KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Giedrius Tamulis

Dalykinės srities kalbų kūrimo UML MagicDraw aplinkoje metodika ir šios metodikos pritaikymas, kuriant veiklos objektų modeliavimo kalbą

Magistro darbas

Vadovas
doc. dr. T. Skersys

KAUNAS, 2009

Turinys

Paveikslėlių turinys

THE METHODOLOGY OF DSL DEVELOPMENT USING CASE TOOL UML MAGICDRAW AND IT'S APPLICATION IN THE DEVELOPMENT OF BUSINESS OBJECTS MODELING LANGUAGE

SUMMARY

Within Information systems engineering process is a major problem – information systems and domain experts can't easily understand each other language. The reason is that the domain and IS expert's uses the different terminology. Different terminology increases the information system development time, because there is a need to re-review the requirements and reassessment of the prototype.

One of the ways of solving this problem is the Domain Specific Language development and use in various problem domains. Usage of common language facilitates the communication between IS and domain experts.

There is no methodology of Domain Specific Language development by CASE tools. Therefore possible solution is proposition of this methodology.

In this paper is described the solution of both problems (methodology of DSL development and proposed language of class model extension with enterprise model objects). Proposed language should optimize existing class model making it more understandable for domain experts.

Key wodrs: Domain Specific Language (DSL); UML; Enterprise model; „MagicDraw";

1.ĮVADAS

Kompiuterizuojant įvairiausias gyvenimo sritis susiduriama su pakankamai didele problema programavimo kalbos, reikalavimų specifikavimo įrankio pasirinkimu. Žinoma programuotojai renkasi kalbą kurią geriausia moka, projektuotojai renkasi tokius įrankius kokius turi ar geriausiai moka jais dirbti. Šitaip pasirinkus įrankius atsiranda supratimo spragos tarp užsakovų ir projektuotojų, tarp projektuotojų ir programuotojų. Dėl nesusikalbėjimo tarp dalykinės strities (DS) specialistų ir informacinių technologijų specialistų (IT) kyla kompiuterizuojamos sistemos kūrimo kaštai, IT specialistams reikia dažnai pristatinėti prototipus norint kuo ankščiau identifikuoti klaidas. Šis nesusikalbėjimas atsiranda dėl to, kad DS specialistai ir IT specialistai vartoja skirtingus terminus, naudojasi skirtingais projektavimo ir programavimo įrankiais. Norint, kad DS ir IT specialistai suprastų vieni kitus adekvačiai, reikia vartoti vieną kalbą.

Vienas iš šios problemos sprendimo būdų yra DSL (angl. Domain Specific Language), toliau bus naudojama DSK (Dalykinės Srities Kalba) kalbų naudojimas. Bendros kalbos įvedimas palengvina kompiuterizuojamos srities ir IT specialistų bendravimą, išgaunami detalesni reikalavimai sistemai, sumažėja programinės įrangos kūrimo kaštai, sumažėja sistemos kūrimo laikas, sumažėja tikimybė, kad sistema iš viso nebus sukurta. Dalykinės kalbos naudojimas įtraukia dalykinės srities atstovus į sistemos kūrimą, dėl to sukurtos sistemos validavimas atliekamas greičiau ir kokybiškiau. Tuo pačiu dalykinės srities atstovas gali įtraukti reikiamus pakeitimus kurie bus vienareikšmiškai suprasti tiek IT specialisto tiek programuotojo.

Problemos sprendimui pasirinktas UML MagicDraw paketas, nes tai yra populiarus sistemų modeliavimo įrankis, be to šiame pakete numatyta galimybė kurti DSL kalbas, tačiau nėra metodikos kaip tai daryti. UML MagicDraw paketas pasirinktas ir dėl to, kad KTU bendradarbiauja su kompanija „NoMagic“ kuri kuria ir palaiko UML MagicDraw paketą. Taip pat Informacijos Sistemų katedroje atliekami bandymai kuriems reikia galimybės metodiškai praplėsti UML metamodelį.

Darbe nagrinėjama *tyrimo sritis* – dalykinių kalbų kūrimo metodikos bei veiklos objektų modeliavimas (T.Skersio ir S.Gudo pasiūlyto veiklos modelio pagrindu).

Galima išskirti du *tyrimo tikslus*:

1) Suteikti galimybę UML MagicDraw sistemos vartotojams metodiškai kurti savo dalykinės srities kalbas pateiktos dalykinės srities kalbų kūrimo metodikos pagrindu;
2) Suteikti galimybę UML MagicDraw sistemos vartotojams modeliuoti veiklos objektus.

Šiuo metu nėra metodikos dalykinės srities kalbų (DSK) kūrimui UML MagicDraw aplinkoje. Sudarius metodiką DSK kūrimui UML MagicDraw vartotojai galės greičiau, ir kokybiškiau kurti dalykinės srities kalbas. Galima išskirti dvi pagrindines vartotojų grupes tai informacinių techno-

logijų (IT) specialistai(informacinių sistemų projektuotojai, programuotojai) ir kompiuterizuojamos srities (*angl. domain*) specialistai.

IT specialistai pasižymi tuo, kad yra gerai susipažinusi su informacinių sistemų kūrimo terminologija, tačiau jiems kyla sunkumų su specifiniais, tam tikros dalykinės srities terminais. Problema su kuria susiduria IT kūrėjai yra ta, kad sudėtinga sukurti sistemą skirtą tam tikrai sričiai, net jeigu ir yra sukurta labai panaši sistema, ši problema sąlygoja ilgą sistemos pateikimo rinkai laiką.

Kompiuterizuojamos srities specialistai, atvirkščiai puikiai žino savo srities terminus ir jų prasmę, tačiau prastai supranta IT specialistų žodyną. DS specialistai ir darbuotojai prastai išmano informacinių sistemų kūrimo subtilybes. Dalykinės srities ekspertai negali būti tikri, kad IT specialistai tinkamai įgyvendino problemos sprendimą, dėl šios problemos galimas nekorektiškas sistemos veikimas.

Atsižvelgiant į vartotojų savybes ir problemas buvo suformuluoti pagrindiniai uždaviniai, padėsiantys pasiekti užsibrėžtą tikslą.

Tyrimo uždaviniai:

1) Išanalizuoti dalykinių kalbų kūrimo metodikas
2) Išanalizuoti UML MagicDraw paketo dalykinės srities kalbų kūrimo posistemį
3) Sukurti dalykinės srities kalbų kūrimo metodiką UML MagicDraw paketui
4) Sukurti veiklos objektų modeliavimo kalbą, taikant pasiūlytąją metodiką
5) Realizuoti veiklos objektų modelių korektiškumo patikrinimą
6) Sudaryti veiklos objektų modelį bei patikrinti jo korektiškumą.

Atlikus darbą buvo išanalizuotos dalykinės srities kalbų kūrimo metodikos ir nustatyta, kad norint sukurti DSK visų pirma reikia išanalizuoti dalykinę sritį ir susidaryti jos modelį(konceptai, ryšiai tarp konceptų, apribojimai).

Pasinaudojant sudaryta metodika buvo sukurta veiklos objektų modeliavimo kalba (Informacijos sistemų katedroje sudarytu) veiklos modelio pagrindu. Veiklos objektų modeliavimo kalba leidžia modeliuoti veiklos procesus klasių diagramoje, tokiu būtu veiklos procesus priartinanti prie informacinių sistemų (IS) inžinerijos ir suteikia galimybę automatizuoti IS inžinerijos procesą.

2.DALYKINĖS SRITIES KALBŲ ANALIZĖ

Šios analizės tikslas, išanalizuoti dalykinės srities kalbų kūrimo metodikas. Išanalizuoti galimybes UML MagicDraw priemone kurti dalykinės srities kalbas.

[1] Visuose mokslo ir inžinerijos šakose galima išskirti du požiūrius: bendrą (*angl. generic*) ir specifinį (*angl. specific*). Bendras metodas duoda nespecializuotą sprendimą daugumai problemų tam tikroje srityje, tačiau tas sprendimas gali būti neoptimalus. Specializuotas metodas yra daug geresnis sprendimas mažesnėje problemų aibėje. Vienas iš specializuotų metodų yra DSL (*domain specific language*) kalbų naudojimas.

[1] *Domain specific language* (Dalykinės Srities Kalba(DSK)) yra maža, dažniausiai deklaratyvi kalba kurios visas dėmesys sufokusuotas į dalykinę sritį. Daugumoje atvejų dalykinės srities kalba išreiškiama per standartines įvairių programavimo kalbų bibliotekas, be to dalykinės srities kalbos paslepia dalį šių bibliotekų detalių, kurios nereikalingos tam tikros dalykinės srities problemoms spręsti.

2.1.Dalykinės srities kalbų tipai ir savybės

Dalykinės srities kalbos kaip ir kitos programavimo kalbos gali būti grafinės arba tekstinės. Grafinės kalbos turi privalumų prieš tekstines, nes jos leidžia sprendimą pateikti vizualiai (diagramomis). Naudojant grafines programavimo kalbas stengiamasis panaudoti kompiuterio resursus tam, kad vartotojui būtų pateiktas sprendimas kiek galima artimesnis jo dalykinei sričiai. Pavyzdžiui kompozitoriui žymiai paprasčiau naudoti penklinę ir visus jos žymėjimus, nei tekstinėje programavimo kalboje kažkokiu būdu užrašinėti natų sekas ar pan. Grafinės programavimo kalbos pradėjo plėtotis visai neseniai, maždaug tuo metu kai išpopulerėjo Unified Modeling Language (UML).

2.1.1.Tekstinės dalykinės srities kalbos

Yra daug tekstinių dalykinės srities kalbų(DSK), tačiau jas vartodami net nesusimastome, kad tai DSK. Galima paminėti kelias dažniausiai vartojamas dalykinės srities kalbas: SQL, HTML, XQUERY/XPATH...Tarkim HTML yra dalykinės srities kalba skirta internetinių puslapių kūrimui. SQL (*angl. Structured Query Language*)- struktūrizuota užklausų kalba.

Paimkime SQL kaip pavyzdį kuo skiriasi dalykinės srities kalba nuo bendros paskirties kalbos, šiuo atveju C#. Pavyzdyje pateikiama užklausa iš pradžių parašyta bendros paskirties kalba, o po to DSK (SQL), ir duodanti tą patį rezultatą. C# kalba parašyta užklausa atrodytų taip:

```
SelectOutput output = new SelectOutput();
foreach (TableRow row in  Tables["Invoices"].FindRecords(new WhereStatement(new
EqualityPredicate("Status", "Open")))
 output.Add(Tables["Customers"].FindRecord(new WhereStatement(new EqualityPredi-
cate("CustomerID", row["CustomerID"]))));
reutrn output;
```

Kaip matome bendros paskirties kalbos naudojimas pakankamai siauroje srityje apkrauna svarbius dalykinės srities aspektus iš pažiūros nereikalinga informacija. Bereikalinga informacija bendros paskirties kalboje užgožia tai ką iš tiesų norima pasakyti. Štai kaip ta pati užklausa atrodo parašyta SQL kalba:

```
SELECT Customers.CustomerID FROM Invoices INNER JOIN Customers ON Invoices.Cus-
tomerID = Customers.CustomerID WHERE Invoices.Status = "Open"
```

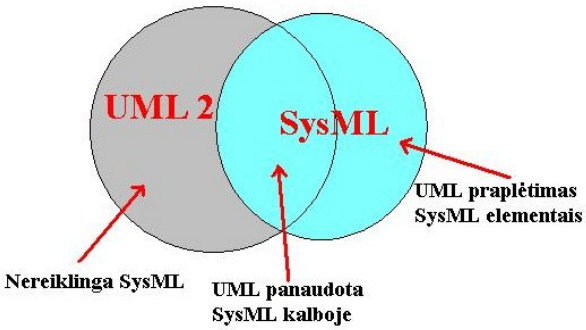
Šis užrašas žymiai aiškesnis. Sunku įsivaizduoti, kad reliacinių duomenų bazių specialistas rašytų užklausas bendros paskirties kalba. Taip pat jam reikėtų žymiai daugiau žynių išeinančių už dalykinės srities ribų, kad tai galėtų atlikti.

2.1.2.Grafinės dalykinės srities kalbos

[6]Grafinės dalykinės srities kalbos nėra vien diagramos. Jei norima sukurti tik diagramas geriau naudotis populiariomis piešimo programomis. Iš tikro grafinėmis dalykinės srities kalbomis kuriami modeliai kurie konceptualiai pristato kuriamą sistemą, kartu vizualiai pristatomas ir sistemos turinys (*angl. content*).

[6]Dauguma vartotojų norėtų, kad dalykinės srities kalboje būtų apjungta ir tekstinė ir grafinė kalba. Daug žmonių pasirinktų tekstinę kalbą kaip įėjimą, nes jie gali greitai rinkti tekstą, o grafinę kalbą kaip išėjimą, nes daug lengviau diagramoje matyti „dideli paveikslą“ nei tekstą. Tekstinės išraiškos labiau tinka skaičiuoti skirtumus ir suliejimus (*angl. merge*), o grafinės – geriau išreiškia ryšius (*angl. relationship*).

Turbūt viena labiausiai žinomų grafinių dalykinės srities kalbų yra SysML (*angl. Systems modeling language*) - sistemų modeliavimo kalba. SysML yra grafinė modeliavimo kalba skirta sudėtingų sistemų analizavimui, projektavimui bei patikrinimui ir galinti apimti aparatinę įrangą, programinę įrangą, informaciją, personalą, procedūras bei patogumus. Šio kalbos grafinė semantika skirta sistemos reikalavimams, elgesiui, struktūrai, ir parametrams pavaizduoti. Taip pat yra galimybė sistemą integruoti su kitais taikomaisiais analizės modeliais. SysML yra UML išplėtimas (1 pav.) skirtas sistemų modeliavimui.



Pav. 1 Ryšys tarp UML 2 ir SysML

Kaip matome iš (1 pav.) SysML naudoja tam tikrą UML elementų poaibį, o likusios dalies atsisako tam, kad kalba nebūtų apkrauta sistemų modeliavimo sričiai nereikalingomis žiniomis. Žinoma kalboje atsiranda ir specifinių dalykinės srities modeliavimui reikalingų elementų.

UML MagicDraw pakete realizuotas UML kalbos praplėtimo mechanizmas, todėl pasinaudojus šiuo mechanizmu galima kurti naujas diagramas vaizduojančias norimą dalykinę sritį ar tos srities dalį.

2.2. DSK kūrimo trūkumai ir galimybės

Pasirinkus dalykinės srities kalba pagrįstą programinės įrangos kūrimo būdą, atsiranda tam tikri pavojai ir tam tikros galimybės. Gerai sukurta dalykinės srities kalba suteikia galimybę rasti balansą tarp trūkumų ir galimybių.

2.2.1.DSK kalbos galimybės:

1) Suteikia galimybę pateikti sprendimą dalykinės srities abstrakcijos lygmenyje. Dėl šios priežasties dalykinės srities ekspertai gali suprasti, validuoti, modifikuoti ir dažniausiai netgi kurti DSL kalbomis pagrįstas programas.
2) Programos yra glaustos, yra dokumentacijos generavimas(*angl. self-documenting*), ir gali būti panaudotos kitiems uždaviniams spręsti.
3) Naudojimas didina produktyvumą, lankstumą, palaikomumą ir portatyvumą
4) Naudojimas apima dalykinės srities išmanymą, tai įgalina saugojimą ir pakartotinį žinių panaudojimą.
5) Kalbos sudaro sąlygas programos validavimui ir optimizavimui dalykinės srities lygyje.

2.2.2.DSK naudojimo trūkumai:

1) Kalbos kūrimo išlaidos.
2) Išlaidos skirtos dalykinės srities ekspertams apmokyti.
3) Ribotas kalbos naudojimas.
4) Sunkumai nustatant tinkamą sritį kurią turi apimti kalba.
5) Sunkumai susiję su balansavimu tarp bendros ir specializuotos programavimo kalbų.
6) Galimas efektyvumo praradimas lyginant su rankomis rašyta programine įranga.

Pradedant kurti dalykinės srities kalbą reikia atsižvelgti tiek į dalykinės srities kalbomis kuriamos sistemos privalumus tiek į trūkumus. Gali nutikti taip, kad kuriant sistemą dalykinės srities kalbos pagrindu sistemos kaštai bus didesni nei kuriant analogišką sistemą įprastiniu būdu.

