VILNIAUS UNIVERSITETAS
KAUNO HUMANITARINIS FAKULTETAS

INFORMATIKOS KATEDRA

Verslo informacijos sistemų studijų programa
Kodas 62103S138

MARTAS AMBRAZIŪNAS

MAGISTRO BAIGIAMASIS DARBAS

VEIKLOS ŽINIŲ BAZE GRINDŽIAMAS UML KLASIŲ DIAGRAMOS GENERAVIMO METODAS

Kaunas 2008

VILNIAUS UNIVERSITETAS
KAUNO HUMANITARINIS FAKULTETAS

INFORMATIKOS KATEDRA

MARTAS AMBRAZIŪNAS

MAGISTRO BAIGIAMASIS DARBAS

VEIKLOS ŽINIŲ BAZE GRINDŽIAMAS UML KLASIŲ DIAGRAMOS GENERAVIMO METODAS

Leidžiama ginti _____

Magistrantas _____
(Parašas)

Darbo vadovas: dr. Audrius Lopata
_____
(darbo vadovo mokslinis laipsnis, pedagoginis vardas, vardas, pavardė)

_____
(Parašas)

Darbo įteikimo data _____ . ___ . ___

RegistracijosNr. _____

Kaunas 2008

TURINYS

SANTRUMPŲ SĄRAŠAS

UML – Unified Modeling Language

OMG – Object Management Group

EMM – Enterprise Metamodel

XML – Extensible Markup Language

XMI – XML Metadata Interchange

IT – informacinės technologijos

IS – informacijos sistema

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

AMBRAZIŪNAS, Martas. (2008) *Method of generation UML 2.0 class diagram based on enterprise model.* MBA Graduation Paper. Kaunas: Vilnius University, Kaunas Faculty of business solutions. Humanities, Department of Informatics. 48 p.

SUMMARY

The work covers Knowledge-Based IS engineering and an enterprise metamodel's place in creation of IS. The main purpose is to traverse possibility of generating UML 2.0 class diagram's based on enterprise model. To achieve this goal there should be made these tasks:

- familiarize with enterprise metamodels and UML class diagram;
- determine and add missing elements in EMM for generation of class diagram;
- create class diagram generation algorithm and implement it;
- create prototype of using generation algorithm in creation of IS;

After analysis was found that EMM doesn't have some elements that are crucial for generation of class diagrams. For this reason enterprise metamodel was appended by two new elements. Created generation algorithm was tested with some case study data to check theoretical assumptions about relationships between EMM and class diagram. This work proved that there is possible to generate class diagrams from EMM. Although some new elements should be added for it. The scope of the work is 51 pages.

ĮVADAS

Pataruoju metu ryškėja tendencija, jog kompanijos, kurių verslo procesas yra judrus, greitai prisitaikantis prie naujų iššūkių (agile), įgyja esminius konkurencinius pranašumus bei sugeba sukurti didesnę pridėtinę vertę. Verslo proceso optimizavimas apima du aspektus: verslo proceso reinžineriją bei IS inžineriją. Sėkmingai realizavus abi minėtas veiklas, sukuriama tokia organizacijos IS, kuri adekvačiai atspindi kompanijos informacinius poreikius bei juos realizuoja. Ką tai reiškia? Visų pirma organizacijos nariai žino veiklos procesus ir taisykles, vienodai supranta struktūrą bei tikslus be to eliminuojami loginiai trūkiai tarp informacinių ir technologinių procesų t.y. sandėlininkas neturi pildyti finansinių ataskaitų, o buhalteris laukti atvykstančių prekių. Būdas apjungti abu minėtus aspektus - veiklos modeliavimas.

Veiklos modeliavimas (Enterprise modeling) yra procesas, kuris susieja veiklos reinžinerijos metodus bei veiklos kompiuterizavimo priemones. Galima teigti, jog šio proceso rezultate įvairių sričių specialistai (vadybininkai, analitikai, projektuotojai, programuotojai ir kt.) interpretuoja ir supranta organizacijoje vykstančius procesus vienodai, o tai yra vienas iš kertinių punktų organizacijos efektyvumo problemoms spręsti. Taigi galime teigti, jog IS inžinerijos metodų vystymas turi plačias perspektyvas, o šis magistrinis darbas yra glaudžiai susijęs su žiniomis grindžiama IS inžinerija. Darbo objektą galime apibrėžti, kaip IS inžinerijos projektavimo etapą, savo ruožtu tikslą formuluojame sekančiai: modernizuoti IS inžinerijos projektavimo etapą, pasinaudojant organizacijos veiklos metamodeliu. Norėdami pasiekti užsibrėžtą tikslą privalėjome realizuoti šiuos uždavinius:

- Atlikti literatūros bei esamos padėties apžvalgą žiniomis grindžiamoje IS inžinerijoje.
- Išanalizuoti veiklos metamodelius;
- Susipažinti su UML 2.0 klasių metamodeliu bei klasių diagramos elementais;
- Nustatyti potencialius sąsajos elementus tarp veiklos metamodelio ir klasių diagramos;
- Nustatyti trūkstamus elementus veiklos metamodelio elementus, kurių nebuvimas neleidžia generuoti klasių diagramos
- Sukurti klasių diagramos generavimo algoritmą;
- Realizuoti sukurtą algoritmą programiškai;
- Atlikti prototipo bandymus su testiniais duomenimis;
- Sukurti generavimo algoritmo integravimo į CASE sistemas prototipą;

Darbas yra suskirstytas į tris dalis. Pirmojoje dalyje yra apžvelgiamas einamasis IS inžinerijos etapas, veiklos metamodelio ir modelio vieta jame. Detaliai aprašomi literatūroje išskirti trys pagrindiniai veiklos metamodeliai [5][8][9] bei paaiškinama, kodėl buvo pasirinktas vienas konkretus metamodelis. Pirmoje dalyje taip pat apžvelgiama UML kalba, diagramos, įrankiai jų panaudojimo atvejai. Antroje darbo dalyje tyrinėjama galimybė iš veiklos metamodelio generuoti klasių diagramas. Nustatomi veiklos metamodelio elementai, kurie gali būti atvaizduoti į klasių diagramos elementus, numatomi trūkstami elementai bei aprašomas generavimo algoritmas. Trečioji dalis apima eksperimento bei jo vykdymo aprašymą.

Darbe naudotus literatūros šaltinius galime suskirstyti į dvi stambias grupes. Pirmoji grupė apima teorinę informaciją apie veiklos metamodelius bei UML kalbą. Ši informacija didžiąja dalimi buvo renkama iš užsienio svetainių, konkrečiai iš oficialių kūrėjų tinklalapių (pvz. UML atveju dokumentacija buvo surinkta iš OMG svetainės). Antroji grupė šaltinių apima Lietuvos mokslininkų straipsnius apie įvairių modelių generavimą iš veiklos žinių duomenų bazės.

Darbo teorinė reikšmė susiveda į tai, jog papildžius veiklos metamodelį naujais elementais, buvo sukurta automatinio klasių diagramos generavimo iš veiklos metamodelio galimybė, o tai savo ruožtu reiškia IS projektavimo etapo intelektualizavimą. Iš praktinės pusės yra sukurtas klasių diagramos generavimo algoritmo panaudojimo prototipas, apibrėžiantis šio algoritmo vietą tarp IS kūrimo įrankių t.y. numatytas CASE įrankių papildymas nauju funkcionalumu.

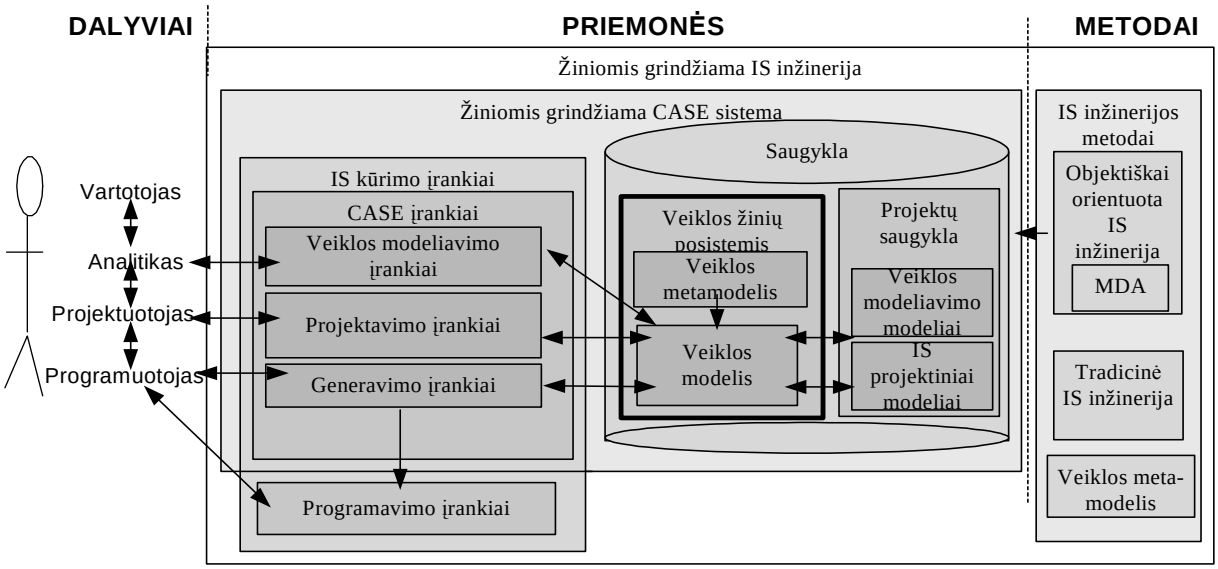
Darbas ir jo rezultatai buvo pristatyti tarpuniversitetinėje magistrantų ir doktorantų konferencijoje “Informacinės technologijos’08”.

Darbą sudaro trys dalys, apimtis - 68 puslapiai, 6 lentelės, 23 paveikslai, trys priedai.

1. UML modelių generavimo galimybės žiniomis grindžiamuose CASE įrankiuose

1.1. Esamos situacijos veiklos modeliavime apžvalga

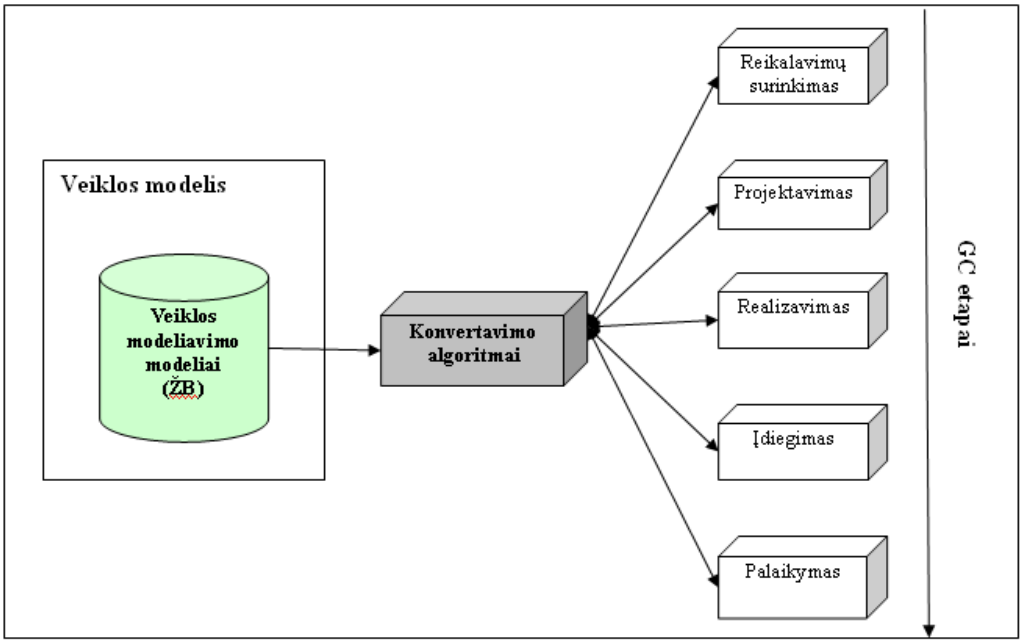
Dabartiniu metu mokslinėje literatūroje yra išskiriamas ketvirtas IS inžinerijos etapas[3]. Pagrindinis šio etapo bruožas – keleto kokybiškai skirtingų sričių apjungimas (tradicinės IS inžinerijos, veiklos modeliavimo bei dirbtinio intelekto) į vieningą visumą, siekiant sukurti CASE sistemas, kurios užtikrintų *kokybišką* ir greitą IS kūrimą bei reinžineriją. Postūmis link šių integruotų sistemų kūrimo nuo empiriškai grįstų susiformavo dėl dažnų IS projektų nesėkmių, kurių pagrindinės priežastys yra nepilnos/netikslios veiklos specifikacijos, neefektyvūs veiklos procesai, nesusikalbėjimas tarp skirtingų sričių specialistų ir iš to sekantys neadekvatūs projekto planai, biudžetai.



Šaltinis: Lopata A.,Gudas S. *Informacijos mokslai T30*, 2004, p. 90

1 pav. Ketvirtasis IS inžinerijos etapas

Ketvirtos kartos IS inžinerijos metodologijose siekiama panaudoti žiniomis grindžiamus metodus. Pagal šią koncepciją probleminės srities žinios tam tikrais būdais (pvz. darbų sekos diagramos) surenkamos į centralizuotą žinių saugyklą - veiklos žinių kaupimo posistemį. Šį posistemį sudaro du pagrindiniai elementai: veiklos modelis ir veiklos metamodelis. Kiekvieno IS GC (gyvavimo ciklo) etape kreipiamasi būtent į šią saugyklą (veiklos modelį), išgaunat konkrečiam GC etapui reikalingas žinias, o ne pakartotinai analizuojant probleminę sritį. Apibendrinant galima teigti, jog veiklos žinių posistemio paskirtis yra užtikrinti galimybę saugomų žinių pagrindu generuoti IS konceptualaus ir detalaus projektavimo etapų modelius ir programinį kodą.



Šaltinis: sudaryta autoriaus

2. pav. Veiklos modelio vieta IS inžinerijoje

Kaip jau minėta ketvirto IS inžinerijos etapo šerdis yra organizacijos veiklos metamodelis (bei iš pastarojo sekantis organizacijos modelis). Tai struktūra, kuri nusako, kaip organizacija turi būti aprašoma, kaip modeliuojama veikla, ją vykdantys žmonės, mašinos etc. Yra įvairių metamodelio koncepcijų. Keletą iš jų apžvelgsime ir paaiškinsime dėl kokių priežasčių buvo pasirinktas vienas konkretus metamodelis.

1.2. Organizacijos veiklos metamodelis

Bendrąja prasme metamodelis yra modeliavimo kalbos *modelis*, kuriame nusakytos pagrindinės kalbos savybės. Tai gramatika (ko reikia, kad sakinys būtų teisingas, kokia elementų tvarka ir pan.), kurios pagalba rašomi sakiniai[1] (kuriami modeliai). Savo ruožtu *veiklos* metamodelis - tai formali žinių struktūra, apibūdinanti dalykinės veiklos srities (organizacijos) esminius elementų tipus ir jų sąveikų tipus. Tai teoriškai pagrįstų taisyklių rinkinys, kurio pagalba patikrinamas dalykinės srities žinių korektiškumas bei teisingumas. Tokiu būdu veiklos modelis atspindi konkrečią organizaciją, jos veiklos procesus, struktūrą, o veiklos metamodelis apibrėžia tai, kas teoriškai yra teisinga modeliuojamai sričiai, pvz. ar verslo procesai atitinka valdymo teorijos principus, šakos verslo logiką ir t.t.. Tai reiškia, jog talpinant veiklos modelį į CASE įrankio žinių bazę, galima išvengti klaidų, kurios yra būdingos empiriniam veiklos modeliavimui, kuris remiasi tik specialisto, aprašančio organizaciją, kompetencija ir patirtimi.

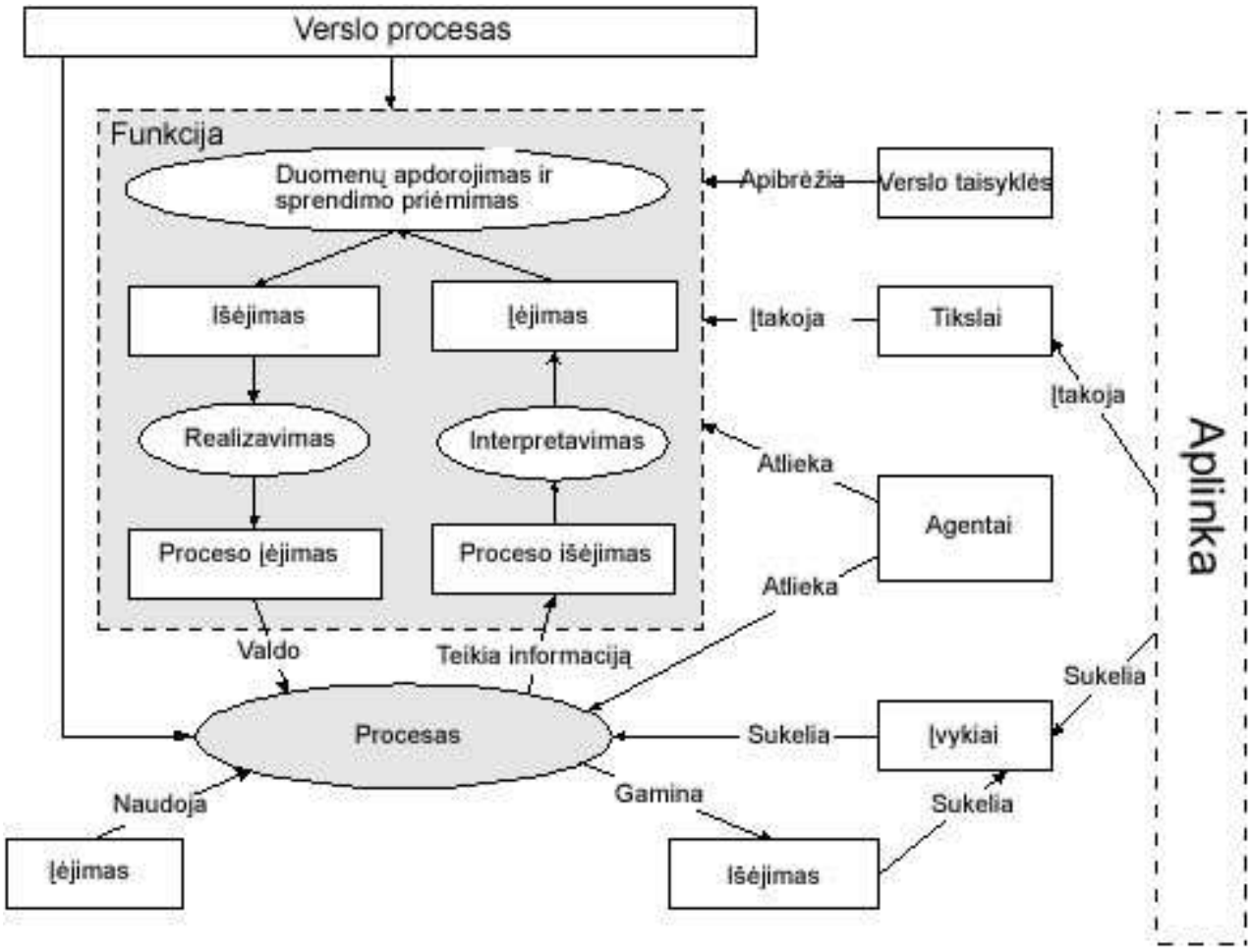
Viena iš žinomiausių ir jauniausių veiklos modeliavimo metodologijų yra UEML. Šios metodologijos kūrimas tėra pradinėje vystymo stadijoje, todėl pasiūlytasis karkasas (framework) nėra detalizuotas. Pats poreikis kurti šią veiklos modeliavimo metodologiją buvo iššauktas keleto priežasčių:

- Daug veiklos modeliavimo kalbų, modelių;
- Nenusistovėjusi terminija ir kintančios modeliavimo paradigmos;
- Daug tarpusavyje nesuderinamų modeliavimo priemonių;
- Nenuoseklus formalus veiklos modeliavimo ir veiklos inžinerijos pagrindas;

Su šiomis problemomis yra dažnai susiduriama įgyvendinant informacijos sistemų projektus. Taigi reikalinga metodologija, kuri minėtas problemas sėkmingai spręstų. Įvedus į organizacijos veiklos modeliavimo procesą EMM, atsiranda galimybė validuoti (tikrinti teisingumą) renkamus duomenis apie organizaciją teoriškai teisingų teiginių pagrindu. Esminiai UEML siūlomi EMM elementai yra iš valdymo teorijos žinomi proceso bei funkcijos objektai. Paveiksle 3 pav. iliustruota pagrindinė UEML siūloma idėja t.y. išorinis įvykis inicijuoja procesą, kuris yra valdomas tam tikros funkcijos. Valdymas vyksta panaudojant signalus (management attributes), kurie formuojami funkcijos informacijos apdorojimo (information processing) modulyje, kuriam informaciją suteikia proceso būsenos atributai. Kiekviena funkcija turi unikalius informacijos apdorojimo, interpretavimo bei realizavimo (information processing, interpretation, realisation) algoritmus/metodus, kuriuos sudaro veiklos taisyklės/apribojimai, reikalavimai, aritmetinės išraiškos. Svarbu paminėti, kad konkreti funkcija nustatytais laiko tarpais gali valdyti vieną ir daugiau procesų – tokia funkcija vadinama

[1] Turėtume prisiminti, jog UML (Unified Modeling Language) yra kalba

bendrąja. Tolesnė veiksmų seka yra tokia: įvykdytas procesas formuoja išeigą (materialų srautą), kuris inicijuoja įvykį, įvykis savo ruožtu gali inicijuoti kitą procesą ir t.t. – tokiu būdu formuojamos darbų sekos (process chain) ir jų hierarchijos.



Šaltinis: Sudaryta autoriaus pagal UEML report by F. Vernadat UEML Interest GroupIFAC-IFIP Task Force, Agenda for Dec. 16th, 1999 meeting in Paris.

