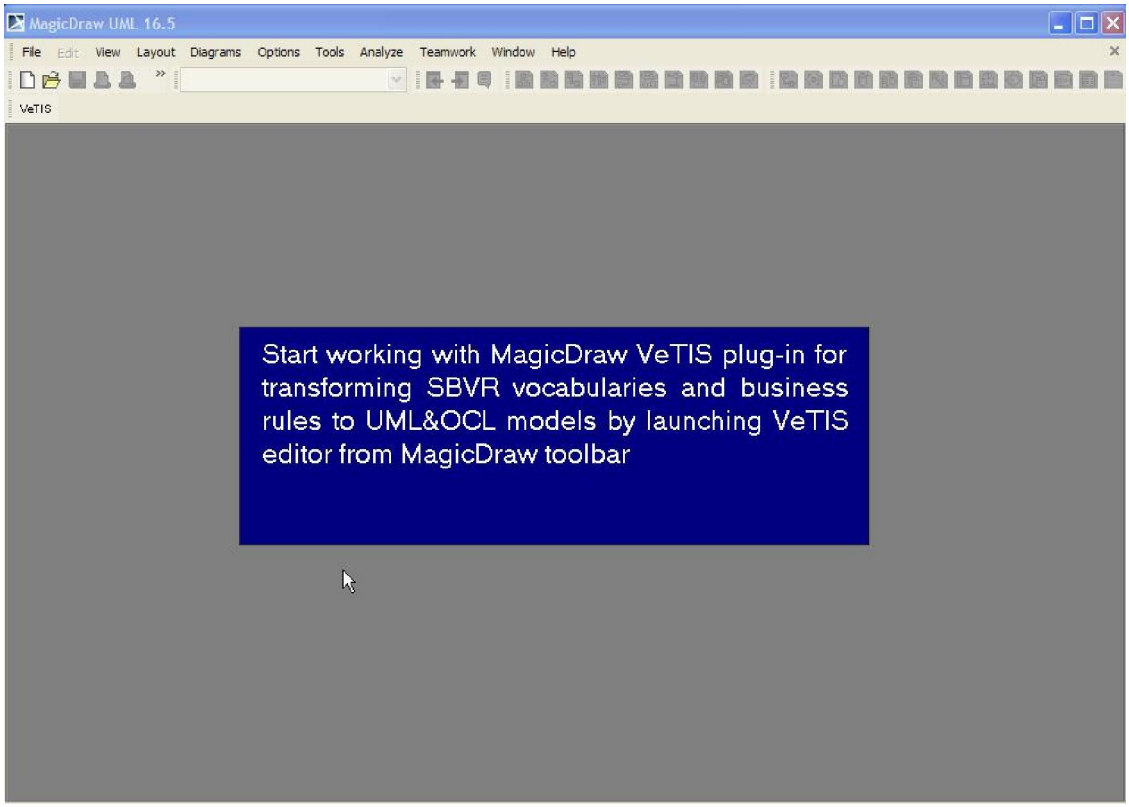
```
            t : UML!Association (
                        name <- thisModule.getNameWithNoSpaceAtr(s.expression.value),
                                    memberEnd <- thisModule.getMemberEnds(s)
      )
}


rule CategorizationFactSymbol2Generalization {

      from

      s:SBVR!FactSymbol(s.preferred=true and
            s.meaning.oclIsTypeOf(SBVR!CategorizationFactType) and not
                        thisModule.isCategorizationFactTypeWithCategorizationScheme(s)
                        )
            to
            t:UML!Generalization (

            isSubstitutable <- thisModule.trueValue
            ,
            general<-thisModule.termAllInstances->select(ss|ss.preferred=true
                        and ss.meaning.oclIsTypeOf(SBVR!ObjectType)
                                    and s.meaning.role->last().objectType->asSequence()->
                                                first()=ss.meaning)->asSequence()-
>first()
      )
}

rule CategorizationFactSymbol2GeneralizationWithGeneralizationSet {

      from

            s:SBVR!FactSymbol(s.preferred=true and
      s.meaning.oclIsTypeOf(SBVR!CategorizationFactType)
            and thisModule.isCategorizationFactTypeWithCategorizationScheme(s))

      to

      t:UML!Generalization (

            isSubstitutable <- thisModule.trueValue
            ,
            general<-thisModule.termAllInstances->select(ss|ss.preferred=true
                        and ss.meaning.oclIsTypeOf(SBVR!ObjectType)
            and s.meaning.role->last().objectType->asSequence()->
                        first()=ss.meaning)->asSequence()->first()
            ,
            generalizationSet <-
                        thisModule.categorizationSchemeNameOfCategorizationFactSymbol(s)
      )
}

rule CategorizationSchemeName2GeneralizationSet {

      from

      s:SBVR!Name(
            --s.preferred=true and
            s.refersTo->asSequence()->first().oclIsKindOf(SBVR!CategorizationScheme))

      to

      t:UML!GeneralizationSet(

      name<-thisModule.getNameWithNoSpace(s.expression.value)
      ,
      isCovering<-
      if s.refersTo->asSequence()->first().oclIsTypeOf(SBVR!Segmentation) then
```

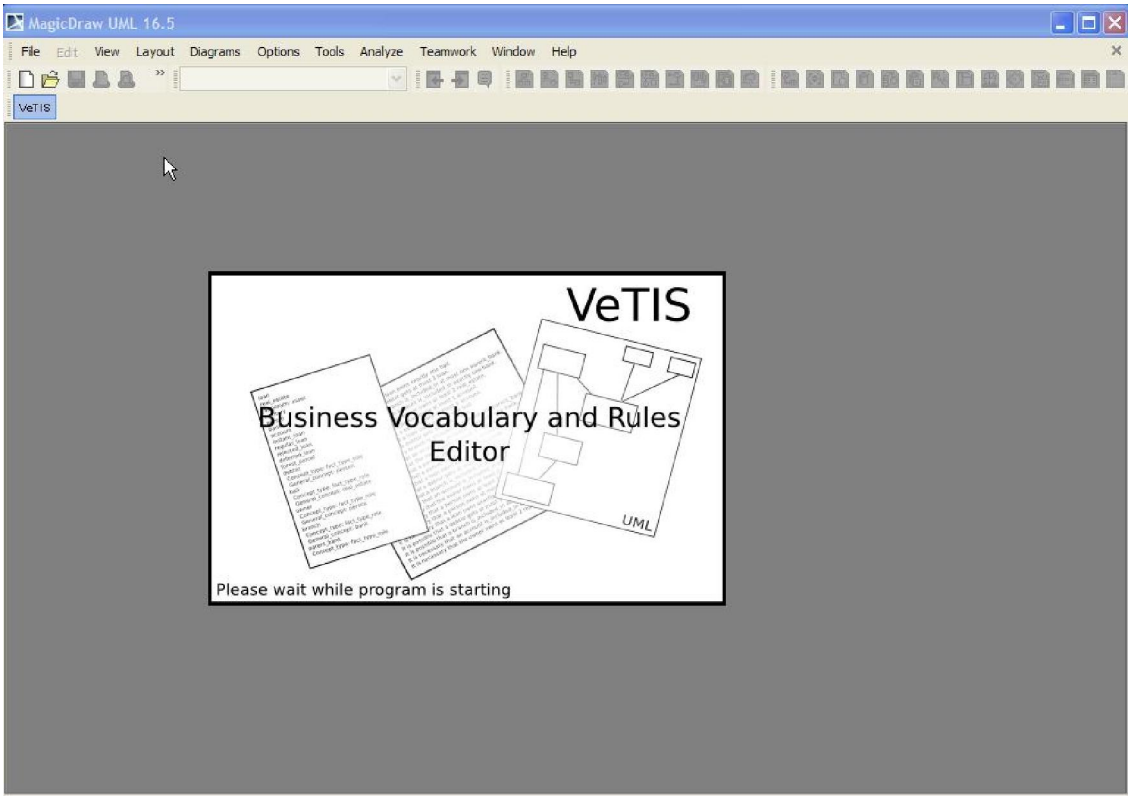
```
            thisModule.trueValue else
                        thisModule.falseValue endif
      ,
      isDisjoint<-thisModule.trueValue
      ,
      powertype<-thisModule.termAllInstances->select(te|te.preferred=true and
            s.refersTo->asSequence()->first().isBasedOn=te.meaning)->
                        asSequence()->first()
      )
      }


rule Statement2StructuralRule {

      from

      s :   SBVR!Statement (thisModule.isStructuralRule(s)
            )


      to

      t : UML!Constraint(
            name <- 'inv'+thisModule.className(s)+thisModule.invCounter.toString()
            ,
            constrainedElement <- thisModule.constrainedElem(s)
            ,
            specification <- ruleSpecification
            )
            ,
            ruleSpecification:UML!OpaqueExpression (
                        name<-'',
                        language<-'OCL2.0',
                        body<-thisModule.bodyOclText(thisModule.centralFormulation(s))
            )
            do {
            thisModule.invCounter <- thisModule.invCounter + 1;
            }
      }

rule StatementWithDeonticImplication2Precondition {

      from

      s :   SBVR!Statement(
            thisModule.isPrecondition(s))

      using
{logicalFormulation1:SBVR!LogicalFormulation=thisModule.centralFormulation(s).antecedent;
            }


      to

      t : UML!Constraint(
            name <- 'pre'+thisModule.opName(s)+thisModule.preCounter.toString()
            ,
            constrainedElement <- Sequence{thisModule.baseFactType(s)}
            ,
            specification <- ruleSpecification
            )
            ,
            ruleSpecification:UML!OpaqueExpression (
                        name<-'',
                        language<-'OCL2.0',
                        body<-thisModule.bodyOclText(logicalFormulation1)
                        )
```

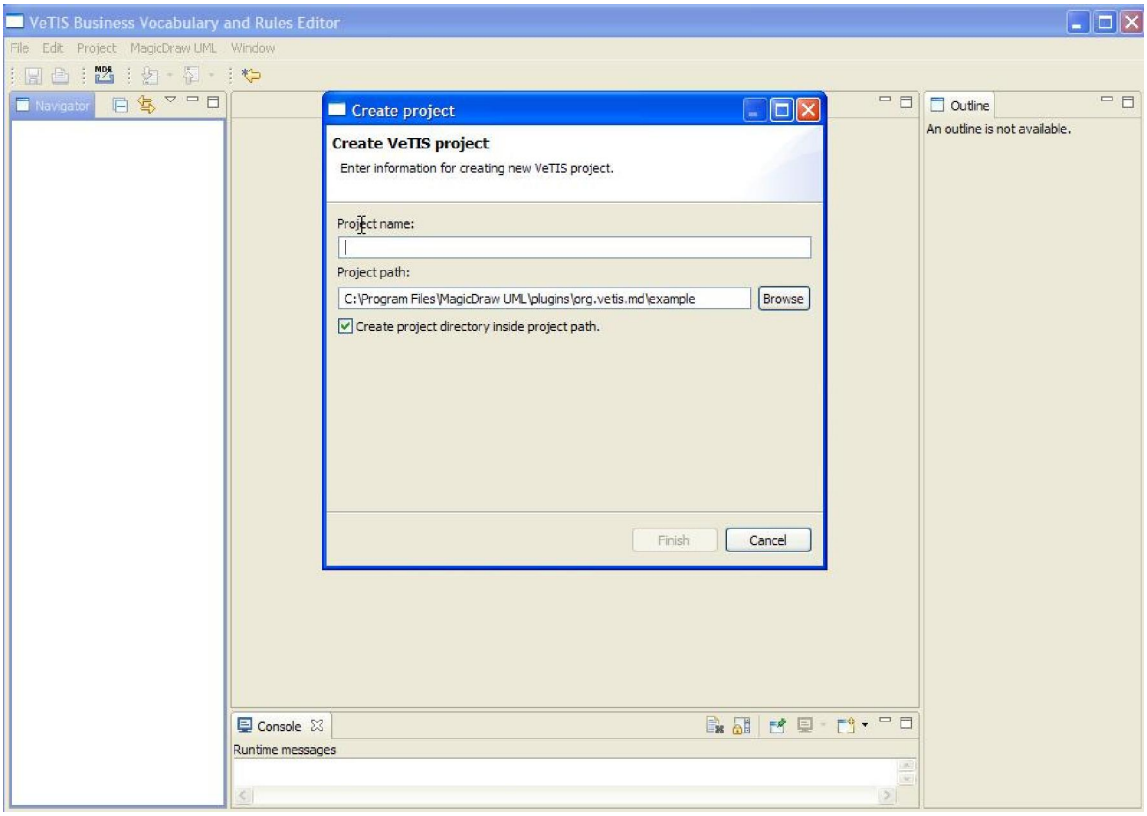
```
            do {
            thisModule.preCounter <- thisModule.preCounter + 1;

            }
      }

rule StatementWithDerivation2Invariant {

      from

      s :   SBVR!Statement(
            thisModule.isDerivationRule(s))


      to

      t : UML!Constraint(
            name <- 'inv'+thisModule.className(s)+thisModule.invCounter.toString()
            ,
            constrainedElement <- thisModule.constrainedElem(s)
            ,
            specification <- ruleSpecification
            )
            ,
            ruleSpecification:UML!OpaqueExpression (
                        name<-'',
                        language<-'OCL2.0',
                        body<-
                                    thisModule.bodyOclText(thisModule.baseFormulation(s))
                  +' implies '
                                    +
thisModule.bodyOclText(thisModule.centralFormulation(s).antecedent)
                                    )

            do {
            thisModule.invCounter <- thisModule.invCounter + 1;
            thisModule.iteratorCounter <- thisModule.iteratorCounter + 1;

            }
     }
```

