XI SEKCIJA

VEIKLOS PROCESŲ IR INFORMACINIŲ POREIKIŲ ANALIZĖ

SISTEMŲ INŽINERIJOS IR INFORMACINIŲ TECHNOLOGIJŲ VAIDMUO LIETUVOS TEISĖKŪROS PROCESE[1]

Albertas Čaplinskas

Matematikos ir informatikos institutas
Akademijos g. 4, 2600 Vilnius

Remigijus Mockevičius

Teisinės informacijos centras
Gedimino pr. 30/1, 2695 Vilnius

Straipsnyje aptariamas informacinės infrastruktūros vaidmuo Lietuvos teisėkūroje, svarstomos tokios infrastruktūros efektyvumo problemos. Parodoma, kad būtina Lietuvos teisėkūros sistemos efektyvumo sąlyga yra visų jos lygmenų (doktrina, procedūros, infrastruktūra) darna. Siūloma teisėkūroje plačiau taikyti sistemų inžinerijos principus, aptariama teisės akto kokybės samprata, pasiūlytas kokybės modelis. Straipsnyje panaudoti sociologinio tyrimo rezultatai.

1. Įvadas

Teisėkūra yra sudėtingas ir daugiaaspektis reiškinys. Terminas „teisėkūra“ - naujadaras. Atrodo, kad pirmą kartą jis oficialiai pradėtas vartoti 2000 metų balandžio 20 dieną Vilniuje (Seime) vykusioje mokslinėje-praktinėje konferencijoje „Teisinės sistemos reforma: teisėkūros problemos“. Šioje konferencijoje buvo diskutuojama dėl šio termino turinio, kai kurie konferencijos dalyviai terminą “teisėkūra” vartojo kaip terminų „teisėdara“ arba „įstatymų leidybą“ sinonimą. Darbe [5] skiriama teisėkūra siaurąja prasme ir teisėkūra plačiąja prasme. Teisėkūra siaurąja prasme apibrėžta kaip formalizuotas teisės norminių aktų (teisės šaltinių) kūrimo procesas, apimantis teisės akto poreikio nustatymą, koncepcijos formulavimą ir pagrindimą, teisės akto projekto rengimą, derinimą, svarstymą bei vertinimą, teisės akto priėmimą, paskelbimą ir registravimą. Kitaip tariant, teisėkūra siaurąja prasme suvokiama kaip procesas apimantis visą teisės akto gyvavimo ciklą nuo jo užmanymo iki įsigaliojimo. Teisėkūra plačiąja prasme apibrėžta kaip visos teisės sistemos, jos struktūrinių dalių (idėjų, doktrinos, tradicijų, teisės norminių aktų, teisinių santykių, teisinės etikos ir kt.) kūrimo bei formalizavimo procesas. Šiame straipsnyje terminas „teisėkūra“ yra vartojamas siaurąja prasme.

Lietuvoje teisėkūros procesas šiuo metu yra labai intensyvus. Per pirmąjį nepriklausomybės dešimtmetį priimta apie pustrečio tūkstančio įstatymų (iš jų apie 900 naujų) ir apie dvylika su puse tūkstančio Vyriausybės nutarimų [12]. 2000 metų pradžioje Lietuvoje galiojo 2002 įstatymai [15]. Pažymėtina, kad įstatymų projektų rengiama gerokai daugiau. Pavyzdžiui, iš šeštosios Lietuvos Vyriausybės parengtų 146 įstatymų projektų Seimas priėmė tik 85, iš aštuntosios Lietuvos Vyriausybės parengtų 114 įstatymų projektų Seimas priėmė tik 51 ir t.t. [15]. Esant tokiam intensyviam teisėkūros procesui, teisės aktai dažnai “rengiami ir priimami chaotiškai, skubotai, neturint aiškios, mokslu pagrįstos koncepcijos” [12]. Dėl to jie yra prastos kokybės, neveiksmingi, juos tenka taisyti ir keisti. Pavyzdžiui, priėmus Pridėtinės vertės mokesčio įstatymą, jis buvo taisytas daugiau kaip du šimtus kartų [11]. Šitokia praktika skatina žmonių nepasitikėjimą įstatymais, sąlygoja daugelį kitų rimtų problemų. Susidariusią padėtį puikiai suvokia Lietuvos teisininkai [5], [9], [15]. Pateikta nemaža rekomendacijų, kaip ją taisyti [5], [12], [15]. Kai kurios iš pateiktų rekomendacijos yra inžinerinio pobūdžio, nors patys tų rekomendacijų autoriai galbūt šitaip ir nemano. Pavyzdžiui, darbe [15] yra kalbama apie būtinybę planuoti teisės akto rengimą, parengti to akto koncepciją ir vertinti to akto kokybę. Seimo ir Vyriausybės reglamentai reikalauja, kad prieš pradedant rengti teisės aktą išreikštine forma būtų formuluojami siekiami tikslai, vertinamos pozityvios ir negatyvios pasekmės, skaičiuojamos tam aktui įdiegti reikalingos lėšos. Tačiau, mūsų nuomone, inžinerinio teisėkūros proceso aspekto svarba vis dar yra nepakankamai suvokta, pateiktos rekomendacijos yra fragmentiškos ir jas įdiegus esamų problemų išspręsti nepavyks. Šio straipsnio tikslas – panagrinėti teisėkūros procesą sistemų inžinerijos požiūriu ir išryškinti tas esmines problemas, neišsprendus kurių, teisėkūra negali tapti efektyvi. Be abejo, mes nagrinėjame tik būtinasias efektyvaus teisėkūros

[1] Darbas atliktas, vykdant Jungtinių Tautų remiamą Lietuvos Vyriausybės projektą LIT/01/004 „Parama teisėkūros reformai“ ir Matematikos ir informatikos instituto planinę temą „Programų sistemų, informacinių sistemų ir verslo sistemų inžinerijos intelektualizavimo problemos“.

proceso prielaidas, arba, kitaip tariant, tik vieną iš sėkmės veiksnių. Išskyrus inžinerinį aspektą, teisėkūros sėkmę lemia teisiniai, politiniai ir kiti veiksniai. Jų šiame straipsnyje mes nenagrinėjame.

2. Sistemų inžinerijos vaidmuo teisėkūros procese

Sistemų inžinerija – tai pati bendriausioji inžinerijos mokslų šaka, nagrinėjanti kaip turi būti kuriamos bet kurios prigimties ar pobūdžio dirbtinės sistemos (taip pat ir kiti sudėtingi artifaktai). Kitaip tariant, sistemų inžinerija nagrinėja tuos principus ir metodus, kurių būtina laikytis kuriant tiek klasikines inžinerines sistemas (mašinas, statinius ir kt.), tiek ir programų, informacines, verslo, veiklos ar kitas sistemas. Išimtis – meno kūriniai. Nors sistemų inžinerijos principai bei metodai yra taikomi, kuriant masinės kultūros gaminius, pavyzdžiui, vadinamąsias „muilo operas“, tikrą meno kūrinį šitaip sukurti negalima, nes meno kūriniai privalo perteikti impresijas, nuotaikas, autoriaus išgyvenimus ir yra vertinami vadovaujantis visiškai kitais kriterijais nei, tarkime, įstatymai, oro tiltai, politinės programos ar poilsiavietės. Sistemų inžinerijos principų bei metodų tiesiogiai panaudoti negalima, juos reikia pritaikyti konkrečios inžinerijos šakos specifikai. Vienaip jie yra taikomi, tarkime, statyboje, kitaip – kuriant mašinas. Inžinerijos šaka, nagrinėjanti inžinerinių metodų panaudojimą teisėkūroje yra vadinama *teisės inžinerija*.

Inžinerinių metodų taikymas teisėkūroje turi ilgą ir turtingą tradiciją. Teisė yra kuriama jau keli tūkstančiai metų. Teisės aktų sistemos yra labai sudėtingos ir, netaikant inžinerinių metodų, jų paprasčiausiai būtų nepavykę sukurti. Teisėkuroje jau daugelį dešimtmečių taikomi teisės akto gyvavymo ciklo modelis, išskiriamos reikalavimų formulavimo, eskizinio projektavimo ir kitos ciklo stadijos, atliekamas formalizuotas teisės aktų vertinimas, naudojami tikslais valdomo projektavimo ir kiti sistemų inžinerijos metodai. Tačiau teisininkai, bent jau Lietuvos teisininkai, kol kas šių metodų nesuvokia ir neįvardina kaip inžinerinių. Kalbama apie „juridinę techniką“, „sisteminį požiūrį“, „ekspertizę“ ar kitus dalykus. Daugelis žmonių (ir ne tik teisininkų) terminą “inžinerija” vis dar sieja su tradicinėmis inžinerijos šakomis, tokiomis, kaip, tarkime, statyba ar mašinų gamyba. Terminai „teisės inžinerija“ ar „kalbos inžinerija“ daugumai humanitarinės pakraipos specialistų vis dar yra nepriimtini. Jų požiūriu, inžinierius – tai ribotas „technokratas“, bandymas inžinerinius metodus taikyti humanitarinės veiklos sferoje yra, švelniai tariant, tiesiog nesusipratimas. Šitokį požiūrį ir toliau formuoja mokymo sistemos. Nors bendrieji inžinerijos principai neabejotinai yra tokia pati bendrojo išsilavinimo dalis, kaip, tarkime, istorija, literatūra, matematika ar geografija, mokykloje apie juos apskritai nieko nėra kalbama. Praeito šimtmečio pirmosios pusės nuostatomis grindžiamas mokyklų profiliavimas, bandymai dezintegruoti humanitarinius ir tiksliuosius mokslus padėtį, matyt, tik pablogins. Galima būtų teigti, kad svarbu, kokie metodai yra naudojami, o ne kaip jie yra vadinami. Tačiau iš tiesų yra truputį kitaip. Nesuvokiant naudojamų metodų inžinerinio pobūdžio, studentai nėra mokomi bendrųjų sistemų inžinerijos principų, verdama savose sultyse, nevyksta idėjų mainai su kitomis inžinerijos šakomis. Todėl yra atsiliekama. Žvelgiant iš sistemų inžinerijos taško, matosi, kad inžineriniai metodai yra taikomi fragmentiškai, kai kurie iš jų yra gerokai pasenę. Tą supranta ir dalis teisininkų. Pavyzdžiui, darbe [1] yra rašoma, kad „nors apie teisės aktų rengimo mechanizmą yra kalbama ir Metmenyse, o 1995 m. Lietuvos Respublikos Seimo buvo priimtas Įstatymų ir kitų teisės norminių aktų rengimo tvarkos įstatymas, kažin ar galėtume pasakyti, jog šis mechanizmas pertvarkytas taip, kad atitiktų šios dienos sąlygas ir keliamus reikalavimus“. Tokių pavyzdžių galima pateikti ir daugiau.

Panagrinėsime teisėkūrą kaip sistemą išsamiau. Sistemų inžinerijos požiūriu, teisėkūra yra viena iš veiklos sistemų. Kitaip tariant, teisėkūros sistema yra naudojama siekti tam tikrų tikslų, turi koncepcinį (teisėje tai vadinama teisėkūros doktrina), technologinį (procedūrų) ir infrastruktūros lygmenis. Trumpai aptarsime kiekvieną iš tų lygmenų.

2.1. Teisėkūros tikslai ir doktrina

Teisės aktai yra kuriami, siekiant reguliuoti visuomeninius santykius. Reguliuojant visuomeninius santykius, „išvengiama atsitiktinumų ir savivalės, socialinio gyvenimo nestabilumo, interesų priešpriešos“ [18]. Kuo sudėtingesni visuomeniniai santykiai, tuo didesnis poreikis juos reguliuoti. Nereikia pamiršti, kad visuomeniniai santykiai nėra stabilūs. Po nepriklausomybės paskelbimo visuomeniniai santykiai Lietuvoje pasikeitė iš esmės. Kadangi buvo neįmanoma iš karto pakeisti visų galiojusių teisės aktų, kurį laiką greta naujų teisės aktų galiojo daugelis senųjų. Kitaip tariant, vyko reinžinerijos procesas. Jis nėra visiškai užbaigtas ir dabar. Netaikant reinžinerijos teorijos metodų, procesas vyko chaotiškai, tai buvo aiškinama „pereinamojo laikotarpio nesklandumais“. Šiandien, atrodytų, viskas čia turėtų būti paprasta ir aišku, „siekis, kad įstatymas tarnautų ne atskirų grupių ar partijų interesams, o ilgalaikiams valstybės ir visuomenės poreikiams, turėtų būti esminė visų šioje srityje dirbančių nuostata“ [11]. Tačiau iš tiesų teisėkūros tikslai nevisuomet yra aiškiai suvokiami, kartais netgi sąmoningai iškreipiami arba pakeičiami, „Teisės aktų rengėjai, o jais paprastai esti valstybės institucijos, dažniausiai *kuria teisės aktus pirmiausiai sau*, t.y. jos yra pasisavinusios atskirų sričių įstatymų projektų rengimą. Dėl to teisės aktai parengiami vienpusiškai ir atspindi tik vienos įstaigos ar kokios nors sistemos siaurus interesus“ [1], kartais koks nors įstatymas yra kuriamas tik todėl, kad jo „reikia Briuselyje „atsiskaityti“, o ar jis bus taikomas Lietuvoje – pagalvota bus vėliau“ [15], „Gana dažnai *teisės akto tikslas – pats teisės aktas*. ... parengti teisės aktą kartais yra greičiau ir paprasčiau nei išspręsti iškilusią problemą iš esmės. Pastaruoju metu galima pastebėti tendenciją įstatymo ar kito teisės akto priėmimą

laikyti visas valstybės ir visuomenės negalia gydančia panacėja, nesirūpinant to teisės akto padariniais ir jo įgyvendinimo realiu efektu“ [1], vadovaujamasi „gaisrų gesinimo“ taktika, „užuot sukūrus strategiją ir parengus aiškią esminių pertvarkymų koncepciją, sprendžiamos šios dienos arba net „post factum“ problemos“ [1]. Netgi Respublikos Prezidentas teigia, kad „pagrindinis įstatymų leidybos tikslas – naikinti biurokratines kliūtis, varžančias žmogaus teises ir laisves“ [4], o ne reguliuoti visuomeninius santykius. Taigi, vartojant sistemų inžinerijos terminiją, galima teigti, kad Lietuvos teisėkūros sistemoje šiuo metu nėra efektyviai veikiančių tikslų formulavimo ir nuokrypių nuo numatytų tikslų reguliavimo mechanizmų.

Tikslų formulavimo problema tampa dar sudėtingesnė, pabandžius atsakyti į klausimą, ko gi yra siekiama reguliuojant visuomeninius santykius. Čia iškyla teisės akto tikslingumo (angl. *suitability*) klausimas arba, kitaip tariant, teisės aktu siekiamų tikslų atitikimo visuomenės interesui laipsnio nustatymo klausimas. Teisės teorija teigia, kad vienas iš pagrindinių visuomeninių santykių reguliavimo tikslų yra teisingumas. „Negalima pasiekti teisingumo tenkinant tik vienos grupės arba vieno asmens interesus ir kartu neigiant kitų interesus. Elgiantis vienašališkai, būtų ignoruojama humaniškoji teisės paskirtis, didėtų socialinių konfliktų galimybė.“ [18]. Vienok, teisingumo ir tuo pačiu teisės aktų tikslingumo vertinimo kriterijai yra subjektyvūs. Jie priklauso nuo pasirinktosios interesų derinimo strategijos (angl. *conflict resoliution strategy*). Strategijos pasirinkimą remia politiniai įsitikinimai ir bendroji politinė kultūra. Pateiksime keleta interesų derinimo strategijos pavyzdžių.

Naudojant „totalitarinę“ strategiją, interesų konfliktas sprendžiamas teikiant pirmenybę „didžiajam tikslui“, pavyzdžiui, komunistiniam rytojui, tūkstantamečiui reichui, laisvajai rinkai, euroatlantinei integracijai ar informacinei visuomenei. „Didysis tikslas“ gali būti racionalus arba iracionalus. Ideologinės šios strategijos ištakos – religinė dogmatika ir jos pagimdytos įvairios totalitarinės pakraipos ideologijos (kalvinizmas, jakobinizmas, fašizmas, bolševizmas ir pan.). Iš tiesų „didžiuoju tikslu“ dažniausiai yra pridengiamas grupinis ar žinybinis interesas ir šitaip visuomenei primetama „elito“, tiksliau, juo apsišaukusių visuomenės sluoksnių ar grupių, valia.

Naudojant „socialdemokratinę“ strategiją, interesų konfliktas sprendžiamas, siekiant maksimalaus skirtingų grupių konsensuso. Šitokia strategija faktiškai yra vadovaujamasi Švedijos teisėkūroje. Panašias nuostatas propaguoja ir kai kurie Lietuvos teisininkai. Pavyzdžiui, darbe [18] teigiama, kad „teisė negali remtis vien tik daugumos ar mažumos interesais, todėl teisėkūroje siekiama derinti interesus panaudojant optimalias šalių susitarimo galimybes“. Ideologinės „socialdemokratinės“ strategijos ištakos - „visuotinės gerovės“ valstybės koncepcija ir kitos moderniosios socialdemokratinės filosofijos srovės. Pagrindinis šios strategijos trūkumas yra tai, kad neretai konsensusas gali būti pasiektas tik atsisakius nesutarimus iššaukusių nuostatų, dėl ko teisės aktas tampa neefektyviu, o kartais ir apskritai nebereikalingu.

Naudojant „demokratinę“ strategiją, interesų konfliktas sprendžiamas teikiant pirmenubę silpniausių ar labiausiai pažeidžiamų visuomenės grupių (tautinių mažumų, kalinių, gėjų, pensininkų ar kt.) teisių apsaugai. Ideologinės šios strategijos ištakos - moderniosios etikos teorijos. Pagrindinis strategijos trūkumas yra tai, kad sutelkus dėmesį ginamų grupių interesams gali būti ignoruojami kitų grupių esminiai interesai, pavyzdžiui, ginant nusikaltimą padariusių asmenų teises, kartais yra pažeidžiami nusikaltimo aukų interesai.

Tai tik keli charakteringi pavyzdžiai. Galimų interesų derinimo strategijų yra daug. Sistemų inžinerijos požiūriu, pirmenybė negali būti teikiama nei vienai iš strategijų. Strategijos parinkimas - neinžinerinė problema. Vienok, formuluojant teisės akto tikslus, būtina aiškiai suvokti, kokia strategija yra vadovaujamasi. Todėl, mūsų nuomone, interesų derinimo strategija turėtų būti išreikštiniu būdu suformuluota teisėkūros doktrinoje. Aišku, “teisės normos kūrėjas negali ignoruoti pagrindinėje teisės normoje – Konstitucijoje įtvirtintų žmogaus teisių ir laisvių, valstybės valdžios institucijų kompetencijos ribų ir kitų konstitucinių nuostatų” [18]. Kitaip tariant, parenkant interesų derinimo strategiją, reikia vadovautis Konstitucija. Ar teisės aktas atitinka Konstitucijai, nustato Konstitucinis Teismas. Tačiau, mūsų nuomone, vien Konstitucijoje išdėstytų teisėkūros principų nepakanka. Beje, teisėkūros doktrina turėtų atsakyti ir į daugelį kitų labai svarbių klausimų: kas vadintina teisės aktu, kokios esti teisės aktų rūšys, kokie visuomeniniai santykiai kokiais teisės aktais reguliuotini, kokia loginė visos teisės aktų sistemos struktūra, kokia šios sistemos ontologinė struktūra, kaip teisėkūros procesas sietinas su logine ir ontologine teisės aktų sistemos struktūra ir kt. Lietuvos teisėkūros doktrina šiuo metu, mūsų nuomone, yra neišsami, miglota, blogai dokumentuota. Taip mano ir kai kurie kiti autoriai. Pavyzdžiui, darbe [13] pastebima, kad lietuviškoji teisės (ne vien teisėkūros) doktrina dar tik kuriasi, kad per nepriklausomybės dešimtmetį nesuskubta suvokti, kad ji yra vienas iš teisės šaltinių. Konstitucinis Teismas taip pat nekartą akcentavo teisės aktų sistemos hierarchinės struktūros, teisės aktų normiškumo klausimus [18]. Neturint pragmatiškos, aiškiai suformuluotos teisėkūros doktrinos, išdėstytos viename dokumente, sukurti efektyvią teisėkūros sistemą vargu ar yra įmanoma. Doktrina turi būti tokia, kad niekam nekiltų klausimas, ar, tarkime, Nacionalinio teatro problemas reikia spręsti specialiai tam skirtu įstatymu, ar kokiu nors kitu teisės aktu. Negalima turėtų tapti ir situacija, kuomet vienu įstatymu nustatomos tam tikros normos, kurios ignoruojamos kitame, taip pat įstatymo statusą turinčiame, teisės akte. Šitokie prieštaravimai yra, pavyzdžiui, tarp Valstybės registrų įstatymo ir kai kurių konkrečių valstybės registrų įstatymų numatytų normų. Pažymėsime, kad, formuojant teisės doktriną, gana daug jau yra padaręs Konstitucinis Teismas. Nemaža šiuo klausimu yra padarę ir atskiri teisininkai. Daug šiuo klausimu taip pat yra padarę V. Andriulis, A. Dziegoraitis, V. Sinkevičius, S. Stačiokas, J. Žilys ir kiti Lietuvos teisininkai. Tačiau, pakartosime, oficialiai įteisintos ir glaustai suformuluotos doktrinos kol kas

nėra. Neapibrėžta netgi pati teisės akto samprata (žr. [9]), teisės aktų rūšis nustato tik Lietuvos Respublikos įstatymų ir kitų teisės aktų registro įstatymo 4 straipsnis (ten kalbama, kokie teisės aktai yra registruojami, bet ne kokie santykiai, kokiais teisės aktais turi būti reguliuojami), nėra tiesiogiai išspręstas konstitucinių ir kitų įstatymų kolizijos klausimas ir t.t.

Grįžtant prie tikslų formulavimo, matome, kad politinė dedamoji dažnai lemia tų tikslų pobūdį. Per teisės aktus yra įgyvendinama valstybės politika. Kaip pažymėta darbe [18], „teisėkūros procesą veikia ekonominiai, politiniai, socialiniai, ideologiniai veiksniai“. Kitą vertus, teisėkūros sistema turi būti maksimaliai depolitizuota, nes kitaip, ypač politinio nestabilumo sąlygomis, visa teisės sistema gali būti destabilizuota. Vykstant dažnai valdžių kaitai, prarandama aiški valstybės politika, teisės aktai skiriami daugiau valdžios problemoms spręsti, o ne valstybės politikai įgyvendinti, jie praranda savo legitimumą, t.y. žmonės nustoja jiems pritarti ir jų laikytis, „tikėdamiesi, jog ta valdžia, o kartu ir jos vykdoma politika ilgai neegzistuos“ [1]. Sistemų inžinerijos požiūriu, čia iškyla teisės akto *universalumo* klausymas [3]. Teisės akto universalumas nusakomas tikimybe, kad teisės aktą, jo nekeičiant, bus galima taikyti pakitus politinei doktrinai, įvykus socialiniams pokyčiams ar atsiradus naujiems visuomeniniams santykiams [1]. Sistemų inžinerijoje universalumo problema yra sprendžiama apibendrinant sistemos funkcinę architektūrą, realizuojant ne konkrečias funkcijas, o tokių funkcijų šeimas [3]. Panašiai ši problema yra sprendžiama ir teisėkūroje, t.y. derinant konkretumą ir abstraktumą, bandant išvengti „dviejų kraštutinumų – pernelyg abstrakčių formuluočių ir pernelyg detalios kazuistikos“ [13]. Kitaip tariant, teisės normos turėtų aprėpti ir situacijas, kurių iš anksto numatyti yra neįmanoma, leisti naujas interpretacijas, be kurių neįmanoma jų pritaikyti naujomis sąlygomis. Taisyklės turi būti „suformuluotos tokiu abstrakcijos lygmeniu, kuris užtikrina tiek šių taisyklių stabilumą, tiek jų lankstumą ir galimybę operatyviai reaguoti į gyvenimo pokyčius, kartu išsaugant jų ilgamžiškumą ir paliekant erdvės žmonių savireguliacijai“ [13]. Kita vertus, jos turi būti preciziškos, neleisti savivalės. Šito negalima padaryti, neturint pakankamai ištobulintos terminijos ir apskritai vadinamosios „dalykinės prozos“, t.y. dalykinės srities, šiuo atveju teisės, kalbos. Deja, čia padėtis panaši į padėtį informatikoje. Lietuviškoji teisinė kalba dar tiktai pradeda formuotis [13]. Mūsų nuomone, šiuo metu Lietuvos teisėkūroje yra stebimos kazuistinės tendencijos, t.y. kiekvienai konkrečiai situacijai bandoma kurti specialią normą. Tačiau tai, matyt, ne tiek teisės inžinerijos, kiek teisėkūros procese dalyvaujančių asmenų kvalifikacijos problema. Šią problemą aptarsime šiek tiek vėliau.

Be abejo, net ir maksimalus sistemos universalumas jos adaptyvumo problemos iki galo išspręsti negali. Tai gerai žinoma sistemų inžinerijoje ir, suprantama, teisinga ir teisės inžinerijos atveju. Sistemų inžinerijoje problema sprendžiama [3] iš anksto numatant sistemos modifikavimo galimybes. Kitaip tariant, taikant moduliarizavimo, parametrizacijos, juodosios dėžės ir kitus panašius principus. Atitinkamos architektūros parinkimas padeda modernizuoti vadinamąsias paveldėtas (angl. *legacy*) sistemas. Teisėkūroje paveldėtų sistemų reinžinerija vadinama *kodifikacija*. Skiriama formalioji ir materialioji kodifikacijos [13]. Atliekant formaliąją kodifikaciją, galiojančios teisės normos susisteminamos pagal tam tikrus kriterijus, panaikinamos pasenusios normos, pašalinami pasikartojimai. Tai formaliais kriterijais grindžiama loginė teisės normų klasifikacija, jų sisteminimas, atliekamas nekeičiant jų turinio. Atliekant materialiąją kodifikaciją, teisės normos yra kokybiškai pertvarkomos, pritaikant jas pasikeitusioms sąlygoms. Tai „teisės racionalizavimo būdas, pasenusių, archaiškų ar absurdiškų teisės institutų ir koncepcijų keitimas moderniais“ [13] arba, vartojant sistemų inžinerijos terminiją, paveldėtos sistemos reinžinerijos procesas. Kodifikacija turi seną tradiciją, pavyzdžiui, Hamurabio kodeksas buvo sudarytas 1000 metų prieš Kristaus gimimą. Lietuvoje ši tradicija taip pat labai sena. Karaliaus Kazimiero Teisynas priimtas 1468 m., pirmasis Lietuvos Statutas – 1529 m. Matyt, su šia patirtimi būtina giliau susipažinti sistemų inžinerijos specialistams ir iš jos pasimokyti. Kita vertus, mūsų nuomone, Lietuvos teisėkūroje kol kas skiriamas nepakankamas dėmesys tiek konkrečių teisės aktų architektūrai, tiek ir visos teisės aktų sistemos architektūrai. Apie tai, pavyzdžiui, kalbama darbe [4]. Be abejo, projektuojant konkretų kodeksą, jo architektūrai skiriama daug dėmesio. Kodeksas grindžiamas idėja, kad teisės normas galima išdėstyti tam tikra racionalia, logiškai pagrįsta tvarka, „kodekso struktūra turi atspindėti jo materialų nuoseklumą ir vientisumą, jo vidinę logiką“ [13] ir „tai reiškia, kad kodifikacijai reikia racionalaus plano pagal kurį būtų galima eiti nuo bendrų prie specialių dalykų“ [13]. Be to, siekiama parengti visą tam tikrų visuomeninių santykių sritį, pavyzdžiui, privatinius santykius, reguliuojantį teisės aktą [13]. Tačiau tiek kuriant teisės aktus iki kodifikacijos, tiek ir darant pakeitimus sukurtuose kodeksuose, šie principai Lietuvos teisėkūroje dažniausiai yra ignoruojami.

2.2. Teisėkūros proceso modelis

Technologiniame lygmenyje yra apibrėžiamas proceso modelis. Sistemų inžinerijos požiūriu [3], sudarant proceso modelį yra siekiama keturių pagrindinių tikslų:

- nustatyti, kokias kokybiškai skirtingas tarpines būsenas pereina artifaktas, jį kuriant (artifakto gyvavimo ciklo modelis),
- nustatyti, kokios technologinės procedūros ir kokia tvarka turi būti vykdomos norint sukurti artifaktą (technologijos modelis),

- numatyti, kokiais laiko momentais ir kas turima būti kontroliuojama, norint patikrinti ar nenukrypstama nuo planuotos darbų eigos ir/arba kokybės reikalavimų (priežiūros modelis),
- nustatyti projekto planavimo, valdymo, vykdymo, audito, inspektavimo ir vertinimo tvarką (projekto modelis).

Kitaip tariant, proceso modelis – tai darnus artifakto gyvavimo ciklo, technologijos, projekto ir jo priežiūros modelių junginys. Kokybės reikalavimams formuluoti ir jų įgyvendinimui kontroliuoti yra reikalingas penktas modelis – kokybės modelis.

Formalizuoti teisėkūros proceso modeliai nei Lietuvoje, nei, kiek mums žinoma, kitose šalyse kol kas nėra naudojami. Tačiau kai kurios šio modelio dalys yra sudarytos ir, bent jau iš dalies, įdiegtos į Lietuvos teisėkūros praktiką. Panagrinėsime šias dalis sistemų inžinerijos požiūriu.

Kiekvienai teisės aktų rūšiai (įstatymui, poįstatyminiui aktui ir t.t.) yra apibrėžtas ir atitinkamais normatyviniais aktais įteisintas tos rūšies gyvavimo ciklo modelis. Pavyzdžiui, įstatymui numatytos šios kokybiškai skirtingos būsenos: koncepcija, projektas, vyriausybės aprobuotas projektas, Seimo komitetų aprobuotas projektas, Seimo priimtas įstatymas, Prezidento pasirašytas įstatymas, įsigaliojęs įstatymas. Kokybiškai skirtingos būsenos dar yra dekomponuotos į tarpines būsenas. Panašiai yra ir su kitais teisės aktais. Tačiau, vieno dokumento, kuriame būtų aprašyti visų teisės aktų rūšių gyvavimo ciklo modeliai nėra. Be to, sistemų inžinerijos požiūriu, kritikuotinas ir pats modelis, tiksliau, jo pradinė dalis. Sutinkamai su šiuo požiūriu, įstatymo rengimas turėtų būti pradėtas poreikių specifikacijos rengimu (būtinumo kurti įstatymą pagrindimas), po to turėtų būti rengiama reikalavimų specifikacija (ką ir kaip turi reguliuoti, kokybės reikalavimai), po to – įstatymo projekto metmenys ir tik po to – jo projektas.

Teisės aktų rengimo stadijas (viršutinį technologijos modelio lygmenį), nustato *Lietuvos Respublikos įstatymų ir kitų teisės norminių aktų rengimo tvarkos įstatymas* (toliau šioje pastraipoje tiesiog *Įstatymas*), žemesnio lygmens procedūras – Seimo ir Vyriausybės reglamentai. Yra numatyti ir kontrolės taškai, kai kurios audito, inspekcijos ir vertinimo procedūros (priežiūros modelis). Iš pirmo žvilgsnio atrodo, kad šio technologijos modelio pakanka. Tačiau, neturint aiškios, vienareikšmės, pragmatiškos teisėkūros doktrinos, ją įgyvendinančio proceso modelio iki galo apskritai yra neįmanoma sukurti. Antra, šį modelį įteisinančios normos turi daug išlygų. Pavyzdžiui, *Įstatymo* 3-čiojo straipsnio 1-oji pastraipa reikalauja, kad teisės akto rengimas prasidėtų to akto paskirties ir jo rengimo užduoties nustatymu, tačiau, kaip teigiama darbe [15], to paprastai nėra paisoma, nes tokią teisę suteikia to paties įstatymo 4-sis straipsnis. Ignoruojant koncepcijos rengimą, susiduriama su teisės akto inkorporavimo į teisės sistemą problemomis [1], kartais bandoma mechaniškai perkelti į Lietuvos teisės sistemą užsienio valstybių teisės aktus [15], kyla kitos neigiamos pasekmės[2]. Trečia, pats modelis taip pat yra kritikuotinas. Tai vadinamasis klasikinis arba „krioklio“ (angl. *waterfall*) modelis. Šis modelis atėjo iš mašinų gamybos, yra gerokai pasenęs. Klasikinio modelio trūkumai gerai žinomi [3]. Kalbant apie šio modelio panaudojimą teisėkūroje, vienu iš svarbiausiųjų trūkumų, matyt, yra tai, kad jis yra grindžiamas vadinamąją „iš viršaus žemyn“ (angl. *top-down*) paradigma [3], nes kaip teigiama darbe [5], „akyvaizdu, kad kuriant teisės norminį aktą būtina įvertinti ir tai, ar yra mechanizmas, kuris sudarytų reikiamas sąlygas projektuojamoms normoms ar jų dariniams (tam tikriems teisiniams institutams) realizuoti“. „Iš viršaus žemyn“ (nuo koncepcijos prie projekto, nuo įstatymo prie poįstatyminių aktų ir t.t.) paradigma kliudo tai atlikti, dėl to „dažnai nenumatomas teisės akto įgyvendinimo mechanizmas arba numatomi nerealūs įgyvendinimo terminai“ [15]. Šį trūkumą galima bandyti taisyti, derinant „iš viršaus žemyn“ ir „iš apačios aukštin“ (angl. *botom-up*) paradigmas, panašiai, kaip tai yra daroma programų sistemų inžinerijoje [3]. Tačiau toks modelio koregavimas nebūtų pakankamas, nes vis vien į jis neatsižvelgtų į kitus rizikos veiksnius, jie nebūtų laiku įvertiname. Sunku įsivaizduoti, kaip Lietuvos teisėkūroje būtų galima panaudoti evoliucinį[3] ar objektinį modelį. Todėl, mūsų nuomone, tikslinga būtų pereiti prie spiralinio modelio [3], kuriame pagrindinis dėmesys skiriamas būtent rizikos faktorių vertinimui.

Pažymėsime, kad kol kas teisėkūroje, bent jau Lietuvos teisėkūroje, visiškai yra ignoruojami teisės aktų testavimo metodai. Mūsų nuomone, sukaupus atitinkamas teisinių precedentų bibliotekas, daugeliu atveju testavimo metodus būtų galima naudoti ir tai neabejotinai palengvintų teisės aktų pasekmių vertinimą.

Dar vienas rimtas esamo modelio trūkumas – technologinių procedūrų lokalizavimas institucijų viduje. Jei, kuriant teisės aktus, vadovaujamasi „iš viršaus žemyn“ paradigma, tai konstruojant patį teisėkūros procesą, ši paradigma yra visiškai ignoruojama, einama iš apačios, nuo procese dalyvaujančių institucijų, į viršų, prie uždavinių. Atrodo, kad kol kas visiškai nesuvokta, kad veiklos sistemos požiūriu institucijų tinklas vaidina tik technologines procedūras vykdančio procesoriaus vaidmenį ir kad tas procedūras šiame tinkle galima įvairiai išskirstyti. Dar daugiau, tobulinant teisėkūros sistema bandoma vadovautis redukcionizmo principais: išsiaiškinti teisėkūros problemas

---

[2] 2001 m. gruodžio 29 d. LR Teisingumo ministras išleido įsakymą Nr. 277, kuriuo patvirtino įstatymų koncepcijų rengimo metodiką ir tvarką. „Koncepcija“ šiame dokumente suvokiama kaip įstatymo metmenys arba, vartojant sistemų inžinerijos terminus, kaip jo eskizinis projektas. Taigi, mūsų aptariamąjį trūkumą yra bandoma pašalinti.

[3] Austrijos-Vengrijos imperijoje, atrodo, buvo bandoma tokį modelį panaudoti. Įstatymas įsigaliodavo tik tam tikroje teritorijoje (berods, Galicijoje), būdavo aprobuojamas ir taisomas, po to jo galiojimas būdavo išplečiamas visai imperijai.

kiekvienos institucijos viduje, jas pašalinti ir šitaip optimizuoti **visą** procesą. Akyvaizdu, kad šitoks požiūris nieko gero duoti negali.

Vienok, blogiausia padėtis yra su projekto valdymo modeliu. Atrodo, kad jokio oficialiai įteisinto projekto valdymo modelio apskritai nėra. Teisės aktai dažniausiai yra vykdomi, atliekant tarnybinius pavedimus, finansiniai ir kiti ištekliai tam nėra planuojami, jų sanaudos neaiškios. Projekto priežiūra, išskyrus pavedimų atlikimo kontrolę (t.y. terminų kontrolę) nevyksta, kokybės modelio nėra, nėra ir kokybės kontrolės, išskyrus atskirų kokybės parametrų ekspertinius vertinimus, kurie yra atliekami *post factum*, jau sukūrus akto projektą. Kitaip tariant, šiuolaikiniai vadybos metodai [2] Lietuvos teisėkūroje kol kas yra visiškai ignoruojami. Nėra suvokta, kas yra kokybiškas teisės aktas ir kad kokybė brangiai kainuoja. Kaip pabrėžta darbe [10], aukštos kokybės įstatymus gali turėti tik labai turtinga valstybė. Tuo tarpu netgi Lietuvos teisėkūros tobulinimo rekomendacijose [12] siūloma „atlikti kompleksinį mokslinį teisės ir teisėkūros būklės Lietuvoje tyrimą (įvertinti situaciją) ir jo pagrindu parengti **optimalų** (pabraukta mūsų) teisėkūros modelį“. Beveik kiekviename teisėkūros problemas nagrinėjančiame darbe kalbama apie būtinybę kurti **kokybiškus** teisės aktus ir visiškai neužsimenama, kad kiekvieno teisės akto kokybę būtina planuoti ir kad tai turi būti daroma atitinkamo kokybės modelio pagrindu ir atsižvelgiant į tam aktui kurti skirtus išteklius. Atskiri teisės akto ir teisėkūros kokybės atributai yra nagrinėjami daugelyje darbų, pavyzdžiui, [1], [6], [10], [12], [13], [15], [18]. Vienok, mūsų nuomone, šiems nagrinėjimams trūksta sistemingumo. Mes manome, kad teisės akto kokybės samprata turėtų būti grindžiama bendrąja kokybės samprata, apibrėžta ISO 8402 standartu [7]. Sutinkamai su šiuo standartu, kokybė yra apibrėžiama kaip „objekto savybių visuma, įgalinanti tą objektą tenkinti išreikštus ir numanomus poreikius“. Kokybė yra nusakoma pasirinktojo kokybės modelio terminais. Kokybės modeliu vadinama objekto savybes nusakančių atributų visuma. Ši visuma paprastai turi hierarchinę struktūrą. Konkretaus objekto kokybė nusakoma užduodant žemiausiojo hierarchijos lygmens atributų reikšmes. Atributai esti prieštaringi, t.y. didėjant vienų atributų reikšmėms kitų atributų reikšmės mažėja. Kitaip tariant, visų poreikių maksimaliai patenkinti neįmanoma. Gerinant objektą kažkuriuo aspektu, jis gali blogėti kitais aspektais. Taigi, apie optimalumą čia galima kalbėti tiktai uždavus atributų prioritetus. Be to, optimali kokybė dažniausiai yra neįmanomas dėl objektui sukurti skirtų išteklių ribojimų. Taigi, bet kurio artifakto, taip pat ir teisės akto kokybė, nustatoma ieškant kompromiso tarp to, ko norėtųsi, ir to, ką galima pasiekti su turimomis lėšomis ir reikiamais terminais [14]. ISO 8402 standartas numato pačius bendriausius kokybės atributus, būdingus bet kurios prigimties artifaktams. Remiantis šiais atributais, kiekvienai inžinerijos šakai reikia konstruoti savą kokybės modulį. Vadovaudamiesi analogais, pavyzdžiui, [8], siūlome 1 lentelėje pateiktą teisės akto trijų lygmenų kokybės modelį.