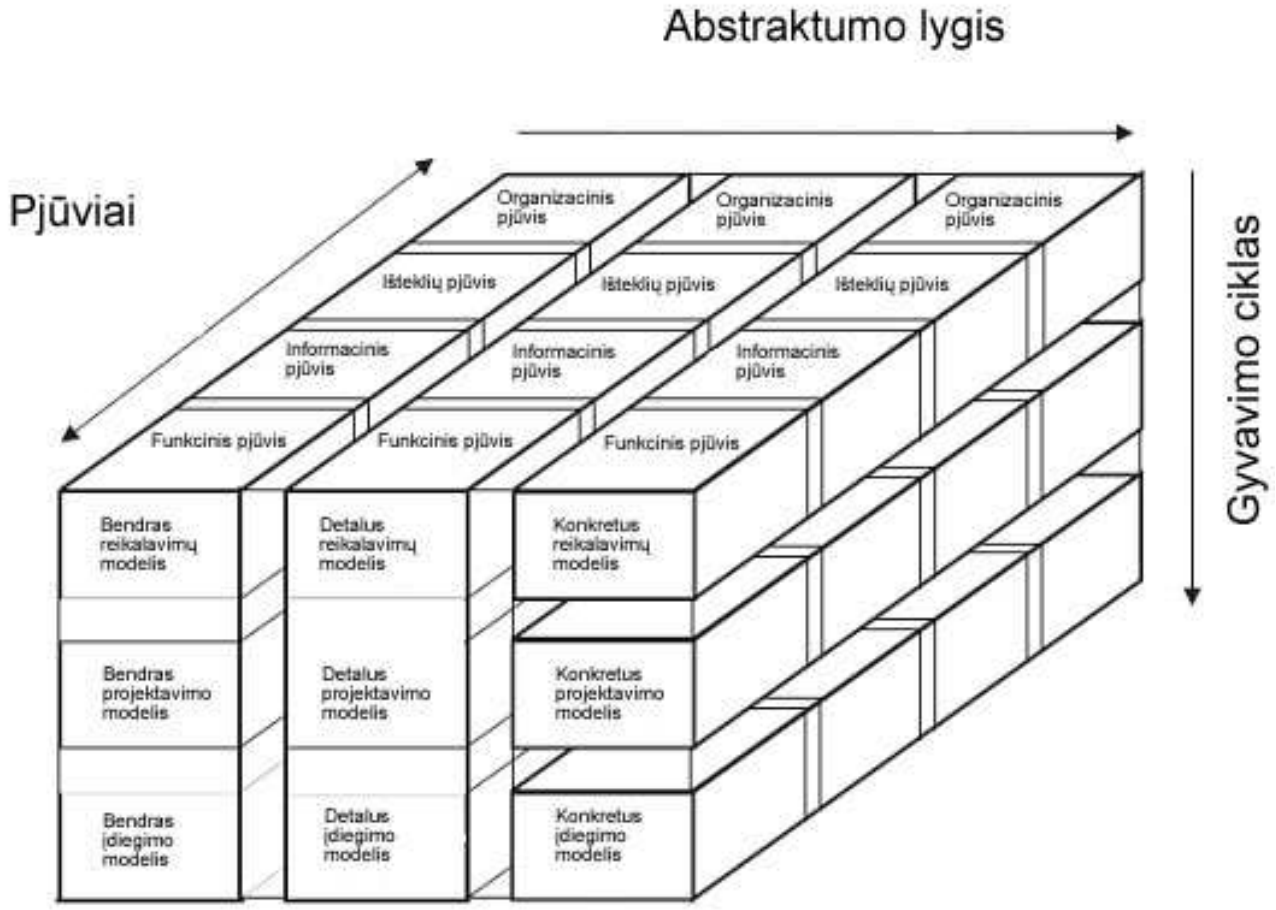
3. Pav. Veiklos metamodelis remiantis UEML

Dar viena organizacijos veiklos modeliavimo metodologija yra CIMOSA. Tai metodologija, kuri aprašo organizaciją, nuo aukščiausio strateginio lygmens iki žemiausio operatyvinio. Ši metodologija susideda iš trijų dalių:

- Veiklos modeliavimo metamodelis (Enterprise modeling framework)
- Veiklos modeliavimo kalba (Enterprise modeling language)
- Pirmą ir antrą punktą apjungianti infrastruktūra (Integrating infrastructure)

Metodologijos idėja vaizdžiai parodoma pasinaudojant trimis kubo dimensijomis: modeliuojamos organizacijos pjūvis(kokia sritis modeliuojama), modelių abstraktumo lygis (kaip

detaliai modeliuojama) bei IS gyvavimo ciklo etapas (kada modeliuojama). Minėtos trys dimensijos teoriškai yra pakankamos aprašyti organizacijos struktūrą, veiklos procesus, aktorius, informacinius srautus skirtingais abstraktumo lygiais bei laiko momentais.



Šaltinis: Sudaryta autoriaus pagal Scheer, A.W. ARIS - Business Process Frameworks

4. Pav. CIMOSA Kubas

CIMOSA kubas apima keturis veiklos modeliavimo aspektus (pjūvius):

- Funkcinis pjūvis aprašo darbų sekas (work flows) ;
- Informacinis pjūvis aprašo funkcijų įėjimus ir išėjimus (Inputs and Outputs of Functions);
- Resursų pjūvis atspindi išteklių struktūrą, hierarchiją (žmoniškųjų, informacinių, materialių);
- Organizacinis pjūvis atspindi organizacijos kompetencijas, gebėjimus bei atsakomybes;

Taip pat yra išskiriami trys abstraktumo (apibendrinimo) lygmenys:

- Bendrasis – svarbus apskritai organizacijai;

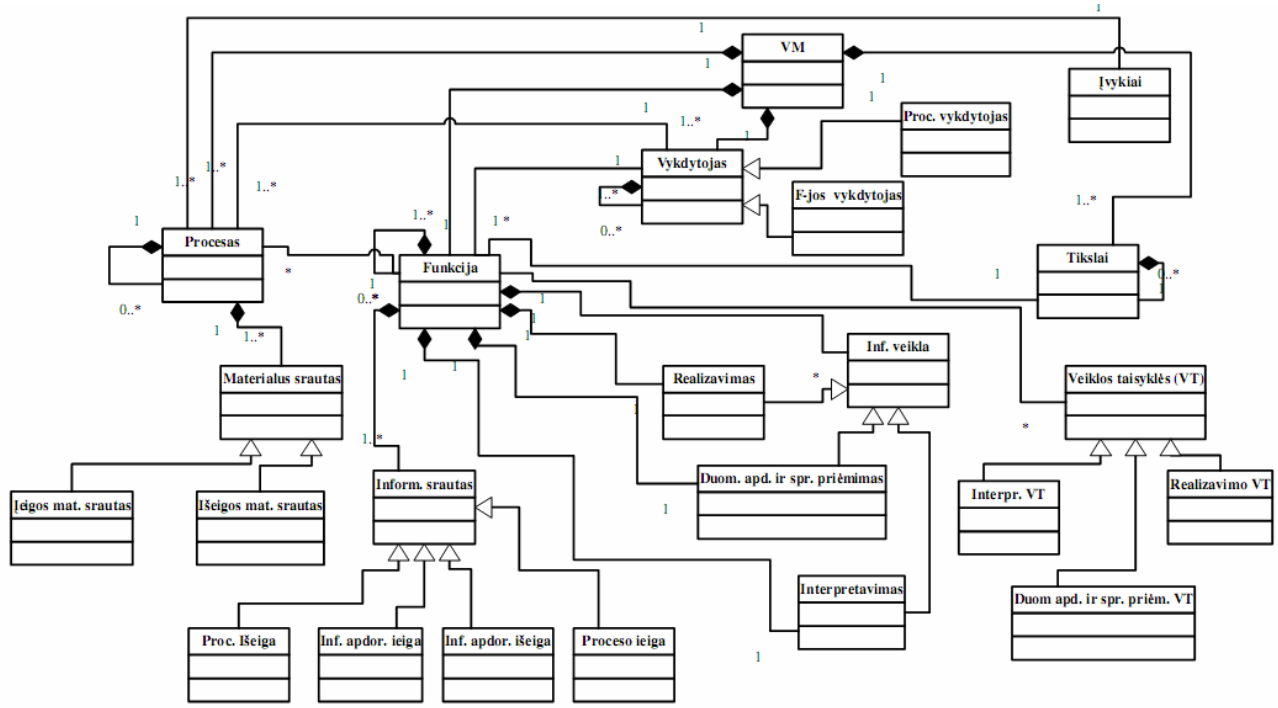
- Detalus – svarbus verslo šakai;
- Konkretus – skirtas konkrečiai organizacijai;

Trečioji kubo dimensija apima informacinės sistemos gyvavimo ciklo etapus. Tam yra naudojami apibendrinti „krioklio“ metodo išskiriami IS gyvavimo ciklo etapai:

- Specifikacijų surinkimas;
- Projektavimas;
- Realizavimas ir testavimas;
- Įdiegimas;
- Palaikymas;

Nors CIMOSA ir yra viena iš populiariausių organizacijos veiklos modeliavimų metodologijų, tačiau einamuoju metu didžiausias dėmesys yra skiriamas Europos Sąjungos lėšomis sukurtam ir tobulinamam UEML standartui.

Šiame darbe yra remiamasis veiklos metamodeliu, kuris buvo sukurtas Kauno Technologijos universiteto, informatikos fakulteto Informacijos sistemų katedroje. Pastarasis metamodelis pasirinktas atsižvelgiant į tai, jog Lietuvos sąlygomis jį vystyti yra efektyviausia be to šis metamodelis buvo kuriama remiantis UEML principais.



Šaltinis: S. Gudas, A. Lopata, Workflow models based acquisition of enterprise knowledge, Information Technology and Control, 2007, p.36

5. Pav. KTU mokslininkų sukurtas veiklos metamodelis

Pagrindiniai metamodelio elementai, kurie yra aktualūs šiam darbui yra: vykdytojas, procesas, funkcija, informacinis bei materealus srautas. Metamodelio pirminė paskirti buvo saugoti žinias panaudos atvejų diagramoms generuoti, tolesnis jo vystymas nukreiptas į kitų UML diagramų generavimo galimybių išplėtimą.

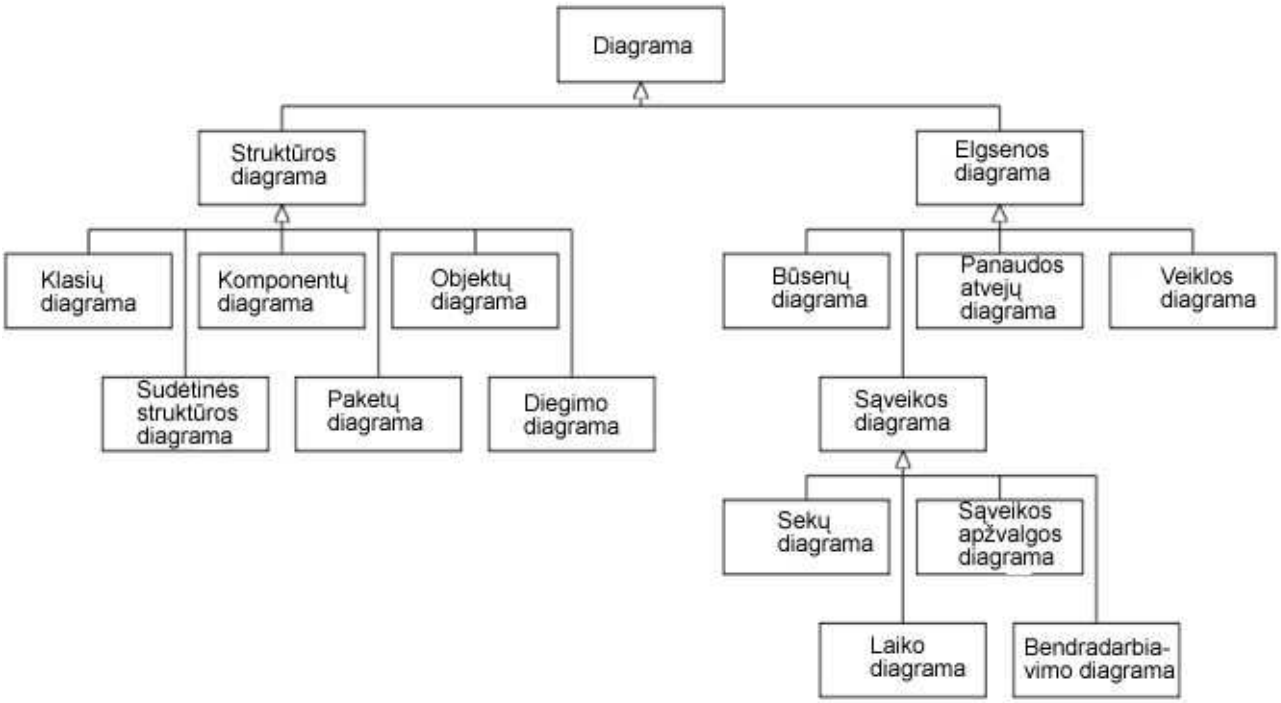
1.3. UML (Unified Modeling Language)

UML galime apibrėžti kaip universalią modeliavimo kalbą, skirtą kurti grafinius sistemų modelius t.y. vizualizuoti bei dokumentuoti programinės įrangos sistemas, aprašant jų struktūrą, elgseną ir bendravimą su kitomis sistemomis. Einamuoju metu UML 2.0 yra laikoma „de facto“ standartu tarp modeliavimo kalbų.

„All models are wrong but some models are useful"(„Visi modeliai yra neteisingi, bet kai kurie modeliai yra naudingi“) - George Box. Taigi norint kurti nors ir neteisingus, bet reikalingus modelius, būtina turėti kokybišką, nuoseklią bei pilną modeliavimo kalbą. Tokios kalbos poreikis yra ypač aktualus stambioms kompanijoms, kurių informacinės sistemos ir produktai yra sudėtingi, kompleksiški, didelės apimties. Sėkmingas tokių sistemų palaikymas ir išplėtimas tiesiogiai priklauso nuo to kaip detaliai ir tiksliai ji yra aprašyta (jos struktūra, funkcionalumas, elgsena). Aiškus ir vienareikšmiškas sistemos struktūros aprašymas leidžia pakartotinai panaudoti objektus/bibliotekas (juos kurti tokiu būdu, kad pakartotinis panaudojimas būtų efektyvus ir kiek įmanoma greitas bei paprastas), nuosekliau plėsti sistemą, nustatyti problemiškas sritis sistemoje, greičiau aptikti ir neutralizuoti logikos bei programavimo klaidas. Nors pabrėžėme, jog kokybiška modeliavimo kalba yra ypač svarbi stambioms kompanijoms dėl jų sistemų apimties ir sudėtingumo, tačiau tai taip pat aktualu ir mažesniems projektams bei sistemoms. Natūralu, jog mažesnės kompanijos yra labiau priklausomos nuo žmogiškojo faktoriaus, todėl dažnai sąmoningai ar nesąmoningai tokiose kompanijose yra vadovaujamasi „Agile“ sistemų kūrimo principais, kurie nors ir rekomenduoja modeliuoti sistemas, tačiau modelių detalumas ir jų kiekis paprastai būna mažesnis nei stipriai formalizuotose didelėse kompanijose. Priklausomai nuo sistemos kūrimo etapo, kuriami skirtingi modeliai (bei jų detalumo lygis). Pavyzdžiui pradiniame IS kūrimo etape surinkti funkciniai sistemos reikalavimai atvaizduojami panaudos atvejų diagramomis, verslo procesai – veiklos diagramomis, organizacijos struktūra – klasių, paketų diagramomis. Priklausomai nuo to kokiems žmonės yra kuriami modeliai (klientams ar vidiniam naudojimui) skiriasi modelių abstraktumo lygis, detalumas. Kiekviena organizacija pasirenka kiek ir kokių modelių reikia kurti priklausomai nuo projekto apimties, sudėtingumo. Atsižvelgiant į tai pasirenkamas diagramų poaibis iš siūlomų trylikos (UML 2.0 yra trylika diagramų). Kaip jau minėta, priklausomai nuo panaudojimo tikslo, modelių abstraktumo lygis gali būti skirtingas t.y. modeliai gali parodyti sistemą įvairiais pjūviais ir skirtingais detalumo lygiais, priklausomai nuo to kokie sistemos elementai yra paslepiami, o kokie akcentuojami, siekiant didesnio aiškumo konkrečiam atvejui.

OMG grupės pateiktuose UML kalbos specifikacijose teigiama, jog su UML 2.0 galima modeliuoti įvairių tipų programinę įrangą, nepriklausomai nuo programavimo kalbos, operacinės sistemos, techninės įrangos, tinklo struktūros. Tai reiškia, jog nors siekiama orientuotis į objektines programavimo kalbas (C++, Java, C#) bei objektiškai kuriamas sistemas, tačiau UML galima sėkmingai naudoti ir ne-OO sistemoms ir programavimo kalboms (VB, Fortran, COBOL). Kalbant apie UML būtina aiškiai suprasti skirtumą tarp pačios modeliavimo kalbos bei įrankių, kurie ją realizuoja per modelių kūrimo galimybes. Priklausomai nuo įrankio, skiriasi palaikomų diagramų skaičius, skirtingų programavimo kalbų palaikymas etc. Esant didelei UML įrankių įvairovei, atsiranda problema su diagramų suderinamumu t.y. ar vienas įrankis teisingai atvaizduos kito įrankio sukurtą diagramą. Šiai problemai spręsti OMG buvo sukurtas duomenų apsikeitimo standartas XMI, kurio pagalba įmanoma UML modelius iš vieno įrankio perkelti į kitą įrankį neprarandant informacijos.

Paskutinėje UML versijoje (2.0) yra 13 modelių (diagramų), kurie yra suskirstytos į tris grupes, pagal tai kokius sistemos aspektus modeliuoja. Šis skirstymas yra detalus, tačiau praktikoje dažnai naudojamas tik dviejų modeliuojamų aspektų skirstymas t.y. modeliai atvaizduojantys statitinę (struktūros diagramos) sistemos informaciją bei modeliai atvaizduojantys dinaminę (elgsenos diagramos) sistemos informaciją.



Šaltinis: Sudaryta autoriaus pagal OMG Unified Modeling Language: Superstructure

6. Pav. UML diagramų hierarchija

Struktūros diagramos (Structure diagrams). Šios grupės diagramos atvaizduoja sistemos struktūros vaizdą tam tikru fiksuotu laiko momentu. t.y. sistemos vaizdas yra statinis ir nekintantis. Modeliuojami elementai yra klasės, paketai, atributai, ryšiai tarp jų ir kt. Grupėje yra šešios diagramos.

- Objektų diagrama (Object Diagram) – parodo sistemos (arba jos dalies) struktūrą. Objektų diagramos yra konkretesnės už klasių diagramas, kadangi modeliuojami realūs atvejai t.y. klasių diagramoje objektas yra apibrėžtas kaip karkasas, o šiose diagramose tas karkasas būna užpildytas realiais duomenimis.
- Komponentų diagrama (Component Diagram) - vaizduoja fizinę sistemos struktūrą t.y. iš kokių bibliotekų, failų, modulių, vykdomųjų failų sistema yra sudaryta bei kaip minėti elementai yra susiję tarpusavyje.
- Sudėtinės struktūros diagrama (Composite Structure Diagram) - detalizuoja klasių vidinę struktūrą bei bendravimo su kitomis klasėmis atvejus.
- Paketų diagrama (Package Diagram) - atvaizduoja tai, kaip sistema yra suskirstyta į logines savarankiškas dalis bei ryšius tarp šių daliu. Projektuojant sistemą, siekiama, kad butų pasiektas maksimalus paketo elementų bendrumas bei minimali priklausomybė nuo kitų paketų.
- Diegimo diagrama (Deployment Diagram) - modeliuojama techninė ir programinė įranga, kuri reikalinga, informacinei sistemai įdiegti, taip pat kokie komponentai diegiami į kokią techninę įranga bei kokiais ryšiais (protokolais) bendraus atskiri objektai

Elgsenos diagramos (Behavior diagrams). Šių diagram paskirtis atvaizduoti dinaminį modeliuojamos srities vaizdą. Svarbų vaidmenį atlieka laiko dimensija bei sąlygos, kurios keičia objekto būseną. Grupėje yra trys diagramos.

- Panaudos atvejų diagrama (Use Case Diagram) - pagrindinė reikalavimų surinkimui ir reikalavimams atvaizduoti skirta diagrama. Joje atsispindi veikiantys aktoriai bei funkcijos, kurias aktoriai gali atlikti.
- Veiklos diagrama (Activity Diagram) - atvaizduojamas sistemos (arba jos dalies) darbas/veikla pažingsniui. Akcentuojamos veiklos, informaciniai bei kontrolės srautai.
- Būsenų diagrama (State Machine Diagram) - aprašo visas galimas objektų (pvz. klasių) būsenas bei sąlygas, kurios įtakoja būsenos pakitimą.

Sąveikos diagramos (Interaction diagrams). Tai poaibis elgsenos diagramų, kurios aprašo objektų tarpusavio bendravimą.

- Sekų diagrama (Sequence Diagram) - atvaizduoja skirtingus procesus bei objektus egzistuojančius tuo pačiu laiko momentu bei apsikeitimą žinutėmis tokia tvarka, kuria šis apsikeitimas vyksta iš tiesų t.y. išlaikytas nuoseklumas laiko atžvilgiu.
- Bendradarbiavimo diagrama (Communication Diagram) - atvaizduoja žinučių srautus tarp objektų OO sistemose bei parodo pagrindinius ryšius tarp objektų.
- Laiko diagrama (Timing Diagram) - sąveikos diagrama, kurioje akcentuota laiko dimensija. Ši diagrama skirta ištirti objektų elgseną duotuoju laiko periodu. Tai specifinis sekų diagramų poaibis.
- Sąveikos apžvalgos diagrama (Interaction Overview Diagram) - koncentruotai aprašo kontrolės srautus, kurie egzistuoja bendravimo tarp objektų metu. Tai veiklos diagramos variantas, kuriame bendravimo atvejai yra atvaizduojami kaip pagrindiniai elementai.

Lentelėje pateiktas trumpas diagramų aprašymas bei panaudojimo aktualumas

1. Lentelė

UML diagramos

Diagrama	Trumpas apibūdinimas	Aktualumas
Activity Diagram	Atvaizduoja verslo procesus, sąlygas, informacinius srautus pažingsniui.	Didelis
Class Diagram	Atvaizduoja statinius sistemos elementus - klasės, jų tipai, sandara bei ryšiai.	Didelis
Communication Diagram	Atvaizduoja klasių realizacijas, bendradarbiavimą su kitais objektais bei žinutes, kuriomis vyksta bendravimas.	Mažas
Component Diagram	Atvaizduoja komponentus, kurie sudaro aplikaciją, sistemą, organizaciją.	Vidutinis
Composite Structure Diagram	Detalizuoja vidinę klasių, komponentų ar panaudos atvejų struktūrą.	Mažas
Deployment Diagram	Atvaizduoja realizuojančią sistemos aplinką t.y. programinę bei techninę įrangą kurioje veikia sistema.	Vidutinis
Interaction Overview Diagram	Veiklos diagramos variantas, kuris skirtas atvaizduoti valdymo informacijos srautus sistemoje arba verslo procese.	Mažas
Object Diagram	Atvaizduoja objektus ir ryšius tarp jų konkrečiu laiko momentu. Paprastai tai išskirtinis klasių arba bendravimo diagramos atvejis..	Mažas
Package Diagram	Atvaizduoja, tai kaip sistema yra suskirstyta į paketus bei ryšius tarp paketų.	Mažas
Sequence Diagram	Atvaizduoja procesus bei objektus egzistuojančius tuo pačiu laiko momentu bei apsikeitimą žinutėmis.	Didelis
State Machine Diagram	Atvaizduoja objekto būsenos kitimą bei perėjimus, kurie sukelia būsenos pakitimus.	Vidutinis
Timing Diagram	Atvaizduoja objekto elgseną tam tikru laiko intervalu.	Mažas
Use Case Diagram	Atvaizduoja panaudos atvejus, aktorius bei jų sąveiką.	Vidutinis

Nors UML pripažįstamas kaip „de facto“ modeliavimo kalbų standartas, tačiau ir ši kalba neišvengia kritikos. UML yra sudėtinga ir kompleksiška kalba. Norint suprasti daugumą diagramų reikalingas specialus pasiruošimas. Dėl pastovaus kalbos tobulinimo sudėtinga ir brangu palaikyti kvalifikaciją. Didelis diagramų skaičius apsunkina mokinimąsi be to realiai naudojamos tik kelios diagramos iš keliolikos, o naujų versijų konstruktai dažnai atneša daugiau painiavos nei naudos. Nors deklaruojama, jog UML vienodai efektyviai gali būti naudojama OO bei ne-OO kalbomis rašomoms sistemoms, tačiau realybėje kuriami modeliai yra labiau orientuoti į OO sistemas. Dar vienas dalykas, kuris teoriškai atrodo naudingas, tačiau praktiškai yra neefektyvus, tai XMI. Problema yra ta, jog vienu įrankiu sukurti modeliai, kitame įrankyje atvaizduojami su įvairiais informacijos praradimais, kas reiškia dalinį arba visišką diagramos sugadinimą.

