KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Birutė Meilutytė

Programinių agentų kūrimo metodika naudojant DBVS priemones

Magistro darbas

Darbo vadovas
Prof. Dr. L. Nemuraitė

Kaunas, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Birutė Meilutytė

Programinių agentų kūrimo metodika naudojant DBVS priemones

Magistro darbas

Recenzentas
Doc. Dr. A. Lenkevičius
2007-05-24

Vadovas
Prof. Dr. L. Nemuraitė
2007-05-24

Atliko
IFM – 1/2 gr. stud.
Birutė Meilutytė
2007-05-24

Kaunas, 2007

Turinys

1. Įvadas

Per keletą paskutinių metų, informacijos kiekis, apdorojamas informacinėmis technologijomis internete kiekvieną dieną, išaugo eksponentiškai. Pagrindinė problema yra ta, kad didelė dalis informacijos yra išmėtyta skirtingose, dažnai tarpusavyje nesusijusiose saugyklose, todėl tampa gana sudėtinga rasti reikiamą informaciją.

Taigi, dėl padidėjusio informacijos šaltinių kiekio yra svarbu įdiegti galingas ir lengvai naudojamas informacines sistemas, kurios turi direktorijų tipo struktūras, palengvinančias informacijos paiešką. Būtent tokioje informacijos paieškoje labai praverčia žinių portalai, kuriuose galima rasti ne tik įvairią su ieškoma tema susijusią informaciją, bet ir kitų vartotojų atsiliepimus bei informacinio turinio vertinimus. Tačiau paprastai tokios sistemos reikalauja pastovios portalo administratoriaus priežiūros bei informacijos atnaujinimo, kad vartotojas visada lengvai rastų pačią naujausią ir aktualiausią informaciją.

Šio darbo tikslas yra padidinti informacinių sistemų dinamiškumą, lankstumą ir automatizavimo laipsnį bei palengvinti jų priežiūrą, naudojant konfigūruojamus įvykių apdorojimo mechanizmus, kurie leistų sistemos administratoriui nustatyti ir keisti automatiškai atliekamus veiksmus.

Tam reikia:

– išanalizuoti įvykių apdorojimo metodus;

– sudaryti metodiką, kuri leistų modifikuoti sistemos programinių agentų atliekamus veiksmus ar papildyti juos naujais;

– ištirti sudarytą metodiką, realizuojant interneto sistemos su save palaikančiomis funkcijomis prototipą.

Prototipu pasirinkta sistema, kurioje vartotojai gali patalpinti straipsnius, vertinti kitų vartotojų straipsnius, o sistema skaičiuoja straipsnių ir vartotojų reitingus, juos rūšiuoja, atrenka geriausių straipsnių sąrašą ir automatiškai patalpina straipsnius į archyvą, praėjus konkrečiam laikui. Naujausi straipsniai patalpinami į naujienų skiltį, o geriausiai lankytojų įvertinti straipsniai patalpinami į „top“ sąrašą. Taip pat sistemoje realizuota galimybė pasirinkti, kada turi būti atliekamas straipsnių reitingavimas: iškarto po naujo įvertinimo atsiradimo, ar vėliau, paties administratoriaus pasirinktu laiku. Tokioje sistemoje administratoriui nereikėtų rūpintis periodišku sistemos administravimu ir tai labai supaprastintų sistemos administravimo darbą, nes sistema būtų „save prižiūrinti“. Kitas siekiamas privalumas yra galimybė keisti ir papildyti sistemos atliekamas funkcijas, neperprogramuojant visos sistemos.

Taigi, pagrindinis darbo tikslas yra užtikrinti laiko įvykių generavimą ir sistemos reakciją į laiko bei kitus įvykius. Mokslinis darbo tikslas būtų sukurti bendresnį įvykių fiksavimo ir sistemos reakcijos į įvykius automatizavimo modelį, kurį būtų galima pritaikyti projektuojant panašias sistemas, ištirti tokio modelio įgyvendinimo galimybes pasirinktai programų platformai ir pateikti rekomendacijas ar net pakartotinio naudojimo komponentus, kuriuos būtų galima taikyti, kuriant panašias sistemas. Toks laiko įvykių valdymas daugiausiai naudojamas įterptinėse (angl. *embedded*) sistemose [14] [17], o interneto sistemose, kuriose vykdomi verslo procesai, tokių modelių ir sprendimų yra palyginus mažai ir jie atskirai nenagrinėjami projektavimo procesuose. Šiuo metu taip pat nėra paplitusių sprendimų kurti internetines sistemas su savaime atsinaujinančiu turiniu, o informacijos atnaujinimas paliekamas administratoriui. Galbūt ir egzistuoja analogiški sprendimai, tačiau jie yra tik firmų paslaptis ir informacija apie juos yra neprieinama.

