KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Donatas Budzinauskas

Verslo transakcijų specifikavimas kuriant verslo valdymo sistemas

Magistro darbas

Darbo vadovas

prof. dr. R. Butleris

Kaunas, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Donatas Budzinauskas

Verslo transakcijų specifikavimas kuriant verslo valdymo sistemas

Magistro darbas

Recenzentas

doc. dr. T. Blažauskas

2008-01-14

Vadovas

prof. dr. R. Butleris

2008-01-14

Atliko

IFM-2/4 gr. stud.

Donatas Budzinauskas

2008-01-14

Kaunas, 2008

Specification of Business Transactions for Enterprise System Modelling

Summary

In nowadays a lot of business companies and organizations are using information systems. This paper analyses business transactions, tools and technologies for transactions and process modelling of information and enterprise systems. Analyzed popular technologies like UML and its extensions, RUP, XMI, MDA. Familiarized with MERODE and Agile methods. More closely analyzed BPMN notation and its possibilities for code generation. BPMN notation augment with data objects for better code generation solutions. It also gives better understanding of data flows and influence of business process and transactions. Defined process and transactions modelling strategy which allows better code generations solution too. Made-up, practically materialized ant tested code generation algorithm.

Turinys

1. Įvadas

Moderniame technologijų amžiuje įmonėms sunku sudaryti ilgalaikius planus ir vykdyti veiklą jais remiantis ilgą laikotarpį. Griežtai ilgalaikių planų besilaikančios įmonės pasmerktos žlugti. Svarbiausia įmonėms būti lanksčioms, nuolatos keistis, tobulėti. Nuolatinis keitimasis apima sudėtingus procesus tokius kaip adaptavimasis, integracija, reinžinerija ir panašiai. Iškyla problemos kaip pernešti sukauptą patirtį, kaip tai panaudoti kuriant naujas strategijas bei valdymo principus. Modernizuojant organizacijas, modernizuojant jų veiklą, įgalinant organizaciją keistis dinamiškai neapsieinama be informacinių sistemų, gamybos procesų valdymo sistemų. Jas kuriant, integruojant, suderinant ir kyla daug problemų, kurios stabdo sėkmingą įmonės veiklą. O esant puikiai suderintoms sistemoms įmonės veikla įgauną pagreitį bei reikiami rezultatai pasiekiami itin greitai. Verslo procesų modeliavimas pagerina jų realizaciją, sumažina kūrimo kaštus ir taip didina įmonių pelningumą.

Modeliavimo galimybių analizės tikslas apžvelgti, susipažinti su sistemų, verslo, veiklos modeliavimo metodologijomis, modeliavimo kalbomis. Suprasti kaip modeliavimo technologijos ir metodologijos naudoja verslo ir sistemos modelius sistemos gyvavimo cikle (GC). Suprasti galimus modelių santykius, modeliavimo eigoje galimu perėjimus. Išanalizuoti projekto vystymo eigos skaidymo į etapus galimybes bei informacijos pernešimą tarp etapų. Išanalizuoti publikacijas ir straipsnius susijusius su verslo ir veiklos modeliavimu bei jo panaudojimo galimybes IS kūrimo metu.

Darbo sritis – informacinių sistemų projektavimo bei detalaus projektavimo etapo problemos, susijusios su verslo transakcijų specifikavimu, modeliavimu siekiant padaryti aiškų projektą, pagal kurį kuriant verslo valdymo sistemą būtų pilnai išpildyti verslo procesų reikalavimai. Transakciją laikant kaip eilę trumpų verslo procesų, veiklos elementų, kurie turi loginę seką, pradžią ir pabaigą. Analizuojama ne tik transakcijų užrašymo notacija, bet jų panaudojimas, interpretavimas tolesniuose etapuose. Sprendžiami uždaviniai, kaip aprašytas transakcijas modeliuoti siekiant aiškesnės, greitesnės ir tikslesnės realizacijos. Tai apima verslo veiklos analizę ir jos aprašymą didelį dėmesį kreipiant į procesų sąveiką. Taip pat analizuojama versle (realioje veikloje) ir projektavime naudojamų objektų atitikmenys, sąveikos ir galimi panaudojimo būdai. Veiklos modelių bei sistemos modelių sujungimo problematika, šių modelių panaudojimas tolesniuose sistemų vystymo etapuose bei modelių sudarymo galimybės. Metodų, metodologijų bei technologijų suderinamumas, galimos kombinacijos, panaudojimo būdai skirtinguose sistemų kūrimo etapuose.

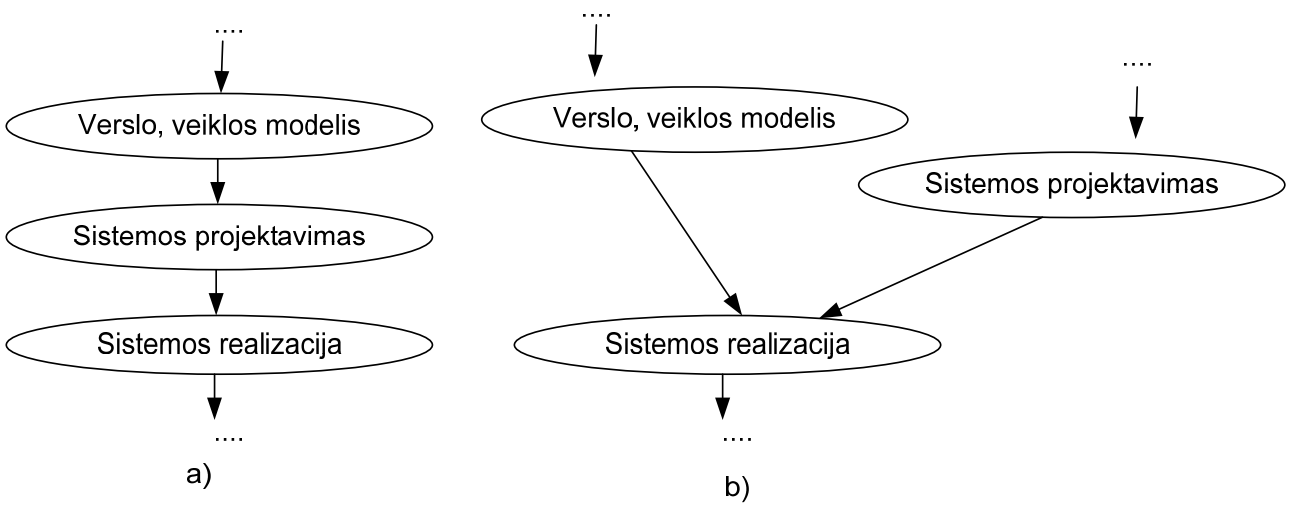
Darbo tikslas rasti ar patobulinti technologiją, metodologiją ar jų kombinacija, kuri tiktų verslo valdymo sistemų detalaus projektavimo etapui. Metodologijos ir technologijos turi leisti lengvai aprašyti, modeliuoti verslo transakcijas, juose dalyvaujančius objektus. Modeliai panaudojami verslo valdymo sistemos ar jos dalių kodo generavimui. Išanalizuoti sistemų kūrimo procesus, tokius kaip: RUP, MERODE ir kitus, surandant ir pritaikant jų tinkamiausias savybes verslo procesų modeliavimui. Surasti sistemų vystymo metodą, tinkamą diagramų notaciją ir ryšius tarp skirtingų GC etapų bei jų modelių. Į verslo valdymo sistemų projektavimą, kuris dažnai koncentruojasi į objektus, jų būsenas, įvesti procesus, transakcijas, kurios suteiktų sistemos realizacijai daugiau dinamiškumo.

Patobulinta BPMN notacija taip kad modeliuojant verslo valdymo sistemas, modelyje atsispindėtų tiek dinaminiai, tiek statiniai sistemos elementai. Aprašyta modeliavimo strategija verslo procesų modeliui. Aprašytas transakcijų savybių panaudojimas, generuojant kodą. Sudarytas kodo generavimo algoritmas iš verslo procesų modelio. Algoritmas patikrintas praktiškai jį realizuojant.

2. Verslo ir jo sistemų modeliavimo galimybių analizė

2.1 Verslo ir veiklos modeliavimas kuriant IS

Kuriantis ir tobulėjant objektiškai orientuotoms sistemoms, objektiškai orientuotam sistemų modeliavimui, o į modeliavimo (CASE) sistemas ir įrankius įtraukus veiklos modeliavimą, iškilo veiklos modeliavimo standartų poreikis. Vystymosi eigos pradžioje buvo įtraukiami standartiniai tuo metu jau gerai žinomi veiklos modeliai (DFD, Work flow ir pan.). Kadangi veiklos modeliai vystėsi atskirai nuo objektinio modeliavimo jie yra nesuderinami ar nepritaikyti derinti su objektiškai orientuotomis modeliavimo kalbomis, jų modeliais. Modeliavimo įrankių uždavinys rasti sąlyčio taškus, rasti metodus, kurie nusakytų ryšį tarp veiklos modeliavimo ir sistemos projektavimo. Verslo, veiklos modeliavimas tampa vis labiau svarbus sistemos analizės dalis, kuri turi didelį vaidmenį realizuojant sistemą ar sudarant sistemos detalų modelį. Verslo ir veiklos modelis gali dalyvauti sistemos kūrime dvejopai: pagal jį sudaromas sistemos modelis (*1 pav. a*), naudojama informacija detaliame sistemos projektavime ar realizavime (*1 pav. b*).

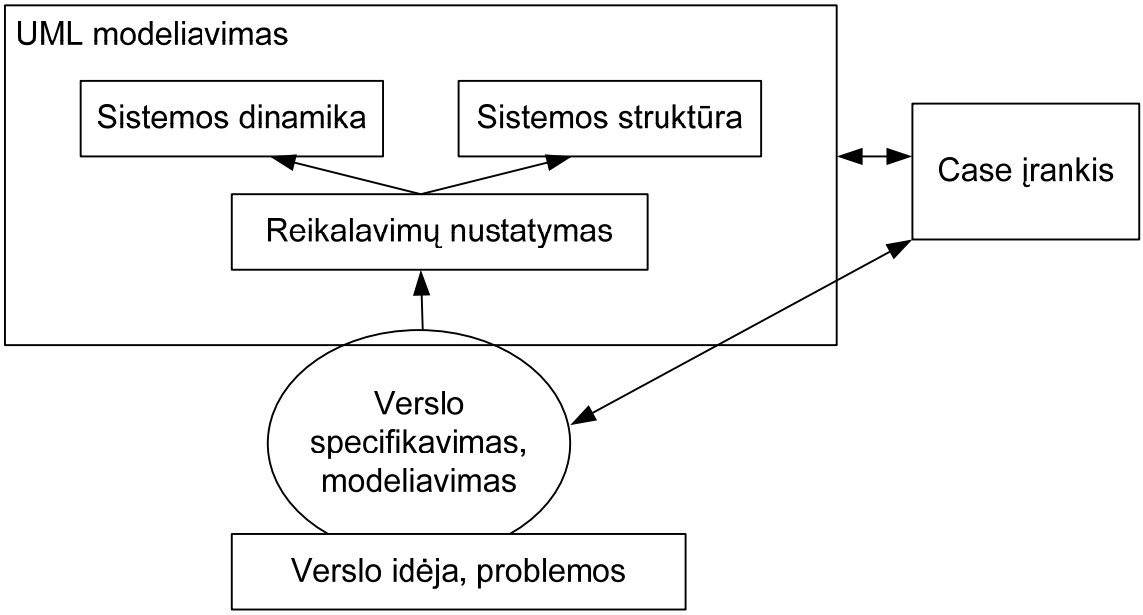


1 pav. Verslo modelio dalyvavimas sistemos kūrime

2.1.1 UML verslo modeliavime

UML viena populiariausių objektiškai orientuotų modeliavimo kalbų, kurios šalininkai drąsiai teigia, jog tai geriausia modeliavimo kalba. Galbūt... Apie projektavimo kalbos naudingumą, tinkamumą galima spręsti, tik tuomet, jei yra žinoma, specifikacijos detalumo lygmuo, verslo sferos, jos veikimo detalumo lygmuo. Jim Heumann savo straipsnyje [1, J. Heumann], kuris publikuojamas IBM tinklalapyje teigia, jog UML itin tinka verslo

modeliavimui. Bet akcentuoja, kad norint modeliuoti reikia tiksliai žinoti visus dalyvius, procesus, technologijas, kaip jos tarpininkauja ir pan. O verslo modeliavimo procese dažniausiai yra žinomi tik pagrindiniai procesai, pagrindiniai dalyviai, o technologijos, objektų tarpininkavimas paaiškėja tik modeliuojant ar specifikuojant sistemą, veiklos sritį. UML modeliavimas puikiai tinka tuomet kai pilnai atlikta reikalavimų analizė ir specifikuoti procesai bei verslo objektai.



2 pav. UML panaudojimas verslo kompiuterizavimas

Modeliuojant sistemą jau turi būti detali sistemos specifikacija, o modeliuojant verslą paprastai nėra žinomos konkrečios technologijos, kurių pagalba bus kompiuterizuojama veikla, nežinoma kiek objektų bus naudojama, kokio lygio išskaidymas į modulius ar objektus. Retai kada verslo objektai atitinka sistemos objektus, ar modulius. Dažnai atliekama kompozicija ar dekompozicija. Programiniai objektai išskiriami iš verslo modelio. Jei verslo sistemų modeliavimą skaidytume į tris etapus: verslo modeliavimas, sistemos reikalavimų sudarymas, sistemos struktūros projektavimas ir dinamikos modeliavimas, tuomet UML tinka antram ir trečiam etapui.

UML modeliai yra grindžiami tvirtais bei griežtais ryšiais: priklauso, paveldi, naudoja. Šie ryšiai apibrėžia objektų sąveika tik vienareikšmiškai ir todėl pasislepia tam tikrus modeliuojamos sistemos veikimo ypatumus. Panagrinėjus viena griežčiausių ryšių – paveldėjimą, galima pastebėti, kad kyla problemų dėl privačių metodų, kintamųjų [2, E. Pakalnickas]. Jie nepasiekiami kitiems objektams, neįmanomas kitoks objektų ryšys ir pan. Visiškai viešas (public) tipas mažina saugumo lygį. Verslo valdymo sistemos yra dinamiškos, turinčios išskirtinių savybių, kurios negali būti nusakytos vienareikšmiškai, o jei negalime

keisti poklasių struktūros, negalime vystyti sistemos. Susiduriame su didžiulėmis tobulinimo ir reinžinerijos problemomis.

2.1.2 MDA idėjos verslo modeliavime

Daugelyje sistemų modeliavimo gyvavimo ciklų randami tokie etapai:

- vartotojo reikalavimų ir sistemos specifikavimas
- sistemos modelio sudarymas (sistemos modeliavimas), remiantis specifikacija bei reikalavimais.

Abu etapai reikalauja daug rankinio darbo, kuris sunkiai gali būti automatizuojamas. Pasikeitimai specifikacijoje lėtai ir sunkiai perkeliami į sistemos modelį. Todėl galime teigti, jog iteracinis sistemos kūrimas, ar sistemos tobulinimas yra labai ilgas, sudėtingas ir brangus procesas. Siekiant pagreitinti šį procesą buvo suformuluotos MDA (Model driven architecture) idėjos (3 pav.). MDA idėja teigia, jog galima:

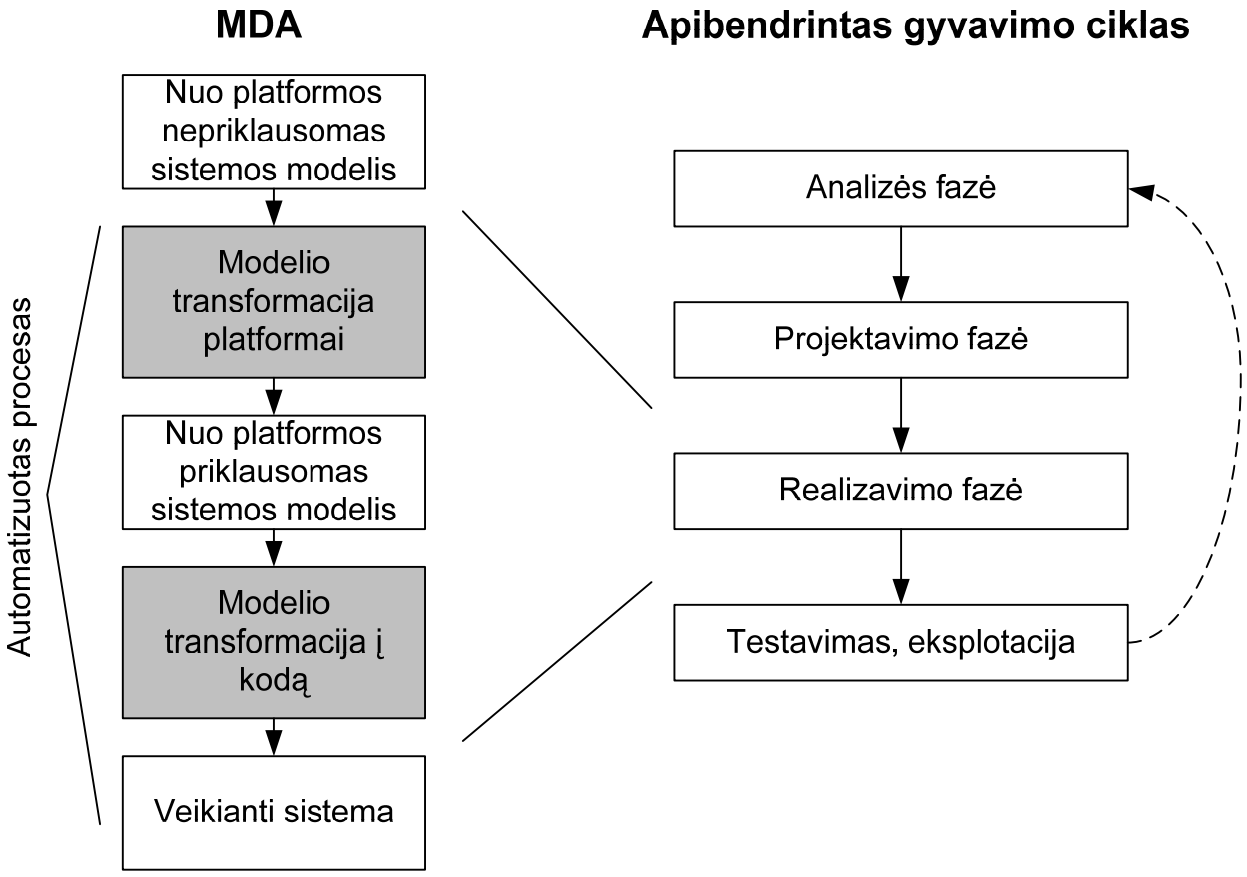
- žymiai pagreitinti programinės įrangos kūrimą;
- padidinti programinės įrangos kokybę;
- sumažinti rankinio darbo kiekį;
- žymiai sumažinti prisirišimą prie technologijų, programavimo kalbų;
- greitai reaguoti į reikalavimų pasikeitimus;

MDA idėjos daug žadančios, bet susiduriama su praktine problema: ar tiksliai modeliavimo įrankiai (CASE) įgyvendins šias idėjas. Įgyvendinus MDA būtų iš specifikacijos padaromas sistemos modelis, o iš jo gaunama jau veikianti sistema. Tokių atvejų sistemų gyvavimo ciklas stipriai sutrumpėtu, o gal tiksliau pagreitėtu. Ypač pagreitėtų tobulino, reinžinerijos procesai. Dėl to iteracinis kūrimo procesas būtų daug efektyvesnis. Po kiekvieno etapo, dalines realizacijas būtų galima pateikti klientui, testuotojams, vartotojams, kurie greitai ir tiksliai pastebėtų trūkumus, o sistemos tobulinimas butų greitas procesas.

„Agile“ projektavimo įrankio kūrėjai, prieš pradėdami kurti CASE įrankį, analizavo sistemų gyvavimo ciklus, jų trūkumus, problemas iškylančias kuriant sistemas [3, Agile]. Jų idėjose susijusiose su kūrimo proceso spartinimu, akcentuojama tai, jog sistemos vartotojas turi kuo dažniau matyti sistemą, kad daugelis jos trūkumų būtų pastebėti kuo ankščiau. Tai svarbus aspektas norint sukurti vartotojišką sistemą.