1.4. Veiklos metamodelių ir UML modelių sąsajos literatūroje

Analizuotoje literatūroje buvo pateiktos idėjos ir metodai, kurie leidžia iš veiklos modelių generuoti tam tikras UML diagramas pvz. straipsnyje „Veiklos modeliu grindžiamas UML activity diagramos generavimas“ [2] analizuojama UML 2.0 veiklos diagramos (activity diagram) generavimo iš veiklos modelio galimybė. „Kadangi kiekvienos UML 2.0 *Activity* diagramos klasės objektas gali būti atvaizduotas iš veiklos modelio, tai reiškia, jog pasirinktojo veiklos modelio sudėtis yra pakankama šios diagramos generavimui.“ Kituose literatūros šaltiniuose bendrai apžvelgtos žinių inžinerijos metodų pritaikymas IS projektavime: “Tokios naujos mokslinės vadybos pakraipos ir sritys kaip žiniomis grindžiamų organizacijų kūrimas (angl. *Enterprise Knowledge),* organizacijų žinių valdymas ir organizacijų projektavimas pradeda naudoti IS inžinerijos priemones – CASE sistemas, savo problemoms modeliuoti ir spręsti. Tuo būdu gali būti taupomas darbo laikas, gerinama sprendimų kokybė, nes veiklos modelyje sukauptos jau patikrintos žinios apie probleminę sritį.” [3]. Pateikti straipsniai atspindi tendenciją, jog žiniomis grįsta inžinerija tampa populiaria ir perspektyvia tyrimų sritimi, kurios tikslas palengvinti visų rūšių organizacijos specialistų darbą bei organizacijos veiklos integravimą IS pagrindu. Atliekant literatūros analizę teko susipažinti su panašaus darbo metu sukurtu klasių diagramos metamodeliu [4]. Pirmasis dalykas, kuris kelia tam tikrų abejonių yra tai, jog pateikiamas klasių diagramos metamodelis labai panašus į organizacijos metamodelį pagal UEML. Kodėl tai keista? Visų pirma organizacijos metamodelis aprašo funkcijas ir procesus plačiai, taip kad informacijos apie probleminę sritį būtų pakankamai įvairiems modeliams generuoti, o tai reiškia, jog klasių diagramos metamodelis turėtų būti labiau specializuotas, kadangi jis naudoja tik poaibį organizacijos duomenų. Kitas aspektas yra tai, jog tokie elementai, kaip postCondition/preCondition labiau panašūs į panaudos atvejų diagramos elementus nei į klasių. Taigi realiai atrodo, jog pateikiamas metamodelis yra EMM pritaikytas generuoti klasių diagramas, o ne klasių diagramų metamodelis. Tiesa, neteko pamatyti konkrečiai probleminei sričiai sugeneruotų modelių, tačiau iš to kas pateikta kyla abejonių kaip tiksliai rezultatas atspindi projektavimo etapui reikalingus duomenis. Manau pateikiamas aukšto lygio procesų ir funkcijų diagrama, kurios duomenys yra pernelyg abstraktūs programinio kodo generavimui, taigi žemesnio lygio diagramas reiktų generuoti vėl remiantis empirinėmis žiniomis. Tačiau net ir tokiu atveju modelių generavimas yra naudingas, kadangi leidžia vizualiai pateikti užsakovams statinę informaciją apie organizacijos procesus bei funkcijas, pagal pateiktus duomenis.

Minėti literatūros šaltiniai liečia teorinius veiklos modeliavimo aspektus remiantis žiniomis grindžiama inžinerija. Kita grupė apžvelgtų šaltinių orientuoti į UML aprašymą bei panaudojimą. Visų

pirma tai OMG (Object Management Group) pateikta UML dokumentacija (techninė informacija, terminai, notacija, pavyzdžiai). Ši grupė yra UML standarto kūrėja, taigi natūralu, jog pateikiama išsami informacija apie UML

"Best Practices for Applying UML, Part I" [6] – internetinis leidinys, leidžiamas vienos iš geriausių IS modeliavimo CASE sistemos kūrėjų (www.magicdraw.com). Kuriame akcentuojami išskirtinai į praktiką orientuoti UML panaudojimo konceptai. Pateikiami atrodytų subjektyvūs, tačiau "de facto" standartais tapę UML panaudojimo principai.

1.5. UML įrankiai

UML yra modeliavimo kalba, kuri nesusieta su konkrečiu įrankiu, tai reiškia, jog norint sukurti sistemos modelį/pjūvį užtenka turėti baltą popieriaus lapą ir pieštuką. Su šiomis priemonėmis sukurtas modelis nebus prastesnis nei tas, kuris buvo kuriamas su brangiu modeliavimo paketu. Kalbant apie egzistuojančius modeliavimo įrankius, nepavyko rasti tokių, kurie generuotų organizacijos modelius remiantis jos metamodeliu ir pagal šį metamodelį surinktais duomenimis. Realu, jog tokios sistemos egzistuoja integruotų sprendimų pagrindu t.y. apima kelias skirtingas sistemas (DB, organizacijos metamodelio konceptas, modeliavimo CASE sistema) ir kurios yra kuriamos pagal specifinius užsakymus. Atsižvelgdami į tai, apibūdinsime bendrąsias savybes, kuriomis pasižymi CASE sistemų rinkos lyderiai. Tiesa, pati sąvoka lyderiai yra pakankamai subjektyvi, tačiau šiuo konkrečiu atveju, tai sistemos, kurios siūlo plačiausią funkcionalumą ir yra pripažintos vartotojų.

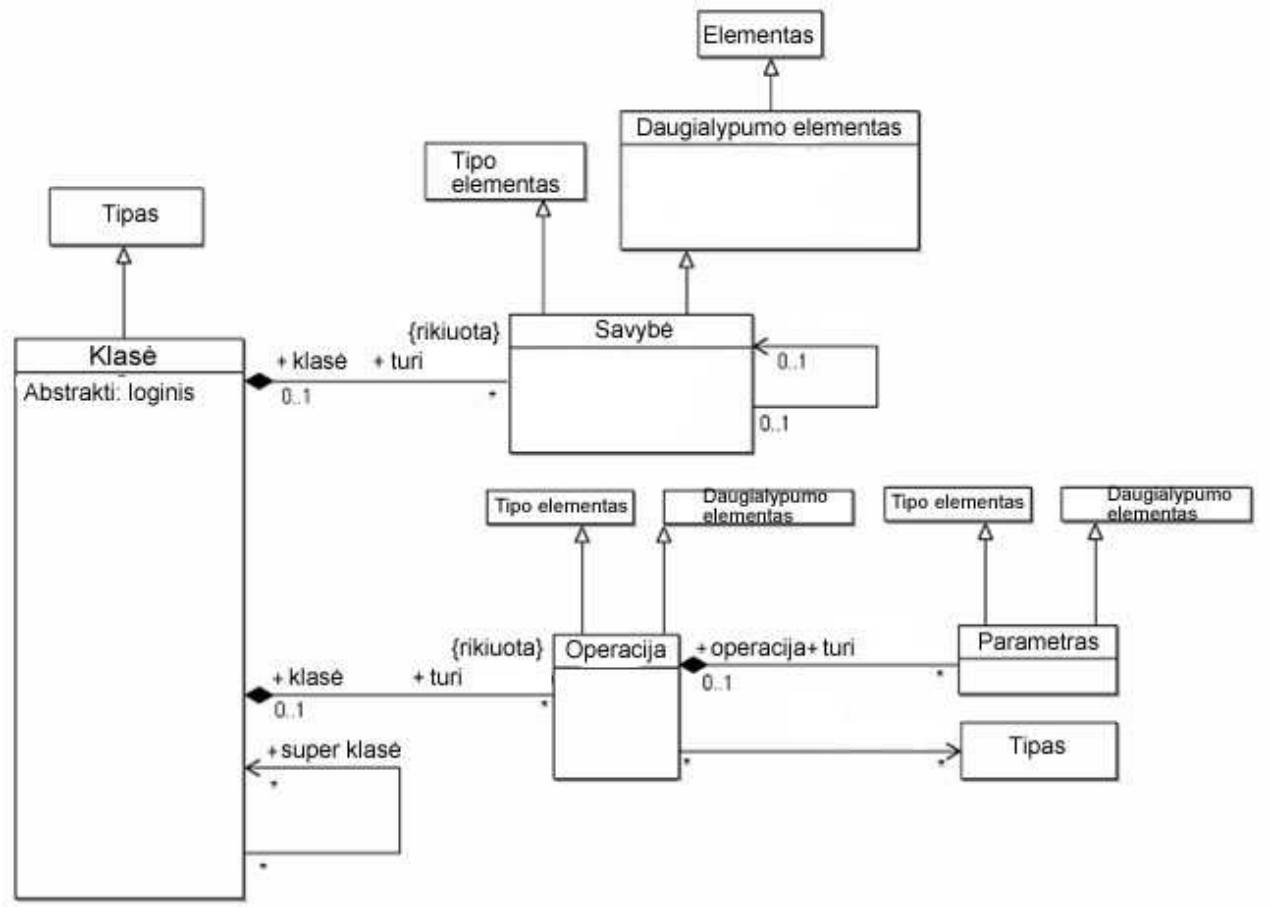
- **UML modelių palaikymas.** Naujausios UML versijos palaikymas yra neatsiejamas nuo modeliavimui skirto CASE įrankio. Paprastai siektina, jog būtų palaikomos visos UML diagramos, tačiau ir čia galioja Pareto (80/20) dėsnis. Taigi pagrindinis dėmesys skiriamas panaudos atvejų, klasių, būsenų bei veiklos modeliams.
- **Pirminė (forward) inžinerija.** Tai funkcionalumas, kuris reiškia, jog iš egzistuojančių modelių yra sugeneruojamas veikiantis programinis kodas. Kol kas ši savybė realizuojama tik klasių diagramos, kitoms diagramos to realizuoti einamuoju metu neįmanoma. Siektina, jog aprašius organizaciją, veikiantį kodą būtų galima generuoti remiantis bet kuriuo modeliu, taigi susiduriame su organizacijos metamodelio būtinybe ir centralizuotu duomenų apie probleminę sritį valdymu.
- **Grįžtamoji (reverse) inžinerija.** Iš principo tai visiška pirminės inžinerijos priešingybė t.y. turint programinį kodą, sugeneruojami modeliai. Vėl gi kol kas realizuotina su klasių diagramomis. Problema yra tai, jog iš kodo gauta informacija nėra transformuojama į organizacijos metamodelio aprašytus elementus, taigi kitokių diagramų generavimas yra neįmanomas.

Iš įrankių, kuriuos išskiria vartotojai yra, „Enterprise Architect“, „MagicDraw“, „Rational Software modeler“. Minėti įrankiai palaiko pirminę bei grįžtamąją inžineriją, gali būti suderinami su įvairiomis šiuolaikinėmis programavimo kalbomis, taip pat su įvairiomis IDE, tačiau vis tiek modeliavimo procesas vykdomas etapais, kurie tik empiriškai susiejami prie prieš tai buvusiais modeliais. Idealus CASE įrankis turėtų leisti suvesti duomenis apie organizaciją, jos procesus,

funkcijas, įvykius t.y. tai ko reikalauja šios sistemos metamodelis. Iš suvestų duomenų turi būti galimybė generuoti norimas diagramas, programinį kodą pasirinktai sričiai, procesui ir pan. Pasikeitus organizacijoms veiklos procesams, modeliai taip pat pergeneruojami taip, kad atspindėtų realią situaciją. Atvirkščias procesas Suvedami duomenys ir matome, jog organizacijos veiklos procesai nėra optimalūs, taigi atliekame jų reinžineriją, kad įtakoja metamodelio pagrindu surinktus duomenis, o tai reiškia ir kitus modelius. Taigi akivaizdu, jog dabartinių CASE sistemų trūkumas yra EMM nebuvimas, tai reiškia, jog IS kūrimas atliekamas empiriškai (dirbančių žmonių patirtis, sugebėjimai), o tai dažnai lemia loginius (logical gap) trūkius ir kitas problemas.

1.6. UML klasių metamodelis ir jo elementai

Pagal UML 2.0 klasifikaciją (žr pav. 6) visos diagramos yra skirstomos į du tipus – struktūros (structural) bei elgsenos (behavioral). Klasių diagrama priskiriama prie statinių struktūros diagramų. Pagal apibrėžimą klasių diagrama yra skirta objektiškai modeliuoti sistemos struktūrą t.y. aprašyti sistemą sudarančius objektus, jų sudėtį (atributai, operacijos) bei tarpusavio ryšius. Praktine prasme šie modeliai leidžia projektuoti/įvertinti detalią visos sistemos struktūrą. Klasių metamodelis pateikiamas 7 pav.

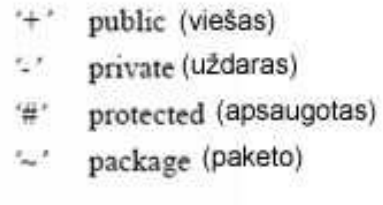


Šaltinis: Sudaryta autoriaus pagal OMG Unified Modeling Language: Superstructure

7. Pav. UML klasių metamodelis

Klasių diagramoje atsispindi du esminiai objektinio projektavimo/programavimo principai t.y. inkapsuliacija (paslėpimas) bei paveldėjimas. Objektų inkapsuliacijos pagrindinis uždavinys yra užtikrinti objekto savarankiškumą bei bendravimą tik per griežtai nustatytas sąsajas t.y. kitiem objektams neturi būti žinoma/matoma, tai kaip klasė, su kuria jie bendrauja, realizuoja sąsajas. Inkapsuliacija yra įgyvendinama pasinaudojant prieinamumo operatoriais (access modifier). Paveldėjimas yra vienas iš keturių ryšių tipų tarp dviejų klasių. Prieinamumo operatoriai priklauso nuo

programavimo kalbos dialekto pvz. programavimo kalboje „Microsoft“ .Net C# yra keturių tipų prieinamumo operatoriai: private, protected, internal, public. UML 2.0. išskiriami taip pat 4 tipų paveldėjimo operatoriai, kurių notacija pavaizduota 8 pav.



Šaltinis: Sudaryta autoriaus pagal OMG Unified Modeling Language: Superstructure

8. Pav. UML prieinamumo operatoriai

Antroje lentelėje yra pateikiami klasių diagramos elementai bei jų aprašymas.

2. Lentelė

UML klasių diagramos elementai

Elementas	Aprašymas
Klasė	Pagal apibrėžimą klasė apibūdina aibę objektų, kurie turi tas pačias savybes (atributai, operacijos). Visi objektai, kuriuos aprašo klasė privalo turėti klasės karkaso užpildymo reikšmes t.y. kai yra kuriamas objektas, klasės karkasas užpildomas konkrečiais objekto duomenimis Klasė yra pagrindinis klasių diagramos elementas. Jo pagalba yra įmanoma atvaizduoti sistemos struktūros elementus bei ryšius tarp jų.
Abstrakti klasė	Specinis klasių poaibis. Šio tipo klasės neturi savo realizacijų ir yra naudojamos konstruoti kitoms klasėms. Jos aprašo bendrą kelioms klasėms struktūrą ir pasinaudojant paveldėjimo ryšiu perduoda šią struktūrą subklasėms. Dažniausiai realiai neegzistuoja objektai, kurie aprašomi šiomis klasėmis t.y. egzistuoja tik subklasių objektai.
Interfeisas/sąsaja	Tai operacijas ir atributus aprašanti struktūrą. Priklauso nuo programavimo kalbos dialekto. Skirtingai nei klasėse, minėti elementai nėra realizuojami pačiame interfeise. Tai reiškia, jog patys interfeisai nėra naudojami sukurti objektams, juos naudoja klasės, tam kad standartizuotų bendravimą. Tai reiškia, jog jeigu klasė naudoja kažkokį interfeisą, tai kitos klasės (kurios bendrauja su pastaraja) turi būti realizavusios minėtą interfeisą.
Ryšys	Ryšių pagalba klasės, abstrakčios klasės bei interfeisai apjungiami į vieningą visumą, kuri reprezentuoja modeliuojamos sistemos statinę struktūrą. Klasių diagramose išskiriami keturių tipų ryšiai. ▪ *Asociacija* – dažniausiai naudojamas ryšys, kuris atspindi, jog viena

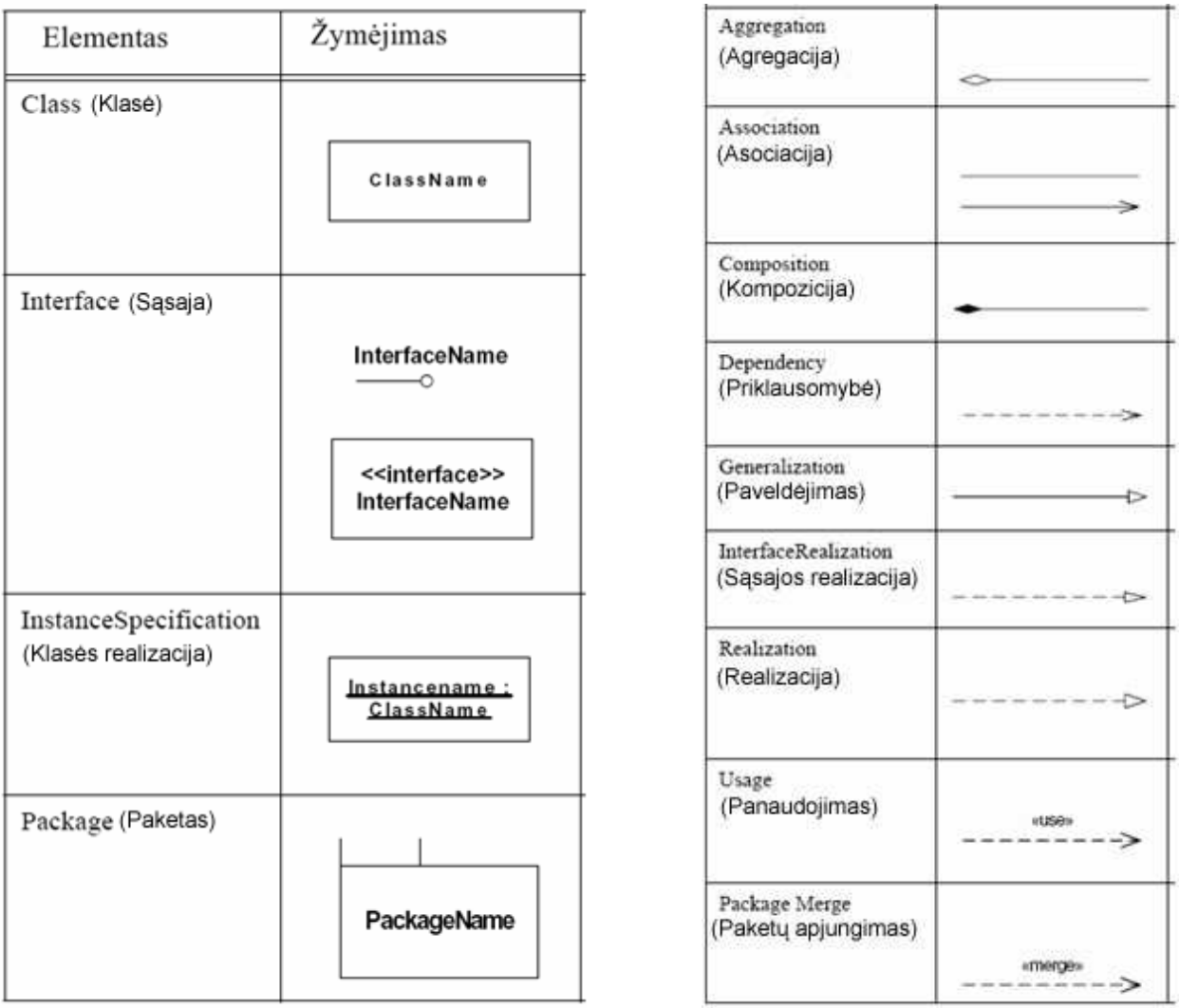
Elementas	Aprašymas
	klasė turi kažkokią sąveiką su kita klase. Klasių statusai yra vienodi. Gali būti kryptinė asociacija. ▪ *Agregacija* – Šiuo ryšiu sujungti objektai nėra vienodo statuso. Jis atspindi visuma-dalis sąveiką. Paprastai sakant, viena klasė būna sudaryta iš kitų klasių. Sudarantys elementai nėra būtini, jie gali būti konkurentiškai naudojami kitų objektų, sukūrimas ir sunaikinimas nepriklauso nuo visumos sukūrimo/sunaikinimo, tačiau paprastai sukuriami ir sunaikinami kartu su visuma. ▪ *Kompozicija* – griežta agregacijos forma. Šiuo atveju pagrindinę klasę sudarantys elementai negali būti konkurentiškai naudojami, dalis priklausomi nuo visumos t.y. sunaikinus visumą, būtinai sunaikinamos ir sudedamosios dalys. ▪ *Paveldėjimas/realizavimas* – ryšys, kuris apibrėžia, jog vienos klasės (superklasė), visi elementai (operacijos, atributai) yra perduodami (paveldimi) kitai klasei (subklasė).
Parametras (Property)	Statinę informaciją saugantys klasės laukai, tai pvz. pavadinimas, id ir pan. Atributus aprašo jo duomenų tipas (gali būti kita klasė) bei prieinamumas (access modifier)
Operacija	Operacijos, kurias gali atlikti klasė pvz. getData(), showDateTime() ir pan. Operacijas aprašo reikiami parametrai, grąžinamas tipas bei prieinamumas (access modifier).

Plačiau aprašysime šiuos elementai bei kaip jie susiję. Visų pirma pagrindinis akcentas yra klasė. Iš OO žinome, jog tai yra objekto karkasas (savybių, operacijų) aprašymas. Imkime paprastą pavyzdį klasė „įmonė“, tokiu atveju objektas (klasės realizacija) galėtų būti UAB „Alna“. Iškart galime pastebėti, jog bet kokia organizacija yra sudaryta iš padalinių/funkcinių skyrių (marketingo, finansų, personalo valdymo ir pan.), taigi šiuo atveju naudotume agregacijos bei kompozicijos ryšius. Perduodama tam tikrus darbus kitoms kompanijoms („outsorcindama“), kompanija susitartų su rangovais, kokias operacijas ir kaip jas turės atlikti, taigi rangovai realizuos reikalaujamą interfeisą (tas pats ir su klientais). Aptartas pavyzdys pateiktų tik aukšto lygio modelį (atliekant sistemą programavimą, naudojami žemo lygio architektūriniai modeliai), tačiau jis leistų pamatyti organizacijos ir jos verslo aplinkos statinę sąveiką, kas leistų patikslinti sistemos specifikacijas.