Sistemos prototipas buvo realizuotas kaip nedidelis informacijos portalas, turintis pagrindines funkcijas ir savybes, reikalingas tokiam portalui funkcionuoti [11] [16]:

- vartotojų registracija/prisijungimas
- straipsnių patalpinimas į sistemą
- straipsnių komentavimas ir vertinimas
- sistemos valdymas iš administratoriaus zonos
- galimybė greitai ir paprastai surasti reikiamą informaciją
- saugumas
- pastovus portalo informacijos tikrinimas ir atnaujinimas
- greitis
- kiek įmanomas savarankiškesnis portalo funkcionavimas

Tokios nedidelės sistemos pilnai pakanka iliustruoti automatinio įvykių valdymo galimybėms. Kuriant sistemos prototipą, buvo nutarta naudoti agentus [2] [6] [12] [13], realizuotus duomenų bazėje [9]. Prototipo realizavimui buvo pasirinkta .NET C# aplinka ir MS SQL 2005 serveris. Microsoft technologija buvo pasirinkta dar ir todėl, kad su ja paprasta kurti internetines sistemas – pateikiama daugelis jau realizuotų komponentų, kuriuos vartotojai gali pritaikyti savo internetinėse sistemose ir tai labai palengviną tokių sistemų kūrimą. Tuo tarpu MS SQL 2005 server yra galingas duomenų bazių serveris, kuris buvo paskutiniu metu labai patobulintas, ir tapo daug patrauklesnis ir aiškesnis naudoti net ir gilių žinių neturinčiam vartotojui. Be to, viena iš naujų MS SQL 2005 server savybių yra ta, kad dabar į duomenų bazės vidines procedūras jau galima įtraukti ir .NET kalba parašytas funkcijas. Ši savybė labai išplėtė MS SQL server naudojimo galimybes. Dar vienas iš

Microsoft produkcijos privalumų yra tas, kad vartotojam į prieinama išsami ir plati pagalbos sistema, kurioje galima rasti atsakymus į beveik visus iškilusius klausimus.

Taigi, šiame darbe aprašyta metodika bei pateiktas pavyzdys turėtų suteikti galimybę panašius sistemos automatizavimo principus pritaikyti tiek didesnėse, tiek ir mažose informacinėse sistemose. Svarbu tai, kad siūloma metodika nėra sudėtinga ir nereikalauja gilių programavimo žinių. Darbe pateiktus principus savo turimose internetinėse svetainėse gali pritaikyti ir paprasti vartotojai, norėdami jas išplėsti ir padaryti kiek įmanoma labiau nepriklausomas.

Darbo struktūra

Darbą sudaro aštuoni skyriai, kurie aprašo atliktą darbą bei eksperimentą, paaiškina metodiką, siūlomą naudoti kuriant panašias sistemas, suteikia informaciją apie išorinius informacijos šaltinius, kurie praverstų tobulinant sistemą ar siūlomus principus pritaikant panašiose sistemose.

Pirmas skyrius – **Įvadas**. Skyriuje pateikiamas šio darbo aprašymas. Nusakomas tikslas, atlikti tyrimai ir darbai.

Antras skyrius - **Šiandieninių interneto portalų reikalavimų ir jų įgyvendinimo priemonių analizė.** Skyriuje apžvelgiami ir išskiriami pagrindiniai reikalavimai, keliami internetiniam portalam [11] [16] [18]. Taip pat pateikiama informacija apie tai, kas tai yra programiniai agentai [2] [4] [6] [12] [13]. Aprašomas agentų klasifikavimas, paaiškinama kodėl agentai nėra objektai, nurodant pagrindinius skirtumus. Taip pat paaiškinama, kokios sistemos vadinamos agentais paremtomis [10] [14] [15] [17]. Pateikiama informacija apie tai, kokiais būdais galima realizuoti agentus sistemose [3] [7] [8] ir paaiškinama, kodėl tokio tipo projektuose yra rekomenduojama naudoti agentus, realizuotus duomenų bazėse [9].

Trečias skyrius – **Interneto portalo programinių agentų modelis ir kūrimo procesas**. Skyriuje pateikiama informacija apie tai, kaip projektuoti ir įgyvendinti internetinius portalus su programiniais agentais. Skyriuje pateikiamas požiūris, kaip agentus projektuoti ir atvaizduoti naudojant standartines UML diagramas. Sudėtingiems projektam rekomenduojama naudoti AOR [5] modeliavimo principą, su kuriuo taip pat supažindinama šiame skyriuje.