3 paveikslėlyje pateikta schema kaip MDA dalyvauja sistemų gyvavimo cikle. MDA apima kone ilgiausiai trunkantį bei daug dėmesio reikalaujantį etapą, kurio metu išryškėja projektavimo klaidos, specifikacijos trūkumai ir t.t. Naudojantis MDA išlieka neišspręstas transformavimas iš specifikacijos ar verslo modelio į sistemos modelį arba transformavimas

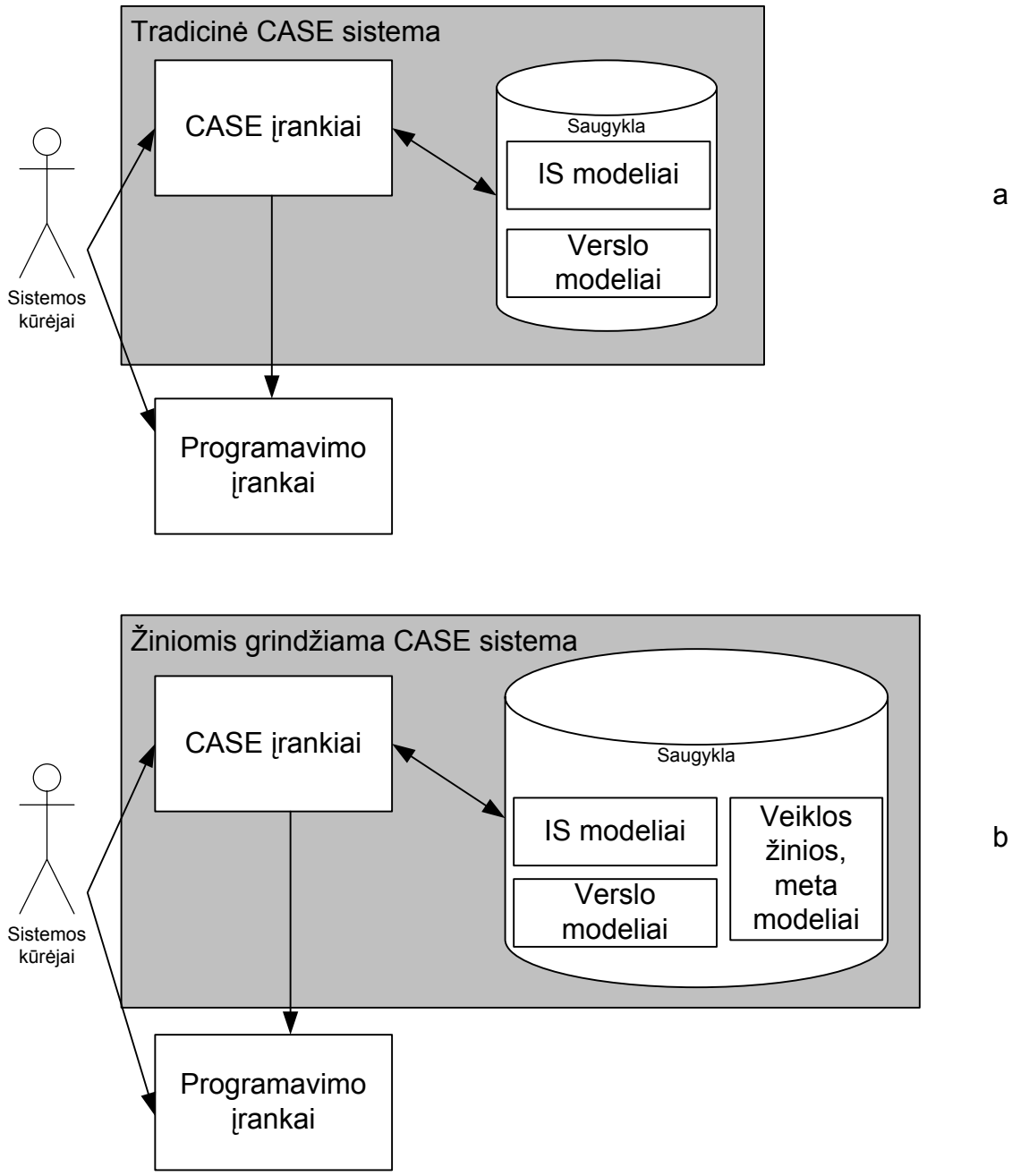
informacijos, modelių tarp analizės ir projektavimo. OMG publikacijoje „MDA guide“ [4, MDA guide] yra užsimenama, jog nuo platformos priklausomas sistemos modelis gaunamas iš nuo platformos nepriklausomo modelio, kuris gali būti sistemos arba verslo modelis. Nėra akcentuojama kaip verslo modelis ir sistemos modelis siejasi. Kurio duomenys kur ir kaip naudojami. MDA apsiriboja tuo, jog modeliai naudojami transformavimui į sistemą. Problemų sprendimas paliekamas transformacijoms. MDA nenagrinėja problemų kaip sudaryti šiuos modelius. Ne kiekviena sistema yra kompiuterinis verslo atitikmuo, bet kuo daugiau verslo procesų kompiuterizuojama, tuo labiau biznis persikelia į kompiuterines sistemas. Tuo tarpu kuriant tokias sistemas itin reikalingas verslo modelis, kurio remiantis būtų projektuojama sistema.



3 pav. MDA ir sistemų gyvavimo ciklas

2.1.3 Žiniomis grindžiama IS inžinerija

Siekiant suartinti verslo ir sistemos modelius, surasti ryšį tarp jų, naudingai tai panaudoti projektavime, realizavime ar kituose GC etapuose, reikalinga daug informacijos susijusios su kuriama sistema ir verslo sfera turėti vienoje aplinkoje. Šiuo metu daugelis įrankių ir metodų siūlo įvairius būdus kaip naudoti jau turimą informaciją, duomenis sekančiuose etapuose, sekančiuose modeliuose. Neretai ši seka yra gana griežtai nusakoma projektavimo eiga, projekto realizavimo eiga kuri skirta palengvinti procesą ar nukreipti projektuotoją taip, kad pats projektuotojas surastu ryšius tarp projektuojamos sistemos modelių. Žiniomis grindžiamos IS inžinerijos esmė tai, kad viso proceso metu naudojama veiklos žinių bazė, kurioje yra detalizuoti veiklos modeliai, o jais remiantis modeliuojama sistema. Veiklos modelio paskirtis žiniomis grindžiamoje IS – specifikuoti veiklos dėsningumą, iš veiklos modelio sudaryti GC, jo pagrindu verifikuoti gautus rezultatus ar kūrėjų priimtus sprendimus [5, S. Gudas]. Žiniomis grindžiamos IS struktūros esminis skirtumas nuo tradicinės tai, kad visas procesas remiasi veiklos, verslo žiniomis ir tai iš esmės keičia patį kūrimo procesą. Sprendimai priimami ne tik modelis iš modelio, modelis iš specifikacijos, realizavimas iš modelio, bet modelis iš modelio, realizavimas iš modelio ir t.t., daugelis etapų iš modelio. Viską grindžiant veiklos žiniomis, visus sprendimus derinant su veiklos žiniomis (4 pav.).



4 pav. a – tradicinė IS inžinerija, b – žiniomis grindžiama IS inžinerija

2.1.4 Modeliavimo metodai ir vartotojas

Daugelis šiuo metu egzistuojančių modeliavimo kalbų, metodologijų reikalauja daug žinių ir patirties. Netik bendrų žinių, bet ir detalaus studijavimo konkrečios technologijos. Didesnės apimties sistemų kūrime dalyvauja žmonės, kurie išmano tik savo darbo sritį. Daugelis šiuolaikinių metodų siekia apimti kuo didesnę GC dalį kaip vientisą procesą, kuo daugiau kompiuterizuoti projektavimą vienoje aplinkoje. Dėl šios priežasties iškyla problemų, kurios susijusios su diagramų, teiginių skirtingu interpretavimu. Skirtingos srities specialistai gali skirtingai suprasti mintis išreikštas diagramose, modeliuose ir t.t. Todėl metodai,

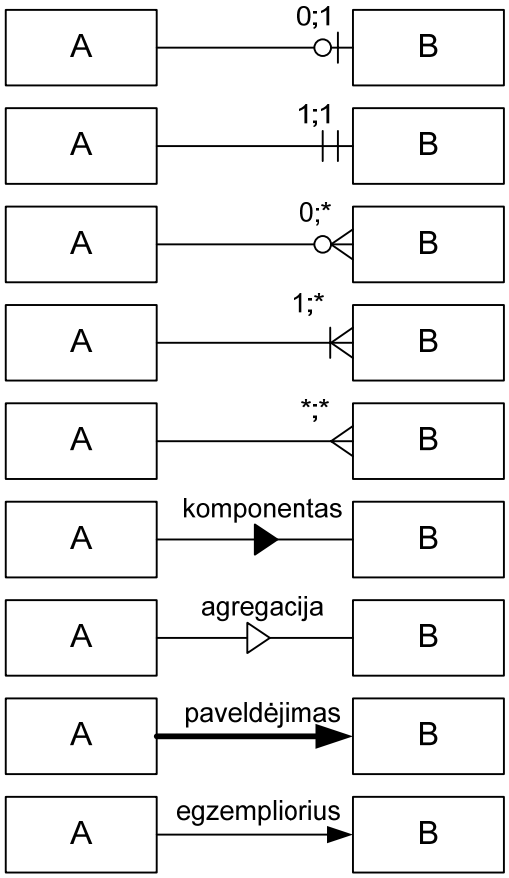
metodologijos, įrankiai turėtu į tai atkreipti dėmesį, kad apjungus skirtingus GC metodus su įrankių galėtu dirbti skirtingą kvalifikaciją bei patirtį turintys vartotojai. Dar didesnė bėda iškyla jei sistemos užsakovas ir vykdytojas yra skirtingos kompanijos ir vykdytojas mažai žino apie kliento veiklos sritį. Tuomet iškyla poreikis derinti tarpinius modelius, sprendimus su klientu. Daugelis modeliavimo technologijų visiškai tam netinka. „Agile“ įrankio kūrėjai tai sprendžia siekdami, kad po tam tikrų projektavimo etapų būtų galima sugeneruoti, dalinę sistemą, dalinai veikiančia, ar vartotojo sąsają, tam, kad sistemą būtų galima pristatyti klientui, o jo pastebėtus trukus ištaisyti, kol nėra sukurta visa sistema.

2.1.5 Semantinis ir pragmatinis modeliavimas

Kuriant ir vystantis UML nebuvo kreipiamas didelis dėmesys į veiklos modeliavimą. Modeliavimas prasideda praktiškai nuo sistemos ar nuo sistemos reikalavimų. Plečiantis požiūriui į modeliavimą ir vis labiau akcentuojant dalykinės srities ar veiklos modelių įtaką projektui buvo pritaikomas UML [11, UML User guide]. Gal labiau nepritaikomas, bet keičiamas požiūris, formuojami kiti UML diagramų panaudojimo būdai. Kadangi UML buvo pritaikomas atsirado ir naujų požiūrių į organizacijos modeliavimą. Vieną iš jų profesorius R. Gustas aprašo savo knygoje „Semantic and Pragmatic dependencies of information systems“ [6, R. Gustas]. Jis siūlo organizacijos veiklą modeliuoti šiais modeliais: objektų būsenų kaitų modeliu, esybių modeliu bei įtakos priklausomybėmis. Objektų būsenų kaita ir esybių modeliai siejasi su UML modeliais. Esybių modelis su klasių diagrama, o būsenų kaitų modelis su būsenų modeliu. Nors ir siejasi, bet notacija ir požiūris skiriasi.

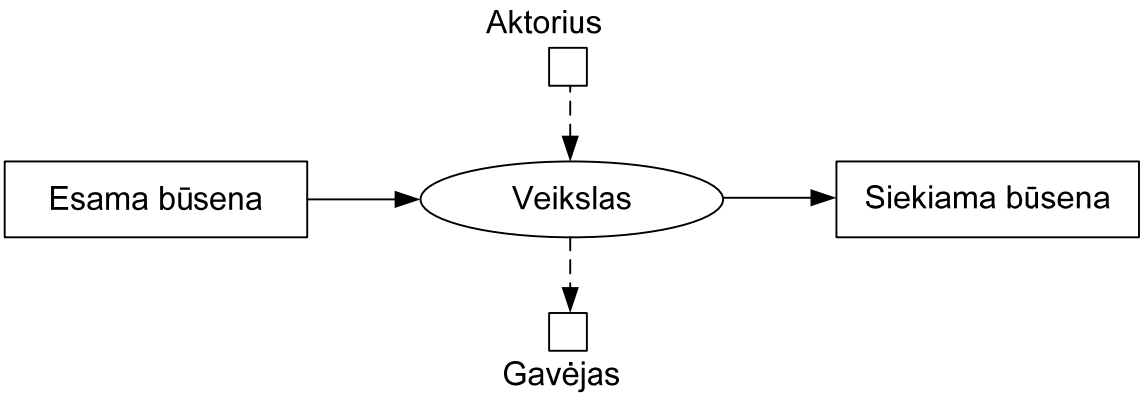
Esybių diagrama artimesnė reliaciniai duomenų bazei nei UML klasių diagrama. Ryšiai tarp esybių labiau grindžiami ne priklausomybėmis (tėvas, vaikas), o santykiais – vienas su daug, vienas su vienu (5 pav.). Galimi, bet rečiau naudojami ir klasių diagramos ryšiai tokie kaip agregavimas, paveldėjimas, yra komponentas, egzempliorius. Tai leidžia UML žinovams ir šalininkams naudotis jiems įprastais ryšiais. Šie objektai gali būti laikomi klasėmis ar komponentais. Kadangi neturi metodų jie organizacijos dinamikos neatspindi ir suprantami kaip statiniai elementai. Todėl juos lengviausia traktuoti kaip reliacinės duomenų bazės lenteles ar galimus statinius sistemos objektus. Todėl esybių modelis netinka griežtai objektiniai modeliavimui.

Nesudėtingose ir vidutinio sudėtingumo sistemose naudojamos reliacinės duomenų bazės, o ypač internetinėse technologijose, todėl kuriant verslo valdymo sistemas lengviausia į objektus žiūrėti kaip į statinius sistemos dalyvius. Šiuo požiūriu R. Gusto modelis turi pranašumą.



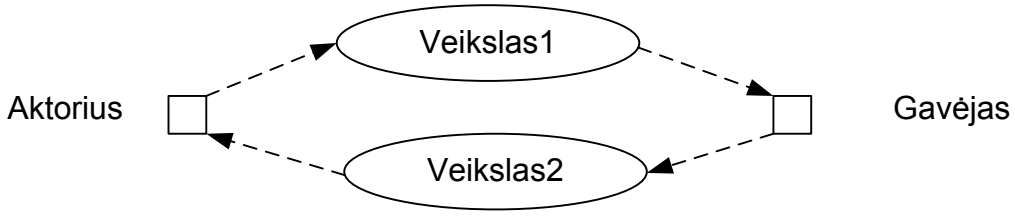
5 pav. Esybių modelio ryšių grafinė notacija

Būsenų kaitų modelis ar UML būsenų modelis? Tarp šių modelių dar didesnis skirtumas nei tarp esybių ir UML klasių diagramos. Būsenų kaitų modelyje dalyvauja netik objektų būsenos, bet veiksmai (gali būti panaudos atvejai) ir aktoriai. Turbūt svarbiausias akcentas aktoriai. Modeliuojant dinaminius procesus, jų ryšius galima interpretuoti dviem požiūriais. [6, R. Gustas] Vienas jų – tai veiksmas pakeičia objekto būseną iš esamos į galimą – perėjimas. Objektai gali pereiti iš esamos būsenos į galimą būseną, o veiksmus (perėjimus) inicijuoja aktoriai (sistemos vartotojai). Kitas požiūris tai – komunikavimas. Čia veiksmai nukreipti į tam tikro tikslo siekimą. Veiksmus reiktu labiau vertinti kaip dinaminius ryšius tarp aktorių (Siuntėjas <-> Gavėjas). Veiksmai taip pat gali keisti objektų būsenas, bet čia esminis akcentas yra ką siuntėjas siunčia ar gavėjas gražina ir koks veiksmas naudojamas tai operacijai atlikti. Apibendrinant būsenų kaitų diagramą galima pasakyti jog tai yra UML būsenų diagramos ir sekų diagramos mišinys. Būsenų kaitų diagramą galima laikyti komunikacinėmis kilpomis. Tipinė komunikacinė kilpa pavaizduota 6 pav. Stačiakampiais žymimos objektų būsenos, ovale – veiksmas. Iš būsenos į veiksmą ir atvirkščiai rodoma būsenų kaitos kryptis, o punktyrine linija – komunikacinis srautas. Šitoks modeliavimas leidžia identifikuoti duomenų šaltinius, išskirti aktorius, susieti juos su atliekamais veiksmais bei specifikuoti būsenas keičiančius veiksmus.



6 pav. Tipinė komunikacinė kilpa

Dar vienas tipinis komunikacinės kilpos atvejis pavaizduotas 7 paveikslėlyje. Tokio tipo diagramos specifikuoja ryšius tarp dviejų objektų siekiant tam tikro tikslo. Komunikaciniai srautai (punktyrinės linijos) dažnai turi ir pavadinimus.



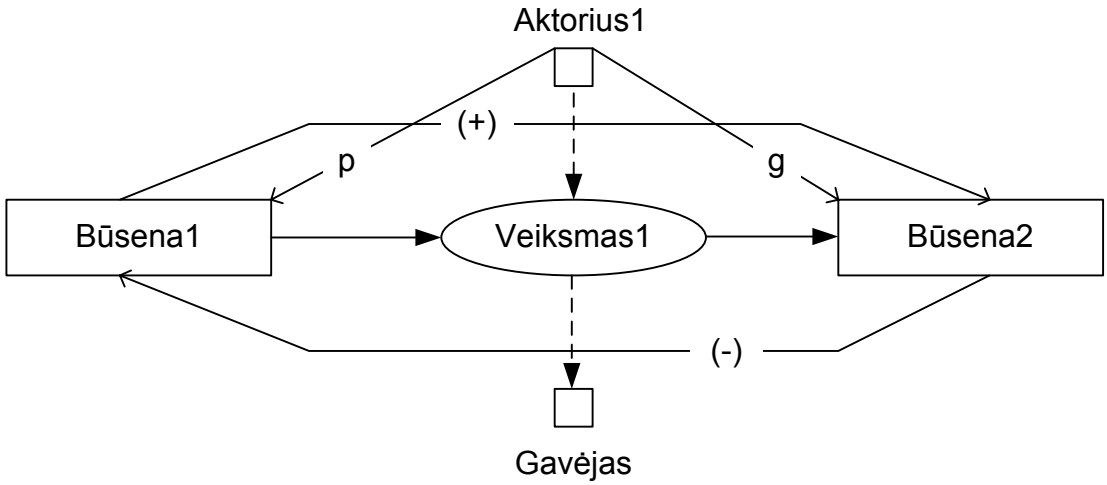
7 pav. Komunikacinė kilpa tarp aktorių

Tarkim jei specifikuojama įmonės ir pirkėjo ryšys tai pirkėjas siunčia mokėjimą (pinigus), o įmonė gražina produktus ar sąskaitą. Tokios specifikacijos labai informatyvios nurodant veiksmus tarp sistemos vartotojų ir pačios sistemos. Panašaus pobūdžio diagramos formuojamas ir iš UML sekų diagramų. Sekų diagramų trūkumas, kad jos sukurtos griežtai objektams bei veiksmai išdėstyti objektų gyvavimo trukmėje. Specifikuojant sistemas, kurios kuriamos internetinių technologijų pagrindu, objektų gyvavimo trukmė nėra esminis akcentas. Jeigu sistema realizuota internetinių technologijų pagrindu bei objektiniu požiūriu, objektai sukuriami ir sunaikinami kiekvienos užklausos (kreipimosi ir scriptą - programą) metu. Galimi ir ilgesni objekto gyvavimo ciklai sesijose. Kadangi sesijos yra pakankamai lėtos ir nelanksčios, jose objektai saugomi tik būtinais atvejais ir nedideliems duomenų kiekiams.

Apibendrinant būsenų kaitų diagramas galima pasakyti, kad komunikacinės kilpos labiau tinka veiklos specifikavimui (ypač jei realizacija planuojama ne objektinė, o pagrįsta internetinėmis technologijomis) nei UML sekų diagramos [10, UML User guide]. Pagrindiniai privalumai - tai informacinių srautų išskyrimas, aktorių išskyrimas, informatyvios aktorių ir sistemos komunikavimo diagramos.