1 lentelė. Siūlomas teisės akto kokybės modelis

Nr	Atributas[4]	Atributo paaiškinimas	Vertinimo būdai
1. Koncepciniai (funkciniai) atributai			
1.1	Racionalumas (angl. *rationality*)	Nusako, kokiu mastu teisės akto koncepcinės nuostatos išplaukia iš pagrindinio tikslo, kurio tuo aktu siekiama	Ekspertinis vertinimas
1.2	Tikslingumas (angl. *suitability*)	Nusako teisės akto koncepcinių nuostatų atitikimo visuomenės interesui laipsnį	Ekspertinis vertinimas
1.3	Išsamumas (angl. *completenes*)	Nusako tikimybę, kad visos teisės akto koncepcinės nuostatos apima visas reguliuotinas situacijas ("spragų" nebuvimas).	Testavimas
1.4	Nepertekliškumas (angl. *side-effect-less*)	Nusako tikimybę, kad teisės akto koncepcinės nuostatos neturi šalutinio (neplanuojamo) poveikio, t.y. kad to akto nebus galima pritaikyti situacijoms, kurių šiuo aktu reguliuoti nesiekiama.	Testavimas
1.5	Vidinė darna (angl. *consistency*)	Nusako tikimybę, kad teisės akto nuostatos neturi vidinių prieštaravimų.	Testavimas, reikalavimų analizė
1.6	Išorinė darna (angl. *compliance*)	Nusako teisės akto atitikimo Konstitucijai ir kitiems galiojantiems teisės aktams laipsnį	Ekspertinis vertinimas
1.7	Procesinė darna (angl. *interoperability*)	Nusako kokiu laipsniu teisės akto taikymo procedūros yra suderintos su kitų su tuo aktu susijusių teisės aktų taikymo procedūromis	Ekspertinis vertinimas
1.8	Adaptuojamumas (angl. *adaptability*)	Nusako teisės akto koncepnių nuostatų apibendrinimo laipsnį (užtikrinantį tų nuostatų pritaikomumą potencialiai galimoms naujoms situacijoms).	Teksto analizė

[4] Atributų pavadinimai kol kas nėra iki galo suderinti su lietuviškąja teisės terminija. Skliausteliuose pateikti (ten, kur tai galima padaryti) atitinkami sistemų inžinerijos terminai anglų kalba.

1 lentelė. Siūlomas teisės akto kokybės modelis (tęsinys)

Nr	Atributas	Atributo paaiškinimas	Vertinimo būdai
	2. Nekoncepcinio pobūdžio (nefunkciniai) atributai		
2.1	Įgyvendinamumas (angl. *feasibility*)		
2.1.1	Operacinis įgyvendinamumas (angl. *operational veto*)	Nusako išteklių, reikalingų operacinio pobūdžio inovaciniams slenksčiams (visuomeninė sąmonė, teisininkų kvalifikacija ir pan.) pašalinti, poreikį	Skaičiavimai, ekspertinis vertinimas
2.1.2	Procedūrinis įgyvendinamumas (angl. *technical veto*)	Nusako išteklių (laiko, lėšų ir kt.), reikalingų teisės akto taikymo mechanizmams sukurti, sąnaudas.	Skaičiavimai
2.2	Našumas (angl. *efficiency*)		
2.2.1	Taikymo trukmė (angl. *time behaviour*)	Nusako, kiek vidutiniškai tęsis konkretaus teisės akto taikymo priocesas (darant prielaidą, kad yra reikiamos lėšos ir neužimtas personalas).	Skaičiavimai
2.2.2	Taikymo kaina (angl. *resource behaviour*)	Nusako, kiek vidutiniškai kainuos konkretus teisės akto taikymas.	Skaičiavimai
2.2.3	Operacionalumas (angl. *operability*)	Nusako vidutines darbo sąnaudas, apimant taikymo priežiūrą bei kontrolę (matuojamas žmogaus darbo valandomis, galbūt struktūrizuotomis pagal specialistų profilį), reikalingas teisės aktui pritaikyti konkrečiu atveju	Skaičiavimai
2.2.4	Reaktyvumas (angl. *response time*)	Nusako per kiek laiko vidutiniškai gali būti atliktas konkretus teisės akto taikymas, atsižvelgiant į tam skiriamas lėšas ir tikėtiną personalo užimtumą.	Skaičiavimai
2.3	Patikimumas (angl. *releability*)		
2.3.1	Branda (angl .*maturity*)	Nusako tikimybę, kad, dėl klaidų, padarytų detalizuojant akto nuostatas arba dėl galimybių jas neteisingai interpretuoti, teisės akto taikymo pasekmės bus netokios kaip tikimasi	Testavimaas
2.3.2	Prižiūrimumas (angl. *fault tolerancy*)	Nusako tikimybę, kad teisės akto taikymo priežiūros mechanizmai sustabdys neteisingą akto taikymą, kilusį dėl nepakankamos to akto brandos.	Ekspertinis vertinimas
2.3.3	Neigiamų pasekmių pašalinamumas (angl *recoverability*)	Nusako tikimybę, kad bus pašalintos neigiamos pasekmės (pvz., atlyginta padaryta žala), kilusios dėl nepakankamos teisės akto brandos.	Ekspertinis vertinimas
2.3.4	Sauga (angl. *safety*)	Nusako tikimybę, kad teisės aktas neprisidės prie korupcijos augimo, kriminogeninės situacijos blogėjimo ar nesukels kitų pavojingų pasekmių.	Ekspertinis vertinimas
2.3.5	Stabilumas (angl. *stability*)	Nusako tikimybę, kad per nurodytą (pvz., 5 metų) laikotarpį teisės akto nereikės keisti ar papildyti.	Ekspertinis vertinimas
2.4	Panaudojamumas (angl. *usability*)		
2.4.1	Suprantamumas (angl. *understandability*)	Nusako piliečio pastangas (intelektas, išsilavinimas, laiko sąnaudos), reikalingas teisės akto nuostatoms ir jo taikymo procedūroms suvokti tokiu mastu, kad būtų suprasta ką ir kaip privalu daryti.	Ekspertinis vertinimas, sociologiniai tyrimai
2.4.2	Įsisavinamumas (angl. *learnability*)	Nusako pareigūnų pastangas (kvalifikacija, laiko sąnaudos), reikalingas teisės akto nuostatoms ir taikymo procedūroms suvokti.	Ekspertinis vertinimas, sociologiniai tyrimai
2.4.3	Konstruktyvumas (angl. *constructivity*)	Nusako, ar yra suformuluotos būtinos ir pakankamos teisės akto taikymo sąlygos ir ar apibrėžta konstruktyvi tų sąlygų nustatymo procedūra.	Testavimas
2.4.4	Konkretumas (angl. *definitivity*)	Nusako, ar atlikta teisės aktu numatytos normos yra susietos su aplinkybėmis tokiu mastu, kad bus užtikrintas nepriklausantis nuo subjektyvių veiksnių to teisės akto taikymą.	Testavimas
2.5	Korektiškumas (angl. *correctness*)		
2.5.1	Juridinis korektiškumas	Nusako teisės akto atitikimo juridinės technikos reikalavimams laipsnį.	Teksto analizė
2.5.2	Lingvistinis korektiškumas	Nusako, kokiu mastu teisės akto kalba tenkina lietuvių kalbos reikalavimus.	Teksto analizė

Mūsų siūlomas modelis nėra išbaigtas: trūksta vertinimo skalių ir procedūrų, svarstytina vartojama terminija. Be to, yra siūlomas tik teisės akto kokybės modelis, o, norint tobulinti teisėkūros procesą, yra reikalingas ir to

proceso kokybės modelis. Tai – tolimesnių tyrimų temos. Čia norime aptarti dar vieną labai svarbų projektų valdymo aspektą – žmogiškųjų išteklių vadybą.

Visuotinai yra manoma, kad teisės aktus, juolab įstatymus, rengia teisininkai. Iš tiesų yra visiškai kitaip. Pateiksime keletą citatų: „Vyrauja nuomonė, kad įstatymus rengti gali bet kas, o *teisė nėra mokslas*. Teisės aktus paprastai *rengia ne teisininkai*, nes dažnai įstaigose jų tėra vienas ar keli. Teisėkūra kol kas yra ne profesionalus, o mėgėjiškas užsiėmimas. ... Neretai teisės aktų ekspertizę atlieka teisės specialybės paskutinių kursų studentai, geriausiu atveju – prieš metus ar dvejus aukštąją mokyklą baigę jauni teisininkai. ... Jau yra susiformavusi praktika, kad neigiama teisininkų išvada dėl parengto projekto nėra pagrindas jo nesvarstyti.“ [1], „Vienas pagrindinių skiriamųjų Vakarų teisės tradicijos bruožų yra tas, kad kultivuoti teisę patikima profesionaliems teisės specialistams. O Lietuvoje? Remiantis apklausos duomenimis, galima daryti išvadą, jog teisininkai gana retai rengia teisės aktų projektus. Be to, ir teisės aktais nenustatomi minimalūs kvalifikacijos reikalavimai asmenims, kurie rengia norminių teisės aktų projektus. Valstybės tarnautojai, rengiantys teisės aktus, dažniausiai neturi tam nei elementarių teisinių žinių, nei techninių įgūdžių.“ [15], „...ministerijose ir vyriausybės įstaigose trūksta patyrusių, turinčių teisinį išsilavinimą valstybės tarnautojų, galinčių sėkmingai dirbti teisės projektų ruošime. ...Dažniausiai ministerijos bei valstybės įstaigose pradeda dirbti jauni, ką tik baigę aukštąsias mokyklas (kartais net paskutiniųjų kursų studentai) specialistai, neturintys patyrimo...Nėra nustatyta kriterijų specialistų ar ekspertų pritraukimui iš išorės ...Iškreipta eksperto-specialisto kvalifikacinės kategorijos suteikimo praktika...“ [9]. Šitokių citatų galima pateikti daug daugiau. Situacija susidarė dėl kelių priežasčių. Visų pirma, nepakanka teisėkūrai skiriamų lėšų [9], „valstybė nelaiko teisėkūros prioritetine veikla“[1], „pastebimai sumažėjo teisės aktų projektams rengti sudaromų komisijų“ [15]. Antra, dėl didelių atlyginimų skirtumų, kvalifikaciją įgyję valstybės tarnautojai, visų pirma teisininkai, pereina į geriau apmokamas vietas arba į privatų sektorių [1], [9]. Trečia, dėl dažnos vyriausybių kaitos vyksta nuolatinė tarnautojų kaita teisės aktų projektus rengiančiose institucijose. Situaciją siūloma spręsti užtikrinant, „kad teisės norminių aktų leidybos institucijose dirbtų kompetetingi, išmanantys savo darbą asmenys, profesionalai“ [5], sukuriant teisėkūros specialistų rengimo ir jų kvalifikacijos kėlimo sistemą [5], [9], [12], [15] peržiūrint atlyginimų politiką [1], steigiant specialią ekspertų instituciją, kuri atliktų įstatymų ekspertizę iki pateikimo Seimui [1], [5], [11], [12], o galbūt ir visų rengiamų įstatymų priežiūrą. Mūsų nuomone, atsižvelgiant į patirtį programinės įrangos kūrimo srityje, daugelį problemų būtų galima išspręsti konkurso tvarka perduodant dalies įstatymų projektų rengimą privačioms advokatų kontoroms. Tiesa, sutinkamai su *LR Įstatymų ir kitų teisės norminių aktų rengimo tvarkos įstatymo* 4-ju straipsniu, kol kas „teisės akto rengėjais gali būti valstybės valdžios institucijos paskirti ar konkurso tvarka parinkti asmenys ar jų grupė, taip pat asmuo arba asmenų iniciatyvinė grupė“. Yra ir kitų kliūčių, pavyzdžiui, bijomasi lobizmo.

2.3. Teisėkūros informacinė instrumentinė infrastruktūra

Teisėkūros informacinė infrastruktūra privalo palaikyti ir teisėkūros technologinius procesus, ir teisėkūros projektų vadyba. Bendruoju atveju ji apima:

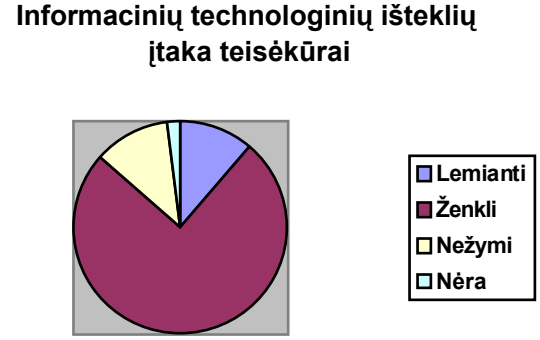
- specializuotos literatūros bibliotekas (kompiuterizuotas ir ne),
- specializuotas duomenų bazes (kompiuterizuotas ir ne),
- teisėkūros proceso dalyvių informacinio aptarnavimo procedūras,
- specializuotas priemones (kompiuterizuotas ir ne),
- kompiuterinę įrangą,
- mokymo programas ir kursus (kompiuterizuotus ir ne).

Nors ši infrastruktūra kuriama jau daugelį metų, kol kas ji nėra labai efektyvi. Veikia Teisinės informacijos centro sukurta kompiuterizuota teisinės informacijos sistema Litlex ir LR Seimo teisės aktų duomenų bazė. Pirmoji skirta profesionaliems vartotojams, antrąja gali naudotis ir neprofesionalūs vartotojai. Seimo tinklapyje pradėjo veikti pradinis Europos Sąjungos teisės duomenų bazės variantas, tiksliau, tos bazės užuomazgos. Naudojamos pavedimų kontrolės programos ir bendros paskirties programinė įranga (tekstų redagavimo sistemos, skaičiuokliai, kompiuterinis paštas ir pan.). Tai, galima sakyti, viskas. Sunku rasti „specialios literatūros ar metodikos, skirtos teisėkūros procesui“ [15], „valstybės tarnautojų kvalifikacijos kėlimas smarkiai atsilieka nuo planų“ [9]. Nėra netgi patikimos informacijos apie galiojančius teisės aktus. Pavyzdžiui, teisės normas nustatantys ministrų įstatymai neretai yra nepaskelbiami netgi „Valstybės Žiniose“, gauti informaciją apie vietos savivaldos institucijų priimtus norminius teisės aktus apskritai beveik neįmanoma.

Kita vertus, kaip rodo tyrimo metu atliktos sociologinės apklausos rezultatai[5] (1 pav), informacinės instrumentinės infrastruktūros poveikį suvokia absoliuti dauguma teisėkūros proceso dalyvių. Pasigendama specializuotų bibliotekų, duomenų bazių, šiek tiek mažiau, specializuotos programinės įrangos. Kokia programinė įranga yra reikalinga, žinoma jau senai [17], paminėtina čia ir plačiai žinoma sistema olandų sistema Leda [17]. Programų

[5] Tyrimas atliktas vykdant Jungtinių Tautų remiamą Lietuvos Vyriausybės projektą LIT/01/004 „Parama teisėkūros reformai“. Jį organizavo Teisės institutas, sociologas – V. Kalpokas.

sistemų inžinerijos patirtis rodo, kad gera instrumentinė įranga gali pakelti darbo našumą keliolika arba net keliasdešimt kartų. Teisės inžinerijoje toks šuolis galbūt kol kas dar neįmanomas. Tačiau padidinti darbo našumą keletą kartų čia irgi visiškai realu. Vienok, iš mūsų pateiktos medžiagos matosi, kad tam visų pirma reikia suformuluoti pragmatišką teisėkūros doktrina ir išspręsti mūsų aptartas technologines ir vadybos problemas.



1 pav. Apklausos respondentų nuomonė apie informacinės technologinės infrastruktūros įtaką teisėkūrai

3. Išvados

Lietuvos teisėkūra šiuo metu yra krizės būklėje. Teisėkūros procesas yra labai intensyvus, nespėjama laiku parengti reikiamų teisės aktų, daugelis jų parengiami nekokybiškai. Visa tai primena garsiosios „programavimo krizės“ apraiškas. Straipsnyje parodyta, kad susidariusias problemas tikslinga spręsti panašiu būdu, kaip tai buvo padaryta programų sistemų inžinerijoje, bandant išeiti iš „programavimo krizės“, t.y. traktuojant teisėkūrą kaip veiklos sistemą, konstruojant tą sistemą, vadovaujantis bendraisiais sistemų inžinerijos principais ir sukuriant efektyvią informacinę instrumentinę infrastruktūrą. Reikalinga visų teisėkūros sistemos lygmenų darna. Visų pirma, turi būti suformuluota pragmatiška teisėkūros doktrina, antra – suprojektuota tą doktriną įgyvendinanti technologija ir, trečia, sukurta efektyvi informacinė instromentinė infrastruktūra, skirta tai technologijai palaikyti.

Literatūros sąrašas

[1] **A. Dapšys, P. Ragauskas**. Lietuvos teisėkūros būklė ir perspektyvos. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 2000, 2(28), 31-54.

[2] Lietuvos teisėkūros tobulinimo rekomendacijos. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 2000, 2(28), 7-12.

[3] **R. Ruškytė**. Teisėkūros procesas ir jo tobulinimas. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 2000, 2(28), 55-62.

[4] Lietuvos Respublikos Prezidento Valdo Adamkaus kalba, pasakyta konferencijoje “Teisinės sistemos reforma: teisėkūros problemos”. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 2000, 2(28), 21-22.

[5] **G. Kaminskas**. PHARE parama Lietuvos Europos integracijai (SEIL Projektas). Subprojektas: Pagalba teisės aktų leidybos proceso tobulinimui. Studija. 1999

[6] **G. Balčiūnas**. Teisinės sistemos reformos eiga ir pagrindinės problemos. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 2000, 2(28), 23-29.

[7] **A. Čaplinskas**. Programų sistemų inžinerijos pagrindai. I dalis. *Matematikos ir informatikos institutas*, ISBN 9986-680-03-4 (1 dalis), Vilnius, 1996

[8] **V. Mikelėnas**. Naujų kodeksų privatinės teisės srityje svarstymo, priėmimo ir įgyvendinimo problemos. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 2000, 2(28), 63-77.

[9] **A. Čaplinskas, J.Misiūnas, R.Merkevičius, V.Pavlovas, V.Poškevičius**. Dar kartą apie baudžiamojo kodekso projektą. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 1997, 4(25), 34-60

[10] ISO/IEC 9126:1991(E): Information technology - Software product evaluation - Quality characteristics and guidelines for their use.

[11] **A. Kasemets**. Impact assesment of legislation for parliament and civil society: a comparative study. Second report. Draft version: 25.11. *ECPRD, Riigikogu*

[12] Guidelines on quality in the legislation. *Document G6, Danish Ministry of Justice*, 2000.

[13] **C.C. Barczyk**. Visuotinės kokybės vadyba. ISBN 9986-05-347-1, Vilnius, 1999.

[14] ISO 8402. Quality management and quality assurance vocabulary. Second edition, 1994-04-01

[15] **A. Ribikauskas, O. Vasilecas, A. Čaplinskas**. Conceptual models of the quality control process. *Mathematical Modelling and Analysis*, *Vilnius Gediminas Technical University, Institute of Mathematics and Informatics*, ISSN 1392-6292, Vol. 5, 2000, 143-152

[16] **A. Tjay Kwie Lim**. Information technology support for legislative development: tools for the legislative engineer. On-line paper. *The Australian National University*, *URL: http://actag.canberra.edu.au/actag/Expert/dev/legtool/ legtool2.html*, 1993

[17] W. Voermans. Computer-assisted legislative drafting in the Netherlands: the LEDA-system. *A National Conference on Legislative Drafting in the Global Village, November 16 & 17, 2000, Ottawa. Canadian Institute for the Administration of Justice*, on-line paper, URL: http://www.ciaj-icaj.ca/legal-drafting-2000/Voermans.htm

[18] **J. Žilys**. Teisėkūra ir konstitucinė justicija. *Teisės problemos, Teisės institutas,* ISSN 1392-1592, 2000, 2(28), 85-98.

Summary

This paper discusses the role of the information infrastructure in the Lithuania legislative system and the problems of the efficiency of this infrastructure. It demonstrates that an efficient infrastructure is impossible without the consistency of all levels (legislative theory, drafting procedures, infrastructure) of the legislative system. The paper suggests that the methods of systems engineering should be applied wider in the legislative engineering, discusses the quality of legal acts and proposes appropriate quality model. It is based partly on the results of sociological research.

VEIKLOS MODELIŲ SUDĖTIES ANALIZĖ

Audrius Lopata

Kauno technologijos universiteto informacijos sistemų katedra
Studentų g. 50, Kaunas
Vilniaus Universitetas KHF,Muitinės 8, 3000 Kaunas

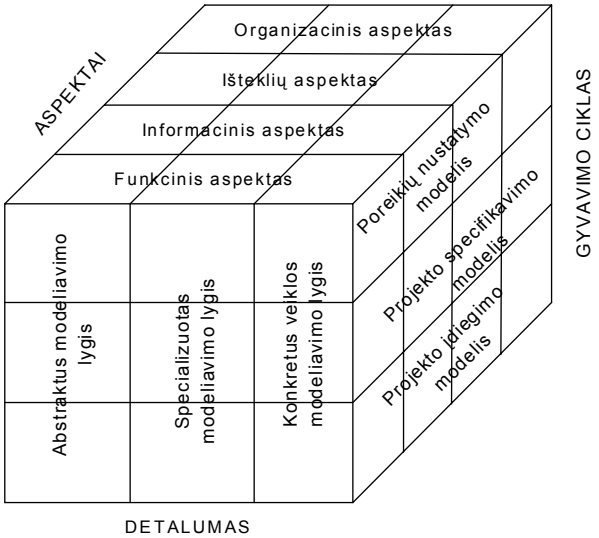
Straipsnyje analizuojami veiklos modeliavimo tarptautiniai standartai. Siekiama nustatyti būtinus veiklos modelio elementus, jų santykius, informacinius veiklos modelio elementų atributus. Praktinis tyrimo tikslas- suprojektuoti duomenų saugyklą, skirtą organizacijos veiklos metamodeliui.

Šiuo metu yra keletas skirtingų organizacijos modeliavimo metodų ir kalbų IDEF, OMT, UML, CIMOSA, ARIS ir kt. Tai sąlygojo tarpusavyje nesuderinamų programinių paketų, skirtų organizacijos veiklos modeliavimui atsiradimą (ARIS ToolSet, System Architect, FirstSTEP, CimTool ir kt.). Pagrindinės organizacijos modeliavimo mokslinės grupės, tokios kaip ODP, OMG, PSL/NIST, didžiausi programinės įrangos gamintojai (Oracle, Microsoft) bei pagrindinės standartizacijos organizacijos (ISO, CEN) stengiasi sukurti bendrą standartą, kuriuo remiantis būtų kuriami nauji organizacijos veiklos modeliavimo metodai, kalbos bei su jais suderinta programinė įranga. Pagrindiniai organizacijos veiklos modeliavimo standartai pateikti 1 lentelėje.

1 lentelė. Pagrindiniai organizacijos veiklos modeliavimo standartai

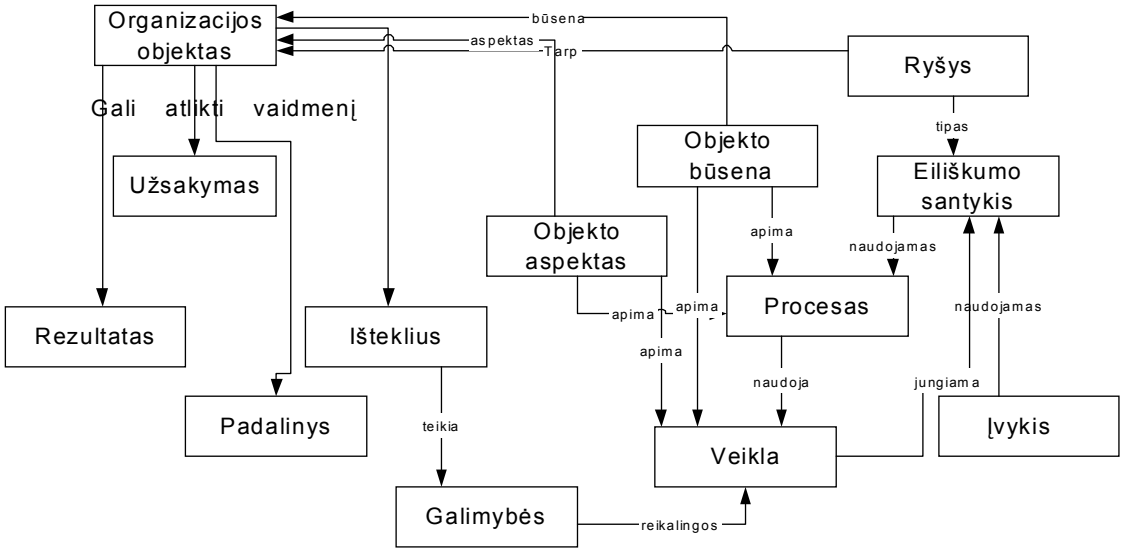
Standarto kodas	Standarto pavadinimas
ISO 14258	Concepts and Rules for Enterprise Models
ISO 15704	requirements for Enterprise reference architectures and methodologies
ISO 10314	Shop floor production model
ISO/IEC 15288	System life cycle processes

Europos standartizacijos komitetas (CEN), bendradarbiaudamas su tarptautine standartizacijos organizacija (ISO), remdamasis 1 lentelėje pateiktais standartais sukūrė CEN ENV 40003 ir CEN ENV 12 204 standartus, kuriuose apibrėžti pagrindiniai organizacijos veiklos modeliavimo principai. CEN ENV 40003 standartas yra sukurtas CIMOSA [5] modeliavimo metodo pagrindu. CEN ENV 40003 standarte veiklos modelio projektavimo procesas pateikiamas kaip kubas, kurio ašys aprašo modeliavimo aspektus, projektavimo gyvavimo ciklo etapus bei modelio detalumo lygius. (1 pav.)



1 pav. CEN ENV 4003 standarte naudojama principinė veiklos modeliavimo schema

CEN ENV 40003 standarte veiklos modeliavimui būtinas sudėtines dalis (konstruktus) apibrėžia CEN ENV 12204 standartas (2 pav.):

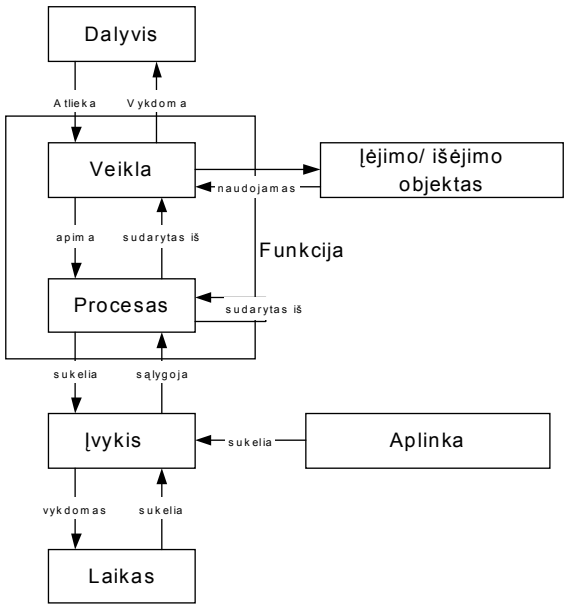


2 pav. Organizacijos modeliavimui būtini pagrindiniai konstruktai, apibrėžti CEN ENV 12204 standarte

Tačiau, kaip parodė praktika CEN ENV 40003 ir CEN ENV 12204 standartai pilnai nepatenkino nei organizacijos veiklos projektuotojų nei programinės įrangos gamintojų poreikių, todėl šiuo metu yra kuriamos naujos šių standartų versijos, o taip pat kuriami nauji standartai (2 lentelė), bei kalbos, tokios kaip UEML [4] (Unified Enterprise Modeling Language) (3 pav.).

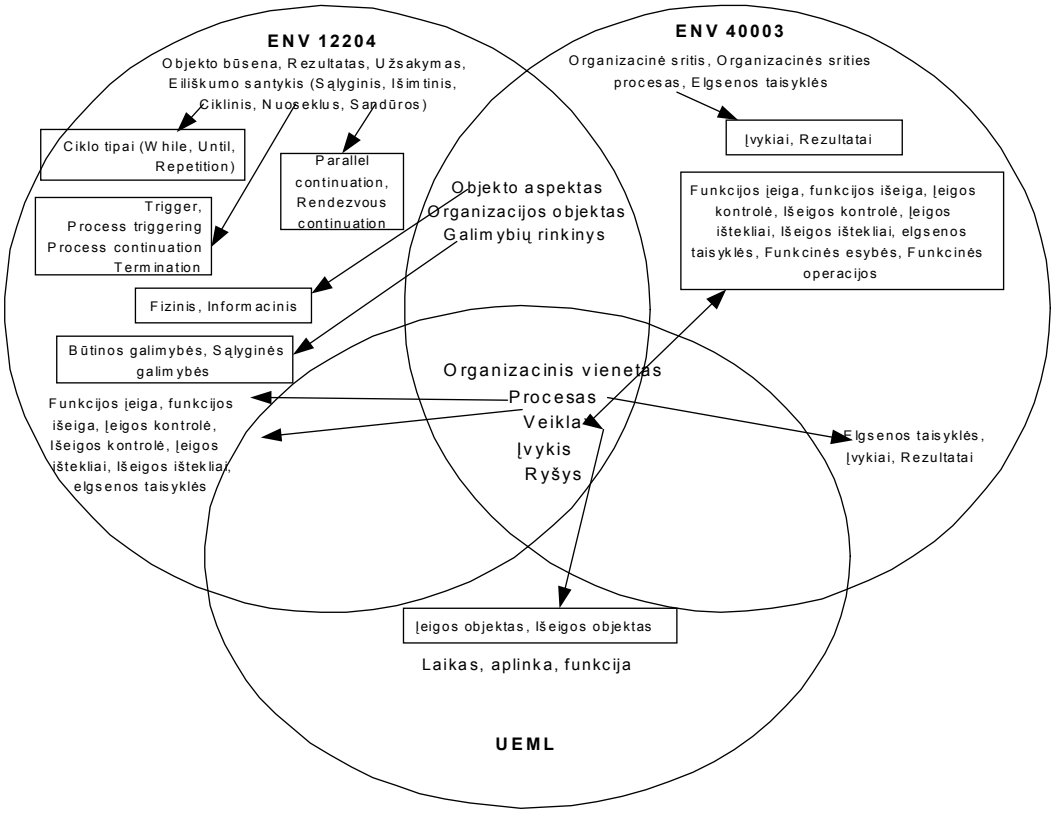
2 lentelė. Šiuo metu kuriami su organizacijos modeliavimu susiję standartai

Standarto santrumpa	**Standarto pavadinimas**
UEML	Universal Enterprise Modeling Language
PSL	Process Specification Language
XBRL	Extensible Business Reporting Language



3 pav. UEML principinė schema

Atlikus palyginamąją CEN ENV 40003, CEN ENV 12204, ir UEML standartų analizę išskiriamos pagrindinės bei papildomos sudėtinės dalys (konstruktai) būtinos organizacijos veiklos modeliavimui (4 pav.)



4 pav. ENV 12204, ENV 40003 ir UEML standartuose išskirtų pagrindinių konstruktų palyginimas

Analizuojant ENV 40003, ENV 12204 ir UEML standartus, organizacijos veiklos modeliavimo konstruktus (4 pav.) tikslinga skirstyti į tris pagrindines grupes: bendrus, dalinai bendrus ir unikalius (3 lentelė):

3 lentelė. Pagrindinės organizacijos veiklos modeliavimui būtinų konstruktų grupės.

Konstruktų grupės	**ENV 12204**	**ENV 40003**	**UEML**
Bendri	Organizacinis vienetas (Organisational unit)		
	Procesas (Business Process)		
	Veikla (Enterprise activity)		
	Įvykis (Event)		
	Ryšys (Relation)		
Dalinai bendri	Organizacijos objektas (Enterprise object)		-
	Objekto aspektas (Object View)		-
	Galimybių rinkinys (Capability set)		-
Unikalūs	Objekto būsena (Object State)	Organizacinė sritis	Laikas
	Rezultatas (Product)	Organizacinės srities procesai	Aplinka
	Užsakymas (Order)	Elgsenos taisyklės	Funkcija
	Eiliškumo santykis (Sequencing relationships)	-	-

Nei vienas iš nagrinėjamų standartų pilnai neatitinka organizacinės sistemos projektuotojo poreikių ne tik dėl riboto konstruktų kiekio ir funkcijų, bet ir todėl, kad nei viename standarte nėra apibrėžta veiklos modelio struktūra

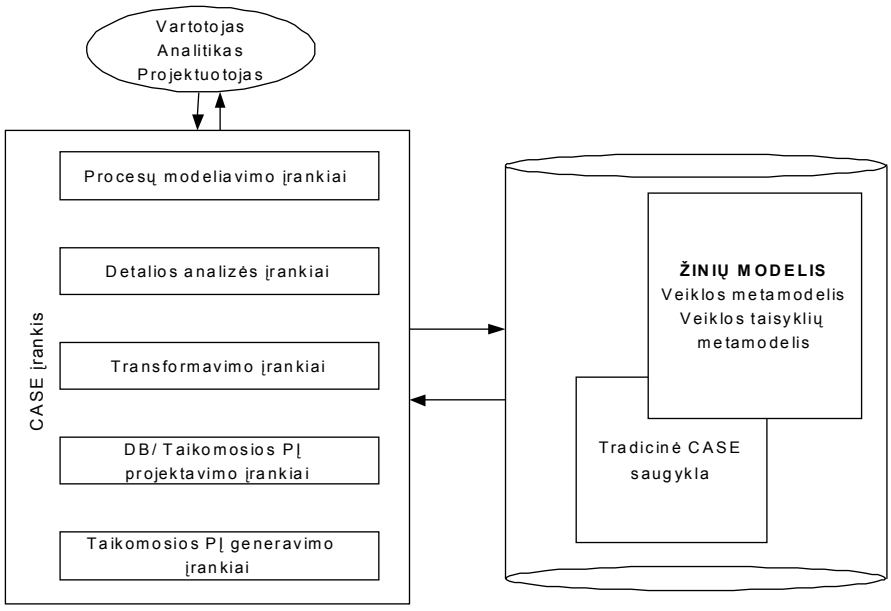
bei jos santykis su pagrindiniais konstruktais. Veiklos modelis [1] sukurtas remiantis Porter vertės grandinės [2] ir J.Henderson [3] modelių pagrindu [1]. Dalis veiklos modelio komponentų (Įvykis, Procesas, Ryšys, Organizacinis vienetas), atitinka bendrus bei dalinai bendrus (Funkcija) standartų konstruktus.

Nagrinėjamuose standartuose nėra detalizuota kaip turėtų būti apibrėžiamos veiklos modelio veiklos taisyklės. Šiuo metu vienas iš veiklos taisyklių klasifikavimo modelių, pretenduojančių tapti bendru standartu yra R. Roso modelis. Šio modelio pagrindu tikslinga kurti veiklos taisyklių saugyklą, kurios komponentai ir trumpas jų aprašymas pateikiamos 4 lentelėje [7]

4 lentelė. Veiklos taisyklių saugyklos sudėtis

Komponento pavadinimas	Komponento paskirtis
Veiklos taisyklės tipas	Saugomi duomenys apie naudojamus veiklos taisyklių tipus
Specialusis elementas	Saugoma informacija apie naudojamus specialiuosius elementus
Šaltinis	Saugomi duomenys apie informacijos šaltinius, su kuriais gali būti susieta viena ar daugiau veiklos taisyklių
Veiklos taisyklių grupė	Saugoma informacija apie abstrakčią veiklos taisyklių grupę
Duomenų tipas	Saugoma informacija apie dalykinę sritį charakterizuojančio modelio duomenų tipus
Veiklos taisyklė	Pagrindinė struktūros lentelė, kurioje saugoma informacija apie VT diagramą.
Pagalbinis komponentas Nr.1	Ryšio “daug su daug” tarp lentelių Saltinis ir Vtaisykle pagalbinė lentelė
Pagalbinis komponentas Nr.2	Ryšio “daug su daug” tarp lentelių DuomTipas ir Vtaisykle pagalbinė lentelė
Pagalbinis komponentas Nr.3	Ryšio “daug su daug” tarp lentelių SpecElem ir VTaisykle pagalbinė lentelė
Veiklos taisyklių sąsaja	Saugoma informacija apie dviejų taisyklių sąryšį

Projektuotojui, kuriančiam organizacijos informacinės sistemos projektą, būtų patogu naudotis ne tik projektavimui reikalinga informacija, esančia tradicinėje CASE įrankio saugykloje (repository), bet ir žinių modelio saugykla, kurioje saugomi organizacijos veikloje egzistuojančių veiklos taisyklių bei organizacijos veiklos metamodeliai. (5 pav.).