Sugebėjimas išskirti esmines sistemos esybes (klases su jų atributais bei operacijomis), leidžia išvengti sudėtingų problemų (funkcijų dubliavimo arba „išmėtymo“ į kelias klases, „pritemptų“ ryšių, sudėtingo sistemos palaikymo bei išplėtimo) vėlesniuose IS kūrimo etapuose. Ryšių išskyrimas nėra toks sudėtingas procesas, kadangi dažniausiai nustatytos esybės intuityviai jau būna susijusios. Kiek kitokia situacija su paveldėjimo ryšiu. Iš principo tai dirbtinis ryšys, kurį išskiria

architektai/programuotojai, siekdami optimizuoti sistemos išplėtimo bei palaikymo galimybes. Interfeisai taip pat yra dirbtinė konstrukcija, kuri sukuriama, siekiant palengvinti klasių bendravimą tiek pačioje sistemoje, tiek su išorinėmis sistemomis.

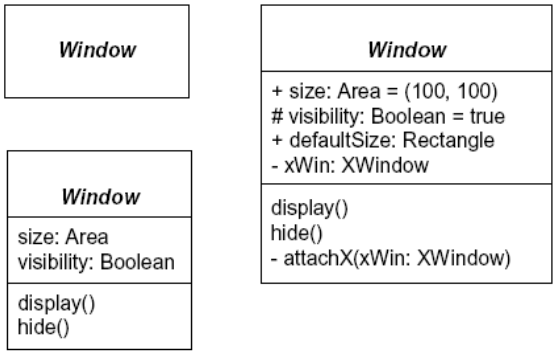
Klasių diagramose naudojama nedaug grafinių elementų. Galima būtų apibendrinti, jog tai tik stačiakampiai su klasės pavadinimu ir atributų ir operacijų aprašymu bei ryšių linijos. Interfeisai ir abstrakčios klasės atvaizduojami taip pat stačiakampiais, kurie turi papildomus elementus, interfeisas turi raktinį žodį „interface“ arba apskritimą prie pavadinimo, abstrakčių klasių stačiakampiai, priklausomai nuo UML įrankio gali būti papildyti raktiniu žodžiu „abstract“ arba apvesti punktyriniu rėmeliu. Elementų naudojamų struktūros diagramose atvaizdavimas parodytas žemiau pateiktuose paveiksluose.



Šaltinis: Sudaryta autoriaus pagal OMG Unified Modeling Language: Superstructure

8 pav. Klasių diagramos elementai

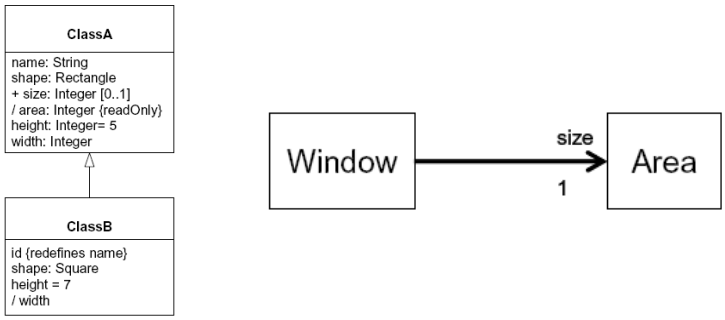
Nedidelis elementų skaičius sąlygoja tai, kad klasių diagramų braižymas gali būti nesudėtingas, greitai išmokstamas ir nereikalaujantis specializuotų įrankių. Visi šie aspektai prisideda prie to, jog klasių diagramos yra pripažįstamos, kaip didelio svarbumo diagramos (žr. lentelė 1). UML kalbos teikiamas privalumas modeliams yra skirtingo abstraktumo lygio pasirinkimas. Tai reiškia, jog modeliai nebūtinai privalo būti detalūs kad būtų teisingi.



Šaltinis: Sudaryta autoriaus pagal OMG Unified Modeling Language: Superstructure

9 pav. Klasių abstraktumo lygiai

Klasių diagramose naudojami keturių tipų ryšiai agregacija, kompozicija, asociacija, paveldėjimas.



Šaltinis: Sudaryta autoriaus pagal OMG Unified Modeling Language: Superstructure

10 pav. Paveldėjimo ir asociacijos ryšys

Verslo analitikas bendraujantis su užsakovu dažniausiai siekia gauti/parodyti abstraktų visos sistemos vaizdą, savo ruožtu programuotojui reikalinga kuo daugiau ir kuo smulkesnės informacijos, kuri gali būti pvz. prieinamumo operatoriai, atributų duomenų tipai ir reikšmės, operacijų parametrai ir kt. Jeigu verslo analitikas rodytų tokio tipo klasių diagramas užsakovams, būtų gaunamas neigiamas efektas, kadangi tik specialistas gali suprasti specifinius diagramų elementus, tuo tarpu kai paprastam vartotojui tai butų tik papildomi nežinomi elementai, kurie apsunkina diagramos skaitymą bei supratimą.

2. Klasių diagramos generavimas iš žiniomis grindžiamo CASE įrankio saugyklos

2.1 Veiklos modelio užpildymas duomenimis, naudojant WorkFlow diagramas

Prieš analizuojant galimus ryšius tarp veiklos ir klasių metamodelių bei vieno atvaizdavimo į kitą galimybes, turime apžvelgti, kaip EMM yra užpildomas duomenimis. Konkrečiu atveju tam yra naudojami Work Flow modeliai. Šie modeliai yra plačiai paplitę praktikoje t.y. organizacijų veiklos procesams aprašyti ir modeliuoti. Teoriškai veiklos modelį galima užpildyti naudojant ir kitokius metodus. WorkFlow modeliai aprašo veiklas (activity), materialius ir informacinius veiklų mainus bei aktorius kurie šias veiklas atlieka. Imant mažiausią šių modelių elementą gauname tokią veiksmų seką:

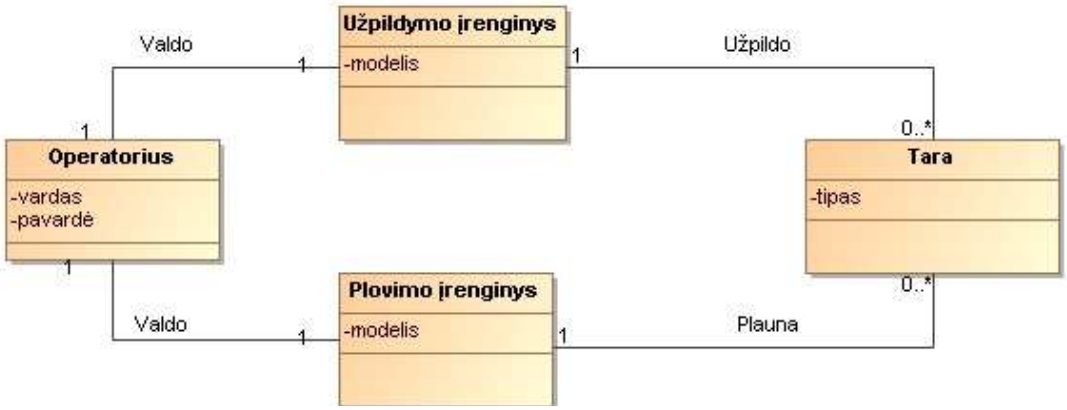
- Įėjimas;
- Įėjimo transformavimas;
- Išėjimas.

Įėjimas ir išėjimas gali būti traktuojamas kaip informacinis ar materialus srautas. Informacinis srautas praktikoje apima dokumentus, įsakymus etc. Materialus srautas apima prekes, fizinį inventorių etc. Įėjimo srautą transformuoja procesas arba funkcija. Pagal valdymo teorijos principus, kiekvienas procesas turi jį valdančią funkciją, kuriai procesas pateikia tam tikrą informaciją. Šią informaciją funkcija interpretuoja bei siunčia atgal veiklos nurodymus. Taigi norint sugeneruoti klasių diagramą, būtina pažvelgti į procesų ir funkcijų vidų, ieškant juose veikiančių esybių t.y. ne taip svarbu, kokie veiksmai atliekami, svarbiau kokie aktoriai (techninė, programinė, žmoniškoji įranga) veikia. Kad galėtume pamatyti, koks kokybinis skirtumas yra tarp WorkFlow diagrama aprašyto organizacijos proceso bei tos pačios srities klasių diagramos, panagrinėsime pavyzdį. Sakykime modeliuojamas procesas, kurio metu yra užpildoma tuščia tara (Stačiakampiai su stačiais kampais vaizduoja objektus, o stačiakampiai užapvalintais kampais veiklas (activity)):

Šaltinis: Sudaryta autoriaus

4. Pav. Taros užpildymo procesas

Reikia pabrėžti, jog konkrečiu atveju analizuojame tik procesą (pilname modelyje kiekvienas procesas turi jį valdančią funkciją). Taigi matome, jog aiškiai galime matyti veiklų eiliškumą, kiekvienos veiklos įeigą bei išeigą, tačiau šiuo konkrečiu atveju nematome (tiksliau matome ne visas) esybių, kurios dalyvauja kiekvienoje iš veiklų. Pavyzdžiui, jeigu sudarytume šiam procesui klasių diagramą ji galėtų atrodyti taip:



Šaltinis: Sudaryta autoriaus

12. Pav. Taros užpildymo proceso klasių diagrama

Taigi galime pastebėti, jog esminis klasių diagramų aspektas yra esybės. Vadinasi, analizuojant tiek procesą, tiek funkciją svarbu išskirti jos dalyvius, kuriais gali būti žmonės, techninė įranga, programinė įranga bei jų sąveika, kurią atspindi ryšiai.

2.2. Veiklos modelio elementų atvaizdavimas į klasių diagramos elementus

Remiantis praktinėje literatūroje rekomenduojamais klasių diagramos kūrimo principais, pagrindiniai elementai, iš kurių gali būti konstruojamos klasės yra aktoriai (veiklos metamodelyje tai yra vykdytojai). Informaciniai bei materialūs srautai taip pat atvaizduojami į klases, kadangi tai yra savarankiški objektai (pvz. sąskaita-faktūra ar konkreti prekė), kurie turi savo parametrus bei operacijų rinkinius. Aktoriai atlieka (dalyvauja) procesus ir/arba funkcijas, todėl pastarieji elementai atsivaizduojami kaip klasių operacijos(metodai). Metodų įėjimo bei išėjimo parametrai yra informaciniai arba materialūs srautai, kurie yra atvaizduojami į klases. Klasių parametrai yra suformuojami iš vykdytoją/informacinį/materialų srautą apibūdinančių atributų.

Kadangi klasių diagrama yra sudaroma iš klasių bei ryšių jungiančių klases, kitas svarbus elementas, kuris turi būti veiklos žinių metamodelyje, tai informacija apie objektų ryšius su kitais objektais. Iš pradinio veiklos metamodelio nustatyti visų tipų ryšius tarp objektų yra neįmanoma. Asociacijos ryšys yra atrenkamas per atributus, tačiau likę trys ryšių tipai neatsispindi pirminiame veiklos metamodelyje. Tai reiškia, jog šią informaciją reikia papildomai surinki. Analogiška padėtis ir su prieinamumo operatoriais t.y. jie taip pat negali būti nustatyti iš egzistuojančio metamodelio elementų. Paveiksle 13 grafiškai yra pavaizduoti egzistuojantys ryšiai tarp metamodelių elementų. Kaip matome veiklos metamodelyje išsiskiria procesas, funkcija vykdytojas, informacinis bei materialus srautai. Statiniai elementai atsivaizduoja į klasę, o dinaminiai (funkcija, procesas) į klasės operacijas.

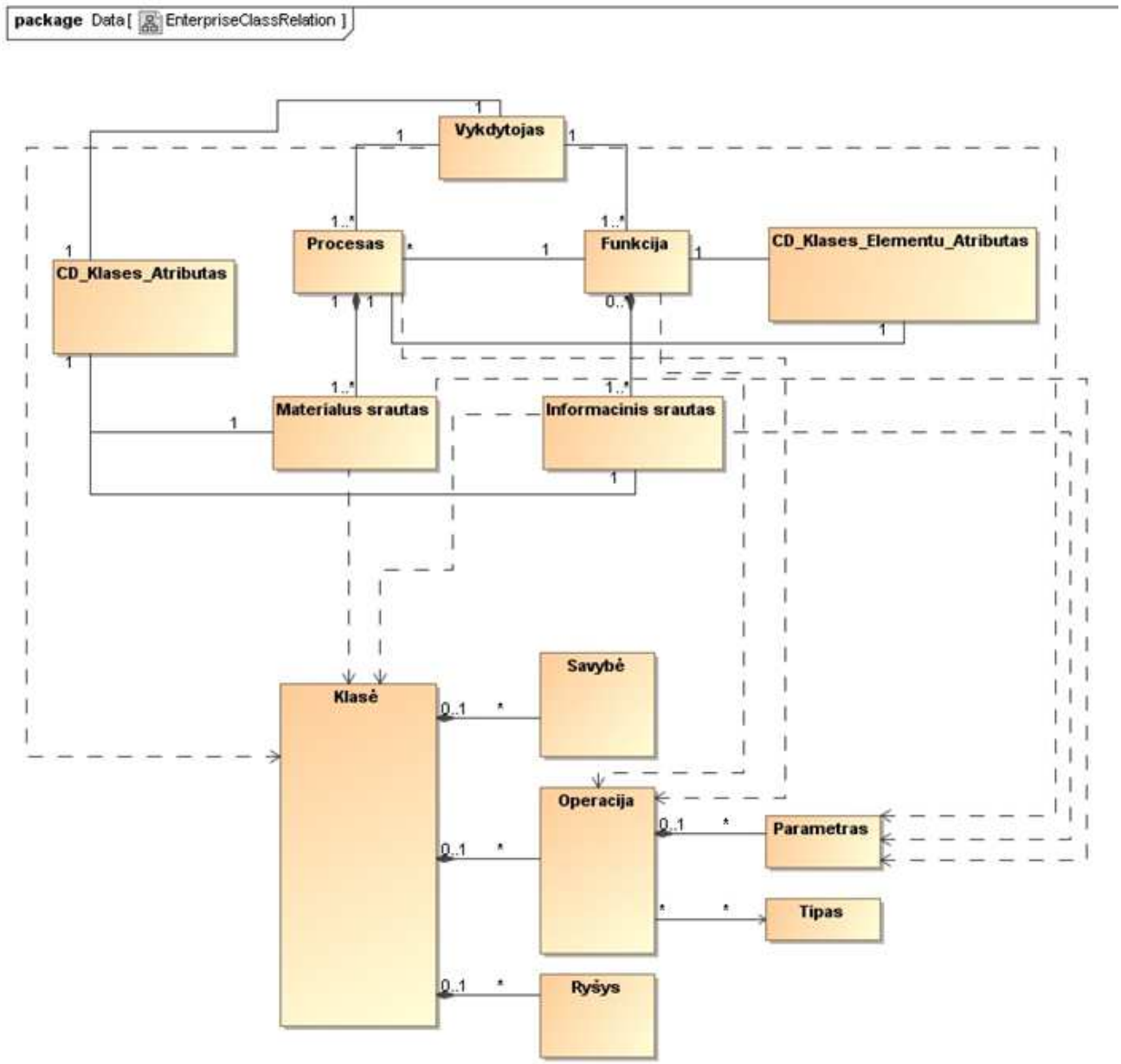
3. Lentelė

Veiklos metamodelio ir klasių metamodelio sąsaja

Veiklos metamodelio elementas	**Atvaizdas**	**Klasių diagramos metamodelio elementas**
Vykdytojas	φ1	Klasė, *abstrakti klasė*
Informacinis srautas	φ2	Klasė, *abstrakti klasė*
Materialus srautas	φ3	Klasė, *abstrakti klasė*
Procesas	φ4	Operacija
Funkcija	φ5	Operacija
Parametras	φ6	Atributas

Šioje vietoje apibendrindami galime teigti, jog bazinis metamodelis, kuris buvo skirtas tik UML 2.0 „Use Case“ diagramoms generuoti, saugo pakankamą žinių kiekį, tam kad būtų įmanoma generuoti

klasių diagramas, kurios paprastai naudojamas pradiniame projektavimo etape t.y. be prieinamumo operatorių bei sudėtingų ryšių tipų.



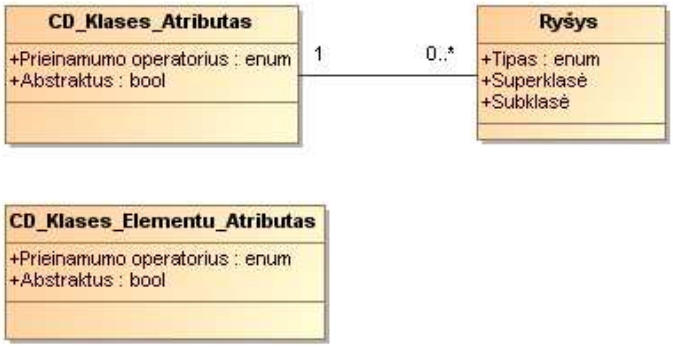
Šaltinis: Sudaryta autoriaus

13. Pav. Veiklos metamodelio ir klasių metamodelio sąsaja

Detalesnių diagramų generavimui, veiklos žinių metamodelį reikia papildyti elementu, kuris saugotų informaciją apie klasių (metodų/atributų) prieinamumą bei abstraktumą, taip pat apie kompozicijos agregacijos bei paveldėjimo ryšius. Taigi veiklos metamodelis turi būti papildytas elementu, kuris saugotų šią papildomą informaciją.

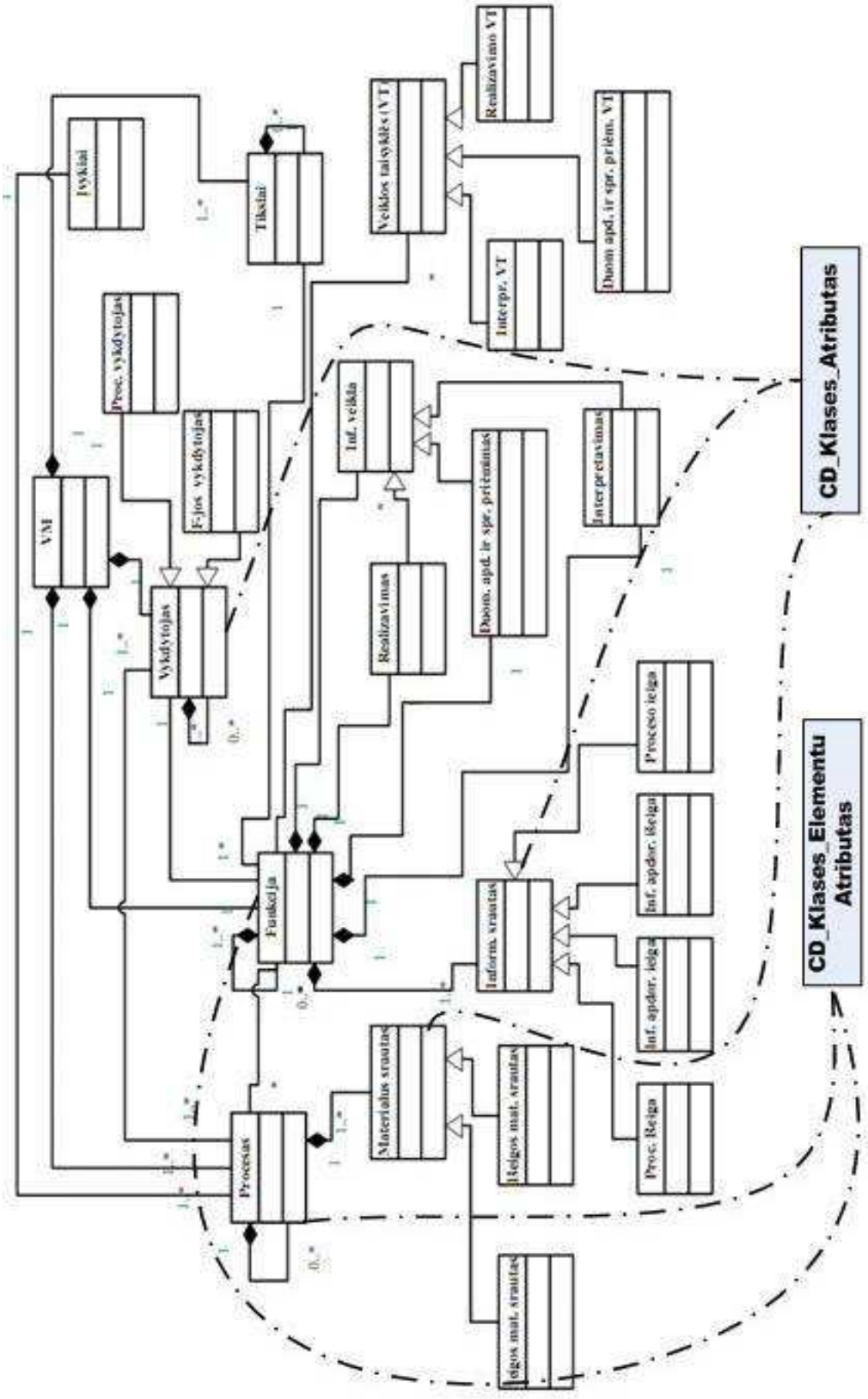
Reikia pabrėžti, jog detalios klasių diagramos gali būti panaudotos programinio kodo generavimui. Efektyviausias būdas realizuoti šią galimybę, yra sugeneruotą klasių diagramą užrašyti formatu, kurį supranta CASE sistema, turinti kodo generavimo funkcionalumą. Konkrečiu atveju iš veiklos modelio domenų bazės galima sugeneruotą klasių diagramą įrašyti į pvz. „MagicDraw“ (viena iš populiariausių CASE priemonių skirtų IS modeliavimui/projektavimui, kurioje yra integruota kodo generavimo galimybė) failą (XML formatas), o pastarasis CASE įrankis turi funkcionalumą generuoti programinį kodą.

Žemiau pateiktame paveiksle pavaizduoti trūkstami veiklos metamodelio elementai. CD_Klases_Atributas skirtas saugoti informaciją apie statinius elementus t.y. aktorius bei informacinius/materialius srautus. Elementas saugo duomenis apie objekto abstraktumą, prieinamumą bei ryšius su kitais elementais. CD_Klases_Elementu_Atributas saugo trūkstama informaciją apie dinaminius elementus t.y. procesus ir funkcijas. Saugomas abstraktumas ir prieinamumas.