2.3.Esama dalykinės srities kalbų kūrimo metodika

Remiantis [1] DSK kūrimo procesas paprastai apima šiuos žingsnius:

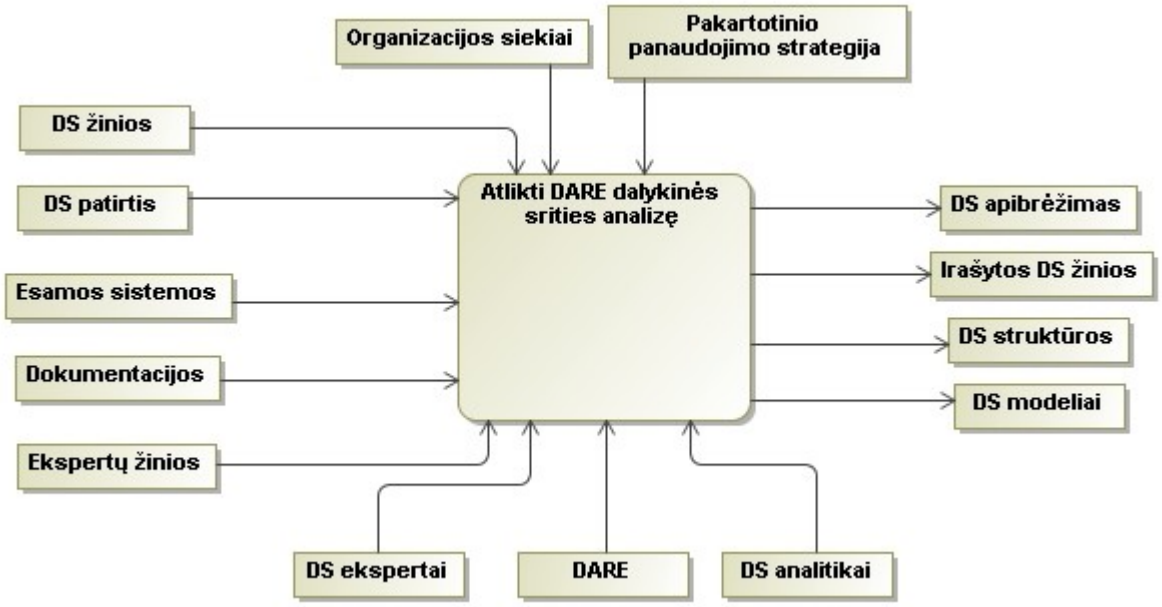
1) Dalykinės srities identifikavimas.
2) Visų reikalingų žinių apie dalykinę sritį surinkimas
3) Žinių sugrupavimas į nedidelius semantinius vienetus ir operacijas tarp jų.
4) Dalykinės srities kalbos kūrimas, kuris glaustai apibūdina taikomuosius uždavinius dalykinėje srityje(Įgyvendinimas).
5) Semantinių vienetų įtraukimas į biblioteką.
6) Kompiliatoriaus kuris išverčia dalykinės srities kalbos programas į kreipinių į biblioteką sekas kūrimas ir įgyvendinimas.
7) Dalykinės srities programų rašymas ir kompiliavimas.

Analizės etapo (apima nuo 1 iki 4 žingsnio) tikslas nuodugniai išsiaiškinti ir suprasti pagrindinius dalykinės srities aspektus.[2] Analizės etapo įėjimai yra tikslios ar netikslios žinios apie dalykinę sritį, tokios kaip techniniai dokumentai, žinios iš dalykinės srities ekspertų, egzistuojantis bendros paskirties kalbos (*angl.General Porpose Language*) kodas, užsakovų (*angl. costumers*) apklausa. Analizės išėjimai kinta plačiame reikšmių diapazone, bet dažniausiai susideda iš dalykinės srities terminologijos ir semantikos, kuri yra daugiau mažiau abstrakčios formos. Egzistuoja labai artimas ryšys tarp dalykinės srities analizės ir žinių inžinerijos (*angl. knowledge engineering*), kuri šiuo metu tik pradedama tirti. Žinių fiksavimas (*angl. capture*), žinių pristatymas ir ontologinis plėtojimas yra naudingas analizės etape.

Dažniausiai dalykinės srities analizė atliekama neformaliai, tačiau kartais naudojami formalūs dalykinės srities analizės metodai. Tokių metodų pavyzdžiai yra: DARE (*Domain Analysis and Reuse Environment)*[3], Savybėmis grindžiama dalykinės srities analizė (*angl. Feature-Oriented Domain Analysis (FODA)*)[5].

Dalykinės srities analizės ir pakartotinio panaudojimo *DARE* metodas grindžiamas dalykinės srities analizės proceso automatizavimu. Į DARE metodą pažvelgsime kaip į juodą dėžę (2 pav.), nesigilindami į logiką. Dalykinės srities analizės procesas turi penkis įėjimus. Pirmi du dalykinės srities žinios ir dalykinės srities patirtis į kuriuos įeina bendros ir kartais neformaliai dokumentuotos žinios apie dalykinę sritį. Šios žinios pirmiausia yra naudojamos dalykinės srities apibūdinimui ir identifikavimui. Tačiau DARE metodas negali šių žinių gauti tiesiogiai iš dalykinės srities, žinias netiesiogiai perteikia analitikai ir ekspertai. Kiti trys įėjimai egzistuojančios sistemos, dokumentai susiję su DS, ekspertų žinios yra DARE metodo įėjimų branduolys. DARE metodas naudoja

parašytą dokumentaciją kaip žinių šaltinį ir naudodamasi teksto analizės technika abstraguoja ir struktūrizuoja žinias apie dalykinę sritį.



Pav. 2 DARE dalykinės srities analizės metodas

Atlikus DARE dalykinės srities analizę gauname DS apibrėžimus, struktūrizuotas žinias apie DS bei DS modelius. Analizės rezultatus naudojame kaip dalykinės kalbos kūrimo proceso įėjimus.

Savybėmis grindžiama dalykinės srities analizė (*FODA*), yra dalykinės srities analizės metodika, sukurta Software Engineering Institute. FODA metodas pagrįstas bendrų savybių sistemos klasėms identifikavimu. Tai yra keleto požiūrių į probleminę sritį studijavimo ir įvertinimo rezultatas.

FODA metodas nustato tris pagrindines veiklas: konteksto analizė, dalykinės srities modeliavimas, architektūros modeliavimas. Atliekant konteksto analizę, dalykinės srities analitikas bendrauja su vartotojais ir DS ekspertais išgaudamas pradinę informaciją. Dalykinės srities modeliavimo etape sukūrimas DS modelis kuris pateikiamas keletu pjūvių. DS analitikas pateikia DS modelį srities vartotojų ir ekspertų peržiūrai. Galutinis modelis susideda iš keturių pjūvių: Savybių modelio, esybių-ryšių modelio, duomenų srautų diagramos modelio ir būsenų perėjimų modelio. Standartinis žodynas taip pat gaunamas DS modeliavimo etape.

Formalaus analizės metodo išėjimas yra dalykinės srities modelis kuris susideda iš:

1) Dalykinės srities apibrėžimas kuris apima visą dalykinę sritį.
2) Dalykinės srities terminologija (žodynas, ontologija).
3) Dalykinės srities sąvokų toliau naudojamų kaip konceptų (*angl. concepts*) apibrėžimai.

4) Požymių(*angl. feature*) modelių aprašančių dalykinės srities vienetų panašumus ir skirtumus ir jų priklausomybes.

Kol kas nėra aiškių gairių kaip iš analizės metu surinktos informacijos sukurti DSL kalbą.

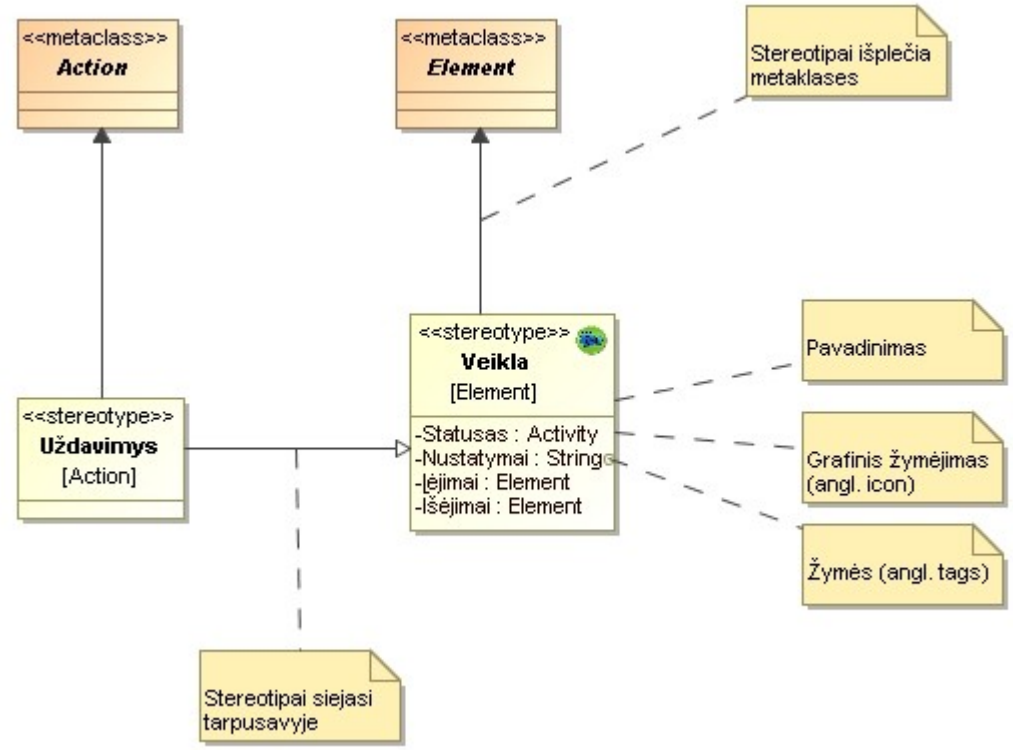
Įgyvendinimo etapas (apima 5 ir 6 žingsnius)gali būti realizuojamas šiais būdais:

1) Klasikinis būdas-kurti naują programavimo kalbą. Šiuo būdu nauja kalba kuriama nuo pamatų, sukuriama kalbos sintaksė, semantika, sukuriamas kompiliatorius interpretuojantis kalbos elementus.
2) Įterptinės kalbos (*angl. embeded languages*) specializuotos dalykinės srities bibliotekos. Pasirinkus šį būdą, operacijoms tarp dalykinės srities vienetų atlikti yra naudojamos standartinės funkcijos ir operatoriai. Pagrindinės (*angl. base*) kalbos sintaksės mechanizmai yra naudojami dalykinės srities savitumui išreikšti.

Išanalizavus UML Magic Draw paketą buvo nustatytos galimybės praplėsti UML metamodelį naujais,skirtais tam tikrai dalykinei sričiai, elementais.

2.4.UML praplėtimo mechanizmas

UML išplėtimo mechanizmai leidžia išplėsti ir pritaikyti UML kalbą, papildant naujais blokais (*angl. blocks*) , sukuriant naujas ypatybes (*angl. properties*), specifikuojant naują semantiką, tam, kad nauja kalba būtų tinkama specifiniams uždaviniams spręsti. Remiantis [4] bei [7] nustatyta, kad yra trys bendrai apibrėžti UML išplėtimo mechanizmai: **stereotipai** (*angl. stereotypes*), **žymėtosios vertės** (*angl. taged values*) ir **apribojimai** (*angl. constraint*). UML praplėtimo modelis pateikiamas (3 pav.)



Pav. 3 UML išplėtimo mechanizmas

Pateiktame UML praplėtimo modelyje matome, kad UML MagicDraw aplinkoje galime susikurti ne tik naujus stereotipus atitinkančius realaus pasaulio konceptus, jiem priskirti grafinius žymėjimus, apriboti stereotipų stereotipus taisyklėmis, bet taip pat sieti vienus stereotipus su kitais atitinkamais ryšiais(paveldėjimo, asociacijos, priklausomybės ir kt.)

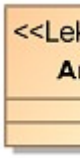
2.4.1.Stereotipai

Stereotipai leidžia išplėsti UML žodyną, taigi galima sukurti naujus modelio elementus gautus iš jau egzistuojančių, bet turinčius išskirtinių savybių kurios yra tinkamos dalykinei sričiai. Stereotipai naudojami klasifikavimui ar UML statybinių (*angl. building*) blokų žymėjimui tam, kad naujas elementas atitiktų realaus pasaulio konceptą.

Pavyzdžiui: kai modeliuojamas tinklas reikia turėti elementus kurie vaizduotų šakotuvus ir skirstytuvus (*angl. routers and hubs*) . Naudojant stereotipus galima padaryti, kad šie elementai atsirastų kaip įprasti elementai (pvz. Class).

Stereotipai taip pat leidžia jums įvesti naujus grafinius simbolius tam, kad aprūpintų elementus regimaisiais ženklais, kurie atspindi realaus pasaulio konceptus

Grafiškai stereotipas yra pavadinimas laužtiniuose skliaustuose (4 pav.), taip pat stereotipas gali būti su pavadinimu ir grafiniu simboliu (5 pav.), ir gali būti vaizduojamas vien kaip grafinis simbolis (6 pav.).



Pav. 4 Pavadintas stereotipas

Pav. 5 Pavadintas stereotipas su grafiniu žymėjimu

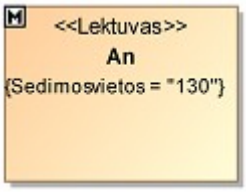
Pav. 6 Stereotipas su grafiniu žymėjimu

2.4.2.Žymėtosios vertės (*angl. taged values*)

Žymėtosios vertės suteikia galimybę išplėsti UML elementų ypatybes (angl. properties), taip sukuriant naują informaciją apie elementą. Žymėmis galima išplėsti jau esamus UML elementus arba individualius stereotipus, taip kiekvienas elementas kuriam bus priskirtas šis stereotipas tuo pačiu turės ir tam tikrą ypatybę. Reikia pabrėžti, kad žymės **nėra lygios** klasės atributams. Žymėtąją reikšmę galima įvertinti kaip metaduomenis, nes jos vertė prisitaiko prie elemento automatiškai.

Kaip žymėtųjų verčių naudojimo pavyzdys gali būti komanda kuri kuria, testuoja, diegia naujas programinės įrangos versijas. Tokiu atveju naudinga žymėtose vertėse saugoti kiekvienos posistemės versijos numerį ir testavimo rezultatus.

Grafiškai žymėtosios vertės vaizduojamos kaip simbolių eilutė laužtiniuose skliaustuose, kuri yra vaizduojama po modelio elemento vardu. Simbolių eilutė susideda iš vardo (žymėtoji vertė), skyriklio(simbolis ‚=') ir reikšmės (žiūr. 7 pav.).



Pav. 7 Stereotipas su žymėtąja verte

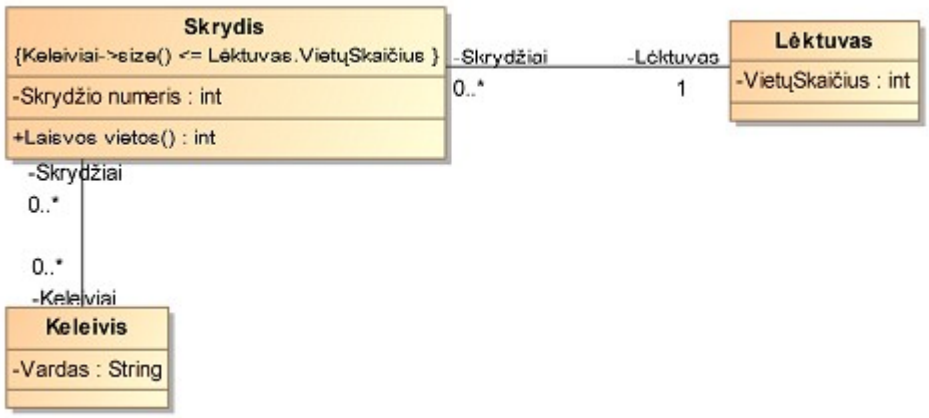
2.4.3.Apribojimai *(angl. constraints)*

Apribojimai yra ypatybės tam, kad apibrėžtų semantika ir/arba sąlygas kurios visada turi būti teisingos kiekvienam modelio elementui. Apribojimai leidžia išplėsti UML elementų semantika, pridedant naujas taisykles ar pakeičiant jau esančias.