5 pav. Veiklos taisyklių saugyklos vieta CASE įrankyje

Organizacijos funkcinių IS projektavimo kompiuterizuotai sistemai (CASE sistemai) teikia papildomą informaciją tokios veiklos modelio klasių sąsajos:

- Pagal funkcijos pavadinimą gali būti generuojamas susietų su šia funkcija procesų sąrašas, pateikiama kiekvieno proceso sudėtis,
- Pagal funkcijos pavadinimą arba proceso pavadinimą gali būti generuojamas susietų padalinių (vartotojų) sąrašas;
- Pagal funkcijos pavadinimą gali būti generuojama darbų sekų modelio, taip pat vartotojo taikomųjų uždavinių modelio (*use case model*) pradinė sudėtis;
- Pagal funkcijos pavadinimą gali būti generuojamas esybių ryšių diagramos (arba klasių modelio) pradinė sudėtis.

Aptarėme tik dalį galimybių, kurias teikia organizacijos veiklos taisyklių bei CASE įrankio duomenų saugyklos informacinės sistemos kūrimo procese.

Literatūra

[1] **A.Lopata, S.Gudas.** Organizacijos informacinių išteklių identifikavimo būdas. - Konferencijos pranešimų medžiaga "Informacinės technologijos 2001", Kaunas,Technologija, 2001.

[2] **Turban E. Turban, E. McLean, J. Wetherbe.** Information Technology for Management. - John Wiley & Sons, Inc., 1999. ISBN0-471-17898-5.

[3] **Henderson, J., Thomas, J.** Aligning business and information technology domains, 1992 Hospital and Health Services Administrative, 37, 1, 71-87.

[4] Universal Enterprise modeling language, IFAC-IFIP Task Force, http://www.cit.gu.edu.au/~bernus/taskforce/archive/UEML-TF-IG.ppt (2001 10 30)

[5] **K.D.Tham.** CIM-OSA: Enterprise modelling. –Enterprise integration laboratory, University of Toronto, www.ie.utoronto.ca/EIL/entmethod/cimosa (2001 07 15)

[6] **S.Gudas**. Organisational System as a Hierarchy of Information Processes – Applications of Artificial Intelligence in Engineering VI (AIENG 91) , Computational Mechanics Publications, Southampton, Boston, 1991, p.p. 1037 -1050, ISBN 1-85342-141-X

[7] **R. Butleris, K. Kapočius.** Struktūrizuotų veiklos taisyklių saugyklos architektūra. Informacijos mokslai, Vilniaus universiteto leidykla, 2001, 17t. 46-56p.

Summary

This paper deals with value chain based enterprise modeling aproach and international standards of enterprise modeling. The enterprise model for identification of information resources is developed. The main idea of this survey is to design the data and knowledge repositories for enterprise activity metamodel.

ONTOLOGIJŲ PANAUDOJIMAS PROJEKTO REPOZITORIJUI INTELEKTUALIZUOTI*

Saulius Maskeliūnas

Matematikos ir informatikos institutas
Akademijos 4, 2600 Vilnius

Darbe pateikiama repozitorijaus samprata (apibrėžimas, ypatybės, realizavimo priemonės); supažindinama su repozitorijų metaduomenų standartu – bendruoju duomenų saugyklų metamodeliu CWM; nusakoma kas tai yra ontologijos; pristatomas ontologijų panaudojimo būdas projekto repozitorijui realizuoti (siūloma architektūra, orientacinės naudotinos ontologijų rūšys, repozitorijaus intelektualizavimo ypatumai).

1.Repozitorijaus samprata ir ypatybės

Repozitorijus gali būti suprantamas kaip:

- duomenų bazė, kurioje saugomi metaduomenys (t.y., duomenys apie duomenis, sisteminiai duomenys) kartu su
- šios duomenų bazės tvarkymo (t.y., joje esančių metaduomenų sukūrimo, saugojimo, apdorojimo ir panaudojimo) priemonėmis.

Patys repozitorijuje saugomi metaduomenys gali būti labai įvairių rūšių, pavyzdžiui: duomenų bazių schemos, esybių–ryšių (ER) modeliai, objektų modeliai, projektų aprašai, kryžminės nuorodos, pakartotinio naudojimo komponentų aprašai, įdiegtų komponentų aprašai, programų konfigūracijos, verslo taisyklės, verslo procesai, darbų srautų aprašai, dokumentų bibliotekų aprašai, tinklalapių žemėlapiai, naudotojų profiliai, žinių bazių struktūros, tinklų aprašai, ir kt. Projekto repozitorijuje yra saugomi duomenys apie vykdomo (realizuojamo, nagrinėjamo) projekto duomenis.

Repozitorijų idėja kilo iš duomenų bazių sistemose naudojamų duomenų žodynų–žinynų. Šiuo metu galima išskirti šias dažniausiai minimas repozitorijų rūšis:

- Programų kūrimo aplinkų (angl.: Software Development Environment, SDE) repozitorijai,
- Automatizuotos [programinių] sistemų inžinerijos priemonių (CASE sistemų) repozitorijai,
- Duomenų saugyklų (angl.: Data Warehouse, DW) repozitorijai,
- Integruoti įmonių repozitorijai, organizacijų atminčių (angl.: Organisational Memory, OM) repozitorijai.

Repozitorijų naudojimo skirtingose aplinkose svarbiausios funkcijos yra analogiškos, tai gali būti apibendrinta repozitorijaus naudojimo duomenų saugyklos kontekste bendromis schemomis (žr.: 1 ir 2 pav.).

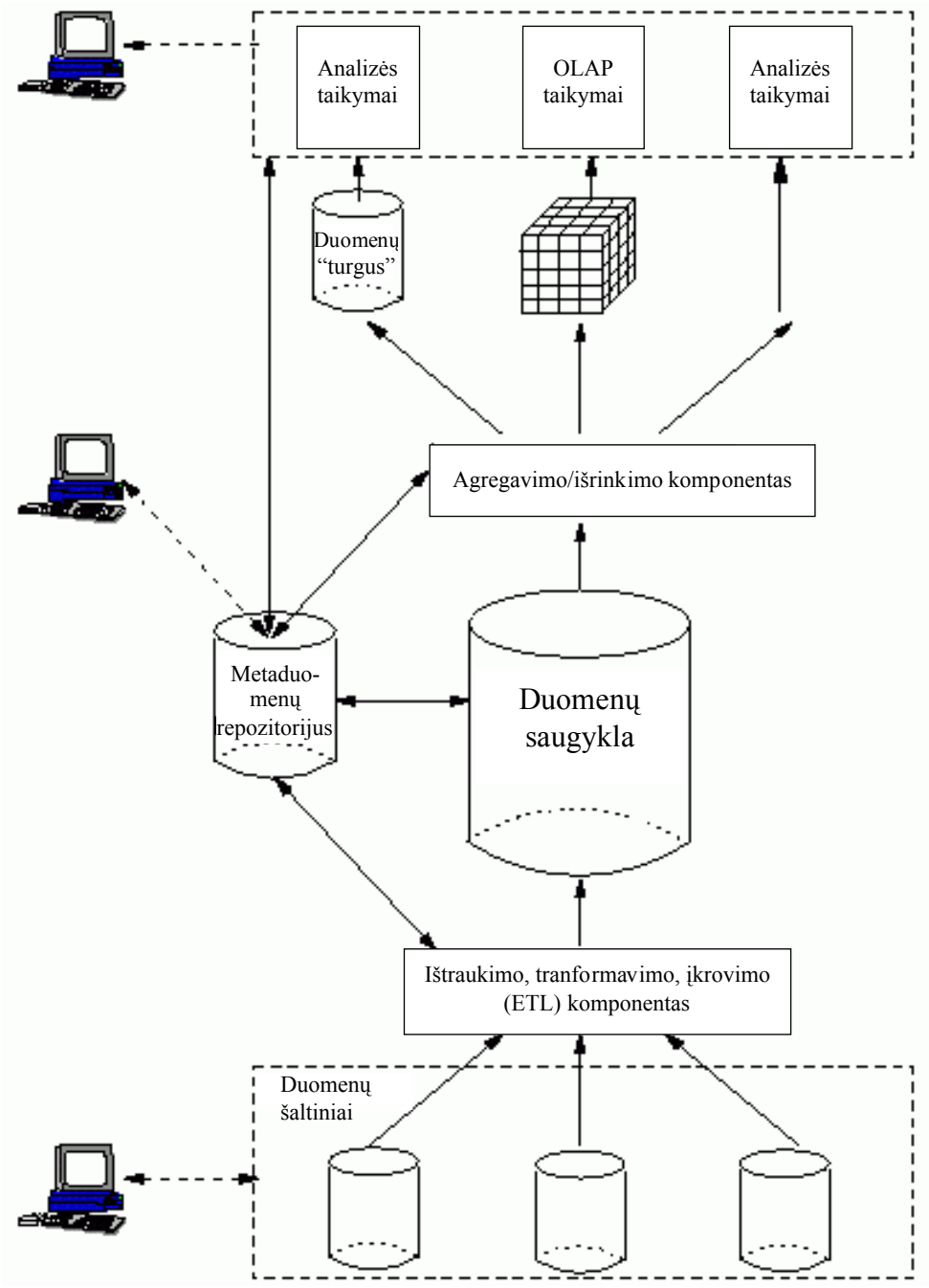
Svarbiausios repozitorijaus tvarkymo funkcijos yra:

- Repozitorijaus objektų (artifaktų) valdymas,
- Sąryšių tarp objektų (semantikos) valdymas,
- Pranešimų apie objektų pokyčius generavimas,
- Objektų versijų valdymas,
- [Nurodytos versijos objektų grupių] konfigūracijų valdymas,
- Naudotojams matomų repozitorijaus objektų konteksto valdymas,
- Dinaminis tipų aprašų ir jas realizuojančių klasių išplėtimas.
- Garsiausios bendrõs paskirties repozitorijų realizavimo priemonės yra:
- ASG–Rochade Repository [priklausanti bendrovei Allen Systems Group; prieš tai buvo R&O bendrovės nuosavybė, dar anksčiau: Viasoft][6],
- Universal Repository / Enterprise Content Server (UREP/ECS) [Unisys][7],

* Darbas atliekamas pagal MII Programų sistemų inžinerijos skyriaus 2000–2002 m. planinę temą "Programų sistemų, informacinių sistemų ir verslo sistemų inžinerijos intelektualizavimo problemos"

[6] http://www.asg.com/pdf/Gartner_ASG_Rochade_Report.pdf

- PLATINUM Repository [Computer Associates International, Inc.][8],
- Enabler Object Repository [Softlab North America][9],



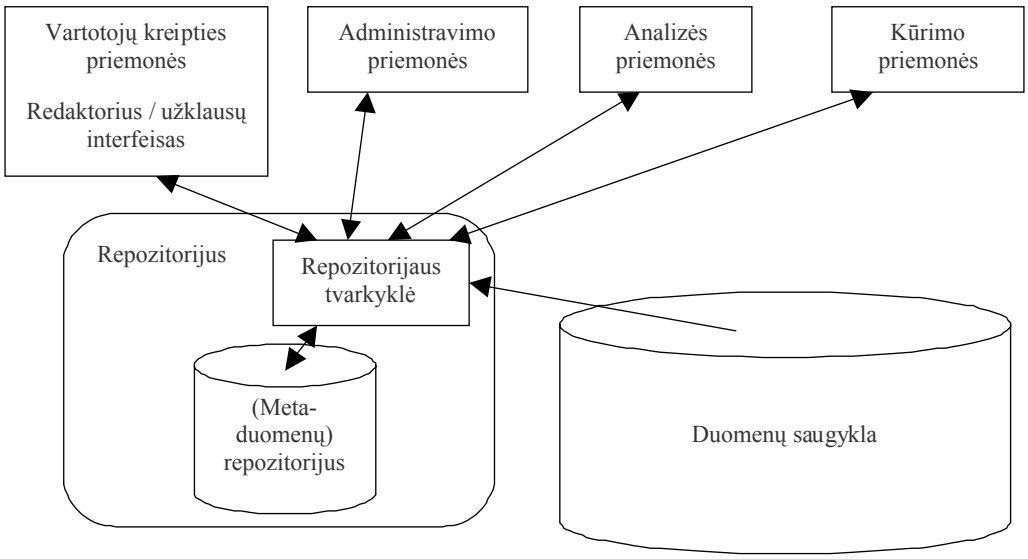
Čia: ——► – duomenų srautai,
- - - - - - ► – naudotojų kreiptis (gali būti, kad naudojant papildomas naršykles ar taikomąsias programas).

1 pav. Metaduomenų repozitorijaus vieta duomenų saugyklos kontekste **[7]**

- Microsoft Repository [Microsoft][10],

[7] http://www.unisys.com/marketplace/urep/whyurep.html
[8] http://www.platinum.com/products/factsht/repos_fs.htm
[9] http://www.softlabna.com/pages/espages/eor.htm
[10] http://research.microsoft.com/~philbe/EDOC Nov98.ppt

- Oracle SCM [Oracle Corporation][11],
- Automated Software Reuse Repository (ASRR) [EWA Inc][12].



2 pav. Repozitorijaus naudojimas

2. Repozitorijų metaduomenų standartas CWM

Realiose taikomosiose srityse labai dažnai yra svarbu, kad repozitorijaus metaduomenys būtų lengvai prieinami nepriklausomai nuo jų realizavimo platformos, naudojamų instrumentinių priemonių ir metaduomenų formatų, kad juos būtų galima bendrai naudoti, integruoti su kitų repozitorijų metaduomenimis. Skirtingose aplinkose naudojamų metaduomenų valdymo ir integravimo užtikrinimui yra reikalinga [1]:

- standartinė kalba formaliam metaduomenų struktūros ir semantikos apibrėžimui,
- standartinis keitimosi bendro naudojimo metaduomenimis (apibrėžtais standartine kalba) mechanizmas ir
- bendroji specifikacija, kuri standartinėje kalboje apibrėžia bendro naudojimo metaduomenų struktūrą ir semantiką duomenų saugyklų ir verslo analizės uždaviniuose.

Atitinkamai šiems reikalavimams sukurtas Bendrasis duomenų saugyklų metamodelis CWM[13] (angl.: Common Warehouse Metamodel), skirtas metaduomenų keitimuisi tarp skirtingų organizacijų, įvairių programinių priemonių, sistemų bei programinių produktų. CWM remiasi Unifikuota modeliavimo kalba UML[14] (nes kitos alternatyvos – XML, CORBA, Java ar pan. – turi nepakankamą išreiškiamąją galią) ir yra priimtas kaip Objektų valdymo grupės (OMG)[15] standartas. Bendrojo duomenų saugyklų metamodelio ypatybės (sąryšyje su UML) yra šios:

- CWM apibrėžtas naudojant MOF[16] (angl.: Meta–Object Facility; t.y., CORBA[17] metainformacijos valdymo bendrąjį standartą);
- Yra naudojamas UML profilis MOF metaduomenų architektūrai, tačiau pats CWM nėra UML profiliu;
- CWM branduolys yra UML poaibis, naudojamas objektiškai orientuotiems ištekliams;
- CWM apima metamodelį Esybių–Ryšių (ER) modelių aprašymui, todėl CWM standartą atitinkančių duomenų saugyklų, repozitorijų kūrėjai nebūtinai turi naudoti UML priemones.

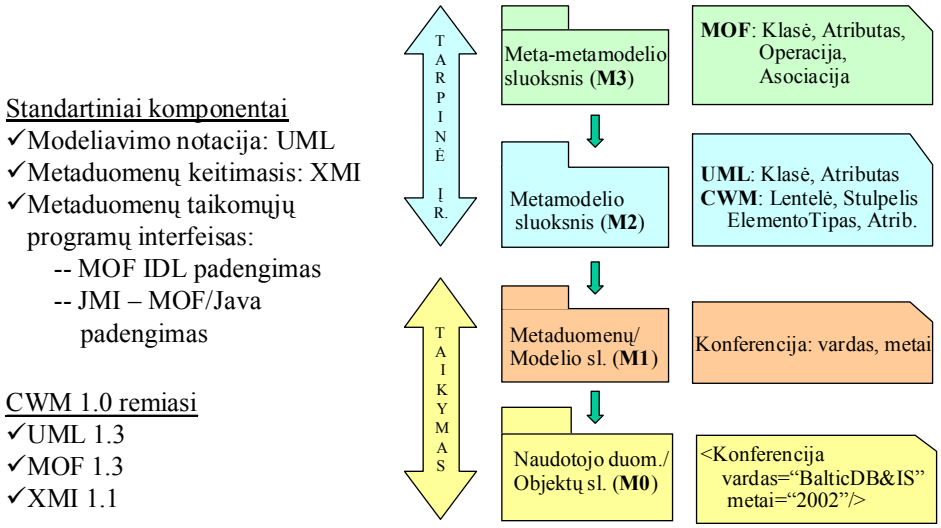
Bendrasis duomenų saugyklų metamodelis yra realizuotas remiantis bendra OMG metamodelių architektūra (žr.: 3 pav.); CWM metamodelio lygių ir komponentų schema pateikiama 4 pav.

[11] http://technet.oracle.com/products/repository/content.html
[12] http://wv.ewa.com/srr_overview.html
[13] http://www.omg.org/technology/cwm/ , http://www.cwmforum.org
[14] http://www.omg.org/technology/uml/
[15] http://www.omg.org/
[16] http://www.dstc.edu.au/Research/Projects/MOF/
[17] http://www.omg.org/gettingstarted/corbafaq.htm , http://cgi.omg.org/corba/beginners.html

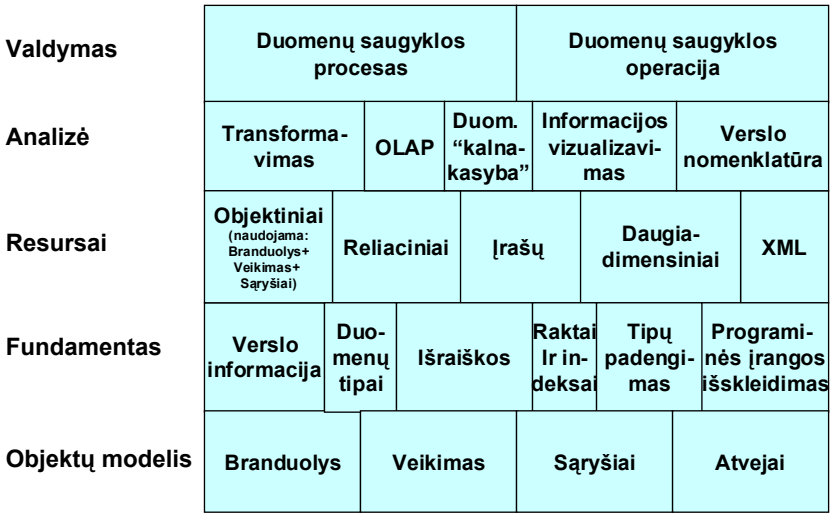
Bendrasis duomenų saugyklų metamodelis iš esmės yra meta–metamodelis, iš kurio gali būti generuojami metamodeliai, pvz. duomenų tipų aprašai (DTD); nebelieka poreikio palaikyti daugybę nesusijusių modelių. CWM įgalina paskirstytose heterogeninėse aplinkose lengvai keistis metaduomenimis tarp:

- Metaduomenų repozitorijų (pvz., Unisys UREP),
- Duomenų saugyklų priemonių įvairiose platformose, daugiamačių duomenų bazių (pvz., Hyperion Essbase),
- XML priemonių (pvz., DimensionEDI XMLmediator),
- Reliacinių duomenų bazių (pvz., IMB DB2),
- Duomenų struktūrų (įrašų) failuose (pvz., COBOL).

Tokia modelių transformacija yra vykdoma remiantis XMI[18] (angl: XML Metadata Interchange) specifikacija, iš MOF automatiškai generuojant XML duomenų tipų aprašus ir XML dokumentus. Du apsikeitimo CWM metaduomenimis pavyzdžiai (remiantis CWM ir UML deriniu bei naudojant CWM XML DTD) pateikiami [5].



3 pav. OMG bendroji metamodelių architektūra [8]



4 pav. CWM metamodelio lygiai ir komponentai

CWM atsiradimo istorijos svarbesni įvykiai yra šie:

- 1998 m. bendrovės IBM, Unisys ir Oracle pradėjo CWM kūrimo darbą;

[18] http://www.oasis–open.org/cover/xmi.html

- siekdama laimėti konkurencijoje Microsoft savo Atvirą informacijos modelį (OIM) perdavė Meta duomenų koalicijai (MDC) [19] ;
- 1999 rugsėjo mėn. aštuonios bendrovės (IBM, Unisys, Oracle, Genesis, Hyperion, NCR, UBS, Dimension EDI) paruošė būsimo CWM standarto paraišką;
- 2000 m. rugsėjo mėn. MDC nutraukia savo veiklą, pripažindama OMG pranašumą;
- 2000 m. vasario mėn. paruošta nauja CWM paraiškos redakcija;
- septynios bendrovės (IBM, Oracle, Unisys, Hyperion, Meta Integration, SAS, Adaptive) baigia kurti CWM pagalbinių priemonių demonstracinį rinkinį;
- 2001 m. balandžio mėn. baigtas ruošti CWM standartas;
- šiuo metu: darbo grupė ruošia nežymiai patikslintą CWM 1.1 versiją[20].

Pagrindinė CWM naudojimo idėja yra tai, kad CWM standartą tenkinančių repozitorijų (ar kitų programinių produktų) metaduomenys gali būti lengvai transformuojami į bet kokią aplinką ir paprastai vaizduojami atitinkamai numatomiems naudotojams ar taikymams.

3. Ontologijų samprata, ontologijų rūšys

Kompiuterijoje terminu "ontologija" vadinamas tam tikros srities sąvokų visumos specifikavimas išreikštu pavidalu (angl.: "explicit specification of a conceptualization" [4]).

Ontologijos apibrėžia nagrinėjimo srities:

- sąvokas, esybių (reiškinių, daiktų) tipus,
- sąvokų hierarchijas, esybių tipų tarpusavio sąryšius, priklausomybes,
- aksiomas, taisykles, dėsningumus apie esybių tipus ir sąryšius [nebūtina dedamoji],
- pavyzdinius atvejus [nebūtina dedamoji].

Pagal formalumą ontologijos skirstomos į:

- neformalias (pvz., terminų katalogai) ir
- formalias, kurios savo ruožtu būna:
  - aksiomatizuotos (pvz.: formalios mokslų teorijos, taisyklių ir freimų rinkiniai ekspertinėse sistemose, duomenų bazių koncepcinių schemų specifikacijos),
  - prototipais paremtos (terminologinės),
  - mišrios.

Pagal išreiškimo galią ontologijos skirstomos į:

- "lengvasvores" ontologijas (kurios išreiškia: sąvokas ir elementarius tipus, sąvokų hierarchiją, sąryšius tarp sąvokų) ir
- "sunkiasvores" ontologijas (kurios papildomai dar išreiškia ir: kardinalumo apribojimus, sąryšių klasifikaciją, galimybes manipuliuoti aksiomomis ir semantika, naudojant logikos formalizmus ir loginio išvedimo sistemas).
- Pagal paskirtį ontologijos skirstomos į [3]:
- žinių vaizdavimo ontologijas;
- bendrąsias ontologijas, visuotinai naudojamų sąvokų ontologijas;
- aukščiausio lygio ontologijas, meta–ontologijas;
- lingvistines ontologijas;
- nagrinėjimo sričių ontologijas;
- užduočių ontologijas, metodų ontologijas, taikomųjų programų ontologijas;
- ir kt.

Kompiuterijoje naudojamų ontologijų ypatybės plačiau yra aptariamos [6] straipsnyje.

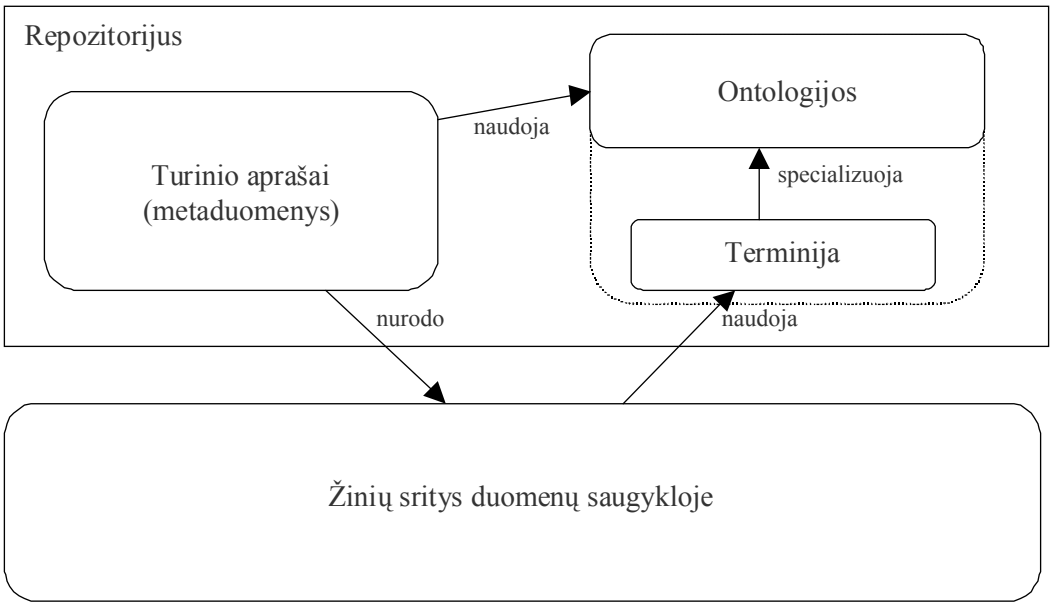
[19] http://xml.coverpages.org/mdc–oim.html
[20] http://www.omg.org/docs/ptc/01–10–03.htm

4. Ontologijų panaudojimo būdas projekto repozitorijui intelektualizuoti

Esamos svarbiausios ontologijų taikymų sritys yra:

- informacijos mokslas, bibliotekininkystė; profesinės terminijos standartizavimas;
- [matematinė] lingvistika; natūralios kalbos generavimas;
- ontologijomis pagrįsta programinių sistemų sąveika;
- duomenų bazių koncepcinės schemos, koncepcinis modeliavimas;
- žinių inžinerija, žinių bazių sistemos;
- brokeriai bei kitókios informacijos paieškos sistemos, pagrįstos ontologijomis;
- korporacinis žinių valdymas;
- ir kt.
- Pagrindiniai ontologijų naudojimo būdai yra:
- Neformalios ontologijos – terminų žodynai, skirti vieningam supratimui tarp tos pačios srities darbuotojų;
- Bendro naudojimo ontologijos ar tarpusavio atitikmenis turinčių ontologijų rinkiniai kreipčiai į duomenis;
- Ontologijos, apsprendžiančios interfeisus (įvairiose tikslinėse kalbose) su tam tikrais bendro naudojimo resursais;
- Ontologijos naudojamos paieškos mašinų indeksavimui;
- Ontologijos kaip keitimosi duomenimis formatai;
- Ontologijos skirtos konvertavimui ir įkomponavimui į taikomąsias programines sistemas.

Mūsų atveju siūloma architektūra ontologijų panaudojimui projekto repozitorijuje pateikiama 5 pav.



5 pav. Ontologijų panaudojimo projekto repozitorijuje architektūra

Orientacinės projektų repozitorijuje naudotinos ontologijų rūšys:

- Taikomoji sritis (žinios ir žinių profiliai);
- Projektas ir/arba įmonė (tikslai, procesai ir struktūra);
- Asmenys (naudotojų profiliai ar/ir modeliai, poreikių profiliai);
- Repozitorijus (t.y., meta–metaduomenys, duomenys apie repozitorijuje saugomus metaduomenis ir kt.).

Kyla klausimas: kiek ontologijų panaudojimas intelektualizuoja projekto repozitorijų? Intelektualizuotos infrastruktūros svarbiausios ypatybės yra aptariamos [2] straipsnyje. Intelektualizuota infrastruktūra yra gyvybinga, adaptyvi, reaktyvi, analitiška, generatyvi, gebanti analizuoti repozitorijuje saugomus duomenis, aktyviai asistuojanti vartotojui. Ontologijų naudojimas projekto repozitorijuje neabejotinai padidina repozitorijaus adaptyvumą (t.y., gebėjimą prisitaikyti prie konkretaus projekto ypatumų ir juo besinaudojančių asmenų darbo stiliaus bei kvalifikacijos)

ir įgalina analizuoti repozitorijuje saugomus duomenis (juos filtruoti, agreguoti bei atlikti jų kitokį semantinį apdorojimą). Kitas projekto repozitorijaus intelektualumo didinimo ypatybes taip pat galima būtų pasiekti naudojant ontologijas, tačiau tokiu atveju reikėtų naudoti sudėtingesnę specializuotą repozitorijaus bendrąją architektūrą, atitinkančią konkrečius projekto repozitorijui keliamus reikalavimus.

Būtina paminėti ir ontologijų nuolatinio tobulinimo, atšviežinimo svarbą. Kompiuterinėse sistemose atvaizduojama taikomųjų sričių informacija nuolatos keičiasi (t.y., atsiranda naujos veiklos, naujos organizacijų struktūros, nauji produktai, paslaugos, ir kt.). Ontologijos turi atspindėti šiuos pokyčius, įvedant naujas [ar išmetant, modifikuojant jau pasenusias] sąvokas ir sąryšius, atspindint naujas sąvokų prasmes. Tad, ontologijas naudojančių taikomųjų programinių sistemų veikimo kokybė tiesiogiai priklauso nuo naudojamų ontologijų "šviežumo". Ontologijų kurios nėra nuolatos palaikomos naudojimas gali turėti prasmę nebent tik labai lėtai besikeičiančiose taikomosiose srityse.

5. Išvados

Repozitorijai yra metaduomenų tvarkymo priemonės. Kuriant projektų repozitorijus būtina numatyti, kad keitimasis metaduominimis ateityje remsis CWM standartu.

Šiuo metu repozitorijuose ontologijos išreikštu pavidalu dar nėra naudojamos. Ontologijų panaudojimas projektų repozitorijuose didina repozitorijų adaptyvumą ir palengvina repozitorijuje saugomos (meta)informacijos analizę.

Literatūros sąrašas

[1] **D.T. Chang**. Common Warehouse Metamodel (CWM), UML and XML. Presentation at *4th Annual Meta Data Conference,* Washington DC, USA, March 19–23, 2000. http://www.cwmforum.org/cwm.pdf

[2] **A. Čaplinskas, A. Lupeikienė**. Sistemų inžinerijos intelektualizavimo problemos. *Informacinės Technologijos–2001*, *Kauno technologijos universitetas*, 2001, pp. 200–205.

[3] **A. Gómez–Pérez, , V.R. Benjamins**. Overview of knowledge sharing and reuse components: ontologies and problem–solving methods. *Proceedings of the IJCAI–99 workshop on Ontologies and Problem–Solving Methods: Lessons Learned and Future Trends*, Stockholm, Sweden. 1999, pp. 1–1 – 1–15. http://CEUR–WS.org/Vol–18/

[4] **T.R. Gruber**. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 1993, Vol. 5, No. 2, pp. 199–220. http://www–ksl.stanford.edu/knowledge–sharing/README.html#ontolingua–intro

[5] **S. Iyengar**. *CWM: the key to integrating enterprise business intelligence*. Whitepaper, Unisys Corporation, 2001. http://www.omg.org/news/releases/pr2000/cwm/whitepaper.htm

[6] **S. Maskeliunas**. Ontological engineering: common approaches and visualisation capabilities. *Informatica*, 2000, Vol. 11, No. 1, pp. 41–48. http://www.vtex.lt/informatica/htm/INFO195.htm

[7] **M. Staudt, A. Vaduva, T. Vetterli**. The role of metadata for data warehousing. *CH/IFuE technical report*, 1999, pp. 1–32. ftp://ftp.ifi.unizh.ch/pub/techreports/TR–99/ifi–99.06.ps.gz

[8] **D. Tolbert**. *The CWM experience implementing a UML–based Data Warehouse metamodel*. Presentation, Unisys Corporation. December 2001. http://www.cwmforum.org/CWM_UML_Experience.ppt

Summary

The notion of a repository (definition, features, and the means of realisation) is presented; the standard of repository metadata representation and interchange – Common Warehouse Metamodel (CWM) is introduced. The definition of ontologies and the main classifications of ontologies are given; the proposed architecture and main principles of using ontologies for project repository intellectualisation are presented. The use of ontologies in a project repository increases the adaptivity level of it, and facilitates the analysis of repository (meta) information.

NEKONTROLIUOJAMO MOKYMO NEURONINIŲ TINKLŲ PROGRAMINIŲ RESURSŲ LYGINAMOJI ANALIZĖ

Gintautas Dzemyda, Olga Kurasova

Matematikos ir informatikos institutas
Akademijos g. 4, 2600 Vilnius

Straipsnyje analizuojamos kelios nekontroliuojamo mokymo neuroninius tinklus realizuojančios programų sistemos: SOM-PAK, SOM-TOOLBOX, Viscovery SOMine Standard Edition, Nenet, Olandijos Groningeno universitete sukurta sistema bei daugiamačių duomenų integruoto vizualizavimo sistema. Dalis jų, kaip laisvai platinamos arba demonstracinės versijos, patalpintos internete. Nekontroliuojamo mokymo neuroninių tinklų išskirtinė savybė – duomenų sugrupavimas (surūšiavimas, klasterizavimas) pagal jų panašumą. Sistemos viena nuo kitos skiriasi ne tik realizavimo detalėmis, bet ir rezultatų pateikimo galimybėmis bei forma. Straipsnyje pateikta sistemų lyginamoji analizė. Kiekvienoje sistemoje neuroninis tinklas apmokomas tais pačias ekologiniais duomenimis, nusakančiais Suomijos pajūrio kopas ir jų vegetaciją. Nagrinėjant gautus rezultatus, atskleisti sistemų privalumai bei trūkumai grafinės sąsajos, vartojimo patogumo, rezultatų vizualizavimo, papildomų galimybių požiūriu.

1. Įvadas

Sparčiai vystantis dirbtinių neuroninių tinklų teorijai ir taikymams, vis daugiau sukuriama ir juos realizuojančių programų sistemų. Kai kurios iš jų, kaip laisvai platinamos arba jų demonstracinės versijos yra patalpintos ir visiems prieinamos internete. Būtina paanalizuoti sistemų įvairovę, išryškinti jų privalumus bei trūkumus, palyginti tarpusavyje. Šiame straipsnyje pateikta kelių nekontroliuojamo mokymo neuroninius tinklus [7] realizuojančių sistemų analizė. Analizuojamos šios programų sistemos:

N1. Olandijos Groningeno universitete sukurta sistema [6].

N2. Helsinkio technologijos universiteto Kompiuterių ir informacijos mokslų laboratorijos Neuroninių tinklų tyrimų centre sukurta **SOM-PAK** sistema [8].

N3. Helsinkio technologijos universiteto Kompiuterių ir informacijos mokslų laboratorijos Neuroninių tinklų tyrimų centre sukurta **SOM-TOOLBOX** sistema [1].

N4. Sistema **Viscovery SOMine** (Standard Edition Version 3.0 [10]), kurios bandomoji versija Internete platinama nemokomai.

N5. Sistema **Nenet** (Version 1.1 [4]), kurios demonstracine versija galima naudotis nemokamai.

N6. Daugiamačių duomenų integruoto vizualizavimo sistema [2, 3].

Visos paminėtos sistemos sukurtos Kohoneno algoritmo pagrindu. Kohoneno tinklas [7] yra neuronų masyvas, paprastai išdėstytų dvimačio tinklelio, dar vadinamo žemėlapiu arba lentele, mazguose. Kiekvieną žemėlapio elementą atitinka n-matis vektorius. Galima ortogonali (stačiakampė) arba heksagonali tinklo struktūra. Tyrimuose naudosimės pirmąja. Stačiakampis tinklelis (žemėlapis) yra sudarytas iš $m \times k$ elementų – n-mačių vektorių: m eilučių ir k stulpelių. Neuroninis tinklas apmokomas jam daug kartų pateikiant v skirtingų objektų, nusakomų n-mačiais vektoriais. Apmokant apskaičiuojami žemėlapio vektoriai ir tuos vektorius atitinkančių objektų numeriai, t.y. objektai pasiskirsto tarp žemėlapio elementų. Dalis žemėlapio elementų lieka nesusieti su jokiu objektu. Šis žemėlapis gali būti interpretuojamas kaip daugiamačių duomenų atvaizdavimas plokštumoje, nes galime vizualiai stebėti objektų išsidėstymą. Išskirtinė tokio atvaizdavimo savybė – duomenų sugrupavimas (surūšiavimas, klasterizavimas) pagal jų panašumą. Apmokymas pradedamas kiekvieną žemėlapio neuroną atitinkančiam n-mačiui vektoriui priskiriant atsitiktines pradines reikšmes. Po to žemėlapiui daug kartų „paduodami“ apmokymo aibės objektai, perskaičiuojant neuronus atitinkančius vektorius.

2. Sistemų programinė įranga

2.1. Sistema N1 (Groningeno universiteto sistema)

Sistemą **N1** sudaro failų rinkinys: **koh.c** – tai Kohoneno žemėlapių sudarymo programa, kompiliuojama *Borland C* arba *Borland C++*; **koh.exe** – tai sukompiliuota **koh.c** programa, veikianti tik *MS-DOS* terpėje; **kohview.cpp** – tai Kohoneno žemėlapių, sukurtų su **koh.exe**, vizualizavimo programa, veikianti tik *MS-DOS* terpėje, kompiliuojama *Borland C 3.1*; **kohview.exe** – tai sukompiliuota **kohview.cpp** programa.

Programai **koh.exe** būtina nurodyti tekstinio failo, kuriame įrašyta įvedamų (pradinių) vektorių, kuriais bus apmokomas žemėlapis, dimensija, jų pavadinimai ir komponentės, vieta kompiuterio diske. Be to, galima nurodyti: failo, į kurį bus surašomi rezultatai, vardą (jo nenurodžius bus sukuriami failai **koh_out.***); apmokymo epochų skaičių; sudaromo žemėlapio dydį; pradinio žemėlapio dydį (apmokymo metu žemėlapio dydis gali kisti); rezultatų pateikimo ekrane būdą. Visi parametrai nurodomi *MS-DOS* komandinėje eilutėje.