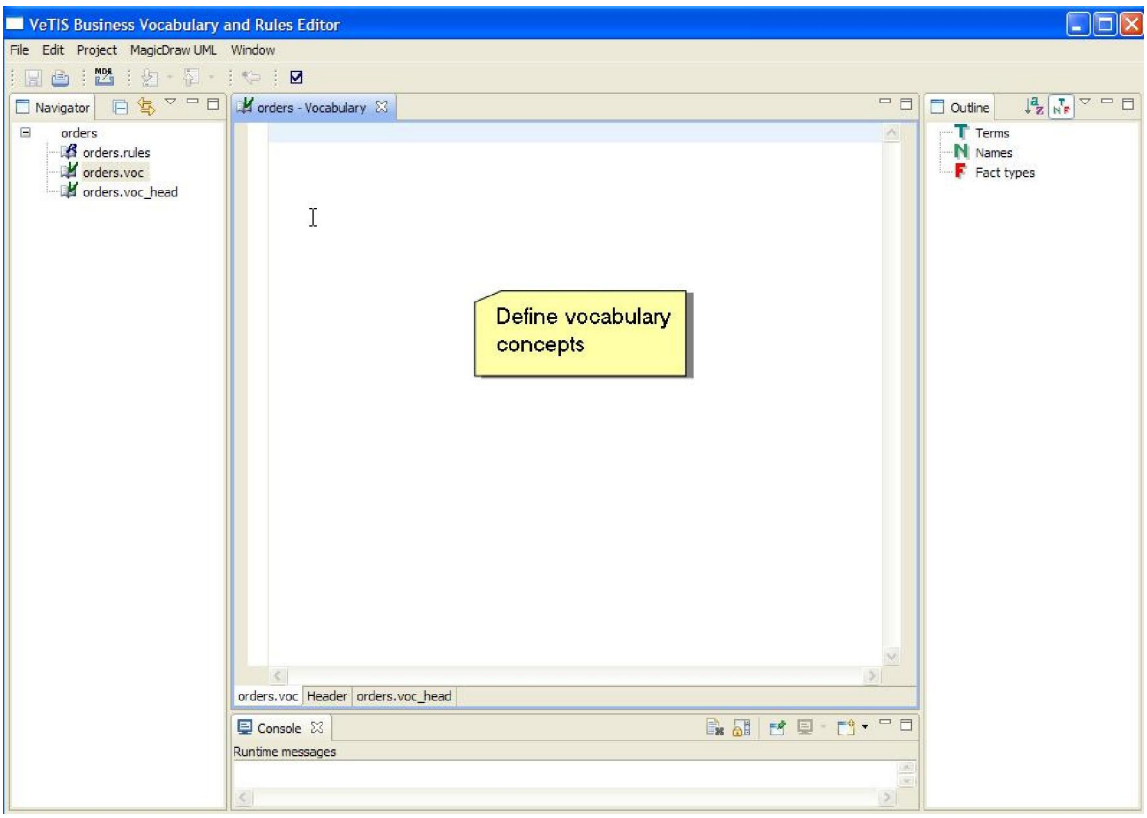
6 priedas. VeTIS įrankio mokomoji medžiaga, demonstracija

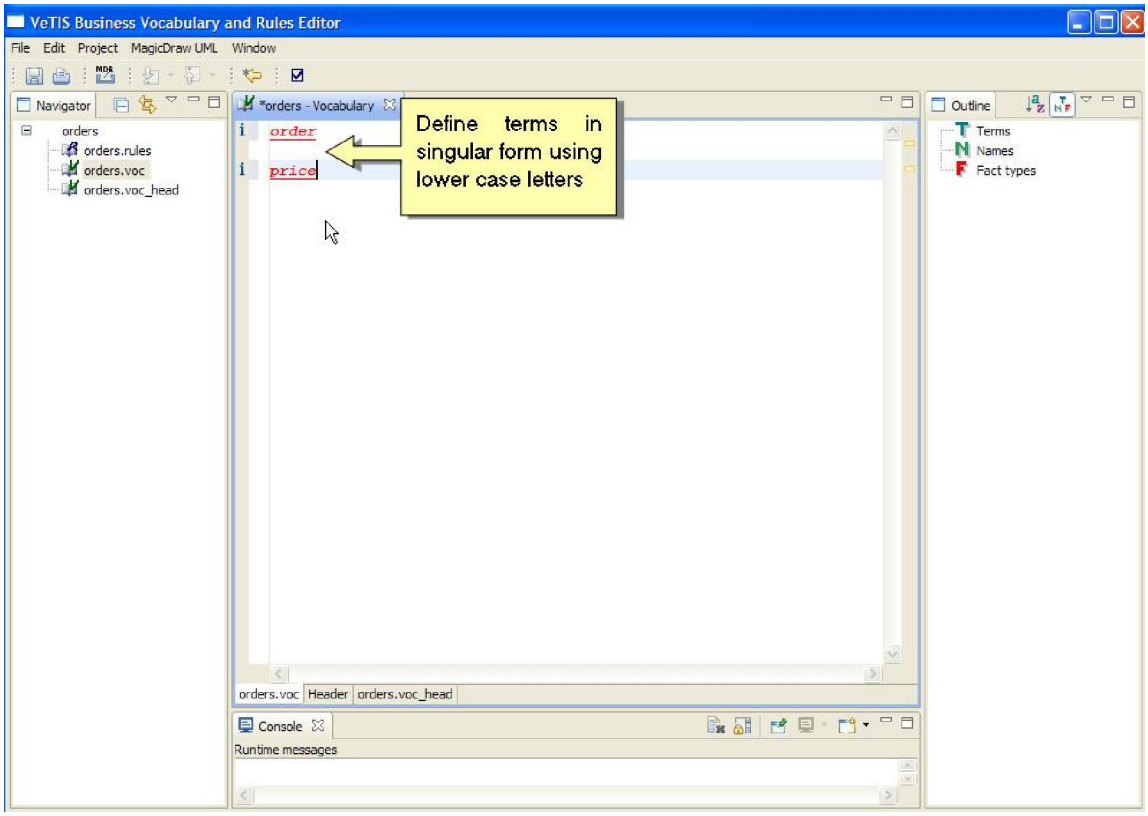
Sukurta VeTIS įrankio demonstracija, kuri randasi: http://www.magicdraw.com/magicdraw_resources.