Pavyzdžiui modeliuojant realaus laiko sistemas gali būti naudinga į modelį įtraukti tam tikrą informaciją: laiko apribojimus, galutinius terminus. Naudojant apribojimus šitie laiko reikalavimai lengvai gali būti patikrinti ir sugauti.

Grafiškai apribojimai vaizduojami kaip simbolių eilutė (*angl. string*) tarp laužtinių skliaustų, skliaustuose yra elementai susieti asociacijos ryšiu arba priklausomybės ryšiu.

8 pav. pavaizduotas skrydžio modelis ir apribojimu aprašyta, kad keleivių skaičius negali viršyti sėdimų vietų skaičiaus.



Pav. 8 Apribojimo pavyzdys

Apribojimai rašomi OCL (*angl. Object Constraint Language*) objektų apribojimų kalba. Taip pat apribojimai gali būti rašomi ir natūralia kalba, tačiau tokiu atveju apribojimai netenka funkcionalumo, nes sistema negali panaudoti taip užrašytų apribojimų tarkim modelio korektiškumo patikrinimui.

2.5.Nefunkciniai sukurtos kalbos reikalavimai

Remiantis [5] naujai sukurta dalykinės srities kalba (naudojantis kuriama metodika) turi atitikti šiuos reikalavimus:

1) Atitikimas - kalbos konstrukcijos turi atitikti svarbius dalykinės srities vienetus (angl. domain concepts)
2) Ortogonalumas – kiekviena kalbos konstrukcija turi atitikti tik vieną dalykinės srities vienetą.
3) Suderinamumas(angl. supportability) – dalykinės srities kalba turi būti suderinta su įrankiu, t.y turi būti numatytas tipinis programos ir modelio valdymas, pvz:. elementų sukūrimas, ištrynimas, testavimas, transformavimas.
4) Integralumas – kalba ir jos įrankis gali būti naudojamas kartu su kitomis kalbomis ir įrankiais su minimaliomis pastangomis.
5) Ilgaamžiškumas – dalykinės srities turi būti naudojamas ir naudingas neribotą laiką.
6) Paprastumas – tai pagrindinis kalbos reikalavimas, kalba turi būti kiek įmanoma paprastesnė.
7) Kokybė – kalba turi užtikrinti, kokybiškos sistemos kūrimą.

UML metamodelio praplėtimas patogus tuo, kad naudojant UML modeliavimo įrankius automatiškai užtikrinami ir kai kurie nefunkciniai naujos kalbos reikalavimai. Automatiškai užtikrinami tokie nefunkciniai reikalavimai kaip: Suderinamumas – nes UML modeliavimo įrankiuose numatytas modelio valdymas(elementų sukūrimas, ištrynimas, testavimas, transformavimas), taip pat integralumas, nes naujai sukurta kalba suderinama su UML kalba, kuri savo ruoštu suderinama su kitomis kalbomis. Reikia paminėti, vienu UML modeliavimo įrankiu realizuota sistema gali būti modifikuojama ar peržiūrima kitu įrankiu, tam panaudojant bendrus standartus tokius kaip XML.

2.6.DSK kūrimo metodikų ir UML praplėtimo analizės rezultatai

Atlikus dalykinės srities kalbų kūrimo metodikų analizę nustatyta, kad:

1) Kuriant naują DSK būtina išgauti žinias apie dalykinę sritį. Žinių išgavimui gali būti naudojami įvairūs analizės metodai(FODA, DARE) taip pat ir neformalūs.
2) Išgavus žinias reikia susidaryti dalykinės srities modelį. Modelį sudaro dalykinės srities konceptai su savo savybėmis, ryšiai tarp konceptų. Dalykinės srities modelis gali būti gautas ir kaip formalaus analizės metodo išėjimas.
3) Kuriant grafines dalykinės srities kalbas UML pagrindu, UML metamodelį galima išplėsti trim būdais : Stereotipais, žymėmis, apribojimai.
4) UML išplėtimas stereotipais leidžia sukurti naujus modelio elementus atitinkančius realius dalykinės srities konceptus. Stereotipai atitinka konceptus ne tik savo pavadinimu, tačiau gali atitikti ir išvaizda (priskirtas grafinis žymėjimas), bei elgsena (ryšiai su kitais elementais, apribojimai)
5) Žymių panaudojimas leidžia išskirti tam tikromis savybėmis pasižyminčius modelio elementus. Žymės gali būti panaudotos tiek jau egzistuojantiems UML elementams tiek naujai sukurtiems stereotipams.

2.7.Veiklos objektų modeliavimo srities analizė

Dabar efektyvių informacinių sistemų (IS) inžinerija ir veiklos modeliavimas yra tiesiogiai susiję problemos. Veiklos modeliavimas yra laikomas kaip veiklos žinių šaltinis kuris prideda vertę verslo procesams ir tuo pačiu daro įtaką IS projektavimo metodams. Dalis IS inžinerijos sprendimų naudoja veiklos modelius kaip struktūrizuotą žinių šaltinį apie tikrą pasaulį (dalykinę sritį) IS kūrimo gyvavimo ciklo etapuose, tokiuose kaip reikalavimų analizės ir specifikavimo, detalaus IS sprendimo sukūrimui ir kt.

[9]Skirtingi veiklos modeliai tampa pagrindiniais CASE (Computer-Aided System Engineering) sistemų komponentais: plačiausiai naudojami modeliai yra duomenų srautų (dataflow) diagramos, darbų sekų (workflow) diagramos, organizacijos hierarchijos, procesų hierarchijos, verslo tikslų, verslo sąveikos diagramos, UML ir IDEF diagramos komplektai, ir taip toliau. Problema yra ta, kad visos šios diagramos yra silpnai susietos, todėl tai veda prie visiško IS kūrimo ciklo nesuderinamumo ir negali duoti lauktų rezultatų.

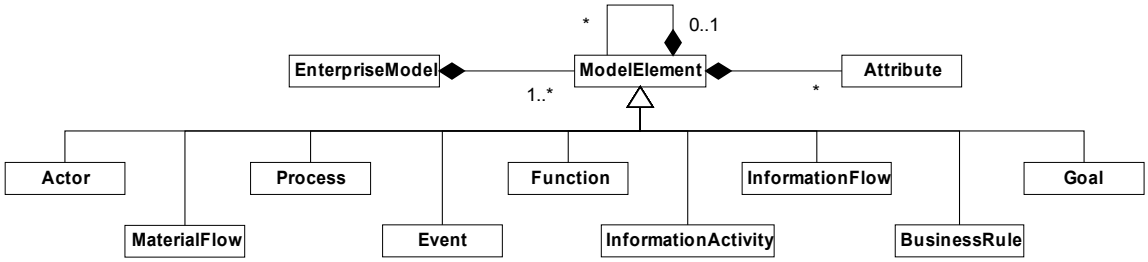
Šios problemos sprendimui buvo pasiūlytas veiklos modelis kuris leidžia visą veiklą modeliuoti vienoje diagramoje. Veiklos objektų (VO) modeliavimui buvo pasirinkta klasių diagrama, nes klasių diagrama yra pamatinė diagrama vykdant IS inžineriją. Dauguma CASE įrankių iš klasių diagramos gali generuoti programinį kodą, taip pat atlikti atvirkštinę inžineriją. Tačiau buvo susidurta su viena problema, kad reiklia pritaikyti klasių modelį veiklos modeliavimui. Klasių modelio praplėtimui panaudosime DSK kūrimo metodiką UML MagicDraw aplinkoje.

Tam, kad pailiustruotumėme kuriamos metodikos veiksmingumą, sukursime dalykinės srities kalbą skirtą veiklos objektų modeliavimui klasių pagringu. Žinios apie dalykinę sritį bus paimtos iš [9] pateikto veiklos modelio, bei jo aprašo. Turint dalykinės srities metamodelį (modelio modelį) nebereikia atlikti formalios analizės tam, kad identifikuotumėme dalykinės srities konceptus, ryšius tarp jų bei apribojimus.

2.8.Formalizuotas veiklos metamodelio aprašas

Remiantis [8] ir [9] išanalizuota veiklos objektų modeliavimo dalykinė sritis ir gautas formalus šios srities aprašas. Konceptai įvardijami ir angliškais terminais, nes esant poreikiui realizuoti smulkiai išdetalizuotą modelį priimta atributų,operacijų, klasių, taipogi ir stereotipų pavadinimus rašyti angliškai.

Pasiūlytąjį veiklos metamodelį VMM (9, 10 pav.) sudaro pagrindiniai veiklos modeliavimo konstruktai, įvertinus veiklos procesų valdymo aspektą. Apibrėšime VMM elementų sudėtį ir paaiškinsime jų paskirtį organizacijos veiklos modeliavimo kontekste.



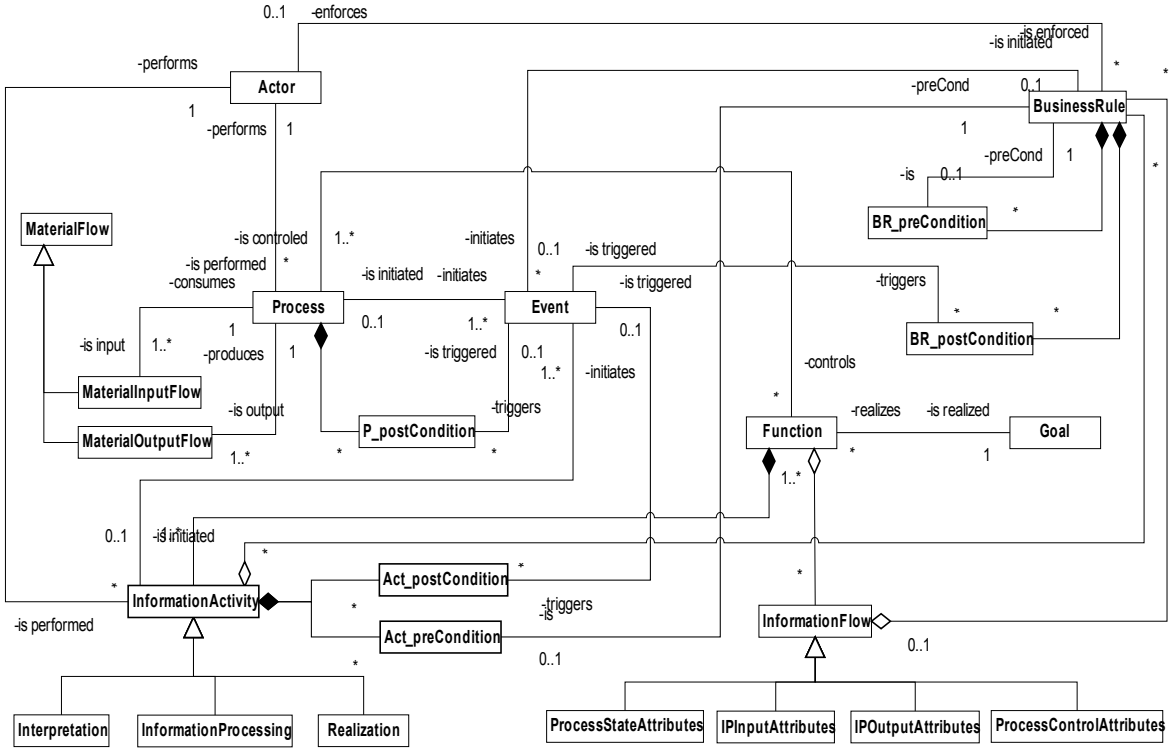
Pav. 9 Veiklos modelio sudėtis (UML notacija)[9]

Veiklos modelį (*EnterpriseModel*) sudaro veiklos modelio elementų (*ModelElement*) ir ryšių (*R*) aibės.

Modelio elementu (*ModelElement*) gali būti aktorius (*Actor*), materialus srautas (*MaterialFlow*), procesas (*Process*), įvykis (*Event*), funkcija (*Function*), informacinis srautas (*InformationFlow*), informacinė veikla (*InformationActivity*), veiklos taisyklė (*BusinessRule*) arba tikslas (Goal):

<ModelElement> ::= <Actor> | <MaterialFlow> | <Process> | <Event> | <Function> | <InformationFlow> | <InformationActivity> | <BusinessRule> | <Goal>.

Kiekvienas modelio elementas, be sisteminių atributų, gali turėti aibę to elemento savybes specifikuojančių specifinių atributų. Modelio elementas taip pat gali būti sudarytas iš kitų to paties tipo elementų. Veiklos modelio elementai ir jų tarpusavio sąsajos pateikiami 7 paveiksle.



Pav. 10 Veiklos metamodelis (UML notacija)[9]

- ***Procesas*** (*Process*) yra organizacijoje vykstančių darbų sekų (angl., *work flow*) elementas, materialų įeigos srautą (*MaterialInputFlow*) transformuojantis į materialų išeigos srautą (*MaterialOutputFlow*), tam kad įgyvendintų organizacijos keliamą tikslą. Procesas yra atliekamas aktoriaus (*Actor*). Proceso vykdymą inicijuoja vienas ar daugiau įvykių (*Event*). Įvykęs procesas sužadina vieną ar daugiau įvykių.
- ***Funkcija*** (*Function*) yra darbų sekų elementas, skirtas vieno ir daugiau procesų (*Process*) valdymui ir kontrolei. Kiekvieną procesą, transformuojantį materialų įeigos srautą į materialų išeigos srautą, valdo bent viena funkcija. Funkciją sudaro ***informacinių veiklų*** (*InformationActivity*) ir ***informacinių srautų*** (*InformationFlow*) aibės; informacinių veiklų įeigos ir išeigos yra informaciniai srautai:

<Function> ::= {(<ProcessStateAttributes>, <Interpretation>, <IPInputAttributes>)}, {(<IPInputAttributes>, <InformationProcessing>, <IPOutputAttributes>)}, {(<IPOutputAttributes>, <Realization>, <ProcessControlAttributes>)}.

Funkcijos informacinių veiklų aibė yra vadinama funkcijos *operacine dalimi* (*OperativePart*). Funkcijos informacinių srautų aibė yra vadinama funkcijos *informacine dalimi* (*InformationPart*).

- ***Informacinis srautas*** (*InformationFlow*) formaliai yra apibrėžiama kaip aibė atributų, būtinų funkcijos informacinės veiklos vykdymui:

<InformationFlow> ::= <ProcessStateAttributes> | <IPInputAttributes> | <IPOutputAttributes> | <ProcessControlAttributes>.

- ***Informacinė veikla*** (*InformationActivity*) yra sudėtinė funkcijos dalis. Pagal formalią funkcijos sudėtį, funkcijoje turi būti bent po vieną kiekvieno iš trijų tipų (interpretavimas (*Interpretation*), duomenų apdorojimas/sprendimų priėmimas (*InformationProcessing*), realizavimas (*Realization*)) informacinę veiklą:

<InformationActivity> ::= <Interpretation> | <InformationProcessing> | <Realization>.

- ***Interpretavimas*** (*Interpretation - IN*) yra funkcijos informacinė veikla, skirta transformuoti proceso būsenos atributams (*ProcessStateAttributes*) ir juos pateikti duomenų apdorojimo/sprendimų priėmimo tipo informacinėms veikloms (*IP*). *IN* yra viena iš būtinų grįžtamojo ryšio kontūro dalių.

Interpretavimo įeiga yra aibė valdomojo objekto (proceso) būsenos atributų (*ProcessStateAttributes*), kurie yra reikalingi konkrečiai valdymo funkcijai vykdyti ir gali būti interpretuojami kaip tos funkcijos įeiga. *IN* išeiga yra interpretuotų atributų informacinis srautas {*IPInputAttributes*}, reikalingas sprendimų priėmimo informacinėms veikloms (*IP*). *IN* yra interpretavimo taisyklių rinkinys. *IN* veikla taip pat gali turėti rinkinį prieš-sąlygų (*Act_preCondition*), kurios turi būti patenkintos, siekiant pradėti tos veiklos vykdymą. Veiklos po-są-

lygos (*Act_postCondition*) skirtos įvykiams (*Event*) sužadinti. Sužadintas įvykis gali inicijuoti sekančios informacinės veiklos funkcijoje vykdymą.

- ***Sprendimų priėmimas*** (*InformationProcessing - IP*) yra funkcijos informacinė veikla, skirta transformuoti interpretuotus proceso būsenos atributus į valdančiuosius poveikius (sprendimus). IP yra viena iš būtinų grįžtamojo ryšio kontūro dalių. *IP* paprastai sudaro duomenų apdorojimo ir sprendimų priėmimo veiklos, tačiau šiame darbe šios dvi veiklos bus apjungiamos į vieną ir vadinamos sprendimų priėmimu..