Programos **koh.exe** rezultate gaunamas ne vienas failas, o keli, tuo pačiu vardu, tačiau skirtingais plėtiniais. Failas **failo_vardas.ps** – tai **PostScript** failas, kuriame yra vizualus rezultatų žemėlapis. Faile **failo_vardas.map** nurodomos gauto žemėlapio ir įvedamų vektorių dimensijos, apmokyto žemėlapio vektorių – neuronų komponentės. Faile **failo_vardas.dif** pateikiami kvadratiniai skirtumai tarp kaimyninių apmokyto žemėlapio vektorių. Faile **failo_vardas.top** pateikiama apmokyto žemėlapio dydis, nurodomos įvestų vektorių koordinatės gautame žemėlapyje. Faile **failo_vardas.log** be gauto (apmokyto) žemėlapio ir įvedamų vektorių dimensijos pateikiamas vektorių ir apmokymo epochų skaičius, taip pat nurodoma, kaip kinta žemėlapio dydis priklausomai nuo epochos. Faile **failo_vardas.spn** pateikiami surūšiuoti didėjimo tvarka mažiausi galimi įvedamų vektorių skirtumai, pagal kuriuos kuriamas vektorių jungimo medis, t. y. visi vektoriai sujungiami linijomis (žr. 1 pav.).

Žemėlapių vizualizavimui, jei nėra priemonės peržiūrėti **PostScript** failus (pvz., **GSView**), sistemoje **N1** yra *MS-DOS* terpėje veikianti programa **kohview.exe**. Šiai programai nurodomas tik vienas parametras – rezultatų failų vardas be plėtinio. Rezultate matomas panašus žemėlapis, kaip faile ***.ps**. Dėl *Windows* ir *MS-DOS* operacinių sistemų skirtumų skiriasi ir linijų žemėlapiuose forma.

2.2. Sistema N2 (SOM_PAK)

Sistemoje **N2 (SOM_PAK)** pateiktos kelios programos: neuroninio tinklo pradinio žemėlapio inicijavimui iš atsitiktinių reikšmių (**mapinit**, **randinit**, **lininit**), jo apmokymui (**vsom)**, „neuronų – nugalėtojų“ sužymėjimui juos atitinkančiais pradinių vektorių pavadinimais (**vcal**), žemėlapio vizualizavimui (**visual, planes**, **umat**, **sammon**). Visos **SOM-PAK** programos atliekamos *MS-DOS* eilutėje nurodžius reikalingus parametrus, kurių šiame darbe nedetalizuosime. Komandinėje eilutėje nurodžius: *programos pavadinimas - help*, parodomas parametrų sąrašas ir jų trumpi aprašai. Vartotojo patogumui **SOM-PAK** sistemoje yra interaktyvi programa **fvind**, kuri skirta žemėlapio inicijavimui ir jo apmokymui. Šiai programai reikia nurodyti pradinių ir gautų (apmokyto žemėlapio) vektorių failų vardus, žemėlapio dimensijas, apmokymo skaičių, „kaimynystės“ funkciją, topologijos tipą. **fvind** rezultate gaunamas apmokytų duomenų failas. Belieka jame sužymėti „neuronų – nugalėtojų“ pavadinimus ir jį vizualizuoti.

2.3. Sistema N3 (SOM-TOOLBOX)

Sistemoje **N3 (SOM-TOOLBOX)** pateiktos funkcijos, sukurtos programa *Matlab 5*, kuri yra būtina sistemos naudojimui. **SOM-TOOLBOX**'e yra žemėlapio struktūros sukūrimo, duomenų paruošimo, duomenų nuskaitymo iš failo, pradinio žemėlapio inicijavimo, žemėlapio sukūrimo ir apmokymo, jo „neuronų – nugalėtojų“ sužymėjimo ir vizualizavimo bei kitos funkcijos. Toliau pateiktos tik kelios pagrindinės funkcijos:

sD = som_read_data(filename) – duomenų nuskaitymo iš failo funkcija, *filename* – failo vardas, *sD* – grąžinamas funkcijos struktūrinio tipo parametras.

mok = som_make (sD) – žemėlapio sukūrimo ir apmokymo funkcija, s*D* – perduodami pradiniai duomenys, *mok* – grąžinamas funkcijos struktūrinio tipo parametras – sukurtas ir apmokytas žemėlapis.

l = som_autolabel(sTo, sFrom) – žemėlapio elementų sužymėjimo funkcija, *sTo* – žemėlapis, kurį reikia sužymėti, *sFrom* – pradinių duomenų matrica, *l* – sužymėtas žemėlapis.

h = som_show(sMap) – žemėlapio vizualizavimo funkcija, *sMap* – vizualizuojamas žemėlapis. Rezultate gaunami keturių rūšių objektai: pagrindinis žemėlapis, kuriame įvedami vektoriai sudaro tam tikrus klasterius, komponentų objektai (kiekvienas komponentų objektas parodo vieno parametro reikšmes kiekviename žemėlapio elemente), tuščias objektas, kuriame gali būti sužymėti žemėlapio elementai, spalvų paletė.

h = som_show_add(mode, D) – sužymėto žemėlapio vizualizavimo funkcija, jei *'mode'='label'*, *D* – žemėlapis, kurį reikia sužymėti ir vizualizuoti.

2.4. Sistema N4 (Viscovery SOMine)

Sistema **N4 (Viscovery SOMine)** – tai programa su sava sąsaja, veikiančia *Windows* operacinėje sistemoje. Šia programa galima sukurti naują žemėlapį arba redaguoti jau sukurtą. Kuriant naują žemėlapį, reikia nurodyti apmokymo duomenų tekstinį arba *Microsoft Excel* failą, sukurtą pagal 3 skyriuje nurodytas taisykles. Prieš apmokymą reikia nurodyti vektorių komponentes, kuriomis apmokysime žemėlapį. Rezultate gaunami apmokomų duomenų klasteriai. Sukurtą žemėlapį galima redaguoti: po apmokymo susidariusius didžiuosius klasterius atskirti linijomis, sužymėti apmokyto žemėlapio elementus ir pan. Demonstracinėje versijoje išsaugoti sukurtą žemėlapį negalima, tačiau yra galimybė eksportuoti jį į grafinį failą.

2.5. Sistema N5 (Nenet)

Sistema **N5 (Nenet)** taip pat turi savo *Windows* grafinę sąsają. Naujo žemėlapio kūrimas suskirstytas į tris dalis: pradinio žemėlapio inicijavimas (*init*); žemėlapio apmokymas (*train*); žemėlapio testavimas (*test*). Inicijuojant pradinį žemėlapį iš atsitiktinių reikšmių reikia nurodyti: **žemėlapio dimensiją** (*SOM Dimensions*) (internete laisvai platinama tik demonstracinė **Nenet** versija, kuri leidžia inicijuoti žemėlapį ne aukštesnės kaip 10 dimensijos); **„kaimynystės" funkciją** (*Neighbourhood Function)*; **topologijos** (*Topology*) ir **inicijavimo** (*Initialization Type*) **tipus**; **atsitiktinių skaičių generatoriaus inicijavimo skaitmenį** (*Random Seed)*; **apmokymo duomenų failą** (*Initialization & Training Data File*). Rezultate gaunamas pasirinkto dydžio pradinis žemėlapis. Vėliau vykdomas žemėlapio apmokymas ir testavimas. Prieš tai reikia nurodyti apmokymų ir testinių duomenų failus bei kitus dydžius. Gautą žemėlapį galima išsisaugoti, vėliau atverti redagavimui.

2.6. Sistema N6 (Daugiamačių duomenų integruoto vizualizavimo sistema)

Sistema **N6** tai dviejų *Fortran* kalba parašytų programų rinkinys, kuriame integruotas Sammono algoritmas ir nekontroliuojamo mokymo neuroniniai tinklai. Sammono projekcija [9] yra netiesinio daugelio kintamųjų objektų (vektorių) atvaizdavimo žemesnio matavimo erdvėje metodas. Nekontroliuojamo mokymo neuroninio tinklo kombinacija su Sammono algoritmu pasiūlyta ir ištyrinėta darbuose [2], [3]. Skirtingai nuo [2], šiame darbe naudojama realizacija pasižymi viena papildoma savybe: vienos iteracijos metu visi *v* objektai tinklui paduodami atsitiktine tvarka. Ši realizacija pasižymi ypač paprastu rezultatų išvedimu – paprastos lentelės forma (žr. 1 lentelę). Gautoje lentelėje dalis langelių yra užpildyta analizuojamų objektų numeriais. Dalis langelių dažnai lieka tušti. Pagal tai, kaip objektai išsidėsto lentelėje, galima spręsti apie jų išsidėstymą *n*-matėje erdvėje. Tačiau lentelė neatsako į klausimą, ar labai nutolę vienas nuo kito objektai, esantys gretimuose lentelės langeliuose. Kiekvieną lentelės langelį atitinka *n*-matis vektorius. Netuščius lentelės langelius atitinkančius vektorius galima analizuoti Sammono algoritmu. Šiuo atveju neuroninis tinklas atliks tam tikrą duomenų rūšiavimą (klasterizavimą), o Sammono algoritmas pateiks rezultatus vizualiai. Darbe [2] tokia daugiamačių duomenų vizualizavimo metodų kombinacija yra ištirta ir pagrįsta eksperimentiškai.

3. Apmokymo duomenų failų struktūra

Kiekvienos sistemos pradinių duomenų, kuriais bus apmokomas neuroninis tinklas, failai turi būti sutvarkyti pagal tos sistemos individualius reikalavimus. Pradiniai duomenys turi būti tekstiniuose failuose (sistemai **N4** gali būti ir *Excel'io* faile). Failų pirmoje eilutėje turi būti nurodyta vektorių dimensija. Sistemai **N4** vektorių dimensijos nurodyti nereikia. Sistemai **N6** turi būti nurodyta ne tik vektorių dimensija, bet ir vektorių skaičius. Sistemai **N3** vektorių dimensija gali būti nurodoma ne tik įvedamų duomenų faile, bet ir duomenų nuskaitymo funkcijos ***som_read_data(filename, dimension)*** argumente. Sistemai **N1** vektorių komponentės turi būti surašytos į vieną eilutę arba stulpelį prieš komponentes nurodant vektorių pavadinimus. Kitose sistemose vektoriai turi būti surašyti į atskiras eilutes, kur vektorių komponentės sudaro stulpelius. Sistemų **N2**, **N3**, **N5** pradiniuose failuose paskutinis stulpelis – vektorių pavadinimai. Sistemoje **N4** neuroninis tinklas gali būti apmokomas ne tik pagal visas duomenų vektorių komponentes, bet ir pagal dalį pasirinktų komponenčių, o vektorių pavadinimai gali būti surašyti bet kuriame stulpelyje.

4. Tyrimų rezultatai

Kadangi internete platinamų sistemų **N4** (**Viscovery SOMine**), **N5** (**Nenet**) demonstracinės versijos yra ribotų galimybių, t.y. apmokymo vektoriai gali būti nedidelių dimensijų, todėl su šiomis sistemomis praktiniai eksperimentai nebuvo atliekami.

Analizuoti ekologiniai duomenys, nusakantys Suomijos pajūrio kopas ir jų vegetaciją [5]. Kopas charakterizuoja šie parametrai:

x_1 – atstumas nuo kranto; x_2 – aukštis virš jūros lygio; x_3 – dirvožemio PH;

x_4, x_5, x_6, x_7 – kalcio (CA), fosforo (P), kalio (K), magnio (Mg) kiekis;

x_8; x_9 – vidutinis smėlio skersmuo ir jo rūšis;

x_{10} – šiaurumas pagal suomišką koordinačių sistemą;

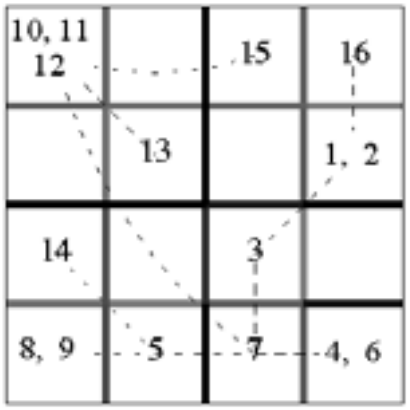
x_{11} – žemės kilimo greitis; x_{12} – jūros lygio svyravimas;

x_{13} – dirvožemio drėgnumas; x_{14} – šlaito tangentas;

x_{15} – smėlio paviršiaus dalis; x_{16} – medžiais apaugusi dalis.

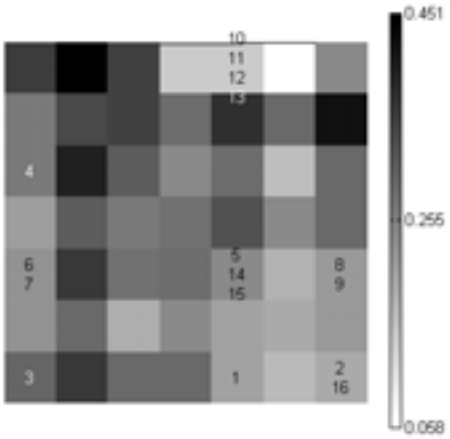
Darbe [5] yra pateikta šių 16 parametrų koreliacinė matrica. Naudojantis darbe [2] pasiūlytu metodu, gauti 16 objektų-vektorių (v=16), atitinkančių parametrams $x_1 - x_{16}$, sudarytų iš 16 komponenčių (n=16). Jie pateikti 3

lentelėje. Visi šie 16-mačiai vektoriai yra vienetinio ilgio. Būtent jie yra naudojami lyginamose sistemose kaip apmokymo ir analizės duomenų vektoriai. Žemiau 1-4 paveiksluose bei 1 lentelėje pateikti sistemomis **N1**, **N2**, **N3**, **N6** gauti žemėlapiai (lentelės) bei jų komentarai.



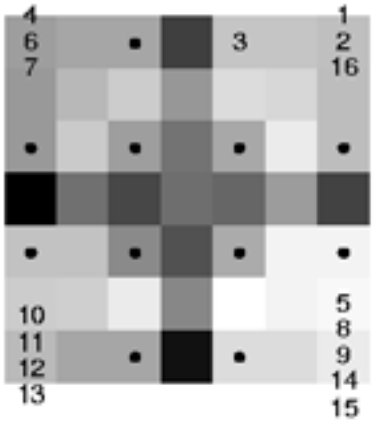
1 pav. Kopas apibūdinančių parametrų išsidėstymas 4×4 dydžio neuronų tinklelyje, gautame sistema N1

Sistemoje **N2 (SOM-PAK)** vektoriai išdėstomi $m \times k$ (šiuo atveju $m=k=4$) tinklelyje su tarpinėmis ląstelėmis (žr. 2 pav.). Tinklelio mazgai atskiriami nuo tarpinių ląstelių, pažymint juos taškais. Tarpinių ląstelių atspalvis nusako gretimų tinklo mazguose išdėstytų objektų panašumą; šviesesnės tarpinės ląstelės tarp tinklelio mazgų reiškia, kad mazguose išdėstyti objektai yra labiau panašūs nei tie, tarp kurių tarpinės ląstelės yra tamsesnės. Tinklelio mazgų spalva taip pat reiškia jų panašumą.



3 pav. Kopas apibūdinančių parametrų išsidėstymas 4×4 dydžio neuronų tinklelyje, gautame sistema N3 (SOM-TOOLBOX)

Tradicinis Kohoneno žemėlapis grupuoja panašius įvedamus vektorius. Iš atstumų tarp vektorių negalima spręsti apie jų panašumą, t. y. jei įvedamus vektorius A, B ir C atitinka gretimi tinklo neuronai, tai sunku pasakyti, kuris iš A ir B arčiau C. Sistemoje **N1** apmokytame žemėlapyje skaičiuojami atstumai tarp kaimyninius neuronus atitinkančių vektorių, ir pagal gautas reikšmes skirtingais atspalviais nuspalvinamos tuos neuronus skiriančios sienelės. Tamsiausia spalva rodo didžiausią skirtumą, šviesiausia – didžiausią panašumą (žr. 1 pav.). Taip pat sistemoje **N1** skaičiuojami mažiausi atstumai tarp apmokymo objektus atitinkančių vektorių ir pagal juos kuriamas vektorių jungimo medis, t. y. visi vektoriai sujungiami linijomis. Objektai jungiami linijomis minimizuojant atstumą tarp jų – kuo linija vientisesnė, tuo atstumas tarp tų objektų yra mažesnis.



2 pav. Kopas apibūdinančių parametrų išsidėstymas 4×4 dydžio neuronų tinklelyje, gautame sistema N2 (SOM-PAK)

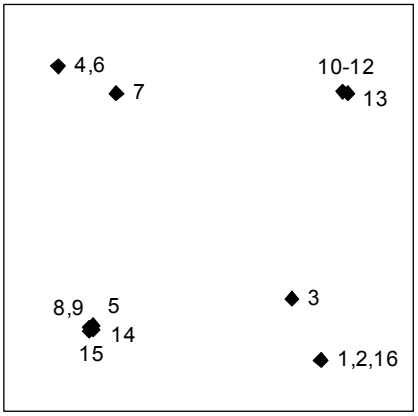
Sistemoje **N3 (SOM-TOOLBOX)** buvo inicijuojamas ir apmokomas 4×4 žemėlapis (tinklelis). Tačiau, kaip matyti 3 pav., rezultate gautas 8×8 tinklelis. Čia kaip ir sistemoje **N2** gaunamas tinklelis su tarpinėmis ląstelėmis, tačiau jos niekaip neatskirtos nuo tinklelio mazgų. Tarpinių ląstelių spalva, pagal jos skaitinį dydį, nurodytą šalia esančioje spalvų paletėje, nusako gretimų vektorių panašumą. Tinklelio mazgų spalva taip pat reiškia juose esančių vektorių panašumą.

Programų rinkiniu **N6** gaunama paprasta lentelė, kurios langeliai yra užpildyti „vektorių–nugalėtojų“ pavadinimais (žr. 1 lentelę). Programų rinkinyje **N6** nėra specialios programos Sammono projekcijai automatiškai gauti. Tai paliekama daryti pačiam vartotojui *Excel*'io priemonių pagalba, tačiau tai savotiškas privalumas, nes vartotojas gali keisti žemėlapio dydį, šriftą, spalvas.

Gautuose žemėlapiuose ir lentelėje matyti susidarę keturi kopas charakterizuojančių parametrų klasteriai. Galbūt nuo klasterio $\{x_1, x_2, x_3, x_{16}\}$ kiek atsiskyręs $\{x_3\}$. Tai matyti 2, 3 ir 4 pav. Neabejotinai tvirtas klasteris $\{x_{10}, x_{11}, x_{12}, x_{13}\}$, kuris susidaro visuose neuroninių tinklų ir Sammon'o žemėlapiuose. (žr.1-4 pav., 1 lentelę). Gana tvirti klasteriai $\{x_5, x_8, x_9, x_{14}, x_{15}\}$, $\{x_4, x_6, x_7\}$ nors 1 pav. Parametras $\{x_{15}\}$, „pabėga“ nuo savo grupės.

1 lentelė. Kopas apibūdinančių parametrų išsidėstymas 4×4 dydžio neuronų tinklelyje (lentelėje), gautame sistema N6.

4,6		5	8,9
7		15	14
3			
1,2,16		13	10,11,12



4 pav. Neuroninio tinklo kombinacija su Sammon'o algoritmu

5. Išvados

Tyrimai parodė, kad šiuo metu yra nemažai nekontroliuojamo mokymo neuroninius tinklus realizuojančių sistemų, turinčių savo privalumų ir trūkumų (jie pateikti 2 lentelėje). Iš 1-4 paveikslų ir 1 lentelės matyti, kad visomis sistemomis gauti panašūs rezultatai. Patogiausia interpretuoti sistema **N6** gautus rezultatus (žr. 1 lentelę kartu su 4 pav.). Sistemomis **N2** ir **N3** gautų žemėlapių vizualizavimo būdas yra labai panašus. **N3** pranašumas tas, kad prie žemėlapio pateikiama atspalvių paletė, su joje nurodomais skaitiniais dydžiais. Profesionalams labiau tinkamos yra sistemos **N3**, **N4**, **N6**, mažiau įgudusiems vartotojams – **N5**.

2 lentelė. Analizuojamų sistemų privalumai ir trūkumai

Sistema	Privalumai	Trūkumai
N1	Gaunamus rezultatus gana lengva interpretuoti. Nereikia įvedinėti daug parametrų.	Galima apmokyti tik kvadratinį žemėlapį. Mažai keičiamų parametrų. Veikia tik MS-DOS terpėje. Parametrus būtina nurodyti komandinėje eilutėje.
N2	Galima keisti nemažai parametrų. Yra interaktyvi programa, atliekanti pradinio žemėlapio inicijavimą ir jo apmokymą.	Veikia tik MS-DOS terpėje. Gaunamus rezultatus gana sunku interpretuoti. Parametrus būtina nurodyti komandinėje eilutėje.
N3	Daug įvairių funkcijų. Įvairūs rezultatų vizualizavimo būdai.	Būtina *Matlab 5* sistema, bei jos minimalus žinojimas.
N4	Veikia Windows sistemoje. Turi savo grafinę sąsają.	Bandomoji versija ribotų galimybių.
N5	Veikia Windows sistemoje. Turi savo grafinę sąsają. Patogi vartotojui, net pradedančiajam.	Bandomoji versija yra ribotų galimybių.
N6	Aiškus, paprastas rezultatų pateikimo būdas. Gautus rezultatus lengva suvokti ir interpretuoti.	Faktiškai tai yra programų rinkinys, darbui su kuriuo reikia papildomų žinių ir mechaninių veiksmų. Galima apmokyti tik kvadratinį žemėlapį.

3 lentelė. Kopas charakterizuojančių parametrų vektoriai

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x_1	0,208	0,541	0,573	-0,252	0,043	-0,052	-0,020	-0,025	-0,153	-0,089	-0,009	0,055	-0,458	-0,123	0,083	0,000
x_2	0,135	0,402	0,599	-0,152	-0,003	-0,107	0,255	-0,210	-0,397	0,104	-0,168	-0,030	0,337	0,067	-0,046	-0,001
x_3	0,364	0,593	0,010	-0,024	0,076	0,082	-0,477	0,301	-0,070	-0,238	0,177	0,064	0,103	0,126	-0,247	0,001
x_4	0,466	0,422	-0,477	0,001	-0,152	-0,009	0,400	-0,126	0,119	0,088	-0,119	-0,056	-0,103	-0,164	-0,319	0,000
x_5	0,215	0,069	-0,005	0,493	0,515	-0,104	-0,285	-0,536	0,064	0,173	-0,028	0,130	-0,060	0,058	-0,050	-0,002
x_6	0,391	0,571	-0,478	-0,040	-0,113	0,027	0,201	0,015	0,050	-0,005	-0,066	0,344	0,023	0,231	0,240	0,000
x_7	0,601	0,447	-0,342	0,020	0,095	-0,007	-0,218	-0,015	-0,063	0,080	0,084	-0,350	0,131	-0,226	0,237	0,000
x_8	0,076	0,102	0,199	0,719	-0,291	0,030	0,283	-0,004	-0,143	0,002	0,489	-0,020	-0,034	0,022	0,018	0,002
x_9	0,176	0,084	0,259	0,706	-0,226	0,009	-0,111	0,257	0,095	-0,111	-0,490	-0,012	0,008	-0,061	0,037	-0,002
x_{10}	0,858	-0,438	0,106	-0,080	-0,015	-0,016	0,063	-0,068	0,031	-0,188	0,025	-0,017	0,000	0,044	0,004	-0,082
x_{11}	0,803	-0,355	0,072	-0,056	0,004	-0,067	0,094	-0,103	0,056	-0,208	-0,048	-0,248	-0,091	0,274	0,004	0,055
x_{12}	0,716	-0,447	0,144	-0,074	-0,007	0,063	-0,013	-0,012	-0,008	-0,099	0,078	0,346	0,148	-0,312	0,009	0,037
x_{13}	0,573	-0,284	0,052	-0,048	-0,112	-0,075	-0,149	0,322	-0,148	0,632	-0,003	0,034	-0,099	0,091	-0,039	0,000
x_{14}	0,059	-0,006	0,009	0,144	0,782	-0,039	0,418	0,432	-0,023	-0,029	0,008	-0,012	0,005	-0,017	0,008	0,000
x_{15}	0,085	0,004	0,114	-0,006	0,087	0,975	0,040	-0,075	-0,019	0,090	-0,050	-0,042	-0,022	0,033	0,004	0,000
x_{16}	0,108	0,319	0,561	-0,132	-0,014	-0,034	0,062	0,028	0,701	0,167	0,123	-0,029	0,113	0,014	0,021	0,000

Literatūros sąrašas

[1] **E. Alhoniemi, J. Himberg, J. Parhankangas, J. Vesanto**. SOM Toolbox for Matlab 5. *Helsinky University of Technology, Report A57, Libella Oy Espoo*, 2000, http://www.cis.hut.fi/projects/somtoolbox/index.htm.

[2] **G. Dzemyda**. Visualization of a set of parameters characterized by their correlation matrix. *Computational Statistics and Data Analysis*, 2001, Vol. 36, No. 1, p. 15-30.

[3] **G. Dzemyda**. Vizualization of environmental data via analyzis of correlations. *IFAC Workshop on Modeling and Control in Environmental Issues, Yokohama, Japan*, August 22-23, 2001. p. 79-83.

[4] **P. Hassinen, J. Elomaa, J. Rönkkö, J. Halme, P. Hodju**. Screen shots taken from program called Nenet v1.1a, Neural Networks Tool. http://www.mbnet.fi/~phodju/nenet/General.html.

[5] **P. Hellemaa**. The Development of Coastal Dunes and Their Vegetation in Finland. *Dissertation. Fenia 176: 1, Helsinki. ISSN 0015-0010.* 1998. http://hul.helsinki.fi/elbanco/julkaisut/mat/maant/vt/developm/index.htm.

[6] **P. Kleiweg**. Neurale netwerken: Een inleidende cursus met practica voor de studie Alfa-Informatica, Master's thesis. *Rijksuniversiteist Groningen*, 1996, http://odur.let.rug.nl/~kleiweg/kohonen/kohonen.html.

[7] **T. Kohonen**. Self-Organizing Maps, 3nd ad. *Springer series in information sciences, Springer-Verlag*, 2001, Vol.30.

[8] **T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen**. SOM_PAK: The Self-Organizing Map Program Package. *Helsinki University of Technology, Laboratory of Computer and Information Science, Neural Networks Research Centre. Technical Report A31, FIN-02150 Espoo, Finland*. 1996, http://www.cis.hut.fi/research/som_lvq_pak.shtml.

[9] **J. W. Sammon**. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 1969, Vol. C-18, p. 401–409.

[10] Viscovery SOMine Standard Edition Version 3.0. http://www.eudaptics.com/technology/somine.html

Summary

In the paper, we analyze the software that realizes the Self-Organizing maps: SOM-PAK, SOM-TOOLBOX, Viscovery SOMine Standard Edition, Nenet, system created at Groningen University in Netherlands, combination of the Self-Organizing Map and Sammon's mapping. A part of the software may be found in the Internet. They are freeware or shareware or demo. The Self-Organizing maps assist in data clustering and analyzing of data similarities. The software is different one from other in the realization and the visualization possibilities. The data on coastal dunes and their vegetation in Finland are used for the experimental comparison of the software. The systems' similarity and their difference, advantage and imperfection are noticed.

VEIKLOS MODELIAVIMO METODŲ ANALIZĖ REIKALAVIMŲ SPECIFIKAVIMO ASPEKTU

Tomas Danikauskas, Rimantas Butleris

Kauno technologijos universitetas, Informacijos sistemų katedra
Studentų 50–308, LT–Kaunas

Parodomos funkcinių reikalavimų specifikavimo metodo galimybės veiklai modeliuoti, siekiant surinkti ir specifikuoti reikalavimus kuriamai informacijos sistemai. Palyginimo tikslu, apžvelgiamas objektinės orientacijos metodas MERODE, išryškinant pastarojo savybes, kurias reikia įvertinti formuojant reikalavimus sistemai funkcinių reikalavimų specifikavimo metodo pagrindu.

1. Įvadas

Sparčiai besivystant programinės įrangos kūrimo technologijoms, svarbu pasiekti ir spartesnį šių technologijų diegimą. Tačiau padidinta sistemos kūrimo bei diegimo sparta įneša į visą procesą didesnę tikimybę suklysti, jei šis procesas nėra griežtai kontroliuojamas. Todėl galima teigti, kad norint pasiekti didesnį našumą, reikia atlikti griežtesnę kokybės kontrolę.

Didelių informacijos sistemų (IS), atliekančių įmonių kompiuterizuotų valdymo sistemų funkcijas, projektavimas ir diegimas, kad ir kaip būtų norima pagreitinti diegimo procesą, užtrunka net kelis metus. Projektavimo kokybė tokiuose projektuose ypač svarbi, tačiau klaidos, padarytos ankstyvųjų sistemos kūrimo etapų metu, dažniausiai aptinkamos tik pradedant diegti sistemą ar net ją eksploatuojant. Šias klaidas sunku ištaisyti dėl didelių sistemos programinio kodo apimčių, be to, tai reikalauja daug papildomų sąnaudų, kas padidina IS kūrimo kaštus ir sumenkina eksploatuojamos sistemos teikiamą naudą.

Egzistuoja visas kompleksas priežasčių, leidžiančių atsirasti klaidoms analizės ar projektavimo procese. Vienas pagrindinių etapų, sudarančių prielaidas klaidoms susiformuoti, yra reikalavimų kuriamai sistemai išgavimas bei jo atlikimo technologija. Vartotojo reikalavimų supratimas įgyjamas reikalavimų inžinerijos procese. Šiuo metu egzistuoja keletas metodų reikalavimams specifikuoti bei perkelti į projekto specifikaciją. Tai *Oracle* CASE metodas [1], R*ational Unified procesas* [4] ir kiti. Tačiau juos taikant, reikalavimų informacijos sistemai specifikacijos sudarymo eiga nėra natūrali ir atitinkanti faktišką reikalavimų įgijimo eiliškumą.

Siekiant geriau suprasti vartotojo reikalavimus, gali būti sudaromas organizacijos veiklos modelis. Šio modelio pagrindu yra kuriamas kompiuterizuotos IS modelis. Daugelyje metodų mažai dėmesio skiriama organizacijos veiklos modeliavimui (*Enterprise modeling*). Organizacijos veiklos modelis kritiškai svarbus reikalavimų inžinerijos procesams sėkmingai atlikti [3].

Atliekant reikalavimų specifikavimą būtina užtikrinti, kad specifikacija būtų neprieštaringa, o tuo pačiu būtų pasiekta maksimali jos kokybė. Kas gali padėti sukurti kokybišką specifikaciją? Tam tikslui reikalingas metodas, kuris turėtų atitikti šiuo reikalavimus:

- Reikalavimų informacijos sistemai sudarymo eiga turi būti natūrali, t.y. turi atitikti tradicinį šio proceso eiliškumą;
- Metodas turi sumažinti atotrūkį tarp vartotojo ir analitiko;
- Modeliavimo procesas turi užtikrinti organizacijos veiklos ar jos išskirto fragmento modelio pilnumą;
- Turi būti galimybė atlikti specifikacijos kokybės kontrolę.
- Toliau apžvelgiami funkcinių reikalavimų specifikavimo ir objektiškai orientuotas MERODE metodai, kurie dalinai atitinka aukščiau iškeltus reikalavimus.

2.Funkcinių reikalavimų specifikavimo metodas

Sumažinti atotrūkį tarp vartotojo (reikalavimų teikėjo) ir projektuotojo nėra paprastas uždavinys. Tuo tikslu turi būti sudaroma vartotojo reikalavimų specifikacija, reikalavimus atvaizduojant notacijoje, kurią be papildomo apmokymo gali suprasti ir vartotojas. Tokio tikslo buvo siekiama kuriant funkcinių reikalavimų specifikavimo metodą [2].

Toliau pateikiami metodo analizės rezultatai, atspindint jo sudėtį ir pagrindines charakteristikas. Naudojant ši metodą, remiamasi tokiomis sistemos kūrimo proceso dalimis:

I. Organizacijos kompiuterizuojamos veiklos modeliavimas.

II. Kompiuterizuotos IS modeliavimas.

III. Kompiuterizuotos IS projektavimas.

IV. Sistemos realizavimas.

Funkcinių reikalavimų specifikavimo metodas apima pirmąsias dvi sistemos kūrimo proceso dalis, tačiau numatoma, kad metodo taikymo rezultatai bus naudojami ir projektuojant bei realizuojant sistemą.

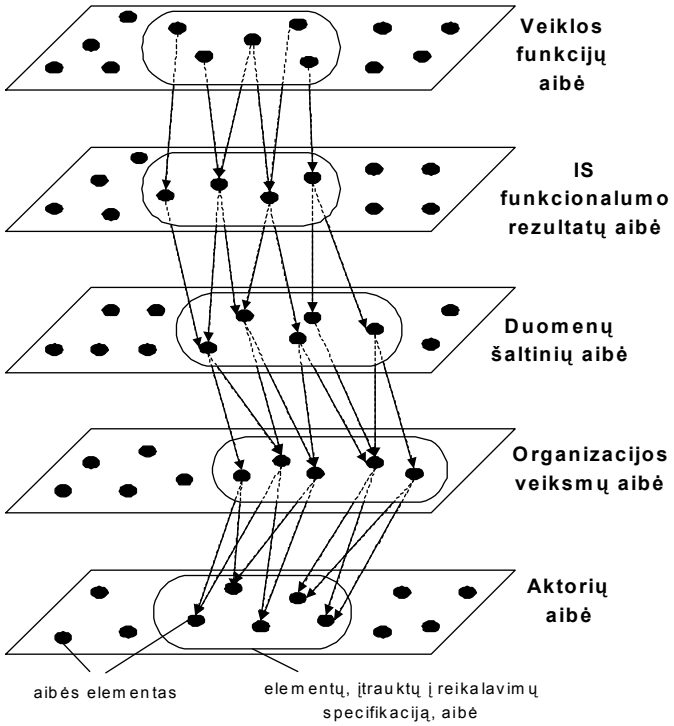
Pirmoji dalis apima organizacijos kompiuterizuotinos veiklos modeliavimą, nes pagrindiniai funkciniai reikalavimai kuriamai sistemai specifikuojami šioje fazėje. Organizacijos kompiuterizuotinos veiklos modeliavimas suskirstytas į keturis etapus [2]:

1. Išskirti kuriamos kompiuterizuotos informacijos sistemos kontekstą.
2. Specifikuoti kuriamos IS funkcionalumo rezultatus.
3. Specifikuoti kuriamos IS duomenų šaltinius.
4. Specifikuoti duomenų šaltinių apdorojimo procesus.

Modeliavimo procesas, taikant siūlomą metodą, yra iteracinis. Į kiekvieną vystymo etapą pereinama nuosekliai, tačiau iš bet kurio etapo galima grįžti į ankstesnį etapą.

Pagrindinė reikalavimų specifikavimo idėja yra tokia (žr. 1 pav.):

1. Specifikuojamos organizacijos veiklos funkcijos, apibrėžiančios kuriamos IS kontekstą.
2. Specifikuojami IS funkcionalumo rezultatai, kurie susiejami su funkcijomis, t.y., specifikuojama, kokią informaciją IS turės išvesti.
3. Kiekvienam IS funkcionalumo rezultatui specifikuojami duomenų šaltiniai (DŠ), kurie naudojami rezultatams formuoti, t. y. specifikuojama informacija, reikalinga kuriamos IS išvedamai informacijai gauti.
4. Kiekvienam DŠ formalizuotai aprašomi juos apdorojantys veiksmai.
5. Kiekvienam veiksmui specifikuojami juos atliekantys aktoriai bei aktoriai, perimantys veiksmo rezultatus.



1 pav. Funkcinių reikalavimų specifikavimo metodo konceptuali schema

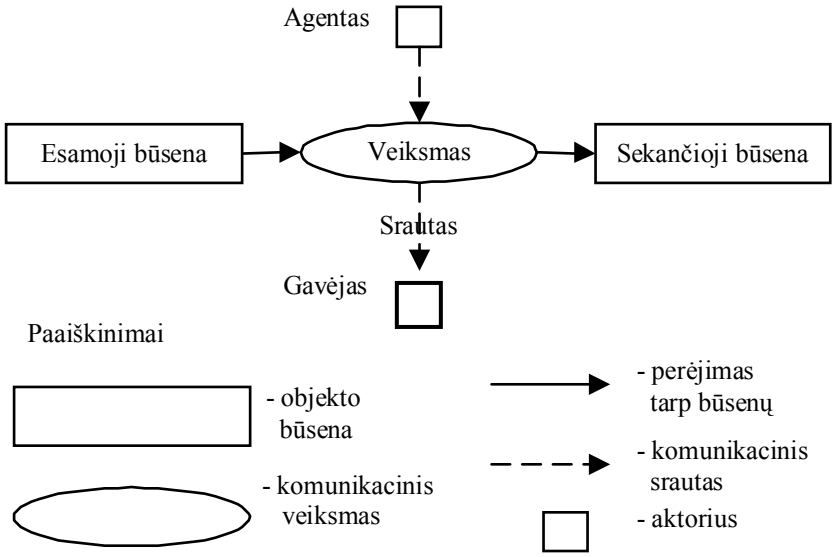
Organizacijos veiklos funkcijoms specifikuoti panaudojama Oracle CASE metodika. Informacijos sistemos apdorojamų duomenų struktūrai specifikuoti remiamasi B.Wangler metodu dalykinės srities koncepcinei schemai sudaryti iš dokumentų formų pavyzdžių [6], tačiau šiuo atveju panaudojami ne dokumentų formų užpildyti

pavyzdžiai, tačiau atliekama dokumentų formų sudėties bei pildymo procesų analizė. Pateiktas modelis IS naudojamų duomenų srautų struktūrai specifikuoti. Šis modelis sudarytas iš dviejų lygių:

- Į vartotoją orientuotas lygis;
- Į analitiką orientuotas lygis;

Į vartotoją orientuotas lygis yra grafinis duomenų šaltinio ar rezultatų projektas, kuriame kiekvienas grafinis elementas vaizduoja tam tikrą specifikacijos konceptą. Į analitiką orientuotas lygis sudaromas atlikus keletą į vartotoją orientuoto lygio specifikacijos analizės iteracijų. Šis suskirstymas leidžia neformaliai specifikuotus reikalavimus informacijos struktūrai susieti su formalizuotais struktūros elementais, kurie yra kaip pagrindas duomenų struktūros elementų specifikacijai parengti. Pagrindinis šios dviejų lygių struktūros bruožas – atotrūkio tarp vartotojo ir analitiko sumažinimas.

Duomenų šaltinių apdorojimo etapams, ryšiams bei duomenų šaltinių būsenų kaitai specifikuoti panaudota R.Gusto pasiūlyta komunikacinių veiksmų ir perėjimų tarp objektų būsenų grafinė notacija (žr. 2 pav.) bei modeliavimo principai [3].