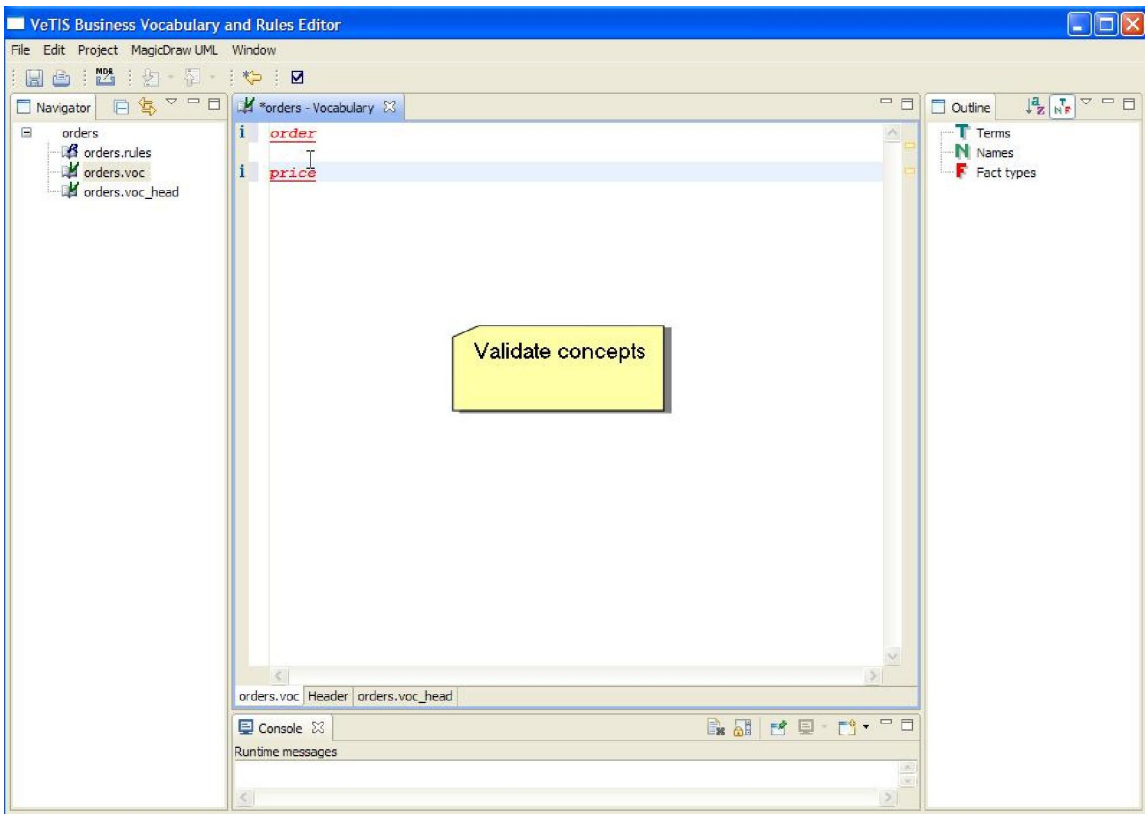


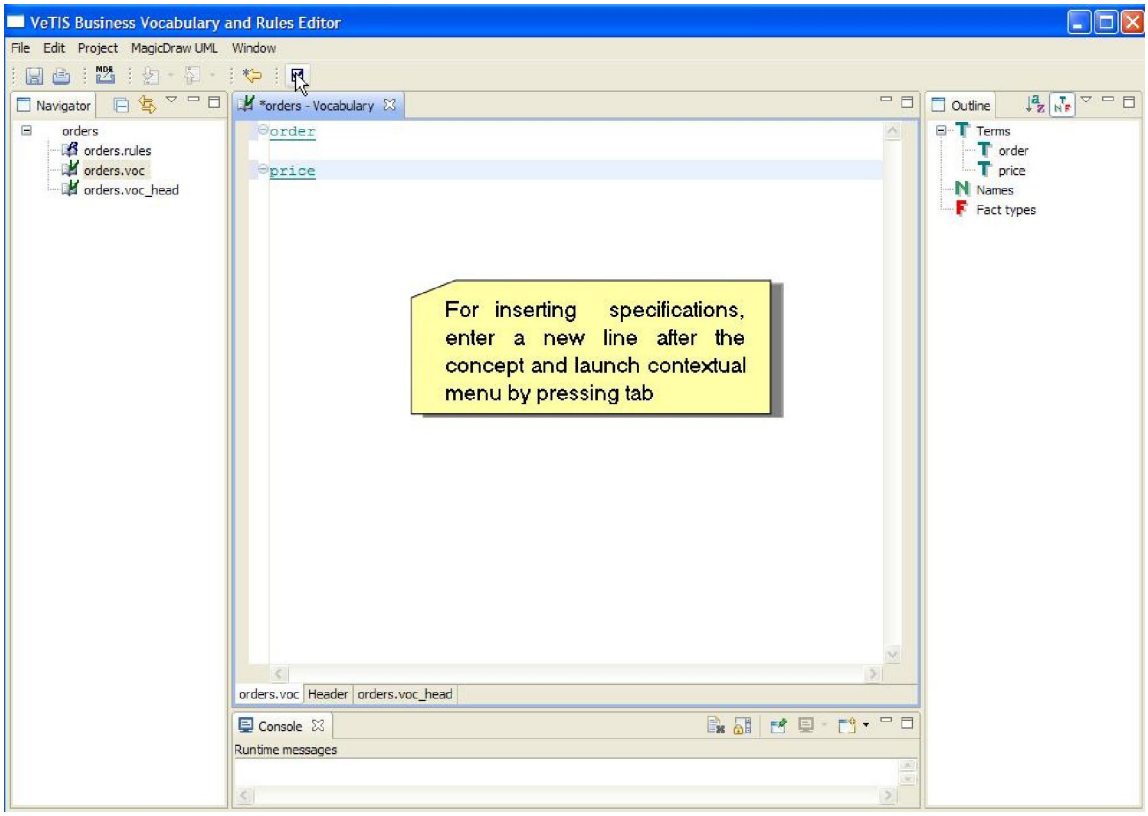


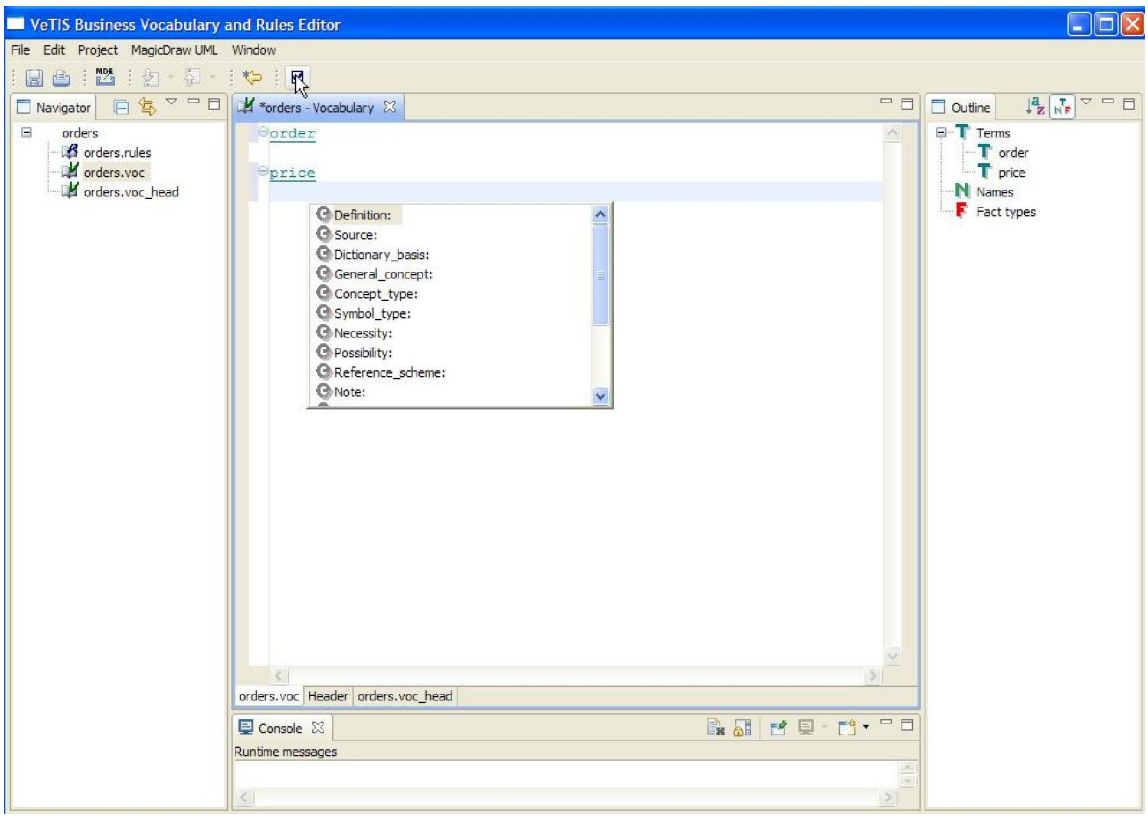

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
Create project
Create VeTIS project
Enter information for creating new VeTIS project.
Project name:
Project path:
C:\Program Files\MagicDraw UML\plugins\org.vetis.md\example
Browse
Create project directory inside project path.
Finish
Cancel
Outline
An outline is not available.
Console
Runtime messages

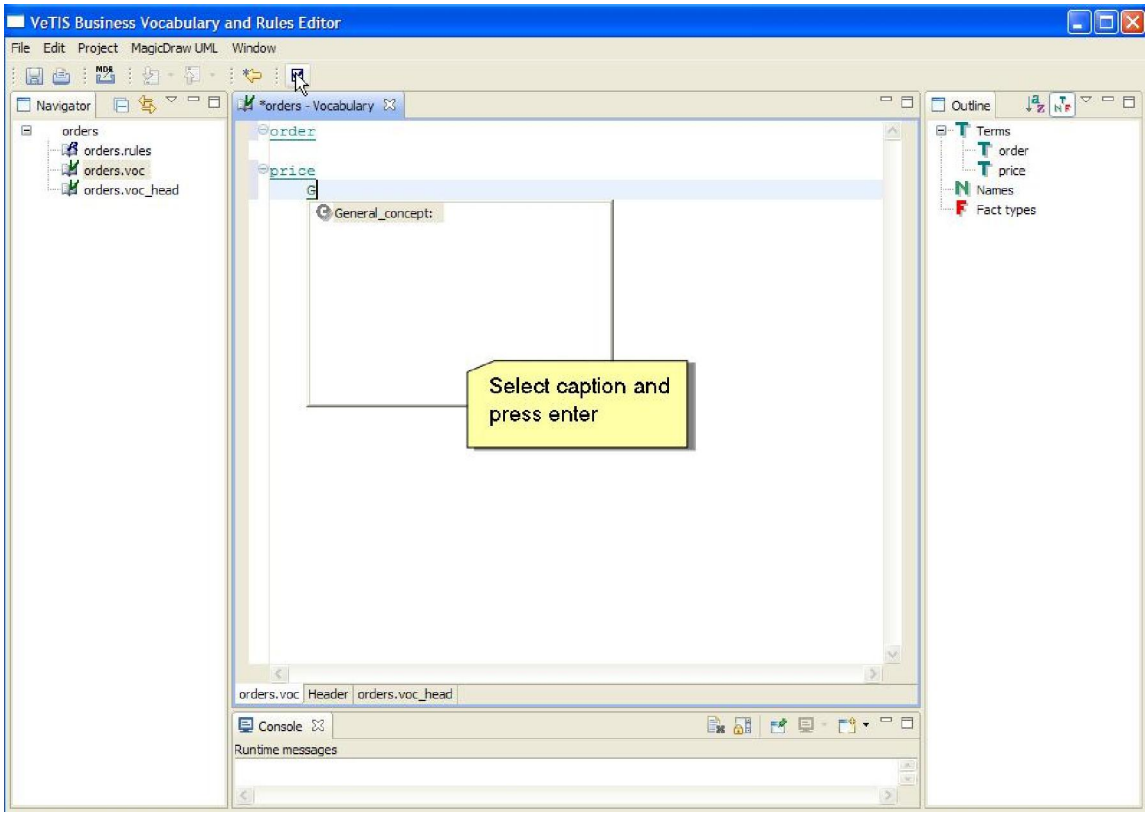

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
orders - Vocabulary
Define vocabulary concepts
orders.voc
Header
orders.voc_head
Outline
Terms
Names
Fact types
Console
Runtime messages

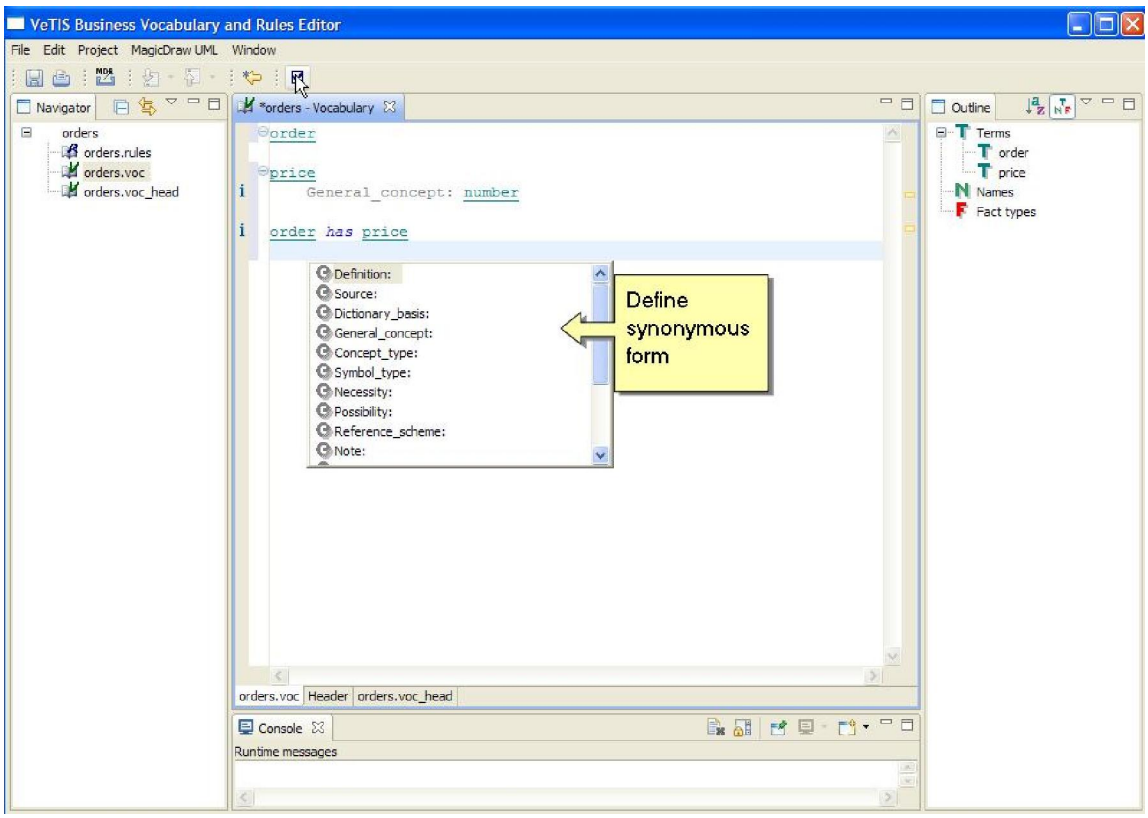

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
Define terms in singular form using lower case letters
Outline
Terms
Names
Fact types
orders.voc Header orders.voc_head
Console
Runtime messages

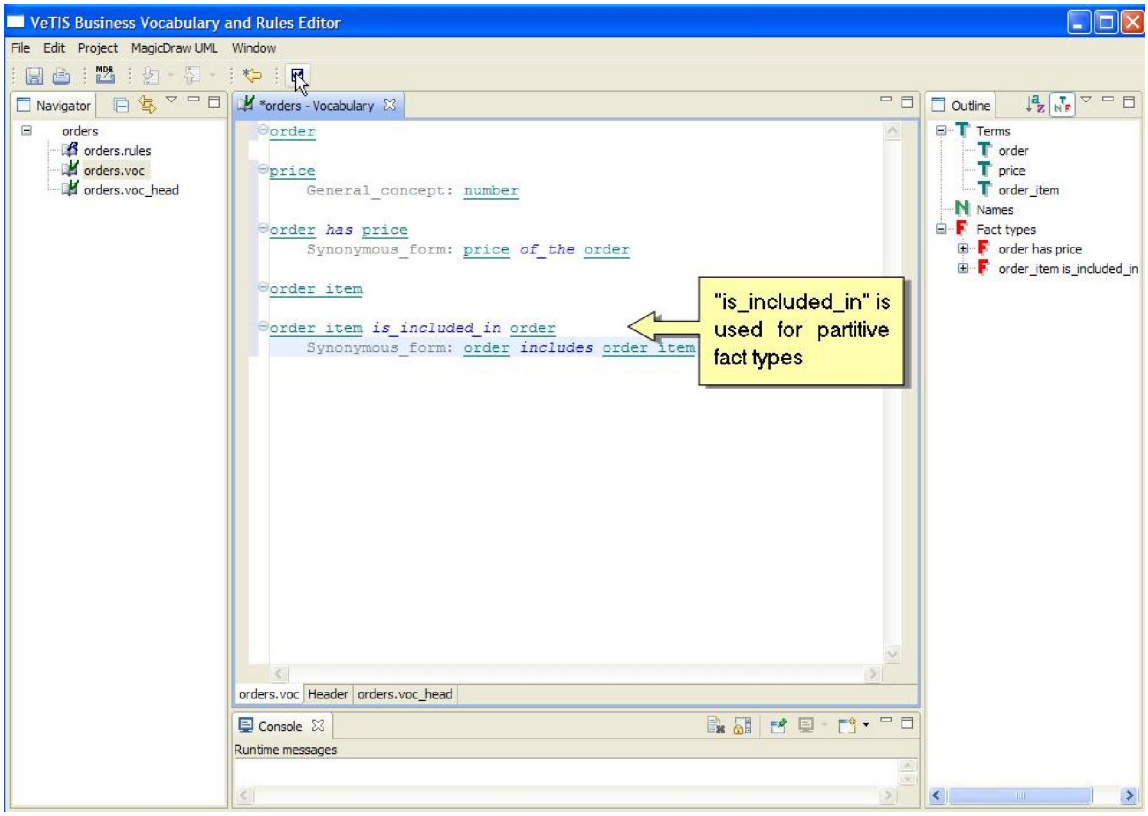

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
Validate concepts
Outline
Terms
Names
Fact types
orders.voc Header orders.voc_head
Console
Runtime messages