R. Gusto aprašomoje veiklos modeliavimo metodikoje analizuojamos ir pragmatinės priklausomybės – tikslų diagramos [6, R. Gustas]. Tikslų diagramose aktorius ir objektų

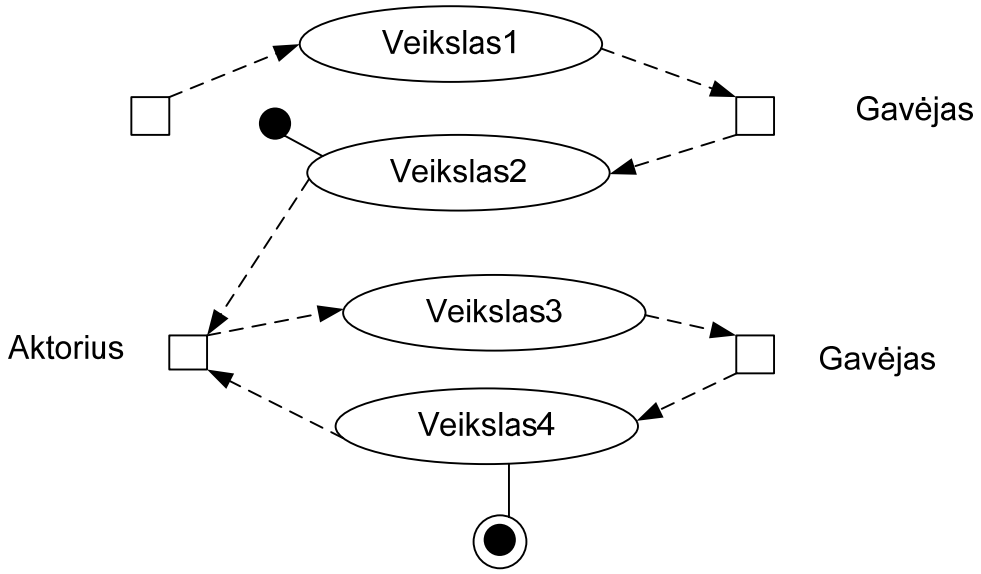
būsenas sieja tikslai, galimybės ir problemos, o vienos objektų būsenos gali neigiamai arba teigiamai įtakoti kitas. Šios sąveikos gali būti vaizduojamos tiek būsenų kaitų diagramose tiek vien tarp būsenų.



8 pav. Problemų ir tikslų bei būsenų sąveikų diagrama

8 paveikslėlyje pavaizduoti tipiniai ryšiai. Rodyklės su simboliai (g) – tai aktoriaus tikslai. Tikslas, kad objektas pasiektų tam tikrą būseną. Atitinkamai rodyklės su simboliais (p) – tai problemiškos objektų būsenos aktoriui. „Būsena1“ teigiamai įtakoja „Būsena2“, o „Būsena2“ – priešingai, neigiamai įtakoja „Būseną1“. Dar galimas ryšys aktoriaus ir būsenos su simboliais (o) – tai reiškia galimą būseną (angl. opportunity).

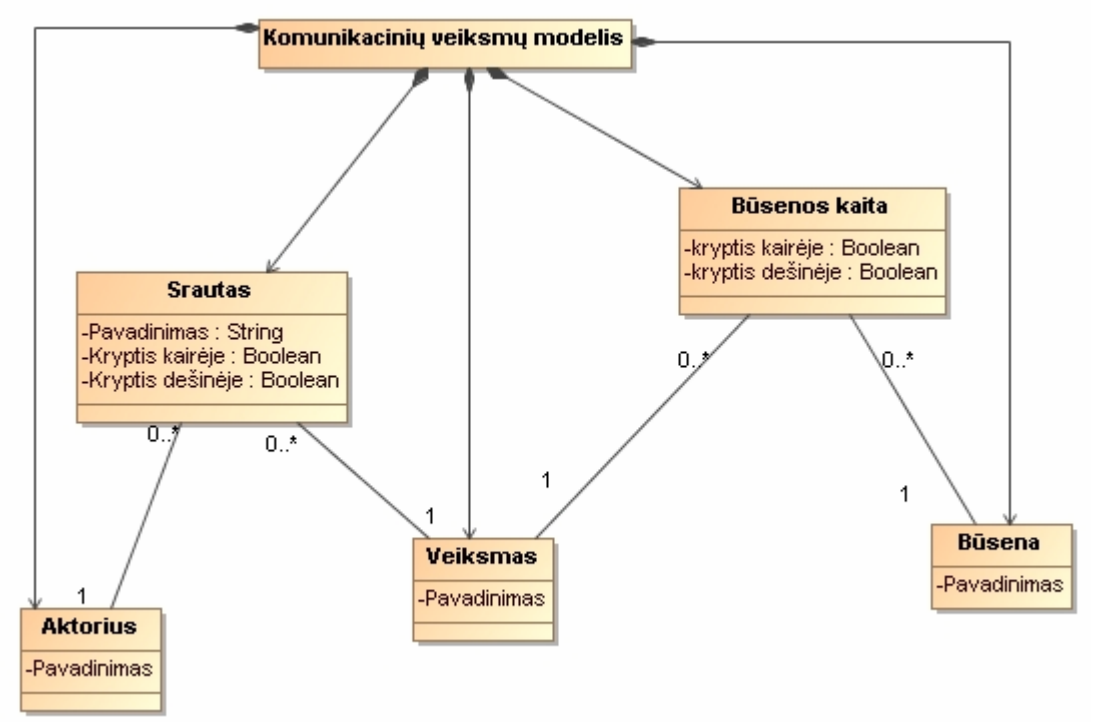
Kadangi transakcijos susideda ir keleto komunikacinių veiksmų tuomet komunikacinių kilpų diagramas (7 pav.) reikia papildyti pradžia ir pabaiga (9 pav.). Sumodeliavus eilę veiksmų būtų galima rasti transakcijų persidengimo taškus, matytųsi jų trukmė. Bet didelis trūkumas – ši notacija nenumato perėjimo sąlygų bei neatskiria realizacijos. Neaišku kurie veiksmai realizuojami kokiais komponentais ir pan. Į panašius klausimus atsakyti ir pagrįsti bus galima pritaikius R. Gusto notaciją modeliavimo įrankiui.



9 pav. Komunikacinė kilpa su transakcija

2.1.6 OO modelių trūkumai ir būsenų kaitų meta modelis

Gana populiarius objektiškai orientuotas projektavimas turi daug gerų savybių modeliuojant dideles, paskirstytas sistemas, bet nelabai tinka veiklos modeliavimui. OO modeliai yra per daug specializuoti ir smulkūs. Dažnai veiklos modeliavime įvyksta nemažai pasikeitimų, o objektai tarpusavyje turi gana tvirtus ryšius, tuomet objektai tampa nestabilūs ir dar labiau priklausomi vienas nuo kito. Komponentinėms projektavimo technologijoms reikia lankstesnio požiūrio, lankstesnio pakartotinio panaudojimo. Akivaizdu, kad objektiškai orientuotos technologijos nepakankamai gerai sprendžia veiklos modeliavimo uždavinius. Geresni veiklos modeliavimo rezultatai pasiekiami komunikacinėmis kilpomis ir panašiais metodais pvz.: Enterprise modeling [6, R. Gustas]. Tačiau šis metodas turi ir didelių trūkumų. Viena jų naudojama gana sudėtinga ir nestandartinė notacija, kurios nepalaiko vaizdinio modeliavimo įrankiai. Kai kurie projektavimo įrankiai suteikia galimybe kurtis savo notaciją. Kad susikurti projektavimo aplinką bus reikalingi projektavimo modelių meta modeliai. 10 paveikslėlyje pateiktas komunikacinių veiksmų arba kitaip būsenų kaitų meta modelis.



10 pav. komunikacinių veiksmų modelis

Komunikacinių veiksmų meta modelis BNF (Backus-Naur forma) išraiška:

<Komunikacinių veiksmų modelis> ::= <Srautas, Veiksmas, Būsenos kaita, Aktorius, Būsena>

<Srautas> ::= <Pavadinimas, Kryptis kairėje, Kryptis dešinėje>

<Veiksmas> ::= <Pavadinimas>

<Būsenos kaita> ::= <Kryptis kairėje, Kryptis dešinėje>

<Aktorius> ::= <Pavadinimas>

<Būsena> ::= <Pavadinimas>

Čia pateiktas meta modelis R. Gusto aprašytos komunikacinių veiksmų modelio. Pritaikant verslo sistemoms jis tūrėtų pasipildyti transakcijoms reikalingais objektais ar sąlygomis.

2.1.7 Internetinės technologijos ir UML

Vis didėjant poreikiui modeliuoti internetinių technologijų pagrindu kuriamas sistemas buvo papildyta UML notacija. Tam tikslui išleistas UML plėtinys WAE (Web Application Extension). Tai nėra kažkas itin naujo, o tiesiog standartiniams UML objektams – klasėms suteikiami stereotipai, kurie atsineša savo notaciją. Populiariausi projektavimo įrankiai turi šiuos plėtinius, o kai kurie tik įtraukinėja į naujas versijas. Pagrindiniai WAE stereotipai:

Server page – interpretuojamas, kompiliuojamas ar kitaip vykdomas puslapis serverio pusėje.

Client page – kliento puslapis. Tai dažniausiai HTML kodas, kuris yra interpretuojamas ir atvaizduojamas kliento pusėje, dažniausiai internetinės naršyklės pagalba.

Form – šis stereotipas apibrėžia duomenų įvedimo, koregavimo formą, kurios pagalba duomenys perduodami serverio puslapiui, kuris atitinkamai gali keisti duomenų bazės duomenis, keisti vartotojo generuojamus puslapius ir t.t.

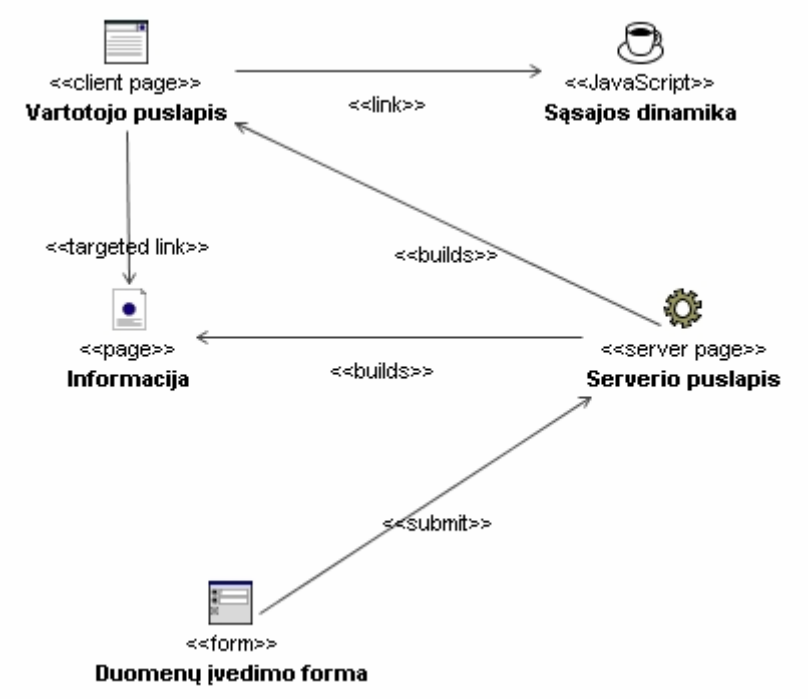
Taip pat išplėtimas turi ir keletą ryšių stereotipų:

Build – tai ryšys dažniausiais tarp serverio ir kliento puslapių. Jis nurodo kad serverio puslapis generuoja vartotojo puslapį

Submit – tai formos duomenų siuntimas kitam objektui.

Link – nurodo ryšį į kitą puslapį. HTML nuoroda.

Kai kurie įrankiai dar turi savų išplėstinių ir įvairesnių stereotipų tiek objektams, tiek ryšiams. 11 pav. pateiktas pavyzdys su pagrindiniais internetinių objektų ir jų ryšių stereotipais. Klasių diagrama nubraižyta Magic Draw 12.1 įrankio pagalba naudojant WAE plėtinį. Panašaus pobūdžio plėtinys yra ir RUP projektavimo procesui. Jame notacija skirta veiklos (dalykinės srities) modeliavimui (12 pav.). Tokie plėtiniai suteikia galimybę vartotojui vizualiai tiksliau atvaizduoti sistemos modelius. Dėl aiškesnio vizualaus vaizdo kyla modelių kokybė, lengvėja realizacija, didinamos kodo generavimo galimybės. Pagrindinis trūkumas – modelių didėjimas. Didinant objektų notacijų kiekį plečiasi ir modeliai. Todėl neretai sistemos modelius tenka skaidyti pagal jų logiką ar ryšius. Atskyrus modelius prarandamas bendras sistemos vaizdas ir atsiranda galimybė klaidoms. Dar vienas trūkumas – šiuos plėtinius turi palaikyti projektavimo įrankiai, o projektuotojai turi su jais susipažinti ir įvaldyti.



11 pav. Klasių diagrama naudojant WAE plėtinį.

2.2 Transakcijos objektiniame projektavime

Net ir populiarūs, objektiškai orientuoti (OO) projektavimo metodai ar metodologijos visą projektavimą perkelia į objektus ir jų panaudojimą, jų būsenas, saugomus duomenis. Objektai yra projektuojami kaip juodos dėžės su griežtai nusakytais įėjimais ir išėjimais. Objektai tarpusavyje sujungiami ryšiais, kurie neretai tampa patys kaip juodos dėžės dėl savo sudėtingumo, perduodamų duomenų sudėtingumo. Projektuojama su mintimi, jog vienas objektas turi atlikti vieną operaciją, atstovauti vienai veiklos sferai ar pan. Verslo procesai apima keletą skirtingų veiklos sričių. Tarkime atliekama operacija „parduoti“, ji turi netik atlikti pardavimą, bet pakeisti sandėlio duomenis, pakeisti buhalterijos duomenis, kasos aparatą, fiksuoti kas parduoda ar pan. Tokius ir panašius procesus tiksliau būtų traktuoti transakcijomis. Nes tai procesas, kuris turi pradžia ir pabaigą. Kad pradėjus pasiektume pabaigą turime atlikti aibę operacijų, kurios turi būti įvykdytos sėkmingai, o jas atliekant nusakomos tam tikros sąlygos, pvz.: kitas operatorius negali parduoti tos pačios prekės. Jei panagrinėtume keletą UML diagramų pvz.: būsenų diagrama, sekų diagrama, pastebėtume, jog procesas nusakomas tokiu požiūriu: kaip nuo objekto pereiti prie objekto, kokį jo metodą iškviesti, kada sukurti objektą, o kada sunaikinti. Nenagrinėjama ar galima šitą metodą kviesti ar galima sukurti objektą. Turbūt daugelis pasakytu, tegu metodo realizacija nusprendžia kokius veiksmus vykdyti. Visu pirma projektavimo metu į metodų realizacija dažniausiai neatsižvelgiama. O kita įsivaizduokime jei kiekvienas metodas turėtų savyje kontrolės sistemą, tuomet greičiausiai programų vykdymo laikas padidėtų dešimtis kartų kas neigiamai įtakotų tiek sistemos darbą tiek jos tobuliną.

Derėtų paminėti, jog nevisas verslo transakcijas galima tiesiogiai kompiuterizuoti, nes juose dalyvauja materialiniai objektai, žmonės, įtakoja nenugalimos jėgos ir pan. Dėl šios priežasties transakcijas reiktų optimizuoti, siekiant, kad jos taptų lengvai realizuojamas, bet nekeistų verslo veiklos , procesų struktūros.

2.3 Verslo valdymo sistemų modeliavimas

Informacinių sistemų skirtų verslui, verslo valdymo sistemų modeliavimas turėtų susidėti iš dviejų pagrindinių etapų, kurie neturi būti atskirti. Vienas jų veiklos modeliavimas, o sekantis sistemos modeliavimas. Sistemos modeliavimas turėtų sekti iš veiklos arba kitaip vadinamos dalykinės srities modelių. Sumodeliavus veiklą iš jos išskiriami sistemos aktoriai, esybės, procesai, o tai palengvina tolesnį sistemos modeliavimą. Modeliuojant sistemą be veiklos specifikacijos gali kilti didelių problemų. Didžiausios bėdos kyla dėl reikalavimų pasikeitimo, o reikalavimų pasikeitimas keičia daug sistemos modelių. Ypač didelės perprojektavimo bėdos kyla jei sistema griežtai objektinė, paskirstyta ar didelės apimties. Modeliavimo problemos, jų sudėtingumas dar priklauso ir nuo modeliavimo eigos bei požiūrio. Objektiškai orientuotas modeliavimas veiklos modeliavimui netinka, dėl stiprių objektų sąsajų. Taip pat kai kuriais momentais sunku realizuoti griežtai objektinius modelius internetinėmis technologijos. Internetinės technologijos stipriai populiarėja ir įgauna naujų formų, panaudojimo galimybių verslo valdymo sistemose. Objektiniam požiūriui alternatyva – komponentinis projektavimas. Komponentinis projektavimas (komponentinis požiūris) daug nesiskiria nuo objektinio. Esminis skirtumas požiūris į objektus. Šis požiūris suteikia daugiau laisvės, objektai tampa mažiau priklausomi tarpusavyje. Komponentinio ir objektinio požiūrio esminiai modelių skirtumai:

Objektinis	**Komponentinis**
Klasių diagramos	Klasių diagrama (DB modelis)
Sekų diagramos	Būsenų diagramos, Srautų diagramos

Galime matyti, kad objektai tampa labiau kaip duomenų bazės lentelės. Jeigu tai reliacinė duomenų bazė joje metodai nesaugomi, todėl reikia labiau detalizuoti jų būsenas, procesus transakcijas. Objektiškai orientuoto požiūrio atvejų sistemos dinamika atspindi sekų diagramos.

Sprendžiant projektų projektavimo ir projektavimo valdymo problemas buvo kuriami įvairus metodai, modeliai, projektavimo sekos. Gana sėkmingas per ilga laiką besivystęs projektavimo proceso standartas RUP. Jis Stipriai siejasi su UML diagramomis, bet neįpareigoja naudoti konkrečius modelius. Tiesiog aprašo darbų sekas, atskirų etapų siektinus rezultatus. Praktikoje galima procesą pildyti naujais modeliais ir notacijomis, bei požiūriu į juos. Kaip pavyzdys galėtų būti veiklos modeliavimas komunikacinėmis kilpomis – R. Gusto metodika [6, R. Gustas]. Taip pat F3 (F kubu) ar MERODE metodika. Šias metodikas tiesiogiai tarpusavyje lyginti ganėtinai sunku, nes visos skiriasi savo idėjomis siekiais ir

notacija. F3 turi nemažai panašumų su R. Gusto aprašytais modeliais ir metodais, o MERODE yra griežtai objektinio požiūrio modeliavimas. Šias metodikas galime palyginti pagal IS kūrimo proceso atžvilgiu ir jų galimybes skirtinguose etapuose [7, R. Butkienė] (1 lentelė).

1 lentelė. IS kūrimo procesų palyginimas

Etapas	F3	R. Gusto	MERODE	RUP
Organizacijos ir veiklos analizė	Apima	Galima naudoti	Pritaikoma	Apima
Sistemos analizė	Apima	Galima naudoti	Pritaikoma	Apima
Sistemos projektavimas	Apima	Iš dalies	Pateikti tik principai	Apima
Sistemos realizavimas	Apima prototipų lygyje		Pateikti tik principai	Apima
Sistemos testavimas	Galima atlikti trasavimą nuo reikalavimų iki realizuojamų komponentų			Apima

Pagal lentelę galime matyti, kad plačiausias procesas yra RUP, o kiti apima tik dalį sistemų kūrimo etapo arba tik nurodo kaip butų galima naudoti.

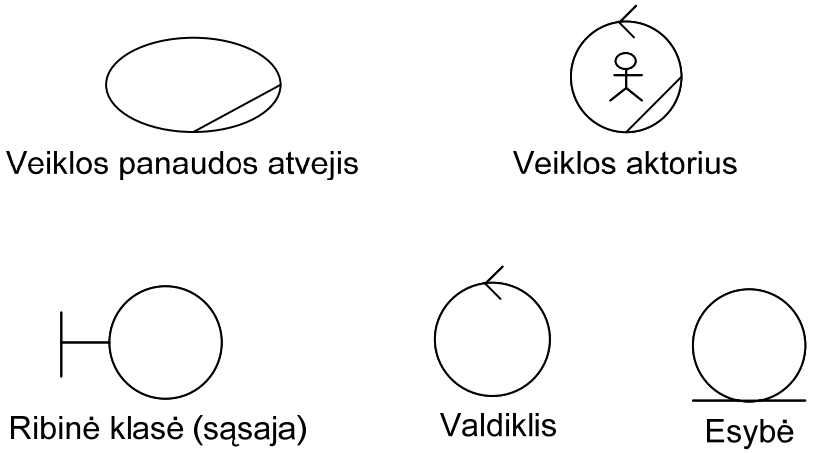
2.4 Rational Unified Process (RUP)

RUP tai gana platus programinės įrangos kūrimo proceso standartas. Jis sudarytas ir paremtas praktiniais pastebėjimais didelio kieko projektų dėka. Šis procesas aprašo visą projektavimo eigą, nusako projektavimo roles. RUP buvo pradėtas seniai (~1980 m.), todėl vystėsi ir tobulėjo kartu su skirtingomis technologijomis ir tapo nepriklausomu ir pilnu programinės įrangos kūrimo procesu [9, R. Dennis Gibbs].