Sprendimų priėmimo veiklos įeiga (*IPInputAttributes*) yra interpretuotų atributų apie valdomą procesą informacinis srautas, reikalingas atitinkamiems sprendimams priimti. *IP* išeiga yra informacinis srautas *IPOutputAttributes*. *IP* sudaro sprendimų priėmimo taisyklių rinkinys – šių taisyklių pagrindu, įvertinus įeigos parametrus, yra priimami sprendimai. *IP* informacinė veikla gali turėti prieš- ir po-sąlygas.

- ***Realizavimas*** (*Realization - RE*) yra funkcijos informacinė veikla, skirta transformuoti IP suformuotą sprendimų informacinį srautą į valdančiuosius poveikius, skirtus tiesiogiai įtakoti (valdyti) valdomą procesą. *RE* yra viena iš būtinų grįžtamojo ryšio kontūro dalių.

RE įeiga yra *IP* suformuotas informacinis srautas *IPOutputAttributes*. *ProcessControlAttributes* yra *RE* išeigos srautas, kurį sudaro valdančiųjų poveikių rinkinys, skirtas konkretaus proceso (procesų) valdymui. *RE* gali turėti prieš- ir po-sąlygas.

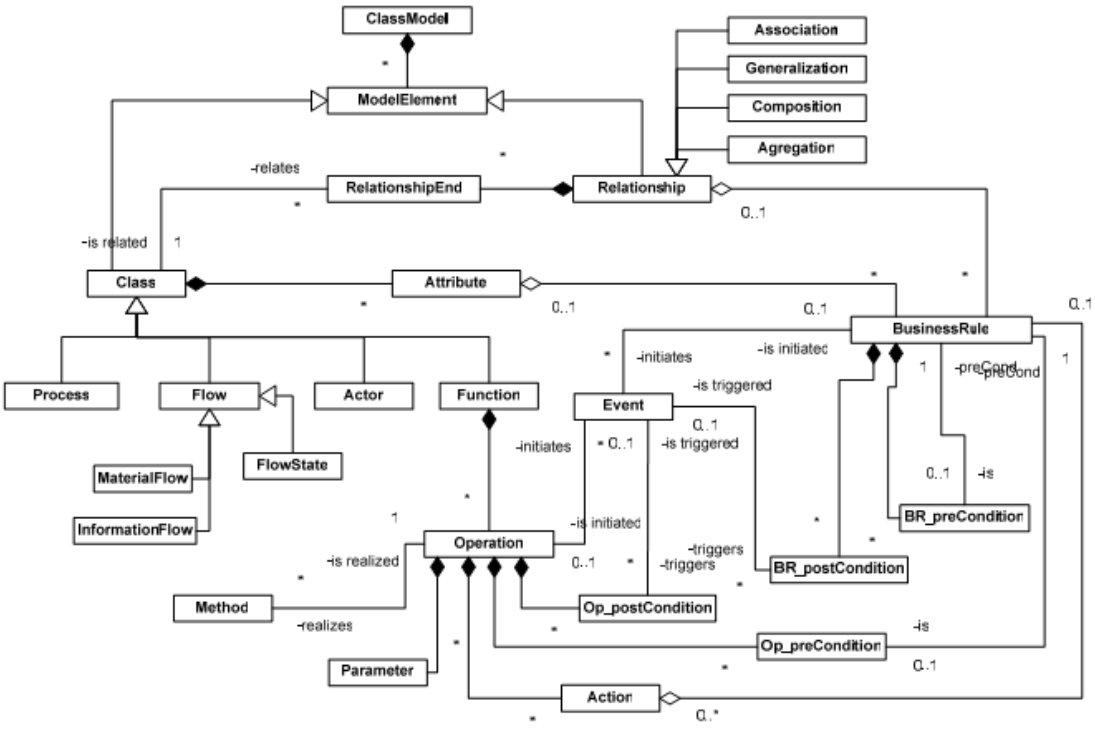
- ***Įvykis*** (*Event*). Įvykiai identifikuoja sistemos (organizacijos) būsenos pokyčius. Pagal pasiūlytąjį veiklos metamodelį, įvykiai inicijuoja procesų ir funkcijų informacinių veiklų arba atskirų veiklos taisyklių vykdymą.
- ***Veiklos taisyklė*** (*BusinessRule - VT*). Veiklos taisyklė, kaip veiklos modelio elementas, gali būti sąlyga, kuri turi būti patenkinta, kad galėtų įvykti kita taisyklė ar informacinė veikla, apribojimas, matematinė išraiška, loginė išvestis arba veiklos objektų tarpusavio sąveikas specifikuojanti išraiška (struktūrinė taisyklė). Veiklos taisyklės, kaip organizacijos veiklos logikos fragmentai, yra sudėtinės funkcijų informacinių veiklų dalys – tai sudaro organizacijos valdymo funkcijų branduolį.
- ***Aktorius*** (*Actor*) apibrėžia organizacinės struktūros elementą, kuris turi savo pareigas organizacijos vykdomoje veikloje. Procesus ir funkcijas (jų informacines veiklas) atliekantis aktorius gali būti ne tik žmogus ar aukštesnio hierarchijos lygio organizacinis elementas, bet ir programinė įranga, įrengimas ir kt. Aktorius taip pat gali būti atsakingas už vienos ir daugiau veiklos taisyklių vykdymą, kontrolę.
- ***Tikslas*** (*Goal*). Tikslas yra organizacijos suformuotas teiginys apie galimas (pasiekiamas, pageidautinas) situacijas organizacijoje esamuoju ir būsimuoju laiku. Organiza-

cijos tikslai įtakoja organizacijos veiklos procesus. Organizacijos tikslai yra realizuojami per organizacijos valdymo funkcijas.

2.9.Klasių metamodelis

[9]Į objektus orientuotose metoduose klasių modeliai paprastai naudojami kaip: dalykinės srities modeliai atvaizduojantys dalykinės srities konceptus; kaip konceptualūs analizės modeliai reikalavimų analizei; kaip sistemų kūrimo modeliai detaliam programinės įrangos kūrimui. Klasių modelis taip pat yra dalis OMG (*Object Management Group*) standarto, būtent UML. Klasių modelis UML grindžiamose CASE įrankiuose tarnauja kaip pagrindinis žinių šaltinis kuriant IS prototipus: duomenų bazių specifikaciją, grafinę vartotojo sąsają, programinį kodą. Tačiau žiūrint iš IS inžinerijos pusės remiantis [10] UML metamodelis yra per daug komplikuotas ir turintis daug nereikalingų elementų todėl UML naudojimas siauroje dalykinėje srityje tampa neišraiškus.

Remiantis [9] pateikta autorių nuomone, UML neturi elementų reikalingų veiklai modeliuoti (pvz. veiklos taisyklių), taip pat negali pasiūlyti veiklos logikos (gryžtamo ryšio, veiklos taisyklių, veiklos objektų tipų).[9] siūlomas klasių metamodelis pateikiamas 11 pav.. Klasių metamodelis grindžiamas UML metamodeliu kartu įtraukiant ir elementus iš veiklos metamodelio (VMM) 10 pav.



Pav. 11 Klasių metamodelis[9]

Klasių modelis yra sudarytas iš klasių (*Class*) ir ryšių (*Relationship*) kurie sujungia klases tarpusavyje. Kiekvienas ryšys turi mažiausiai du ryšio galus (Relationship ends) ir taip pat gali būti apribotas veiklos taisyklėmis (BusinessRule).

Klasių modelis yra papildytas naujais elementais: proceso (Process), aktoriaus (Actor), funkcijos (Function), ir srauto (Flow). Toks papildymas pagrįstas veiklos metamodeliu pav. 10.

Srauto tipo klasės gali turėti būsenas (FlowState).

Paprastai klasės turi atributus (*Atribute*) ir operacijas(*Operation*).Pasiūlytame klasių metamodelyje visų tipų klasės gali turėti atributus, tačiau operacijas gali turėti tik funkcijos tipo klasės. Operacijos apibrėžia algoritmiškai sudėtingus veiksmus. Algoritmiškai paprasti veiksmai neaprašomi siekiant neapkrauti modelio nereikalingais elementais. Klasių atributai taip pat gali turėti eilę veiklos taisyklėmis užrašytų apribojimų.

Operacijos yra sudarytos iš veiksmų(*Action*) ir gali turėti parametrus ir metodus kurie realizuoja operacijas pasirinktoje programavimo platformoje.

Veiksmai pristato veiklos taisykles kurių tipai yra: skaičiavimo(*Computation*), veiksmo(Action), išvados (Inference).

Veiklos taisyklės ir operacijos gali būti inicijuojamos įvykių(Events), taip pat jos gali sužadinti įvykius pačios.

Kaip Veiklos modelis atsivaizduoja išplėstame klasių modelyje pateikiama lentelėje Nr.1

Lentelė 1 Veiklos ir klasių modelio ryšys

Veiklos metamodelio elementas	**Klasių metamodelio elementas**
ModelElement	ModelElement
Function	Class, Function
Process	Class, Process
MaterialFlow	Class, Flow, FlowState
InformationFlow	Class, Flow, FlowState
Actor	Class, Actor
Event	Event
InformationActivity	Operation
Atribute	Atribute
Realationships	Realationship, RealationshipEnd
BusinessRule	Atribute, BusinessRule
BusinessRule	Operation, BusinessRule
BusinessRule	Realationship, BusinessRule

Veiklos modelio elementai į klasių modelio elementus atvaizduojami taip: (pvz. veiklos modelio elementas „Function“ atsivaizduoja į klasių modelio elementą „Class“ kurio tipas „Function“. Analogiškai atsivaizduoja ir kiti elementai.

2.10.Veiklos objektų modeliavimo analizės rezultatai

Atlikus dalykinės srities analizę galima nustatyti kuriamos DSK konceptus, ryšius tarp konceptų bei apribojimus. Remiantis 1 lentele klasių modelį reikia išplėsti elementais kurių klasių modelyje nėra. Iš 1 lentelės dešinio stulpelio atmetame elementus kurie yra UML klasių modelyje(Class, Event, Operation, Atribute, Relationship, RelationshipEnd) ir lieka elementai kuriais reikia išplėsti klasių modelį. Taigi elementai reikalingi išplėsti klasių modelį norint modeliuoti veiklas pateikiami lentelėje Nr. 2.

<ModelElement> ::= <Actor> | <Flow> | <Process> | <Function> | <BusinessRule> | <FlowState>.

Lentelė 2 Klasių modelį išpečiantys stereotipai

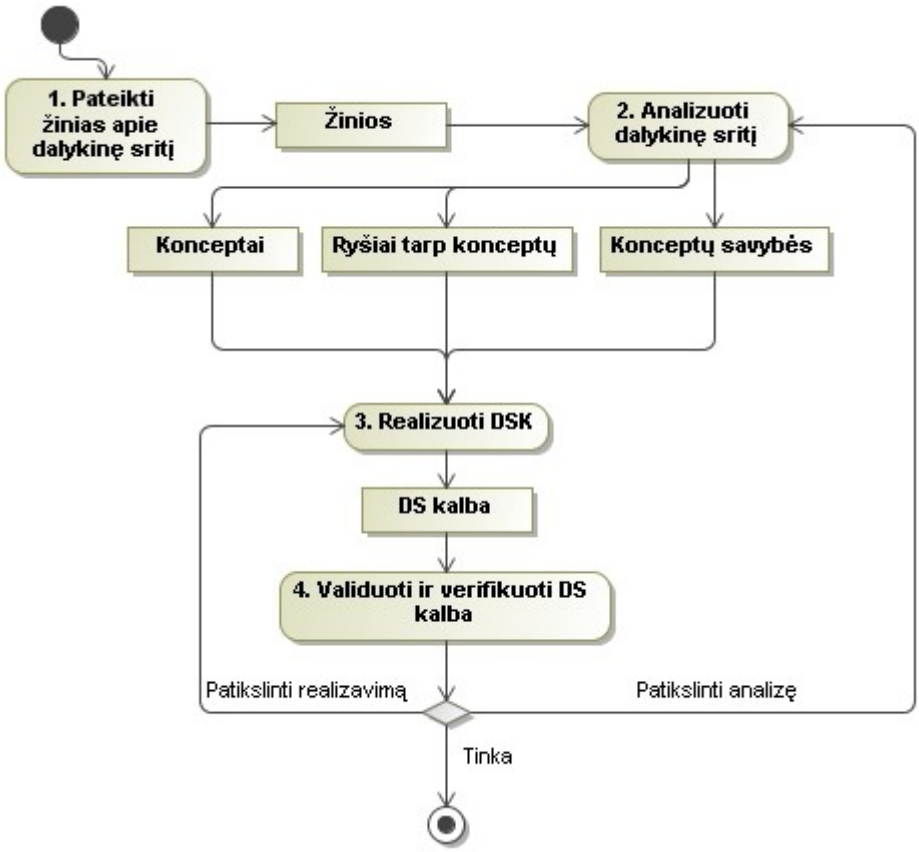
Išplečiamas elementas	**Stereotipas**	**Išplečiama metaklasė**	**Apribojimai**
<Actor>	<<Actor>>	<<Class>>	Aktoriaus tipo klasė neturi operacijų
<Flow>	<<Flow>>	<<Class>>	Srauto tipo klasė neturi operacijų
<Process>	<<Process>>	<<Class>>	Proceso tipo klasė neturi operacijų
<Function>	<<Function>>	<<Class>>	-
<FlowState>	<<FlowState>>	<<Class>>	Srauto būsenos tipo klasė neturi operacijų
<BusinessRule>	<<BusinessRule>>	<<Constraint>>	-

Naudojantis 2.7.1 skyriumi išskirti dalykinės srities modelio validavimui reikalingi apribojimai:

1. Procesas yra atliekamas aktoriaus.
2. Kiekvieną procesą, transformuojantį materialų įeigos srautą į materialų išeigos srautą, valdo bent viena funkcija.
3. Funkciją sudaro bent viena informacinė veikla.

3.SIŪLOMA DALYKINĖS SRITIES KALBŲ KŪRIMO METODIKA

Dalykinės srities kalbų kūrimas susideda iš trijų pagrindinių etapų: dalykinės srities analizės, dalykinės srities kalbos įgyvendinimas pasirinktoje platformoje, bei DS kalbos verifikavimas ir validavimas . DS analizės etape gaunamas DS modelis turintis savo konceptus, ryšius tarp konceptų, bei konceptų ypatybes. Įgyvendinimo etape iš dalykinės srities modelio gaunama kalba kurios žodynas, vaizdavimas, elgsena artima realiai dalykiniai sričiai. Validavimo ir verifikavimo etape nustatoma ar dalykinės srities kalba atitinka reikalavimus. Dalykinės srities kalbos kūrimo koncepcija pavaizduota (pav. 12).



Pav. 12 DS kalbos kūrimo veiklos diagrama.

Žinių apie dalykinę sritį išgavimas ir analizė gali būti atliekama tiek aukščiau aprašytais formaliais metodais tokiais kaip FODA, DARE, tiek neformaliais. Metodų pasirinkimas priklauso nuo pačios dalykinės srities, jeigu dalykinė sritis siaura, gerai dokumentuota tai nebūtina naudoti formalius metodus norint gauti daugiau, mažiau pilną dalykinės srities modelį. Siūlomos metodikos aprašas pateikiamas žemiau.

1. **Pateikti žinias apie dalykinę sritį**

Žinias apie dalykinę sritį pateikia dalykinės srities ekspertai, darbuotojai, įmonių savininkai. Žinios gali būti pateikiamos dokumentacijos forma, gal būti organizuojama ekspertų apklausa, stebimi darbuotojai savo darbo vietose ir dokumentuojamas pastarųjų elgesys. Kuo

detalesnes ir tikslesnės žinios apie dalykinę sritį pateikiamos, tuo tikslesnis dalykinės srities modelis gaunamas.

2. **Analizuoti dalykinę sritį**

Dalykinės srities analize užsiima informacinės sistemos ekspertai. IS ekspertai iš pateiktų dalykinės srities žinių sudaro dalykinės srities modelį kuris susideda iš: (konceptų su savo savybėmis bei ryšių tarp konceptų). DS analizė gali būti atliekama formaliais metodais tokiais kaip jau mitėti (FODA, DARE), tiek neformaliais pvz. analizuojant dalykinės srities metamodelį.