Ketvirtas skyrius – **Eksperimentinė portalo su programiniais agentais realizacija**. Skyriuje aprašomas eksperimentas, kuriuo metu buvo sukurtas nedidelis informacinis portalas, kūrimo metu pritaikius siūlomą metodiką. Skyriuje pateikiamas projekto aprašymas, jam keliami reikalavimai. Pateikiamos visos pagrindinės diagramos, nusakančios projekto realizaciją bei veikimą.

Penktas skyrius – **Metodikos tyrimas**. Šiame skyriuje aprašoma siūloma metodika. Taip pat aprašomas taikymo galimybių tyrimas, kūrimo ir modifikavimo galimybių tyrimas bei įvertinama pati metodika, pabrėžiant kokio tipo projektams ji rekomenduojama ir šios metodikos privalumus.

Šeštas skyrius – **Išvados**. Skyriuje pateikiamos pagrindinės atlikto tyrimo ir darbo išvados.

Septintas skyrius – **Terminų ir santraukų žodynas**. Skyriuje aprašomi ir paaiškinami pagrindiniai terminai ir santraukos, naudojamos šiame darbe.

Septintas skyrius – **Literatūros sąrašas**. Skyriuje pateikiamos nuorodos į išorinius šaltinius, kurie buvo naudojami rašant šį darbą.

2. Šiandieninių interneto portalų reikalavimų ir jų įgyvendinimo priemonių analizė

2.1. Reikalavimai interneto portalams

Per keletą paskutinių metų, informacijos kiekis, apdorojamas informacinėmis technologijomis internete kiekvieną dieną, išaugo eksponentiškai. Pagrindinė problema yra ta, kad didelė dalis informacijos yra išmėtyta skirtingose, dažnai tarpusavyje nesusijusiose saugyklose, todėl tampa gana sudėtinga rasti reikiamą informaciją. Taigi, dėl padidėjusio informacijos šaltinių kiekio yra svarbu įdiegti galingas ir lengvai naudojamas informacines sistemas, kurios turi direktorijų tipo struktūras, palengvinančias informacijos paiešką. Būtent tokioje informacijos paieškoje labai praverčia žinių portalai, kuriuose galima rasti ne tik įvairią su ieškoma tema susijusią informaciją, bet ir kitų vartotojų atsiliepimus bei informacinio turinio vertinimus. Taigi, dėl padidėjusio informacijos šaltinių kiekio yra svarbu įdiegti galingas, bet vis dar lengvai naudojamas informacines sistemas, tokias, kurios turi direktorijų tipo struktūras, palengvinančias informacijos paieškas. Būtent tokioje informacijos paieškoje mums labai praverčia žinių portalai, kuriuose galima rasti ne tik įvairius, su ieškoma tema susijusius straipsnius, bet ir kitų vartotojų atsiliepimus bei straipsnių vertinimus.

Tačiau paprastai tokios sistemos reikalauja pastovios portalo administratoriaus priežiūros bei informacijos atnaujinimo, kad vartotojas visada lengvai rastų pačią naujausią ir aktualiausią informaciją.

Taigi, portalas – tai informacinė sistema, į kurią informacija ateina iš daugelio šaltinių. Tam kad internetiniai portalai būtų naudingi jų vartotojam, jie turi tenkinti keletą pagrindinių reikalavimų:

- Portalo patikrinimai ir atnaujinimai turėtų vykti pastoviai. Patikrinimai turėtų užtikrinti, kad visa pateikta informacija yra nauja ir teisinga. Kad naujausia informacija yra prieinama vartotojam. Kad išorinės nuorodos yra teisingos. Nuorodos į kitus puslapius turėtų būti naudojamos tik tada, jei jos yra tikrai reikalingos. Portalų kūrėjai turėtų sukurti įrangą ir įrankius, leidžiančius portalo prižiūrėtojui atlikti šias funkcijas greitai ir tiksliai. Dar geriau būtų, jei už daugelį šių veiksmų būtų atsakingas pats portalas – t.y. sistema būtų pati „save prižiūrinti“ [16].
- Portalai turėtų būti padaryti taip, kad vartotojas greitai galėtų surasti reikiamą informaciją. Ekranų ir žingsnių skaičius, reikalingas pasiekti informaciją, turi būti minimalus. Kai vartotojas prisijungia, didžioji dalis informacijos turėtų būti pasiekiama

ne daugiau kaip per 3 žingsnius. Taip pat kiekvienas puslapis turėtų parodyti savo vietą bendroje portalo hierarchijoje [16].