Procesas skaidomas į keturias fazes: pradžia; parengimas; konstravimas; perėjimas [8, RUP]. Visuose etapuose vyksta įvairūs programinės įrangos gyvavimo ciklo procesai: veiklos modeliavimas, reikalavimai, analizė ir projektavimas, realizavimas, testavimas, įdiegimas, konfigūravimas ir pokyčių valdymas, projekto valdymas. Šie procesai skirtingu intensyvumu dalyvauja skirtingose fazėse. Nors RUP yra ganėtinai universalus, neapribotas technologijomis šis kūrimo procesas puikiai tinka dideliems projektams, kurie yra detaliai planuojami, prie projekto dirba didelės komandos. Todėl RUP mažiems projektams gali tapti tiesiog našta.

Kadangi RUP ir UML buvo vystoma lygiagrečiai IBM korporacijos, kurioje tuo metu dirbo vienas iš UML pradininkų - Grady Booch, tad RUP procesas siejasi su UML modeliais arba UML modeliais galima sumodeliuoti praktiškai visus RUP etapus [9 R. Dennis Gibbs].

Į šį projektavimo procesą žiūrint iš verslo sistemų projektavimo pozicijos galima išskirti pirmuosius proceso etapus: veiklos modeliavimas, reikalavimų specifikavimas, analizė ir projektavimas. Šiuose etapuose RUP išsiskiria tuo, kad gana griežtai atskiria veiklos modeliavimą nuo sistemos modeliavimo. Sumodeliavus veiklą dažnai paaiškėja pirminiai reikalavimai sistemai, sudaromas veiklos žodynas. Veiklos modeliavimo tikslas suprasti veiklos struktūrą ir dinamiką, užtikrinti projekte dalyvaujančių žmonių vienodą veiklos interpretavimą, išskirti veiklos aktorius, panaudos atvejus – pasiruošti sistemos reikalavimų specifikavimui. Reikalavimų specifikavimo etape jau dirbama labiau su sistema. Jame tvirtinami sistemos reikalavimai, apibrėžiamas sistemos funkcionalumas. Šiuose etapuose sudaromi veiklos modeliai: panaudos atvejų, veiklos objektų sąveikų modeliai bei sistemos panaudos atvejų modeliai. Analizės ir projektavimo etapuose jau kuriami tikslūs sistemos modeliai ir čia priartėjama prie standartinio UML modeliavimo. Veiklos projektavime ir reikalavimo specifikavimo etapuose naudojama speciali notacija (12 pav.) , kuri leidžia atskirti veiklos objektus nuo sistemos, o analizės ir projektavimo etape naudojama plačiai paplitusi standartinė UML notacija. Po veiklos modeliavimo ir reikalavimų specifikavimo galima projektuoti sistemą. Ir šiame etape galima rinktis metodikas, rinktis projektavimo požiūrį. Todėl RUP proceso nebūtinai laikytis visą projektavimo eigą, o tik reikalavimų sudarymui. Vėliau pagal projekto specifika pasirinkti modeliavimo strategijas.

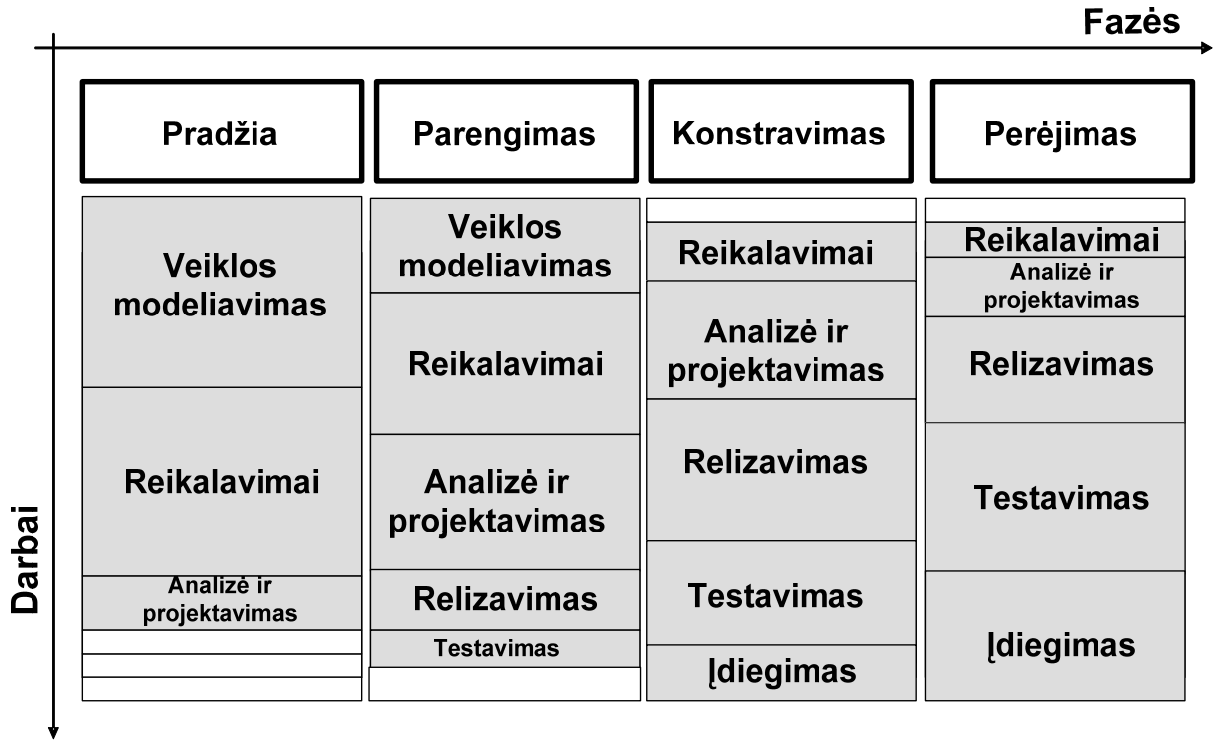


12 pav. RUP veiklos modeliavimo notacija

Analizės ir projektavimo etape reikėtu apsispręsti ir pasirinkti modeliavimo požiūrį: objektiškai orientuotą ar komponentinį, jei sistemai naudinga galima panaudoti kitų metodologijų modelius. Pasirinkus modeliavimo strategiją detalizuojamas sistemos projektas. Reikalavimus pilnai perkėlus į sistemos specifikaciją galima pereiti į realizavimo etapą. Šis ir tolesni etapai mažai siejasi su veiklos modeliais. Tai sistemos specifikavimo, realizavimo ir testavimo etapai. Prieš pradedant realizacija po projektavimo ir analizės etapo priimami sprendimai realizacijai, sistemos architektūrai. Realizavimo etapo tikslas apibrėžti realizacijos

lygius, apibrėžti posistemius bei realizuoti sistemos elementus juos testuoti ir integruoti. Priešpaskutinis etapas paskirstymas. Šio etapo tikslas pateikti galutinę sistemos versiją, įdiegti programinę įrangą, apmokyti vartotojus. Po jo seka testavimo etapas. Šio etapo tikslas patvirtinti sistemos elementų bendradarbiavimą, integraciją, realizuotus sistemos reikalavimus. Sistema gali būti testuojama įvairiausiais požiūriais: dalimis, vientisa ir pan.

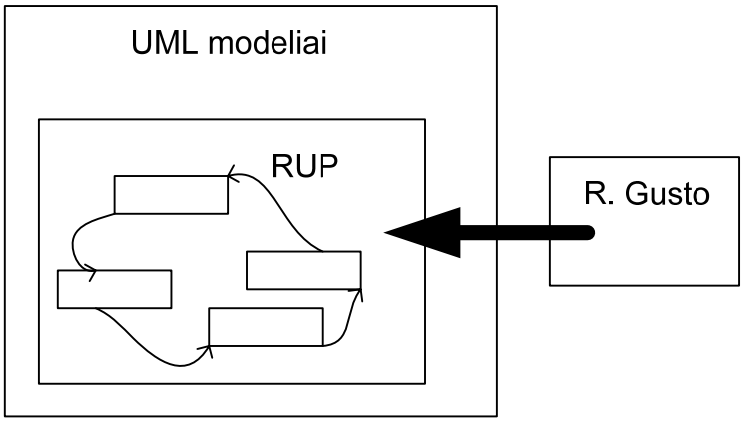
Visi aptarti etapai nėra seka (vienas po kito). Tai iteracinis procesas. Iteracinis viso proceso atžvilgiu bei atskirų etapų. Viso proceso atžvilgiu skirtingu intensyvumu dalyvauja skirtingi etapai (13 pav.).



13 pav. RUP veiklos modeliavimo notacija

2.5 Iteracinio projektavimo proceso papildymas modeliais

Išanalizavus įvairias metodikas, apibendrinus jų galimybes verslo valdymo sistemų modeliavimui labiausiai tinka RUP projektavimo procesas, kurio veiklos modeliavimas papildomas R. Gusto modeliais. Modeliuojant sistemos procesus, sąveikas laikytis RUP ir UML standartų, o į UML įtraukti RUP notaciją, o veiklos modelius tikslinti R. Gusto modeliais.



14 pav. Siūlytina projektavimo architektūra.

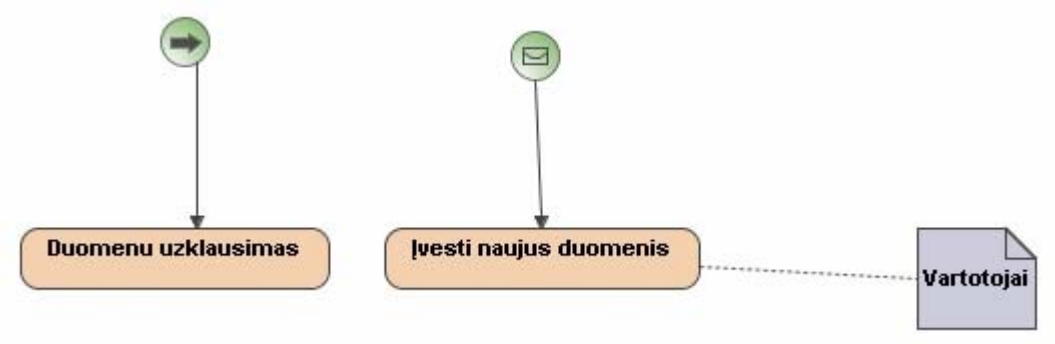
Panaudojus MDA idėjas galima greitesnė sistemos realizacija. MDA susietų, ir panaudotų dinaminius sistemos modelius kodo ar pseudo generavime, nes daugelis kodo generavimo sprendimai nenaudoja dinaminių sistemos modelių, o į programinį kodą perkelia tik sistemos statiką.

2.6 Verslo ir verslo sistemos procesų modeliavimas UML diagramomis

Verslo modeliavimas tai viena iš reikalavimų išgavimo, pirminio modeliavimo dalių, kurios pagrindinis uždavinys analizuoti turimą pradinę informaciją, informacinės sistemos ar programinės įrangos kūrimui [6, R. Gustas]. Tai dažniausiai apima informacijos apibendrinimą, jos procesų specifikavimą su žvilgsniu į verslo niuansus ir specifiką. Verslo procesų, objektų ir duomenų perkėlimas į informacines sistemas, taikomąsias programos, tiesiog į elektroninę erdvę yra vienas pagrindinių uždavinių sprendžiamų įmonėse, tobulinant jų valdymą. Kuriant, projektuojant sistemą daugelis autorių ir metodikų siūlo pradėti nuo verslo modelių, o vėliau projektuoti sistemą. Tai geras požiūris, kuris padeda detalizuoti ir labiau suprasti verslo niuansus sistemų kūrėjams. Tačiau daugelyje įrankių ar modeliavimo kalbų trūksta detalumo perėjimo momente. Analizuojant UML pastebėsime, kad veikla siūloma modeliuoti panaudos atvejų modeliu, veiklos ar sekų modeliais ir dar klasių modeliu, bet į pastarąjį žiūrint gana abstrakčiai. Taip pat galima ir kitais modeliais, bet vėlgi žiūrint į juos tik abstrakčiai. Tarp šių modelių nėra tokio, kuris atskleistų sistemos tikslus ar problemas, trūksta detalumo aktorių, programų ir sistemoje dalyvaujančių objektų veikloje, jų sąveikoje, Objektų dalyvavimo procesuose. Todėl dažnai iš panaudos atvejų modelio yra kuriama klasių ar objektų diagrama. Šis šuolis yra gana didelis ir reikalauja detalaus supratimo apie sistemą, detalios papildomos informacijos apie ją. Sprendžiant šias detalumo problemas buvo sugalvota papildomų notacijų, stereotipų standartiniams UML modeliams. Viena iš standartizuotų OMG notacijų BPMN (Business Process Modeling Notation) [11, OMG]. Ši notacija skirta verslo procesų modeliavimui. Notacija ir jos panaudojimo principas panašus į

UML veiklos modelį tik daugiau žymenų, kurios įgalina detaliau atvaizduoti verslo niuansus. Ši notacija labiau orientuota į verslo sistemos procesų modeliavimą nei verslo procesų modeliavimo. Naujos notacijos naudojimas padeda lengviau skaityti modelius bei juos interpretuoti, bet dar nepadaro aiškaus perėjimo tarp veiklos modelių ir sistemos modelių.

BPMN notacija apima tik veiksmus, įvykius, įvykių „susijungimus“ ir „išsiskyrimus“, perėjimus, veiksmų pradžią ir pabaigą. Šioje notacijoje nedalyvauja nei aktoriai, nei objektai. Pastaruosius galima atvaizduoti tik abstrakčiai kaip artefaktus ar komentarus (15 pav. „Vartotojai“).

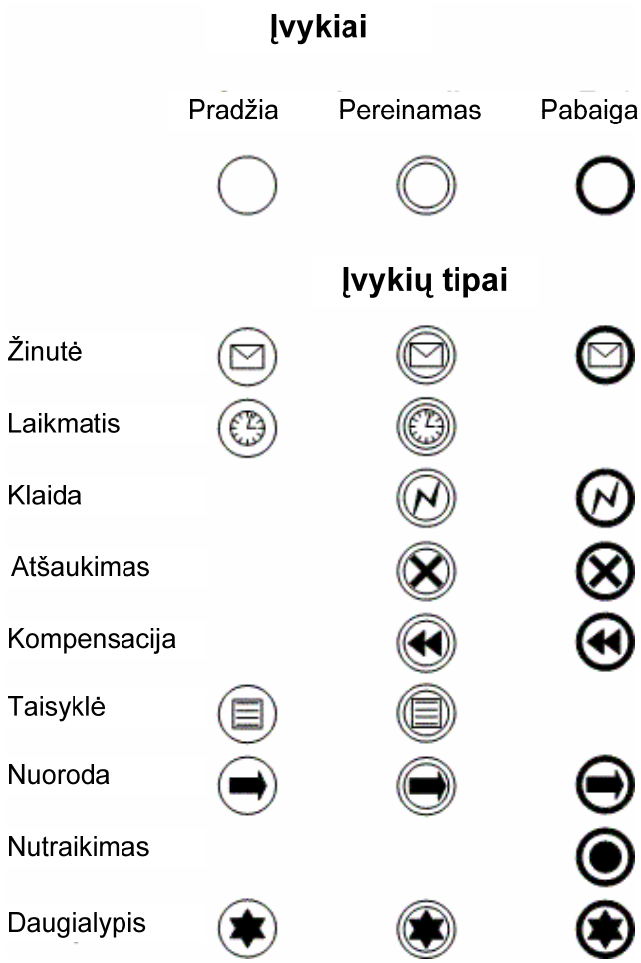


15 pav. BPMN notacija nubraižyta diagrama

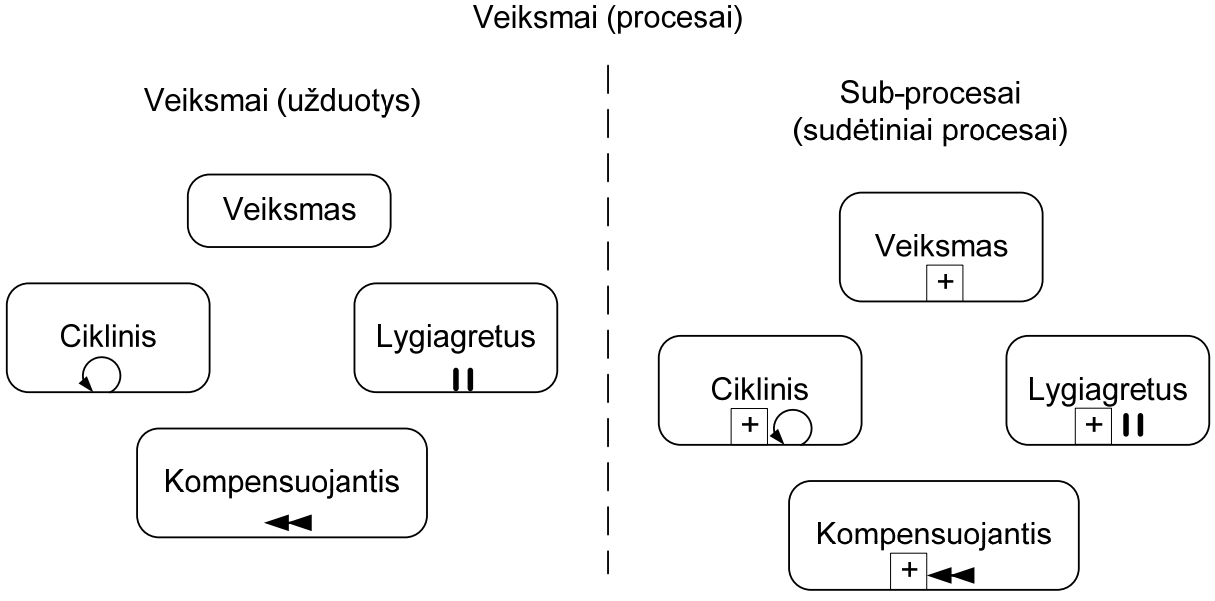
BPMN notacija turi tokias grafinės notacijos klases:

- ✓ Įvykiai
- ✓ Veiksmai arba procesai
- ✓ Sprendimų, įsišakojimo, sujungimo mazgai
- ✓ Ryšiai
- ✓ Artefaktai
- ✓ Procesų juostos (Swimlane)

Labiausiai praturtinta yra įvykių klasė (16 pav.). Joje aprašyta daugelis standartinių situacijų verslo valdymo sistemose, kurių metų pradedama nutraukiama vykdyti tam tikrus veiksmus. Veiksmų ir procesų notacija taip pat pakankamai detali, bet nėra itin išsiskirianti ar pateikianti naujo žymėjimo. Gerokai palengvina ir aiškius modelius padaro sudėtinių procesų žymėjimas (17 pav.).