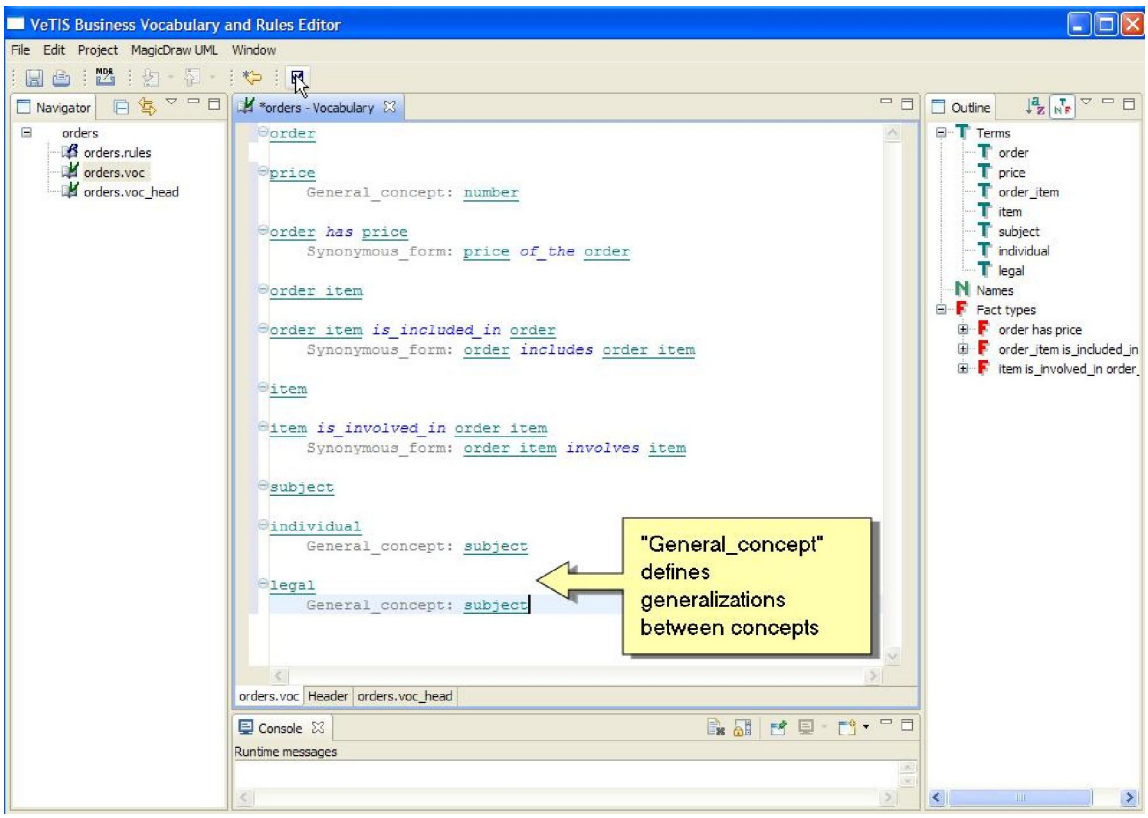

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
For inserting specifications, enter a new line after the concept and launch contextual menu by pressing tab
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
Names
Fact types

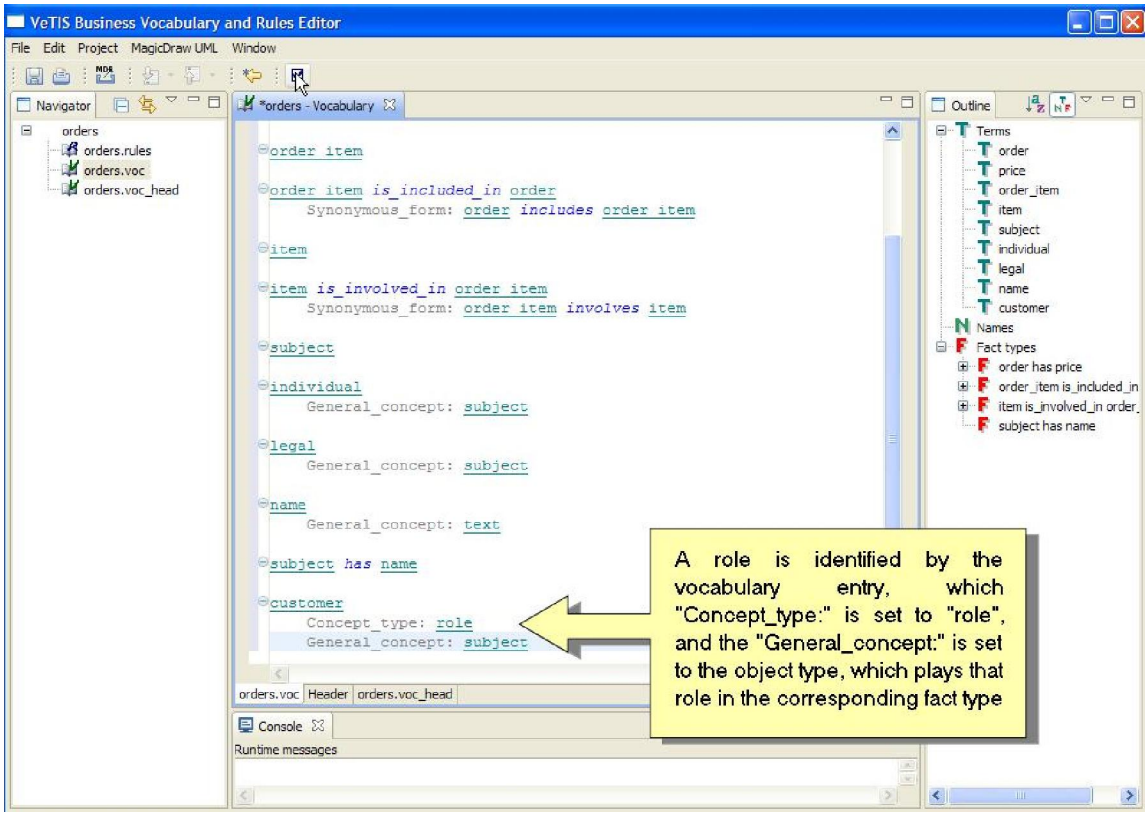

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
Definition:
Source:
Dictionary_basis:
General_concept:
Concept_type:
Symbol_type:
Necessity:
Possibility:
Reference_scheme:
Note:
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
Names
Fact types

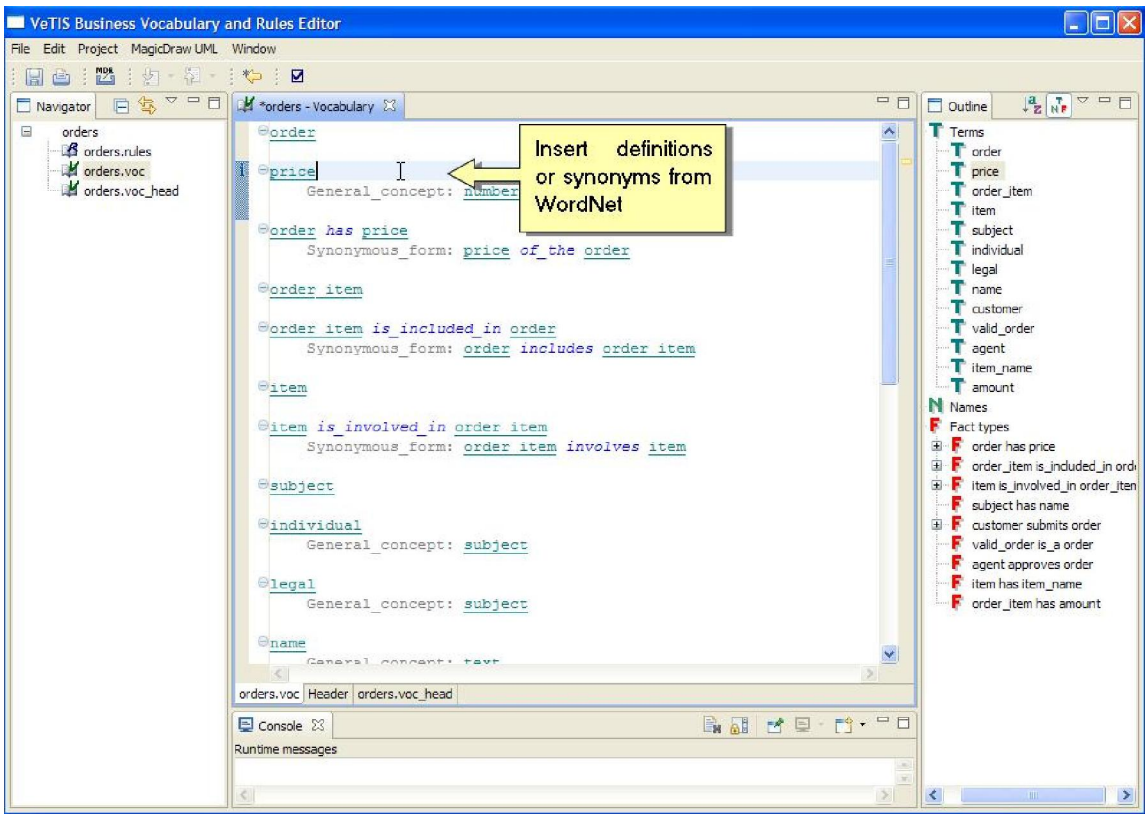

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
General_concept:
Select caption and press enter
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
Names
Fact types

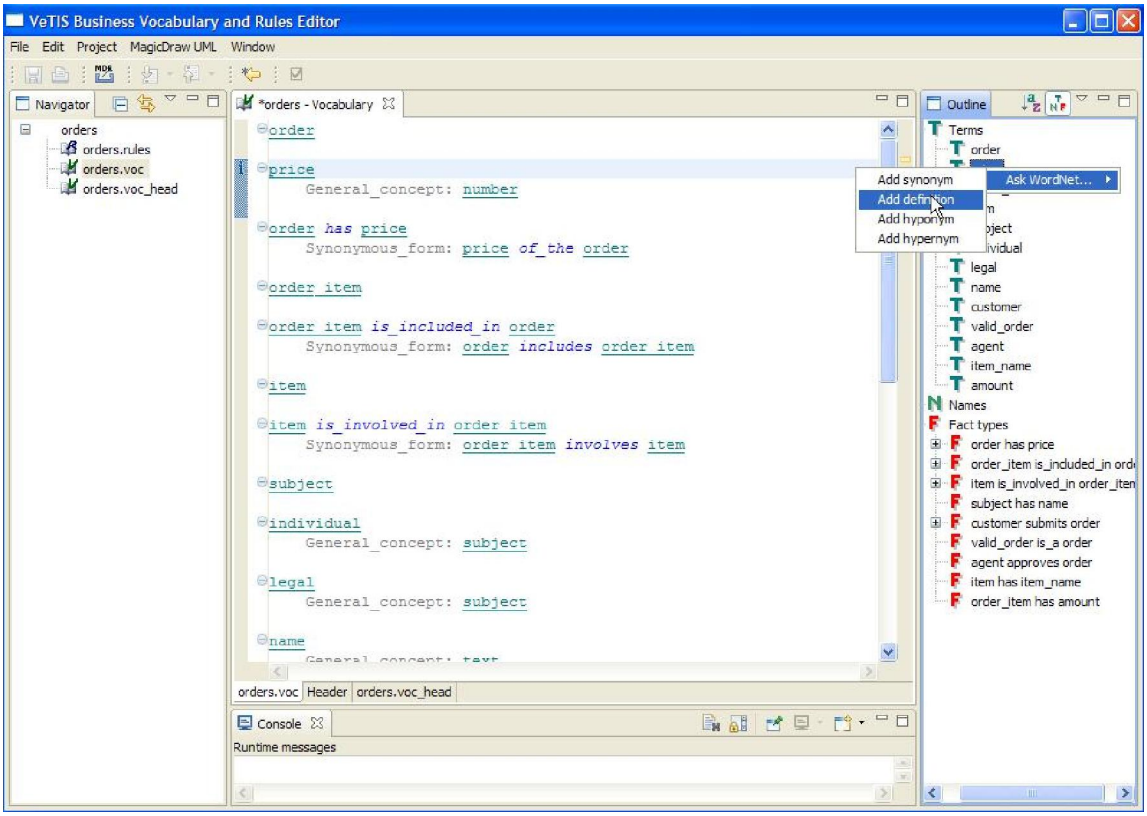

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
General_concept: number
order has price
Definition:
Source:
Dictionary_basis:
General_concept:
Concept_type:
Symbol_type:
Necessity:
Possibility:
Reference_scheme:
Note:
Define synonymous form
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
Names
Fact types