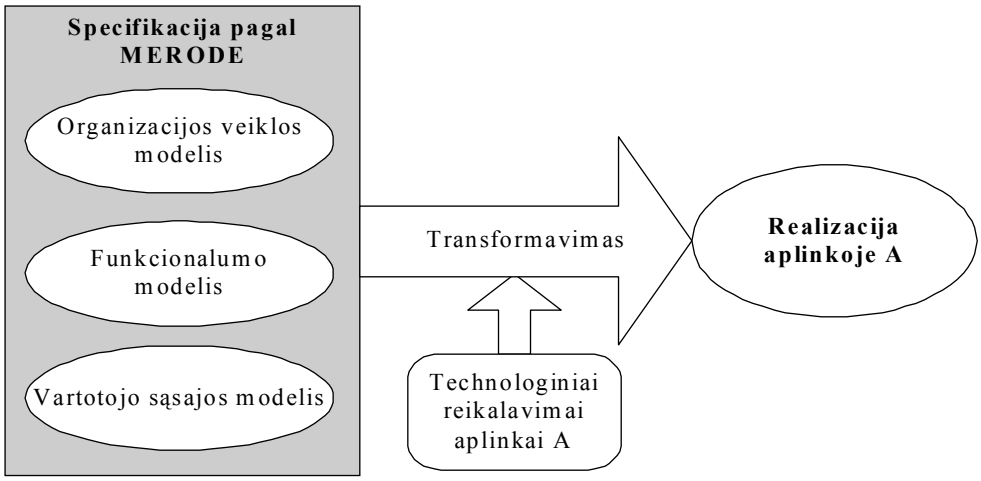
2 pav. Komunikacinių veiksmų ir būsenų kaitos priklausomybių grafinė notacija

Metode naudojamas formalus modelių aprašymas, leidžiantis patikrinti specifikacijos neprieštaringumą ir korektiškumą. Ši savybė leidžia patikrinti ne tik ankščiau minėtus kriterijus, bet ir sudarytos specifikacijos pakankamumą, toliau pereinant prie sekančio sistemos vystymo etapo. Šiam mechanizmui atlikti pritaikyti R.Gusto semantinių diagramų kokybės analizės ir įvertinimo principai [3].

Bendrai vertinant funkcinių reikalavimų metodą, jį galima įvardinti kaip metodą, palengvinantį analitiko darbą tiek išgaunant reikalavimus iš vartotojo, tiek atliekant šių reikalavimų specifikavimą. Nuosekli ir natūralizuota metodo etapų atlikimo technologija padidina sudaromos specifikacijos kokybę, tačiau sulėtina visos analizės tempą.

3. MERODE – objektiškai orientuotas metodas, pagrįstas egzistavimo priklausomybių sąryšiu

MERODE (*Model–based Existence–dependency Relationship Object–oriented Development*) – tai objektiškai orientuotas metodas, pagrįstas egzistavimo priklausomybės sąryšiu. Šio metodo pagrindas yra egzistavimo priklausomybės koncepcija. Metodo kūrėjai siekė, kad jį naudojant būtų užtikrintas modeliavimo išbaigtumas, lankstumas bei paprastumas. Be to buvo siekiama, kad modeliavimo metu būtų galima naudoti ir formalų modelių aprašymą, toliau leidžiantį atlikti gaunamų rezultatų kontrolę. Kita šiam metodui būdinga savybė – aiškiai matomas skirtumas tarp reikalavimų specifikavimo ir specifikacijos realizavimo. Sistema, specifikuota taikant šį metodą, gali būti realizuota įvairiais būdais, taikant bet kurią galimą sistemos kūrimo aplinką. Tai gali būti objektiškai orientuota arba tradicinė kūrimo aplinka (žr. 3 pav.). Pabrėžtina, kad realizavimas atliekamas transformuojant, o ne nuosekliai plėtojant specifikaciją. MERODE metodas skiriasi nuo kitų objektiškai orientuotų metodų tuo, kad garantuoja pakankamai didelį nepriklausomumo laipsnį tarp organizacijos veiklos modelio ir funkcionalumo modelio, o taip pat tarp komponentų, esančių šių modelių viduje. Šios savybės pagerina kuriamos sistemos lankstumą ir išplečiamumą [5].



3 pav. Bendra MERODE metodo sandaros ir panaudojimo schema

Sistemos kūrimas su MERODE turi natūraliai susidarančią lygių struktūrą, grupuojant reikalavimų specifikacijas pagal tai, kas yra jų atsiradimo šaltinis. Pirmoji grupė kyla iš esminių veiklos reikalavimų, įskaitant veiklos objektus, veiklos įvykius, o taip pat apribojimus veiklai. Šis reikalavimų tipas yra galiojantis, net jeigu ir nėra kompiuterizuotos informacinės sistemos. Kita reikalavimų grupė yra orientuota į IS aprašymą, kadangi šios grupės reikalavimai tiesiogiai kyla iš apribojimų IS komponentėms: duomenų įvedimo priemonėms, ataskaitų formoms ir kitoms IS realizacijos detalėms.

Pirmojo tipo specifikacija atitinka organizacijos veiklos modelį, kurį sudaro pagrindinė informacija apie organizacijoje vykstančią veiklą. Sudarius veiklos modelį, yra sukuriamas funkcionalumo modelis, atitinkantis aibę paslaugų, apdorojančių duomenų šaltinius ir pateikiančių rezultatus, tenkinant vartotojo pageidaujamą funkcionalumą informacinei sistemai. Daugelis programinės įrangos kūrimo metodų neatskiria organizacijos veiklos modeliavimo, nuo informacinio funkcionalumo modeliavimo. Dažniausiai vartotojas, formuluodamas reikalavimus sistemai, pateikia informacijos apie organizacijos veiklą ir funkcionalumo reikalavimų mišinį. Daugeliu atvejų organizacijos veikla yra stabilesnė ir kinta nesinchroniškai su reikalavimų IS funkcionalumui kaita. Todėl šį kaitos skirtumą reikia įvertinti, kad būtų lengviau prižiūrėti eksploatuojamą sistemą ir valdyti visus galimus pasikeitimus. Todėl patartina, nagrinėjant dalykinę sritį, objektus suskirstyti pagal jų kaitos pobūdį, o tai ir yra pagrindinė priežastis organizacijos veiklos modeliui atskirti nuo funkcionalumo modelio. Būtent šios idėjos ir yra laikomasi MERODE metode [5].

MERODE metodą sudaro trys fazės:

- Organizacijos veiklos modeliavimas;
- Sistemos funkcionalumo modeliavimas;
- Vartotojo sąsajos modeliavimas.

Organizacijos veiklos ir sistemos funkcionalumo etapai turi tokią struktūrą:

1.Veiklos modeliavimas

1.1. Modeliuojami veiklos objektų tipai ir veiklos įvykių tipai.

a) Veiklos objektų tipų identifikavimas

b) Veiklos įvykių tipų identifikavimas

c) Egzistavimo priklausomybės grafo sudarymas

d) Objektų–įvykių lentelės sudarymas

1.2. Įvykių sekų specifikavimas

1.3. Būsenų vektorių ir objektų įvykių metodų specifikavimas

a) Būsenos vektorių apibrėžimas

b) Objektų–įvykių metodų apibrėžimas

c) Apribojimų duomenims apibrėžimas

2. Funkcionalumo modeliavimas

2.1. Visų reikiamų paslaugų identifikavimas

2.2. Kiekvienai paslaugai reikia:

a) Identifikuoti informacinio objekto tipą

b) Duomenų įvedimo paslaugai identifikuoti sugeneruotus veiklos įvykius

c) Sudaryti paslaugų specifikacijos diagramą

d) Apibrėžti būsenos vektorių

e) Apibrėžti metodą

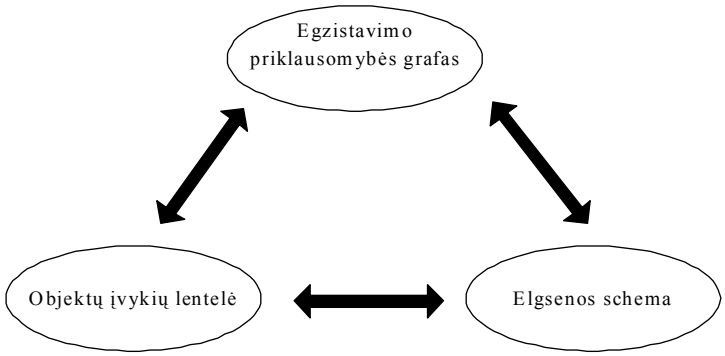
Vartotojo sąsajos modeliavimas nepateikiamas, nes analizuojant dėmesys sutelkiamas detalesniam organizacijos veiklos modeliavimui.

MERODE metodo pagalba modeliuojant organizacijos veiklą yra sudaromos trys schemos:

- Egzistavimo priklausomybės grafas;
- Objektų–įvykių lentelė;
- Elgsenos schema.

Kiekviena iš schemų aprašoma formaliai. Turint kiekvienos iš jų formalų aprašymą, galima atlikti organizacijos modelio neprieštaringumo patikrinimą, kitaip tariant, kokybės kontrolę (žr. 4 pav.). Organizacijos veiklos modeliavimo kokybė turi didžiausią įtaką realizuojamos sistemos kokybei.

MERODE metode specifikacijos kokybė apibrėžiama, kaip vidinis modelio neprieštaringumas ir korektiškumas. Egzistavimo priklausomybės grafas, kuris yra metodo pagrindinė konstrukcija, naudojamas kaip pirminis objektas atliekant organizacijos modelio kokybės kontrolę. Metode nurodoma, kaip užtikrinti objektų įvykių lentelės ir elgsenos schemos neprieštaringumą egzistavimo priklausomybės grafui. Papildomai atliekamas ir elgsenos schemos bei objektų įvykių lentelės tarpusavio neprieštaringumo patikrinimas. Tai leidžia gerokai padidinti organizacijos modelio kokybę, nei tuo atveju, kai apsiribojama schemų sintaksės korektiškumo tikrinimu.



4 pav. Trijų organizacijos modelio schemų tarpusavio neprieštaringumo patikrinimas

MERODE metodo specifikacijoje apibrėžiami reikalavimai modeliuose naudojamų sąvokų sintaksei bei semantikai, kad būtų galima verifikuoti modelių neprieštaringumą ir korektiškumą:

1. **Sintaksė:** metodologijos sintaksė turi būti tiksliai apibrėžta.
2. **Semantika:** kiekvieną metodologijos sąvoką turi lydėti formalus aprašas; sąvokos reikšmė negali būti aprašyta natūralia kalba.
3. **Neprieštaringumas tarp schemų:** turi būti apibrėžti formalūs procesai schemų tarpusavio neprieštaringumui patikrinti.
4. **Visos sistemos elgsena:** turi būti priemonės numatyti sistemos elgseną, integruojant atskirų objektų apibrėžtą sąveiką ir atliekamus veiksmus.
5. **Anomali sistemos elgsena:** jei visos sistemos veikla yra specifikuota, turi būti galimybė patikrinti sistemos elgseną, esant tam tikroms kritinėms situacijoms.

Pagrindinis MERODE pranašumas yra organizacijos veiklos modeliavimo metu atliekama kokybės kontrolė.

4.Išvados

Atlikta dviejų savitų dalykinės srities modeliavimo metodų analizė, įvertinant juos reikalavimų specifikavimo aspektu. Tiek objektiškai orientuotas MERODE metodas, tiek struktūrinis funkcinių reikalavimų specifikavimo metodas pradiniame reikalavimų specifikavimo etape orientuojasi į organizacijos veiklos modeliavimą ir jos specifikavimą, su tikslu gauti reikalavimų specifikaciją informacinei sistemai.

Funkcinių reikalavimų specifikavimo metodo išskirtinis bruožas yra prie natūralaus bei iteracinio dalykinės srities analizės proceso priartinta reikalavimų specifikavimo eiga. Vienas iš tai lėmusių faktorių yra pirmiau

sudaroma reikalavimų specifikacija išvedamai informacijai, jos pagrindu formuojama reikalingų duomenų šaltinių specifikacija bei šių specifikacijų pagrindu formuojamas duomenų modelis. Metodo naudojamas dualus modelis sumažina atotrūkį tarp dalykinės srities eksperto ir analitiko.

Objektinės orientacijos MERODE metodo išskirtinė savybė yra modelio kokybės kontrolė organizacijos veiklos modeliavimo proceso metu.

Toliau numatoma MERODE metodo kokybės kontrolės technologiją integruoti su funkcinių reikalavimų specifikavimo metodu, siekiant užtikrinti šio metodo pagrindu sudaromo duomenų modelio korektiškumą

Literatūros sąrašas

[1] **R.Barker, C.Longman**. *CASE*METHOD: Function and Process Modelling*. Addison – Wesley Publ. Co., New York, 1992.

[2] **R.Butkienė, R.Butleris**. The Approach for the User Requirements Specification. 5th East–European conference ADBIS'2001, Research Communictions, Ed. by A.Čaplinskas, J.Eder, Vilnius, 2001, p.225–240.

[3] **R.Gustas**. *Semantic and Pragmatic Dependencies of Information Systems*. Monograph, Technologija, Kaunas, 1997.

[4] **Rational Software Corporation**, Rational Unified Process 2000, žinių bazė (komercinis produktas), [žiūrėta 2001 12 15]. Prieiga per Internetą*: <http://www.rational.com/products/rup/index.jsp>*.

[5] **M.Snoeck, G.Dedene, M.Verhelst, A.M.Depuydt**. Object–Oriented Enterprise Modelling with MERODE. Leuven University Press, Belgium, 1999.

[6] **B.Wangler**. *Contributions to Functional Requirements Modelling*. Doctoral Thesis. Stockholm University, Royal Institute of Technology, DSV. Akademitryck AB, Edsbruk, 1993, pp. 189–230.

Summary

Functional requirements specification method in earliest stages of development of computerized information system is presented. Abilities for enterprise modeling with purpose of requirements elicitation and specification for information system development are described. Object oriented method – MERODE like alternative for aforementioned method is reviewed. Accomplishments of MERODE method are highlighted. It should be considered when requirements for IS are elicited and specified using functional requirements specification method.

REIKALAVIMŲ KIS ĮVEDAMAI IR IŠVEDAMAI INFORMACIJAI SPECIFIKAVIMAS

Rita Butkienė, Vaidas Savickas

Informacijos sistemų katedra, KTU
Studentų 50, LT-3031 Kaunas

Straipsnyje pateiktas vienas iš modelių vartotojo keliamiems reikalavimams kompiuterizuotai informacijos sistemai specifikuoti. Modelis apibrėžia vartotojo reikalavimus kuriamos sistemos įvestinos ir išvedamos informacijos struktūrai. Reikalavimų specifikacijos sudarymas yra pagrįstas vartotojo pateiktų dokumentų formų analize. Straipsnyje pateikta reikalavimų specifikacijos meta-modelio struktūra, aprašytas specifikavimo procesas bei CASE įrankis specifikacijai sudaryti ir patikrinti.

1. Įvadas

Vartotojo reikalavimų supratimas įgyjamas atliekant reikalavimų inžineriją. Šiuo metu egzistuoja keletas metodų reikalavimų inžinerijai atlikti. Iš labiausiai išplėtotų ir skirtingoms metodologijoms bei tradicijoms atstovaujančių metodų galima išskirti *Oracle* CASE [1, 2, 3], *Rational Unified Process* [10], *Extreme Programming* [8] *ir* F^3 [9]. Tačiau taikant juos, informacijos sistemai (IS) keliamų reikalavimų specifikacijos sudarymo eiga yra nenatūrali. Pavyzdžiui, duomenų modelio elementai į specifikaciją įtraukiami anksčiau, nei išvedamos informacijos vieneto (pavyzdžiui, ataskaitos) elementai, kurie motyvuoja duomenų modelio elementų atsiradimą specifikacijoje. Dėl šios priežasties nėra formalizuotos galimybės patikrinti, ar dalykinės srities duomenų modelis yra pakankamas. Taikant tradicines metodologijas, reikalavimų specifikavimo eiga, yra paranki IS projektuoti, t.y., vėlesnei IS kūrimo fazei, ir visai nepritaikyta reikalavimams išgauti, analizuoti, t.y., ankstyvajai IS kūrimo fazei.

R.Butkienės ir R.Butlerio [4] siūlomas reikalavimų specifikavimo procesas yra artimas natūraliai reikalavimų analizės eigai. Šis procesas prasideda nuo kuriamos kompiuterizuotos informacijos sistemos (KIS) konteksto apibrėžimo ir reikalavimų įvedamos ir išvedamos informacijos specifikavimo.

Išvedamoji informacija yra pagrindinis dalykas, kurio tikisi KIS vartotojas. Todėl KIS išvedamosios informacijos tipas bei struktūra yra pagrindinis vartotojo funkcinis reikalavimas, nuo kurio priklauso kiti kuriamai sistemai keliami funkciniai reikalavimai. Pateikdamas reikalavimus KIS išvedamai informacijai, vartotojas juos pateikia neformaliai, operuodamas tokiomis sąvokomis kaip ataskaita, suvestinė, sąrašas. Analitikas šiuos reikalavimus formalizuoja, paversdamas juos esybėmis, klasėmis, konceptais, asociacijomis. Šis formalizavimas turi daug privalumų, tačiau dėl jo tarp vartotojo ir analitiko atsiranda atotrūkis, apsunkinantis jų bendravimą.

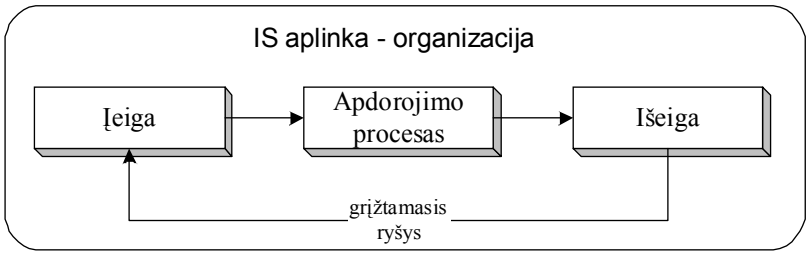
Taikant tradicinius sistemų kūrimo metodus, tokius kaip *Rational Unified Process* (RUP) [7], vartotojo neformaliai pateikti reikalavimai užrašomi natūralia kalba ir (arba) specifikuojami formalizuotai. Tačiau vartotojo reikalavimai KIS išvedamai informacijai pasižymi tuo, kad būdami neformalūs jie yra dalinai struktūrizuoti. O perėjimas nuo dalinai struktūrizuotos specifikacijos prie formalizuotos yra paprastesnis, nei perėjimas nuo natūralios kalbos, nes atotrūkis tarp vartotojo ir analitiko yra mažesnis. Egzistuojantys automatizuoti dokumentų formų analizės metodai [5, 6, 7] parodė kaip dalinai struktūrizuota informacija dokumentų formų pavyzdžių pavidale gali būti panaudota kuriamos sistemos koncepcinei schemai išgauti. Tačiau šie metodai nėra plačiai naudojami, nes paruošiamieji darbai atima daug laiko, o visiška garantija, jog gauta specifikacija yra teisinga, negaunama. Tradiciniai sistemų kūrimo metodai siūlo analizuoti dokumentų formas reikalavimams KIS statikai išgauti, tačiau formalizuotų priemonių šiam tikslui jie neturi.

Šiame straipsnyje pateiktas modelis, leidžiantis neformaliai specifikuotus reikalavimus informacijos struktūrai susieti su formalizuotais struktūros elementais. Tokiu būdu pagrindžiamas duomenų struktūros elementų atsiradimas specifikacijoje. Be to, sumažinamas supratimo atotrūkis tarp vartotojo ir analitiko. Tradiciniuose metoduose duomenų struktūrą atitinkantys esybių-ryšių (naudojamų Oracle CASE metoduose) ar klasių (naudojamuose RUP) modeliai yra pirminiai, sudaromi neprisirišant prie konkrečios metodikos, išanalizavus gautą informaciją apie dalykinė sritį.

2. Reikalavimų informacijos struktūrai specifikavimas

Informacijos sistema – tai sistema, kuri surenka, apdoroja, saugo ir platina informaciją, padedančią priimti sprendimus, koordinuoti ir kontroliuoti organizacijos veiklą, analizuoti problemas, vizualizuoti sudėtingus objektus,

kurti naujus produktus. Informacijos sistemoje (žr. 1pav.) galima išskirti dvi posistemes: informacijos įvedimo ir išvedimo. Įvedimo posistemė suteikia vartotojui galimybę įvesti informaciją (t.y. sistemos įeiga), kurią apdoroja ir pateikia vartotojui išvedimo posistemė kaip išeigą. Įvedamos informacijos struktūra priklauso nuo to, kokią informacijos struktūrą vartotojas pageidauja gauti sistemos išeigoje. IS įeigos objektus vadinsime duomenų šaltiniais (DŠ), o išeigos objektus – KIS funkcionalumo rezultatais (R). Duomenų šaltinius transformuoja apdorojimo veiksmai.



1 pav. Bendras informacijos sistemos modelis

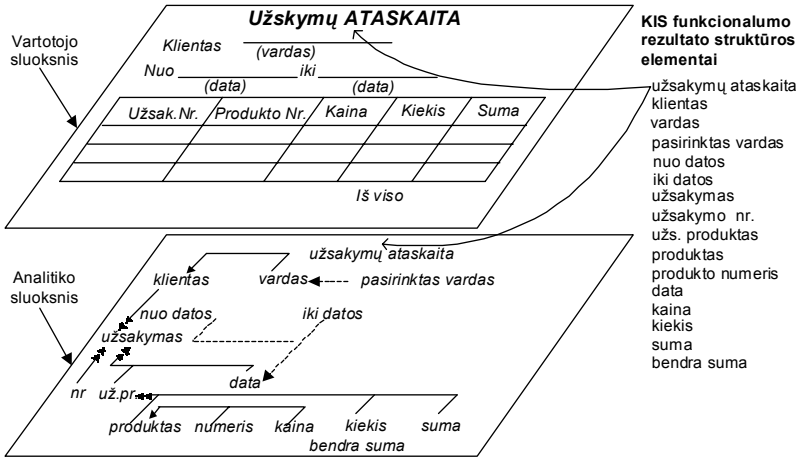
KIS funkcionalumo rezultatai – tai įvairiausios ataskaitos, rodikliai, kiti duomenų srautai, kuriuos vartotojui turi išvesti kuriamoji KIS. Vartotojui juos apibrėžti yra nesunku, nes jis žino (o jei nežino, tai privalo išsiaiškinti), kokios informacijos jam reikia, norint vykdyti kasdieninę organizacijos veiklą ir priimti sprendimus šiai veiklai koordinuoti ir kontroliuoti.

Duomenų šaltiniai – tai organizacijos objektai, saugantys informaciją apie organizacijos veiklą. Duomenų šaltiniais gali būti įvairūs organizacijos dokumentai, žodiniai pranešimai, egzistuojančių kompiuterizuotų sistemų ekraninės formos, elektroniniu būdu perduodami duomenų paketai ir pan. Duomenų šaltiniai atspindi organizacijos veiklos transakcijas. Tuo tarpu, KIS funkcionalumo rezultatai apibendrina, pateikia tam tikru aspektu duomenų šaltiniuose esamą informaciją.

Pagrindinis dalykas, kurio tikisi vartotojo iš kuriamos KIS yra tai, kad sukurtoji sisitema jam pateiks reikiamą informaciją. Galimybė įvesti duomenis į sistemą vartotojui taip pat yra svarbi (tačiau tai nėra pagrindinis vartotojo pageidavimas), nes be jos sistema nepateiks reikiamos informacijos. Kokia informacija turės būti įvedama į sistemą priklauso nuo to, kokia informaciją bus pageidaujama iš jos gauti. Taigi, visai natūralu pradėti aiškintis užsakovo reikalavimus pradedant nuo KIS funkcionalumo rezultatų, kurie ir apibrėžia duomenų šaltinius reikalingus jiems suformuoti.

Norint specifikuoti reikalavimus KIS įvedamai ir išvedamai informacijai, pirmiausiai reikia specifikuoti įvedamos ir išvedamos informacijos struktūrą. Šiam tikslui ir pasiūlytas specialus modelis.

2.1. Informacijos struktūros specifikavimo modelis



2 pav. Reikalavimų informacijos struktūrai specifikacijos modelis

KIS funkcionalumo rezultatų ir duomenų šaltinių (R/DŠ) struktūrai specifikuoti naudojamas dviejų sluoksnių modelis (2 pav.). Pirmasis sluoksnis yra skirtas vartotojui ir yra vadinamas formos šablonu. Antrasis sluoksnis (2 pav. apačioje) yra skirtas reikalavimų inžinieriui (analitikui) ir vadinamas R/DŠ struktūros modeliu. Pavyzdžiui, jei specifikuojamas KIS funkcionalumo rezultatas, tai vartotojo sluoksnyje įvedama tokia jo forma, kokią ją tikisi gauti

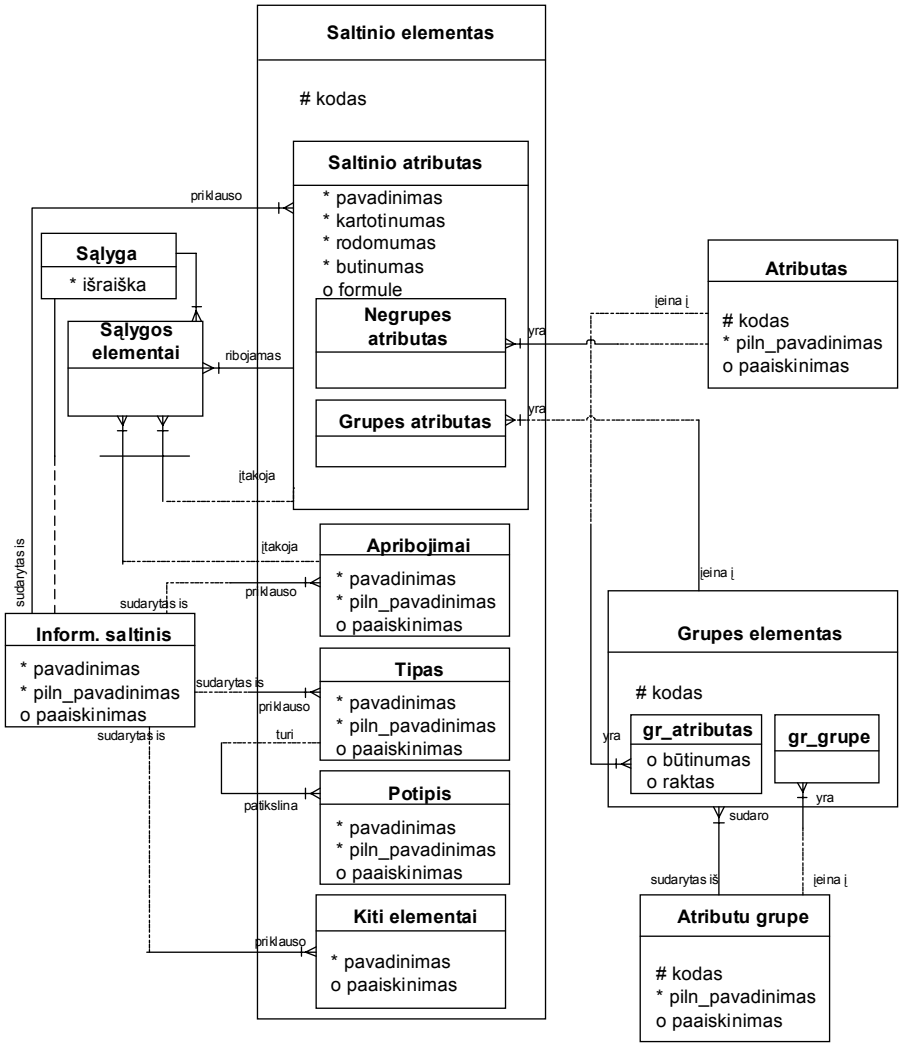
KIS vartotojas. Tarkim KIS funkcionalumo rezultatas yra tam tikra ataskaita. Tai šiame sluoksnyje tiesiog įvedamas ataskaitos pavadinimas, nurodomi ataskaitos rekvizitai (pavyzdžiui, stulpelių pavadinimai). Visi elementai išdėstomi taip, kaip juos turės išvesti kuriamoji KIS. Jei specifikuojamas duomenų šaltinis, kuris, pavyzdžiui, yra tam tikra dokumento forma, tai vartotojo sluoksnyje tiesiog įvedama šio dokumento kopija. Jei duomenų šaltinis yra žodinis pranešimas, tai šablone gali būti nurodomi jo turinį apbūdinantis rekvizitai.

Antrasis sluoksnis atspindi analitiko požiūrį į vatotojo sluoksnio specifikaciją. Remdamasis savo žiniomis bei vartotojo pagalba, analitikas interpretuoja pirmajame sluoksnyje pateiktą informaciją ir sudaro R/DŠ struktūros modelį. Kiekvienas vartotojo sluoksnio elementas yra tiesiogiai susiejamas su atitinkamu struktūros elementu analitiko sluoksnyje. Struktūros elementus į antrąjį sluoksnį galima įtraukti ir nepanaudojant formos šablono. Tokiu atveju, visi R/DŠ struktūros elementai bus specifikuojami kaip neturintys interpretacijos vartotojo lygyje. Tokios galimybės būtinybė atsiranda, pavyzdžiui, dėl paslėptos semantikos formos šablone.

Dviejų sluoksnių modelis turi dvejopą tikslą:

- Vartotojo sluoksnis leidžia lengviau susišnekėti vartotojui ir analitikui, nes vartotojas mato tai, ką jis ir tikisi gauti iš kuriamos KIS (pavyzdžiui, kada kalbama apie KIS funkcionalumo rezultatus). Be to naudojamos dokumentų formos vartotojui yra gerai pažystamos, todėl jam lengviau perteikti savo žinias analitikui. Projektuotojui belieka rasti būdą kaip teisingai perteikti savo sudaryto modelio semantiką vartotojui. Tuo būdu vyksta abipusis apsimokymas: inžinierius gilinasi į dalykinę sritį, o vartotojas į formalų informacijos aprašymą.
- Toks modelio suskirstymas į sluoksnius, leidžia specifikuoti kiekvieno struktūros elemento kilmę, t.y., kiekvienas struktūros elementas projektuotojo sluoksnyje gali būti motyvuotas atitinkamu elementu iš vartotojo sluoksnio.

Meta-modelio koncepcinė schema, kuriamos KIS funkcionalumo rezultatų ir duomenų šaltinių struktūrai specifikuoti analitiko sluoksnyje esybių-ryšių modelio notacijoje pateikta 3 pav.



3 pav. Informacijos struktūros meta-modelio koncepcinė schema

Informacijos šaltinis – tai esybė, atitinkanti tamt tikrą duomenų šaltinį arbą KIS funkcionalumo rezultatą.

Atributas – tai kiekybinė arba kokybinė tam tikro organizacijos objekto, apie kurį turi būti saugoma informacija savybė. Taip pat gali būti savybė, kuri klasifikuoja arba identifikuoja šiuos organizacijos objektus. Atributas įtraukiamas į KIS specifikaciją, jei jis yra įtrauktas bent į vieną rezultatą ar duomenų šaltinį (t.y. aprašo jo struktūrą).

Atributų grupė – tai atributai ar jų grupės, aprašančios tą patį objektą. Skirtingiems objektams gali būti būdingos tos pačios savybės. Taigi, tas pats atributas ar jų grupė gali būti įtrauktas į kelias skirtingas atributų grupes. Atributų grupė gali būti specifikuota, jei specifikuojami ją sudarantys atributai ar atributų grupės.

Šaltinio elementas – tai informacijos šaltionio (duomenų šaltinio ar KIS funkcionalumo rezultato) struktūrą aprašantys elementai.

Šaltinio atributas – tai tam tikro rezultato ar duomenų šaltinio struktūros elementas, kurio reikšmės bus įvedamos, saugomos, apdorojamos, ar išvedamos kuriamojoje KIS. Vieni atributai aprašo tik rezultatą ar duomenų šaltinį (kurie taip pat yra organizacijos objektai). Šie atributai bus vadinami neįtrauktais į atributų grupę. Kiti atributai aprašo kitų (ne duomenų šaltinių ar rezultatų) objektų savybes, kurios yra pateikiamos tam tikrose duomenų šaltiniuose ar rezultatuose. Šie atributai bus vadinami įtrauktais į atributų grupes.

Tipas – tai šaltinio struktūros elementas, identifikuojantis šaltinio tipą. Šaltinio tipas identifikuoja tik vieną rezultatą. Kiekvienam šaltiniui gali būti specifikuotas tik vienas tipas. Šaltinio tipu gali būti jo pavadinimas, pavyzdžiui, „*Užsakymų suvestinė*“ .

Potipis – tai šaltinio struktūros elementas, identifikuojantis šaltinio tipo potipį. Pavyzdžiui, tipo „*Užsakymų suvestinė*“ potipiu gali būti „*metinė*“. Šaltinio tipo potipis identifikuoja tik vieną šaltinį. Kiekvienam šaltiniui gali būti specifikuoti keli tipo potipiai.

Ribojantis elementas – tai šaltinio atributų reikšmes ribojantis elementas, kuris yra tam tikra atributo reikšmė. Pavyzdžiui, „*skubus*“ yra atributo „*Užsakymo tipas*“ reikšmė. Kiekvienam šaltiniui gali būti specifikuota keletas ribojančių elementų.

Kiti elementai (neklasifikuoti elementai) - tai šaltinio struktūros elementas, kuris nėra atributas, nei tipas, nei potipis, nei ribojantis elementas. Kiekvienam šaltiniui gali būti specifikuota keletas tokių elementų. Tokiais elementais gali būti kiekvienas šaltinio struktūros elementas, jei jis neperklasifikuojamas į bent vieną iš ankščiau aprašytų struktūros elementų tipų.

Sąlyga – tai sąlyga, kurios reikia laikytis formuojant tam tikrą rezultatą ar duomenų šaltinį. Dažnai, formuojant tam tikrą rezultatą, uždedami apribojimai rezultatų atributo reikšmėms (pavyzdžiui, įvairūs laiko intervalai). Kas ir kaip įtakoja išvedamų rezultato atributų reikšmes, nurodoma sąlygos išraiškoje. Jei šaltiniui yra specifikuotas bent vienas ribojantis elementas, vadinasi šaltiniui turi būti specifikuota sąlyga. Pavyzdžiui, skubiam užsakymui suformuoti, turi būti patenkinta tokia sąlyga „*Užsakymo tipas = ‚skubus‘* “. Koks šaltinio elementas ir kokį kitą elementą įtakoja, yra nurodyta specifikuojant sąlygos elementus.

Sąlygos elementas – tai elementas, parodantis koks šaltinio elementas (atributas ar ribojantis elementas) įtakoja kitą šaltinio elementą (atributą). Sąlygą turi sudaryti bent vienas elementas. Pavyzdžiui, pora „*Užsakymo tipas*“ ir „*skubus*“ yra salygos elementas.

2.2. Informacijos struktūros specifikavimo procesas

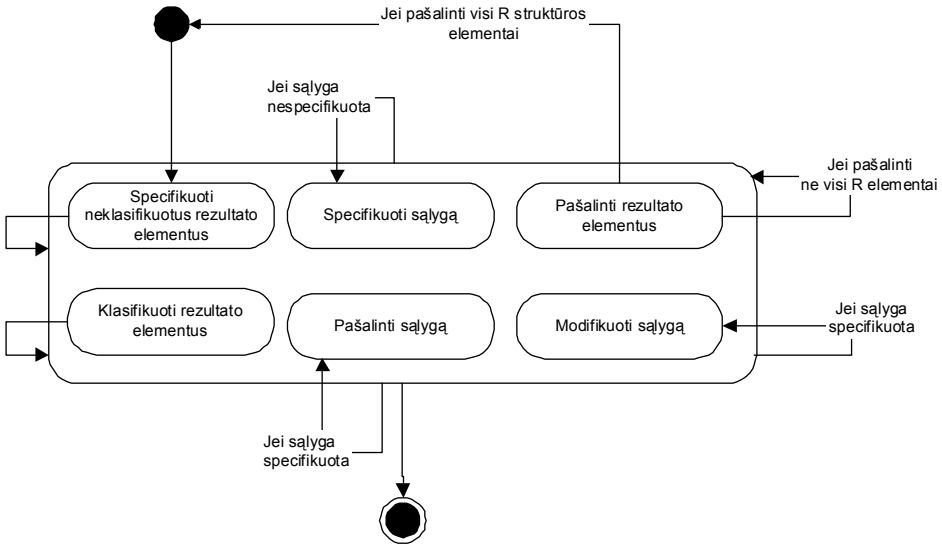
KIS funkcionalumo rezultatų ir duomenų šaltinių specifikavimo procesai yra panašūs, todėl aptarsime tik vieną – rezultatų specifikavimą. Rezultato struktūros specifikavimo procesas yra pateiktas 4 pav.

Vartotojo sluoksnyje specifikuotas KIS funkcionalumo rezultatą aprašantis elementas analitiko sluoksnyje specifikuojamas kaip neklasifikuotas struktūros elementas. Specifikavus bent vieną rezultato struktūrą aprašantį neklasifikuotą elementą galima atlikti vieną iš trijų veiksmų:

1. specifikuoti sekantį rezultato struktūrą aprašantį neklasifikuotą elementą;
2. klasifikuoti specifikuotą elementą, t.y. priskirti tam tikram elementų tipui: atributai, atributų grupės, apribojimai, kiti elementai, DŠ/R šaltinio tipas, DŠ/R šaltinio potipis;
3. pašalinti specifikuotą struktūroselementą.

Tam tikri rezultato struktūros elementai specifikuojami, koreguojami arba pašalinami tik tuomet jei patenkinamos tam tikros sąlygos:

- Jei specifikuojami bent du rezultato atributai arba bent vienas rezultato atributas ir elementas, ribojantis atributo reikšmę, tai galima specifikuoti sąlygą, įtakojančią rezultato formavimą.
- Jei specifikuojama rezultato suformavimą įtakojanti sąlyga, tai galima ją koreguoti, arba pašalinti.
- Jeigu pašalinamas vienintelis rezultato struktūrą aprašantis elementas, tai tuomet galima tik specifikuoti kitą, rezultato struktūrą aprašantį neklasifikuotą elementą.