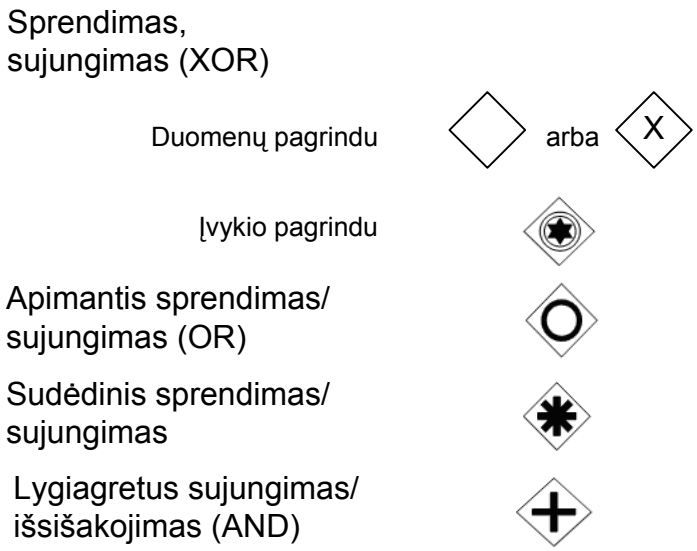


16 pav. Įvykiai BPMN notacija



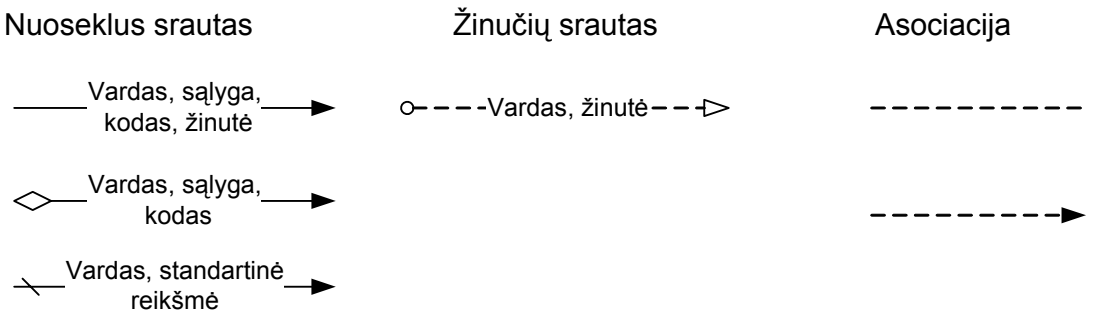
17 pav. BPMN notacija veiksmams (procesams)

Sprendimo, sujungimo, išsišakojimų mazgai

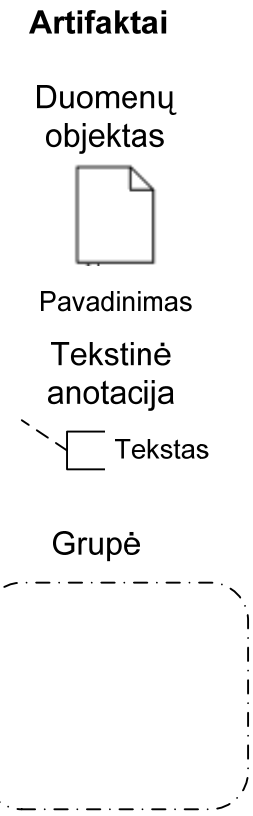


18 pav. BPMN notacija sprendimo, sujungimo įsišakojimo mazgams

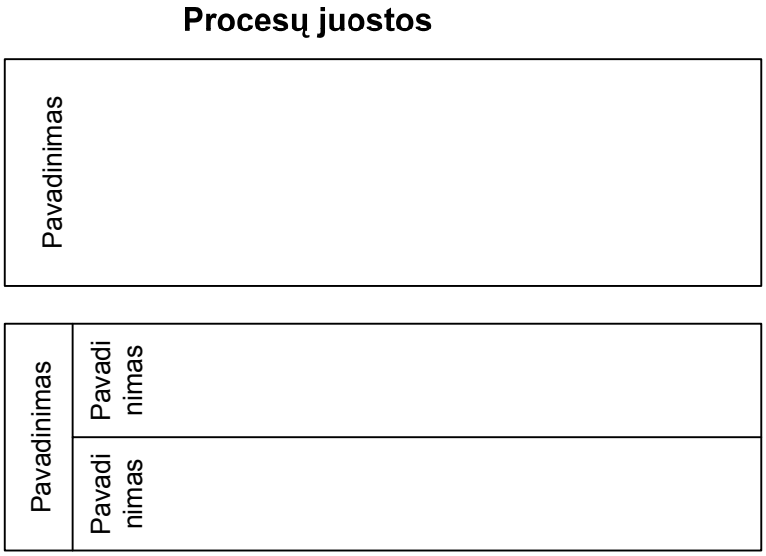
Ryšiai



19 pav. BPMN notacija ryšiams



20 pav. BPMN notacijos artefaktai

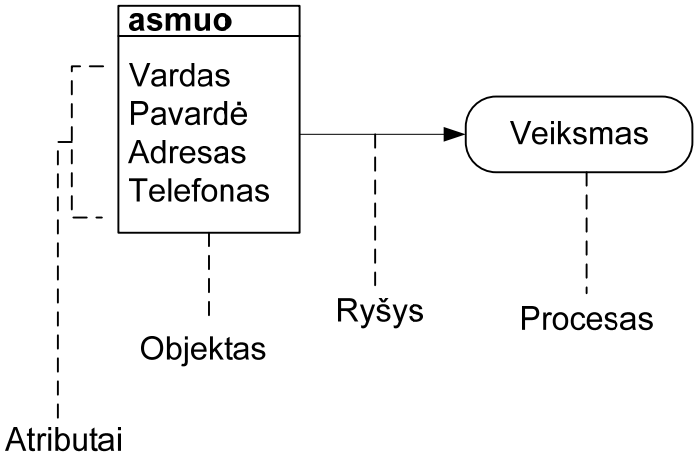


21 pav. BPMN procesų juostos

2.7 UML modelių aprašymas XML failuose

Modeliavimo įrankiai puikiai leidžia modeliuoti sistemas, jų procesus įvairiais požiūriais ir lygiais grafinėje aplinkoje. Kuriant modelius grafinėje, vizualioje aplinkoje yra lengviau juos suvokti ir jais manipuliuoti nei turint eile išraiškų. Daugelis modeliavimo įrankių ir metodu sukurti grafinio modeliavimo pagrindu [12, IBM]. Grafiniai modeliai

netinka modelių apsikeitimui tarp skirtingu įrankių. Panaudojant šiuo modelius sistemose, kurios atlieka testavimą ar kodo generavimą, joms reikia atlikti jų analizę modelių analizę. Tam itin tinka XML standartas skirtas modeliams saugoti ir jais apsikeisti – XMI. Reikalingas tik „žemėlapis“ (*angl. map*) ir kiekvieną diagramos elementą bei jo atributus ir parametrus galima aprašyti XML struktūra (22 pav.). Turint sistemos modelius XML faile galima ne tik juo pernešti į kitą modeliavimo įrankį, bet ir panaudoti kodo generavimui.



22 pav. Diagramos interpretavimas XML kalba

2.8 Verslo modeliavimo galimybių analizės apibendrinimas

Daugelis išanalizuotų metodų ir technologijų susikoncentravusios į greitą programinės įrangos, informacinių sistemų - bendrai sistemų kūrimą. Galima pastebėti, kad šiuo metu tarp įrankių ir metodų vis dar vyrauja verslo modeliavimas ir sistemų modeliavimas kaip atskiros šakos. Nors naujausi analizuoti įrankiai ir naujausi metodai naudoja verslo modelius sistemos projektavime. Vienas ryškiausių verslo modelio naudojimo metodas tai – žiniomis grindžiama IS inžinerija. Jos idėja, postulatai teigia, kad viskas kuriama, tikrinama ir derinama su žinių baze, kuri sudaroma iš veiklos srities modelių – verslo modelių.

Kita problema automatizavimas. Čia MDA siūlo idėjas, bet kartu ir teigia, kad jas nėra lengva praktiškai realizuoti ir pritaikyti. Tai savotiškas iššūkis CASE priemonėms. Taip pat CASE priemonių uždavinys kaip į kompiuterizuotą projektavimo procesą įtraukti kuo daugiau sistemos GC etapų ir kaip suderinti skirtingos informacijos vaizdavimą ir komunikavimą tarp sistemos kūrėjų. Skirtinguose GC etapuose efektyvios skirtingo pobūdžio diagramos, kurios turi būti apjungiamos CASE įrankio.

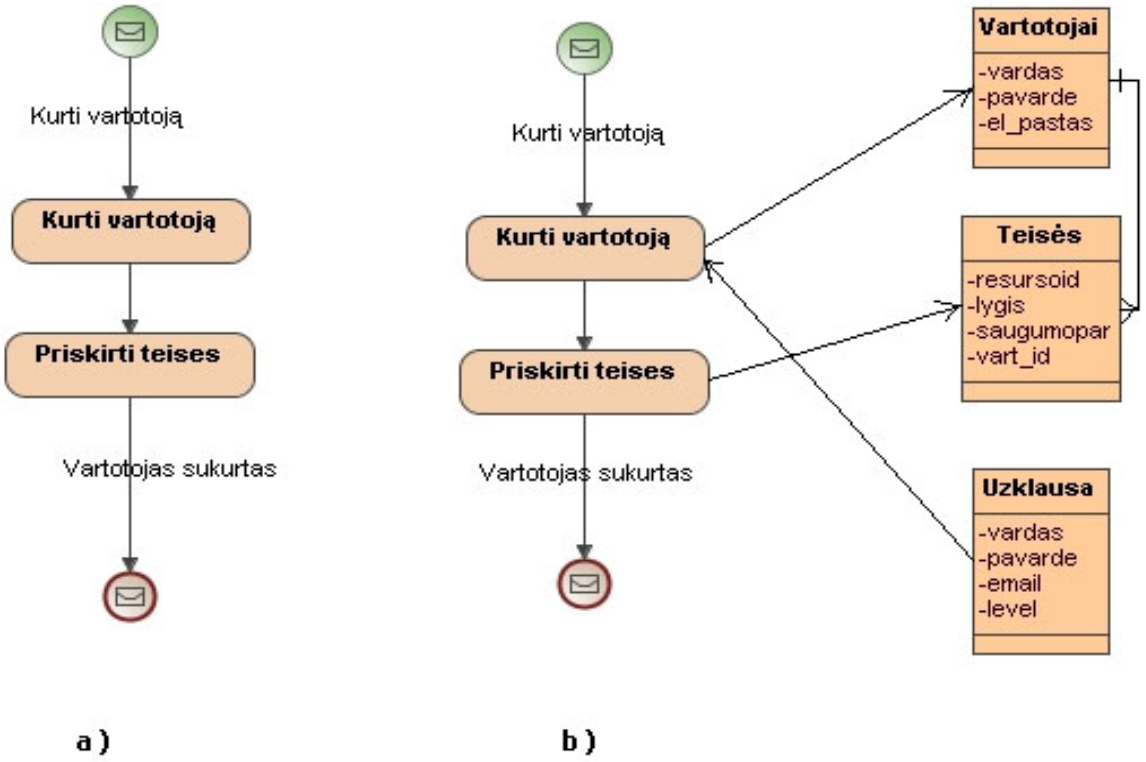
Verslo transakcijų, procesų specifikavimas susiduria su informacijos, modelių, diagramų integracijos problema. Kyla klausimas kaip sistemos analitikams atvaizduoti

surinktą informaciją, kad sistemos projektuotojai ir kūrėjai tinkamai suprastų? Sprendimas yra sistemų GC vystymo metodas, tinkama diagramų notacija, ryšiai ir sąlyčio taškai tarp skirtingu GC etapų bei modelių, kitoks požiūris į tradicinius modelius.

3. Verslo valdymo sistemų procesų ir transakcijų specifikavimo ir jų kodo generavimo tobulinimas

3.1 Duomenų objektais praturtintas BPMN modelis

Verslo procesų modelyje su BPMN notacija vienas pagrindinių įžvelgtų trūkumų – nėra duomenų objektų. Įtraukus duomenų objektus modeliai tampa detalesni ir aiškesni. Modeliuose atsiranda galimybė nurodyti kokia dalykinės srities informacija yra naudojama. Tarkim, kad tam tikras procesas ar veiksmas sukuria vartotoją ir nustato jam teises (23 pav. pavaizduota diagrama su objektais ir be objektų). Nenurodžius duomenų objektų modelis yra nepilnas. Procesų ir objektų ryšiai – kryptiniai, nurodantys informacijos naudojimą. „Kurti vartotoją“ veiksmas turi rodyklę į duomenų objektą „Vartotojai“ ir tai rodo, kad veiksmas keičia objekto duomenis. Objektas „Uzklausa“ – tai duomenys pateikiami procesui, o objektai „Vartotojai“ ir „Teisės“ gali būti duomenų bazės lentelės. Iš „Uzklausa“ objekto duomenys yra paimami, o į duomenų bazės lenteles („Vartotojai“, „Teisės“) rašomi. Iš šios diagramos tampa aišku į kurias duomenų bazės lenteles bus rašomi duomenys ir kokai informacija bus pateikta veiksmui ar procesui. Ši diagrama tampa naudinga netik valdančiai programai parašyti, bet ir vartotojo sąsajos kūrimui.



23 pav. a) BPMN notacija diagrama; b) BPMN diagrama papildyta objektais

Įvedus šiuos pakeitimus į modelius ir modeliavimo įrankius reikia numatyti galimybę įkelti iš klasių modelio objektus arba sukurti juos tiesiog verslo procesų diagramoje. Objektus kurti procesų diagramoje reikia tik tuos, kurių trumpa gyvavimo trukmė (vienos transakcijos, reikalingi tik vienam ar keliems veiksmams). Objektai, kurie yra duomenų bazės lentelės ar dalyvauja keliuose procesuose ar transakcijose turi būti įkeliami kaip nuorodos. Įkėlus objektus iš klasių modelio gali atsirasti objektų parametrų perteklius. Tam tikrose situacijose gali būti naudojama tik dalis objekto parametrų, todėl vienas iš sprendimų būtų į modelį ir diagramą įtraukti tik nuorodą į klasės modelio objektą ir jo atitinkamus parametrus. Kitas sprendimas ant ryšių nurodyti naudojamus parametrus. Tokiu atveju ryšiai taptų itin sudėtingi ir visa diagrama taptų sunkiau skaitoma. Aiškesnis ir geresnis sprendimas – sukurti nuorodas į objektų parametrus. Įtraukus tik reikiamus parametrus būtų visiškai aišku, kas naudojama konkrečiuose veiksmuose.

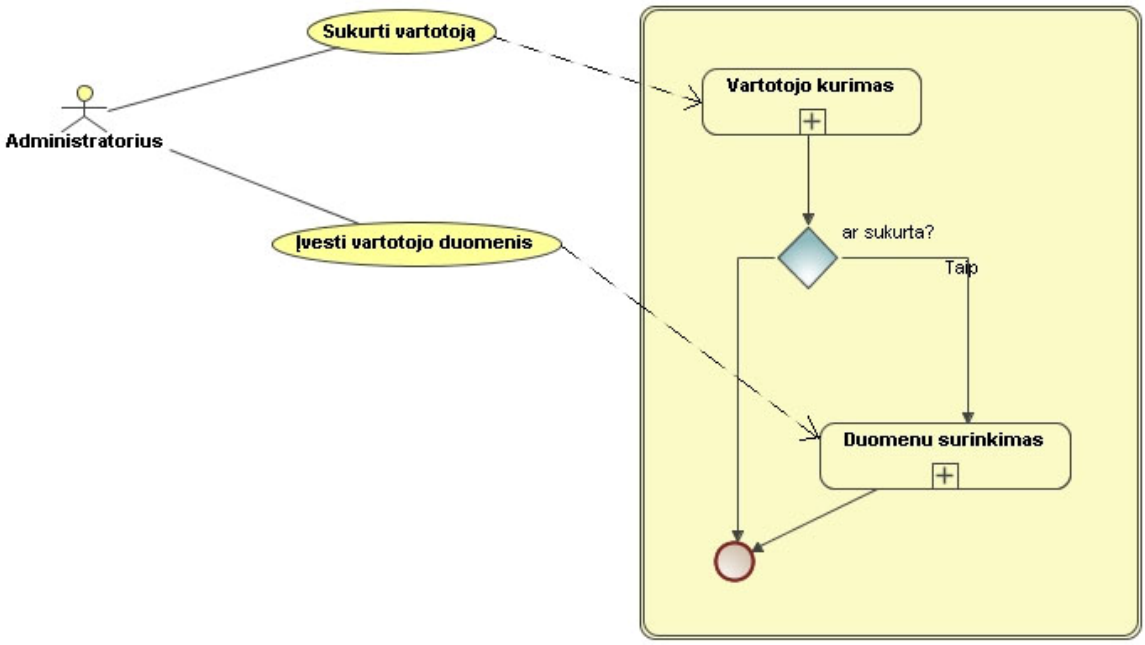
Šis detalizavimas naudingas tik gana atominiams veiksmams, kurie realizuojami viename vykdomajame faile ar nedideliame sistemos programos fragmente. Priešingu atvejų diagramos gali tapti labai didelės ir griozdiškos, panašiai kaip sekų diagramos.

Modeliavimo įrankis (*Magic Draw 12.0*), kuris palaiko BPMN ir buvo naudojamas testuojant, suteikia galimybę įterpti nuorodą į objektą, bet nėra galimybės pasirinkti parametrus bei įterpti ryšius tarp objektų ir veiksmų. 23 pav. pateiktame paveikslėlyje duomenų objektai buvo įterpti ne modeliavimo įrankio pagalba.

3.2 Procesų ir sudėtinių procesų interpretavimas ir sąsaja su kitais modeliais

BPMN standartas turi notacija skirta veiksmas skirstyti pagal jų pobūdį (17 pav.) [11, OMG]. Pagrindinis skirstymas yra ar veiksmas sudėtinis ar ne. Sudėtinis veiksmas savyje gali turėti kitų veiksmų ar procesų. Sudėtiniu procesu modeliavimo įrankiuose dažniausiai laikomas toks, kuris turi ryšį į kitą procesų modelį ar diagramą.

Dėka tokios notacijos galima modeliuoti dviejų ir daugiau lygių procesų diagramas. Pirmas lygis – visos sistemos ar jos dalies procesai. Pirmojo ar kitų lygių išskyrus aukščiausiojo (detaliausio) procesus galima sieti su panaudos modeliu, o tiksliau su jo panaudos atvejais (24 pav.). Panaudos atvejų modelyje vaizduojami pagrindiniai vartotojo veiksmai, kurie nėra detalūs kaip ir pirmųjų verslo procesų lygių procesai ir veiksmai.



24 pav. Procesų diagramos sąryšis su panaudos atvejais ir jų sąveika.

Antras ir tolimesni lygiai - jau verslo valdymo sistemos veiksmų specifikacijos iki atominių veiksmų. Daugelių atvejų apsiribojama dviem arba trimis lygiais. Trys ir daugiau lygių galima ir siūlytini didelėse komercinėse sistemose, kurios apima visą įmonės ar kelių įmonių valdymą ir reikalingi keli specifikavimo lygiai. Detalizavimo gylis gali sumažinti sukurtos sistemos perdarymo ar tikslinimo kaštus, bet tuo pačiu sistemos modelius apsunkina ir padaro didžius bei sunkiai skaitomus. Siūlomas paskutinius du procesų diagramos lygius traktuoti taip: priešpaskutiniame lygyje pateikiami sudėtiniai veiksmai, kurie sprendžia nedidelius verslo uždavinius ir verslo valdymo sistemoje atlieka keletą veiksmų, o realizacija naudingiausia sutelkti keliuose vykdomuosiuose failuose ar paprogramėse. O žemiausią lygį – vienoje paprogramėje, vykdomajame faile realizuojamais veiksmais. Laikantis tokio požiūrio galimi gana efektyvūs programinio kodo generavimo sprendimai.

BPMN notacija numato galimybe sudėtinius veiksmus modeliuoti ir ne atskiroje diagramoje, o tiesiog juos sukeliant į tam tikrą elementą. Jei veiksmas susideda iš kelių nesudėtingų veiksmų tai patogu. Realizuojant kodo generavimą tai jau apsunkiną diagramų analizę ir reikalauja papildomų įvertinimų.

3.3 BPMN transakcijų notacija ir jų elementų interpretavimas

Transakcijų notacija (BPMN notacija) nesudėtinga, bet aiški ir lanksti. Transakcijos turinys išskiriamas specialiu objektu – stačiakampis suapvalintais kampais ir apvestas dviguba linija. Transakcija yra sudėtinis procesas apribotas tam tikromis sąlygomis ir visiems aiškiai suprantamas, turi būti pilnai užbaigtas arba tinkamai sustabdytas [11, OMG]. Pradžiai ir

pabaigai yra specialių simbolių, kurie žymi standartinius įvykius tokius kaip: žinutė, tam tikras pradžios laikas, klaida ir panašiai (16 pav.). Ko nėra standartinėje UML notacijoje ir gali itin pasitarnauti, pvz.: „žinutės“ (message) įvykis. Žinutę gali traktuoti kaip komandą iš išorės: vartotojo ar sistemos. Verslo, o ypač internetinėse sistemose didžioji dalis veiksmų yra atliekama tik esant tam tikrai komandai.

Dar vienas svarbus transakcijoms elementas – tai kompensacija. Daugelis transakcijų, ypač sudėtingesnių turi turėti kompensacijas, o BPMN notacijoje tai yra. Šie elementai žymimi dviguba rodykle (◄◄). Kompensaciniai elementai gali būti tiek atominiai veiksmai tiek sudėtiniai.

Transakcijose dažnai prisireikia kintamųjų ar duomenų objektų, kurie saugotų darbinę transakcijos informaciją, pvz.: sukurto elemento identifikacijos numerį, žingsnio numerį, kompensavimui reikalingas komandas ir panašiai. BPMN standartas [11, OMG] nenurodo tikslaus apibrėžimo kaip naudoti ir kam skirti transakcijos parametrai (*angl. properties*). Todėl juos puikiai galima panaudoti transakcijos duomenų saugojimui. Nurodžius parametrus, kodo generatoriai gali sugeneruoti tokių duomenų sukūrimą, saugojimą ir sunaikinimą darbinėje atmintyje. Internetinėse sistemose itin aktualu, kad šie duomenys būtų įkelti į sesijas (*angl. sessions*).

Apibendrinus, BPMN notacija leidžia pilnai modeliuoti transakcijas, nes turi visus būtinus ir reikiamus elementus. Interpretuojant verslo procesu modelius dviem detalumo lygiais galima tiksliai ir kokybiškai specifikuoti kuriamas verslo sistemų transakcijas, o modelius panaudoti kodo generavimui.