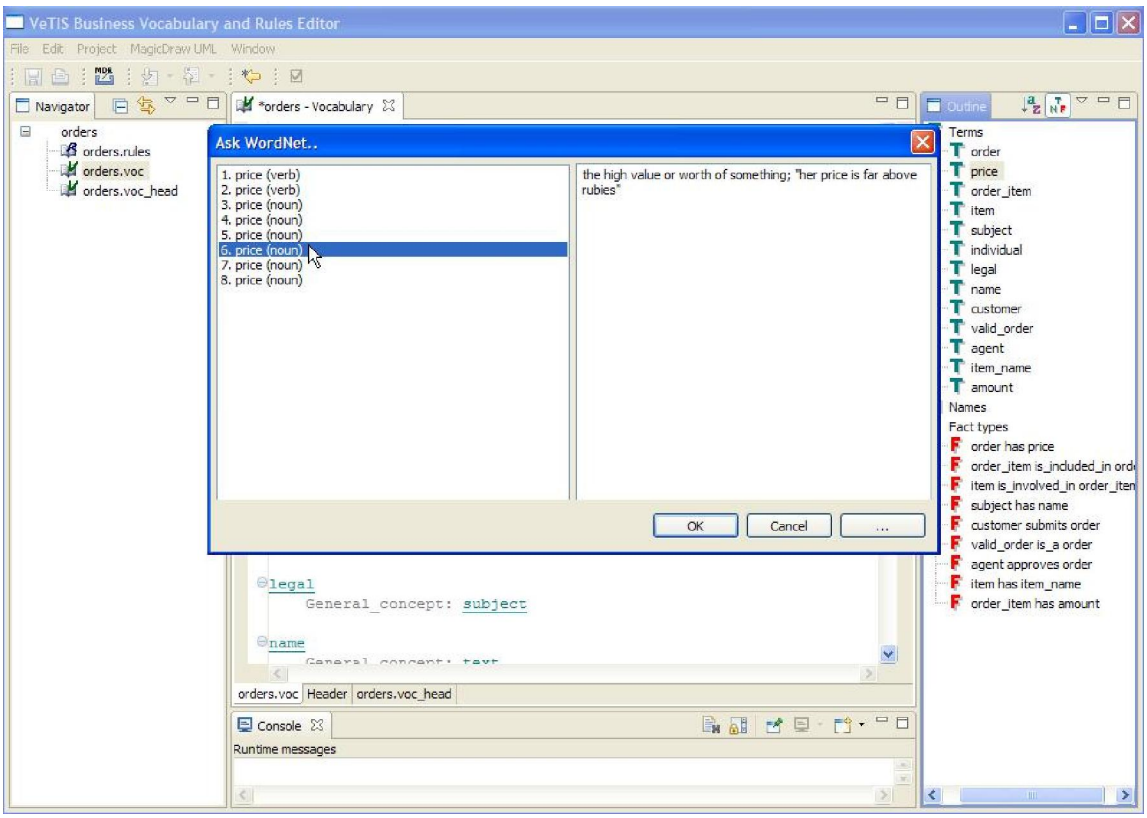

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
General_concept: number
order has price
Synonymous_form: price of the order
order item
order item is_included_in order
Synonymous_form: order includes order item
"is_included_in" is used for partitive fact types
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
order_item
Names
Fact types
order has price
order_item is_included_in

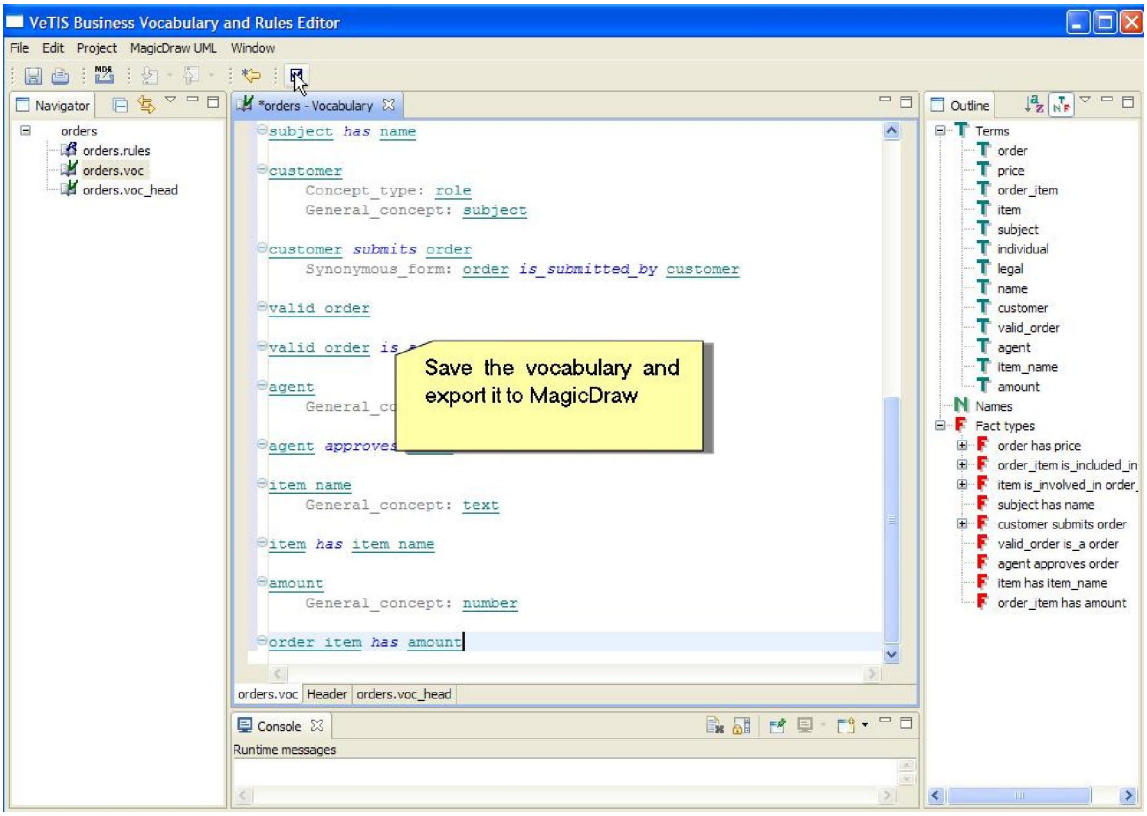

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
General_concept: number
order has price
Synonymous_form: price of the order
order item
order item is_included_in order
Synonymous_form: order includes order item
item
item is_involved_in order item
Synonymous_form: order item involves item
subject
individual
General_concept: subject
legal
General_concept: subject
"General_concept" defines generalizations between concepts
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
order_item
item
subject
individual
legal
Names
Fact types
order has price
order_item is_included_in
item is_involved_in order_

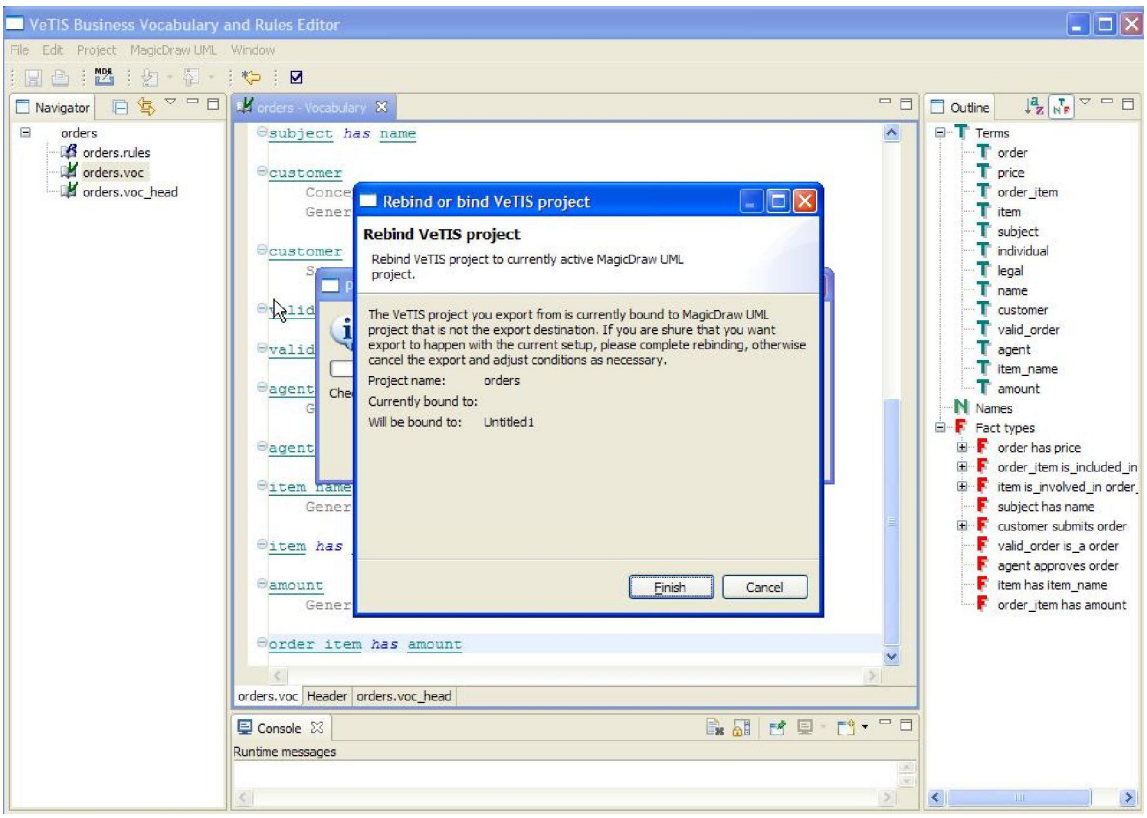

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order item
order item is_included_in order
Synonymous_form: order includes order item
item
item is_involved_in order item
Synonymous_form: order item involves item
subject
individual
General_concept: subject
legal
General_concept: subject
name
General_concept: text
subject has name
customer
Concept_type: role
General_concept: subject
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
order_item
item
subject
individual
legal
name
customer
Names
Fact types
order has price
order_item is_included_in
item is_involved_in order_
subject has name
A role is identified by the vocabulary entry, which "Concept_type:" is set to "role", and the "General_concept:" is set to the object type, which plays that role in the corresponding fact type

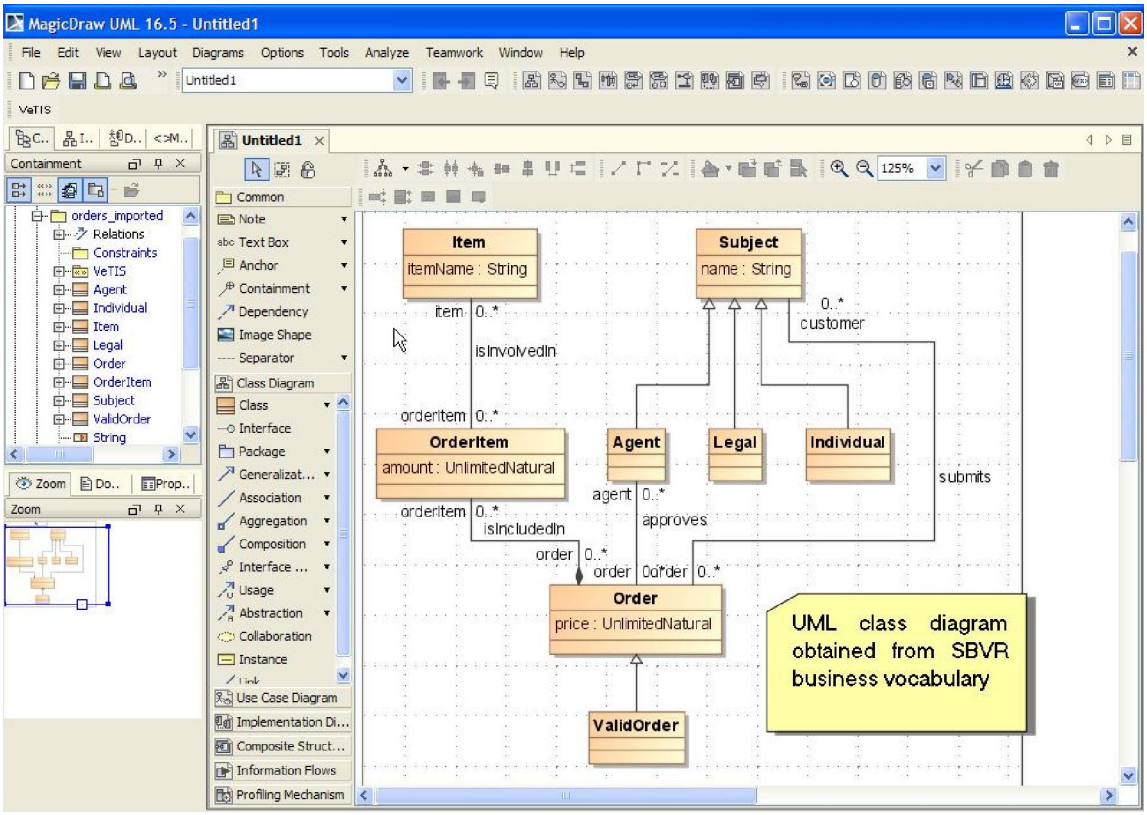

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
General_concept: number
order has price
Synonymous_form: price of the order
order item
order item is_included_in order
Synonymous_form: order includes order item
item
item is_involved_in order item
Synonymous_form: order item involves item
subject
individual
General_concept: subject
legal
General_concept: subject
name
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
order_item
item
subject
individual
legal
name
customer
valid_order
agent
item_name
amount
Names
Fact types
order has price
subject has name
customer submits order
valid_order is_a order
agent approves order
item has item_name
order_item has amount
Insert definitions or synonyms from WordNet