- Portalas paprastai turi turėti keletą informacijos kategorijų, kuriose yra nuorodos į dokumentus, paslaugas ir kitas funkcijas. Kuomet kuriamos kategorijos, reikėtų atsižvelgti į tai, jog nuorodas rekomenduojama išdėstyti abėcėlės tvarka arba priklausomai nuo jų naudojimo. Jei nuorodos išdėstomos abėcėlės tvarka, paieškos laikas sumažėja 50%. Svarbiausios ir naudingiausios nuorodos turėtų būti lengvai matomos ir aiškiai išsiskirti iš kitų [11].
- Tapatybės nustatymas yra svarbi dalis portalo įgyvendinime. Tai ne tik garantuoja, jog portale pasisakys bei straipsnius rašys patikimi vartotojai, tačiau registruotus vartotojus taip pat išskirs ir bendros minios, suteikiant jiems papildomas galimybes ir teises. Tačiau būtų nelogiška portalais naudotis leisti tik registruotus vartotojus. Daug geriau minimalias galimybes suteikti ir neregistruotiem vartotojam, leidžiant jiem tik dalinai naudotis esama informacija. Norėdami gauti pilnas portalo vartotojo teises jie būtinai užsiregistruos. Iš kitos pusės, sistema turi bent kažkiek žinoti apie savo vartotojus, nes tai naudinga siekiant užtikrinti naudingos informacijos pateikimą. Vartotojai turi daug svarbių charakteristikų, be savo vardo ar gyvenamosios vietos. Tai ir jų išsilavinimas, užimamos pareigos ir t.t. Tai, ką vartotojas apie save parašė, gali būti labai vertinga, siekiant patenkinti vartotojų informacijos poreikius. Taip pat galima atlikti vartotojų apklausas, siekiant išsiaiškinti pagrindinius portalo trūkumus ar vartotojų pageidavimus, kas leistų portalą padaryti dar labiau atitinkantį vartotojų poreikius. Tačiau nereikia pamiršti, jog visa informacija, gauta apie vartotojus turi būti saugoma ir viešai neplatinama [11].

Tam, kad būtų galima realizuoti tokius portalus, nereikalaujančius pastovios administratoriaus priežiūros, patariama naudoti Web 2 technologiją. Nuo tada, kai 2004 metais O‘Reilly prabilo apie Web 2 ir suorganizavo daugybę konferencijų šia tema, Web 2 sulaukia vis daugiau ir daugiau vartotojų ir programinės įrangos kūrėjų dėmesio. Web 2 – tai web-paremtų paslaugų antroji karta, suteikianti galimybę geriau realizuoti socialinio bendravimo tinklapius (angl. *Social networking sites*), wiki tipo tinklapius (wiki – portalas, kuriame vartotojai gali patys keisti ir redaguoti portalo turinį), įvairius bendravimo įrankius (angl. *Communication tools*) bei folksonomijas (angl. *Folksonomy*) (folksonomija – tai vartotojų sudaryta taksonomija, skirta rūšiuoti Web puslapius, nuorodas bei turinį naudojant žymas) [18]. 2006 Tim O‘Reilly Web 2 apibūdino taip: Web 2 yra verslo revoliucija kompiuterių pramonėje, kuri atsirado dėl to, kad internetas vis labiau panašėja į

platformą bei bandoma suprasti taisykles, kurios garantuotų sėkmingą šios platformos veikimą. Pagrindinė iš šių taisyklių yra: kurkite programas išnaudojančias visus tinklo veikimo elementus, leidžiančius jūsų programoms tapti tuo geresnėmis, kuo daugiau vartotojų jomis naudojasi (angl. *Web 2.0 is the business revolution in the computer industry caused by the move to the internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: Build applications that harness network effects to get better the more people use them*).