3.4 Kodo generavimas siekiant spartaus kūrimo ir kodo kokybės

Programavimo technologijos nuolat tobulėja ir jau per dešimtmečius išsprendė ir patobulino daug įvairių metodų, kurie šiandien padeda kurti informacines sistemas. Jau gana gerai žinomi ir įsisavinti metodai tokie kaip objektiškai orientuoti metodai, formalaus specifikavimo kalbos, komponentais pagrįstas sistemų kūrimas puikiai tarnauja sistemų kūrime ir padeda spręsti seniai žinomas problemas. Ir tai tik trumpas sąrašas ką būtų galima vardinti. Jau seniai gerai žinoma ir suprantama programinio kodo kokybės problema ir ji intensyviai sprendžiama. Jau išvardinti metodai stipriai tarnauja šioje srityje, bet kol kas nėra tokio, kuris tarnautų idealiai [13, MDA in practice]. Visi metodai (išskyrus ekstremalų programavimą) dažniausiai siekia, kad būtų kuo mažiau rašoma tiesioginio programinio kodo. Siekiant tokių tikslų atsiranda savų pliusų ir minusų:

- ✓ Gaunamas kokybiškesnis kodas;
- ✓ reikalinga mažesnės kvalifikacijos darbo jėga;

- ✓ greitesnis programų kūrimas;
- ✓ įveda didesnius apribojimus;
- ✓ atsiranda „juodos dėžės“ problemos;

Sistemų kūrimas nerašant kodo dažniausiai turima galvoje kūrimas pagrįstas modeliavimu arba laikantis tokios krypties. Pasiekus gana aukštų rezultatų vis labiau plinta metodai pagrįsti MDA idėjomis. Bet koks metodas tai sisteminis ir į tikslą nutaikytas darbas, kurio dėka pasiekiami trokštami rezultatai. Norint sukurti aukštos kokybės produktą pirmiausia reikia geros reikalavimų specifikacijos, sistemos architektūros specifikacijos, sistemos realizavimo, išdėstymo specifikacijos ir t.t. Išpildžius šiuos reikalavimus galima sukurti gana kokybišką produktą, bet, deja, pilnai išpildyti šiuos reikalavimus ne visada pavyksta. O laiko sąnaudos kuriant sistemą rankomis (programuojant) gana didelės. Todėl stengiamasi šį etapą kuo labiau automatizuoti. Egzistuojantys dažniausi kodo generavimo sprendimai remiasi dviem požiūriais:

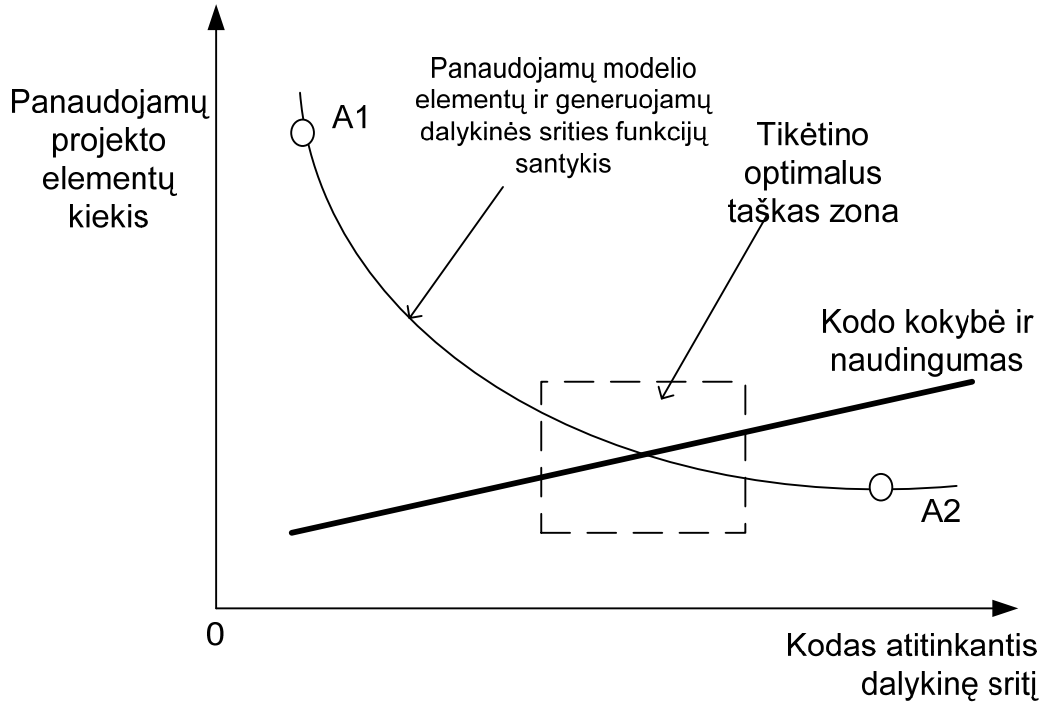
- ✓ procesais pagrįstas metodas, kurie susitelkia į įvykius, kurie atsiranda sistemos eigoje;
- ✓ duomenų struktūra pagrįsti metodai, kurie remiasi sistemos architektūra, joje naudojama duomenų struktūra

Kiekvienas iš šių dviejų metodų gana stipriai skiriasi savo technologijomis, tikslais, notacija, naudojamais modeliais ir t.t. Taip pat skiriasi ir jų rezultatai. Vieni labiau dinaminiai, o kiti statiniai. Sunku būtų išskirti ryškius pranašumus vieno ar kito metodo, nes vienose sistemose didesni pranašumai procesais grįstų metodų, kitose – duomenų struktūra pagrįstų metodų.

Papildžius verslo procesų modelį (su BPMN notacija) galimas programinio kodo generavimo sprendimas, kuris remtųsi tiek procesais tiek duomenų objektais. Tokiu atveju bus sugeneruota tiek statinė, tiek dinaminė sistemos dalis. Su generuoto kodo kiekis ir kokybė labai priklausys nuo generatoriaus ir kodo generavimo strategijos. Siekiant sugeneruoti daug naudingo programinio kodo atsiranda apribojimai generavimui, modeliavimui, o mažinant apribojimus sumažėja kodo kokybė ir naudingumas. 25 paveikslėlyje pavaizduota kodo generavimo priklausomybė. Kodo kokybė ir naudingumas labai priklauso nuo jo panaudojimo, nuo įmonės ar organizacijos kūrimo strategijos. Jeigu įmonėje naudojama specifinė sistemų kūrimo platforma ar įrankiai tuomet standartiniai kodo generavimo sprendimai gali būti mažiau naudingi nei specializuoti. Todėl 25 paveikslėlyje kodo kokybės ir naudingumo kreivės tiksli kryptis priklausys nuo verslo valdymo sistemų kūrimo strategijos ir naudojamų įrankių bei metodų, bet optimalus taškas, bus punktyru pažymėtos zonoje ar kažkur šalia. Taško pokytį gali taip pat įtakoti tarpinių įrankių ar tarpinio kodo panaudojimas

kodo generavimo procese. Tarpiniai įrankiai gali stipriai įtakoti netik kodo kokybės ir naudingumo kreivę, bet ir programinio kodo kiekio kreivę. Tarkim turime situacija kai naudojamas tarpinis įrankis tikslinantis kodo generavimą, vartotojo sąsają ir pan. Tuomet modeliui apribojimai pakankamai maži galima kodo sugeneruoti daug, o generavimą tikslinant papildomu įrankiu didinama jo kokybė ir naudingumas. Bet reikia įvertinti ir tai kad tokie sprendimai gana sudėtingi.

Taške A1 galima naudoti daug modelių, generuoti kiekvieną elementą pagal atskirą strategiją ir tokių atveju bus panaudojamas didelis kiekis elementų prigeneruojama kodo daug, bet kodas neišbaigtas, daug trūksta iki jo tikslinio panaudojamo, kodas sugeneruojamas neatsižvelgiant į modelio elementų ryšius. Taške A2 – tai būtų sprendimas, kuris detaliai analizuoja visa modeli, nustato kiekvienos diagramos detalumo lygį, analizuoja objektų priklausomybes, generuojama tam tikrai platformai skirtas kodas. Tokiu atveju panaudojama mažiau elementų, bet sugeneruotų elementų kokybė aukšta, galimas net tiesioginis jo panaudojimas su minimaliais pakeitimais arba išvis be pakeitimų. Kreivė kurioje atidėti taškai A1 ir A2 gali keistis priklausomai nuo projektuotojų patirties, nuo projekto sudėtingumo, nuo programinės platformos dinamiškumo ir pritaikomumo.



25 pav. Programinio kodo generavimo priklausomybės

3.5 Modelių analizė kodo generavimo sistemose.

Norint generuoti kodą reikia sistemos modelius saugoti tokiu būdu, kad programos galėtų juos suprasti. Akivaizdu, kad grafinė notacija netinka. Modeliavimo įrankiai dažnai turi savo specifines duomenų bazes todėl jie gali informaciją pasiimti nesudėtingai. Bet jei

programinio kodo generavimo sprendimas yra atskiras įrankis šį modelį reikia jam perduoti. Universaliausias ir standartizuotas perdavimo būdas yra XML failais [14, XMI]. Šiuos failus nesunku perduoti, nuskaityti, o turint visą informaciją apie modelius atlikti jų analizę reikalingą kodo generavimui.

3.6 Kodo generavimas remiantis statiniais sistemos modeliais

Dažniausias kodo generavimas modeliavimo įrankiuose yra iš klasės diagramos (duomenų struktūra pagrįstas metodas). Jo principas kiekviena klasę generuoti atskiru failu, o jame surašyti klasės atributus ir funkcijas. Šitoks generavimas palengvina techninį darbą kuriant programinius failus, bet nesukuria jokio funkcionalumo. Detalesniam programiniam kodui sugeneruoti reikia ir detalesnių modelio transformavimo programinio kodu sąlygų. Siekiant aukšto modelių naudingumo generuojant programinį kodą atsiranda taisyklės, kurios suformuoja tam tikrą modeliavimo požiūrį, apriboja modeliuotojo veiksmus. Todėl kodo generavimui būtų itin sunku panaudoti visus projekto modelius. Vengiant apribojimų kuriami aukšto lygio, abstraktūs modeliai, kurie nėra naudojami kodo generavimui. Jie naudojami tik projektuojant sistemą.

Generuojant sistemos dinamiškumą reikalingi standartinių veiksmų aprašymai, standartinių situacijų interpretavimas. Todėl labai sunku sukurti universalią kodo generavimo metodą, nes egzistuoja vienas kitą slopinantys veiksniai (25 pav.). Panaudojus programinius šablonus ar programavimo platformą tampa lengviau sukurti naudingus kodo generavimo sprendimus. Praturtinus verslo procesų modelį su BPMN notacija duomenų objektais galima sugeneruoti gana naudingą programinį kodą (3.1 skyrius). Kad tai padaryti dar yra reikalingas dviejų lygių požiūris į verslo procesų modelius (3.2 skyrius).

3.7 Kodo generavimas remiantis dinaminiais modeliais

Kodo generavimo sprendimai iš veiklos diagramų, sekų diagramų ar kitų sistemos dinamiką atspindinčių modelių ganėtinai reti. Šie sprendimai yra sudėtingesni dėl savo specifikos, dėl to kad nėra tokios diagramos, kuri lengvai ir pilnai leistų modeliuoti sistemos dinamiką. Sprendimus apsunkina netik modelių gausą, bet ir pati dinamikos (veiksmų) prigimtis: kiekvienoje situacijoje tie patys veiksmai skiriasi. Kaip sprendimas ir išeitis yra šablonai, standartizuotos situacijos. Šablonai taip pat nesuteikia galimybės sukurti viską, bet veda prie tikslo – gauti kuo daugiau kokybiško ir naudingo programinio kodo.

3.8 Kodo generavimo strategijos įtaka programinio kodo kokybei

Generuojant kodą remiantis duomenų struktūromis palengvina techninį darbą, gaunamas kokybiškesnis kodas, pagreitinamas kūrimas. Itin geri rezultatai pasiekiami, jeigu kodas generuojamas tam tikrai programinei platformai tuomet į kodo generavimą įtraukiama sutelkta patirtis. Programinė platforma taip pat turi standartinius veiksmus ir operacijas, kurios yra ištestuotos, diktuoja kūrimo strategiją ir greičiau atveda iki siektinų rezultatų. Tiesioginis programinio kodo generavimas iš modelio nėra geriausias sprendimas. Geresnių rezultatų pasiekiama, jeigu kodo generavime dalyvauja dar vienas etapas – tarpini įrankis. Labai sunku gana abstrakčiuose modeliuose nurodyti visas veiklos ypatybes ypač susijusias su vartotojo sąsaja. Todėl siūloma panaudoti tarpinį kodą ar įrankį, kuris leistu patikslinti tam tikrus nustatymus remiantis modelio informacija. Tiesiogiai generuojant kodą iš sistemos modelių gautume kur kas prastesnės kokybės, su mažesniu funkcionalumu programinį kodą. Jeigu turėtume itin detalų modelį tuomet generatorius taptu itin sudėtingas ir su daug apribojimų, kas lemtu taip pat prastesnę kokybę ar mažesnį funkcionalumą.

Šablonų naudojimas diktuoja gana griežtus apribojimus, todėl skirtingi programų kūrėjai gali naudoti skirtingus šablonus. Dėl šios priežasties taip pat naudinga įvesti kelių etapų generavimą. Pirmame etape atliekama modelio analizė, kuri pateikia reikiamą informaciją kodo generavimo įrankiui, kuris leidžia nustatyti papildomus pasirinkimus tokius kaip: duomenų vaizdavimo išdėstymas, vaizduojami laukai, naudojami filtrai rikiavimas, paprogramės iškvietimo būdas ir t.t. Šis tarpinis įrankis gali būti keičiamas. Keičiami jo šablonai, bibliotekos, papildomas naujomis galimybėmis ir t.t. Jeigu šios sąlygos išpildomos kiekvienas sistemų kūrėjas gali susikurti sau reikalingus šablonus, bet atgalinis procesas (iš kodo į modelį) tampa praktiškai nebeįmanomas. Jeigu kodas pakankamai kokybiškai generuojamas tuomet nebereikalingas grįžimo atgal nuo kodo prie modelio, nes generuojant kodą iš naujo praradimai būtų nedideli.

3.9 Strategija generuojant transakcijų programinį kodą

Kodo generavimo pagrindą sudaro verslo procesu modelis su BPMN notacija, papildytas duomenų objektais iš klasių modelio. Strategija skirta generuoti PHP programinį kodą ir jam skirtos vartotojo sąsajos (HTML, CSS ir JavaScript). Svarbu tai, kad analizuojama dviejų ir daugiau lygiu logika nubraižytos diagramos. Programinis kodas generuojamas tam tikrai programavimo platformai, kuri turi standartines bibliotekas, programavimo logiką, taisykles, joje naudojami tam tikri šablonai. Pasirinkti tokie apribojimai, nes prie tokių apribojimų yra sukuriama didžioji dalis (daugiau kaip 60% - duomenys iš įmonės, kuri naudoja minėtą programavimo platformą) vidutinio sudėtingumo verslo sistemos.

3.9.1 Programinio kodo generavimo atskaitos nustatymas

Vienas pirmųjų programinio kodo generavimo žingsnių – nusistatyti kodo generavimo atskaitos tašką. Šis atskaitos taškas turėtų būti pirmo lygio diagrama (dviejų lygių modelyje arba priešpaskutinė daugiau kaip dviejų lygių diagramoje), kurioje yra transakcija su sudėtiniais procesais. Lygio nustatymas vykdomas ieškant diagramos, kuri neturi sudėtiniu procesų ir į ja vedanti diagrama bus pirmo arba žemesnio lygio. Neradus tokios diagramos galima ieškoti diagramos su transakcija, kuri turi sudėtinių procesų ir patikrinti ar sudėtiniuose procesuose nėra daugiau sudėtinių procesų. Jeigu nėra atskaitos taškas rastas. Neradus ir pagal tokią seką, kodo generavimas tampa nebeįmanomas. O modeliavimo metu greičiausiai nebuvo laikytasi dviejų lygių modeliavimo strategijos. Jeigu randama antro lygio diagrama su transakcija, kuri turi sudėtinių procesų galime toliau vykdyti kodo generavimą. Tuomet kiekvienas transakcijos procesas ar sudėtinis procesas bus atskiras vykdomasis failas. Transakcijos parametrai (*angl. properties*) bus traktuojami kaip sesijos kintamieji, kurie bus reikalingi visos transakcijos metu ir sunaikinami užbaigiant transakciją.

Galimi ir didesnėse sistemose tikrai bus keli programinio kodo generavimo atskaitos taškai. Radus kelis atskaitos taškus, kiekvieno taško generavimas turėtų būti vykdomas atskirai.

3.9.2 Perėjimų tarp procesų interpretavimas

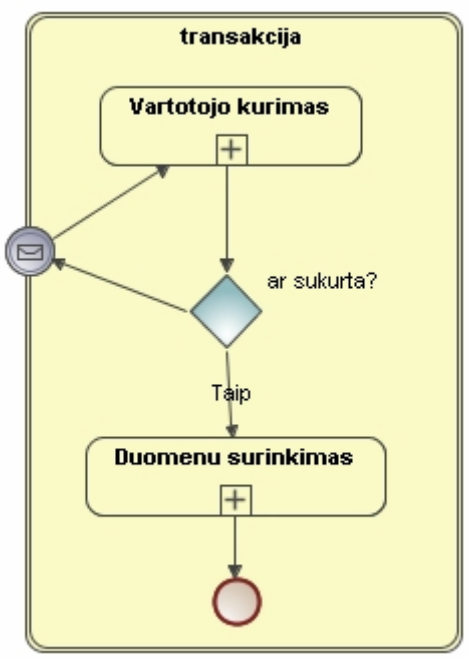
Perėjimą nuo į kitą vykdomąjį failą gali apriboti sąlygos, tam tikrų laukų ar parametrų tikrinimas. Esant teisingai sąlygai perėjimas aiškus – kreipinys į kitą failą. O esant neigiamam atsakymui galimos trys situacijos:

- ✓ einama į transakcijos pabaigą;
- ✓ pereinama prie kito proceso, kuris greičiausiai, bet nebūtinai, bus kompensuojantis arba tinkamai užbaigiantis transakciją procesas;
- ✓ grįžtama į tą patį failą per vieną iš pereinamųjų (*angl. intermediate*) įvykių elementų (16 pav.);

Pereinant tiesiog į transakcijos pabaigą, kodo generavimas apsiriboja sesijos kintamųjų pašalinimu. Po šio perėjimo tikėtina, kad sistemos darbas nebus baigtas, o nukreipiama į pradinį puslapį ar kitą veiksmų pasirinkimo puslapį. Šitas momentas labai priklauso nuo vartotojo sąsajos subtilybių ir nėra verslo sistemos pagrindinis turinys, todėl paliekama rankiniam programavimui.

Pereinant prie kito proceso esant tiek neigiamai sąlygai, tiek teigiamai sąlygai generuojamas kreipinys į kitą programinį failą.

Perėjimas tarp procesų per pereinamąjį elementą suteikia daug įvairių interpretavimo ir realizacijos galimybių (26 pav.). Tikėtini atvejai - tai klaidos ar informacinio pranešimo parodymas, komandos nusiuntimas. Jeigu panaudojamas klaidos ar žinutės perėjimo elementas generuojama žinutė, o kitu atveju paliekama programuotojui pačiam įrašyti perėjimo programinį kodą.



26 pav. Transakcija su pereinamuoju elementu

Perėjimo tarp procesų metu galimas ir lygiagretus procesų įsišakojimas. Kadangi kodo generavimo strategija analizuojama internetinėms sistemoms ir PHP kodo generavimui čia lygiagretumo nelabai yra. Tokiu atveju tiesiog vienas po kito turėtų būti iškviestos abu failai. Visi išskyrus vieną (nebūtinai taip, bet tikėtina pagal tokių sistemų logiką) neturės vartotojo sąsajos, o tik atliks atitinkamus veiksmus, galbūt iškvies dar kitus failus – suveiks panašiai kaip tinklo servisas. Viena iš šakų gali turėti vartotojo sąsają, kuri prašys tolimesnių vartotojo veiksmų transakcijos vykdymui.