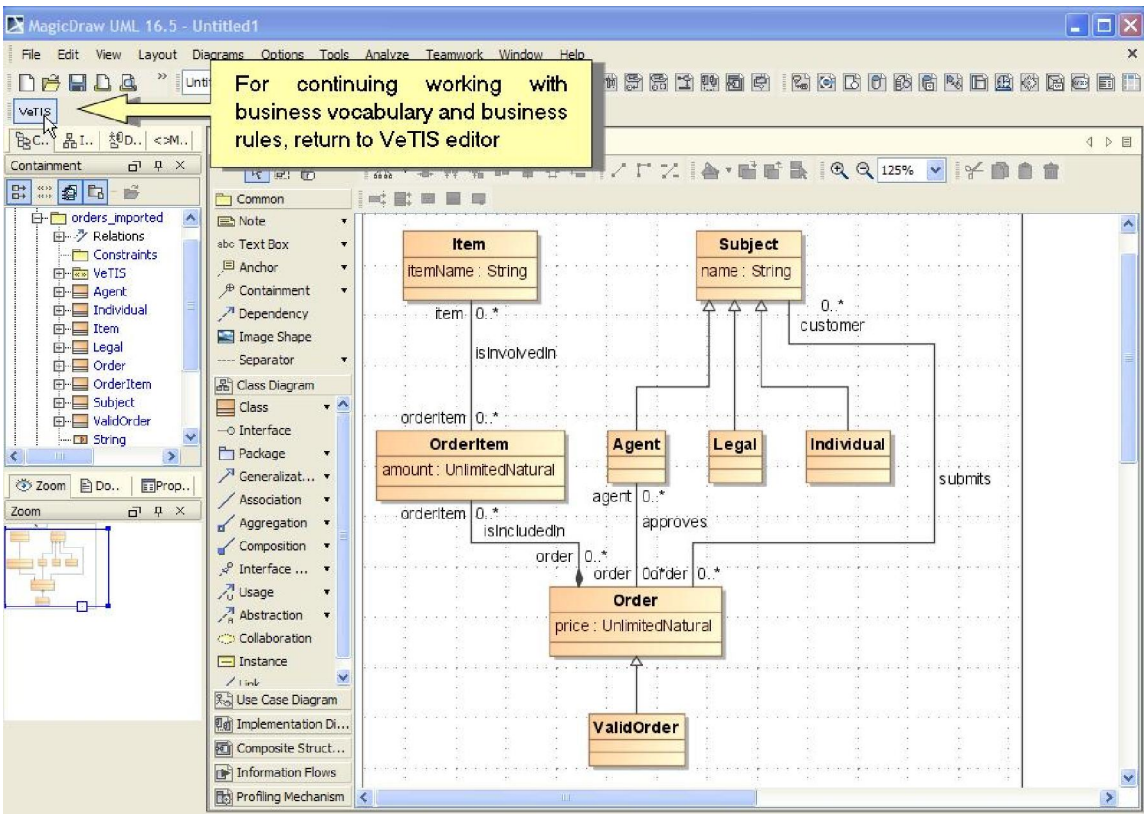

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
order
price
General_concept: number
order has price
Synonymous_form: price of the order
order item
order item is included in order
Synonymous_form: order includes order item
item
item is involved in order item
Synonymous_form: order item involves item
subject
individual
General_concept: subject
legal
General_concept: subject
name
orders.voc Header orders.voc_head
Console
Runtime messages
Add synonym
Add definition
Add hyponym
Add hypernym
Ask WordNet...
Outline
Terms
order
legal
name
customer
valid_order
agent
item_name
amount
Names
Fact types
order has price
subject has name
customer submits order
valid_order is_a order
agent approves order
item has item_name
order_item has amount

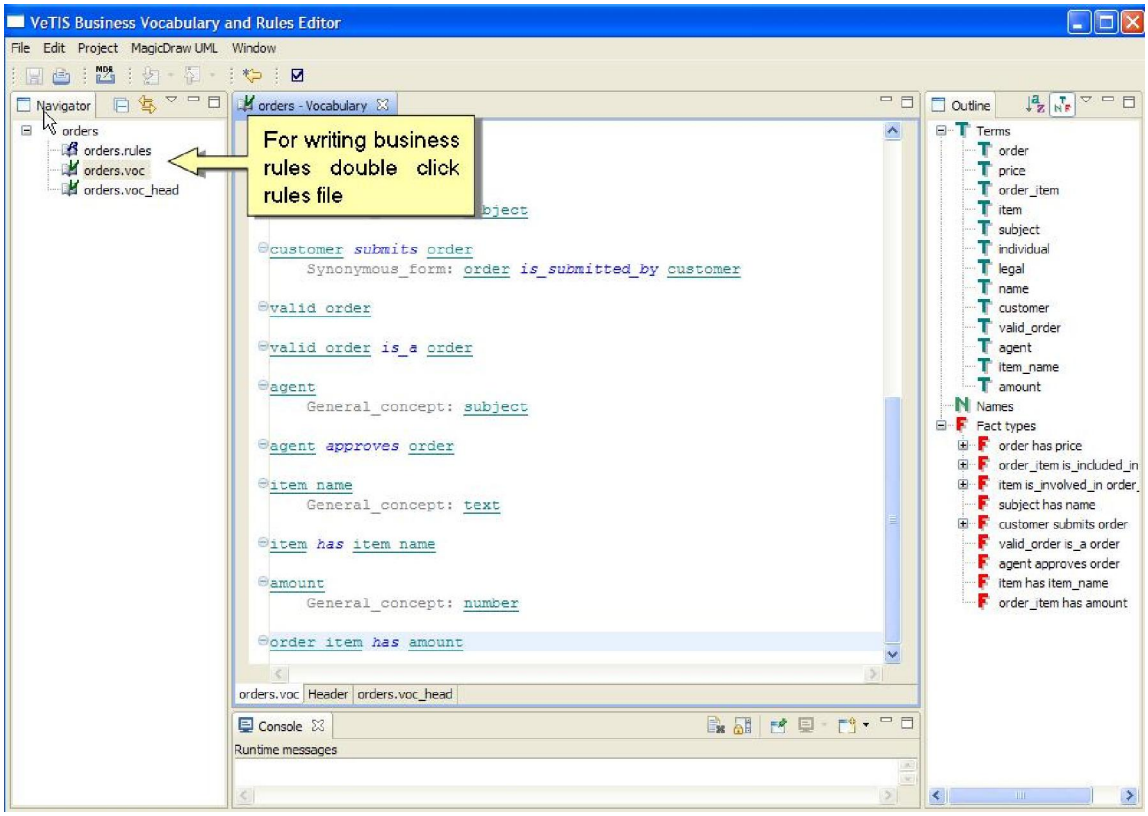

VeTIS Business Vocabulary and Rules Editor
Ask WordNet..
1. price (verb)
2. price (verb)
3. price (noun)
4. price (noun)
5. price (noun)
6. price (noun)
7. price (noun)
8. price (noun)
the high value or worth of something; "her price is far above rubies"
OK
Cancel
...
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
Outline
Terms
order
price
order_item
item
subject
individual
legal
name
customer
valid_order
agent
item_name
amount
Names
Fact types
order has price
subject has name
customer submits order
valid_order is_a order
agent approves order
item has item_name
order_item has amount
Console
Runtime messages

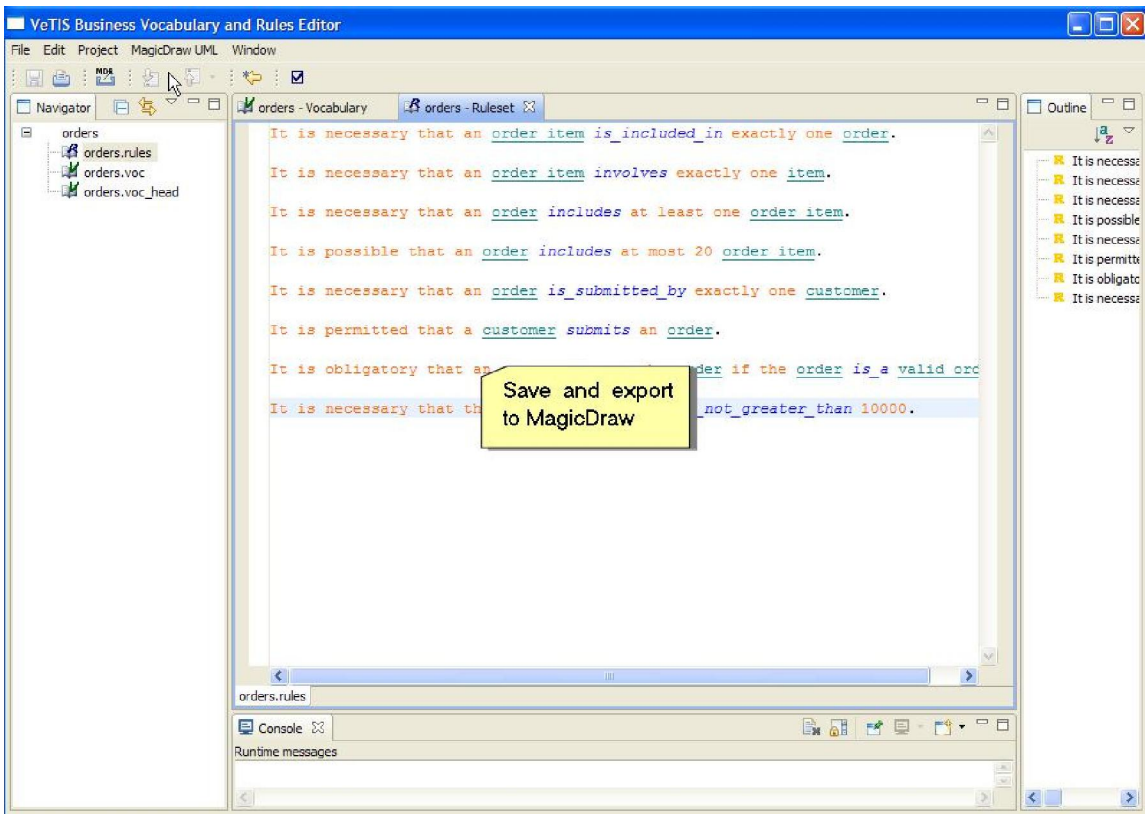

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
*orders - Vocabulary
subject has name
customer
Concept_type: role
General_concept: subject
customer submits order
Synonymous_form: order is_submitted_by customer
valid order
agent
General_co
agent approves
item name
General_concept: text
item has item name
amount
General_concept: number
order item has amount
Save the vocabulary and export it to MagicDraw
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
order_item
item
subject
individual
legal
name
customer
valid_order
agent
item_name
amount
Names
Fact types
order has price
order_item is_included_in
item is_involved_in order_
subject has name
customer submits order
valid_order is_a order
agent approves order
item has item_name
order_item has amount

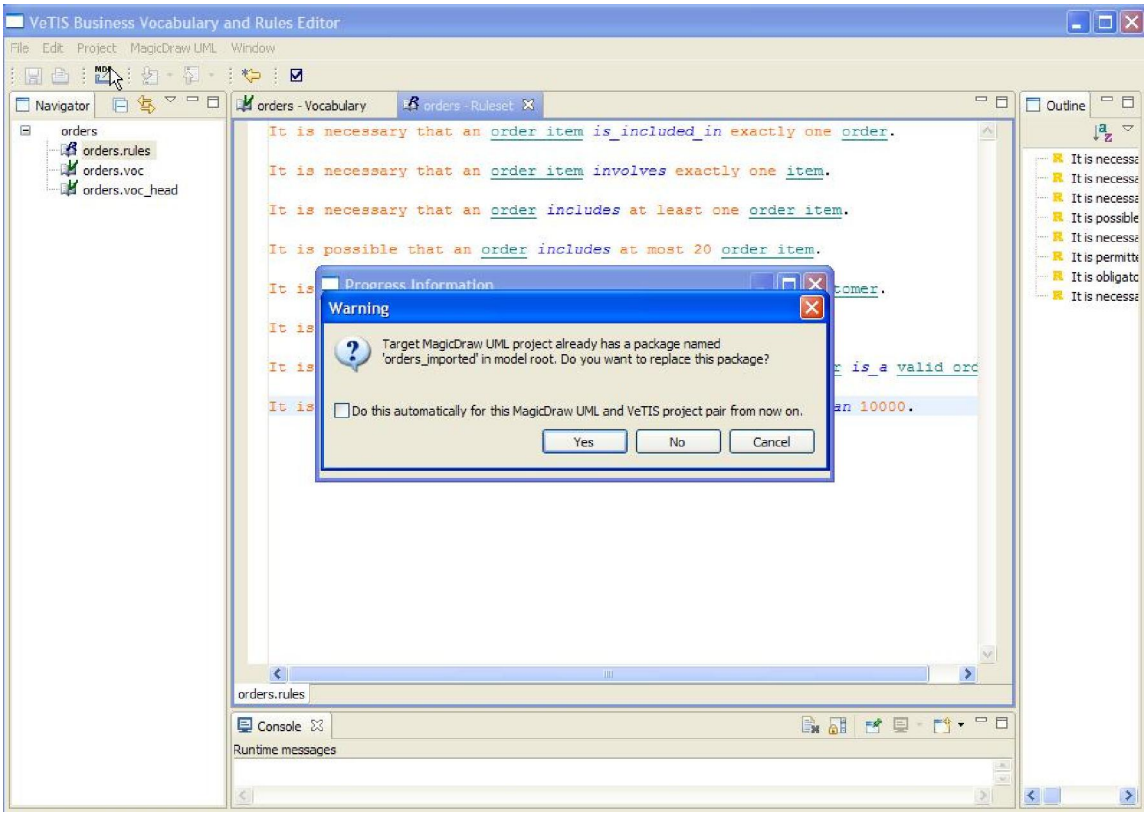

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
orders - Vocabulary
subject has name
customer
item has
amount
order item has amount
Rebind or bind VeTIS project
Rebind VeTIS project
Rebind VeTIS project to currently active MagicDraw UML project.
The VeTIS project you export from is currently bound to MagicDraw UML project that is not the export destination. If you are shure that you want export to happen with the current setup, please complete rebinding, otherwise cancel the export and adjust conditions as necessary.
Project name: orders
Currently bound to:
Will be bound to: Untitled1
Finish
Cancel
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
order_item
item
subject
individual
legal
name
customer
valid_order
agent
item_name
amount
Names
Fact types
order has price
order_item is_included_in
item is_involved_in order_
subject has name
customer submits order
valid_order is_a order
agent approves order
item has item_name
order_item has amount