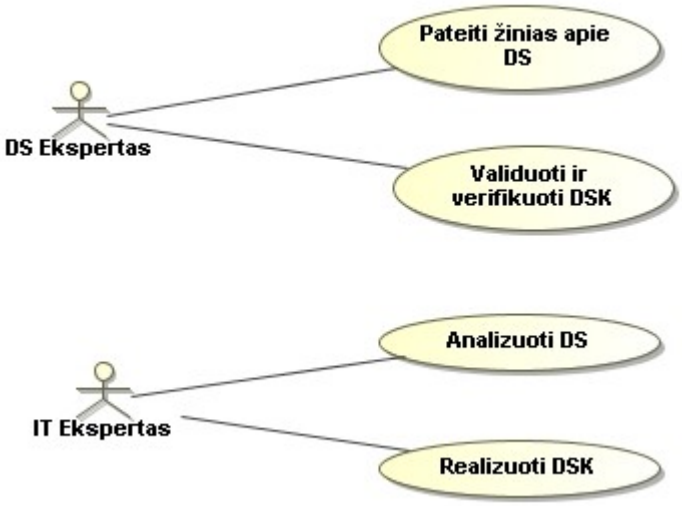
3. Realizuoti DSK

Dalykinės srities kalbos realizavimas UML MagicDraw aplinkoje pateikiamas 4.2 skyriuje.

4. **Validuoti ir verifikuoti DS kalbą.**

Dalykinės srities kalbos validavimas ir verifikavimas atliekamas išbandant sukurtąją kalbą. Naują kalbą turi patikrinti DS ekspertai ir DS darbuotojai kurie kurs programas su šia kalba. Apie pastebėtas klaidas (konceptų, jų savybių, sąryšių neatitikimas dalykinei sričiai, naudojimo patogumas, ...) pranešama dalykinės srities kalbos kūrėjams, kurie nusprendžia kaip bus ištaisomos šios klaidos. Gal nutikti taip, kad klaidos atsiradimo priežastis yra netinkamai išanalizuota dalykinė srities, bei sudarytas nekorektiškas DS modelis, tuomet reikia atlikti pakartotinę analizę atsižvelgiant į kalbos tikrinimo metu gautus netikslumus. Jeigu nustatoma, kad netiksliai realizuota pati kalba, tai reikia grįžti ir patikslinti kalbos realizavimą.

Metodikos panaudos atvejų diagramoje (pav. 13) matome kokius vaidmenis DSK kūrimo procese atlieka DS ekspertas ir IT ekspertas.



Pav. 13 DSK kūrimo metodikos panaudos atvejai

3.1.DSK realizavimas UML MagicDraw pakete

Atlikus dalykinės srities analizę ir turint DS modelį reikia nuspręsti kokios UML diagramos pagrindu bus kuriama nauja dalykinės srities kalba. UML diagramos pasirinkimas priklauso nuo to kokiu aspektu norima matyti sistema. Jeigu norima matyti sistemos statika tai už pagrindą reikia pasirinkti vieną iš statiką vaizduojančių UML diagramą. Jeigu domina kaip sistema kinta – tai reikia pasirinkti dinamiką vaizduojančią UML diagramą. UML diagramų tipai:

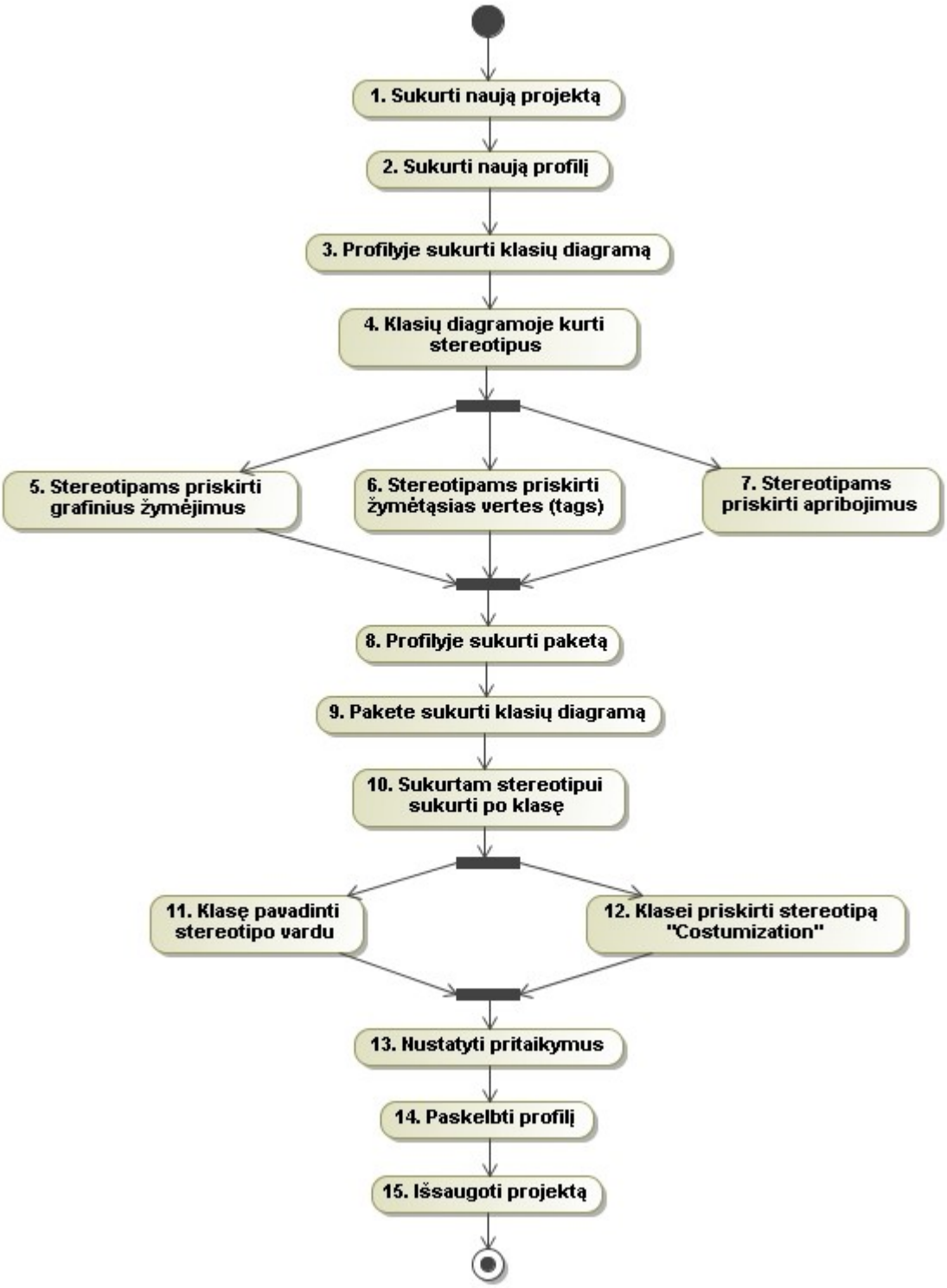
- Statinės (klasių (*angl*. *class*), komponentų (*angl. components*), objektų (*angl. objects*), ...)
- Dinaminės(veiklos (*angl. activity*), panaudos atvejų (*angl. use case*), sekų (*angl. sequence*), ...)

Pasirinkus diagramą, keičiami pasirinktos diagramos elementai – elementams sukuriami stereotipai, priskiriamos žymės, aprašomi apribojimai, paslepiamos nereikalingos UML notacijos detalės.

UML MagicDraw pakete kiekvienos diagramos metamodelis saugomas profilyje(angl. profile). Standartiškai UML MagicDraw turi UML standart profilį, BPMN, DDL, EDOC, DoDAF ir tt. profilius. Kiekvienas profilis susideda iš rinkinio stereotipų (angl. stereotype). Profiliai yra apibrėžiami kaip atskiras modulis. Profiliai užkraunami paketo paleidimo metu. Vartotojai gali kurti naujus profilius. Vartotojui sukūrus naują profilį reikia paleisti UML MagicDraw paketą iš naujo, kad būtų užkrautas profilis su naujomis savybėmis.

3.1.1.Profilio kūrimas

Naujo profilio kūrimas susidaro iš veiksmų sekos pavaizduotos veiklos diagrama (14 pav.). Norint metodiškai kurti profilį reikėtų laikytis šios veiksmų sekos.



Pav. 14 Profilio sukūrimo veiklos diagrama

Detalesnis profilio sukūrimo mechanizmas išaiškintas žemiau.

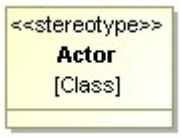
1 žingsnis. Sukurti naują projektą. Naujas projektas sukuriamas meniu juostoje pasirinkus File->New Project, ir atsivėrusiame lange įvedant projekto pavadinimą.bei nurodant vietą kompiuteryje.

2 žingsnis. Sukurti naują profilį. Naujas profilis kuriamas navigacijos meniu pasirinkus objektą „Data“ ir paspaudus dešnįjį pelės klavišą pasirenkama New Element -> Profile.

3 žingsnis. Profilyje sukurti klasių diagramą. Sukūrus profilį, jame analogiškai kuriama klasių diagrama (New Diagram->Class Diagram) kur bus realizuojamas DSK metamodelis panaudojant stereotipus.

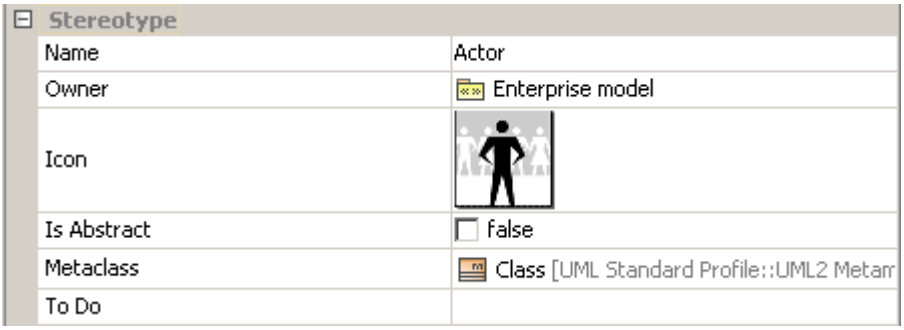
4 žingsnis. Klasių diagramoje kurti stereotipus. Stereotipai kuriami diagramos menių pasirenkant (Profiling Mechanism->Stereotype). Nurodžius elemento vietą diagramoje, automatiškai atsiveria langas reikalaujantis pasirinkti kokią UML metaklasę paveldės naujas stereotipas. Metaklasės pasirinkimas priklauso nuo to, kokį UML elementą norima praplėsti nauju stereotipu. Mūsų atveju stereotipas aktorius (*Actor*) praplės klasės (*Class*) tipo UML elementą (15 pav.).

Pastaba: *Renkantis kurį UML elementą praplėsti, reikia atsižvelgti į tai kokiu požiūriu (statiniu ar dinaminiu) bus modeliuojama sukurtaja DSK. Svarbu prisiminti, kad stereotipais išplėsti statinių diagramų elementai nebus matomi dinamiką vaizduojančiose diagramose ir atvirkščiai.*



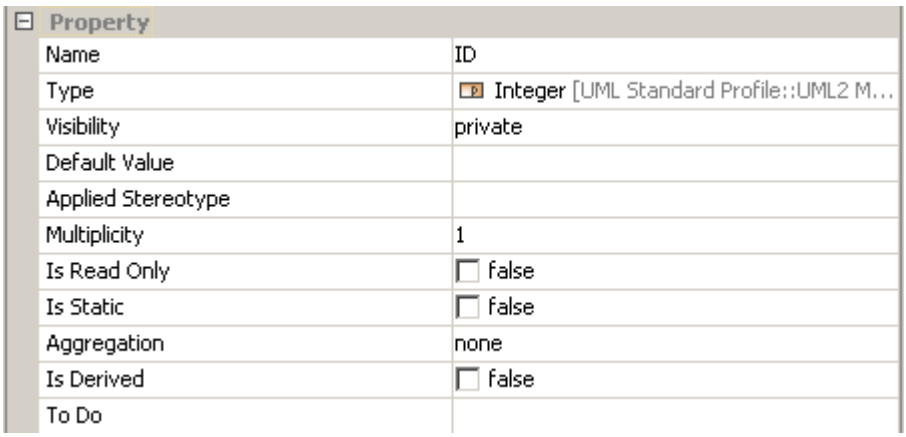
Pav. 15 Aktoriaus stereotipas

5 žingsnis. Stereotipams priskirti grafinius žymėjimus. Stereotipams grafiniai žymėjimai priskiriami stereotipo specifikacijos meniu pasirinkus punktą (*Icon*) (16 pav.). Stereotipo specifikacijos langas atveriamas-ant stereotipo pele spustelėjus du kartus arba ant stereotipo paspaudus dešinį pelės klavišą ir pasirinkus punktą (*Specification*).



Pav. 16 Aktoriaus (Actor) stereotipas su grafiniu žymėjimu

6 žingsnis. Stereotipams priskirti žymes (*tags*). Stereotipams žymėtosios vertės priskiriamos taip pat stereotipo specifikacijos meniu pasirinkus punktą (*Tag Definitions*) ir paspaudus (*Create*). Atsivėrusiame lange (17 pav.) specifikuojama žymėtoji vertė.



Pav. 17 Žymėtosios vertės specifikacija

Reikia pastebėti, kad žymės tipas gali būti ir kitas UML elementas arba sukurtas naujas stereotipas.

7 žingsnis. Stereotipams priskirti apribojimus. Stereotipams apribojimai prisikiriami naudojantis apribojimus aprašančiu UML elementu (*Constraint*). Stereotipui naujas apribojimas sukuriamas analogiškai kaip ir žymė, tik stereotipo specifikacijoje pasirinkus punktą (*Constraints*).

8 žingsnis. Profilyje sukurti paketą. Paketas kuriamas analogiškai kaip profilis (*žiūr. 2 punktas*), tik iškrentančiame meniu pasirenkama (*Package*). Pakete bus kuriama klasių diagrama skirta stereotipų pritaikymams modeliuoti.

9 žingsnis. Pakete sukurti klasių diagramą. Klasių diagrama kuriama analogiškai kaip klasių diagrama skirta stereotipams modeliuoti (*žiūr. 3 punktas*). Sukurtoje klasių diagramoje bus atliekami stereotipų pritaikymai.

10 žingsnis. Sukurtam stereotipui sukurti po klasę. Kiekvienam sukurtam stereotipui pritaikymams skirtoje diagramoje sukuriama po klasę. Klasės kuriamos diagramos meniu pasirinkus (*Class*) ir pele nurodant vietą diagramoje.

11 žingsnis. Klasę pavadinti stereotipo vardu. Stereotipų pritaikymams modeliuoti skirtoms klasėms turi būti suteikiami analogiški vardai kaip ir patiems stereotipams.

12 žingsnis. Klasei priskirti stereotipą <<Customization>>. Pritaikymams modeliuoti skirtoms klasėms būtina priskirti stereotipą <<Customization>>. Stereotipas priskiriamas ant klasės paspaudus dešiniu pelės mygtuku ir iškrentančiame meniu pasirenkama (*Stereotype->Customization*).

13 žingsnis. Nustatyti pritaikymus. Tam, kad sukurtas naujas stereotipas skirtųsi nuo standartinių UML elementų reikia stereotipams nustatyti pritaikymus. Nustačius pritaikymus galima paslėpti nereikalingus UML notacijos elementus. Pritaikymai nustatomi klasės (*su priskirtu <<customization>> stereotipu*) specifikacijos meniu pasirinkus punktą (*Tags*). Pritaikymų reikšmės pateikiamos 3 lentelėje.