4 pav. KIS funkcionalumo rezultatų struktūros specifikavimo procesas

Galutinis specifikavimo proceso rezultatas yra KIS funkcionalumo rezultato struktūra. Struktūra turi tenkinti tokias taisykles:

1. kiekvienas rezultatas turi turėti bent vieną atributą;
2. kiekvienas rezultatas turi turėti šaltinio tipą;
3. kiekviena atributų grupė turi turėti bent vieną elementą: atributų grupę arba atributą;
4. kiekvienai rezultato suformavimą sąlygojančiai sąlygai turi būti specifikuotas bent vienas jos elementas;
5. kiekviena atributo reikšmes ribojanti reikšmė turi būti įtraukta į rezultato suformavimą įtakončią sąlygą.

3. CASE įrankis

Aukščiau nagrinėtam modeliui ir procesui įgyvendinti buvo sukurtas CASE įrankis. Pagrindinis šio rankio vartotojas yra reikalavimų inžinierius. Tačiau juo gali būti ir kuriamos KIS vartotojas. Įrankis sukurtas panaudojant *Visio/2000* programų paketą, kuris leidžia sukurti pageidaujamus grafinius objektus, o trūkstamas funkcijas bei procedūras realizuoti Visual Basic for Application (VBA) pagalba [11]. Šį įrankį sudaro (žr. 5 pav.):

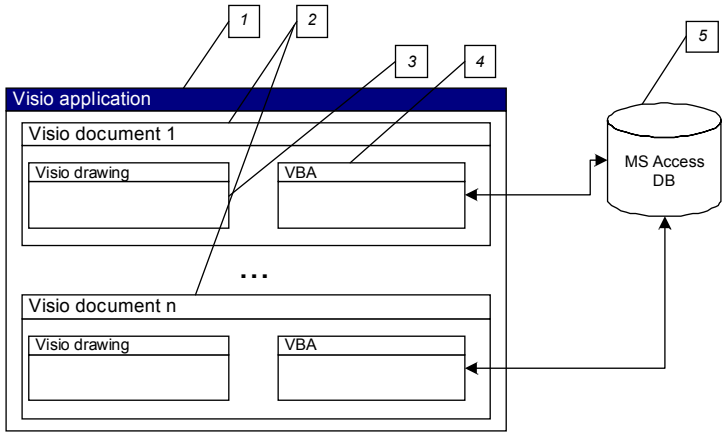
1 – *Visio/2000* aplinka, kurioje funkcionuoja sukurtas CASE įrankis,

2 – *Visio/2000* dokumentai, grafinei specifikacijai sudaryti,

3 – specifikacija grafinėje notacijoje,

4 – papildomos VBA funkcijos specifikacijai sudaryti,

5 – *MS Access* duomenų bazė.



5 pav. CASE įrankio struktūrinė schema

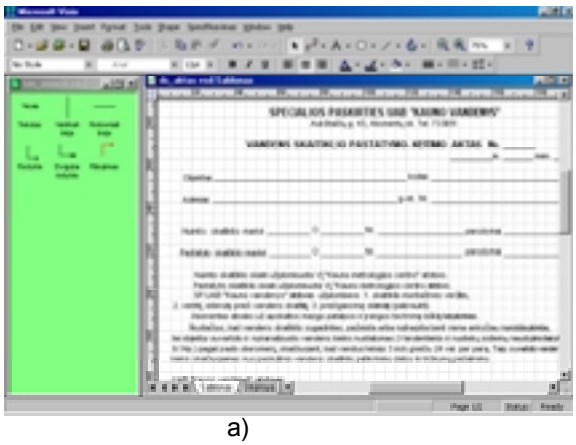
Įrankis funkcionuoja *Visio/2000* aplinkoje. Specifikacijai sudaryti ir saugoti grafinėje notacijoje naudojami šiam tikslui paruošti du *Visio/2000* failai-šablonai: vienas skirtas KIS funkcionalumo rezultatams specifikuoti, kitas – duomenų šaltiniams specifikuoti. Pagal failą-šabloną sukuriamas naujas *Visio/2000* dokumentas. Naujai sukurtas dokumentas saugo savyje ne tik specifikaciją grafinėje notacijoje, bet ir visas priemones, leidžiančias vykdyti informacijos struktūros specifikavimo procesą *Visio/2000* aplinkoje.

Reikalavimų specifikacijos saugykla yra ne vienalytė. Ją sudaro *MS Access* duomenų bazė ir *Visio/2000* dokumentai, saugantys specifikaciją grafinėje notacijoje. *MS Access* duomenų bazė yra sukurta pagal 3 pav. pateiktą meta-modelio koncepcinę schemą. Duomenų bazėje, vartotojo pageidavimu, išsaugoma informacijos struktūros specifikacija.

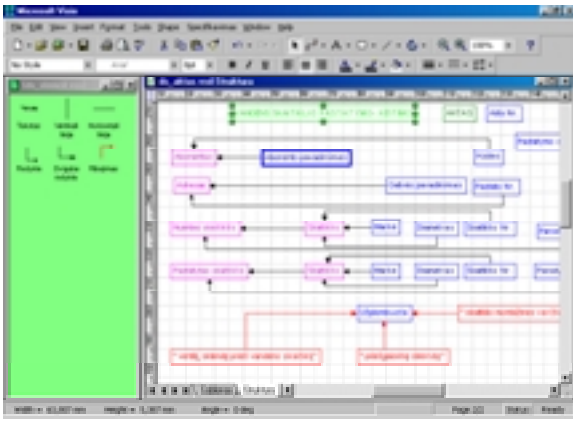
Šio CASE įrankio pagrindinės funkcijos yra:

1. įvesti KIS funkcionalumo rezultato ar duomenų šaltinio (R/DŠ) formos šabloną (grafinė notacija),
2. modeliuoti R/DŠ struktūrą (grafinė notacija),

Specifikacijai sudaryti šiame įrankyje realizuotas 2.1 skyriuje aprašytas dviejų sluoksnių modelis (2 pav.). Modeliui realizuoti panaudoti du vieno *Visio/2000* dokumento lapai (6 pav.).



a)



b)

6 pav. CASE įrankio dokumentas specifikacijai sudaryti: a) vartotojo lapas „Šablonas“; b) analitiko lapas „Struktūra“

Pirmasis lapas skirtas vartotojui (atitinka vartotojo sluoksnį). Šiame lape atvaizduojami dokumentų šablonai įprastine vartotojui forma. Modifikuoto *Visio/2000* kontekstinio meniu pagalba automatiškai visi (arba, inžinieriui pageidaujant, tik naujai sukurti) įvesti tekstiniai objektai perkeliami į antrąjį lapą – DŠ/R struktūrą. Jame analitikas, remdamasis savo patirtimi, bei bendraudamas su vartotoju, klasifikuoja elementus, jungia juos į grupes, įveda sąlygas, nurodo ribojančius ir ribojamus atributus ir, jei reikia, specifikuoja naujus struktūros elementus, neturinčius atitikmens šablone. Klasifikuoti objektai vaizduojami tam tikra spalva (jei pageidaujama ir vartotojo lape).

CASE įrankis turi dar ir tokias pagalbines funkcijas:

1. Pasirinkti duomenų bazę. Skirtingiems projektams saugoti naudojamos skirtingos duomenų bazės.
2. Patikrinti klaidas.
3. Išsaugoti specifikaciją duomenų bazėje. Norint struktūros modelius panaudoti tolimesniame sistemos kūrime, reikia grafinėje notacijoje išreikšta specifikaciją perkelti į specifikacijos saugyklą duomenų bazėje.
4. Pašalinti su pasirinktu KIS funkcionalumo rezultatu ar duomenų šaltiniu susijusią specifikaciją iš duomenų bazės.
5. Suvienodinti struktūros elementų vardus. Ši funkcija ir užtikrina į skirtingus KIS funkcionalumo rezultatus ar duomenų šaltinius įtrauktų tų pačių atributų ar atributų grupių pavadinimų vienodumą.

4. Išvados ir tolimesni darbai

Pateiktas modelis leidžia neformaliai specifikuotus reikalavimus informacijos struktūrai susieti su formalizuotais struktūros elementais. Tokiu būdu pagrindžiamas duomenų struktūros elementų atsiradimas specifikacijoje. Be to, sumažinamas supratimo atotrūkis tarp vartotojo ir analitiko.

Ateityje planuojama sukurti CASE įrankį, kuris, įvedamos informacijos struktūrai keliamų reikalavimų specifikacijos pagrindu, generuotų dalykinės srities koncepcinę (esybių-ryšių modelio tipo) schemą.

Literatūra

[1] **R.Barker (1990).** *CASE*METHOD: Entity Relationship Modelling*. Addison – Wesley Publ. Co., New York.

[2] **R.Barker (1990).** *CASE* Method: Tasks and Deliverables*. Addison-Wesley.

[3] **R.Barker, C.Longman** (1992). CASE*METHOD: Function and Process Modelling. Addison – Wesley Publ. Co., New York.

[4] **R.Butkienė, R.Butleris**. *The Approach for the User Requirements Specification*. 5 East-European conference ADBIS'2001, Research Communications, Ed. by A.Čaplinskas, J.Eder, Vilnius, p.255-240

[5] **J.Chobineh, M.Mannino**. An Expert Database Design System Based on Analysis of Forms. *IEEE transactions on software engineering*, Vol.14, 1998

[6] **M.Mannino, V.P.Tseng**. Inferring database requirements from examples in forms. *Seventh International Conference on Entity – Relationship approach*. Rome, 1988, 1-25.

[7] **B.Wangler**. Contributions of Function Requirements Modeling. *Doctoral Thesis. Stockholm University*, 1993, 189–230

[8] Extreme Programming Internet svetainė: [žiūrėta 2001 07 02]. Prieiga per Internetą: *<http://www.extremeprogramming.org/>*

[9] F^3 Consortium (1994). F3 Reference Manual, ESPRIT Project 6612, Swedish Institute for Systems Development, Electrum 212, S-16440, Kista, Sweden.

[10] Rational Software Corporation, Rational Unified Process 2000, žinių bazė (komercinis produktas), [žiūrėta 2001 08 11]. Prieiga per Internetą: *<http://www.rational.com/products/rup/index.jsp>*

[11] *Developing Visio Solutions for Microsoft® Visio 2000.* Visio 2000 paketo dokumentacija.

Summary

The model for specification of user requirements for computerized information system is discussed. This model describes the user requirements for structure of input and output information. The creation of model is based on analysis of document forms given by user. The meta-model of requirements specification is presented. The specification process and CASE tool for requirements specification are described.

INFORMACINIŲ SISTEMŲ PROJEKTAVIMO METODŲ TOBULINIMAS IR AUTOMATIZAVIMAS, TAIKANT UML

Lina Nemuraitė, Lina Kavaliauskaitė

KTU, Informacijos sistemų katedra
Studentų 50 - 308, LT-3028 Kaunas

Šiame straipsnyje nagrinėjamos galimybės patobulinti *CASE* įrankiais vykdomus informacinių sistemų projektavimo metodus bei padidinti jų automatizavimo laipsnį: taikant tiesioginę bei atvirkštinę inžineriją įvairiems *UML* diagramų tipams; pasirenkant projektavimo metodiką su minimaliu diagramų skaičiumi; generuojant vieno tipo diagramas iš kitų diagramų tipų. Pasiūlytas algoritmas generuoti būsenų diagramas iš sekų diagramų, kuris palengvintų projektuotojo darbą ir padidintų projekto modelių suderinamumą.

1. Įvadas

Šiandieninis informacinių sistemų (IS) projektavimas neįsivaizduojamas be automatizuoto projektavimo (*CASE*) įrankių, kurių daugelis naudoja unifikuotą modeliavimo kalbą *UML* [1], skirtą vizualizuoti, specifikuoti, konstruoti ir dokumentuoti objektiškai orientuotas sistemas. *UML* turi devynių tipų - klasių, objektų, panaudojimo atvejų, sekos, bendradarbiavimo, būsenų, veiklos, komponentų bei įdiegimo - diagramas. IS projektavime didelis dėmesys skiriamas modeliavimui, bet vis dėlto galutinis darbo rezultatas yra programa, o ne diagramos. *UML* modelius galima tiesiogiai atvaizduoti įvairiose programavimo kalbose (*JAVA*, *C++, Visual Basic* ir t.t), taip palengvinant projektuotojų ir programuotojų darbą.

Šiuo metu egzistuoja du kraštutiniai *UML* pagrįsti programinės įrangos kūrimo procesai: *Rational Unified Process* (*RUP*) ir *Extreme Programming* (*XP*). *RUP* yra platus ir sudėtingas, o *XP* – minimalus, orientuotas į kodo rašymą. Kuriant programas, tikslinga naudinga minimalų *UML* diagramų kiekį, kuris užtikrintų kuo trumpesnį ir paprastesnį analizės bei projektavimo procesą. Bet diagramų kiekis turi būti pakankamas, kad projektas būtų aprašytas teisingai ir pilnai. Iš IS projektavimo metodų galima išskirti *ICONIX* procesą [2-6], kuris pagrįstas minimaliu *UML* diagramų skaičiumi ir turi metodiką, kurios dėka kūrimo procesas „nuo panaudojimo atvejų iki kodo" yra greitas ir efektyvus. Kaip ir *RUP*, jis remiasi panaudojimo atvejų diagramomis, bet nėra toks sudėtingas ir platus. Pagrindinis *ICONIX* proceso rezultatas – sekos diagramos, kurios kartu su statine klasių diagrama toliau naudojamos kodo rašymui. Tačiau *ICONIX* projektavimas būtų išsamesnis ir geriau užtikrintų produkto korektiškumą, jei šį procesą papildytų būsenų diagramos, automatiškai generuojamos iš sekos diagramų.

2. Tiesioginės ir atvirkštinės inžinerijos taikymo įvairioms *UML* diagramoms galimybės

Tiesioginė inžinerija – tai procesas, kurio metu modelis transformuojamas į kodą, taip atvaizduojant jį reikiamoje programavimo kalboje. Tiesioginės inžinerijos pašalinis efektas – informacijos praradimas. Taip yra todėl, kad *UML* modeliai yra semantiškai turtingesni, negu bet kuri dabartinė objektiškai orientuota programavimo kalba. Tai yra ir viena pagrindinių priežasčių, kodėl norint suprasti programą, be pačio programos kodo reikalingi ir *UML* modeliai. Atvirkštinė inžinerija – tai procesas, kurio metu kodas transformuojamas į modelį, panaudojant kodą kaip modelio atvaizdavimą specifinėje programavimo kalboje. Atvirkštinės inžinerijos pašalinis efektas - informacijos perteklius, nes dažnai informacija yra detalesnė, negu reikia naudingam modeliui sukurti. Bet tuo pačiu metu atvirkštinės inžinerijos rezultatas yra nepilnas. Tiesioginės inžinerijos metu informacija yra prarandama, todėl negalima pilnai atkurti modelio iš kodo, nebent *UML CASE* priemonės tiesioginės inžinerijos proceso metu informaciją, kuri yra už programavimo kalbos semantikos ribų, surašo į komentarus ir vėliau panaudoja juos modelio atkūrimui.

Pakaitomis taikant tiesioginę ir atvirkštinę inžineriją, vykdomas iteracinis rekursyvus projektavimas (*Round Trip Engineering*), kuriuo suprantamas toks *CASE* įrankių pagalba atliekamas iteracinis projektavimo procesas, kuris jungia tiesioginės ir atvirkštinės inžinerijos galimybes: patobulinus kuriamos sistemos modelį, *CASE* įrankio pagalba šiuos pakeitimus galima automatiškai perkelti į programos kodą, ir atvirkščiai – pakoregavus programos kodą, pakeitimus galima automatiškai atspindėti modeliuose. Iteracinis rekursyvus projektavimas įgalina palaipsniui tobulinti sistemą kūrimo metu ir adaptuoti, prižiūrint jos funkcionavimą. Šio proceso metu neturėtų būti informacijos praradimų, deja, realiai tai ne visuomet įgyvendinama.

Kodo generavimo ir atvirkštinės inžinerijos panaudojimo galimybės priklauso ir nuo tam naudojamų *UML* diagramų tipų. Klasių diagramos yra aiškiausiai iš visų diagramų susiję su visomis objektiškai orientuotomis programavimo kalbomis (*Java, C++, Smalltalk, Eiffel, Ada, ObjectPascal, Forte, Visual Basic* ir t.t.). Klasių diagramų tiesioginė ir atvirkštinė inžinerija yra paprasčiausia, lyginant su kitomis diagramomis, ir nesunkiai realizuojama, nes generuojama ir atkuriama tik programos struktūra (klasės).

Objektų diagrama – tai diagrama, kuri parodo objektų aibę ir jų ryšius vienu laiko momentu, programos vykdymo metu. Objektų diagramos tiesioginė inžinerija yra teoriškai galima, bet praktiškai jos vertė yra nedidelė. Tuo tarpu objektų diagramos atvirkštinė inžinerija yra realiai naudinga. Ji realiai vyksta kiekvieną kartą programos derinimo metu (tai atlieka žmogus arba įrankis, derinantis programą), nes norint rasti klaidą, pastebėtą programos vykdymo metu, patogu įsivaizduoti ar nubraižyti tą momentą atitinkančią objektų diagramą, kad pamatyti to momento objektų būsenas bei ryšius ir taip rasti klaidą. Dažniausiai objektų diagramos atvirkštinei inžinerijai parenkamas siauresnis kontekstas, negu visa programa. Patogu pasirinkti vieną pagrindinę klasę ir nagrinėti tos klasės būsenas bei su ja bendradarbiaujančius objektus.

Panaudojimo atvejų diagrama nuo kitų *UML* diagramų skiriasi tuo, kad ji greičiau atspindi, o ne specifikuoja sistemos, posistemės ar klasės realizaciją. Panaudojimo atvejai aprašo, kaip elementas elgiasi, o ne kaip tas elgesys realizuojamas. Todėl panaudojimo atvejų diagrama negali būti panaudojama automatizuotoje tiesioginėje ar atvirkštinėje inžinerijoje, nes per daug informacijos prarandama realizavimo metu, kai reikia pereiti nuo specifikacijos, kaip elementas elgiasi, prie to, kaip jis realizuotas. Grįžtant nuo realizacijos prie modelio, panaudojimo atvejų diagramai atstatyti nebeužtenka informacijos. Neautomatizuotą atvirkštinę panaudojimo atvejų diagramos inžineriją galima atlikti taip: reikia išstudijuoti sistemos elgesį ir jo pagrindu sudaryti šią diagramą. Tai realiai daroma kiekvieną kartą, kai reikia išanalizuoti nedokumentuotą programinę įrangą. Panaudojimo atvejų diagrama šiuo atveju yra tiesiog standartizuota ir aiški kalba, kuria galima aprašyti išsiaiškintas konkrečios sistemos funkcines galimybes.

Sekos ir bendradarbiavimo diagramos – abi vadinamos sąveikos diagramomis – tai dvi iš penkių diagramų, kurios naudojamos dinaminių sistemos aspektų modeliavimui. Sąveikos diagrama rodo sąveiką, susidedančią iš aibės objektų ir jų tarpusavio ryšių, apimančių ir pranešimus, kurie gali būti perduodami tarp objektų. Sekos diagrama akcentuoja pranešimų išsidėstymą laike, tuo tarpu bendradarbiavimo diagrama akcentuoja objektų, kurie siunčia ir priima pranešimus, struktūrinę organizaciją. Tiesioginė ir atvirkštinė inžinerija yra galima abiem sąveikos diagramoms, ypač jeigu diagramos kontekstas yra operacija. Bet atvirkštinė inžinerija yra sudėtingesnė, nes jos metu gaunama per daug informacijos, todėl reikalinga atranka, kurias detales palikti.

Būsenų diagrama naudojama dinaminių sistemos aspektų modeliavimui. Ši diagrama vaizduoja būsenų mašiną. Būsena yra objekto gyvavimo ciklo sąlyga arba situacija, kurios metu objektas tenkina kokią nors sąlygą, vykdo kokį nors veiksmą arba laukia kokio nors įvykio. Būsenų mašina formaliai aprašo visą galimą klasės objektų ar kito modelio elemento elgesį. Būsenų mašiną sudaro įvykiai, objektų reakcijos į įvykius ir būsenų sekos, kurias įgyja objektai reakcijos į įvykius išdavoje. Tiesioginė inžinerija būsenų diagramoms galima, ypač jeigu diagramos kontekstas yra klasė. Atvirkštinė inžinerija yra teoriškai galima, bet praktiškai nenaudinga. Atvirkštinės inžinerijos metu ne visos gautos būsenos yra prasmingos, o *CASE* įrankiai neturi abstrakcijos sugebėjimo ir todėl negali automatiškai generuoti prasmingų būsenų diagramų.

Veiklos diagrama naudojama dinaminių sistemos aspektų modeliavimui. Veiklos diagramoje vaizduojamas veiksmų grafas yra būsenų mašina, kur būsenos vaizduoja veiksmų vykdymą ir perėjimus iššaukia veiksmų užbaigimas. Tiesioginė ir atvirkštinė inžinerija veiksmų diagramoms galima, ypač jeigu diagramos kontekstas yra operacija.

Komponentų diagrama modeliuoja fizinius objektiškai orientuotos sistemos aspektus. Ši diagrama rodo sistemos komponentų aibės organizaciją ir priklausomybes. Komponentas – tai dažniausiai loginių elementų (klasių, interfeisų ir bendradarbiavimo atvejų) fizinis apipavidalinimas. Tiesioginė ir atvirkštinė komponentų inžinerija yra galima ir atliekama nesudėtingai. Iš tikrųjų kiekvieną kartą, kai vykdoma klasės ar bendradarbiavimo atvejo tiesioginė inžinerija, kartu generuojamas ir komponentas - tos klasės ar bendradarbiavimo kodas, biblioteka, ar vykdomasis failas. Taip pat yra ir atvirkštinės inžinerijos atveju - vykdant programos kodo, bibliotekos ar vykdomojo failo atvirkštinę inžineriją, gaunama aibė komponentų, kurie savo ruožtu atitinka klases ir bendradarbiavimo atvejus.

Įdiegimo diagrama modeliuoja fizinius objektiškai orientuotos sistemos aspektus. Ji rodo fizinį techninių įrenginių išdėstymą bei vykdomųjų programų komponentų paskirstymą juose. Tiesioginė inžinerija realiai nėra realizuojama. Tuo tarpu atvirkštinės įdiegimo diagramos inžinerijos vertė yra didžiulė, ypač paskirstytoms sistemoms, kuriose nuolat vyksta pakeitimai. Atvirkštinės inžinerijos metu naudojamas įrankis, kuris pereina per analizuojamą sistemą, nustatydamas jos aparatūrinės įrangos topologiją bei įdiegtą programinę įrangą bei užregistruodamas šiuos elementus įdiegimo diagramoje.

Sėkminga iteracinio rekursyvaus projektavimo realizacija yra sudėtingas reikalavimas, kurį *UML CASE* priemonių gamintojai ne visada pasiekia. Nors teoriškai aštuonios iš devynių *UML* diagramų turi tiesioginės ar atvirkštinės inžinerijos galimybes, (visos, išskyrus panaudojimo atvejų diagramas), bet realiai tik klasių, objektų ir komponentų diagramos yra aktyviai naudojamos iteraciniame rekursyviame projektavime. Dauguma įrankių sugeba

atvirkštinės inžinerijos metu išgauti *Java ir C++* programų struktūrą, tai yra, klasių antraštes, operacijas ir atributus, bet didžioji dalis funkcinio kodo lieka nematoma, nebent kaip komentaras. *Rational Software* realizavo iteracinį rekursyvų projektavimą tik klasių ir komponentų diagramoms. Tačiau yra keletas gamintojų (*Softera, Together Software, Popkin, ObjectDomain*) [9], kurių sukurti *UML* modeliavimo įrankiai turi tiesioginės ir/arba atvirkštinės inžinerijos galimybes sekos, bendradarbiavimo ir/arba būsenų diagramoms, nors pastarosios diagramos galimybių realizavimo pilnumas yra skirtingas.

Iteracinio projektavimo galimybių realizavimas ir tobulinimas *CASE* įrankiuose nėra išsemtas ir dar daugelis vartotojui naudingų ir projektavimo procesą palengvinančių ar pagreitinančių priemonių nėra realizuotos ar realizuotos nepilnai.

3. *ICONIX* procesas

Projektavimui tikslinga naudoti minimalų *UML* diagramų kiekį, kad analizės procesas būtų kuo paprastesnis ir kad kuo greičiau nuo analizės bei projektavimo būtų galima pereiti prie programos kodo rašymo ar generavimo. Bet diagramų kiekis turi būti pakankamas, kad projektuojamas produktas būtų aprašytas teisingai ir pilnai. Šiais principais remiasi *ICONIX* procesas [2] – programinės įrangos kūrimo metodas, pagrįstas minimaliu *UML* diagramų kiekiu ir efektyvia metodika, kurio dėka kūrimo procesas „nuo panaudojimo atvejų iki kodo" yra greitas ir efektyvus. *ICONIX* labai daug dėmesio skiria reikalavimų apibrėžimui. Šio proceso etapai:

1. *Probleminės srities aprašymas.* [3] Visų pirma sukuriamas srities modelis, kuris praktiškai yra pradinis klasių diagramos variantas. Terminas „probleminė sritis" apima realaus pasaulio daiktus ir sąvokas, susijusias su problema, kuriai išspręsti kuriama sistema. Srities modeliavimo metu randami objektai (klasės), kurie atspindi tuos daiktus ir sąvokas. Srities modelio kūrimo tikslas *ICONIX* procese yra sujungti statinę ir dinaminę šio metodo dalis. Kadangi srities modelis sukuriamas pirmiausia, tai panaudojimo atvejų modeliavimas jau vyksta objektinio srities modelio, o ne abstrakčiai aprašytų vartotojo reikalavimų kontekste. Srities modelis gali būti panaudotas kaip sistemą aprašančių sąvokų žodynas, kurį panaudojimo atvejų diagramų kūrėjai gali naudoti darbo pradžioje. Be to, srities modelis yra kuriamos sistemos statinės klasių diagramos pagrindas.
2. *Panaudojimo atvejų aprašymas.* [4] Antrajame žingsnyje sudaromos glaustai ir tiksliai aprašytos panaudojimo atvejų diagramos, kuriose pateikiami funkciniai reikalavimai kuriamai sistemai. Panaudojimo atvejų aprašymo forma kuo paprastesnė ir aiškesnė skaitytojui – tai vartotojo veiksmų ir sistemos reakcijos į tuos veiksmus aprašymas. Panaudojimo atvejų modelis naudojamas aprašyti vartotojo reikalavimus kuriamai sistemai, detalizuojant vartotojo atliekamų veiksmų scenarijus. Šis modelis yra dinaminės *ICONIX* dalies, o kartu ir viso šio proceso pagrindas. Kuriant panaudojimo atvejų modelį, kiekvienas panaudojimo atvejis trumpai aprašomas, nurodant pagrindinį ir alternatyvius vartotojo veiksmų scenarijus. Kiekvienas panaudojimo atvejis privalo turėti aiškų ryšį su atskiru kuriamos sistemos vartotojo vadovo skyriumi. Panaudojimo atvejų aprašymo metu, giliau išanalizavus sistemą ir radus naujų objektų ar ryšių, turi būti atnaujinamas ir nauja informacija papildomas ankstesniame etape sukurtas srities modelis.
3. *Išbaigtumo (Robustness) analizė*. [5] Sistemos kūrimo metu iškyla du svarbūs klausimai: kas (klausimas keliamas analizės metu) ir kaip (klausimas keliamas projektavimo metu). *ICONIX* procese naudojama išbaigtumo diagrama yra ryšys tarp analizės proceso (kas) ir projektavimo proceso (kaip). Šiame etape atliekama išbaigtumo analizė, kurios dėka patikrinamas ir pataisomas panaudojimo atvejų aprašymo glaustumas ir tikslumas bei sudaromos geresnės sąlygos detalesniam projektavimui. Išbaigtumo analizę pirmasis pasiūlė *Ivar Jacobson*, bet ji nebuvo panaudota *Rational* sukurtoje *UML* versijoje. Išbaigtumo diagrama kuriama kiekvienam panaudojimo atvejui, analizuojant tą panaudojimo atvejį aprašantį tekstą. Išbaigtumo analizė vaidina ne vieną svarbų vaidmenį *ICONIX* procese. Šios analizės metu analizuojamas ir taisomas panaudojimo atvejų modelis bei panaudojimo atvejus aprašantys tekstai, kurie labai svarbūs vėlesniame, sekos diagramų kūrimo procese. Sudarinėjant išbaigtumo diagramas, atliekamas tikrinimas, ar panaudojimo atvejuose aprašyti visi pagrindiniai ir alternatyvūs scenarijai, ar jie įmanomi ir logiški. Taip pat išbaigtumo analizės įgalina naujų probleminės srities objektų atradimą ir tuo užtikrina, kad prieš pradedant kurti sekos diagramas, dauguma srities klasių bus rasta ir aprašyta klasių diagramoje. Būtent išbaigtumo analizės metu gaunamas pilnai aprašytas ir teisingas panaudojimo atvejų modelis bei iš srities modelio gaunama detali statinė klasių diagrama
4. *Sekos diagramų sudarymas.* [6] Paskutiniame etape sukuriamos sekos diagramos, kuriose aprašomas panaudojimo atvejų modelyje aprašytas sistemos objektų elgesys. *ICONIX* procese sekos diagramos yra pagrindinis ir svarbiausias projektavimo darbo produktas. Sekos diagramų sudarymo etape sekos diagrama sutikrinama su išbaigtumo diagrama, tuo užtikrinant, kad kuriama sistema tikrai atitinka vartotojo poreikius.

4. Pasiūlymas *ICONIX* proceso patobulinimui

Pagrindinis *ICONIX* proceso rezultatas – sekos diagramos, kurios pagal šio proceso reikalavimus kartu su statine klasių diagrama toliau panaudojamos kodo rašymui. Vis dėlto *UML* suteikia galimybių ir plačiau bei išsamiau aprašyti kuriamą sistemą, panaudojant ne tik panaudojimo atvejų, sekos ir klasių, bet ir kitas *UML* diagramas. *ICONIX* procesas būtų pilnesnis ir jo metu atliekamas projektavimas būtų išsamesnis, jei šį procesą papildytume būsenų diagramomis, kurios formaliai aprašo visą galimą klasės objektų ar kitų modelio elementų elgesį.

Sekos diagramose pateikiama informacija daugeliu atvejų persidengia su informacija, pateikiama būsenų diagramose. Bet būsenų diagrama apima ne vieną sekos diagramą, todėl turint vien sekos diagramas iš pirmo žvilgsnio būtų neįmanoma susidaryti bendro vaizdo apie sistemos būsenas. Todėl labai naudinga būtų automatizuoti būsenų diagramų sudarymą iš sekos diagramų [7,8]. Panašius procesus šiuo metu realizuoja daugelis *CASE* įrankių – tai automatizuotas bendradarbiavimo diagramų generavimas iš sekos diagramų ir atvirkščiai. Tuo tarpu būsenų modelio generavimas iš sekos diagramų *CASE* įrankiuose nėra realizuotas, nors tai labai palengvintų projektavimo procesą. Būsenų diagramos panaudojimas būtų naudingas ir papildytų kuriamos sistemos aprašymą *UML* modeliais, tuo palengvindamas sekantį etapą – programos kodo rašymą ar generavimą. Vis dėlto scenarijuose pateikiamas tik dalinis sistemos aprašymas, todėl ir iš jų sugeneruota būsenų diagrama negali būti pilna ir išbaigta. Praktiškai iš sekos diagramų sugeneruojamas pradinis būsenų diagramų vaizdas, kurį projektuotojas vėliau turi pakoreguoti ir papildyti.

5. Būsenų diagramų generavimo iš scenarijų algoritmas

Čia aprašytas algoritmas gali būti pritaikytas automatizuotam būsenų diagramos generavimui arba būsenų diagramų suderinimui su sekų diagramomis. Kadangi algoritmas yra nesudėtingas, jį galima pritaikyti ir rankiniu būdu sudarinėjant būsenų diagramas iš turimų sekos diagramų. Laikantis šio algoritmo, net ir neautomatizuoto naudojimo atveju paprasčiau išlaikyti kuriamos sistemos aprašymo vientisumą bei užtikrinti būsenų diagramų ir scenarijų atitikimą. Algoritmas susideda iš dviejų pagrindinių etapų: sekos diagramų aprašymo įvykių sekomis ir būsenų diagramos sudarymo iš tų įvykių sekų. Pirmajame etape kiekviena sekos diagrama (scenarijus) aprašoma įvykių seka. Tarkime, turime sekos diagramą D, kurioje dalyvauja klasės C objektas. Įvykių seka diagramai D objekto C atžvilgiu sudaroma taip:

1. Sekos diagrama pakoreguojama taip, kad pirmasis pranešimas objektui C būtų išeinantis iš objekto C. Jei tokio pranešimo nėra, įvedamas „tuščias pranešimas" iš objekto C į kitą objektą, pavadintas *NULL*. Jei diagramos pabaigoje objektas C nėra sunaikinamas, diagramą reikia papildyti paskutiniu pranešimu, pavadintu *VOID*.
2. Pradedant nuo viršaus, kiekvienai pranešimų porai e_i ir e_j, susijusiai su objektu C, sudaromą įvykių seką reikia papildyti šia pranešimų pora. Kiekviena pora (e_i, e_k) nurodo, kad tam tikru laiko momentu sekos diagramoje objektas C siunčia pranešimą e_i kitam objektui, o po to gauna pranešimą e_k kaip reakciją į e_i iš to kito objekto. Taip sudarytoje įvykių sekoje pranešimas e_i nurodo veiksmą, apibrėžiantį konkrečią būseną, o pranešimas e_k nurodo veiksmą, keičiantį tą būseną į kitą. Jei panešimo e_k (reakcijos į objekto C nusiųstą pranešimą) nėra, vietoje jo rašoma *NULL*. Tada pranešimų pora atrodys taip: (e_i, *NULL*). Kadangi sekos diagrama jau pakoreguota, tai paskutinė sekos diagramos D pranešimų pora bus (e_n, *VOID*). Gaunamas galutinis rezultatas – įvykių seka (e_1, e_2) (e_3, e_4)… (e_{i-2}, e_{i-1}) (e_i, e_{i+1})…(e_{n-1}, e_n).

Antrajame etape generuojama būsenų diagrama. Generavimui naudojamas toks algoritmas (duota – sekos diagramų aibė, aprašyta įvykių sekomis iš porų (e_i, e_{i+1}); rezultatas – būsenų diagrama.):

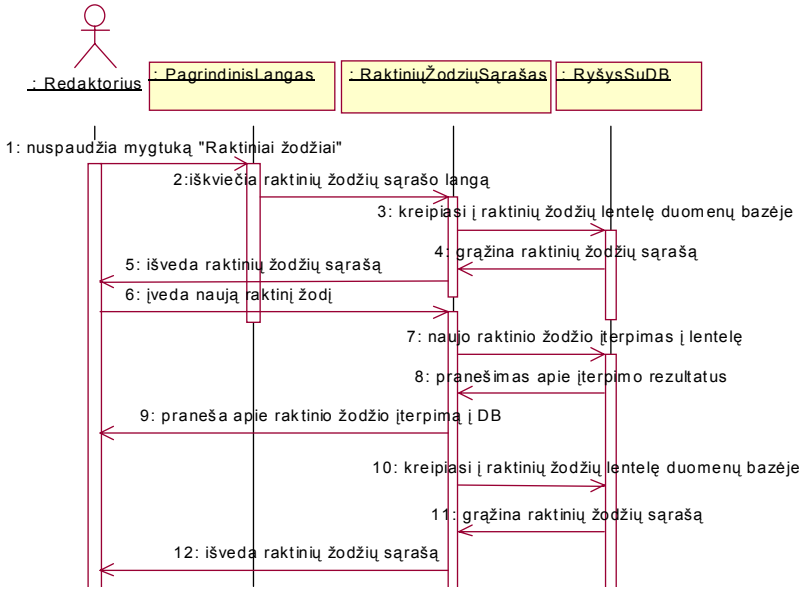
```
WHILE (yra nepatikrintų porų (e_i, e_{i+1})) DO BEGIN
Palyginti porą (e_i, e_{i+1}) su visomis ankstesnėmis poromis.
IF (e_i sutampa su kitos būsenos aprašymu e_{i-n}) THEN
  IF (e_{i-1}=e_{(i-n)-1}) or (e_i=e_{i-n}=NULL) THEN {sutapo ne tik būsenos bet ir perėjimai į jas, arba turime pradinę būseną}
    SutikrintiPerėjimus(e_{i+1}, e_{i-n});
  ELSE {būsenos sutapo, bet prieš tai esantys perėjimai ne - galima gauti ciklą,}
    Klausti vartotojo, ar galimas toks ciklas;
    IF (vartotojas atsako Taip) THEN e_{i-1} perėjimą prijunti prie būsenos e_{i-n}; SutikrintiPerėjimus(e_{i+1}, e_{i-n});
    ELSE Sukurti būseną, aprašytą e_i ir perėjimą iš tos būsenos e_{i+1}; paskutinis perėjimas:= e_{i+1};
ELSE {jei e_i nesutapo su jokiu ankstesniu būsenos aprašymu}
    Sukurti naują būseną, aprašytą veiksmu e_i ir perėjimą iš tos būsenos e_{i+1}; paskutinis perėjimas:= e_{i+1};
IF (paskutinis perėjimas e_{i+1}=VOID) THEN
    Sukurti galutinę būseną; {sekanti būsena, jei tokia bus, neturės prieš ją einančio perėjimo}
```

END *{WHILE}*
PROCEDURE SutikrintiPerėjimus(e_{i+1}, e_{i-n});
Sutikrinti perėjimą e_{i+1} su visais perėjimais iš būsenos e_{i-n};
IF (yra toks perėjimas $e_{(i-n)+1}$, kuris lygus e_{i+1}) THEN paskutinis perėjimas:= $e_{(i-n)+1}$;
ELSE Iš būsenos e_{i-n} sukurti perėjimą e_{i+1};paskutinis perėjimas:= e_{i+1};
END *{Procedure}*

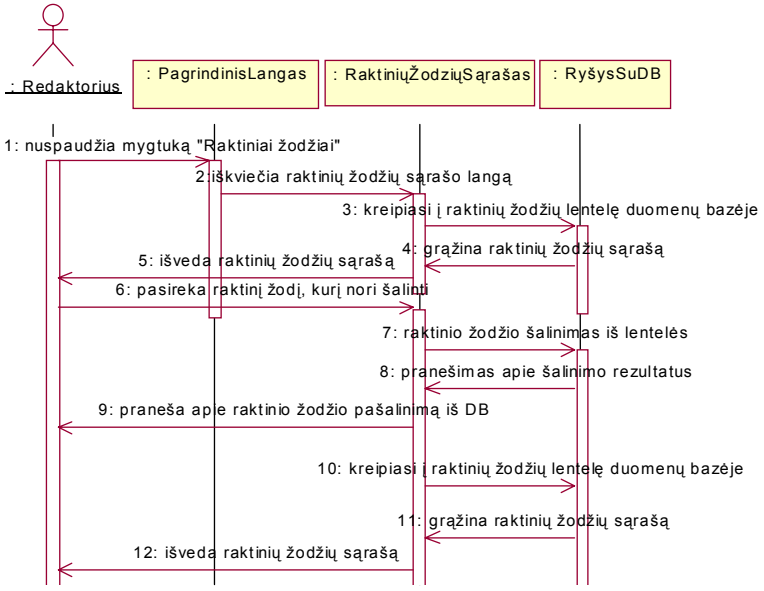
1 pav. Būsenų diagramos generavimo iš scenarijų algoritmas

6. Būsenų diagramų generavimo iš scenarijų pavyzdys

Pavyzdyje analizuojami kai kurie elektroninio katalogo sistemos (elektroninių leidinių katalogas internete) elgesio aspektai. Šio katalogo leidiniai yra suskirstyti pagal temas ir turi jiems priskirtus raktinius žodžius, naudojamus leidinio paieškai. Katalogo redaktorius turi galimybę sukurti ir šalinti raktinius žodžius bei priskirti juos leidiniams. Turime dvi sekos diagramas („Raktinio žodžio įterpimas" ir „Raktinio žodžio šalinimas"), iš kurių reikia sudaryti būsenų diagramą objektui „Raktinių žodžių sąrašas".