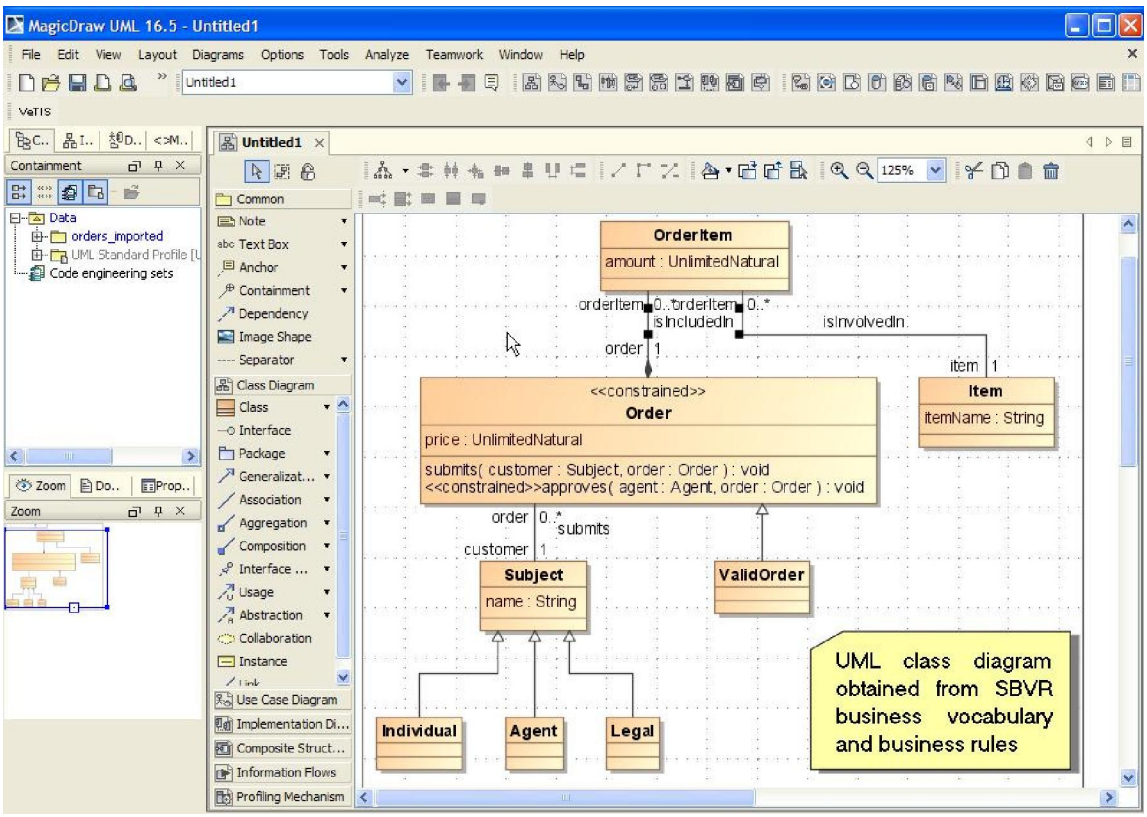

MagicDraw UML 16.5 - Untitled1
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
VeTIS
Containment
orders_imported
Relations
Constraints
VeTIS
Agent
Individual
Item
Legal
Order
OrderItem
Subject
ValidOrder
String
Zoom
Untitled1
125%
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
Class Diagram
Class
Interface
Package
Generalizat...
Association
Aggregation
Composition
Interface ...
Usage
Abstraction
Collaboration
Instance
Use Case Diagram
Implementation Di...
Composite Struct...
Information Flows
Profiling Mechanism
Item
itemName : String
Subject
name : String
item 0..*
isInvolvedIn
0..*
customer
orderItem 0..*
OrderItem
amount : UnlimitedNatural
Agent
Legal
Individual
submits
agent 0..*
approves
orderItem 0..*
isIncludedIn
order 0..*
order 0..*
order 0..*
Order
price : UnlimitedNatural
ValidOrder
UML class diagram obtained from SBVR business vocabulary

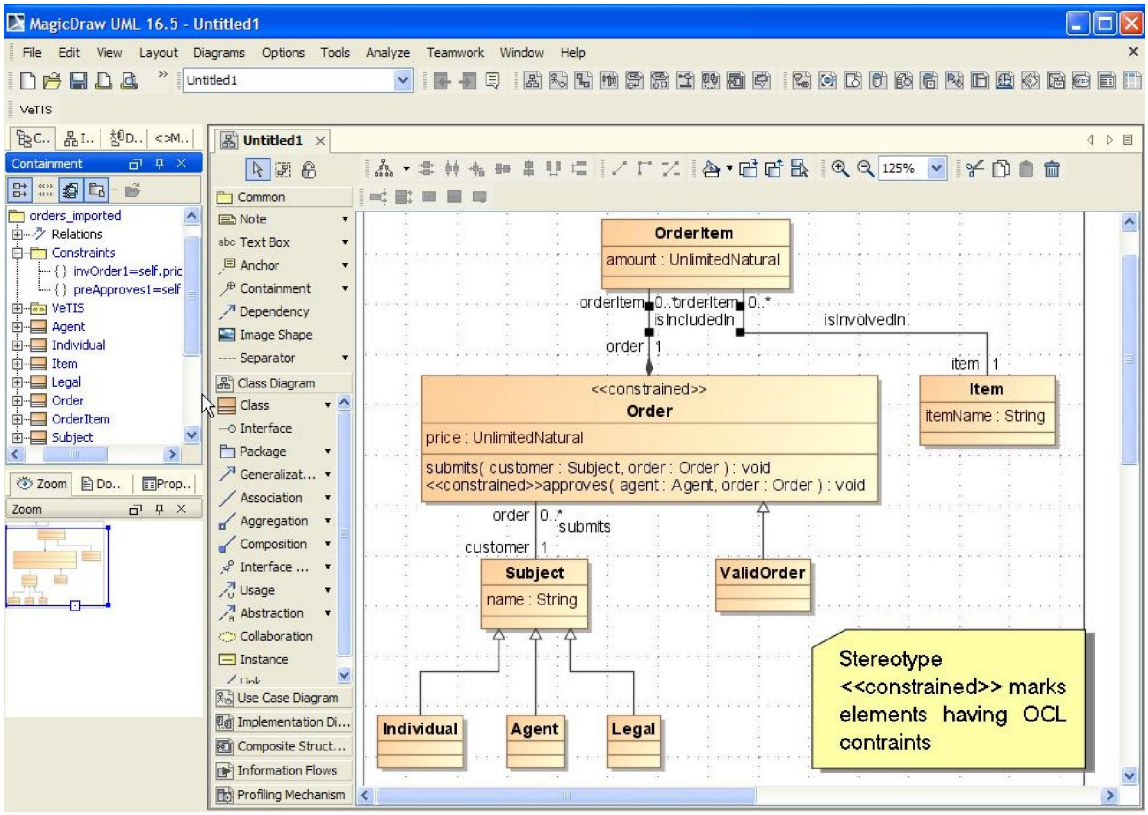

MagicDraw UML 16.5 - Untitled1
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
VeTIS
For continuing working with business vocabulary and business rules, return to VeTIS editor
Containment
orders_imported
Relations
Constraints
VeTIS
Agent
Individual
Item
Legal
Order
OrderItem
Subject
ValidOrder
String
Zoom
125%
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
Class Diagram
Class
Interface
Package
Generalizat...
Association
Aggregation
Composition
Interface ...
Usage
Abstraction
Collaboration
Instance
Use Case Diagram
Implementation Di...
Composite Struct...
Information Flows
Profiling Mechanism
Item
itemName : String
Subject
name : String
item 0..*
isInvolvedIn
0..*
customer
orderItem 0..*
OrderItem
amount : UnlimitedNatural
Agent
Legal
Individual
submits
agent 0..*
approves
orderItem 0..*
isIncludedIn
order 0..*
order 0..*
order 0..*
Order
price : UnlimitedNatural
ValidOrder

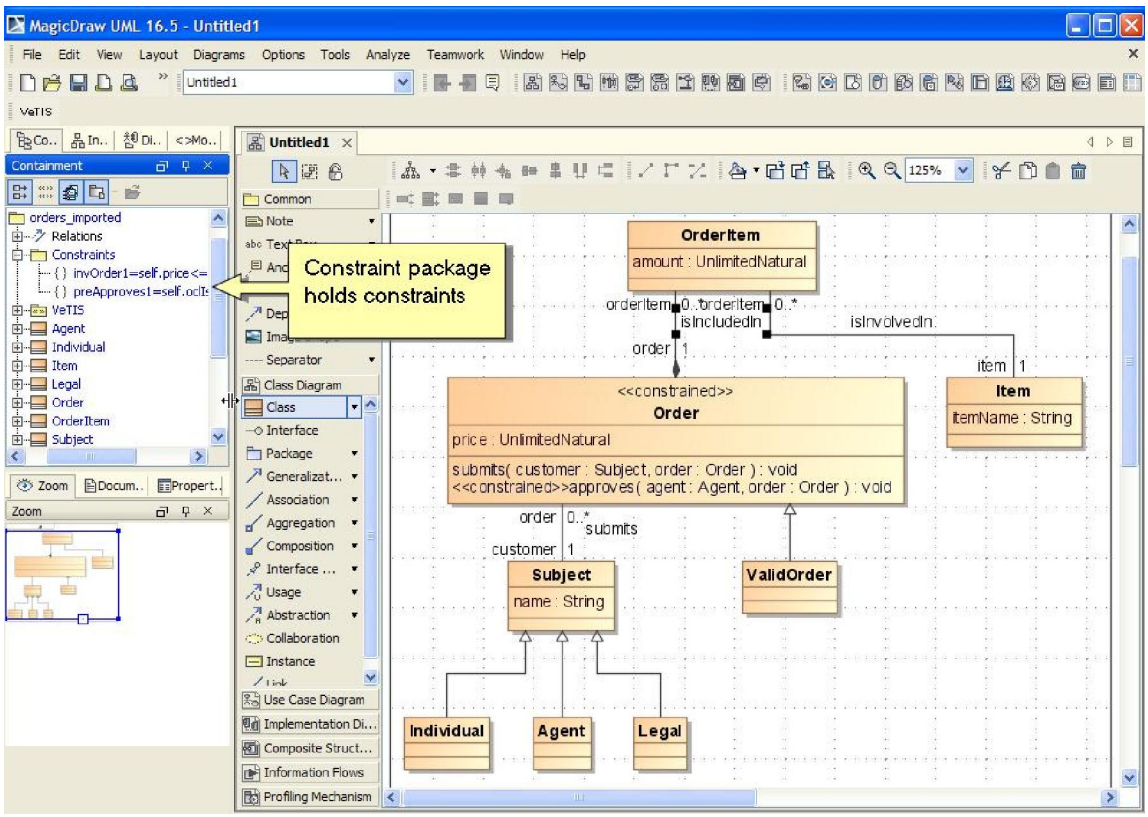

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
For writing business rules double click rules file
orders - Vocabulary
customer submits order
Synonymous_form: order is_submitted_by customer
valid order
valid order is_a order
agent
General_concept: subject
agent approves order
item name
General_concept: text
item has item name
amount
General_concept: number
order item has amount
orders.voc Header orders.voc_head
Console
Runtime messages
Outline
Terms
order
price
order_item
item
subject
individual
legal
name
customer
valid_order
agent
item_name
amount
Names
Fact types
order has price
order_item is_included_in
item is_involved_in order_
subject has name
customer submits order
valid_order is_a order
agent approves order
item has item_name
order_item has amount