Lentelė 3 Pritaikymų parametrai

Pritaikymų grupė	Pritaikymas	Reikšmė
Connection rules	Allowed realationships	Ryšiai kuriuos galima jungti su šiuo elementu. Tik šie ryšiai bus leidžiami jei specifikuota.
	Disallowed realationships	Ryšiai kurių neleidžiama jungti su šiuo elementu.
	TypesForSource	Metaklasės ar stereotipai kurie galimi jungti kaip šaltinis šiam ryšio tipui. Nustatymas neturi prieštarauti UML specifikacijai. Jeigu ryšys neturi krypties tai pirmas prijungtas elementas laikomas šaltiniu, o antras taikiniu (*angl. target*)
	TypesForTarget	Metaklasė ar stereotipas kurie galimi jungti kaip taikinys su šiuo ryšio tipu.
General	CustomizationTarger	Stereotipas kurį ketinama pritaikyti
	HideMetatype	Jeigu reikšmė „true“ tai reiškia, kad stereotipas elgiasi kaip naujas standartinis elementas MagicDraw pakete
	QuickApplyingFor	Kai kurie elementai elgiasi labai panašiai kaip standartiniai UML elementai, todėl norisi, kad stereotipą šiam elementui praplėsti butų galima pasirinkti iš greitojo meniu, šis nustatymas skirtas įtraukti naują stereotipo parinktį į jau esančio elemento meniu.
Model Initialization	ApplyToSource	Stereotipas kuris bus priskiriamas šio ryšio šaltinio elementui po to sujungimo
	ApplyToTarget	Stereotipas kuris bus priskiriamas šio ryšio taikinio elementui po to sujungimo
	SuperTypes	Tipai kurie turi būti supertipais šiam elementui. Paveldėjimas bus sukuriamas modelyję po šito stereotipo panaudojimo.

OwnedElements	SuggestedOwnedDiagrams	Diagramų tipai kuriose bus leidžiama kurti šio stereotipo elementus, kitos diagramos bus paslėptos
	HiddenOwnedDiagrams	Diagramų tipai kurie bus paslėpti
	SuggestedOwnedTypes	Sąrašas stereotipų kurie gali būti naudojami kaip vidiniai šio stereotipo elementai.
	HiddenOwnedTypes	Sąrašas stereotipų kurie negali būti naudojami kaip vidiniai šio stereotipo elementai.
	PossibleOwners	UML elementai kurie gali naudoti šio stereotipo elementą kaip savo vidinį elementą
Properties	InShortcutMeniu	Stereotipo nustatymai kurie bus matomi greitajame meniu
	multiLineTextProperties	Stereotipo nustatymai kurie bus vaizduojami keliose eilutėse
	ShowPropertiesWhenNotApplied	Jeigu „true“ tai bus vaizduojami ir tušti nustatymų laukai
	StandartExpertConfiguration	Nustatymai skirti paslėpti arba matyti UML savybėms
	UsedUmlProperties	UML savybės kurios bus naudojamos

14 žingsnis. Paskelbti profilį. Tam, kad būtų galima sukurtus pakeitimus naudoti kituose projektuose profilį reikia paskelbti (*angl. share*). Profilio paskelbimas atliekamas programos meniu pasirinkus (*File->Share Packages*).

15 žingsnis. Išsaugoti projektą. Projektas išsaugomas programos meniu pasirinkus (File->Save Project).

Atlikus visus šiuos žingsnius reikia paleisti UML MagicDraw paketą iš naujo, tam, kad sukurti elementai elgtųsi taip kaip nustatyta.

4.VEIKLOS OBJEKTŲ (VO) MODELIAVIMO KALBOS KŪRIMAS IR KALBOS PANAUDOJIMO PAVYZDYS

4.1.VO modeliavimo kalbos kūrimo tikslas

Pagrindinis šios kalbos kūrimo tikslas - pavaizduoti sukurtos dalykinės kalbos kūrimo metodikos veiksmingumą. Parodyti, kad laikantis metodikos lengva sukurti įvairiausių sričių DSK.

Antras tikslas suteikti UML MagicDraw vartotojams galimybę modeliuoti veiklos objektus pasinaudojant sukurta dalykinės srities kalba.

Veiklos objektų modeliavimo kalbos sukūrimas reikalingas modeliai grindžiamos IS inžinerijai. Veiklos objektų modelis turėtų būti formali dalykinės srities žinių evoliucijos struktūra naudojama visuose IS inžinerijos gyvavimo ciklo etapuose.

Taip pat klasių modelio išplėtimas veiklos modeliavimui reikalingas Informacijos sistemų katedroje vykdomiems tyrimams, tokiems kaip klasių modelio generavimui iš žinių bazės ir kt.

4.2.VO modeliavimo kalbos kūrimo planas

1) Išanalizuoti dalykinę sritį (veiklos objektų modeliavimas).
2) Nustatyti DS konceptus, ryšius tarp konceptų, konceptų žymes, apribojimus.
3) Pritaikyti sudarytą DSK kūrimo metodiką.
4) Nustatyti modelio validavimo taisykles.
5) Pamėginti modeliuoti veiklos objektus.

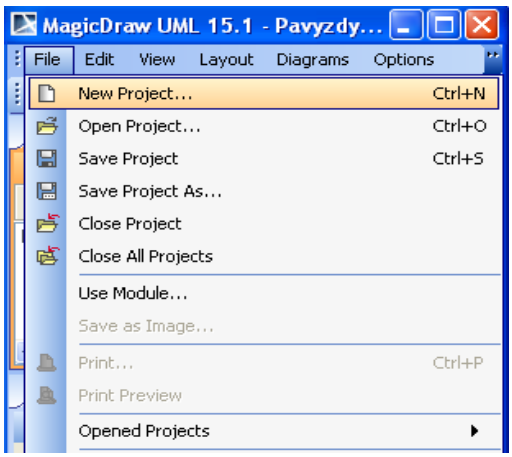
4.3.VO modeliavimo kalbos kūrimo eiga

Išanalizavus pasiūlytą veiklos modelį (2.7.1 skyriuje) ir nustačius dalykinės srities konceptus, ryšius tarp konceptų, konceptų žymes, ir apribojimus (2.8 skyrius) bei turint dalykinės kalbos kūrimo metodiką skirtą UML MagicDraw paketui (3.1 skyrius) galime pradėti kurti organizacijos veiklos modeliavimo kalbą.

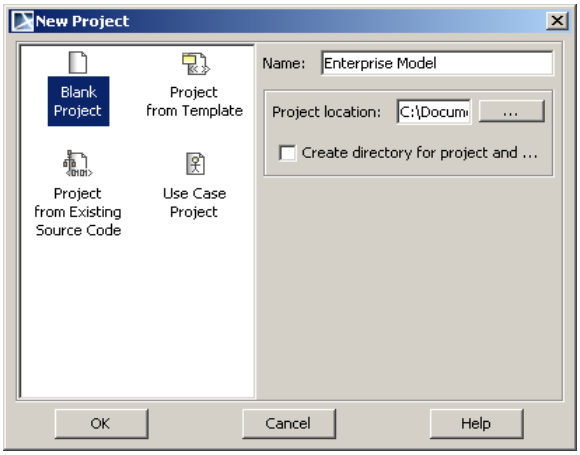
Organizacijos veiklos modeliavimo kalbą kursime naudodamiesi (3.1 skyriuje) sudaryta metodika. Metodikos panaudojimas bus aprašytas pažingsniui.

1) Sukurti naują projektą.

Naujas projektas sukuriamas menių juostoje pasirinkus (File->New Project) bei nurodžius projekto pavadinimą bei vietą kompiuteryje (pav. 18, 19).



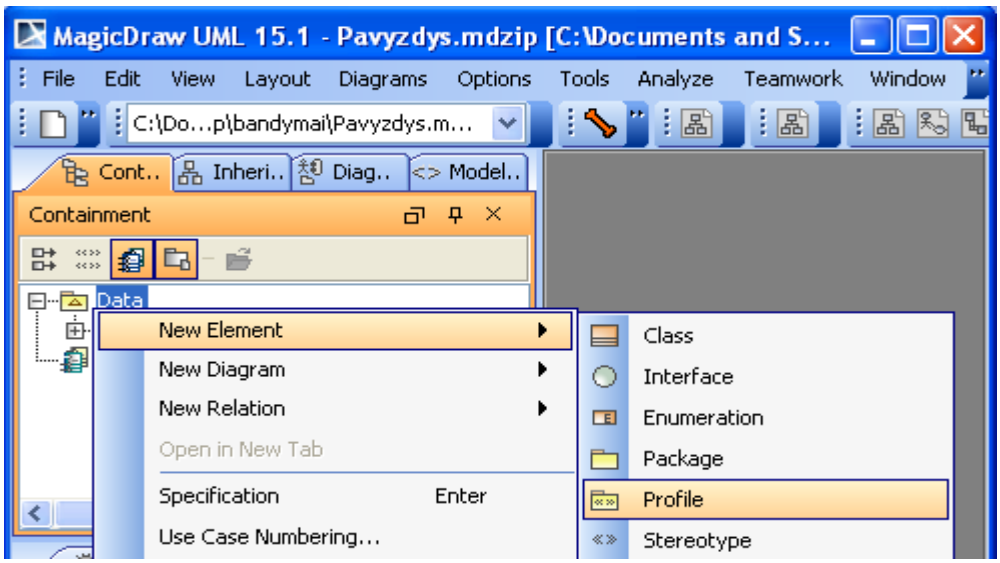
Pav. 18 Naujo projekto sukūrimas



Pav. 19 Projekto pavadinimas ir vieta kompiuteryje

2) Sukurti naują profilį

Naujas profilis kuriamas navigacijos meniu pasirinkus objektą „Data“ ir paspaudus dešinįjį pelės klavišą pasirenkama New Element -> Profile (pav.20)



Pav. 20 Naujo profilio kūrimas

3) Profilyje sukurti klasių diagramą.

Sukūrus profilį, jame analogiškai kuriama klasių diagrama (New Diagram->Class Diagram) kur bus realizuojamas DSK metamodelis panaudojant stereotipus ir jų tarpusavio sąryšius.

4) Klasių diagramoje kurti stereotipus.

Klasių diagramoje kuriame stereotipus atitinkančius dalykinės srities konceptus (lentelė Nr. 1). Stereotipus jungiame tarpusavyje, taip pavaizduodami konceptų tarpusavio priklausomybes.

Stereotipai kuriami klasių diagramos meniu pasirinkus (*Profiling Mechanism->Stereotype*) ir pele nurodžius vietą klasių diagramoje. Kuriant naują stereotipą automatiškai iškrenta sąrašas metaklasių kurias gali paveldėti kuriamas stereotipas. Konkretaus atvejo stereotipų paveldimos klasės pateiktos 4 lentelėje.

Lentelė 4 Stereotipų paveldimos metaklasės

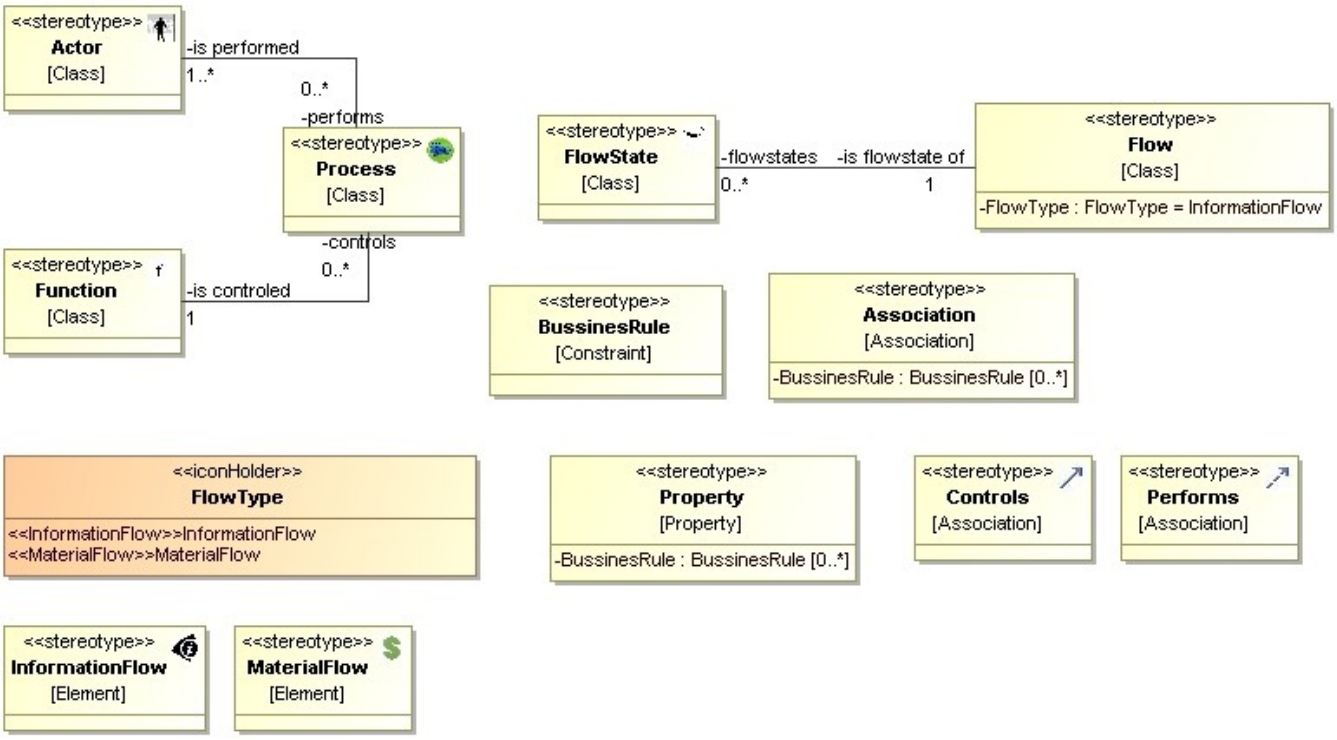
Stereotipas	Paveldima metaklasė	Reikšmė
Process	Class	Stereotipas naudojamas procesui vaizduoti, iš metaklasės Class naudoja atributus ir pavadinimą. Procesą gali valdyti funkcija ir atlikti aktorius.
Actor	Class	Stereotipas naudojamas aktoriui vaizduoti, iš metaklasės Class naudoja atributus ir pavadinimą. Aktorius gali atlikti procesus.
Function	Class	Stereotipas naudojamas funkcijai vaizduoti, iš metaklasės Class naudoja atributus, operacijas ir pavadinimą. Klasės operacijos naudojamos informacinėms veikloms identifikuoti.
Flow	Class	Stereotipas naudojamas srautui vaizduoti, iš metaklasės Class naudoja atributus ir pavadinimą. Srautai gali būti informaciniai arba materialūs
FlowState	Class	Stereotipas naudojamas srauto būsenai vaizduoti, iš metaklasės Class naudoja atributus ir pavadinimą.
BussinesRule	Constraint	Stereotipas naudojamas veiklos taisyklei. Paveldima Constraint metaklasė, nes veiklos taisyklė kaip ir Constraint įgauna reikšmę true arba false

Stereotipai naudojami ne vien konceptų realizavimui, bet ryšių tarp konceptų vaizdavimui. Tarkim reikia pavaizduoti, kad konceptas <<*Function*>> valdo konceptą <<*Process*>>. Tokiu atveju sukuriame stereotipą <<*Controls*>> kuris paveldi metaklasę [Dependency]. Sukurtas stereotipas pavaizduotas (21 pav.)

Pav. 21 Ryšio, control‘ stereotipas

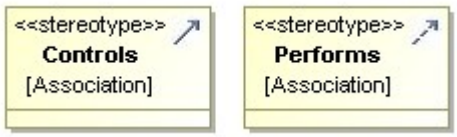
Ryšio stereotipui galima priskirti grafinius žymėjimus kaip ir bet kuriam kitam stereotipui, teisingai parinkus grafinius žymėjimus padidėja dalykinės kalbos vaizdingumas ir naudojimo intuityvumas.

Naudojantis išplėstu klasių metamodeliu buvo sudarytas dalykinės kalbos stereotipų modelis. Modelis pateiktas (pav.22)



Pav. 22 Klasių modelio išplėtimas stereotipais

Stereotipizuotiems ryšiams tarp konceptų vaizduoti buvo sukurti stereotipai paveldintys *[Association]* metaklasę. Stereotipizuoti ryšiai pateikti (pav.23).



Pav. 23 Stereotipizuoti ryšių pavadinimai.

5) Stereotipams priskirti grafiniu žymėjimus.

Visiems ryšio stereotipams buvo priskirti skirtingi grafiniai žymėjimai, kad ryšiai akivaizdžiai skirtųsi vienas nuo kito. Kaip priskirti grafinius simbolius aprašyta 3.1.1 skyriaus 5 punkte.

6) Stereotipams priskirti žymes

Šiuo atveju stereotipų žymėmis tapo asociacijų galai (*angl. Association ends*) kurie sieja vieną stereotipą su kitu. Kaip pavyzdys pateikiamas <<Flow>> stereotipo žymių sąrašas (24 pav.)