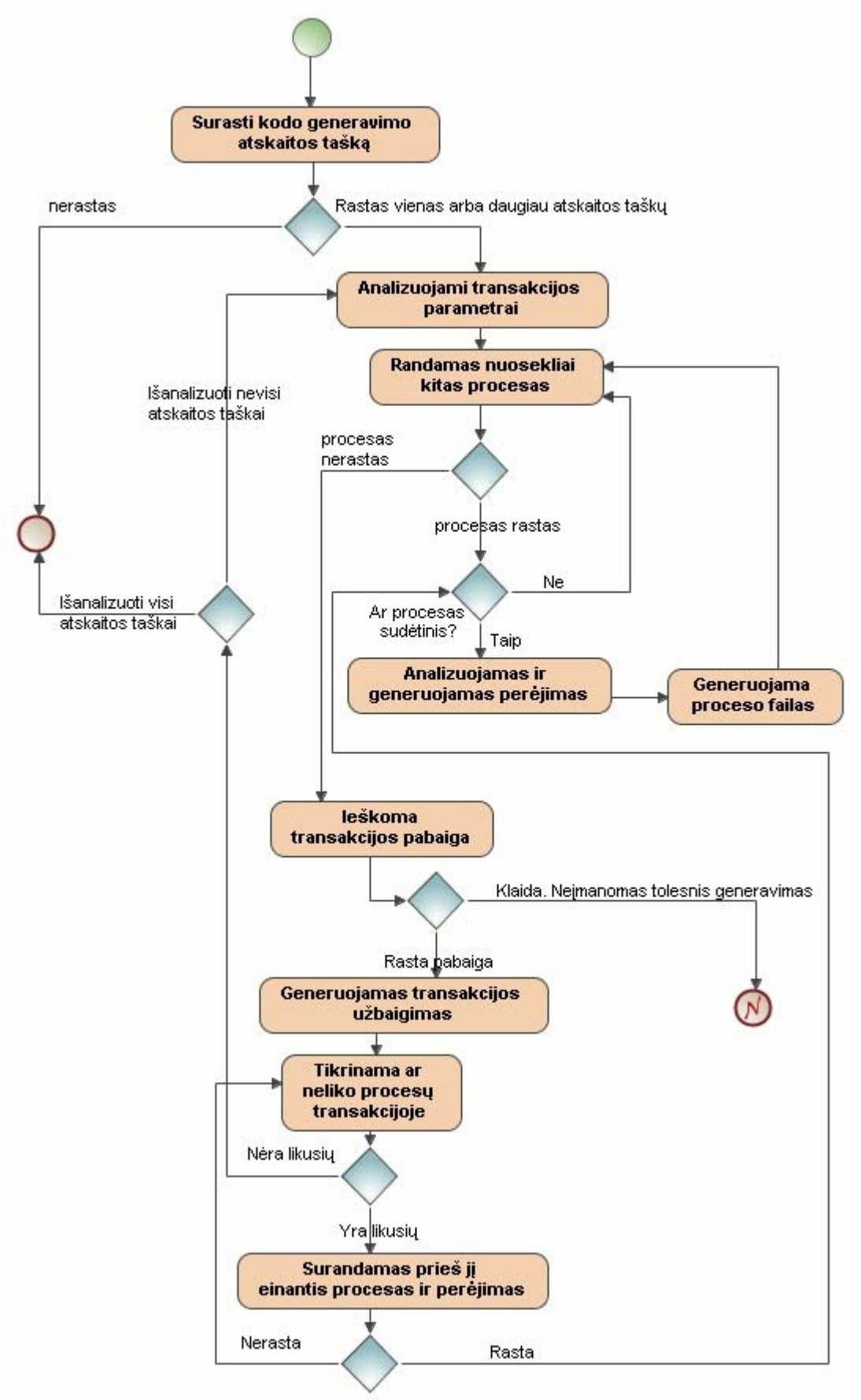
3.9.3 Pirmojo arba prieš paskutiniojo lygių generavimo rezultatai

Pirmojo lygio diagramos generavimo rezultatai bus programinio kodo failas kiekvienam procesui ar sudėtiniam procesui, o jame aprašytas perėjimų logika bei sesijos kintamųjų sukūrimas, sunaikinimas. Toliau turi būti analizuojamas kiekvienas sudėtinis procesas atskirai ir failas papildomas programiniu kodu, o jei reikia sukuriamos papildomi failai tokie kaip vartotojo sąsaja ar pan. Kada šie failai sukuriami, priklauso nuo generavimo

strategijos. Teisinga ir logiška būtų, jeigu failai būtų sukuriami tik paskutiniame programinio kodo generavimo etape.

Pirmame kodo generavimo žingsnyje, duomenų objektai nėra naudojamai. Jeigu jie ir specifikuoti diagramoje, tai generavimui vis tiek įtakos neturės, o bus tik informacija detalizuojant procesus aukštesniame lygyje. 27 paveikslėlyje pateiktas kodo generavimo procesas. Jis apima visų atskaitos taškų generavimą ir vienos transakcijos visų procesų generavimą. Pastebėtina, kad kiekviena transakcija privalo turėti vieną ar kelis pabaigos taškus. Priešingu atveju generavimas neįmanomas.

Nuosekliai einant per procesus ir generuojant programinį kodą, tikėtina, kad nebus aplankytos visos šakos. Todėl pasiekus bent vieną pabaigą reikia atlikti nesugeneruotų procesų paiešką. Radus - pratęsti generavimą. Suradus likusius procesus būtina, kad į jį vestų iš jau sugeneruoto proceso, nes reikalinga perėjimo logikos analizė. Jeigu nerandamas toks procesas iš kurio pereinama į nesugeneruotą, jis tiesiog atmetamas.



27 pav. Programinio kodo generavimo proceso diagrama

3.9.4 Antrojo arba paskutinio lygio programinio kodo generavimo strategija

Šiame lygyje pateikiama detaliausia sistemos specifikacija. Joje jau nebegali būti sudėtinių procesų. Diagramoje turėtų, bet nėra būtina panaudoti duomenų objektus. Veiksmai turi būti susiję su viena konkrečia užduotimi. Gana griežti reikalavimai, kurių dėka galima sugeneruoti gana pilno funkcionalumo verslo sistemos elementus. Duomenų objektai nusako vartotojo sąsajoje naudojamus duomenų įvedimo elementus, taip pat vaizduojamus duomenis. Procesų diagrama nusako veiksmus ar veiksmų sekas.

Kadangi kodo generavimas analizuojamas PHP programavimo kalbai veiksmų atlikimas vykdomas esant tam tikrai užklausai, o vykdytinų veiksmų išrinkimas dažniausiai nustatomas pagal užklausos parametrus ir pateiktus duomenis. Todėl šio generavimo atskaitos taškas(-ai) bus pradžios įvykiai. Paprasčiausi, kurie generuojami tiesiog kaip komandos tai: žinutės, taisyklės, nuorodos ir standartinis pradžios elementai. Laiko elemento įgyvendinimas, kiek sudėtingesnis, nes jis turi būti vykdomas esant tik tam tikram laikui. Šie įgyvendinimai serveriuose realizuojami serverio paslauga angl. vadinama cron-job, todėl kodo generavimui nelabai tinka. Analogiškai su lygiagretumo arba kitaip daugialypiu elementu kodo generavimo sprendimas itin sudėtingas.

Akivaizdu, kad iš dinaminių elementų galima sugeneruoti veiksmų-komandų išrinkimą bei jų vidinio vykdymo seką. Detalesniam generavimui reikalingi duomenų objektai. Pagal juos galima sugeneruoti įvedamų laukų duomenų tikrinimą, duomenų nuskaitymą iš užklausos ir duomenų skaitymo ar įrašymo į duomenų bazę užklausas. Tai gana pilnas programinio failo generavimas. Lieka nesugeneruoti tik konkretūs kiekvieno veiksmo-proceso veiksmai ir jo niuansai. Norint sugeneruoti gana išbaigtą programinį failą reikalinga nemažai šablonų ir jau paruoštų bibliotekų. Tai sukelia tam tikrų apribojimų, bet leidžia generuoti kokybišką programinį kodą.

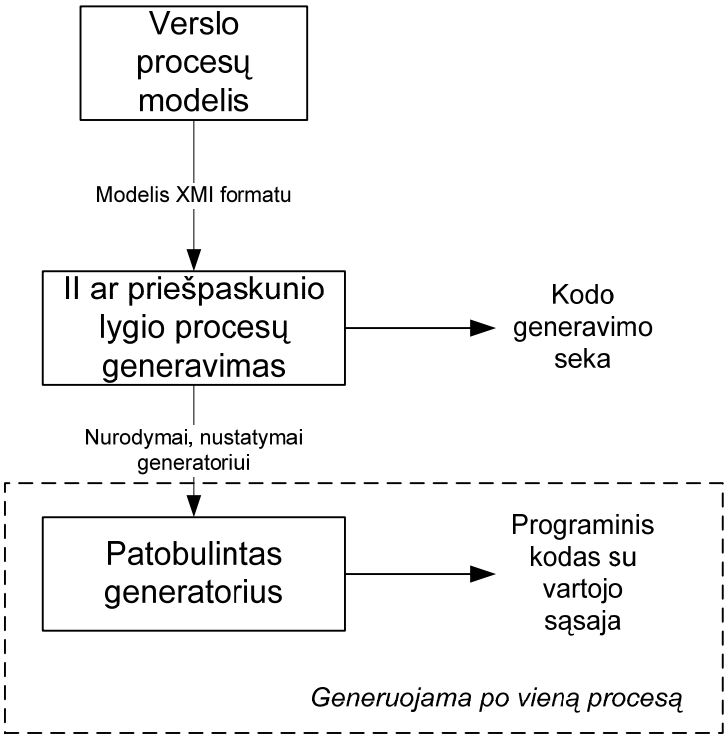
Kadangi šis generavimo etapas pakankamai detalus jį galima atlikti ne tik remiantis sistemos procesų modeliu, bet ir tikslinti papildomais nustatymais, specialiai pritaikytais generuojamai programinei platformai. Nustatymai gali apimti išdėstomos informacijos pobūdį, naudojamas bibliotekas, vartotojo sąsajos langotipą ir kitus papildomus elementus.

3.10 Kodo generavimo praktinės realizacijos galimybių analizė

Tyrimo eiga ir siekiamus tikslus įtakojo asmeninė patirtis dirbant su programavimo platforma, kuri parašyta PHP programavimo kalba, kuri turi pagrindinių veiksmų bibliotekas, vartotojo sąsajos šablonus ir kūrimo taisykles. Šiai platformai yra sukurtas kodo generatorius, kuris generuoja kodą remiantis tik duomenų baze ir vartotojo nustatymai. Nustatymus riboja

generatoriaus funkcijos ir galimybės . Jo rezultatas yra PHP failas ir jam skirta vartotojo sąsaja.

Į generavimo procesą siekiant įtraukti verslo procesų modelį buvo išanalizuotas ir sukurtas XMI failų skaitymas, kodo generavimo atskaitos taško paieška bei generuojamų procesų ir perėjimų tarp jų analizė. Pavyko dalinai realizuoti kodo generavimo algoritmą iš pirmojo arba priešpaskutiniojo lygio diagramų. Kadangi realizaciją riboja programavimo platforma, tad generavimo procesas nėra visiškai universalus. Reikalingi gana korektiškai (laikantis aprašytų taisyklių) specifikuoti procesai. Praplėsta programinio kodo generavimo seka pavaizduota 28 paveikslėlyje.



28 pav. Kodo generavimo realizacijos seka

Siekiant išsiaiškinti galima šio modelio naudingumą ir įtaką programinio kodo generavimui buvo apklausti programuotojai ir projektų vadovas, kurie dirba su minėta programavimo platforma bei jai skirtu generatoriumi. Jų buvo paprašyta eksperimentiškai, iš savo patirties ar kitaip argumentuotai įvertinti verslo valdymo sistemų kūrimo pranašumus, kūrimui naudojant programinio kodo generavimo įrankius, naudojant programavimo platformai skirtą įrankį bei patobulintą kodo generavimo procesą.

Įverčiai pateikti vertinimo rėžyje nuo 1 iki 10, kur 1 atitinka programinį kodą parašyta ranka. Kuo didesnis įvertis tuo geriau, išskyrus perkūrimo kaštus. Apibendrinti apklausos rezultatai 2 lentelėje.

Iš rezultatų galime spręsti, kad įtrauktas transakcijų generavimas nesuteikia itin didelio pranašumo kodo kokybei. Taip yra greičiausiai dėl to, kad jis neįneša daug naujų funkcijų ar daug didesnio programinio kodo pilnumo ir funkcionalumo. Didžiausia įtaka sistemų kūrimo spartai. O jei sistema kuriama sparčiau, bet taip pat kokybiškai tai jau naudinga.

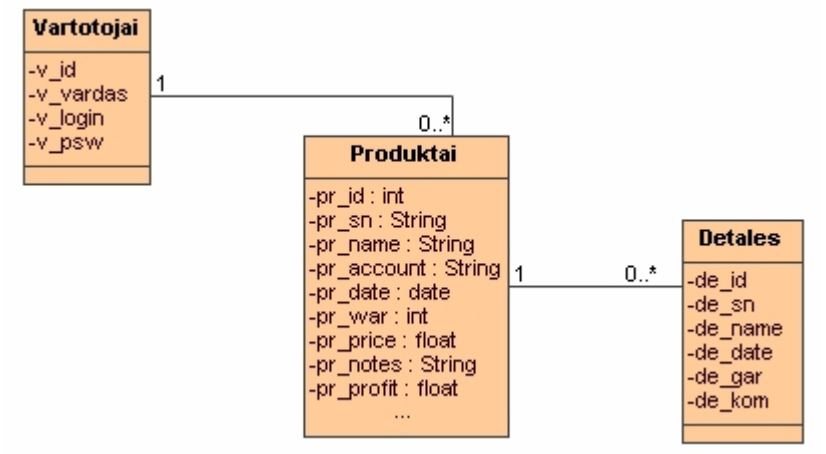
2 lentelė. Kodo generavimo apklausos rezultatai

	Naudojant modeliavimo įrankių generavimą	Įmonėje naudojamas generatorius	Kūrimas naudojant tarpinį įrankį transakcijų generavimui
Sistemų kūrimo sparta	3 – 4	8	9
Bendra sistemos kodo kokybė	2	7	8
Generuojamo kodo kokybė	8 – 9	8	8 – 9
Sugeneruoto kodo naudingumas	2	8	9
Perkūrimo kaštai	6-7	4	5

3.11 Kodo generavimo praktinis pavyzdys

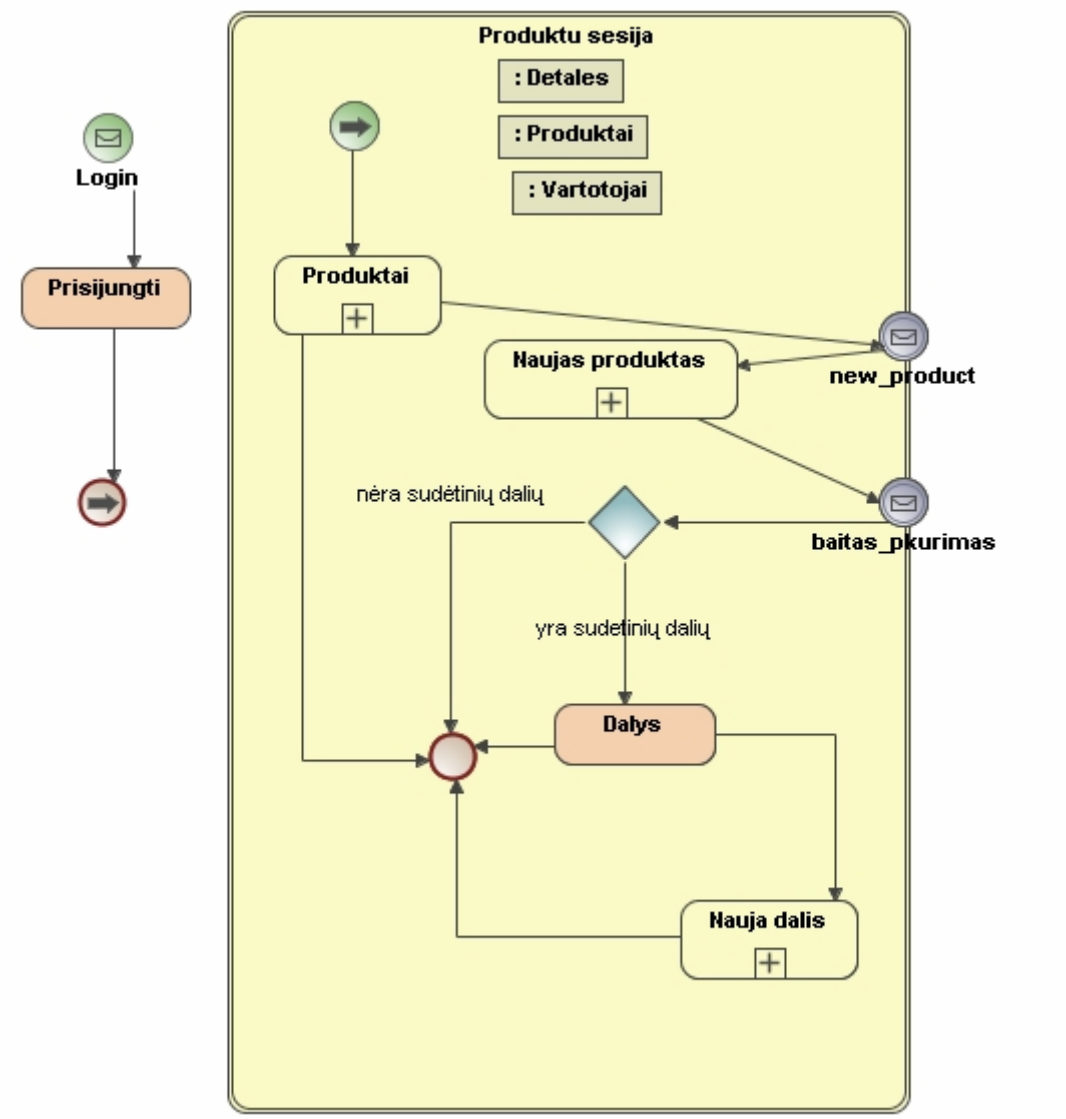
3.11.1 Dalykinė sritis ir jos modelis

Sumodeliuota prekių garantijų informacinės sistemos dalis – garantinio produkto įtraukimas. Garantinis produktas tai parduodamas produktas klientui pvz.: kompiuteris. Jis gali turėti sudėtines dalis, kurioms garantija galioja skirtingais terminais ar sąlygomis. 29 paveikslėlyje pateikta duomenų bazės (objektų) diagramos dalis, kuri aktuali tolesnėje praktinio generavimo pavyzdžio analizėje. Analizuosime garantinio produkto sukūrimo (įvedimo) funkcijos realizaciją.



29 pav. Sistemos objektų diagramos dalis

30 paveikslėlyje pateiktoje diagramoje sumodeliuotai garantinės sistemos dalis skirta garantinių produktų sukūrimui. Joje taip pat nubraižytas prisijungimas, kad matytųsi akivaizdus skirtumas tarp verslo transakcijos ir paprasto veiksmo (proceso). Kadangi modeliavimo įrankyje nėra realizuota duomenų objektų įkėlimas (proceso modelio praturtinimas duomenų objektais 3.1 skyrius) ir ryšių sudarymas su procesais teks tai padaryti rankiniu būdu XMI faile (31 pav.).