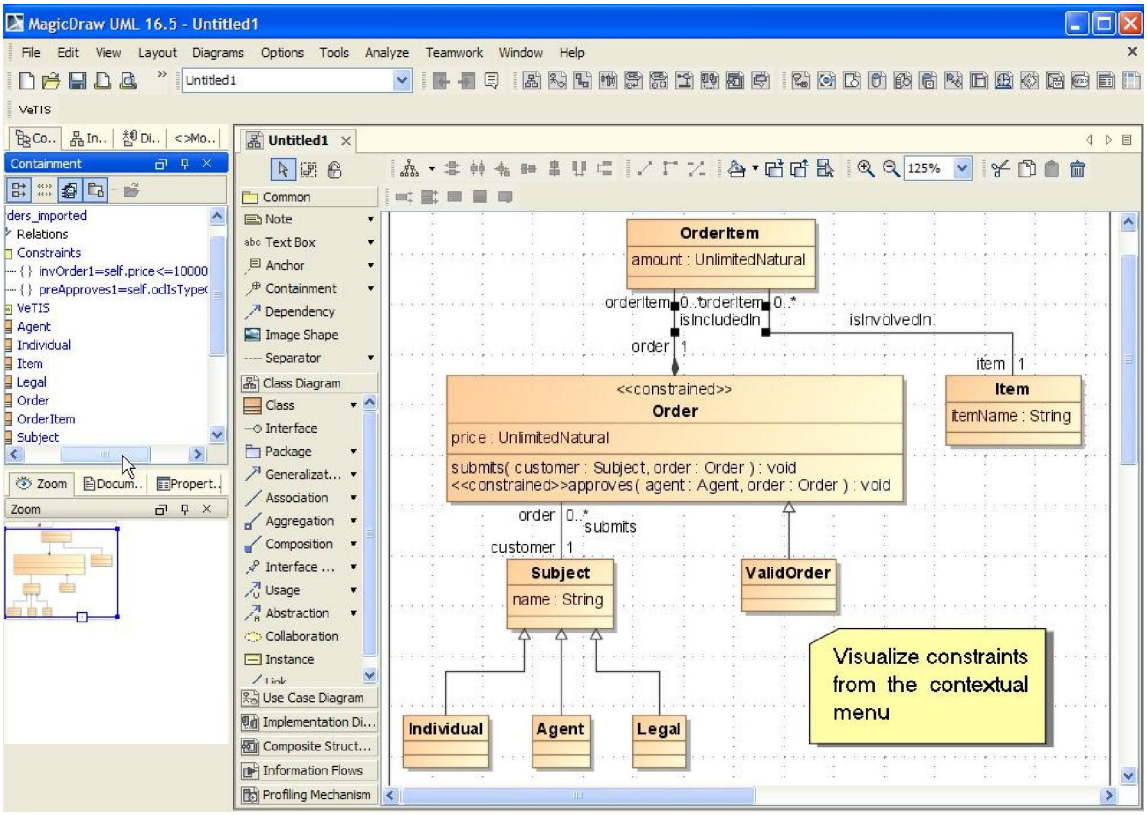

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
orders - Vocabulary
orders - Ruleset
It is necessary that an order item is_included_in exactly one order.
It is necessary that an order item involves exactly one item.
It is necessary that an order includes at least one order item.
It is possible that an order includes at most 20 order item.
It is necessary that an order is_submitted_by exactly one customer.
It is permitted that a customer submits an order.
Save and export to MagicDraw
orders.rules
Console
Runtime messages
Outline

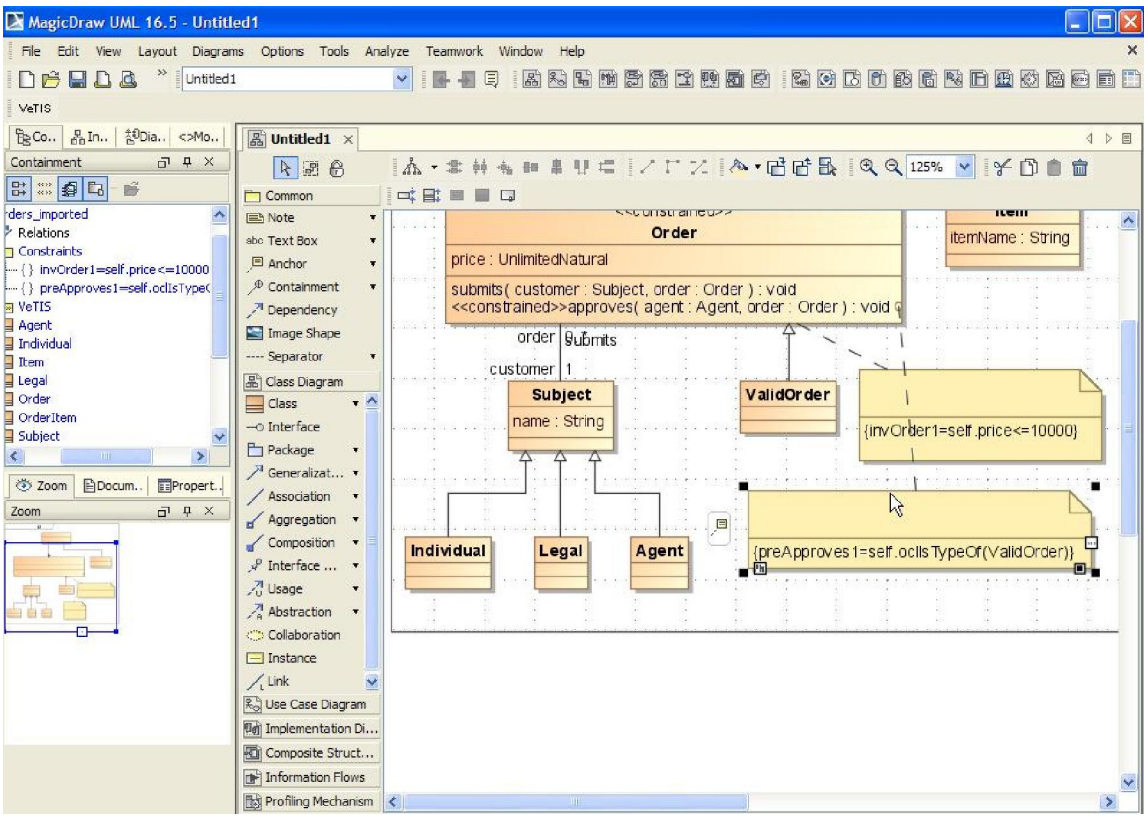

VeTIS Business Vocabulary and Rules Editor
File Edit Project MagicDraw UML Window
Navigator
orders
orders.rules
orders.voc
orders.voc_head
orders - Vocabulary
orders - Ruleset
It is necessary that an order item is_included_in exactly one order.
It is necessary that an order item involves exactly one item.
It is necessary that an order includes at least one order item.
It is possible that an order includes at most 20 order item.
Progress Information
Warning
Target MagicDraw UML project already has a package named 'orders_imported' in model root. Do you want to replace this package?
Do this automatically for this MagicDraw UML and VeTIS project pair from now on.
Yes
No
Cancel
Outline
orders.rules
Console
Runtime messages


MagicDraw UML 16.5 - Untitled1
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
VeTIS
Containment
Data
orders_imported
UML Standard Profile
Code engineering sets
Untitled1
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
Class Diagram
Class
Interface
Package
Generalizat...
Association
Aggregation
Composition
Interface ...
Usage
Abstraction
Collaboration
Instance
Use Case Diagram
Implementation Di...
Composite Struct...
Information Flows
Profiling Mechanism
OrderItem
amount : UnlimitedNatural
orderItem 0..*
isIncludedIn
order 1
isInvolvedIn
item 1
<<constrained>>
Order
price : UnlimitedNatural
submits(customer : Subject, order : Order) : void
<<constrained>>approves(agent : Agent, order : Order) : void
Item
itemName : String
order 0..*
submits
customer 1
Subject
name : String
ValidOrder
Individual
Agent
Legal
UML class diagram obtained from SBVR business vocabulary and business rules


MagicDraw UML 16.5 - Untitled1
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
Untitled1
VeTIS
Containment
orders_imported
Relations
Constraints
{ } invOrder1=self.pric
{ } preApproves1=self
VeTIS
Agent
Individual
Item
Legal
Order
OrderItem
Subject
Zoom
Do..
Prop..
Zoom
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
Class Diagram
Class
Interface
Package
Generalizat...
Association
Aggregation
Composition
Interface ...
Usage
Abstraction
Collaboration
Instance
Use Case Diagram
Implementation Di...
Composite Struct...
Information Flows
Profiling Mechanism
Untitled1
125%
OrderItem
amount : UnlimitedNatural
orderItem
0..*
orderItem
0..*
isIncludedIn
isInvolvedIn
order
1
item
1
<<constrained>>
Order
price : UnlimitedNatural
submits(customer : Subject, order : Order) : void
<<constrained>>approves(agent : Agent, order : Order) : void
Item
itemName : String
order
0..*
submits
customer
1
Subject
name : String
ValidOrder
Individual
Agent
Legal
Stereotype
<<constrained>> marks
elements having OCL
contraints


MagicDraw UML 16.5 - Untitled1
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
Untitled1
VeTIS
Containment
orders_imported
Relations
Constraints
{ } invOrder1=self.price<=
{ } preApproves1=self.ocl
VeTIS
Agent
Individual
Item
Legal
Order
OrderItem
Subject
Zoom
Docum..
Propert..
Zoom
Constraint package
holds constraints
Common
Note
Separator
Class Diagram
Class
Interface
Package
Generalizat...
Association
Aggregation
Composition
Interface ...
Usage
Abstraction
Collaboration
Instance
Use Case Diagram
Implementation Di...
Composite Struct...
Information Flows
Profiling Mechanism
Untitled1
125%
OrderItem
amount : UnlimitedNatural
orderItem
0..*
orderItem
0..*
isIncludedIn
isInvolvedIn
order
1
item
1
<<constrained>>
Order
price : UnlimitedNatural
submits(customer : Subject, order : Order) : void
<<constrained>>approves(agent : Agent, order : Order) : void
Item
itemName : String
order
0..*
submits
customer
1
Subject
name : String
ValidOrder
Individual
Agent
Legal


MagicDraw UML 16.5 - Untitled1
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
Untitled1
VeTIS
Containment
ders_imported
Relations
Constraints
{ } invOrder1=self.price<=10000
{ } preApproves1=self.oclIsType(
VeTIS
Agent
Individual
Item
Legal
Order
OrderItem
Subject
Zoom
Docum..
Propert..
Zoom
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
Class Diagram
Class
Interface
Package
Generalizat...
Association
Aggregation
Composition
Interface ...
Usage
Abstraction
Collaboration
Instance
Use Case Diagram
Implementation Di...
Composite Struct...
Information Flows
Profiling Mechanism
Untitled1
125%
OrderItem
amount : UnlimitedNatural
orderItem
0..*
orderItem
0..*
isIncludedIn
isInvolvedIn
order
1
item
1
<<constrained>>
Order
price : UnlimitedNatural
submits(customer : Subject, order : Order) : void
<<constrained>>approves(agent : Agent, order : Order) : void
Item
itemName : String
order
0..*
submits
customer
1
Subject
name : String
ValidOrder
Visualize constraints from the contextual menu
Individual
Agent
Legal


MagicDraw UML 16.5 - Untitled1
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
Untitled1
VeTIS
Containment
ders_imported
Relations
Constraints
{ } invOrder1=self.price<=10000
{ } preApproves1=self.oclIsType(
VeTIS
Agent
Individual
Item
Legal
Order
OrderItem
Subject
Zoom
Docum..
Propert..
Zoom
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
Class Diagram
Class
Interface
Package
Generalizat...
Association
Aggregation
Composition
Interface ...
Usage
Abstraction
Collaboration
Instance
Link
Use Case Diagram
Implementation Di...
Composite Struct...
Information Flows
Profiling Mechanism
Untitled1
125%
Order
price : UnlimitedNatural
submits(customer : Subject, order : Order) : void
<<constrained>>approves(agent : Agent, order : Order) : void
itemName : String
order
submits
customer
1
Subject
name : String
ValidOrder
{invOrder1=self.price<=10000}
{preApproves1=self.oclIsTypeOf(ValidOrder)}
Individual
Legal
Agent

7 priedas. Tyrimo planas 2008-2010 metais

Prognozuojama tyrimo darbų vykdymo pradžia – 2008 m. spalio-lapkričio mėnesiai. Tyrimas bus vykdomas iki 2010 m. pradžios. Projekto vykdymo darbų planas, darbus paskirstant metų ketvirčių tikslumu, pateikiamas 9.1 lentelėje. Detalesnis suplanavimas nėra tikslingas. Greta darbų pavadinimų nurodoma jų numatoma apytikslė trukmė mėnesiais.

9.1 lentelė. Tyrimo planas 2008-2010 metais

Nr.	Darbo pavadinimas	Trukmė mėn.	2008 m.		2009 m.				2010 m.	
			III	IV	I	II	III	IV	I	II
1.	Esamų veiklos taisyklių (VT) surinkimo ir specifikavimo metodų analizė.	*1-2*								
2.	Veiklos žodyno išgavimo iš probleminės srities tyrimas.	*1-2*								
3.	VŽ integravimo į informacinių sistemų (IS) kūrimą analizė.	*2-4*								
4.	VŽ atvaizdavimo programiniame kode galimybių analizė.	*2*								
5.	VŽ transformavimo metodo sukūrimas.	*3-4*								
6.	Sukurtojo metodo realizacija ir jo įvertinimas.	*6*								
7.	Sukurtojo įskiepio testavimas ir metodo įvertinimas.	*1*								