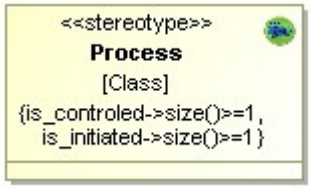
	Name	Type	Default Value	Classifier
⊟	General			
-	FlowType	FlowType [Bussine...	InformationFlow	«» Flow [Class...
⊟	Association End			
-	flowstates	FlowState [Class] ...		«» Flow [Class...

Pav. 24 <<Actor>> stereotipo žymių sąrašas

<<Flow>> stereotipui buvo priskirta žymė (FlowType) kuri identifikuoja ar srautas yra informacinis, ar materialus.

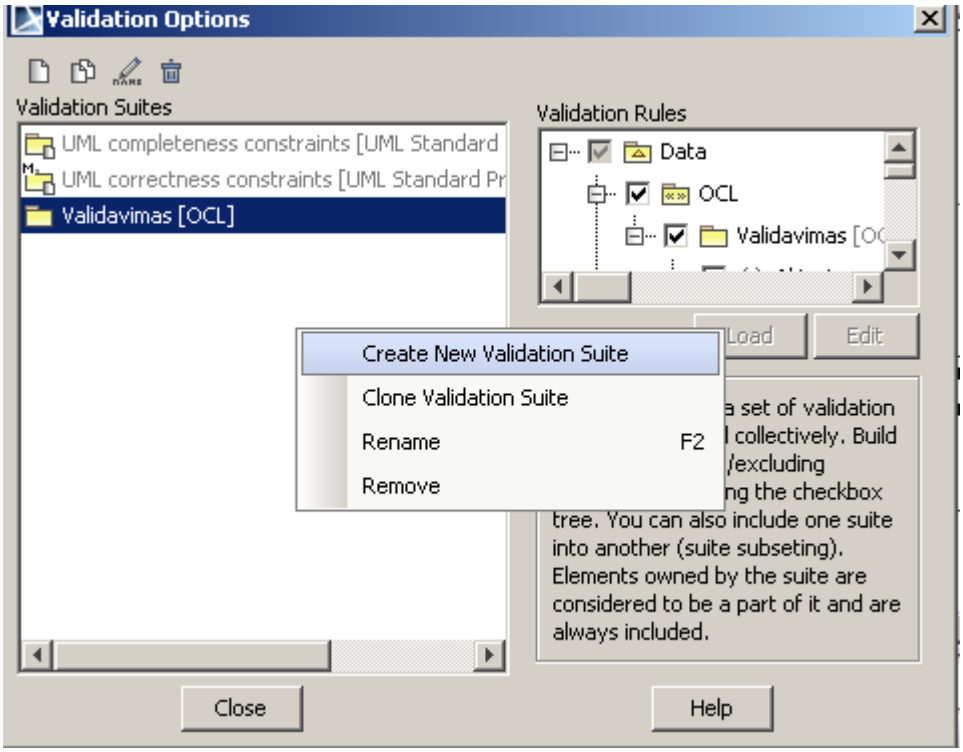
7) Stereotipams priskiriami apribojimai

Apribojimai priskiriami stereotipo specializacijos meniu pasirinkus punktą (*Constraint*). Apribojimai gali būti užrašomi šnekamąja kalba arba formalia OCL (Object Constraint Language) kalba. Šnekamąja kalba užrašytas apribojimas netenka savo prasmės, nes paketas neinterpretuoja šnekamosios kalbos ir automatiškai nevykdo apribojimų. OCL kalba užrašytus apribojimus labai patogu naudoti modelio validavimui. Proceso stereotipas su užrašytais apribojimais pateiktas (25 pav.)



Pav. 25 Stereotipas su apribojimais

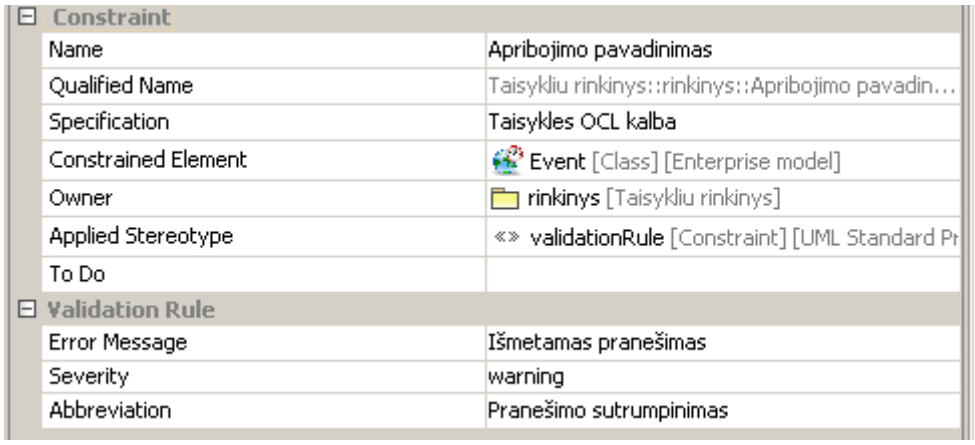
Modelio validavimui reikia susidaryti taisyklių rinkinį, taisyklių rinkinys gali būti iš jau egzistuojančių apribojimų arba galima sukurti naujus apribojimus. Taisyklių rinkinys sukuriamas UML magicDraw meniu pasirinkus (Analyze->Validation->Validation Options).Naujas taisyklių rinkinys kuriamas spragtelėjus dešiniu pelės mygtuku atsivėrusio lango kairė pusėje ir pasirinkus (*Create new Validation Suit*) (žiūr. 26 pav.)



Pav. 26 Modelio validavimo taisyklių rinkinio kūrimas

Toliau sukuriamas taisyklių rinkinio savininkas (*Create Owner*), pasirinkus ar savininkas bus profilis ar paketas įvedamas savininko pavadinimas ir atsivėrusiame lange įvedamas taisyklių rinkinio pavadinimas. Pasirenkame sukurtą taisyklių rinkinį ir spaudžiame (*Edit*). Atsivėrusiame lange pasirenkame

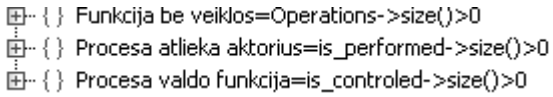
(*Inner Elements*) ir spaudžiame (*Create*) iš iškritusio sąrašo pasirenkame (*Constraint*). Sukurto apribojimo laukus užpildome kaip parodyta (27 pav.)



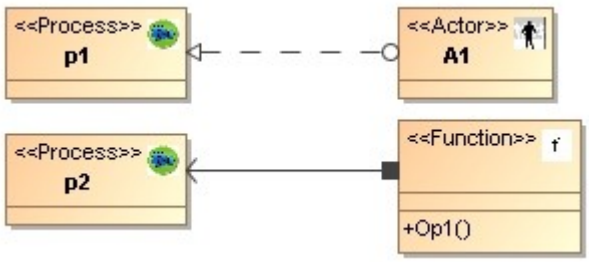
Pav. 27 Validavimo taisyklės aprašas

Sukūrus taisykles paleidžiame modelio validavimo procesą. Validavimo proceso paleidimas atliekamas meniu juostoje pasirinkus (*Analyze->Validation->Validation*). Tuomet pasirenkamas sukurtas taisyklių rinkinys, nustatoma validavimo apimtis ir spaudžiama (*Validate*).

Turėdami taisyklių rinkinį pavaizduota (28 pav.) ir modelio fragmentą pavaizduotą (29 pav.) gauname pranešimus pavaizduotus (30 pav.)



Pav. 28 Validavimo taisyklių rinkinys



Pav. 29 Realizuotas modelio fragmentas

Element	Severity	Abbreviation	
p2	wa...	PAA	Proceso neatlieka nei vienas aktorius
p1	wa...	PVF	Proceso nevaldo nei viena funkcija

Pav. 30 Gaunami pranešimai apie taisyklių neatitikimus

Atlikdami modelio validavimą galime greitai aptikti modelio klaidas ir jas ištaisyti, tačiau, kad validavimas būtų patikimas pirmiausia reikia patikrinti ar korektiškai sudarytas pats validavimo taisyklių rinkinys.

8) Profilyje sukurti paketą

Paketo sukūrimas analogiškas profilio sukūrimui žiūr. (2 punktas), tik vietoje (*Profile*) pasirenkama (*Package*).

9) Klasių diagramos sukūrimas pakete

Klasių diagramos sukūrimas aprašytas 3) punkte. Klasių diagrama kuriama pritaikymams modeliuoti skirtame pakete.

10) Sukurtam stereotipui sukurti po klasę.

Kiekvienam sukurtam stereotipui pritaikymams skirtoje diagramoje sukuriama po klasę. Klasės kuriamos diagramos meniu pasirinkus (*Class*) ir pele nurodant vietą diagramoje.

11) Klasę pavadinti stereotipo vardu.

Stereotipų pritaikymams modeliuoti skirtoms klasėms turi būti suteikiami analogiški vardai kaip ir patiems stereotipams.

12) Klasei priskirti stereotipą <<Customization>>

Pritaikymams modeliuoti skirtoms klasėms būtina priskirti stereotipą <<Customization>>. Stereotipas priskiriamas ant klasės paspaudus dešiniu pelės mygtuku ir iškrentančiame meniu pasirenkama (*Stereotype->Customization*).

13) Nustatyti pritaikymus

Pritaikymai nustatomi naudojantis lentele Nr.2, būtinas parametras yra (CustomizationTarget). Pritaikymai nustatomi pagal dalykinės srities koncepto prigimtį. Vieni konceptai gali idealiai atitikti UML notaciją, kitiems galbūt iš UML notacijos reikės tik pavadinimo. Sudarytos dalykinės srities kalbos stereotipų pritaikymai pateikti priede Nr. 1

14) Paskelbti profilį

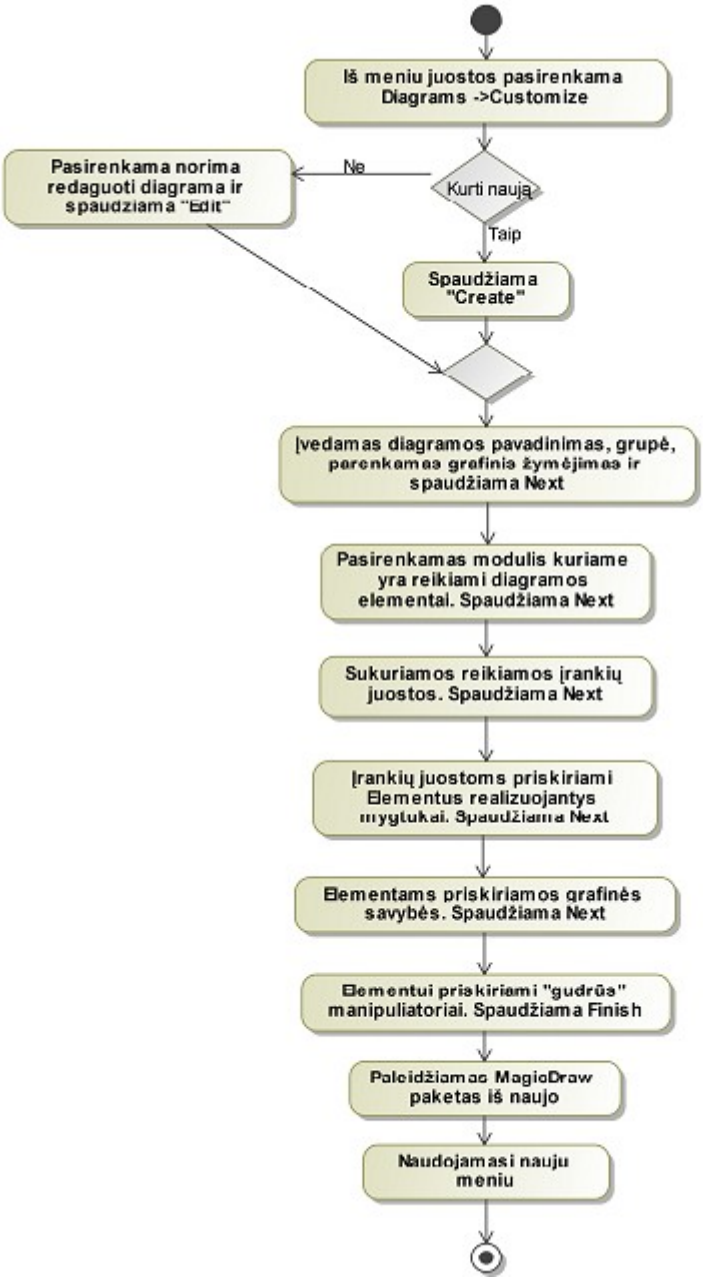
Tam, kad būtų galima sukurtus pakeitimus naudoti kituose projektuose profilį reikia paskelbti (*angl. share*). Profilio paskelbimas atliekamas programos meniu pasirinkus (*File->Share Packages*).

15) Išsaugoti projektą

Projektas išsaugomas programos meniu pasirinkus (File->Save Project).

4.4.Diagramos valdymo meniu kūrimas

Naujai sukurtus elementus galima įtraukti į jau esamas diagramas arba kurti naują diagramą. Diagramos papildymui arba sukūrimui MagicDraw pakete realizuotas diagramos pritaikymo (*angl. customize*) mechanizmas. Šio diagramos pritaikymo mechanizmo naudojimo modelis pareikiamas veiklos diagrama (31 pav.)



Pav. 31 Diagramos meniu sukūrimo veiklos diagrama

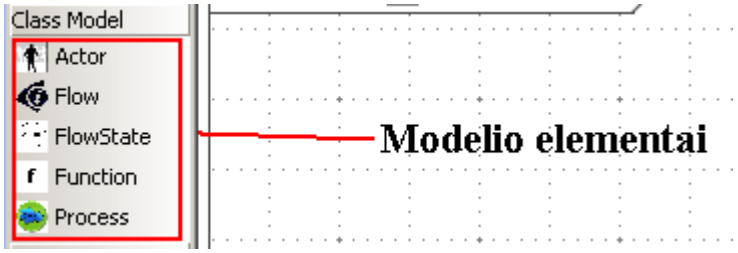
Kuriant naują diagramos menių, mes galime valdyti elementų panaudojimo teisingumą, neleidžiant piešti nekorektiškų elementų. Galime valdyti elemento vaizdavimą (pvz. vaizduoti tik grafinį žymėjimą ir pavadinimą paslepiant atributus). Galime susikurti gudrius manipuliatorius įgalinančius korektiškai ir greitai panaudoti reikiamus elementus.

4.5.Sukurtos DSK diagramos valdymo meniu

Diagramos valdymo meniu buvo sukurtas taip, kad atitiktų analizės dalyje užsibrėžtus reikalavimus. Reikalavimai dalykinės srities modeliui buvo tokie:

<ModelElement> ::= <Actor> | <Flow> | <Process> | <Function> | <FlowState> | <BusinessRule>

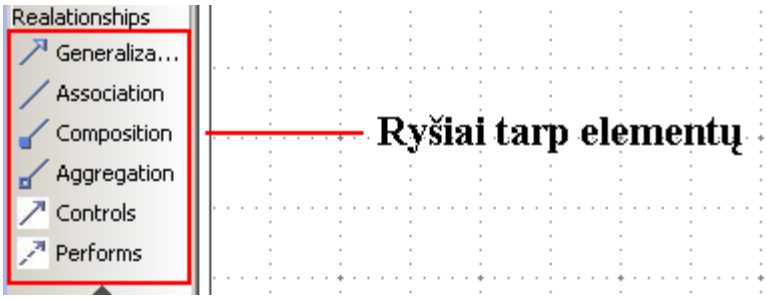
Modelio elementai turi būti: aktoriai, materialus srautai, procesai, įvykiai, funkcijos, informaciniai srautai, veiklos taisyklės, tikslai. 32 pav. pateikiamas sukurtas DSK valdymo meniu.



Pav. 32 VO modelio elementai

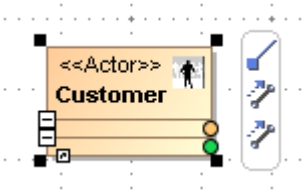
Reikia pastebėti, veiklos taisyklės elemento modelyje pavaizduoti negalime, nes UML notacijoje nėra galimybės atvaizduoti „Constraint“ elementą, kurį būtent ir paveldi veiklos taisyklės elementas.