Šaltinis: Sudaryta autoriaus

14. Pav. Trūkstami veiklos metamodelio elementai

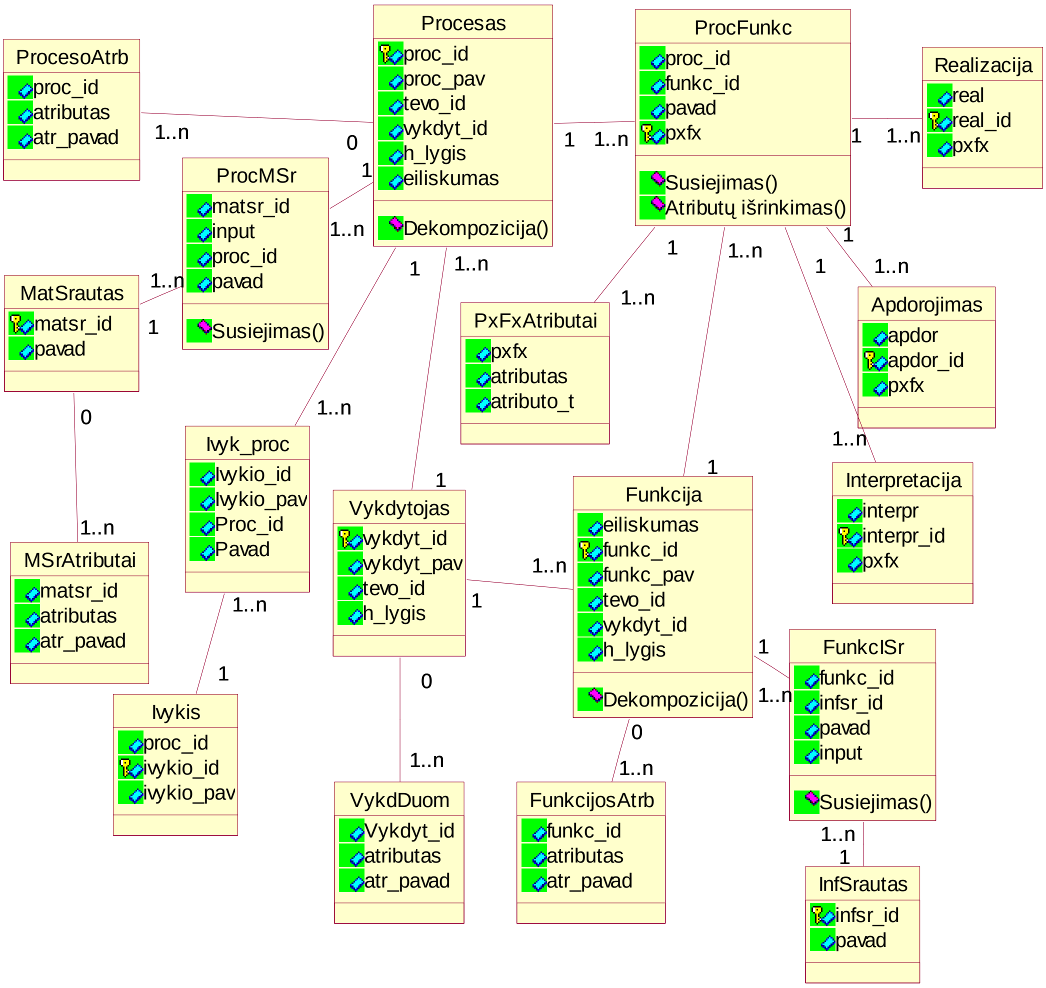


Šaltinis: Sudaryta autoriaus

15. Pav. Papildytas veiklos metamodelis

2.3. EMM duomenų bazės patobulinimas

Sudarinėjant eksperimentinę EMM duomenų bazę, pradinė duomenų bazės loginė struktūra buvo pakeista, kadangi buvo pastebėti tam tikri trūkumai. Prieš pradedant realizuoti duomenų bazę fiziškai, šie trūkumai buvo ištaisyti. Pagrindinis pakeitimas buvo susijęs su duomenų bazės elementus (esybes) aprašančių atributų priskyrimo schema.

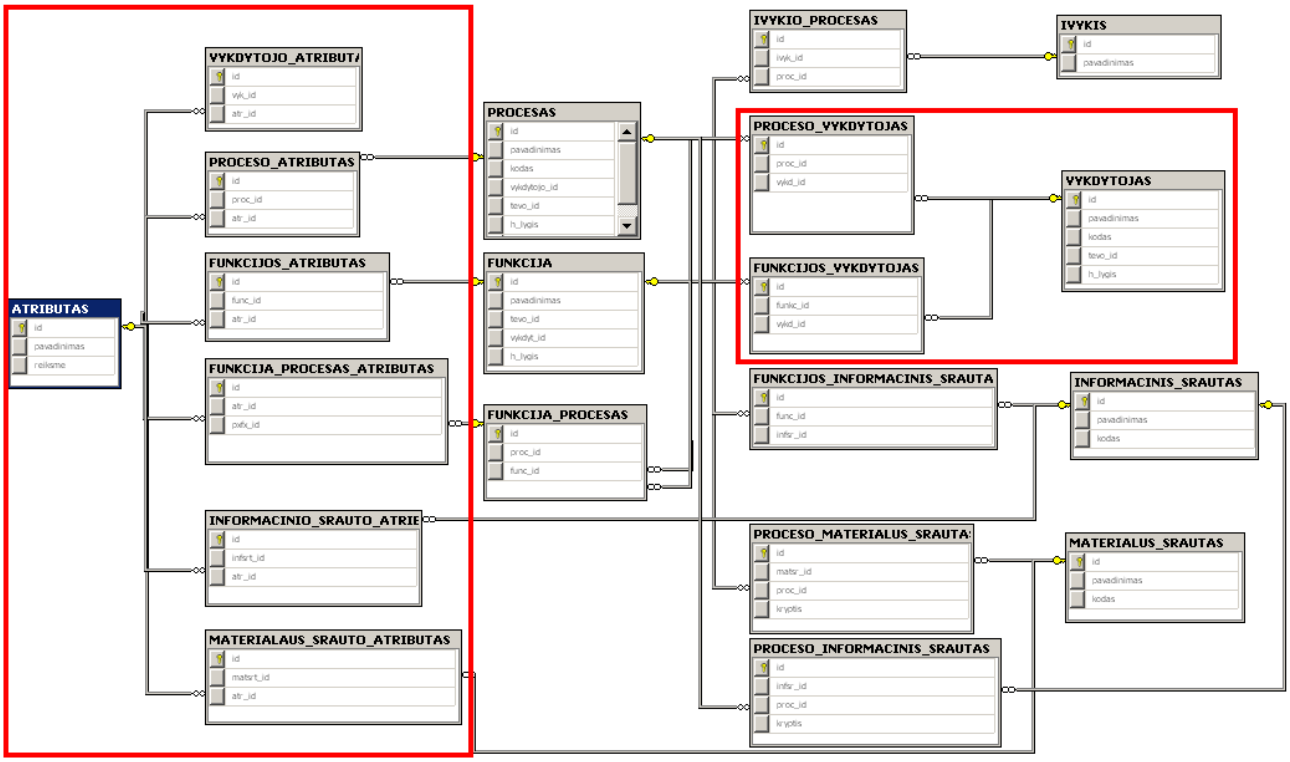


Šaltinis: Sudaryta A.Lopata

17. Pav. Žinių duomenų bazės loginė struktūra

Pradinėje loginės struktūros versijoje buvo kelios lentelės, kurios iš principo aprašė tokią pačią duomenų struktūrą(„ProcesoAtrb“, „FunkcijosAtrbt“). Tai susiję su tuo, jog pradiniame duomenų bazės modelyje atributai buvo naudojami tik su kai kuriais metamodelio elementais („Procesas“, „Funkcija“). Savo ruožtu kiti elementai (pvz. „Vykdytojas“) turėjo papildomus elementus, kurie atliko analogišką funkciją („VykdDuom“). Siekiant optimizuoti ER modelį buvo išskirta nauja esybė „Atributas“, kurioje buvo saugomos metamodelio elementų atributų reikšmės. Elementai su atributais yra susiejami per

tarpinę lentelę, kadangi kitu atveju turėtume ryšio tipą „daug du daug“. Reikia pabrėžti, jog atliktas pakeitimas leis lengviau ir paprasčiau ateityje praplėsti duomenų bazę naujais elementai bei priskirti jiems atitinkamus atributus.



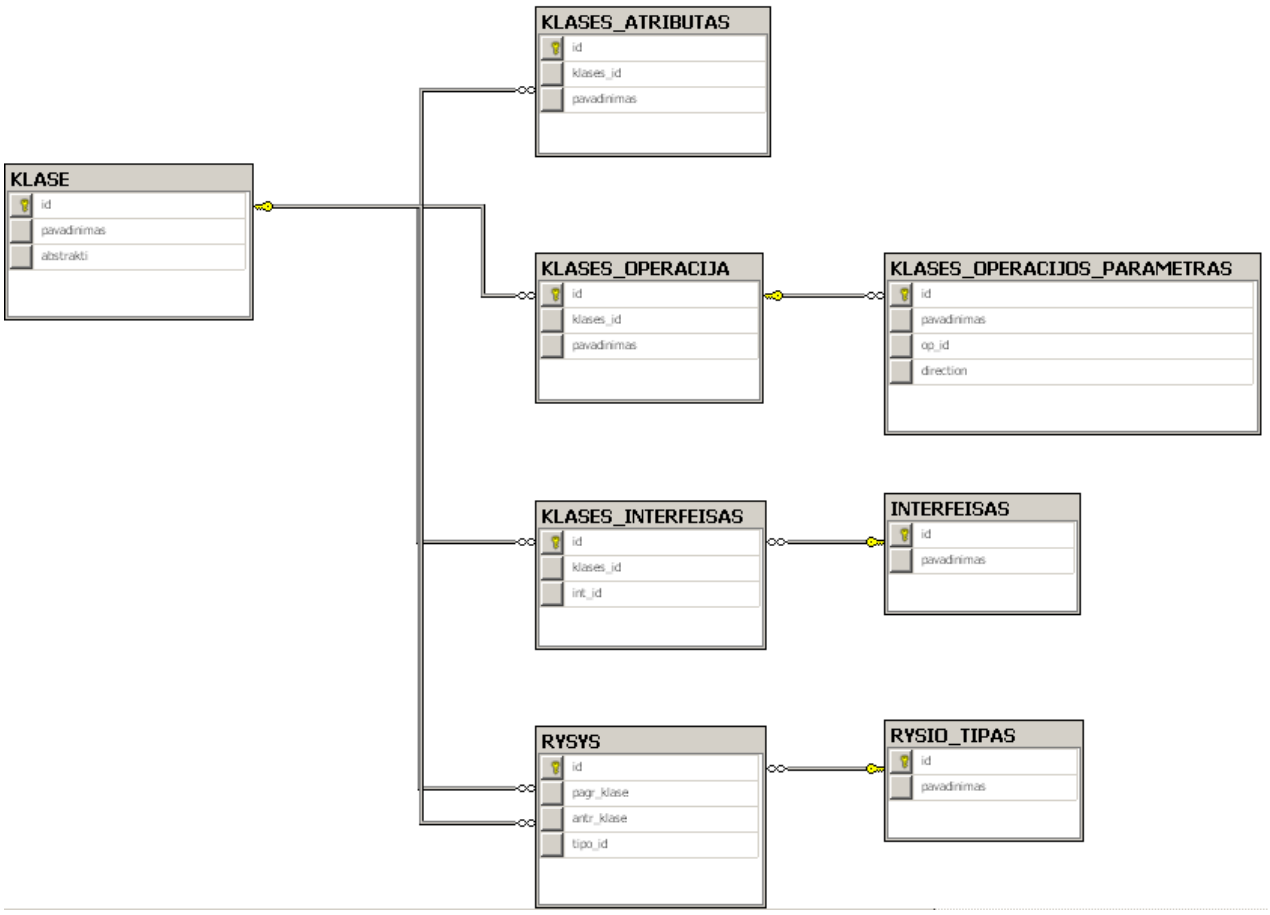
Šaltinis: Sudaryta autoriaus

18. Pav. Atnaujintos duomenų bazės loginė struktūra

Siekiant numatyti išimtinius atvejus buvo nuspręsta pakeisti dar vieną loginės struktūros dalį, konkrečiai vykdytojo priskyrimą procesui arba funkcijai. Priskyrimas yra atliekamas panaudojant tarpinę lentelę t.y. tokiu atveju procesas/funkcija gali turėti kelis vykdytojus. Tokia situacija gali susidaryti modeliuojant skirtingus organizacijos hierarchijos lygius, taigi susidūrus su tokia situacija nereikėtų modifikuoti duomenų bazės struktūros. Norint atlikti eksperimentus (remiantis EMM generuoti klasių diagramas) buvo būtina sukurti fizinę klasių metamodelio duomenų struktūros realizaciją. Klasių diagramos pagrindinis elementas yra esybė „Klasė“. Kiekviena klasė gali turėti 0..n atributų bei 0..n operacijų. Kiekviena operacija (metodas) gali turėti 0..n parametrų, kurie gali būti dviejų tipų (krypties): „in“ bei „out“. „In“ tipas yra perduodamas metodui, kaip išankstinės sąlygos/duomenys, “out“ kryptis apibūdina parametrą, kuris yra metodo veiklos rezultatas.

Klasė gali paveldėti 0..n tėvinių klasių bei realizuoti 0..n interfeisų. Būtina pabrėžti, jog paveldėjimo specifika yra priklausoma nuo konkrečios programavimo kalbos (pvz. C# neleidžia daugialypio paveldėjimo, o JAVA leidžia). Sudarant klasių diagramos metamodelį buvo vadovaujamasi

OMG (Object Management Group) standartais, kurie leidžia daugialypį paveldėjimą. Paveldėjimas yra vienas iš keturių apsibrėžtų ryšių tipų. Kiekvienas ryšys sujungia dvi klases. Šiuo atveju turime pirminę (pagrindinę) klasę bei antrinę. Pirminė klasė yra pvz. paveldėjimo atveju tėvinė klasė, antrinė savo ruožtu – vaiko klasė. Interfeiso realizavimas yra įgyvendinamas atskiros lentelės pagalba, kadangi interfeisas savo prasme yra kitokios prigimties objektas nei klasė. Jis apibrėžia sąsajas, kurias privalo turėti klasė.



Šaltinis: Sudaryta autoriaus

19. Pav. Klasių diagramos duomenų bazės loginė struktūra

Savo prasme klasė yra visuma atributų bei metodų ir verslo logikos sluoksnyje egzistuoja kaip vienetas, tačiau duomenų bazėje ši duomenų struktūra yra saugoja kaip aibė lentelių bei ryšių (kiekviena lentelė turi raktinį lauką, kurio pagalba vienareikšmiškai identifikuojami įrašai bei sudaromi ryšiai) tarp jų. Sukurtas fizinis modelis yra skirtas atlikti klasių diagramos generavimo eksperimentus, identifikuoti teoriškai nenumatytus elementus bei nustatyti kitas galimas problemas.

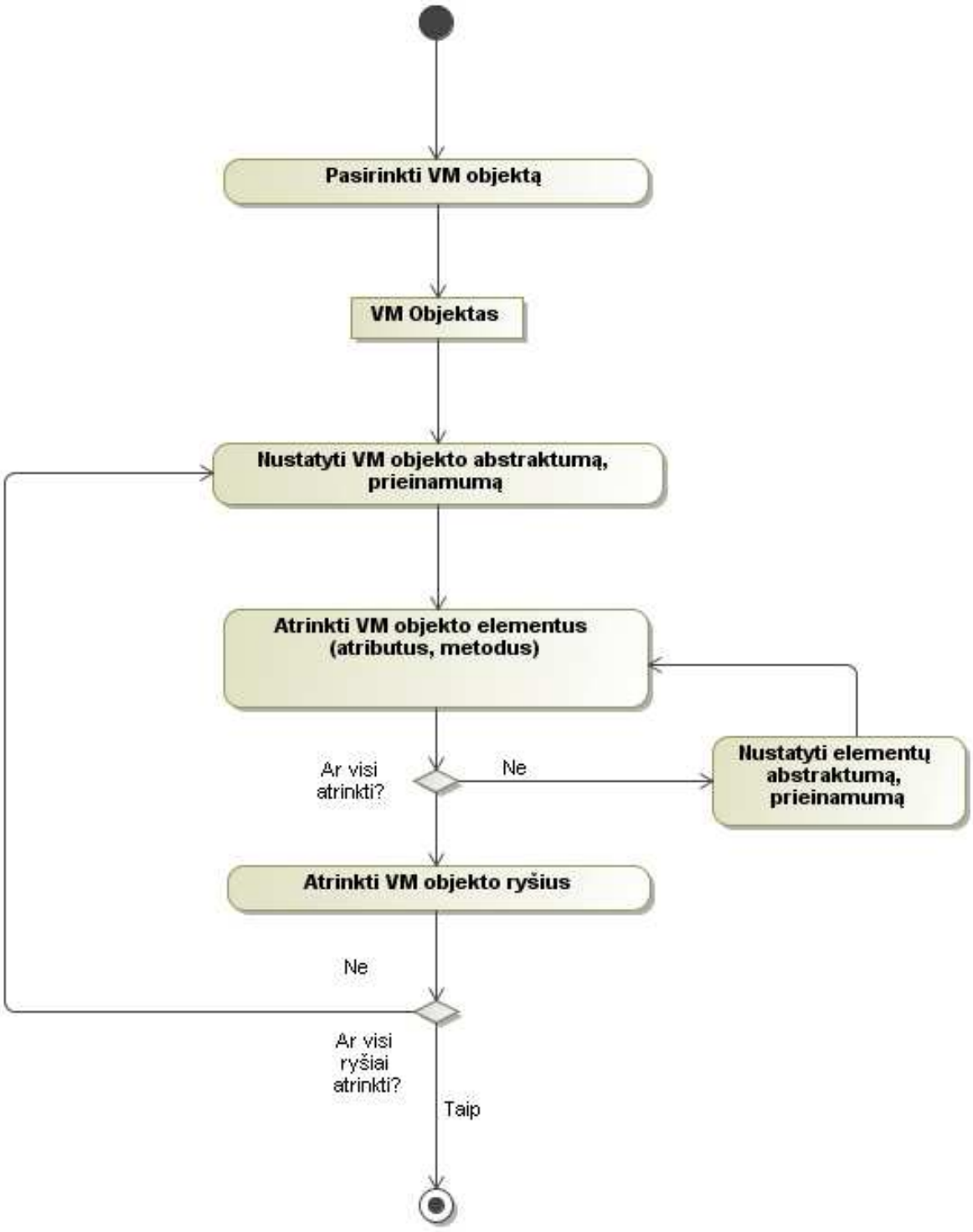
2.3. Klasių diagramos generavimo algoritmas

Pagrindinę klasių diagramos generavimo algoritmo idėją galime apibūdinti taip: algoritmas atrenka atitinkamus elementus (elementus, kurie gali būti atvaizduoti į klasių diagramos objektus) iš veiklos modelio duomenų bazės, pakeičia jų formatą (suformuoja naujus klasių diagramos elementus) ir įrašo į klasių diagramos duomenų bazės lenteles.

Algoritmas atrenka šiuos veiklos modelio elementus: vykdytojus, informacinius bei materialius srautus, procesus, funkcijas. Kartu su šiais algoritmais atrenkami ir jų parametrus apibūdinantys elementai t.y. esybės kuriose saugoma informacija apie ryšius, prieinamumo bei abstraktumo operatorius. Klasių diagramos elementų atrinkimas gali būti įgyvendinamas dviem būdais: atrinkti visus veiklos elementus arba pasirinkti tik tam tikrus veiklos elementus. Pirmuoju atveju visi vykdytojai, srautai, procesai, funkcijos yra atvaizduojami į klasių diagramos elementus. Antruoju atveju atrenkami konkretūs pasirinkti elementai tik su jais susiję (pvz. ryšiais) elementai.
Klasių diagramos generavimo algoritmą galėtume aprašyti šiais žingsniais:

1) Iš duomenų bazės išrenkami aktoriai, informaciniai/materialūs srautai (klasės). Gali būti nurodomi konkretūs objektai, kuriuos reikia atrinkinėti. Kartu su šiais objektais atrenkami ir juos apibūdinantys atributai.;
2) Atrenkami aktorių atliekami procesai/funkcijos arba srautų apdorojime veikiantys procesai /funkcijos(metodai); Kartu su šiais objektais atrenkami ir juos apibūdinantys atributai.;
3) Jeigu atrinktos funkcijos/procesai turi įėjimo/išėjimo srautus, minėti srautai atrenkami ir atvaizduojami kaip parametrai;
4) Sukuriamas naujas klasės objektas bei užpildomas atitinkamais duomenimis;
5) Atrenkami ryšiai, kuriuose dalyvauja klasė, sukuriamas ryšio objektas;
6) Sukurti klasių diagramos elementai išskaidomas į duomenų bazės lenteles ir įrašomas į DB;
7) Sukurti klasių diagramos elementai užrašomi XML formatu;
8) Veiksmai kartojami, kol atvaizduojami visi susiję veiklos modelio elementai.

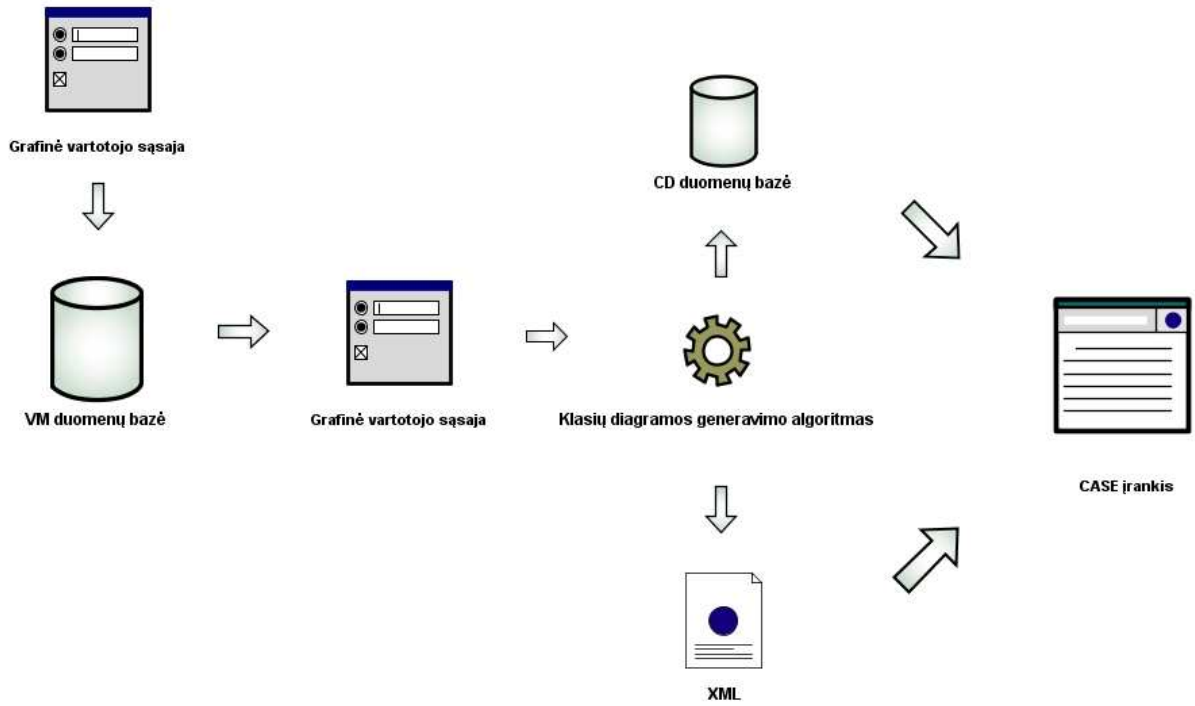
Reikia pabrėžti, jog realiai turi būti numatytas susijusių elementų atrinkimo gylis, kadangi gali susidaryti situacija, kai šis rekurentinis procesas ims generuoti itin dideles diagramas, kas iš praktinio požiūrio nėra naudinga. Tiesa, šis sprendimas turi priklausyti konkrečiai nuo situacijos. Algoritmo principinė schema pateikta 20 pav.



Šaltinis: Sudaryta autoriaus

20. Pav. Klasių diagramos generavimo algoritmas

Nors pats algoritmas nusako generavimo principus, tačiau realiai jis turi būti integruotas į kitą sistemą, kadangi generavimui reikalingi duomenys gali būti gaunami tik su veiklos modelio duomenų bazę užpildančia sistema. Klasių diagramos atvaizdavimas grafiškai bei kodo generavimas yra atliekamas panaudojant modeliavimo CASE įrankius, kurie duomenis gautų iš generavimo algoritmo. Paveiksle 21 yra pateikta klasių diagramos generavimo algoritmo vieta IS inžinerijoje.