2 pav. Naujo raktinio žodžio įterpimo į duomenų bazę scenarijus



3 pav. Raktinio žodžio šalinimo iš duomenų bazės scenarijus

Iš šių dviejų diagramų sudaromos tokios įvykių sekos objektui ,,Raktinių žodžių sąrašas”:

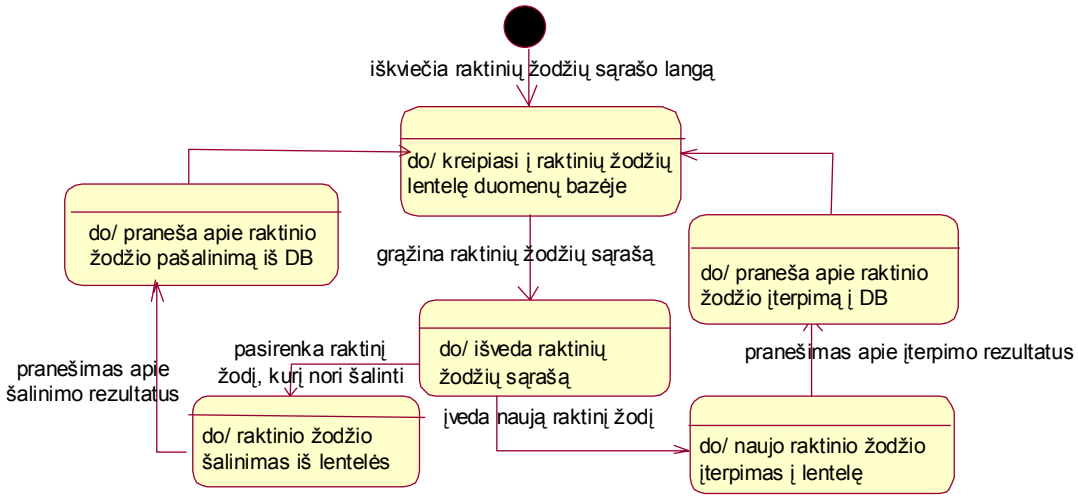
Įvykių seka nr.1 (scenarijus RAKTINIO ŽODŽIO ĮTERPIMAS, objektas RAKTINIŲ ŽODŽIŲ SĄRAŠAS)	
(NULL,	iškviečia raktinių žodžių sąrašo langą)
(kreipiasi į raktinių žodžių lentelę duomenų bazėje,	grąžina raktinių žodžių sąrašą)
(išveda raktinių žodžių sąrašą,	įveda naują raktinį žodį)
(naujo raktinio žodžio įterpimas į lentelę,	pranešimas apie įterpimo rezultatus)
(praneša apie raktinio žodžio įterpimą į DB,	NULL)
(kreipiasi į raktinių žodžių lentelę duomenų bazėje,	grąžina raktinių žodžių sąrašą)
(išveda raktinių žodžių sąrašą,	VOID)

4 pav. Įvykių seka raktinio žodžio šalinimo scenarijui

Įvykių seka nr.2 (scenarijus RAKTINIO ŽODŽIO ŠALINIMAS, objektas RAKTINIŲ ŽODŽIŲ SĄRAŠAS)	
(NULL,	iškviečia raktinių žodžių sąrašo langą)
(kreipiasi į raktinių žodžių lentelę duomenų bazėje,	grąžina raktinių žodžių sąrašą)
(išveda raktinių žodžių sąrašą,	pasirenka raktinį žodį, kurį nori šalinti)
(raktinio žodžio šalinimas iš lentelės,	pranešimas apie šalinimo rezultatus)
(praneša apie raktinio žodžio pašalinimą iš DB,	NULL)
(kreipiasi į raktinių žodžių lentelę duomenų bazėje,	grąžina raktinių žodžių sąrašą)
(išveda raktinių žodžių sąrašą,	VOID)

5 pav. Įvykių seka raktinio žodžio šalinimo scenarijui

Iš šių dviejų įvykių sekų, remiantis aukščiau aprašytu algoritmu, sudaroma tokia objekto ,,Raktinių žodžių sąrašas” būsenų diagrama:



6 pav. Sudaryta objekto ,,Raktinių žodžių sąrašas” būsenų diagrama

7. Išvados

Egzistuoja daug neišnaudotų galimybių patobulinti *CASE* įrankiais vykdomus informacinių sistemų projektavimo metodus bei padidinti jų automatizavimo laipsnį:

- Realizuoti tiesioginę bei atvirkštinę inžineriją kitiems *UML* diagramų tipams, ne tik klasių ir komponentų diagramoms;
- Pasirinkti gerą projektavimo metodiką su minimaliu diagramų skaičiumi, pavyzdžiui, *ICONIX* procesą,, kurį pasiūlyta papildyti būsenų diagramų panaudojimu geresniam suderinamumo užtikrinimui;
- Automatizuoti diagramų generavimą iš kitų diagramų tipų. Straipsnyje pateikiamas algoritmas generuoti būsenų diagramas iš sekų diagramų, kuris palengvintų projektuotojo darbą ir padidintų projekto modelių suderinamumą.

Literatūros sąrašas

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

[2] **D.Rosenberg, K.Scott**. Driving Design with Use Cases. – *Software Development*, 2000. - http://www.sdmagazine.com/articles/2000/0012/0012c/0012c.htm.

[3] **D.Rosenberg, K.Scott**. Driving Design: The Problem Domain. – *Software Development*, 2001. - http://www.sdmagazine.com/articles/2001/0101/0101c/0101c.htm.

[4] **D.Rosenberg, K.Scott**. Top Ten Use Case Mistakes. – *Software Development*, 2001. - http://www.sdmagazine.com/articles/2001/0102/0102c/0102c.htm.

[5] **D.Rosenberg, K.Scott**. Successful Robustness Analysis. – *Software Development*, 2001. - http://www.sdmagazine.com/articles/2001/0103/0103c/0103c.htm.

[6] **D.Rosenberg, K.Scott**. Sequence Diagrams: One Step at a Time. – *Software Development*, 2001. - http://www.sdmagazine.com/articles/2001/0104/0104c/0104c.htm.

[7] **J.Whittle, J.Schumann.** Generating Statechart Designs From Scenarios. *International Conference on Software Engineering*, 2000. - http://www.riacs.edu/research/detail/ase/icse2000.ps.gz..

[8] **E.Makinen, T.Systa.** An Interactive Approach for Synthesizing UML Statechart Diagrams from Sequence Diagrams. http://www.cs.uta.fi/~cstasy/oopsla2000/papers/Makinen.pdf.

[9] **J.Surveyer**. JAVA and UML. http://www.devx.com/upload/free/features/javapro/1999/10mid99/js1399/js1399.asp.

Summary

This article discuses the posibilities of using forward and reverse engineering for different types of UML diagrams, the ICONIX process and the posibility of improving this process with statecharts, which can be generated from scenarios. The algorithm for statechart generation is proposed which facilitates the work of the designner and partially automates it.

ELEKTRONINIO VERSLO PROCESŲ MODELIAVIMAS UML

Lina Nemuraitė, Tatjana Perepliotčikova

Kauno Technologijos Universitetas
Studentų 50-308, LT 3028 Kaunas

Straipsnyje analizuojami reikalavimai elektroninio verslo procesų modeliavimo metodologijai bei jų įgyvendinimas *ebXML* projekte, kuris įgalina sukurti modulinį verslo modelį (schemą), kurį galima vykdyti atitinkama programine įranga. Pateikiamas realaus *B2B* verslo modelio pavyzdys.

1. Įvadas

Šioji diena – elektroninio verslo epochos pradžia. Elektroniniu verslu suprantamas organizacijos darbuotojų, pirkėjų, tiekėjų ir verslo partnerių sąveikų automatizavimas, apimantis tiek organizacijos vidinius, tiek išorinius, tarp organizacijų vykdomus procesus. Elektroninis verslas remiasi interneto technologijomis, kurios iš esmės pertvarko vidinių ir išorinių sąveikų pobūdį. Organizacijos, kurios orientuojasi į elektroninį verslą, siekia įgyti finansinio ir konkurencinio pranašumo. Elektronine komercija suprantama elektroninio verslo dalis, apimanti prekybą elektroniniame tinkle tiek tradicinėmis prekėmis, tiek elektroniniais produktais. Prekybos galimybė pasauliniame tinkle paverčia internetą globalia elektronine rinka.

Siekiant verslo efektyvumo, didėja veiklos procesų automatizavimo ir integravimo poreikiai. Sąveikaujantys skirtingų organizacijų verslo procesai turi remtis bendru procesų modeliu. Bendram procesų apibrėžimui keliami mažiausiai trys principiniai reikalavimai [1]:

- Turi būti bendras veiklos procesų metamodelis ir su juo susijusi schemos apibrėžimo kalba;
- Turi būti mechanizmas, padedantis kurti sudėtingus procesų bei su jais susijusių dokumentų aprašus iš bazinių daugkartinio panaudojimo elementų. Šie baziniai elementai turi turėti modifikavimo bei pritaikymo vertikalių veiklos sričių poreikiams galimybes;
- Turi būti mechanizmas, kuris įgalintų organizacijas skelbti apie savo galimybes vaidinti tam tikras roles bendruose veiklos procesuose, teikiant tinklo paslaugas (*web services*), kad potencialūs partneriai galėtų rasti vienas kitą ir kartu vykdyti veiklos procesus.

Šiuo metu egzistuoja keletas stambių organizacinių struktūrų, kurios sprendžia elektroninio verslo procesų integravimo ir tinklo paslaugų teikimo problemas, pavyzdžiui, *WWW Consortium (W3C), ebXML, RosettaNet*, *Microsoft .Net/BizTalk, UDDI*. Iš jų išsiskiria ***ebXML*** požiūris, kuris siekia įvertinti ankstesnę pramonės patirtį, įgytą taikant elektroninio keitimosi duomenimis standartą *EDI*, bei stengiasi patenkinti visus tris anksčiau minėtus reikalavimus. *EbXML* yra specifikacijų aibė, kuri įgalina sukurti modulinį elektroninio verslo modelį. *EbXML* grupė sukūrė bendrų veiklos procesų metamodelio standartą - specifikavimo schemą, kurioje procesų sąveikos suvienijamos dokumentų srautų pagalba, apibrėžė bazinius *XML* komponentus, kurie įgalina daugkartinį dokumentų ir veiklos procesų schemų panaudojimą, bei pasiūlė saugyklą, kurioje galima registruoti ir rasti verslo partnerius, tinklo paslaugas bei *XML* schemas.

Be modelių ir specifikacijų, *ebXML* apima techninę architektūrą bei informacines technologijas. *EbXML* vizija – sudaryti galimybes sukurti globalią elektroninę rinką, kurioje bet kokio dydžio įmonės, esančios bet kurioje pasaulio vietoje, gali susitikti ir kartu vykdyti verslą *XML* pranešimų pagalba. *EbXML* atsirado bendra Jungtinių tautų centro *UN/CEFACT* ir struktūrizuotos informacijos standartų tobulinimo organizacijos *OASIS* iniciatyva, orientuota globaliam panaudojimui.

EbXML veiklos procesų apibrėžimas pradžioje sudaromas *UML* kalba, vėliau jis transformuojamas į *XML* schemą. *EbXML* procesų modeliavimo metodiką galima taikyti tiek bendriems, tarporganizaciniams, tiek vidiniams įmonės procesams modeliuoti. Proceso specifikacija transformuojama į *ebXML* reikalavimus atitinkančią programinę įrangą, pavyzdžiui, tinklo paslaugų interfeisų konfigūraciją. Proceso vykdymo aplinka kartu yra programų integravimo įranga, kadangi sujungiant skirtingus proceso elementus, tenka jungti skirtingų organizacijų skirtingų tipų programas.

Šiame straipsnyje nagrinėjama *ebXML* projekto dalis, kuri apima veiklos procesų modeliavimą *UML*, siekiant išsiaiškinti jų siūlomos metodikos privalumus bei galimybes taikyti praktinėje veikloje.

2. Verslo procesų modeliavimo metodologija

Dauguma šiuo metu egzistuojančių procesų modeliavimo metodologijų siekia sudaryti ištisinį veiklos proceso aprašą, apimantį visus proceso žingsnius nuo pradinio sužadinančio įvykio iki proceso rezultato gavimo. Tai būdinga *Proforma Corporation* metodikai (realizuotoje *Provision Workbench* pakete) [2], *Action Workflow* [3], *DEMO* [4], *UML* paremtiems metodams [5,6] ir eilei kitų. Procesų automatizavimui orientuoti modeliavimo metodai siekia sudaryti proceso apibrėžimą. Pagal *Workflow Management Coalition* (*WfMC*) standartus [7] proceso apibrėžimu, arba schema, suprantamas kompiuterinis formalizuotas jo aprašas, kurį sudaro veiksmų seka, skirta pasiekti bendram tikslui. Ši seka gali būti pakankamai sudėtinga – ji apima lygiagrečius, nuoseklius veiksmų rinkinius, ciklus bei rekursyvų proceso žingsnių aprašymą – veiksmas gali būti ne tik atominis, bet gali reikšti kito proceso iškvietimą. Proceso apibrėžimas apima ne tik procesą sudarančius veiksmus, bet ir dalyvius, perėjimus nuo vieno veiksmo prie kito, su proceso vykdymu susijusius duomenis bei iškviečiamų taikomųjų programų aprašus. Sudarytas *WfMC* proceso apibrėžimas transformuojamas į *XPDL* (*XML Process Definition Language*). *XPDL* aprašus gali vykdyti *WfMC* standartus atitinkanti darbų sekų valdymo programinė įranga.

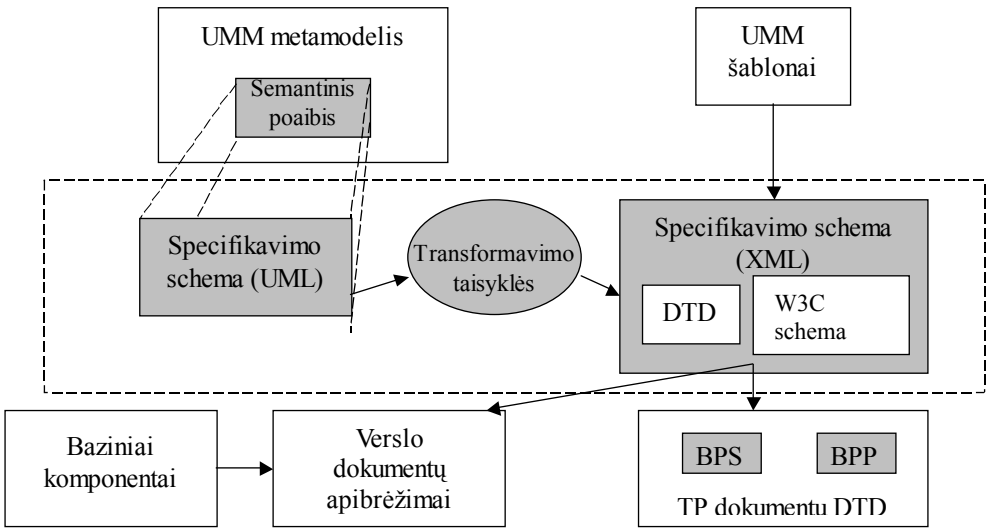
Ištisiniu proceso aprašu paremtos schemos turi apibrėžti visus galimus proceso vykdymo scenarijus. Griežta vykdymo tvarka daugiau būdinga vidiniams organizacijų procesams, tuo tarpu tarporganizacinių elektroninio verslo procesų scenarijų dažnai neįmanoma tiksliai apibrėžti: atskiri procesų elementai (transakcijos) gali kartotis neapibrėžtą skaičių kartų, jų vykdymo tvarka ne visada svarbi. *EbXML* schema sukonstruota moduliniu principu, kuris užtikrina reikiamą vykdymo laisvę.

Kitas svarbus reikalavimas, kurį turi tenkinti elektroninio verslo modeliavimo metodologija – ji turi remtis ekonominėmis sąvokomis, nes elektroninis verslas visų pirma turi būti gerai suformuluotas verslas. Šio reikalavimo netenkina anksčiau minėti metodai. *EbXML* metodologijoje analizuojami ekonominiai įvykiai, kurių metu veiklos dalyviai keičiasi ekonomine verte.

EbXML veiklos procesų modeliavimas remiasi ***UMM*** (*UN/CEFACT Modeling Methodology*) metodologija [8], kuri buvo sukurta, pritaikant *Rational Corporation* sukurtą *Rational Unified Process (RUP)* [9] metodologiją *UN/CEFACT* tikslams modeliuoti veiklos procesus. Tam buvo įvesti stereotipai bei modeliavimo procedūra, kuri įgalina gauti pilną veiklos procesų ir informacijos apibrėžimą, nepriklausomą nuo realizavimo technologijos.

Veiklos proceso aprašas, paremtas *UMM* metodologija, nusako, kokias roles, įsipareigojimus ir ryšius prisiima veiklos dalyviai, sąveikaudami su kitais verslo partneriais. Rolių sąveika apibrėžiama verslo transakcijų aibe. Kiekviena transakcija reiškia apsikeitimą elektroniniais verslo dokumentais. Apsikeitimų seką apibrėžia verslo procesas ir pranešimų siuntimo bei saugumo sumetimai. *UMM* metamodelis turi keletą verslo proceso pjūvių, kurie užtikrina procesų ir informacijos integralumą. *EbXML* veiklos procesų specifikavimo schema [10-12] yra papildomas *UMM* metamodelio pjūvis, kuris apima tik tuos elementus, kurie reikalingi vykdomos programinės įrangos konfigūravimui.

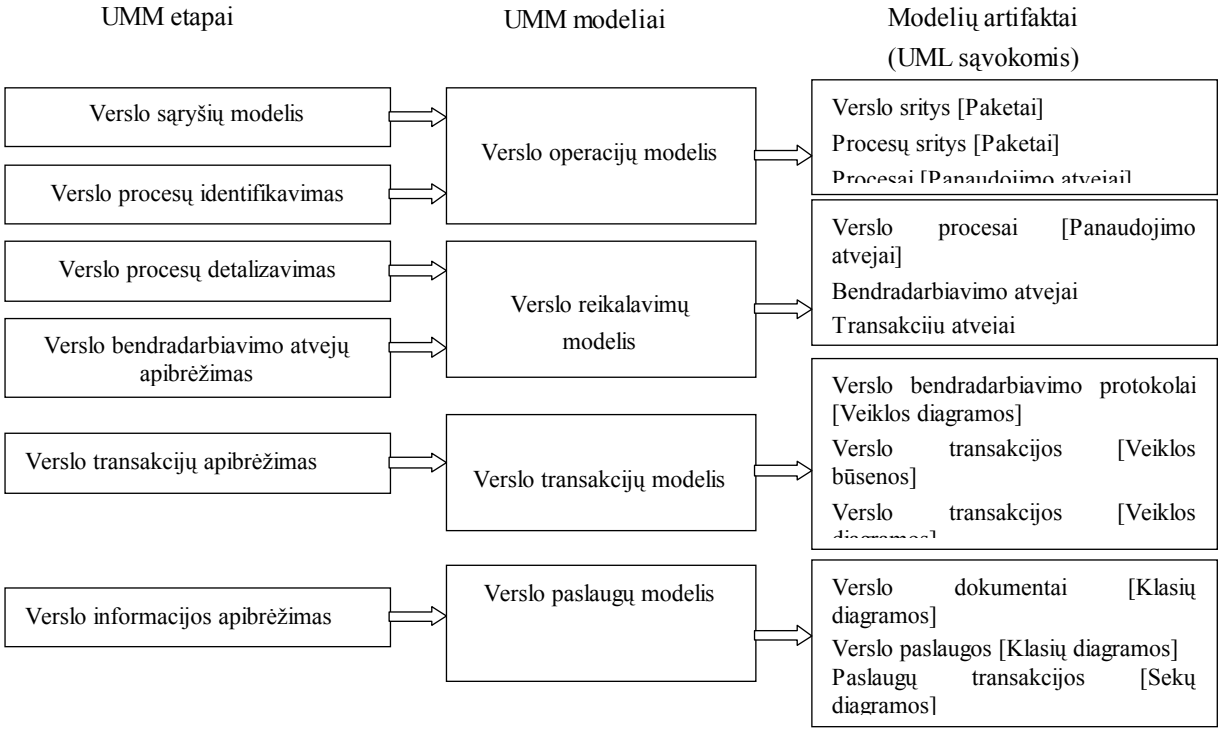
EbXML veiklos procesų specifikavimo schema gali būti pateikta *UML* arba *XML* pavidale (*XML* schemą gali interpretuoti su *ebXML* suderinama programinė įranga). 1 pav. parodytas ryšys tarp *UMM* metamodelio ir *ebXML* schemos. Naudodamas *UMM* metodologiją ir verslo bibliotekas, vartotojas gali sukurti veiklos procesų ir informacijos modelį, po to iš jo automatiškai išgauti *ebXML* schemą.



1 pav. *UMM* ir *ebXML* schemos ryšys

Modeliuojant verslo procesus, vartotojui padeda baziniai daugkartinio naudojimo verslo procesų ir dokumentų komponentai bei standartiniai verslo transakcijų šablonai. Šie šablonai atvaizduoja apsikeitimą verslo dokumentais ir signalais tarp verslo partnerių (verslo signalu suprantamas pranešimas taikomosios programos lygmenyje, kuris praneša apie transakcijos būseną). Specifikavimo schemą *XML* galima generuoti *XML DTD* (*Document Type Definition*) arba *W3C* schemos pavidale (nors pastaroji schema pasižymi geresnėmis savybėmis, tačiau daug kur dar naudojamos *DTD*). *EbXML* procesų specifikavimo schema tarnauja kaip įėjimas, apibrėžiant bendradarbiavimo protokolų profilius (*BPP*) ir bendradarbiavimo protokolų sutartis (*BPS*).

Veiklos procesų modeliavimo etapai ir jų rezultatai, naudojami *ebXML*, parodyti 2 pav.



2 pav. Veiklos procesų modeliavimo etapai ir modeliai, naudojami *ebXML*

Verslo sąryšių modelis nusako pagrindinius konceptus ir jų tarpusavio ryšius, duoda bendrą verslo proceso vaizdą. **Verslo proceso identifikavimo** metu apibrėžiamas aukšto lygio panaudojimo atvejų rinkinys, kuris toliau detalizuojamas **verslo procesų detalizavimo** etape. Pastarajame apibrėžiamos aktorių rolės bei verslo procesų pradinės ir galinės būsenos. **Verslo operacijų modelis** nusako nagrinėjamas verslo sritis ir procesų sritis, kurios atvaizduojamos verslo modelio paketų struktūra. **Verslo bendradarbiavimo atvejų** apibrėžimo metu identifikuojami **ekonominiai įvykiai**, kuriais suprantamas ekonominių išteklių valdymo perdavimas iš vieno dalyvio kitam. Ekonominiai įvykiai nusako procesų pabaigos sąlygas ir sistemos ribas. **Bendradarbiavimo protokolai** apibrėžia bendradarbiaujančius dalyvius ir jų perduodamus informacinius pranešimus. Kiekvieną verslo bendradarbiavimo atvejį sudaro suderinta **verslo transakcijų** seka (*choreography*). *EbXML* galimi du bendradarbiavimo tipai: binarinis ir daugybinis (*multiparty*). Daugybinį bendradarbiavimą tarp keleto verslo partnerių visuomet galima išreikšti binarinių bendradarbiavimo atvejų aibe. Binarinis bendradarbiavimas apima verslo veiksmų aibę. Verslo veiksmas gali būti verslo transakcija arba kitas binarinis verslo bendradarbiavimo atvejis – tokiu būdu galima rekursyviai apibrėžti bendradarbiavimo atvejus. **Verslo transakcija** yra atominis (daugiau neskaidomas) darbo vienetas, kuriame dalyvauja du partneriai ir kuris visuomet baigiasi sėkminga arba nesėkminga būsena. Verslo transakcijų sekas apibrėžia galimi perėjimai iš vieno veiksmo būsenos į kitą. **Verslo transakcijų apibrėžimo** etapas labiau techniškai orientuotas, negu ankstesni etapai. Jame apibrėžiami realūs veiksmai ir organizacijų rolės, kurios pradeda ir užbaigia transakcijas. **Verslo informacijos apibrėžimo** etape aprašomi pranešimai ir verslo paslaugos (klasių diagramomis) bei paslaugų transakcijos (sekų diagramomis). Visi projekto rezultatų elementai (procesai, bendradarbiavimo atvejai, transakcijos, perėjimai ir kiti), be 2 pav. parodytų *UML* artifaktų, aprašomi specifikacijomis, kurios vaidina pagrindinį vaidmenį, generuojant *XML* schemą.

Lyginant šiuos etapus ir modelius su *RUP* metodologija, pažymėtina, kad elektroniniame versle programinės įrangos kūrimą valdo veiklos procesų modeliai. Galutiniame rezultate gauta proceso schema gali būti iš karto vykdoma tam tikra programine įranga (iš anksto turi būti sukurtos atitinkamos tinklo paslaugos). Modeliavime naudojami transakcijų šablonai ir baziniai komponentai įgalina pagreitinti projektavimo procesą, vykdyti jį tada, kai

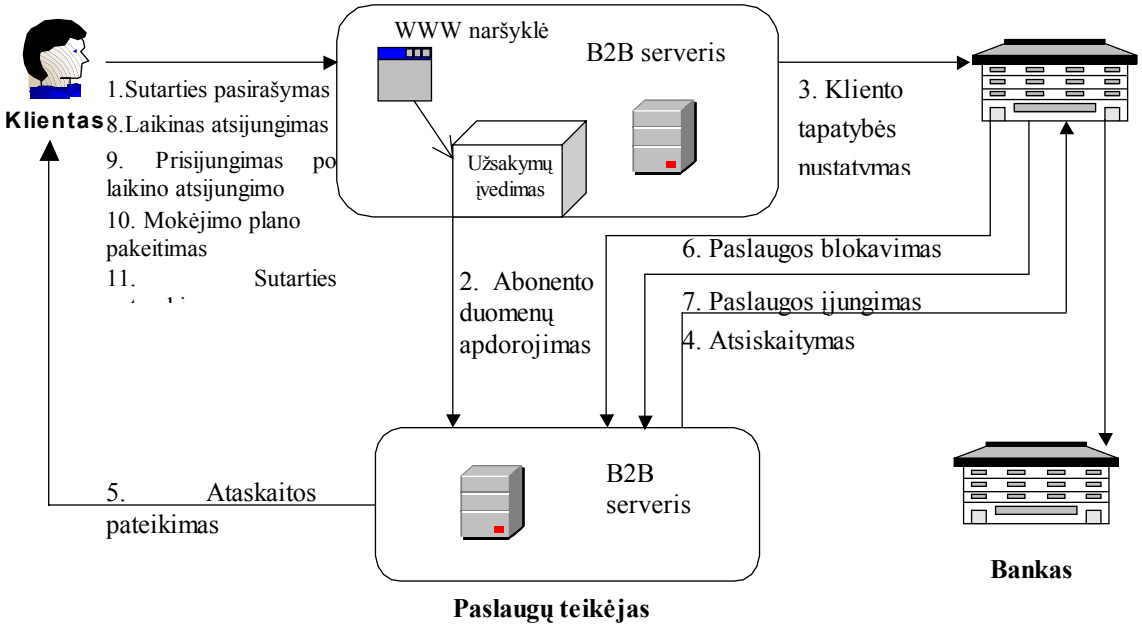
reikia. Modeliams keliami griežti reikalavimai, kiekvienas įvestas stereotipas apibrėžiamas specifikacija, kuri panaudojama schemos generavimui.

3. Interneto paslaugų teikimo verslo modeliavimas, remiantis *ebXML metodologija*

Šiame skyriuje pateikiamas *ebXML* metodologijos taikymo realaus verslo verslui (*Business to Business - B2B*) proceso modeliavimo pavyzdys, kur viena organizacija priima klientų užsakymus interneto paslaugoms, o kita jas teikia. Bendruose procesuose dalyvauja kreditą tvirtinanti organizacija, kuri užtikrina veiklos dalyvių autentiškumą bei saugumą, darant pervedimus už paslaugas iš banko sąskaitų.

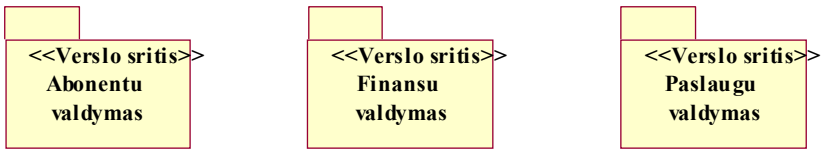
3.1. Verslo sąryšių modelis ir verslo procesų identifikavimas

Verslo sąryšių modelis pateiktas 3 pav. Sąveikų numeracija yra sąlyginė, ji tik apytiksliai nusako procesų eigą. Pvz., 8, 9, 10 procesai gali vykti daug kartų ir nebūtinai tokia eilės tvarka (pagal vykdymo eiliškumą susiję 8 ir 9 procesai). Šis modelis nusako pagrindines sąvokas, ryšius ir bendrą verslo vaizdą.

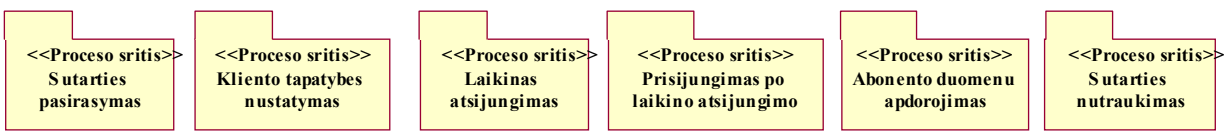


3 pav. Interneto paslaugų teikimo verslo sąryšių modelis

Verslo procesų identifikavimo metu apibrėžiami paketai, kuriuose bus formuojami kiti modeliai (4 ir 5 pav).



4 pav. Interneto paslaugų teikimo verslo sričių modelis



5 pav. Abonentų valdymo procesų sričių modelis

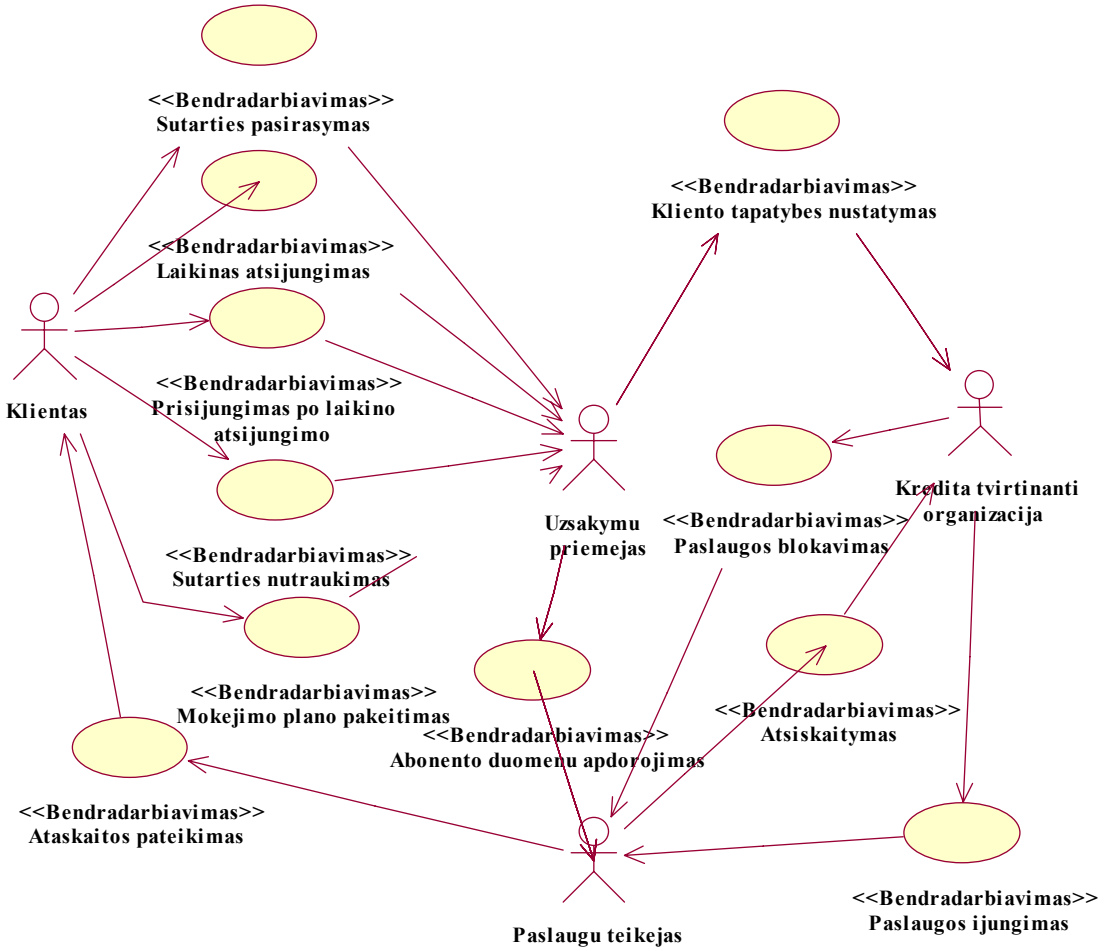
3.2. Verslo procesų detalizavimas ir bendradarbiavimo atvejų apibrėžimas

Verslo procesų detalizavimo metu apibrėžiami aukšto lygio panaudojimo atvejai (procesai), kuriuos sudaro binariniai ir daugybiniai bendradarbiavimo atvejai, kurie išreiškiami binarinio bendradarbiavimo atvejais. Panaudojimo atvejų diagrama, aprašanti binarinius bendradarbiavimo atvejus, pateikta 6 pav.

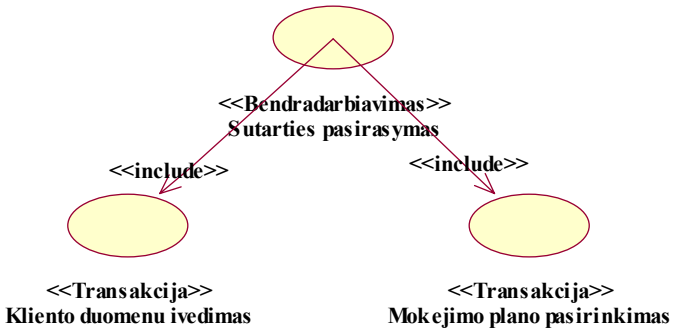
Binariniai bendradarbiavimo atvejai toliau išreiškiami detalesniais bendradarbiavimo arba transakcijų atvejais (7 pav.). Bendradarbiavimo ir transakcijų panaudojimo atvejų specifikacijos sudaro verslo reikalavimų modelį.

3.3. Verslo transakcijų apibrėžimas

Šiame etape verslo bendradarbiavimo protokolai ir verslo transakcijos aprašomos veiklos diagramomis bei specifikacijomis. 8 ir 9 pav. pateikti bendradarbiavimo protokolo ir verslo transakcijos diagramų pavyzdžiai.



6 pav. Panaudojimo atvejų diagrama, aprašanti binarinius interneto paslaugų teikimo verslo bendradarbiavimo atvejus (rodyklės rodo bendradarbiavimo inicijavimo kryptį)



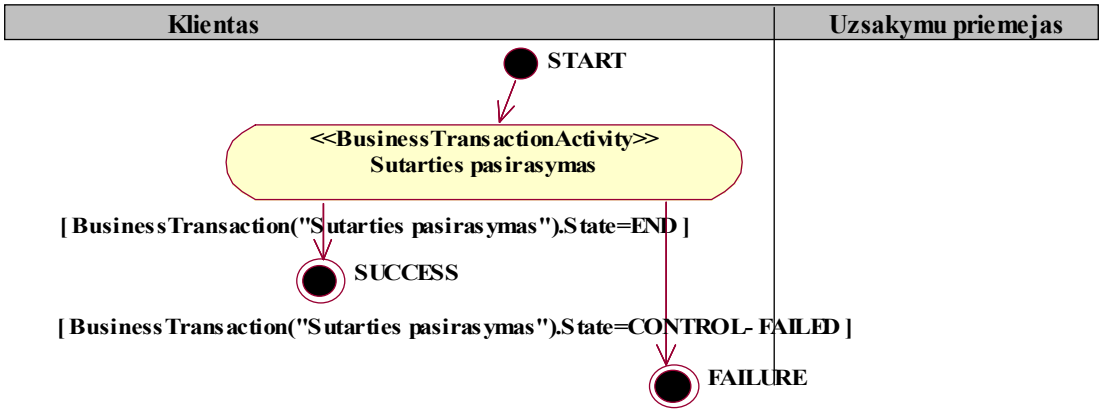
7 pav. Bendradarbiavimo detalizavimas transakcijomis

3.4. Verslo informacijos apibrėžimas

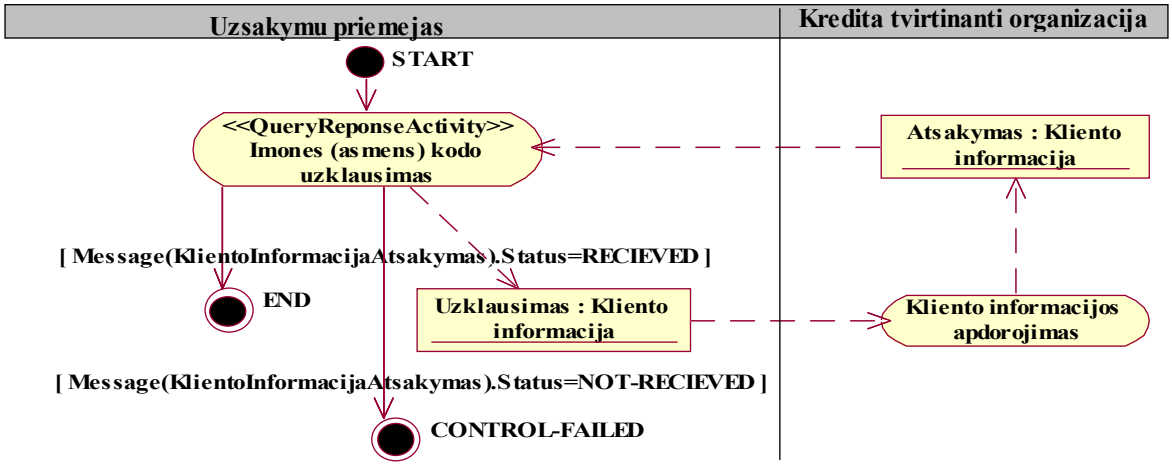
Verslo informacijos apibrėžimas apima dokumentų ir paslaugų apibrėžimus. *EbXML* metodologija nenagrinėja dokumentų apibrėžimo, ji naudoja jau egzistuojančius verslo dokumentus arba duoda galimybę sudaryti juos iš bazinių komponentų.