Taip pat buvo realizuoti ir ryšiai tarp modelio elementų, ryšiai buvo realizuoti taip, kad jų prasmė atitiktų išplėstą klasių metamodelio (žiūr. pav.12) prasmę. Ryšiai pavaizduoti 33 pav.



Pav. 33 ryšiai tarp VO modelio elementų

Ryšius tarp modelio elementų taip pat galima pasirinkti iš greitojo meniu. Greitojo menių pavyzdys pateikiamas 34 pav.

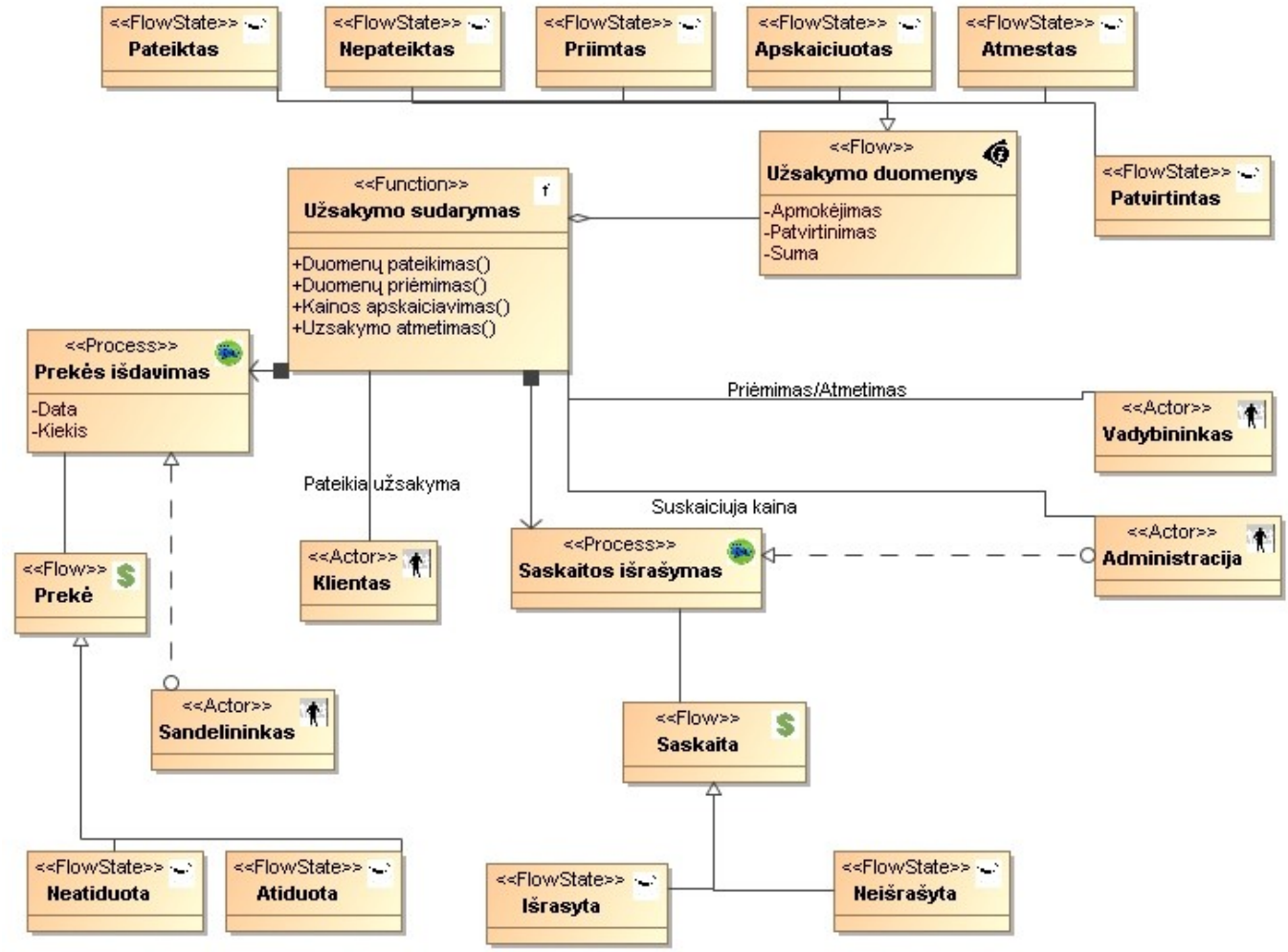


Pav. 34 Aktoriaus greitasis meniu

4.6.Prekių pardavimo veiklos modeliavimas

Parodydami sukurtos dalykinės srities galimybes sumodeliuosime „prekių pardavimo“ veiklą. Veiklos modeliavimui identifikuosime „Aktorius“, vykstančius procesus, valdančias funkcijas ir informacinius bei materialius srautus. Prekių pardavimo veikloje dalyvauja tokie aktoriai: <Klientas>, <Vadybininkas>, <Administracija> ir <Sandėlininkas>. <Sandėlininkas> atlieka <prekės išdavimo> procesą. <Prekės išdavimo> procesas transformuoja materialų srautą <prekė> , kur proceso įėjimas yra <prekė neišduota>, o išėjimas <prekė išduota>. Administracija atlieka <sąskaitos išrašymo> procesą transformuojanti materialųjį srautą <sąskaita neišrašyta> į <sąskaita išrašyta>. Administracija taip pat dalyvauja <kainos suskaičiavimo> veikloje kurią atlieka <užsakymo sudarymo> funkcija. <Užsakymo sudarymo> funkcija naudojasi <užsakymo duomenų> informaciniu srautu ir keičia jo būsenas. <Klientas> ir <Vadybininkas> dalyvauja informacinėje veikloje, <Klientas> <Pateikia užsakymą>, <Vadybininkas> jį <Priima> arba <Atmeta>.

Sukurtąja kalba sudarytas prekės pardavimo veiklos modelis pateikiamas 35 pav.



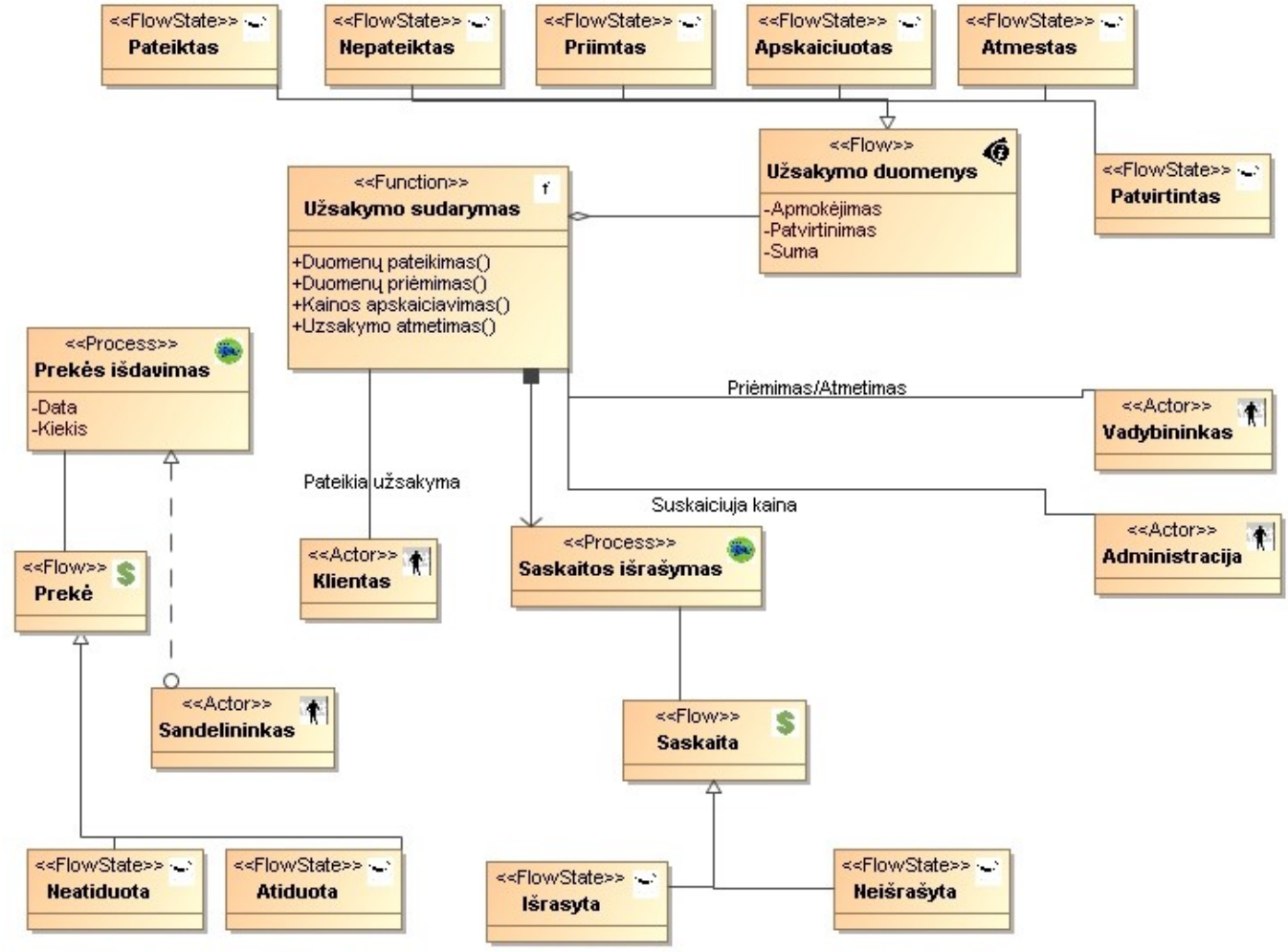
Pav. 35 Prekės pardavimo veiklos modelis

Iš paveikslėlio matome, kad realizuoti visi klasių modelio išplėtimui reikalingi elementai:

Aktoriai, Srautai, Srautų būsenos, Procesai, Funkcijos. Informaciniai ir materialieji srautai netgi turi skirtingus grafinius žymėjimus, leidžiančius lengviau juos įdentifikuoti. Realizuoti stereotipizuoti ry-

šiai tarp <Aktoriaus> ir <Proceso> bei tarp <Funkcijos> ir <Proceso>. Šitokie ryšiai leidžia greičiau aptikti elementų pavaldumą ir pavaldumo kryptį.

Modelyje pašalinkime ryšius tarp <Administracijos> ir <Sąskaitos išrašymo> bei tarp <Užsakymo sudarymo> ir <Prekės išdavimo> (36 pav.) ir pamėginkime atlikti modelio validavimą.



Pav. 36 Nekorektiškas prekės pardavimo veiklos modelis

Atlikus šio modelio validavimą išmetami pranešimai pateikiami 37 pav.

Element	Severity	Abbreviation	
Saskaitos išrašymas	warning	PAA	Proceso neatlieka nei vienas aktorius
Prekės išdavimas	warning	PVF	Proceso nevaldo nei viena funkcija

Pav. 37 Pranešimai apie nekorektišką elementų panaudojimą

Validavimo mechanizmas identifikuoja modelio elementus kuriems netenkinamos validavimo taisyklių rinkinyje užduotos sąlygos ir išmeta pranešimus apie neatitikimus taisyklėms.

Atlikus šį eksperimentą įsitikinome sukurtos dalykinės srities kalbų kūrimo metodikos veiksmingumu. Taip pat parodėme galimybę modeliuoti veiklos objektus išplėstu klasių modelio pagrindu. Patikrinkime ar sukurta kalba atitinka 2.5 skyriuje užsibrėžtus nefunkcinius reikalavimus naujai kalbai. Reikalavimai pateikiami 5 lentelėje .

Lentelė 5 Kalbos atitikimas funkciniams reikalavimams

Reikalavimas	Atitikimas
Atitikimas	Sukurtos kalbos elementai **atitinka** svarbius dalykinės srities vienetus
Ortogonalumas	Kalbos elementai **atitinka** tik vieną dalykinės srities vienetą
Suderinamumas	Kalba **suderinta** su įrankiu, nes yra galimybės kurti, trinti, modifikuoti modelio elementus
Integralumas	UML MagicDraw paketo galimybė transformuoti modelius į XML **garantuoja** integralumą
Paprastumas	Sukurta kalba yra **paprasta**, dėl paslėpto UML funkcionalumo kalba tapo daug suprantamesnė ir lengviau naudojama.
Kokybė	Kokybės tyrimas nebuvo atliktas

.

5.IŠVADOS

1. Analizės metu buvo nusistatytos gaires dalykinės kalbos kūrimo metodikai kurti. Nustatyti pagrindiniai naujos kalbos kūrimo aspektai: dalykinės srities analizė, DS modelio sudarymas, modelio realizavimas pasirinktoje platformoje, modelių korektiškumo patikrinimo realizavimas.
2. Išanalizuotos UML MagicDraw paketo galimybės kurti dalykinės srities kalbas. Buvo nustatyta, kad naujus kalbos elementus galima gauti trimis būdais: kuriant naujus **stereotipus**, stereotipams pritaikant naujas **žymes**(*angl. tag*), panaudojant apribojimus(*angl. constraint*).
3. Projektinėje dalyje buvo pasiūlyta dalykinės srities kalbų kūrimo metodika skirta UML MagicDraw paketui. Ši metodika leidžia ir mažą patirtį UML modeliavimo srityje turintiems vartotojams susikurti savo dalykinės srities kalbą.
4. Buvo sukurta veiklos objektų modeliavimo kalba išplečianti UML metamodelį ir leidžianti UML MagicDraw vartotojams modeliuoti veiklos procesus. Veiklos procesų modeliavimas klasių diagramos pagrindu suteikia galimybę, generuoti programinį kodą iš detalaus veiklos modelio.
5. Patikrintas sukurtos kalbos atitikimas užsibrėžtiems nefunkciniams reikalavimams, ir įsitikinta, kad MagicDraw UML įrankis tinka dalykinės srities kalboms kurti

6.LITERATŪRA

1. Deursen A., Klint P., Visser J.: Domain-Specific Languages: An Annotated Bibliography, – Prieiga per internetą <http://homepages.cwi.nl/~arie/papers/dslbib/>. [žiūrėta 2007-11-05]
2. Mernik M., Heering J., Sloane A.M.: When and how to develop domain-specific languages. REPORT SEN-E0517 DECEMBER 2002 – Prieiga per internetą http://citeseer.ist.psu.edu/mernik03when.html [žiūrėta 2007-12-13]
3. Szyperski C. Component software-Beyond Object Oriented Programing. Addison Wesley/ ACM Press, sedond edition, 2002 [žiūrėta 2007-12-20]
4. NoMagic MagicDraw manual, - Prieiga per internetą http://www.drake.edu/mathcs/rieck/Spring2006/cs146/md_manual.pdf [žiūrėta 2008-06 -18]
5. Dimitrios S. Kolovos, Richard F. Paige, Tim Kelly, and Fiona A.C.:Requirements for Domain-Specific Languages, Polack Department of Computer Science, University of York – Prieiga per internetą http://phoenix.labri.fr/DSPD/final/req_dsls.pdf [žiūrėta 2007-11-25]
6. Domain Specific Development - Prieiga per internetą http://www.devsource.com/article2/0,1759,2136202,00.asp [žiūrėta 2007-10-21]
7. NoMagic: UML profiling and DSL, Prieiga per internetą http://www.magicdraw.com/files/manuals/15.0/UML%20Profiling%20and%20DSL%20UserGuide.pdf?NMSESSID=dd1c1f2b65a6d827437e928689185354 [žiūrėta 2008-06-10]
8. Gudas S., Lopata A., Skersys T.: Approach to Enterprise modelling for Information Systems engineering. 2007. [žiūrėta 2008-11-25]
9. Skersys T., Gudas S.: Enterprise Model-based Generation of the Class Model. 2006 - [žiūrėta 2008-11 -24]
10. Thomas, D.: UML - Unified or Universal Modeling Language? Journal of Object Technology, Vol. 2, No. 1 (2003)

TERMINŲ ŽODYNAS

DSK –Dalykinės srities kalba (*angl. Domain Specific Language*)

Konceptas-Dalykinės srities sąvoka(*angl. concept*)

OCL- (*angl. Object Constraint Language*) objektų apribojimų kalba

VO – veiklos objektas

Žymė - (*angl. tag*) tai specifinė informacija apie elementą

Validavimas – atitikimo poreikiams patikrinimas

CASE – Computer Aided Software Enginiering

Priedas Nr.1

Veiklos objektų stereotipų pritaikymai (*angl. customizations*)