Šaltinis: Sudaryta autoriaus

21. Pav. Klasių diagramos generavimo algoritmo vieta IS inžinerijoje

3. Eksperimentas

Siekiant nustatyti trūkstamus EMM elementus, teorinių prielaidų nepakako, kadangi visų galimų variantų dėl trūkstamų klasių diagramos elementų, nebuvo įmanoma iš anksto tiksliai numatyti. Taigi praktinis eksperimentas buvo labai pageidautinas. Jo metu buvo atliekama Case Study t.y. modeliuojama reali situacija.

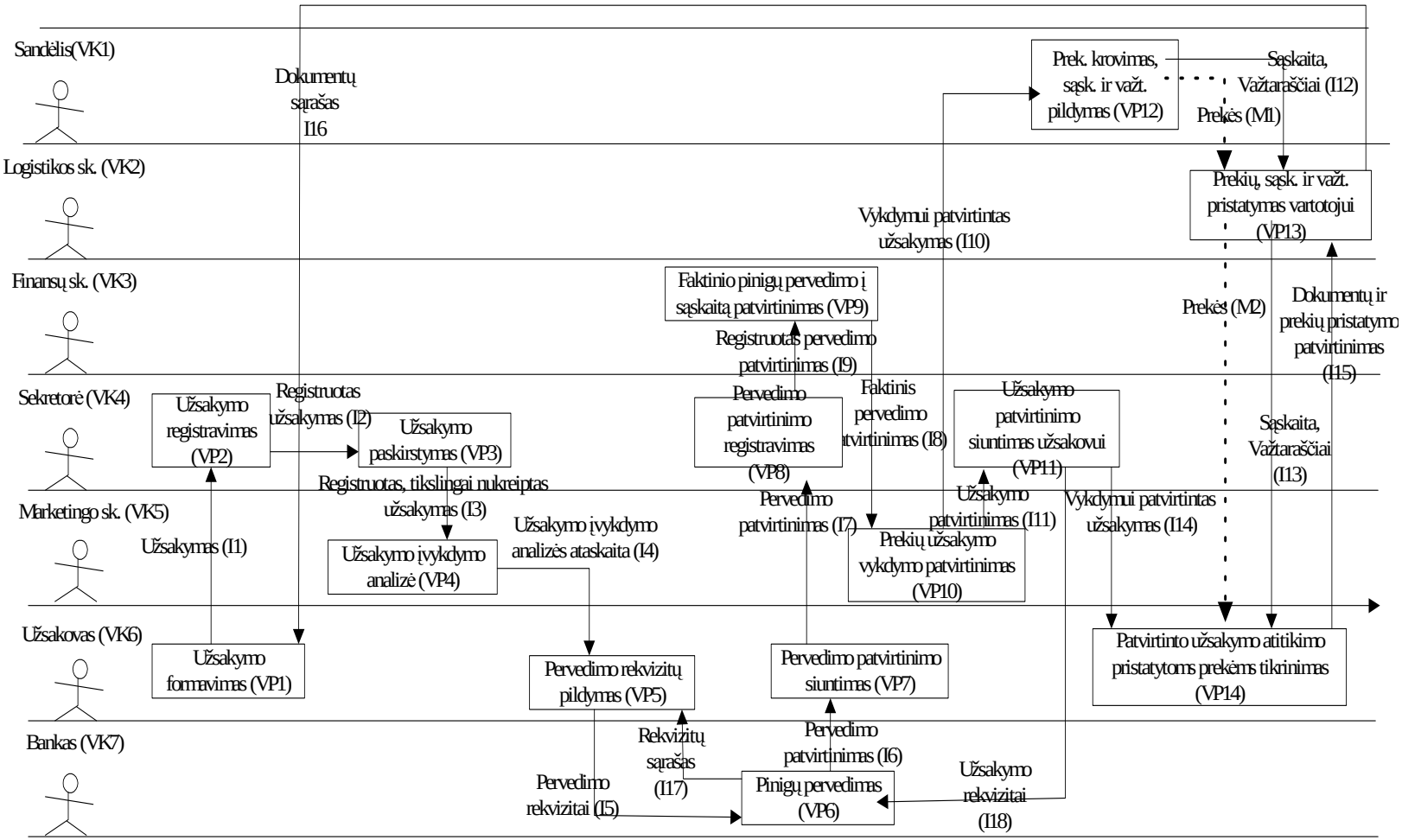
Eksperimento metu realios probleminės srities duomenys buvo suvesti į veiklos modelio duomenų bazę. Duomenys buvo pateikti WorkFlow diagrama, kuri apėmė prekių užsakymo/pristatymo veiklą. Su šiais duomenimis buvo atliekamas klasių diagramos generavimas. Pradiniame etape modeliai buvo generuojami pagal teoriškai suformuluotas sąsajas. Vėliau atsižvelgus į pastebėtus trūkumus veiklos modelio bei klasių diagramos buvo papildytos naujais elementais. Taip pat buvo atnaujintas generavimo algoritmas. Pats generavimo procesas arba algoritmas apėmė EMM elementų susiejimą/konvertavimą į klasių diagramos elementus.

Generavimas buvo atliekamas automatiškai, tai reiškia, jog generatoriaus prototipas buvo realizuotas praktiškai. Eksperimentui atlikti buvo pasirinktos keli skirtingas funkcijas realizuojantys įrankiai, programavimo darbai buvo atliekami su .Net Framework C#, duomenų bazėms sukurti buvo pasirinkta „MS SQLServer“ DBVS, kurti grafiniams modeliams buvo naudojamas modeliavimo įrankis „MagicDraw“, XML failams analizuoti buvo naudojamas Altova XMLSpy redaktorius. Apibendrinant galima pasakyti, jog eksperimento įgyvendinimui buvo naudojamos sekančios technologijos:

- .Net Framework C#
- IDE Microsoft Visual Studio 2005.
- Microsoft SqlServer 2005.
- MagicDraw 12.0.
- Altova XMLSpy

Reikia pabrėžti, jog šio darbo rezultatas yra nepriklausomas nuo platformos t.y. eksperimentas buvo atliekamas su konkrečiomis technologijomis, tačiau praktiškai sistema gali būti realizuojama ir su kitais įrankiais. Šios technologijos buvo pasirinktos atsižvelgiant į pakankamumą įgyvendinti eksperimentą bei asmenines žinias ir patirtį dirbant su jomis.

Eksperimento atlikimo metu buvo laikomasi prielaidos, jog WorkFlow aprašantys organizaciją jau yra surinkti. Testinis atvejis apima organizacijos užsakymo įvykdymą. Šio proceso esmė - užsakovas padaro užsakymą tam tikroms prekėms (materialus srautas), užsakymo įgyvendinimo etapai yra aprašyti darbų sekos diagrama.



Sudaryta: A.Lopata

22. Pav. Užsakymo veiklos procesas

Diagramos pagalba yra specifikuojami šie veiklos procesai: Užsakymo formavimas (VP1), Užsakymo registravimas (VP2), Užsakymo paskirstymas (VP3), Užsakymo įvykdymo analizė (VP4), Pavedimo rekvizitų pildymas (VP5), Pinigų pervedimas (VP6), Pervedimo patvirtinimo siuntimas (VP7), Pervedimo patvirtinimo registravimas (VP8), Faktinio pinigų pervedimo į sąskaitą patvirtinimas (VP9), Prekių užsakymo vykdymo patvirtinimas (VP10), Užsakymo patvirtinimo siuntimas užsakovui (VP11), Prekių krovimas, sąskaitų ir važtaraščių pildymas (VP12), Prekių, sąskaitų ir važtaraščių pristatymas vartotojui (VP13), Patvirtinto užsakymo atitikimo pristatytoms prekėms tikrinimas (VP14). Patys savaime veiklos procesai suteikia mažai informacijos. Svarbūs yra juos siejantys ryšiai – materialūs bei informaciniai. Materialūs ryšiai apima bet kokius materialius srautus – prekes, žaliavas, pusgaminius etc. Informaciniai ryšiai apima ataskaitas, paklausimus, įsakymus, važtaraščius ir pan. Nagrinėjamoje darbų sekos diagramoje yra išskirti sekantys ryšiai: materialūs (Prekės (M1), Prekės (M2)) bei informaciniai: Užsakymas (I1), Registruotas užsakymas (I2), Registruotas, tikslingai nukreiptas užsakymas (I3), Užsakymo įvykdymo analizės ataskaita (I4), Pervedimo rekvizitai (I5), Pervedimo patvirtinimas (I6), Pervedimo patvirtinimas (I7), Faktinis pervedimo patvirtinimas (I8), Registruotas pervedimo patvirtinimas (I9), Vykdymui patvirtintas užsakymas (I10), Užsakymo patvirtinimas (I11), Sąskaita, važtaraščiai (I12, I13), Vykdymui patvirtintas užsakymas (I14), Dokumentų ir prekių pristatymo patvirtinimas (I15), Dokumentų sąrašas (I16), Rekvizitų sąrašas (I17), Užsakymo rekvizitai (I18). Klasių objektų pagrindu yra veiklos dalykinėje srityje egzistuojantys aktoriai (vykdytojai), kurių šiuo atveju yra išskiriama septyni Sandėlis (VK1), Logistikos sk. (VK2), Finansų sk. (VK3), Sekretorė (VK4), Marketingo sk. (VK5), Užsakovas (VK6) ir Bankas (VK7).

4. Lentelė

Procesai

Usakymo formavimas	VP1
Usakymo registravimas	VP2
Usakymo paskirstymas	VP3
Usakymo ivykdymo analize	VP4
Pavedimo rekvizitu pildymas	VP5
Pinigu pervedimas	VP6
Pervedimo patvirtinimo siuntimas	VP7
Pervedimo patvirtinimo registravimas	VP8
Faktinio pinigu pervedimo i saskaita patvirtinimas	VP9
Prekiu usakymo vykdymo patvirtinimas	VP10
Usakymo patvirtinimo siuntimas usakovui	VP11
Prekiu krovimas, saskaitu ir vataraciu pildymas	VP12
Prekiu, sask., vataraciu pristatymas vartotojui	VP13
Patvirtinto usakymo atitikimo prekems tikrinimas	VP14

5. Lentelė

Aktoriai(vykdytojai)

Sandelis	VK1
Logistikos sk.	VK2
Finansu sk.	VK3
Sekretore	VK4
Marketingo sk.	VK5
Usakovas	VK6
Bankas	VK7

6. Lentelė

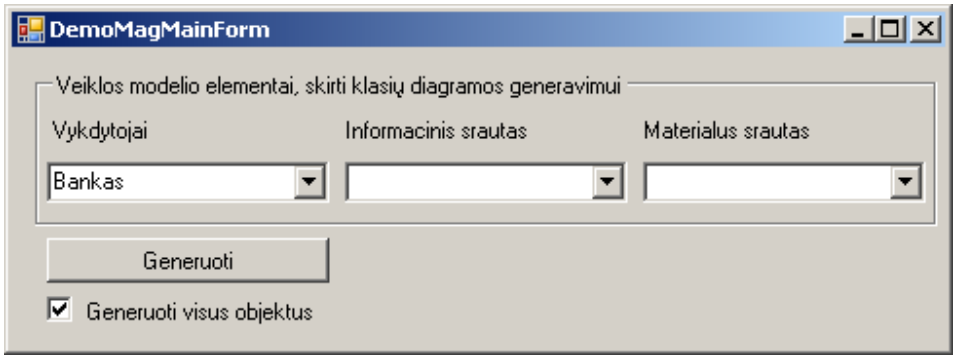
Informaciniai bei materialūs srautai

Usakymas	I1
Registruotas usakymas	I2
Registruotas, tikslingai nukreiptas usakymas	I3
Usakymo ivykdymo analizes ataskaita	I4
Pervedimo rekvizitai	I5
Pervedimo patvirtinimasI6	I6
Pervedimo patvirtinimasI7	I7
Faktinis pervedimo patvirtinimas	I8
Registruotas pervedimo patvirtinimas	I9
Vykdymui patvirtintas usakymas	I10
Usakymo patvirtinimas	I11
Saskaita	I12
Vataraciai	I13
Vykdymui patvirtintas usakymas	I14
Dokumentu ir prekiu pristatymo patvirtinimas	I15
Dokumentu saraas	I16
Rekvizitu saraas	I17
Usakymo rekvizitai	I18
Prekes1	M1
Prekes2	M2

3.1. Eksperimento eiga

Eksperimentui atlikti buvo programiškai realizuotas klasių diagramos generavimo algoritmas, sukurta veiklos modelio bei klasių diagramos duomenų bazės. Teoriškai šią užduotį buvo galima realizuoti ir nenaudojant programavimo, tačiau tokiu būdu būtų ženkliai apribota eksperimento apimtis be to pakeitimų bei papildomų eksperimento duomenų įvedimas ir įvertinimas būtų sunkiai realizuojamas ir imlus laikui.

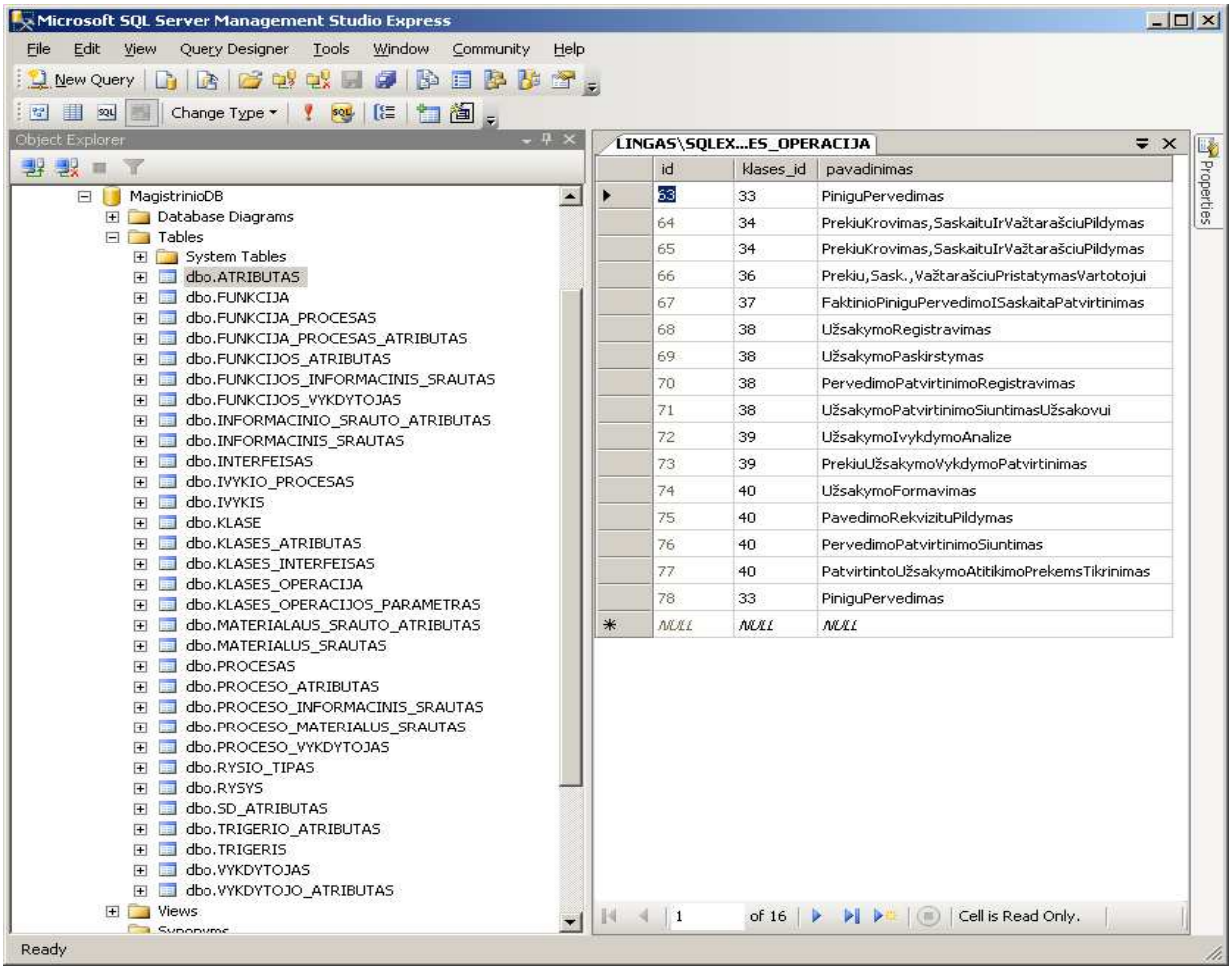
Nors teoriškai duomenys aprašantys organizaciją turėtų būti įvesti į duomenų bazę, tačiau konkrečiu atveju WorkFlow diagrama aprašyta organizacijos veikla buvo suvedinėjama į duomenų bazę eksperimento metu.



Šaltinis: sudaryta autoriaus

23. Pav. Generavimo programos langas

Atlikus klasių diagramos generavimą, buvo sukurti aktorius bei srautus reprezentuojantys objektai. Procesai ir funkcijos buvo atvaizduoti į metodus. Vykdant eksperimentą buvo pastebėtas vienas WorkFlow diagramos trūkumas, kurio sprendimas turėtų priklausyti konkrečiai nuo organizacijos pasirinktos vardų suteikimo procedūros (naming convention) t.y. procesas ar srautas buvo aprašomas kelių žodžių fraze pvz.: „Pavedimo rekvizitų pildymas“, „Užsakymo įvykdymo analizės ataskaita“, tai sąlygojo, kad sukurta klasė pirminiu savo variantu nebuvo tinkamai užrašyta kodo generavimo atžvilgiu, kadangi jos pačios/metodų/parametrų pavadinimai buvo per ilgi.



Šaltinis: Sudaryta autoriaus

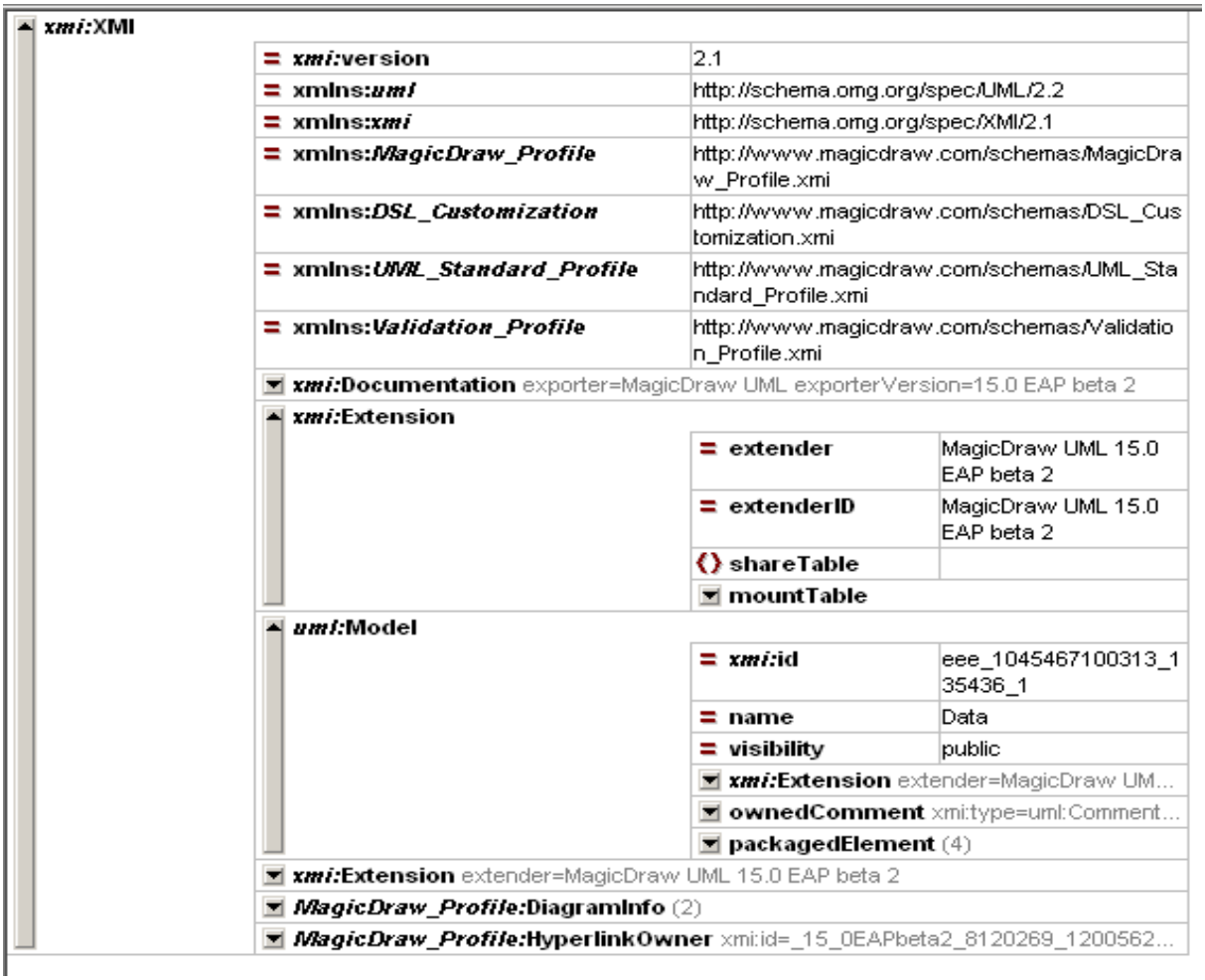
24. Pav. Klasių operacijų (metodų) lentelė

```
<Class>Užsakovas
  <Metodas>UžsakymoFormavimas
    <Parametras>Užsakymas<Kryptis>Out</Kryptis></Parametras>
    <Parametras>DokumentuSarašas<Kryptis>In</Kryptis></Parametras>
  </Metodas>
  <Metodas>PavedimoRekvizituPildymas
    <Parametras>RekvizituSarašas<Kryptis>In</Kryptis></Parametras>
    <Parametras>PervedimoRekvizitai<Kryptis>Out</Kryptis></Parametras>
    <Parametras>UžsakymoIvykdymoAnalizesAtaskaita<Kryptis>In</Kryptis></Parametras>
  </Metodas>
  <Metodas>PervedimoPatvirtinimoSiuntimas
    <Parametras>PervedimoPatvirtinimasI6<Kryptis>In</Kryptis></Parametras>
    <Parametras>PervedimoPatvirtinimasI7<Kryptis>Out</Kryptis></Parametras>
  </Metodas>
  <Metodas>PatvirtintoUžsakymoAtitikimoPrekemsTikrinimas
    <Parametras>VykdymuiPatvirtintasUžsakymas<Kryptis>In</Kryptis></Parametras>
    <Parametras>Važtarašciai<Kryptis>In</Kryptis></Parametras>
    <Parametras>DokumentuIrPrekiuPristatymoPatvirtinimas<Kryptis>Out</Kryptis></Parametras>
  </Metodas>
</Class>
```

Šaltinis: Sudaryta autoriaus

25. Pav. Autoriaus sukurto XML failo struktūra

Eksperimento metu sugeneruoti klasių diagramos elementai buvo įrašomi į duomenų bazę bei į autoriaus sukurtą XML duomenų failą. Taikant algoritmą konkrečiai CASE sistemai, rezultatų išvedimo formatas priklausytų būtent nuo šios sistemos. Natūralu, jog tai turėtų būti XMI t.y. OMG sukurtas UML modelių užrašymo standartas XML kalba. Modelius XMI užrašo pvz. modeliavimo paketas „MagicDraw“. Konkrečiu atveju atlikus nedidelę XMI struktūros analizę, buvo nustatyta, jog informacija apie elementus yra saugoma „packagedElement“ mazge.