4. Išvados

EbXML veiklos procesų modeliavimo metodologija pranašesnė už daugelį egzistuojančių metodų, nes ji remiasi ekonomine verslo verte ir aprašo verslo procesus moduliniu principu, nereikalaudama ištisinės žingsnių sekos. Verslo procesai konstruojami iš bendradarbiavimo atvejų ir transakcijų. Tai suteikia reikiamą proceso apibrėžimo ir vykdymo laisvę. Įvertinant paslaugų požiūrį į programinės įrangos kūrimą, galima prognozuoti, kad ateityje vyraus į *EbXML* panašūs veiklos procesų modeliavimo metodai.



8 pav. <<Verslo bendradarbiavimo protokolas>> Sutarties pasirašymas



9 pav. <<Verslo transakcija>>Įmonės arba asmens kodo užklausa
(diagrama paremta užklausos-atsako transakcijos šablonu)

EbXML projektas orientuotas į praktinę realizaciją, jame nenagrinėjami semantiniai specifikacijų validavimo, korektiškumo tikrinimo, informacijos integralumo klausimai, kurie būdingi teoriniams darbams procesų modeliavimo srityje. *EbXML* projekte tokie klausimai sprendžiami, taikant šablonus, bazinius komponentus bei laikantis specifikavimo schemos sintaksės. Tolesnis šio darbo tikslas būtų sudaryti procedūrą, kuri įgalintų patikrinti proceso specifikacijų teisingumą pagal vartotojo apibrėžtas veiklos taisykles.

Literatūros sąrašas

[1] **U.Dayal, M.Hsu, R.Ladin**. Business Process Coordination: State of Art, Trends and Open Issues. – Proceedings of the 27th International Conference of Very Large Data Bases. – Roma, Italy, September 11-14, 2001. – pp. 3-13.

[2] Business Process Modeling - Busness Process Improvement. –*Proforma Corp.*, 2001. – *www.proformacorp.com.*

[3] Business Process Management Sofware For Collaborative Commerce. – *Action Technologies Corp*., 2001. – *www.actiontech.com*.

[4] **V.Reijswoud, M.Lind**. Comparing Two Business Modelling Approaches in the Language Action Perspective. -LAP'98, 1998.

[5] **M.Penker, H.E.Erikson**. Business Modeling With UML: Business Patterns at Work. - *John Wiley & Sons*, 2000.

[6] **Ch.Marshall.** Enterprise Modeling with UML: Designing Succesfull Software Through Busness Analysis. – *Addison-Weesley*, 1999.

[7] WFMC Standards. – *Workflow Management Coalition*, 2001. – http.://wfmc.org/standards/docs.html.

[8] UN/CEFACT Modelling Methodology. – *UN/CEFACT WG*, 2001 July. – *www.uncefact.org*.

[9] Rational Unified Process. – *Rational Software Corp*., 2001 – www.rational.com/products/rup.

[10] **P.Levine, M.McLure** et all. Business Process Analysis Worksheets and Guidelines. - 2001 May 10. - http://www.ebxml.org/specs.

[11] **P.Levine, M.McLure et all.** Business Process and Business Information Analysis Overview, 2001 May 11. - http://www.ebxml.org/specs.

[12] **P.Levine, M.McLure et all**.Business Process Specification Schema. - 2001 May 11. - *http://www.ebxml.org/specs*.

Summary

In this paper requirements for e-business modeling and their implementation in ebXML project are investigated. EbXML business process modeling methodology enables to develop modular business model (schema) which contains information required to configure ebXML compliant software. B2B modeling example is presented.

PANAŠIŲ ATVEJŲ PAIEŠKA, TAIKANT SQL UŽKLAUSAS

Lina Stankevičiūtė, Lina Nemuraitė

Kauno Technologijos universiteta, Informatikos fakultetas
Studentų g. 50, LT–3031 Kaunas

Sparčiai besivystančios elektroninės komercijos sistemos turi daug trūkumų, ir vienas iš jų – ribotas vartotojui teikiamų paslaugų funkcionalumas, kurį galėtų pagerinti intelektualių sistemų taikymas šioje srityje. Mokslinių tyrimų apžvalga rodo, kad tam perspektyvios yra atvejais paremtos sprendimų paieškos sistemos. Straipsnyje nagrinėjamas panašių atvejų paieškos metodas, naudojant *SQL* užklausas.

1. Įvadas

Šiais laikais elektroninė prekyba yra plačiai paplitusi. Dauguma organizacijų bei individualių asmenų naudojasi elektroninės komercijos paslaugomis. Tačiau elektroninė parduotuvė pateikia pirkėjui gana ribotus pasiūlymus: jis turi arba pasirinkti iš pateikto prekių sąrašo, arba iš viso nieko negaus. Tuo tarpu vartotojas dažnai tiksliai nežino reikiamos prekės tipo ar sudėties – jam reikia eksperto, kuris padėtų teisingai pasirinkti. Paieška pagal raktinius žodžius daugeliu tokių atvejų netinka. Ekspertinių sistemų pritaikymas elektroninėje komercijoje suteiktų vartotojui geresnį aptarnavimą: jis galėtų nurodyti tam tikras savybes ir sistema parinktų tinkamą produktą arba pasiūlytų ką nors panašaus. Ekspertinė sistema galėtų pasiūlyti ir sudėtingą produktą, surinktą iš atskirų dalių.

2. Ekspertinių sistemų tipai

Pagal *D.A.Waterman* [1], ekspertinė sistema – tai kompiuterinė programa, naudojanti ekspertines žinias, kad užtikrintų efektyvų uždavinių sprendimą tam tikroje siauroje srityje. Ekspertinės sistemos yra dviejų tipų: paremtos taisyklėmis (*rule– based*) ir paremtos atvejais (*case– based*). Taisyklėmis paremtų ekspertinių sistemų naudojimas, sprendžiant sudėtingas realaus pasaulio problemas, yra sunkus uždavinys. Vienas iš pagrindinių sunkumų yra tai, kad taisykles reikia išgauti iš ekspertų, kurie puikiai sugeba spręsti savo srities problemas, bet dažniausiai sunkiai gali paaiškinti, kaip jie tą daro. Žinių bazes kuriantys specialistai (žinių inžinieriai) iš patirties žino, kaip sudėtinga apibrėžti ekspertų žinias loginėmis taisyklėmis, kai viską reikia išreikšti formaliai. Be to, taisyklėmis paremtas žinių bazes reikia nuolat papildyti, įvedant naujas žinias ir modifikuojant senas.

Atvejais paremtų sprendimų paieškos (***Case Based Reasoning*** – ***CBR***) sistemų metodai yra geresni tuo, kad tokios sistemos gali mokytis iš patirties ir jų žinios didėja, augant apdorotų atvejų skaičiui. Ekspertai gali kalbėti apie savo sritį, pateikdami pavyzdžius, o ne formuodami taisykles. *CBR* pateikia metodologijas įvairių taikomųjų programų kūrimui, validavimui ir palaikymui. *CBR* metodai vertingi, kai problemos nėra pilnai suprantamos ir taisyklės turi daug išimčių. Tokiais atvejais didelis sudėtingų ar specifinių kontekstų skaičius padaro taisyklėmis paremtus taikymus neadekvačiais

Atvejais paremtų sistemų (toliau vadinsime jas *CBR*) panaudojimas plečiasi, nes jos turi potencialias galimybes pagerinti elektroninės komercijos sistemų veikimą. *CBR* padeda nekvalifikuotam vartotojui surasti paslaugas ar produktus, kai jis tiksliai nežino, kokių produktų jam reikia, ir gali nurodyti tik pageidaujamas jų savybes. Tokios sistemos gali pasiūlyti panašius produktus, jei reikiamo produkto nėra. Viena iš naujų galimybių realizuoti atvejais paremtos sprendimų paieškos sistemas – naudoti *SQL* užklausas. Tai paranku tiek mažoms įmonėms, tiek ir didelėms organizacijoms, kadangi šiuo atveju nereikia diegti naujo programinio produkto, galima panaudoti jau egzistuojančią įmonės ar organizacijos duomenų bazę.

3. Atvejais paremta sprendimų paieška

Pagrindinė visų atvejais paremtų sprendimų paieškos metodų idėja yra panašios problemos radimas atvejų bazėje, kuri buvo sukaupta, sprendžiant realias problemas ir mokantis iš praktikoje patvirtintų sėkmingų ar nesėkmingų sprendimų bei klaidų. *CBR* procesą galima aprašyti keturiais pagrindiniais žingsniais [2]:

1. Rasti panašiausią atvejį;
2. Dar kartą panaudoti šį atvejį, mėginant išspręsti naują problemą;
3. Jei reikia, patikrinti siūlomą sprendimą;

4. Išsaugoti naują sprendimą, kaip naują atvejį.

Nepaisant to, kad uždaviniai gali būti skirtingi, pagrindinė idėja išlieka ta pati.

Radimo metu panašiausias atvejis ar atvejų rinkinys atvejų bazėje apibrėžiami, remiantis naujų problemų aprašymu. Daugkartinio panaudojimo metu informacija ir žinios radimo atveju naudojami naujai problemai spręsti. Naujos problemos aprašymas turi būti suderinamas su informacija, esančia senuose atvejuose, suformuotuose pagal išspręstus atvejus.

Peržiūrėjimo metu siūlomo sprendimo tinkamumas įvertinamas realiame pasaulyje. Jeigu reikia ir įmanoma, siūlomas atvejis turi būti pritaikomas keliose srityse. Jeigu atvejo sprendimas sudaromas patikrinimo fazėje, jis turi būti saugomas ateities problemų sprendimui, o išsaugojimo fazėje atvejų (žinių) bazė atnaujinama naujai sužinotais atvejais. Laikui bėgant, žinios sensta, todėl pilnas *CBR* veikimo ciklas apima ir žinių bazės atnaujinimą, kurio metu pasenusios žinios archyvuojamos, pašalinamos iš aktyvios atvejų bazės.

Išreikšti žinias, naudojant *CBR*, nėra paprastas veiksmas, kurį galima įvykdyti vienu žingsniu. Tam turi būti paruoštas procesas, kurio metu būtų galima:

- surinkti verslo parametrus, kurie bus naudojami, priimant sprendimą;
- pagal šiuos parametrus aprašyti reikiamus atvejus (žinias);
- nuolat palaikyti žinių bazės kokybę.

4. Atvejų pateikimo būdai

Kuriant atvejais paremtas žinių bazes, pirmiausia reikia nuspręsti, kaip aprašyti atvejus kompiuterio viduje [3]:

- Kaip atvejai bus saugomi (duomenų bazėje, binariniame faile ar kituose elektroniniuose dokumentuose);
- Koks bus paieškos mechanizmas;
- Kokio tipo žinias bus galima išgauti.

Komercinėse *CBR* sistemose atvejų pateikimui taikomi trys skirtingi metodai: tekstinis, dialoginis ir struktūrinis. **Tekstiniame** *CBR* metode atvejai pateikiami laisva tekstine forma. **Dialoginiame** *CBR* metode atvejis aprašomas klausimų ir atsakymų sąrašu. Kiekvienam atvejui atsakymai gali būti skirtingi. **Struktūriniame** *CBR* metode atvejais paremtų sprendimų projektuotojas iš anksto apibrėžia atvejų struktūrą: atvejų klasifikaciją ar atvejus aprašančių klasių hierarchiją arba/ir klasių savybes. Pavyzdžiui, norint pirkti namą, realią situaciją gali aprašyti kaina, kambarių skaičius, plotas, miesto rajonas ir panašiai, o paieškos rezultatas gali būti konkrečių parduodamų namų sąrašas. Tokių struktūrizuota atvejų tipų aibė su jų tarpusavio ryšiais ir savybėmis vadinama dalykinės srities modeliu. Šiame straipsnyje orientuojamasi į struktūrinį atvejų pateikimą.

5. Atvejų paieška, paremta panašumų skaičiavimu

Išskirtinė *CBR* sistemų savybė yra mechanizmas, kuris įgalina bazėje rasti panašius atvejus. Šį mechanizmą galima supaprastinti, paverčiant panašaus atvejo radimą skaičiavimo uždaviniu, kuris randa „artimiausią kaimyną". *CBR* srityje egzistuoja daug panašių atvejų paieškos metodų: k–d medžiai, atvejų tinklai ir kiti. Kadangi šiame darbe atvejai aprašomi dalykinės srities modeliu, toliau nagrinėjami šiam požiūriui tinkami paieškos metodai. Galima išskirti tokius panašumų skaičiavimo variantus:

- Paprasčiausiu atveju panašumą tarp dviejų atvejų (pvz., produktų) galima apibrėžti panašumo matu, kuris įgyja reikšmes iš intervalo [0,1]. Tokių elementarių atvejų panašumai aprašomi, sudarant **panašumų matricą**, kurios elementai nusako panašumus tarp dviejų objektų: **panašumas** $(e_i, e_j) \in [0,1]$.

Atvejų panašumų matrica gali būti nesimetrinė, t.y., **panašumas** $(e_i, e_j) \neq$ **panašumas** (e_j, e_i).

- Jei atvejai aprašomi kaip objektų tipai, turintys savybių rinkinius, dviejų atvejų panašumą galima apskaičiuoti kaip tų atvejų savybių panašumų sumą, prieš tai apibrėžus visų atvejį apibrėžiančių savybių panašumų įvertinimus. Apskaičiuojant individualių savybių reikšmių panašumą, naudojamos įvairios metrikos, priklausančios nuo duomenų tipo (skaičiai, simboliai ir panašiai). Simboliniu duomenų tipu išreiškiamų savybių panašumai, kaip ir elementarių atvejų panašumai, išreiškiami tų savybių reikšmių panašumo matricomis, pavyzdžiui, spalvų panašumo matrica. Skaitmeninio tipo duomenų, datų savybių panašumui nustatyti naudojama **atstumo** sąvoka. Kiekvienai tokio tipo savybei reikia apibrėžti galimą kitimo diapazoną, ir dviejų savybių reikšmių panašumas skaičiuojamas kaip absoliutinio tų reikšmių skirtumo ir diapazono santykis:

$$\textbf{panašumas}\ (e_i, e_j) = |e_i - e_j| / (e_{max} - e_{min})$$

Kiekviena savybė gali turėti **svorį**, nusakantį jos įtaką bendram atvejo aprašymui.

- Jei nagrinėjamas produktas, sudarytas iš sudėtinių dalių, panašumų skaičiavimas analogiškas panašumų skaičiavimui savybių sąrašo atveju.
- Jei atvejai aprašomi, naudojant klasifikavimo schemą, skirtingų klasifikacijos lygių atvejų panašumai skaičiuojami pagal tam tikrą algoritmą. Jei vartotojas užklausoje nurodo „tėvą", visų to tėvo vaikų ir užklausos panašumai lygūs vienetui. Norint pateikti didesnį rezultatų skaičių, galima nagrinėti ir kitus „tėvo" lygio atvejus, turinčius su juo bendrą protėvį, analizuojant žemiausio lygio vaikų panašumų matricą. Čia reikia įvertinti klasių *inter* ir *intra* panašumus. Tokie algoritmai nagrinėjami [4].
- Sudėtingiausias panašumų įvertinimas tuo atveju, kai atvejai aprašomi klasių apibendrinimo/specializavimo hierarchija, kai savybes gali turėti ir tarpinės, ir žemiausio lygio klasės, kurių egzemplioriai yra konkretūs atvejai (pvz., produktai). Ir šiuo atveju panašumų skaičiavimą galima privesti prie analogiško algoritmo [5].

Jei atvejų bazė yra pakankamai didelė, ji turi būti indeksuota. Galimi skirtingi indeksavimo mechanizmai. Medžiai naudojami, indeksuojant dideles atvejų bazes. Sprendimų medis yra hierarchinis saugomų atvejų išskaidymas, paremtas savybių reikšmėmis. Vyriausia viršūnė (kamienas, šaknis) apima visus atvejus, žemesnių lygių viršūnės palaipsniui išskaido atvejus į poaibius, priklausomai nuo įvairių savybių, taikant tam tikrą išskyrimo tvarką. Sprendimų medis nustato paieškos vykdymo kelią, pagal kurį galima rasti reikiamus atvejus.

6. Egzistuojantys *CBR* produktai

Egzistuoja nemažai sistemų, paremtų panašių atvejų paieška. Ankstesni *CBR* produktai dažniausiai buvo kuriami kaip uždaros sistemos, skirtos specifiniams tikslams, dažniausiai tam naudojant paieškos kompiuterio atmintyje algoritmus arba specifines žinių bazes. Viena iš didžiausių ir tobuliausių šiuolaikinių *CBR* sistemų – organizacijos *Tech:inno GmbH* sukurta programa *CBR–Works4*. Ji pritaikyta *CBR* sistemų kūrimui ir jų atvaizdavimui internete. Šalia atvejų atrankos, joje galimas atvejų struktūros modeliavimas ir atvejų bazės priežiūra. *CBR–Works4* naudojama objektiškai orientuota atvejų struktūra. Jos veikimo mechanizmas apima visą *CBR* ciklą, pradedant nuo atrinkimo ir baigiant atvejų bazės atnaujinimu. *CBR–Works4* yra išbaigta aplinka, galinti dirbti kaip *CBR s*erveris keletui klientų. Šioje sistemoje naudojama specialiai tam sukurta atvejų užklausų kalba *CQL* (*Case Query Language*). Be to, *CBR–Works4* turi atvirą interfeisą atvejų bazių kūrimui iš egzistuojančių duomenų. Ši programa veikia įvairiose platformose: *Win9x/Win NT, Solaris, HP–UX, IBM–AIX, Apple Macintosh, OS/2.*

CBR–Works4 yra labai galinga sistema atvejais paremtų taikomųjų sistemų kūrimui. Tačiau nedaugelis įmonių ar organizacijų nori prisirišti prie paketo, specializuoto konkrečiam panaudojimui, kadangi panašių atvejų paieškos funkcijos reikalingos įvairiems tikslams – jos naudojamos elektroniniuose kataloguose, organizacijų portaluose, elektroninės prekybos ir elektroninio verslo sistemose, kuriose nuolat vyksta programinės įrangos perkonfigūravimas. Daugelis organizacijų savo duomenų saugojimui naudoja reliacines ir objektines–reliacines duomenų bazes, kurios remiasi *SQL*. Todėl tikslinga išplėsti egzistuojančių duomenų bazių valdymo sistemų (DBVS) funkcionalumą, sukuriant jose panašių atvejų paieškos galimybes, taikant *SQL*, panašiai kaip jau realizuotos duomenų gavybos (*Data Mining*), analitinio apdorojimo (*OLAP*) funkcijos.

7. Atvejais paremta sprendimų paieška, naudojant *SQL* užklausas

Šio metodo idėja remiasi panašių atvejų išgavimu iš reliacinių duomenų bazių, tam panaudojant *SQL* užklausas su panašumų skaičiavimu. Tai padėtų išvengti atvejų duomenų dubliavimo ar indeksavimo struktūrų sudarymo. Toks metodas tiktų didelėms duomenų bazėms, kuriose dažnai keičiasi duomenys. Tipinės tokių duomenų bazių savybės:

- Produktas yra saugomas egzistuojančioje reliacinėje duomenų bazėje, kuri palaikoma produktų tiekėjų ir naudojama daugelyje kitų paslaugų, susijusių su produktu;
- Produktų kiekis duomenų bazėje yra labai didelis;
- Produkto duomenų bazėje vyksta greiti ir nuolatiniai pokyčiai, ypač tuomet, kuomet produktai yra atskiri daiktai (pvz., padėvėtos mašinos), kurie tampa nereikalingi, juos pardavus;
- Kiekvieną produktą galima laikyti atskiru atveju. Produktų aprašymas paprasčiausiu atveju yra paprastas – analizuojami atskiri produktai arba jų savybių sąrašai, sudėtingesniais atvejais galima produktų klasifikacija ir klasių hierarchija.

Yra galimos kelios produktų bazės sujungimo su atvejais paremto išgavimo komponentais architektūros: išgavimas iš vidinės duomenų bazės ir išgavimas iš išorinės duomenų bazės.

Iš techninės pusės, idealus sprendimas būtų integruoti atvejais paremtą išgavimo funkciją į duomenų bazę. Tai suteiktų efektyvų priėjimą prie duomenų, o taip pat užtikrintų nuoseklią esamo momento duomenų peržiūrą. Duomenų bazė galėtų automatiškai apdoroti atvejais paremtą užklausą, išreikštą standartine užklausos forma, kuri apimtų lankstų panašumo matų atvaizdavimą. Deja, tokių savybių dar neturi komercinės duomenų bazių valdymo

sistemos. Taip pat nėra standartizuotos kalbos, skirtos tokioms atvejais paremtų užklausų formuluotėms, kurios būtų pripažintos plačios bendruomenės. Šiame straipsnyje bandoma suformuluoti tokių galimybių įgyvendinimo metodą.

Kitas variantas remiasi atvejais paremtu išgavimu iš duomenų bazės, esančios išorėje – atskirai nuo kliento programos [6]. Išgavimo veiksmas tampa atskiru moduliu, kuris, priėjimui prie produkto duomenų, sąveikauja su atitinkama duomenų baze. Čia taip pat įmanomi keli variantai:

- Išgavimo procedūros vartojimas*:* šis metodas vykdomas produktų duomenų bazės išgavimo procedūros viduje. Jo trūkumas – reikalaujamos atminties kiekis yra dvigubinamas, kadangi yra būtina kopijuoti atvejų bazę kiekvienąkart atnaujinant produktų duomenų bazę, todėl atsiranda nuoseklumo problemos;
- Atvejais paremto išgavimo indeksinės struktūros sudarymas ir saugojimas išgavimo mechanizme. Čia išvengiama duomenų dvigubinimo problemos, tačiau išlieka nuoseklumo problema;
- Panašumu paremtas išgavimas, naudojant *SQL* užklausas: kiekvienai užklausai yra sugeneruojama *SQL* užklausų seka tam, kad iš duomenų bazės paimti atvejus, kurie vėliau yra panaudojami panašumo nustatymui. Šis metodas išvengia nuoseklumo problemos ir visada remiasi esamo momento duomenimis. Problema yra efektyvios programos sukūrimas.

7.1. Panašių atvejų radimas

Panašių atvejų radimas priklauso nuo reliacinėje duomenų bazėje saugomų duomenų struktūros ir norimo rezultato. Skirstant panašumų paiešką pagal norimą rezultatą, galima būtų išskirti tokius atvejus: kai ieškome konkretaus produkto (atvejo); kai ieškome produkto (atvejo) pagal nurodytas savybes, nurodžius „tėvą"; kai ieškome produkto pagal jo sudėtį. Kiekvienam atvejui reikia sukurti algoritmą, kuris padėtų vartotojui surasti norimą kiekį panašių produktų pagal pateiktą užklausą.

Pirmuoju atveju, kai ieškome panašaus produkto, produktas suprantamas kaip atvejis. Tokie atvejai gali būti lengvai atvaizduojami reliacinės bazės vienoje lentelėje: kiekvienas atributas atitinka vieną lentelės stulpelį, kiekviena lentelės eilė galutinai nusako atskirą atvejį. Atributas yra neapibrėžtas, jei atitinkama duomenų bazės lentelės reikšmė yra lygi *null*. Tokiems atvejams sąlyginai lengva apskaičiuoti panašumą į užklausą: kiekvienam atributui galima priskirti vietinio panašumo matą, kuris nusakytų panašumą tarp skirtingų atributo reikšmių. Tarkime, kad vietinis panašumas lygus 1, jeigu užklausos atributas yra neapibrėžtas, kas reikštų „nesvarbumo" semantiką, ir lygus 0, jei nėra apibrėžtas tik atvejo atributas, kuris išreiškia "baudą" atvejui, neturinčiam vartotojo užklaustos savybės. Bendras panašumas tarp dviejų objektų gali būti apskaičiuotas, įvertinant kiekvieno atributo vietinį panašumą. Taigi, panašumus tarp atvejų galima nusakyti naudojantis panašumų matrica, kurią sudaro ekspertas, nusimanantis atitinkamoje dalykinėje srityje.

To pasėkoje jau turimą duomenų bazę reikia papildyti, sukuriant papildomą lentelę – panašumų matricą. Lentelėje nurodomi atvejai, tarp kurių nustatomas panašumas, ir panašumo skaitinė vertė, priklausanti intervalui nuo 1 iki 0 (1 – labai panašus, 0 – visai nepanašus). Papildomos lentelės stulpeliai būtų: atvejis1, atvejis2, panašumas. Joje turi būti kiekvieno atvejo panašumai su visais kitais atvejais, įskaitant ir jį patį.

Panašių atvejų radimo metu galima nurodyti, koks mažiausias panašumas tenkina vartotoją. Taip pat nurodamas ir pageidaujamų rezultatų kiekis. Pavyzdžiui, *SQL* užklausa panašiausių atvejų radimui galėtų atrodytų taip:

```
select top 20 atvejis2
from Panasumai
where atvejis1 = 'loreal'
and panasumas > 0.5
order by panasumas desc
```

Ši užklausa iliustruoja 20 panašių produktų paiešką produktams „*loreal*", kurių panašumas nemažesnis už 0.5. Rasti produktai išdėstomi panašumų mažėjimo tvarka.

Naudojant tokio tipo užklausas sistemoje, reikėtų jas labiau apibendrinti: vietoj konkretaus produkto („*loreal*") bei panašumo reikia įvesti kintamuosius, kuriuos vartotojas pasirenka iš pateikto sąrašo, arba įveda klaviatūros pagalba.

Panagrinėsime sudėtingesnį atvejį, kai nurodomas ne tik produktas, bet ir jo savybės, ir produktai duomenų bazėje yra suklasifikuoti, tai yra, tam tikras produktas gali būti kitų produktų „tėvas". Pavyzdžiui, „šampūnas" yra „tėvas" „*Elvital*", „*Avon*", „*Timotei*" šampūnams. Tuo tarpu „*Elvital šampūnas*" yra „tėvas" šampūnams „*Elvital NutriFilter"*, „*Elvital Normal"*.

Šiuo atveju vartotojo užklausoje nurodomas produkto „tėvas" ir savybių apribojimai. Kad būtų galima išspręsti šį uždavinį, turimą duomenų bazę reikia papildyti lentelėmis, kurios apibrėžtų skaitmeninių savybių kitimo diapazonus, simbolinių savybių panašumų matricas. Panašių atvejų paieškos algoritmas turėtų būti vykdomas tokiu būdu: pirmiausiai ieškome tame pačiame hierarchijos lygyje, jei nerandame reikiamo kiekio atvejų, ieškome „vaikų", turinčių bendrą „protėvį" su užklausoje nurodytu „tėvu", kol pagaliau randamas reikiamas kiekis panašių atvejų. Jei savybių reikšmės yra simbolinės, panašumus randame iš savybių reikšmių panašumų matricos, jei

skaitmeninės, skaičiuojame atstumų įvertinimus iki užklausoje nurodytos apribojimo reikšmės. Visa tai realizuojama vienoje procedūroje, kuri kreipiasi į panašių atvejų paieškos *SQL* užklausą. Procedūros įėjimo parametrais turi būti produkto „tėvas", savybių apribojimai bei pageidaujamas panašių atvejų kiekis. Viena iš galimų procedūros struktūrų pavaizduota 1 paveiksle.

Užklausos panašių atvejų radimui, kai nurodyta savybė yra simbolinė reikšmė, pavyzdys pateiktas 2 paveiksle. Šiame pavyzdyje parodyta, kad ieškome šampūno normaliems plaukams, t.y. savybės „plaukų tipas" reikšmė yra „normalūs ". Rasti produktai išdėstomi pagal panašumus mažėjimo tvarka, kadangi mums svarbiausi yra labiausiai panašūs produktai. Žinoma, kaip ir pirmuoju atveju, pateikta užklausa turėtų būti apibendrinta visiems galimiems atvejams, įvedant kintamuosius, o taip pat įvertinant, kad savybei gali būti nurodytas daugiau nei vienas apribojimas.

```
Rastų_atvejų_skaičius = 0;
WHILE Rastų_atvejų_skaičius < pageidaujamas_atvejų_skaičius DO
   Vykdyti SQL užklausą panašių atvejų radimui;
   Surašyti užklausos duomenis į rezultatų lentelę;
   Rastų_atvejų_skaičius = Rezultatų_lentelės_įrašų_skaičius;
   IF  Rastų_atvejų_skaičius  <  pageidaujamas_atvejų_skaičius
THEN
            Išrinkti visus "vaikus";
   END; {if}
END
Pateikti vartotojui rezultatų lentelę.
```

1 pav. Panašių atvejų išgavimo procedūra

```
select top 20 Atvejo_savybe.atvejis, Atvejo_savybe.savybe,
Simboline_reiksme.reiksme, Savybiu_panasumai.panasumas
from Atvejo_savybe, Savybiu_panasumai, Simboline_reiksme
where Atvejo_savybe.reiksmes_id = Savybiu_panasumai.savybe2
and Simboline_reiksme.id = Savybiu_panasumai.savybe1
and Savybiu_panasumai.savybe1 = 'normalūs'
and Savybiu_panasumai.panasumas > 0.5
order by Savybiu_panasumai.panasumas desc
```

2 pav. Panašių atvejų radimo *SQL* užklausos pavyzdys

7.2. Atvejų atvaizdavimo metamodelis

Apibendrinant galimus atvejų ir užklausų struktūros variantus, buvo sudarytas atvejų atvaizdavimo metamodelis (3 pav.), kuris įgalintų sudaryti tiems variantams tinkamus panašumų skaičiavimo funkcijų algoritmus.

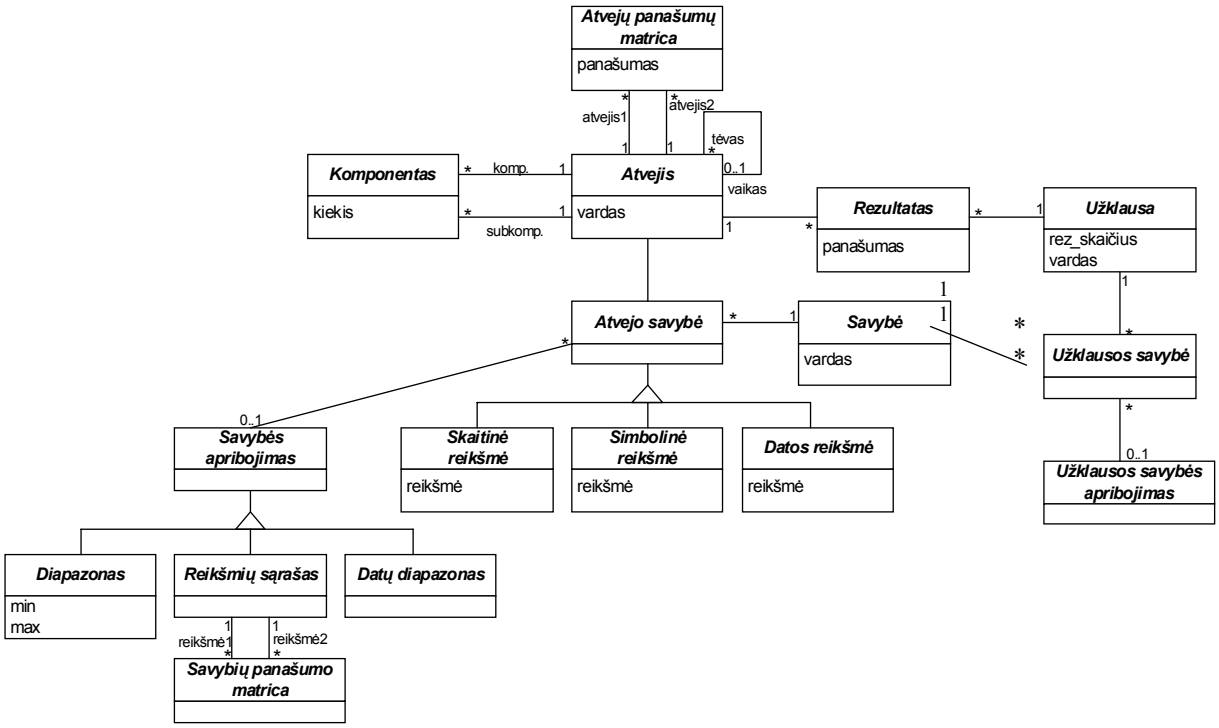
Šiame metamodelyje įvertinta atvejų hierarchija (ryšiu „tėvas – vaikas"), atvejų savybės, atvejų sudėtis (klasė „Komponentas"), galimybė aprašyti savybes skirtingais duomenų tipais (simbolinėmis, skaitmeninėmis, datų ir kitomis reikšmėmis – reikšmių sąrašą galima plėsti), bei papildomos klasės, reikalingos panašumams apibrėžti: skaitmeninių reikšmių diapazonai, savybių bei atvejų panašumo matricos. Suprantama, norint panaudoti *CBR* metodus organizacijų duomenų bazėse, reikės papildomų duomenų – savybių ir produktų panašumo įvertinimų, savybių kitimų diapazonų ir panašiai. Tačiau *SQL* užklausas su panašumų skaičiavimu bus galima panaudoti įvairiose taikomosiose programose.

8. Išvados

Tyrimo darbų gausumas *CBR* srityje rodo, kad atvejais paremtų sprendimų paieškos sistemų kūrimo metodikos klausimai yra aktualūs, tokios sistemos gali žymiai pagerinti elektroninės komercijos ar informacinės paieškos (ypač didelės apimties) sistemas, pavyzdžiui, elektronines rinkas, portalus. Kadangi vartotojo paslaugų pagerinimas, jo „aptarnaujamumas" teigiamai veikia prekybinių organizacijų ekonominius rezultatus, elektroninės komercijos sistema gali pasiūlyti vartotojui papildomas jo profilį atitinkančias prekes ar paslaugas.. *CBR* gali išplėsti elektroninės prekybos sistemų funkcionalumą, suteikti joms intelektualių savybių.

Lyginant su ankstesnėmis *CBR* sistemomis, kurios buvo monolitinės, skirtos nedideliam individualių vartotojų ratui, šiuolaikinių atvejais paremtų sprendimų paieškos sistemoms nauja yra tai, kad jas reikia integruoti į

elektroninės komercijos serverio architektūrą ir suderinti su naujomis informacinėmis technologijomis bei standartais.



3 pav. Panašių atvejų paieškos pagal nurodytas savybes metamodelis UML

Praktikoje gali prireikti įvairių *CBR* metodų taikymo, pavyzdžiui, priklausomai nuo poreikių gali būti labiau tinkama tekstinė, dialoginė ar struktūrinė užklausų forma. Šiuo metu praktikoje labiau paplitusios tekstinės užklausos, kurios pasiteisina informacinėje paieškoje, knygų prekyboje, bet nėra labai tinkamos tuo atveju, kai vartotojas gali nurodyti tik pageidaujamo produkto tipo charakteristikas.

Šiame straipsnyje siūloma idėja išplėsti komercinių DBVS funkcionalumą, papildant jas galimybe vykdyti *SQL* užklausas su panašumų skaičiavimu. Išnagrinėti įvairūs panašumų skaičiavimo algoritmų variantai, pasiūlytas metamodelis, apibendrinantis atvejų atvaizdavimo variantus. Šiuo metu suformuluota ir išbandyta dalis algoritmų, kurių pagalba galima atlikti panašumų skaičiavimą, tačiau norint įgyvendinti šią idėją, reikia įdėti dar daug pastangų.

Literatūros sąrašas

[1] **D.A.Waterman**. Rukovodstvo po ekspertnym sistemam. *Moskva, "Mir"*, 1989, 390p.

[2] INRECA projects. 2000*, http://www.inreca.org/data/cbr/* .

[3] **G.Saward**. Publications 2000. ES200 Conference on Expert Systems, Cambridge, Springer Lecture Notes in AI – . 2000 *http://www.cs.herts.ac.uk/~comqgrs/papers.htm*

[4] **R.Bergman**. On the Use of Taxonomies for Representing Case Features and Local Similarity Measures.*University of Kaiserslautern Centre for Learning Systems and Applications (LSA),* 1997.

[5] **R.Bergman, A.Stahl.** Similarity Measures for Object–Oriented Case Representatioons. *University of Kaiserslautern Centre for Learning Systems and Applications (LSA),* 1997.

[6] **J.Schumacher, R.Bergmann.** Similarity–Based Retrieval on Top of Relational Databases. – *University of Kaiserslautern Centre for Learning Systems and Applications (LSA),* 1999.

Summary

There are a lot of shortcomings in rapid development of electronic commerce systems, one of them – limited functionality of services provided to user. The functionality could be improved by intellectual systems' applications in this area. The review of scientific researches shows that promising are case based reasoning systems. This paper analyzes the most similar cases retrieval with SQL queries.