30 pav. Garantinių produktų sukūrimo proceso diagrama

```
<ownedMember xmi:type="uml:Activity" xmi:id="_vtBNRcEyEdy4r6KVv0AyWw" name="bendra_scehma">
    <eAnnotations xmi:id="_vtBNRsEyEdy4r6KVv0AyWw" source="appliedStereotypes">
      <contents xmi:type="BPMNProfile_0:BPMNProfile__businessProcess"
xmi:id="_vtBNR8EyEdy4r6KVv0AyWw"/>
    </eAnnotations>
    <node xmi:type="uml:Action" xmi:id="_vtBNSMEyEdy4r6KVv0AyWw" name="Login"
clientDependency="_vtBNeMEyEdy4r6KVv0AyWw">
      <eAnnotations xmi:id="_vtBNScEyEdy4r6KVv0AyWw" source="appliedStereotypes">
        <contents xmi:type="BPMNProfile_0:BPMNProfile__messageStartEvent"
xmi:id="_vtBNSsEyEdy4r6KVv0AyWw" message="login"/>
      </eAnnotations>
```

```
    </node>
    <node xmi:type="uml:StructuredActivityNode" xmi:id="_vtBNS8EyEdy4r6KVv0AyWw"
name="Prisijungti" clientDependency="_vtBNdcEyEdy4r6KVv0AyWw">
      <eAnnotations xmi:id="_vtBNTMEyEdy4r6KVv0AyWw" source="appliedStereotypes">
        <contents xmi:type="BPMNProfile_0:BPMNProfile__task"
xmi:id="_vtBNTcEyEdy4r6KVv0AyWw"/>
      </eAnnotations>
    </node>
    <node xmi:type="uml:Action" xmi:id="_vtBNTsEyEdy4r6KVv0AyWw">
      <eAnnotations xmi:id="_vtBNT8EyEdy4r6KVv0AyWw" source="appliedStereotypes">
        <contents xmi:type="BPMNProfile_0:BPMNProfile__linkEndEvent"
xmi:id="_vtBNUMEyEdy4r6KVv0AyWw"/>
      </eAnnotations>
    </node>
    <node xmi:type="uml:StructuredActivityNode" xmi:id="_vtBNUcEyEdy4r6KVv0AyWw"
name="Produktu sesija">
      <eAnnotations xmi:id="_vtBNUsEyEdy4r6KVv0AyWw" source="appliedStereotypes">
        <contents xmi:type="BPMNProfile_0:BPMNProfile__subProcess"
xmi:id="_vtBNU8EyEdy4r6KVv0AyWw" isATransaction="true">
          <properties>produkto_id</properties>
          <properties>vartotojo_id</properties>
        </contents>
      </eAnnotations>
      <containedNode xmi:type="uml:Action" xmi:id="_vtBNVMEyEdy4r6KVv0AyWw"
clientDependency="_vtBOIMEyEdy4r6KVv0AyWw">
        <eAnnotations xmi:id="_vtBNVcEyEdy4r6KVv0AyWw" source="appliedStereotypes">
          <contents xmi:type="BPMNProfile_0:BPMNProfile__linkStartEvent"
xmi:id="_vtBNVsEyEdy4r6KVv0AyWw"/>
        </eAnnotations>
      </containedNode>
```

31 pav. Sistemos modelio XMI failo fragmentas

3.11.2 Pavyzdinio modelio programinė analizė

Pirmasis algoritmo žingsnis surasti atskaitos tašką. Atskaitos taškas laikomas tokia diagrama, kurioje yra sudėtinių procesų, o tie sudėtiniai procesai neturi sudėtinių procesų. Neradus atskaitos taško tolesnė modelio analizė neįmanoma. Diagramoje yra du pradžios elementai damas pradžios įvykio elementas ir tolesnis procesas t.y. „Prisijungti“. Kadangi tai ne sudėtinis procesas jį atmetame ir ieškome sekančio. Sekančio nėra, todėl grįžtame prie kito startinio elemento.

Jeigu suradus startini elementą, kuris priklauso arba veda į transakciją, o ji turi parametrų, tuomet kiekviename faile bus sugeneruojamas jų sukūrimas arba nuskaitymas iš sesijos. Kiekvienam sudėtiniam procesui, bus sugeneruojamas toks programinis kodas:

```
session_start();
if( !isset($_SESSION[ argumento_pavadinimas1 ]) )
      $_SESSION[ argumento_pavadinimas1 ] = “”; // transakcijos
transakcijos_pavadinimas argumento_pavadinimas1 sukurimas
```

```
else $_argumento_pavadinimas1 = $_SESSION[ argumento pavadinimas1 ] //
transakcijos transakcijos_pavadinimas argumento_pavadinimas1
nuskaitymas

if( !isset($_SESSION[ argumento_pavadinimas2 ]) )
      $_SESSION[ argumento_pavadinimas2 ] = "“; // transakcijos
transakcijos_pavadinimas argumento_pavadinimas2 sukurimas
else $_argumento_pavadinimas2 = $_SESSION[ argumento pavadinimas2 ] //
transakcijos transakcijos_pavadinimas argumento_pavadinimas2
nuskaitymas
...
...
```

Ieškant sekančio proceso einama pagal ryšius nuosekliai. Kadangi ryšiai gali išsišakoti, todėl priėjus pabaigą, atliekama likusių neanalizuotų sudėtinių procesų paieška. Jeigu tai sudėtinis procesas, iš kurio patenkama į pabaigą jo pabaigoje bus įdedamas kodas:

```
unset( $_SESSION[ argumento_pavadinimas1 ]);
unset( $_SESSION[ argumento_pavadinimas2 ]);

.....
```

Daugiau įvairių variantų atsiranda generuojant perėjimus. Jeigu sudėtinis procesas turi tiesiogiai ryšį su kitu procesu jo pabaigoje bus įdedama:

```
header( „location: sekantis_procesas.php“);
```

Toks užrašas gali ne visada tinkavai veikti dėl jo specifikos, todėl tokiai atvejais gali prireikti rankinės korekcijos šiai komandai.

Jeigu procesas eina per tarpinį (intermediate) elementą priklausys nuo jo pobūdžio. Dažniausias yra „Message“. Kadangi kodo generavimas vyksta tam tikrai programiniai platformai, o vadovaujantis jos taisyklėmis kiekviename vykdomajame faile naudojamas veiksmų išrinkimas („Swich (veikmas) {....}“). Tokiu atveju į bus įdedama į „Swich“ elementą kodo dalis:

```
case „elemento_message“:
      header( „location: sekantis_procesas.php“);
break;
```

Klaidos elemento atveju vienoje iš php failo pradinių daliu bus įdedamas kodas:

```
$emsg[ „elemento_vardas“ ] = „elemento žinutė“;
```

Perėjimų per kitus tarpinius elementus nedetalizuosiu. Perėjimai per „sprendimo“ elementus generuojami kaip „Jeigu“ sąlygos. Tokiu atveju atsiranda du keliai – taip sąlyga ir ne sąlyga. Tuomet bendriausias kodo šablonas atrodys taip:

```
if (//sąlyga (elemento vardas)) { // sąlyga1}
      else {//salyga2}
```

Jeigu atitinkamoje šakoje yra perėjimas į kitą procesą generuojamas kodas:

```
header( „location: sekantis_procesas.php“);
```

O jeigu perėjimas į pabaigą, generuojamas sesijos kintamųjų pašalinimas:

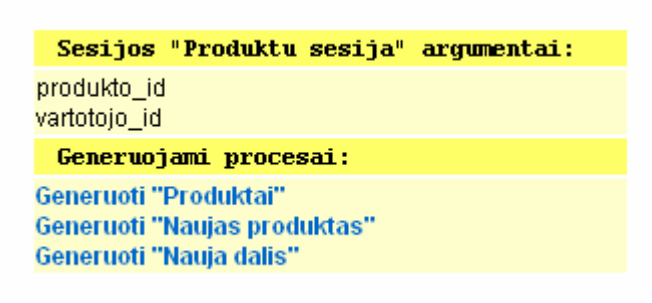
```
unset( $_SESSION[ argumento_pavadinimas1 ]);
unset( $_SESSION[ argumento_pavadinimas2 ]);
.....
```

Ši sąlyga faile patalpinama arti pradžios. Kadangi sunku nustatyti šio tikrinimo vietą, ta padaryti paliekama programuotojui, išskyrus tuos atvejus kai prie sąlygos ateinama per tarpinį „Message“ elementą. Tuomet sąlyga įdedama į „Swich“ elementą atitinkamą „case“ dalį.

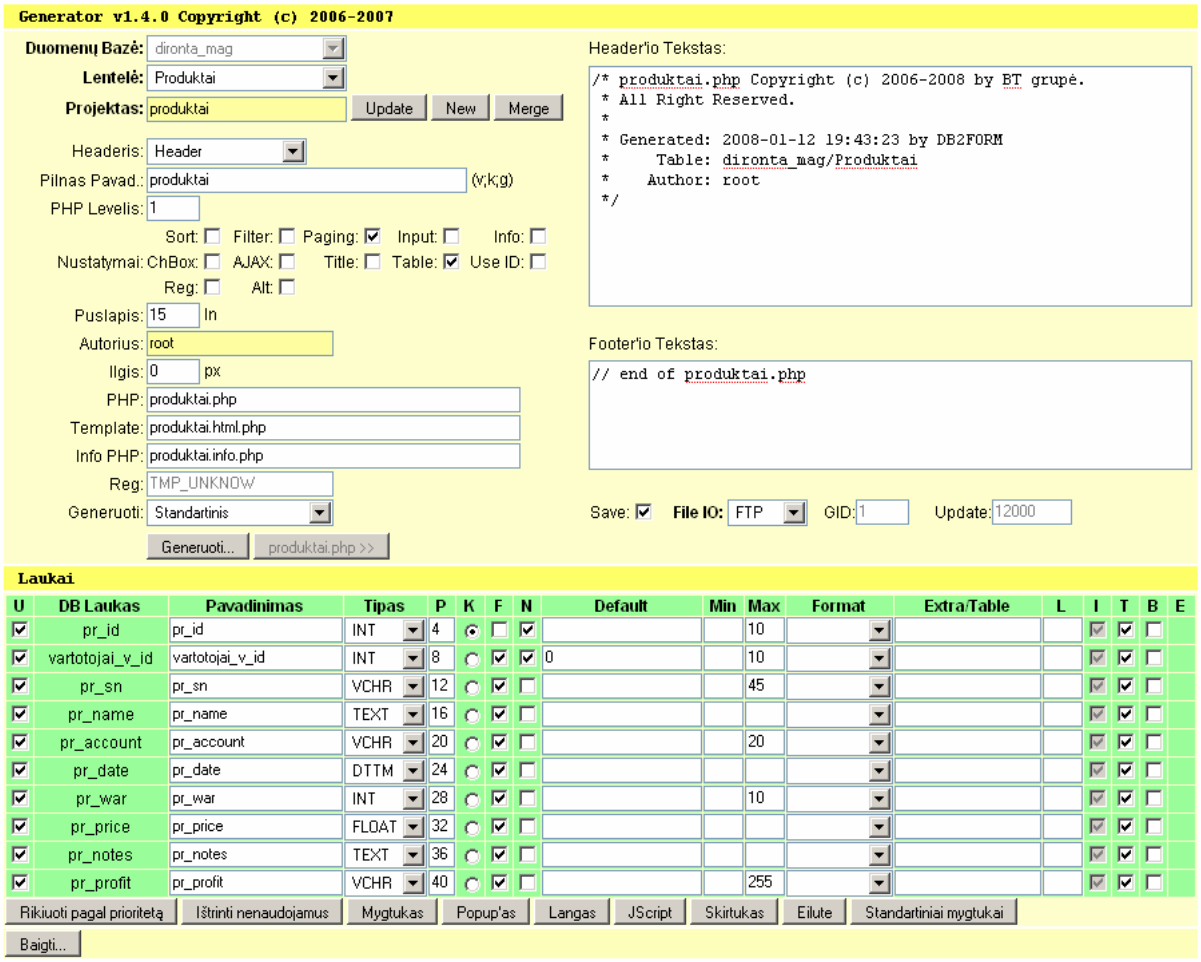
Jeigu sekoje pasitaiko nesudėtinis procesas, jis tiesiog praleidžiamas. O jeigu į sudėtinį procesą ateinama ir nesudėtinio – nereikia atlikti perėjimų generavimo.

3.11.3 Pavyzdinio modelio programinės analizės ir generavimo rezultatai

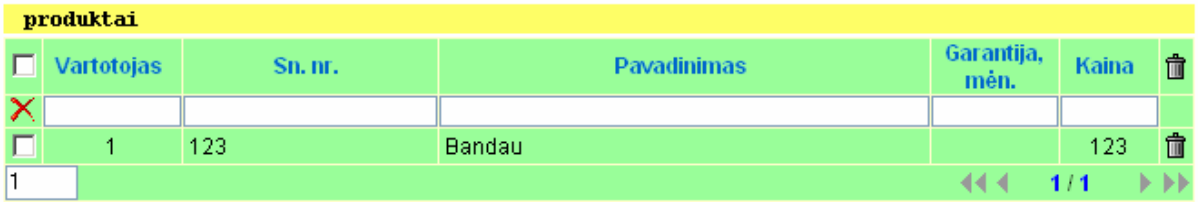
Šio algoritmo analizės rezultatas yra nustatymai (objektai, failų pavadinimai), ir kodo dalys kiekvienam sudėtiniam procesui. Pateikiamas procesų sąrašas, iš kurio po vieną procesą galima toliau tikslinti ir generuoti (32 pav.). Sekančiame generavimo etape nustatomi detalūs proceso parametrai, vartotojo sąsajos tipas ir t.t. (33 pav.). Čia jau generavimui naudojama ir duomenų bazė.



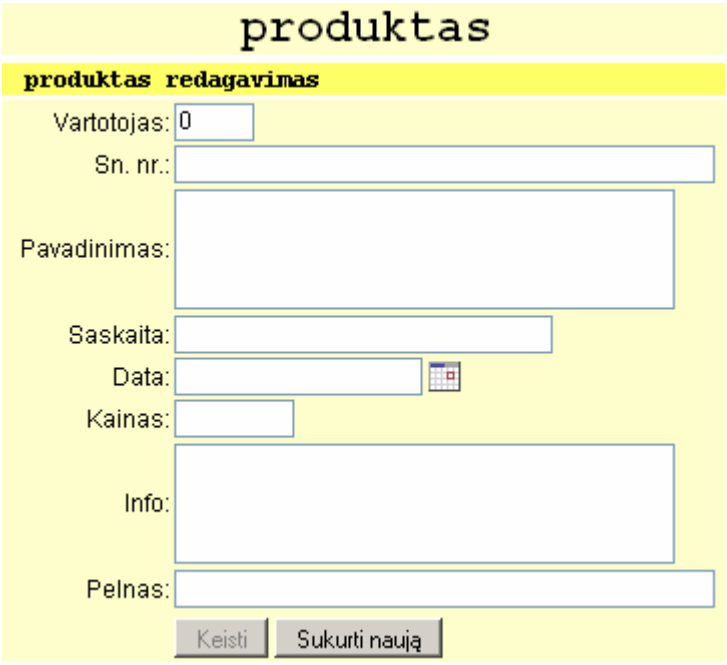
32 pav. I-o žingsnio generavimo langas (modelio analizės rezultatas)



33 pav. II žingsnis. Detalūs proceso nustatymai



34 pav. Sugeneruotas procesas „Produktai“



35 pav. Sugeneruotas procesas „Naujas produktas“

34 ir 35 paveikslėliuose pateikti sugeneruotų procesų vartotojo sąsajos langai. Jie ganėtinai pilnai sugeneruoti ir reikia tik nedidelių pakeitimų (pvz.: lange produktai nėra mygtuko, kuris iškviestų komandą „new_product“), kad juos naudoti veikiančioje sistemoje. Vykdomajame faile kodas šiai komandai yra sugeneruotas:

```
<?php
/* produktai.php Copyright (c) 2006-2008 by BT grupė.
 * All Right Reserved.
 *
 * Generated: 2008-01-12 19:43:23 by DB2FORM
 *     Table: dironta_mag/Produktai
 *    Author: root
 */

session_start();
if( !isset($_SESSION[ "produkto_id" ]) )
        $_SESSION[ "produkto_id" ] = ""; // transakcijos Produktu sesija
produkto_id sukurimas
else $_produkto_id = $_SESSION[ "produkto_id" ] // transakcijos Produktu
sesija produkto_id nuskaitymas

if( !isset($_SESSION[ "vartotojo_id" ]) )
        $_SESSION[ "vartotojo_id" ] = ""; // transakcijos Produktu
sesija vartotojo_id sukurimas
else $_vartotojo_id = $_SESSION[ "vartotojo_id" ] // transakcijos
Produktu sesija vartotojo_id nuskaitymas


require_once "lib/base/settings.inc.php";
require_once "lib/base/".LIB_COMMON;
require_once "lib/base/".LIB_HEADER;


define( "PAGE_LEVEL",1 );
define( "PAGE_TITLE","produktai" );
html_header();
```

```
define( "SORT_FIELDS","vartotojai_v_id,pr_sn,pr_name,pr_war,pr_price" );
require_once "lib/base/".LIB_SORT;

$w = &new parser( "templates/produktai" );
switch( $ac ) {
        case "del":
                    if( $id ) {
                                $sql = "DELETE FROM Produktai WHERE
pr_id=$id LIMIT 1";
                                query( $sql );
                    }
                    $id = 0;
        break;
        case "new_product":
                    header( "location: produktas_new.php");
        break;
.........
```

Šio generavimo metu daugelis nustatymų, susijusių su duomenų objektais buvo atlikti pačiame generatoriuje. Bet jeigu modeliavimo įrankyje pilnai realizavus BPMN modelių praturtinimą duomenų objektais, tuomet mažiau reiktų nustatymų daryti generatoriuje, o visa informacija būtų pateikiama iš modelio. Tai suteiktų patogumo, darbo sklandumo bei paprastesnio pakeitimų realizavimo.

4. Išvados

1. Išanalizuotos verslo sistemų specifikavimo ir modeliavimo technologijos ir metodai. Detalesniam tyrimui pasirinktas UML verslo procesų modelis. Analizuojant verslo procesų modelį pastebėta, kad BPMN notacijoje nėra naudojami duomenų objektai, kurie suteikia daugiau programinio kodo generavimo galimybių, o modeliams statinės sistemos dalies informacijos.
2. Modelių analizei ir perkėlimui iš modeliavimo įrankio į generavimo įrankį pasirinktas XMI standartas, skirtas UML modelių saugojimui. Jo dėka galimas kodo generavimo sprendimas, nepriklausantis nuo modeliavimo įrankio. XML failuose perduodama visa modelių informaciją, kurios reikia generatoriams.
3. Papildyta duomenų objektais verslo procesų ir transakcijų specifikavimui skirta BPMN notacija, todėl verslo procesų modelis tapo aiškesnis, pasipildė statine informacija, padidėjo kodo generavimo galimybės.
4. Siekiant generuoti pilnai funkcionuojantį kodą aprašyta transakcijų savybių panaudojimo galimybės kodo generavime.
5. Aprašyta detali verslo procesų ir transakcijų specifikavimo strategija, kuri leidžia sugeneruoti pilnai funkcionuojančią ir atitinkančią dalykinę sritį verslo valdymo sistemą ar jos dalį. Aprašyta strategija skirta generuoti kodą konkrečiai programavimo platformai, jai skirtoms bibliotekoms, šablonams ir įrankiams.
6. Sukurta praktinė kodo generavimo realizacija naudojant verslo procesų modelį su BPMN notacija panaudojant ir papildant jau egzistuojantį programinio kodo generavimo įrankį. Apklausos ir eksperimento rezultatai parodė jog realizacija galima ir suteikia pranašumo, generavimo spartai bei bendrai sugeneruoto kodo kokybei visoje sistemoje.
7. Eksperimentas parodė jog tobulinant įrankių ergonomiką, išplečiant generuojamų situacijų aibę būtų galima pasiekti didesnio kodo generavimo naudingumo. Taip pat naudinga būtų tobulinti sistemos procesų programinę modelio analizę, kurios dėka būtų galima generuoti įvairesnių specifikuotų verslo valdymo sistemos situacijų.

5. Literatūros sąrašas

1. J. Heumann, Requirements management evangelist, „Introduction to business modeling using the Unified Modeling Language (UML)“, IBM, (http://www-128.ibm.com/developerworks/rational/library/360.html)
2. E. Pakalnickas, S. Gudas, „S*ąsajomis grįstas is projektavimas“,* Kauno technologijos universitetas, 2006 m. IT konferencija,
3. A. Balvočius, D. Šilingas, *„Agile modeliavimo įrankio projektas“,* Vytauto Didžiojo universitetas, UAB “Baltijos programinė įranga”, Kaunas.
4. J. Mukerji, Joaquin Miller, „*MDA Guide Version 1.0.1*“, OMG, (http://www.omg.org/docs/omg/03-06-01.pdf)
5. S.Gudas, *„Žiniomis grindžiama IS inžinerija“,* KTU, 2005 m. gegužė 25 d. 17 paskaita.
6. R. Gustas „Semantic and pragmatic dependencies of information systems“, Kaunas, 1997, ISBN 9986-13-5346
7. R. Butkienė, T120 M051 paskaitų skaidrės 2007 m. „Sąsajos tarp skirtingu modelių ir metodikų“, KTU, Kaunas, 2007
8. *„Rational Unified Proces“*, IBM, (http://www-306.ibm.com/software/awdtools/rup/)
9. R. Dennis Gibbs , *„Project Management with the IBM® Rational Unified Process®: Lessons from the Trenches*“, IBM Press, 2006 m., ISBN-10: 0-321-33639-9.
10. G. Booch, J. Rumbaugh, I. Jacobson , *„The Unified Modeling Language User Guide SECOND EDITION“*, Addison Wesley Professional, 2005 05 19
11. OMG, Business Process Modeling Notation (BPMN) Specification Final Adopted Specification, 2006, (http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf)
12. B. Marchal, „Working XML: UML, XMI, and code generation“, IBM, 2004 (http://www.ibm.com/developerworks/xml/library/x-wxxm23/)
13. O. Pastor, J. C. Molina „Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling“, ISBN 978-3-540-71867-3, Springer, 2007
14. OMG, XML Metadata Interchange (XMI), v2.1.1, 2007-12-01 (http://www.omg.org/technology/documents/formal/xmi.htm)