Šaltinis: sudaryta autoriaus

26. Pav. „MagicDraw“ duomenų failo struktūra (XMI)

Išvados

Atlikus esamos padėties IS inžinerijoje analizę, buvo nustatyta, jog einamuoju metu yra ketvirtasis IS inžinerijos etapas, kuriame vyrauja *žiniomis grįstos* IS inžinerijos kryptis. Pastaroji savyje apjungia tradicinės IS inžinerijos, veiklos modeliavimo bei dirbtinio intelekto principus.

Nagrinėjant įvairius veiklos metamodelius, nustatyta, kad Lietuvos sąlygomis tikslinga naudoti KTU informatikos fakulteto Informacijos sistemų katedroje sukurta teorinį modelį, kuriame yra įsisavintos UEML praktikos.

Iki šiol minėtasis metamodelis buvo pritaikytas reikalavimų surinkimo bei atvaizdavimo („Use Case“ diagramos) etapui, todėl išplėtimas ir pritaikymas klasių diagramų generavimui, reiškia kokybiškai naują pritaikymo perspektyvą.

Išanalizavus veiklos metamodelį buvo nustatyta, jog jame yra nepakankamas žinių kiekis, siekiant generuoti UML 2.0 klasių diagramas, kadangi dabartinis metamodelis nenumato galimybės klasėms suteikti prieinamumo atributus, abstraktumo lygius bei susieti ryšiais, taigi metamodelį būtina papildyti elementais, kurie saugotų minėtą trūkstamą informaciją.

Pagal analizės duomenis buvo sukurtas algoritmas įgalinantis atlikti klasių diagramos generavimą iš veiklos modelio duomenų bazės. Generavimas atliekamas pagal pasirinktus veiklos modelio elementus.

Sukurtas algoritmas buvo išbandytas su testiniais duomenimis, kurie patvirtino jog papildžius veiklos metamodelį naujais elementais, klasių diagramos generavimas tampa įmanomas.

Automatinis klasių diagramos generavimas, remiantis veiklos metamodeliu, intelektualizuoja projektavimo etapą, kadangi leidžia išvengti projektavimo klaidų dėl specifikacijų nepilnumo bei žmogiškojo faktoriaus.

Darbas buvo pristatytas tarpuniversitetinėje magistrantų ir doktorantų konferencijoje “Informacinės technologijos’08”.

LITERATŪRA

1. *Unified Modeling Language: Superstructure, version 2.0*, formal/05-07-04. (2005). August 2005 [žiūrėta 2006 12 08]. Puslapio nuoroda: http://www.omg.org/docs/formal/05-07-04.pdf

2. Lopata A.; Paradauskaitė V. *Informacinės technologijos 2006*, Vilnius: Vilniaus universiteto leidykla 2006, p. 72-77

3. Lopata A.;Gudas S. *Informacijos mokslai T30*,Vilnius:Vilniaus universiteto leidykla,2004, p. 90

4. "Object Management Group" internetinė svetainė [žiūrėta 2006 11 26]. Puslapio nuoroda: http://www.uml.org/

5. CIMOSA - European Enterprise Integration Concept [žiūrėta 2006 11 26]. Puslapio nuoroda: http://pera.net/Arc_cimosa.html

6. Šilingas D, "*Best Practices for Applying UML, Part I*" [žiūrėta 2006 10 12]. Dokumento nuoroda: https://secure.nomagic.com/files/whitepapers/Best_Practices_for_Applying_UML_Part1.pdf

7. Vernadat F. *Unified Enterprise Modelling Language*. [interaktyvus]. [žiūrėta 2007 03 10]. Prieiga per internetą: http://www.cit.gu.edu.au/~bernus/taskforce/archive/UEML-TF-IG.ppt

8. UEML. [interaktyvus]. [žiūrėta 2007 04 10]. Prieiga per internetą: http://athena.troux.com/akmii/Default.aspx?WebID=249

9. Williams T. CIMOSA - European Enterprise Integration Concept. [interaktyvus]. [žiūrėta 2007 04 12]. Prieiga per internetą: http://pera.net/Arc_cimosa.html

10. Chitnis M.; Tiwari P., Ananthamurthy L. *UML Tools*. [interaktyvus]. [žiūrėta 2007 04 10]. Prieiga per internetą: http://www.developer.com/design/article.php/1593811

11. Lopata A.; Gudas S. *Vartotojo reikalavimų modelių sudarymas CASE žinių saugyklos pagrindu Informacijos mokslai T36*, Vilnius: Vilniaus universiteto leidykla, 2006 p. 127-138.

12. UML Forum. [interaktyvus]. [žiūrėta 2007 04 10]. Prieiga per internetą: http://www.uml-forum.com/tools.htm

13. Lopata A.; Gudas S. Darbų sekų modeliais grindžiamas veiklos žinių surinkimo būdas *Informacijos mokslai T34*, Vilnius: Vilniaus universiteto leidykla, 2005, p. 277-287.

14. Skersys T. *Informacijos sistemų kompiuterizuoto kūrimo metodas, grindžiamas veiklos taisyklėmis papildytu veiklos modeliu*. Kaunas: Technologija, 2006

15. Danielaitytė D. *UML veiklos modelio generavimas veiklos žinių saugyklos pagrindu*. Magistro darbas, KTU.

PRIEDAI

```
USE [MagistrinioDB]
GO
/****** Object:  Table [dbo].[IVYKIS]    Script Date: 06/02/2008 11:05:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[IVYKIS](
	[id] [int] NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
 CONSTRAINT [PK_IVYKIS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[FUNKCIJA]    Script Date: 06/02/2008 11:04:43 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[FUNKCIJA](
	[id] [int] NOT NULL,
	[pavadinimas] [varbinary](50) NULL,
	[tevo_id] [int] NULL,
	[vykdyt_id] [int] NULL,
	[h_lygis] [int] NULL,
 CONSTRAINT [PK_FUNKCIJA] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[MATERIALUS_SRAUTAS]    Script Date: 06/02/2008 11:05:13 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[MATERIALUS_SRAUTAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[pavadinimas] [varchar](50) NULL,
	[kodas] [varchar](20) NULL,
 CONSTRAINT [PK_MATERIALUS_SRAUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[INFORMACINIS_SRAUTAS]    Script Date: 06/02/2008 11:04:56 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
```

```
GO
CREATE TABLE [dbo].[INFORMACINIS_SRAUTAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[pavadinimas] [varchar](50) NULL,
	[kodas] [varchar](20) NULL,
 CONSTRAINT [PK_INFORMACINIS_SRAUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[VYKDYTOJAS]    Script Date: 06/02/2008 11:05:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[VYKDYTOJAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[pavadinimas] [varchar](50) NULL,
	[kodas] [varchar](20) NULL,
	[tevo_id] [int] NULL,
	[h_lygis] [int] NULL,
 CONSTRAINT [PK_VYKDYTOJAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[PROCESAS]    Script Date: 06/02/2008 11:05:16 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[PROCESAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[pavadinimas] [varchar](100) NULL,
	[kodas] [varchar](20) NULL,
	[vykdytojo_id] [int] NULL,
	[tevo_id] [int] NULL,
	[h_lygis] [int] NULL,
	[eiliskumas] [varchar](50) NULL,
 CONSTRAINT [PK_PROCESAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[KLASE]    Script Date: 06/02/2008 11:05:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[KLASE](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
	[abstrakti] [bit] NOT NULL,
 CONSTRAINT [PK_KLASE] PRIMARY KEY CLUSTERED
(
```

```
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[RYSIO_TIPAS]    Script Date: 06/02/2008 11:05:26 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[RYSIO_TIPAS](
	[id] [int] NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
 CONSTRAINT [PK_RYSIO_TIPAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[TRIGERIS]    Script Date: 06/02/2008 11:05:33 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[TRIGERIS](
	[id] [int] NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
 CONSTRAINT [PK_TRIGERIS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[SD_ATRIBUTAS]    Script Date: 06/02/2008 11:05:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[SD_ATRIBUTAS](
	[id] [int] NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
	[reiksme] [varchar](50) NOT NULL,
 CONSTRAINT [PK_SD_ATRIBUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[INTERFEISAS]    Script Date: 06/02/2008 11:04:57 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[INTERFEISAS](
```

```
	[id] [int] NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
 CONSTRAINT [PK_INTERFEISAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[ATRIBUTAS]    Script Date: 06/02/2008 11:04:40 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[ATRIBUTAS](
	[id] [int] NOT NULL,
	[pavadinimas] [varchar](50) NULL,
	[reiksme] [image] NOT NULL,
 CONSTRAINT [PK_ATRIBUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[IVYKIO_PROCESAS]    Script Date: 06/02/2008 11:04:59 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[IVYKIO_PROCESAS](
	[id] [int] NOT NULL,
	[ivyk_id] [int] NULL,
	[proc_id] [int] NULL,
 CONSTRAINT [PK_IVYKIO_PROCESAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FUNKCIJOS_INFORMACINIS_SRAUTAS]    Script Date: 06/02/2008 11:04:50 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FUNKCIJOS_INFORMACINIS_SRAUTAS](
	[id] [int] NOT NULL,
	[func_id] [int] NULL,
	[infsr_id] [int] NULL,
 CONSTRAINT [PK_FUNKCIJOS_INFORMACINIS_SRAUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FUNKCIJOS_ATRIBUTAS]    Script Date: 06/02/2008 11:04:48 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FUNKCIJOS_ATRIBUTAS](
	[id] [int] NOT NULL,
	[func_id] [int] NULL,
	[atr_id] [int] NULL,
 CONSTRAINT [PK_FUNKCIJOS_ATRIBUTAS] PRIMARY KEY CLUSTERED
```

```
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FUNKCIJA_PROCESAS]    Script Date: 06/02/2008 11:04:44 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FUNKCIJA_PROCESAS](
	[id] [int] NOT NULL,
	[proc_id] [int] NOT NULL,
	[func_id] [int] NOT NULL,
 CONSTRAINT [PK_Funkcijos_Procesas] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FUNKCIJOS_VYKDYTOJAS]    Script Date: 06/02/2008 11:04:52 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FUNKCIJOS_VYKDYTOJAS](
	[id] [int] NOT NULL,
	[funkc_id] [int] NOT NULL,
	[vykd_id] [int] NOT NULL,
 CONSTRAINT [PK_FUNKCIJOS_VYKDYTOJAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[PROCESO_MATERIALUS_SRAUTAS]    Script Date: 06/02/2008 11:05:22 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PROCESO_MATERIALUS_SRAUTAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[matsr_id] [int] NOT NULL,
	[proc_id] [int] NOT NULL,
	[kryptis] [bit] NOT NULL,
 CONSTRAINT [PK_PROCESO_MATERIALUS_SRAUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[PROCESO_INFORMACINIS_SRAUTAS]    Script Date: 06/02/2008 11:05:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PROCESO_INFORMACINIS_SRAUTAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[infsr_id] [int] NOT NULL,
	[proc_id] [int] NOT NULL,
	[kryptis] [bit] NOT NULL,
 CONSTRAINT [PK_PROCESO_INFORMACINIS_SRAUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[PROCESO_VYKDYTOJAS]    Script Date: 06/02/2008 11:05:24 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PROCESO_VYKDYTOJAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[proc_id] [int] NOT NULL,
	[vykd_id] [int] NOT NULL,
 CONSTRAINT [PK_PROCESO_VYKDYTOJAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[PROCESO_ATRIBUTAS]    Script Date: 06/02/2008 11:05:18 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PROCESO_ATRIBUTAS](
	[id] [nchar](10) NOT NULL,
	[proc_id] [int] NULL,
	[atr_id] [int] NULL,
 CONSTRAINT [PK_ProcesoAtrb] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[MATERIALAUS_SRAUTO_ATRIBUTAS]    Script Date: 06/02/2008 11:05:11 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MATERIALAUS_SRAUTO_ATRIBUTAS](
	[id] [int] NOT NULL,
	[matsrt_id] [int] NULL,
	[atr_id] [int] NULL,
 CONSTRAINT [PK_MATERIALAUS_SRAUTO_ATRIBUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FUNKCIJA_PROCESAS_ATRIBUTAS]    Script Date: 06/02/2008 11:04:46 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FUNKCIJA_PROCESAS_ATRIBUTAS](
	[id] [int] NOT NULL,
	[atr_id] [int] NOT NULL,
	[pxfx_id] [int] NOT NULL,
 CONSTRAINT [PK_FUNKCIJA_PROCESAS_ATRIBUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[INFORMACINIO_SRAUTO_ATRIBUTAS]    Script Date: 06/02/2008 11:04:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[INFORMACINIO_SRAUTO_ATRIBUTAS](
	[id] [int] NOT NULL,
	[infsrt_id] [int] NOT NULL,
	[atr_id] [int] NOT NULL,
 CONSTRAINT [PK_INFORMACINIO_SRAUTO_ATRIBUTAS] PRIMARY KEY CLUSTERED
```

```
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[KLASES_OPERACIJOS_PARAMETRAS]    Script Date: 06/02/2008 11:05:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[KLASES_OPERACIJOS_PARAMETRAS](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
	[op_id] [int] NOT NULL,
	[direction] [bit] NOT NULL,
 CONSTRAINT [PK_KLASES_OPERACIJOS_PARAMETRAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[KLASES_INTERFEISAS]    Script Date: 06/02/2008 11:05:06 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[KLASES_INTERFEISAS](
	[id] [int] NOT NULL,
	[klases_id] [int] NOT NULL,
	[int_id] [int] NOT NULL,
 CONSTRAINT [PK_KLASES_INTERFEISAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[KLASES_ATRIBUTAS]    Script Date: 06/02/2008 11:05:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[KLASES_ATRIBUTAS](
	[id] [int] NOT NULL,
	[klases_id] [int] NOT NULL,
	[pavadinimas] [varchar](50) NOT NULL,
 CONSTRAINT [PK_KLASES_ATRIBUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[KLASES_OPERACIJA]    Script Date: 06/02/2008 11:05:07 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[KLASES_OPERACIJA](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[klases_id] [int] NOT NULL,
```

```
	[pavadinimas] [varchar](50) NOT NULL,
 CONSTRAINT [PK_KLASES_OPERACIJA] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[RYSYS]    Script Date: 06/02/2008 11:05:28 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[RYSYS](
	[id] [int] NOT NULL,
	[pagr_klase] [int] NOT NULL,
	[antr_klase] [int] NOT NULL,
	[tipo_id] [int] NOT NULL,
 CONSTRAINT [PK_RYSYS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TRIGERIO_ATRIBUTAS]    Script Date: 06/02/2008 11:05:31 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TRIGERIO_ATRIBUTAS](
	[id] [int] NOT NULL,
	[trigerio_id] [int] NOT NULL,
	[atributo_id] [int] NOT NULL,
 CONSTRAINT [PK_TRIGERIO_ATRIBUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[VYKDYTOJO_ATRIBUTAS]    Script Date: 06/02/2008 11:05:37 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[VYKDYTOJO_ATRIBUTAS](
	[id] [int] NOT NULL,
	[vyk_id] [int] NULL,
	[atr_id] [int] NULL,
 CONSTRAINT [PK_VYKDYTOJO_ATRIBUTAS] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  StoredProcedure [dbo].[GetProcesaiByVykdytojas]    Script Date: 06/02/2008 11:04:38 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[GetProcesaiByVykdytojas]
(
	@vykdytojoId int
)
AS
	SET NOCOUNT ON;
SELECT    PROCESAS.pavadinimas, PROCESAS.id
FROM      PROCESAS INNER JOIN
            PROCESO_VYKDYTOJAS ON PROCESAS.id = PROCESO_VYKDYTOJAS.proc_id
```

```
WHERE     (PROCESO_VYKDYTOJAS.id = @vykdytojoId)
GO
/****** Object:  StoredProcedure [dbo].[GetProcesaiByVykdytojoId]    Script Date: 06/02/2008 11:04:38 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[GetProcesaiByVykdytojoId]
(
	@vykdytojas int
)
AS
	SET NOCOUNT ON;
SELECT     PROCESAS.id, PROCESAS.pavadinimas
FROM         PROCESAS INNER JOIN
                      PROCESO_VYKDYTOJAS ON PROCESAS.id = PROCESO_VYKDYTOJAS.proc_id
WHERE     (PROCESO_VYKDYTOJAS.vykd_id = @vykdytojas)
GO
/****** Object:  StoredProcedure [dbo].[GetIdByName]    Script Date: 06/02/2008 11:04:38 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[GetIdByName]
(
	@pavadinimas varchar(50)
)
AS
	SET NOCOUNT ON;
SELECT     id
FROM         KLASE
WHERE     (pavadinimas = @pavadinimas)
GO
/****** Object:  ForeignKey [FK_FUNKCIJA_PROCESAS_FUNKCIJA]    Script Date: 06/02/2008 11:04:45 ******/
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS]  WITH CHECK ADD  CONSTRAINT [FK_FUNKCIJA_PROCESAS_FUNKCIJA] FOREIGN
KEY([func_id])
REFERENCES [dbo].[FUNKCIJA] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS] CHECK CONSTRAINT [FK_FUNKCIJA_PROCESAS_FUNKCIJA]
GO
/****** Object:  ForeignKey [FK_FUNKCIJA_PROCESAS_PROCESAS]    Script Date: 06/02/2008 11:04:45 ******/
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS]  WITH CHECK ADD  CONSTRAINT [FK_FUNKCIJA_PROCESAS_PROCESAS] FOREIGN
KEY([proc_id])
REFERENCES [dbo].[PROCESAS] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS] CHECK CONSTRAINT [FK_FUNKCIJA_PROCESAS_PROCESAS]
GO
/****** Object:  ForeignKey [FK_FUNKCIJA_PROCESAS_ATRIBUTAS_ATRIBUTAS]    Script Date: 06/02/2008 11:04:46 ******/
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_FUNKCIJA_PROCESAS_ATRIBUTAS_ATRIBUTAS] FOREIGN KEY([atr_id])
REFERENCES [dbo].[ATRIBUTAS] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS_ATRIBUTAS] CHECK CONSTRAINT
[FK_FUNKCIJA_PROCESAS_ATRIBUTAS_ATRIBUTAS]
GO
/****** Object:  ForeignKey [FK_FUNKCIJA_PROCESAS_ATRIBUTAS_FUNKCIJA_PROCESAS]    Script Date: 06/02/2008 11:04:47 ******/
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_FUNKCIJA_PROCESAS_ATRIBUTAS_FUNKCIJA_PROCESAS] FOREIGN KEY([pxfx_id])
REFERENCES [dbo].[FUNKCIJA_PROCESAS] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJA_PROCESAS_ATRIBUTAS] CHECK CONSTRAINT
[FK_FUNKCIJA_PROCESAS_ATRIBUTAS_FUNKCIJA_PROCESAS]
GO
/****** Object:  ForeignKey [FK_FUNKCIJOS_ATRIBUTAS_ATRIBUTAS]    Script Date: 06/02/2008 11:04:48 ******/
ALTER TABLE [dbo].[FUNKCIJOS_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_FUNKCIJOS_ATRIBUTAS_ATRIBUTAS]
FOREIGN KEY([atr_id])
REFERENCES [dbo].[ATRIBUTAS] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJOS_ATRIBUTAS] CHECK CONSTRAINT [FK_FUNKCIJOS_ATRIBUTAS_ATRIBUTAS]
GO
/****** Object:  ForeignKey [FK_FUNKCIJOS_ATRIBUTAS_FUNKCIJA]    Script Date: 06/02/2008 11:04:49 ******/
ALTER TABLE [dbo].[FUNKCIJOS_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_FUNKCIJOS_ATRIBUTAS_FUNKCIJA]
FOREIGN KEY([func_id])
```

```
REFERENCES [dbo].[FUNKCIJA] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJOS_ATRIBUTAS] CHECK CONSTRAINT [FK_FUNKCIJOS_ATRIBUTAS_FUNKCIJA]
GO
/****** Object:  ForeignKey [FK_FUNKCIJOS_INFORMACINIS_SRAUTAS_FUNKCIJA]    Script Date: 06/02/2008 11:04:50 ******/
ALTER TABLE [dbo].[FUNKCIJOS_INFORMACINIS_SRAUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_FUNKCIJOS_INFORMACINIS_SRAUTAS_FUNKCIJA] FOREIGN KEY([func_id])
REFERENCES [dbo].[FUNKCIJA] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJOS_INFORMACINIS_SRAUTAS] CHECK CONSTRAINT
[FK_FUNKCIJOS_INFORMACINIS_SRAUTAS_FUNKCIJA]
GO
/****** Object:  ForeignKey [FK_FUNKCIJOS_INFORMACINIS_SRAUTAS_INFORMACINIS_SRAUTAS]    Script Date: 06/02/2008 11:04:51
******/
ALTER TABLE [dbo].[FUNKCIJOS_INFORMACINIS_SRAUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_FUNKCIJOS_INFORMACINIS_SRAUTAS_INFORMACINIS_SRAUTAS] FOREIGN KEY([infsr_id])
REFERENCES [dbo].[INFORMACINIS_SRAUTAS] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJOS_INFORMACINIS_SRAUTAS] CHECK CONSTRAINT
[FK_FUNKCIJOS_INFORMACINIS_SRAUTAS_INFORMACINIS_SRAUTAS]
GO
/****** Object:  ForeignKey [FK_FUNKCIJOS_VYKDYTOJAS_FUNKCIJA]    Script Date: 06/02/2008 11:04:52 ******/
ALTER TABLE [dbo].[FUNKCIJOS_VYKDYTOJAS]  WITH CHECK ADD  CONSTRAINT [FK_FUNKCIJOS_VYKDYTOJAS_FUNKCIJA]
FOREIGN KEY([funkc_id])
REFERENCES [dbo].[FUNKCIJA] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJOS_VYKDYTOJAS] CHECK CONSTRAINT [FK_FUNKCIJOS_VYKDYTOJAS_FUNKCIJA]
GO
/****** Object:  ForeignKey [FK_FUNKCIJOS_VYKDYTOJAS_VYKDYTOJAS]    Script Date: 06/02/2008 11:04:53 ******/
ALTER TABLE [dbo].[FUNKCIJOS_VYKDYTOJAS]  WITH CHECK ADD  CONSTRAINT [FK_FUNKCIJOS_VYKDYTOJAS_VYKDYTOJAS]
FOREIGN KEY([vykd_id])
REFERENCES [dbo].[VYKDYTOJAS] ([id])
GO
ALTER TABLE [dbo].[FUNKCIJOS_VYKDYTOJAS] CHECK CONSTRAINT [FK_FUNKCIJOS_VYKDYTOJAS_VYKDYTOJAS]
GO
/****** Object:  ForeignKey [FK_INFORMACINIO_SRAUTO_ATRIBUTAS_ATRIBUTAS]    Script Date: 06/02/2008 11:04:55 ******/
ALTER TABLE [dbo].[INFORMACINIO_SRAUTO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_INFORMACINIO_SRAUTO_ATRIBUTAS_ATRIBUTAS] FOREIGN KEY([atr_id])
REFERENCES [dbo].[ATRIBUTAS] ([id])
GO
ALTER TABLE [dbo].[INFORMACINIO_SRAUTO_ATRIBUTAS] CHECK CONSTRAINT
[FK_INFORMACINIO_SRAUTO_ATRIBUTAS_ATRIBUTAS]
GO
/****** Object:  ForeignKey [FK_INFORMACINIO_SRAUTO_ATRIBUTAS_INFORMACINIS_SRAUTAS]    Script Date: 06/02/2008 11:04:55
******/
ALTER TABLE [dbo].[INFORMACINIO_SRAUTO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_INFORMACINIO_SRAUTO_ATRIBUTAS_INFORMACINIS_SRAUTAS] FOREIGN KEY([infsrt_id])
REFERENCES [dbo].[INFORMACINIS_SRAUTAS] ([id])
GO
ALTER TABLE [dbo].[INFORMACINIO_SRAUTO_ATRIBUTAS] CHECK CONSTRAINT
[FK_INFORMACINIO_SRAUTO_ATRIBUTAS_INFORMACINIS_SRAUTAS]
GO
/****** Object:  ForeignKey [FK_IVYKIO_PROCESAS_IVYKIS]    Script Date: 06/02/2008 11:04:59 ******/
ALTER TABLE [dbo].[IVYKIO_PROCESAS]  WITH CHECK ADD  CONSTRAINT [FK_IVYKIO_PROCESAS_IVYKIS] FOREIGN
KEY([ivyk_id])
REFERENCES [dbo].[IVYKIS] ([id])
GO
ALTER TABLE [dbo].[IVYKIO_PROCESAS] CHECK CONSTRAINT [FK_IVYKIO_PROCESAS_IVYKIS]
GO
/****** Object:  ForeignKey [FK_IVYKIO_PROCESAS_PROCESAS]    Script Date: 06/02/2008 11:04:59 ******/
ALTER TABLE [dbo].[IVYKIO_PROCESAS]  WITH CHECK ADD  CONSTRAINT [FK_IVYKIO_PROCESAS_PROCESAS] FOREIGN
KEY([proc_id])
REFERENCES [dbo].[PROCESAS] ([id])
GO
ALTER TABLE [dbo].[IVYKIO_PROCESAS] CHECK CONSTRAINT [FK_IVYKIO_PROCESAS_PROCESAS]
GO
/****** Object:  ForeignKey [FK_KLASES_ATRIBUTAS_KLASE]    Script Date: 06/02/2008 11:05:04 ******/
ALTER TABLE [dbo].[KLASES_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_KLASES_ATRIBUTAS_KLASE] FOREIGN
KEY([klases_id])
REFERENCES [dbo].[KLASE] ([id])
GO
ALTER TABLE [dbo].[KLASES_ATRIBUTAS] CHECK CONSTRAINT [FK_KLASES_ATRIBUTAS_KLASE]
GO
```

```
/****** Object:  ForeignKey [FK_KLASES_INTERFEISAS_INTERFEISAS]    Script Date: 06/02/2008 11:05:06 ******/
ALTER TABLE [dbo].[KLASES_INTERFEISAS]  WITH CHECK ADD  CONSTRAINT [FK_KLASES_INTERFEISAS_INTERFEISAS]
FOREIGN KEY([int_id])
REFERENCES [dbo].[INTERFEISAS] ([id])
GO
ALTER TABLE [dbo].[KLASES_INTERFEISAS] CHECK CONSTRAINT [FK_KLASES_INTERFEISAS_INTERFEISAS]
GO
/****** Object:  ForeignKey [FK_KLASES_INTERFEISAS_KLASE]    Script Date: 06/02/2008 11:05:06 ******/
ALTER TABLE [dbo].[KLASES_INTERFEISAS]  WITH CHECK ADD  CONSTRAINT [FK_KLASES_INTERFEISAS_KLASE] FOREIGN
KEY([int_id])
REFERENCES [dbo].[KLASE] ([id])
GO
ALTER TABLE [dbo].[KLASES_INTERFEISAS] CHECK CONSTRAINT [FK_KLASES_INTERFEISAS_KLASE]
GO
/****** Object:  ForeignKey [FK_KLASES_OPERACIJA_KLASE]    Script Date: 06/02/2008 11:05:08 ******/
ALTER TABLE [dbo].[KLASES_OPERACIJA]  WITH CHECK ADD  CONSTRAINT [FK_KLASES_OPERACIJA_KLASE] FOREIGN
KEY([klases_id])
REFERENCES [dbo].[KLASE] ([id])
GO
ALTER TABLE [dbo].[KLASES_OPERACIJA] CHECK CONSTRAINT [FK_KLASES_OPERACIJA_KLASE]
GO
/****** Object:  ForeignKey [FK_KLASES_OPERACIJOS_PARAMETRAS_KLASES_OPERACIJA]    Script Date: 06/02/2008 11:05:10 ******/
ALTER TABLE [dbo].[KLASES_OPERACIJOS_PARAMETRAS]  WITH CHECK ADD  CONSTRAINT
[FK_KLASES_OPERACIJOS_PARAMETRAS_KLASES_OPERACIJA] FOREIGN KEY([op_id])
REFERENCES [dbo].[KLASES_OPERACIJA] ([id])
GO
ALTER TABLE [dbo].[KLASES_OPERACIJOS_PARAMETRAS] CHECK CONSTRAINT
[FK_KLASES_OPERACIJOS_PARAMETRAS_KLASES_OPERACIJA]
GO
/****** Object:  ForeignKey [FK_MATERIALAUS_SRAUTO_ATRIBUTAS_ATRIBUTAS]    Script Date: 06/02/2008 11:05:11 ******/
ALTER TABLE [dbo].[MATERIALAUS_SRAUTO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_MATERIALAUS_SRAUTO_ATRIBUTAS_ATRIBUTAS] FOREIGN KEY([atr_id])
REFERENCES [dbo].[ATRIBUTAS] ([id])
GO
ALTER TABLE [dbo].[MATERIALAUS_SRAUTO_ATRIBUTAS] CHECK CONSTRAINT
[FK_MATERIALAUS_SRAUTO_ATRIBUTAS_ATRIBUTAS]
GO
/****** Object:  ForeignKey [FK_MATERIALAUS_SRAUTO_ATRIBUTAS_MATERIALUS_SRAUTAS]    Script Date: 06/02/2008 11:05:12
******/
ALTER TABLE [dbo].[MATERIALAUS_SRAUTO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_MATERIALAUS_SRAUTO_ATRIBUTAS_MATERIALUS_SRAUTAS] FOREIGN KEY([matsrt_id])
REFERENCES [dbo].[MATERIALUS_SRAUTAS] ([id])
GO
ALTER TABLE [dbo].[MATERIALAUS_SRAUTO_ATRIBUTAS] CHECK CONSTRAINT
[FK_MATERIALAUS_SRAUTO_ATRIBUTAS_MATERIALUS_SRAUTAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_ATRIBUTAS_ATRIBUTAS]    Script Date: 06/02/2008 11:05:18 ******/
ALTER TABLE [dbo].[PROCESO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_PROCESO_ATRIBUTAS_ATRIBUTAS]
FOREIGN KEY([atr_id])
REFERENCES [dbo].[ATRIBUTAS] ([id])
GO
ALTER TABLE [dbo].[PROCESO_ATRIBUTAS] CHECK CONSTRAINT [FK_PROCESO_ATRIBUTAS_ATRIBUTAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_ATRIBUTAS_PROCESAS]    Script Date: 06/02/2008 11:05:18 ******/
ALTER TABLE [dbo].[PROCESO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_PROCESO_ATRIBUTAS_PROCESAS] FOREIGN
KEY([proc_id])
REFERENCES [dbo].[PROCESAS] ([id])
GO
ALTER TABLE [dbo].[PROCESO_ATRIBUTAS] CHECK CONSTRAINT [FK_PROCESO_ATRIBUTAS_PROCESAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_INFORMACINIS_SRAUTAS_INFORMACINIS_SRAUTAS]    Script Date: 06/02/2008 11:05:20
******/
ALTER TABLE [dbo].[PROCESO_INFORMACINIS_SRAUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_PROCESO_INFORMACINIS_SRAUTAS_INFORMACINIS_SRAUTAS] FOREIGN KEY([infsr_id])
REFERENCES [dbo].[INFORMACINIS_SRAUTAS] ([id])
GO
ALTER TABLE [dbo].[PROCESO_INFORMACINIS_SRAUTAS] CHECK CONSTRAINT
[FK_PROCESO_INFORMACINIS_SRAUTAS_INFORMACINIS_SRAUTAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_INFORMACINIS_SRAUTAS_PROCESAS]    Script Date: 06/02/2008 11:05:20 ******/
ALTER TABLE [dbo].[PROCESO_INFORMACINIS_SRAUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_PROCESO_INFORMACINIS_SRAUTAS_PROCESAS] FOREIGN KEY([proc_id])
REFERENCES [dbo].[PROCESAS] ([id])
```

```
GO
ALTER TABLE [dbo].[PROCESO_INFORMACINIS_SRAUTAS] CHECK CONSTRAINT
[FK_PROCESO_INFORMACINIS_SRAUTAS_PROCESAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_MATERIALUS_SRAUTAS_MATERIALUS_SRAUTAS]    Script Date: 06/02/2008 11:05:23
******/
ALTER TABLE [dbo].[PROCESO_MATERIALUS_SRAUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_PROCESO_MATERIALUS_SRAUTAS_MATERIALUS_SRAUTAS] FOREIGN KEY([matsr_id])
REFERENCES [dbo].[MATERIALUS_SRAUTAS] ([id])
GO
ALTER TABLE [dbo].[PROCESO_MATERIALUS_SRAUTAS] CHECK CONSTRAINT
[FK_PROCESO_MATERIALUS_SRAUTAS_MATERIALUS_SRAUTAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_MATERIALUS_SRAUTAS_PROCESAS]    Script Date: 06/02/2008 11:05:23 ******/
ALTER TABLE [dbo].[PROCESO_MATERIALUS_SRAUTAS]  WITH CHECK ADD  CONSTRAINT
[FK_PROCESO_MATERIALUS_SRAUTAS_PROCESAS] FOREIGN KEY([proc_id])
REFERENCES [dbo].[PROCESAS] ([id])
GO
ALTER TABLE [dbo].[PROCESO_MATERIALUS_SRAUTAS] CHECK CONSTRAINT
[FK_PROCESO_MATERIALUS_SRAUTAS_PROCESAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_VYKDYTOJAS_PROCESAS]    Script Date: 06/02/2008 11:05:25 ******/
ALTER TABLE [dbo].[PROCESO_VYKDYTOJAS]  WITH CHECK ADD  CONSTRAINT [FK_PROCESO_VYKDYTOJAS_PROCESAS]
FOREIGN KEY([proc_id])
REFERENCES [dbo].[PROCESAS] ([id])
GO
ALTER TABLE [dbo].[PROCESO_VYKDYTOJAS] CHECK CONSTRAINT [FK_PROCESO_VYKDYTOJAS_PROCESAS]
GO
/****** Object:  ForeignKey [FK_PROCESO_VYKDYTOJAS_VYKDYTOJAS]    Script Date: 06/02/2008 11:05:25 ******/
ALTER TABLE [dbo].[PROCESO_VYKDYTOJAS]  WITH CHECK ADD  CONSTRAINT [FK_PROCESO_VYKDYTOJAS_VYKDYTOJAS]
FOREIGN KEY([vykd_id])
REFERENCES [dbo].[VYKDYTOJAS] ([id])
GO
ALTER TABLE [dbo].[PROCESO_VYKDYTOJAS] CHECK CONSTRAINT [FK_PROCESO_VYKDYTOJAS_VYKDYTOJAS]
GO
/****** Object:  ForeignKey [FK_RYSYS_KLASE]    Script Date: 06/02/2008 11:05:28 ******/
ALTER TABLE [dbo].[RYSYS]  WITH CHECK ADD  CONSTRAINT [FK_RYSYS_KLASE] FOREIGN KEY([pagr_klase])
REFERENCES [dbo].[KLASE] ([id])
GO
ALTER TABLE [dbo].[RYSYS] CHECK CONSTRAINT [FK_RYSYS_KLASE]
GO
/****** Object:  ForeignKey [FK_RYSYS_KLASE1]    Script Date: 06/02/2008 11:05:28 ******/
ALTER TABLE [dbo].[RYSYS]  WITH CHECK ADD  CONSTRAINT [FK_RYSYS_KLASE1] FOREIGN KEY([antr_klase])
REFERENCES [dbo].[KLASE] ([id])
GO
ALTER TABLE [dbo].[RYSYS] CHECK CONSTRAINT [FK_RYSYS_KLASE1]
GO
/****** Object:  ForeignKey [FK_RYSYS_RYSIO_TIPAS]    Script Date: 06/02/2008 11:05:28 ******/
ALTER TABLE [dbo].[RYSYS]  WITH CHECK ADD  CONSTRAINT [FK_RYSYS_RYSIO_TIPAS] FOREIGN KEY([tipo_id])
REFERENCES [dbo].[RYSIO_TIPAS] ([id])
GO
ALTER TABLE [dbo].[RYSYS] CHECK CONSTRAINT [FK_RYSYS_RYSIO_TIPAS]
GO
/****** Object:  ForeignKey [FK_TRIGERIO_ATRIBUTAS_SD_ATRIBUTAS]    Script Date: 06/02/2008 11:05:32 ******/
ALTER TABLE [dbo].[TRIGERIO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_TRIGERIO_ATRIBUTAS_SD_ATRIBUTAS]
FOREIGN KEY([atributo_id])
REFERENCES [dbo].[SD_ATRIBUTAS] ([id])
GO
ALTER TABLE [dbo].[TRIGERIO_ATRIBUTAS] CHECK CONSTRAINT [FK_TRIGERIO_ATRIBUTAS_SD_ATRIBUTAS]
GO
/****** Object:  ForeignKey [FK_TRIGERIO_ATRIBUTAS_TRIGERIS]    Script Date: 06/02/2008 11:05:32 ******/
ALTER TABLE [dbo].[TRIGERIO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_TRIGERIO_ATRIBUTAS_TRIGERIS] FOREIGN
KEY([trigerio_id])
REFERENCES [dbo].[TRIGERIS] ([id])
GO
ALTER TABLE [dbo].[TRIGERIO_ATRIBUTAS] CHECK CONSTRAINT [FK_TRIGERIO_ATRIBUTAS_TRIGERIS]
GO
/****** Object:  ForeignKey [FK_VYKDYTOJO_ATRIBUTAS_ATRIBUTAS]    Script Date: 06/02/2008 11:05:37 ******/
ALTER TABLE [dbo].[VYKDYTOJO_ATRIBUTAS]  WITH CHECK ADD  CONSTRAINT [FK_VYKDYTOJO_ATRIBUTAS_ATRIBUTAS]
FOREIGN KEY([atr_id])
REFERENCES [dbo].[ATRIBUTAS] ([id])
GO
ALTER TABLE [dbo].[VYKDYTOJO_ATRIBUTAS] CHECK CONSTRAINT [FK_VYKDYTOJO_ATRIBUTAS_ATRIBUTAS]
```

GO


```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using DemoMag.MainDataSetTableAdapters;

namespace DemoMag
{
   public partial class DemoMagMainForm : Form
   {
      public DemoMagMainForm()
      {
         InitializeComponent();
      }

      private void DemoMagMainForm_Load(object sender, EventArgs e)
      {
         this.mATERIALUS_SRAUTASTableAdapter.Fill(this.mainDataSet.MATERIALUS_SRAUTAS);
         this.iNFORMACINIS_SRAUTASTableAdapter.Fill(this.mainDataSet.INFORMACINIS_SRAUTAS);
         this.vYKDYTOJASTableAdapter.Fill(this.mainDataSet.VYKDYTOJAS);
      }

      private void btnGeneruoti_Click(object sender, EventArgs e)
      {
         if (!this.ckbSaveAll.Checked)
         {
            Klase klase = new Klase();
            klase.Name = this.cmbVykdytojai.Text;
            VYKDYTOJO_PROCESASTableAdapter vykdytojoProcesoAdapteris = new VYKDYTOJO_PROCESASTableAdapter();
            DataTable vykdytojoProcesuLentele =
               vykdytojoProcesoAdapteris.GetDataByVykdytojas((int) this.cmbVykdytojai.SelectedValue);

            PROCESO_INFORMACINIS_PARAMETRASTableAdapter informaciniuParametruAdapteris =
               new PROCESO_INFORMACINIS_PARAMETRASTableAdapter();
            PROCESO_MATERIALUS_PARAMETRASTableAdapter materialiuParametruAdapteris =
               new PROCESO_MATERIALUS_PARAMETRASTableAdapter();

            foreach (DataRow procesas in vykdytojoProcesuLentele.Rows)
            {
               Operacija klasesOperacija = new Operacija(procesas["pavadinimas"].ToString());
               foreach (
                  DataRow informacinisParametras in
                     informaciniuParametruAdapteris.GetInfParametrasByProcesas((int) procesas["proc_id"]).Rows)
               {
                  if (!string.IsNullOrEmpty(informacinisParametras["Parametras"].ToString()))
                  {
                     klasesOperacija.Parametrai.Add(informacinisParametras["Parametras"].ToString(),
                                    (bool) informacinisParametras["kryptis"]);
                  }
               }

               foreach (
                  DataRow materialinisParametras in
                     materialiuParametruAdapteris.GetMatParametrasByProcesoId((int) procesas["proc_id"]).Rows)
               {
                  if (!string.IsNullOrEmpty(materialinisParametras["Parametras"].ToString()))
                  {
                     klasesOperacija.Parametrai.Add(materialinisParametras["Parametras"].ToString(),
                                    (bool) materialinisParametras["kryptis"]);
                  }
               }
               klase.Operacijos.Add(klasesOperacija);
            }
            klase.SaveToDB();
            klase.Serialize();

         }
         else
```

```
        {
            foreach (DataRow row in this.mainDataSet.VYKDYTOJAS.Rows)
            {
                Klase klase = new Klase();
                klase.Name = row[1].ToString();
                VYKDYTOJO_PROCESASTableAdapter vykdytojoProcesoAdapteris = new VYKDYTOJO_PROCESASTableAdapter();
                DataTable vykdytojoProcesuLentele =
                    vykdytojoProcesoAdapteris.GetDataByVykdytojas((int)row[0]);

                PROCESO_INFORMACINIS_PARAMETRASTableAdapter informaciniuParametruAdapteris =
                    new PROCESO_INFORMACINIS_PARAMETRASTableAdapter();
                PROCESO_MATERIALUS_PARAMETRASTableAdapter materialiuParametruAdapteris =
                    new PROCESO_MATERIALUS_PARAMETRASTableAdapter();

                foreach (DataRow procesas in vykdytojoProcesuLentele.Rows)
                {
                    Operacija klasesOperacija = new Operacija(procesas["pavadinimas"].ToString());
                    foreach (
                        DataRow informacinisParametras in
                            informaciniuParametruAdapteris.GetInfParametrasByProcesas((int)procesas["proc_id"]).Rows)
                    {
                        if (!string.IsNullOrEmpty(informacinisParametras["Parametras"].ToString()))
                        {
                            klasesOperacija.Parametrai.Add(informacinisParametras["Parametras"].ToString(),
                                                (bool)informacinisParametras["kryptis"]);
                        }
                    }

                    foreach (
                        DataRow materialinisParametras in
                            materialiuParametruAdapteris.GetMatParametrasByProcesoId((int)procesas["proc_id"]).Rows)
                    {
                        if (!string.IsNullOrEmpty(materialinisParametras["Parametras"].ToString()))
                        {
                            klasesOperacija.Parametrai.Add(materialinisParametras["Parametras"].ToString(),
                                                (bool)materialinisParametras["kryptis"]);
                        }
                    }
                    klase.Operacijos.Add(klasesOperacija);
                }
                klase.SaveToDB();
            }
        }
    }
}


using System;
using System.Collections.Generic;
using System.Text;
using DemoMag.MainDataSetTableAdapters;
using System.Xml;

namespace DemoMag
{
    [Serializable]
    public class Klase
    {
        #region Fields
        private string name = string.Empty;
        private List<Operacija> operacijos = new List<Operacija>();
        private List<Operacija> atributai = new List<Operacija>();
        #endregion

        #region Constructors
        public Klase()
        {

        }
        #endregion

        #region Properties
```

```
        public List<Operacija> Operacijos
        {
            get
            {
                return this.operacijos;
            }
            set
            {
                this.operacijos = value;
            }
        }
        public List<Operacija> Atributai
        {
            get
            {
                return this.atributai;
            }
            set
            {
                this.atributai = value;
            }
        }
        public string Name
        {
            get
            {
                return this.name;
            }
            set
            {
                this.name = value;
            }
        }
        #endregion
        #region Public Methods
        private string RemoveSpace(string tempString)
        {
            string[] temp = tempString.Trim(' ').Split(' ');
            StringBuilder newString = new StringBuilder();
            for(int i = 0; i < temp.Length; i++)
            {
                string firsLetter = temp[i][0].ToString().ToUpper();
                newString.Append(firsLetter);
                newString.Append(temp[i].Remove(0,1));
            }
            return newString.ToString();
        }
        #endregion

        #region Public Methods
        public void SaveToDB()
        {
            KLASETableAdapter klasesAdapteris = new KLASETableAdapter();
            klasesAdapteris.Insert(RemoveSpace(this.name), false);
            int klases_id = (int)klasesAdapteris.GetIdByName(RemoveSpace(this.name));
            KLASES_OPERACIJATableAdapter klasesOperacijosAdapteris = new KLASES_OPERACIJATableAdapter();
            KLASES_OPERACIJOS_PARAMETRASTableAdapter klasesOperacijosParametras = new
KLASES_OPERACIJOS_PARAMETRASTableAdapter();
            foreach(Operacija operacija in this.operacijos)
            {
                klasesOperacijosAdapteris.Insert(klases_id, RemoveSpace(operacija.Name));
                int operacijos_id = (int)klasesOperacijosAdapteris.GetOperationIdByName(RemoveSpace(operacija.Name));
                foreach (KeyValuePair<string,bool > parametras in operacija.Parametrai)
                {
                    klasesOperacijosParametras.Insert(RemoveSpace(parametras.Key), operacijos_id, parametras.Value);
                }
            }
        }
        public void Serialize()
        {
            XmlDocument doc = new XmlDocument();
            XmlElement rootElem = doc.CreateElement("Class");
            rootElem.InnerText = RemoveSpace(this.name);
```

```
            foreach (Operacija operacija in operacijos)
            {
                XmlElement opElem = doc.CreateElement("Metodas");
                opElem.InnerText = RemoveSpace(operacija.Name);
                foreach (KeyValuePair<string,bool> param in operacija.Parametrai)
                {
                    XmlElement paramNameElem = doc.CreateElement("Parametras");
                    paramNameElem.InnerText = RemoveSpace(param.Key);
                    XmlElement paramDirectionElem = doc.CreateElement("Kryptis");
                    if(param.Value)
                    {
                        paramDirectionElem.InnerText = "Out";
                    }
                    else
                    {
                        paramDirectionElem.InnerText = "In";
                    }
                    paramNameElem.AppendChild(paramDirectionElem);
                    opElem.AppendChild(paramNameElem);
                }
                rootElem.AppendChild(opElem);
            }
            doc.AppendChild(rootElem);
            doc.Save("Klase_"+this.name+".xml");
        }
        #endregion
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace DemoMag
{
    [Serializable]
    public class Operacija
    {
        #region Fields
        private string name = string.Empty;
        Dictionary<string,bool> parametrai = new Dictionary<string, bool>();
        #endregion
        #region Constructors
        public Operacija(string name)
        {
            this.name = name;
        }
        #endregion
        #region Properties
        public Dictionary<string,bool> Parametrai
        {
            get
            {
                return this.parametrai;
            }
            set
            {
                this.parametrai = value;
            }
        }
        public string Name
        {
            get
            {
                return this.name;
            }
            set
            {
                this.name = value;
            }
        }
        #endregion
    }
```

}

Programoje naudojamo DataSet‘o struktūra