Tim O'Reilly taip pat pateikė kompanijų bei produktų pavyzdžius, kurie naudoja Web 2 technologijos principus. Tim O'Reilly apibrėžė „keturis plius vienas" (angl. *Four plus one*) lygius Web 2 naudojamumo hierarchijoje [18]:

- 3 lygio programos gali egzistuoti tik internete. Šios programos įgyja savo galias (angl. *power*) dėl vartotojų bendradarbiavimo, o jų naudingumas didėja sulig kiekvienu nauju vartotoju. Tai tokios programos kaip eBay, craigslist, Wikipedia, del.icio.us, Skype, dodgeball bei Adsense.
- 2 lygio programos gali veikti ir be interneto (angl. *Offline*), tačiau jos prisijungus prie interneto, jos įgyja privalumų. Šiam lygiui O'Reilly priskiria Flickr, kurio pranašumas yra paviešinta (angl. *Shared*) nuotraukų duomenų bazė bei vartotojų sugeneruota žymų (angl. *Tag*) duomenų bazė.
- 1 lygio programos taip pat gali veikti be interneto, tačiau prisijungus prie interneto jos įgyja pranašumų. Šiai kategorijai O'Reilly priskiria Writely (nuo 2006-10-10: Google Docs & Spreadsheets, suteikiantį galimybę grupiniam redagavimui internete) ir iTunes (dėl jo sukaupto muzikos kiekio)
- 0 lygio programos taip pat gali dirbti ir be interneto. O'Reilly šiai kategorijai priskiria MapQuest Yahoo! Local ir Google Maps. Žemėlapių programos, kuriose yra vartotojų indėlis, gali būti priskiriamos ir antrai kategorijai.
- Ne Web programos, tokios kaip el.paštas, susirašinėjimo programų klientai bei telefonas.

2.2. Programinių agentų modelių apžvalga

Paprastai kuriamos pasyvios informacinės sistemos, kuriose veiksmus inicijuoja sistemos vartotojai arba informacinę sistemą aptarnaujantis personalas. Tačiau dabar jau atsirado galimybė kurti naujos kartos sistemas, kurios yra aktyvios. Tai reiškia, jog pati sistema seka situacijas, valdo

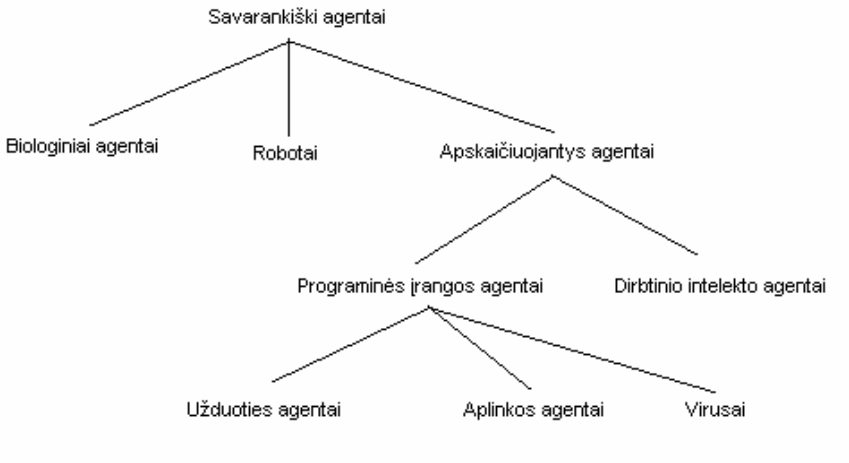
įvykius, inicijuoja reikiamus veiksmus bei informuoja vartotojus apie esamą situaciją ar net apie galimus ar rekomenduojamus veiksmus.

Tokios informacinės sistemos gali būti įgyvendinamos naudojant agentines technologijas. Vienas svarbiausių šių agentinių sistemų skirtumų nuo tradicinių yra jų autonomiškumas. Agentinės sistemos sprendžia apibrėžtus uždavinius ir siekia savo tikslų be nurodymų ir komandų iš aplinkos. Tačiau IS vartotojai turi turėti galimybę specifikuoti ar keisti agento savarankiškumą. Gali būti kuriami ir visiškai autonomiški, ir valdomi agentai [2].

2.2.1. Agentų klasifikavimas

Agentas yra apibrėžiamas kaip esybė, kuri veikia savarankiškai, vykdydama sau ir/arba kitoms esybėms reikalingas užduotis. Stan Franklin bei Art Graesser [13] agentus apibūdina taip: savarankiškai veikiantis agentas - tai sistema, įdiegta kitoje sistemoje ir esantis tos sistemos dalis, kuri jaučia sistemą ir veikia joje priklausomai nuo sistemos būsenos. (angl. *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future*)

Yra keletas požiūrių, kaip galima būtų klasifikuoti agentus. Stan Franklin bei Art Graesser savo darbe [13] agentus klasifikuoja naudojantis biologiniu modeliu. Toks modelis yra atvaizduojamas kaip medis, kurio viršūnėje savarankiškas agentas, o žemesniuose lygiuosi galimos jo klasifikacijos. Stan Franklin bei Art Graesser sudarytas agentų klasifikavimo modelis pavaizduotas 1 pav.

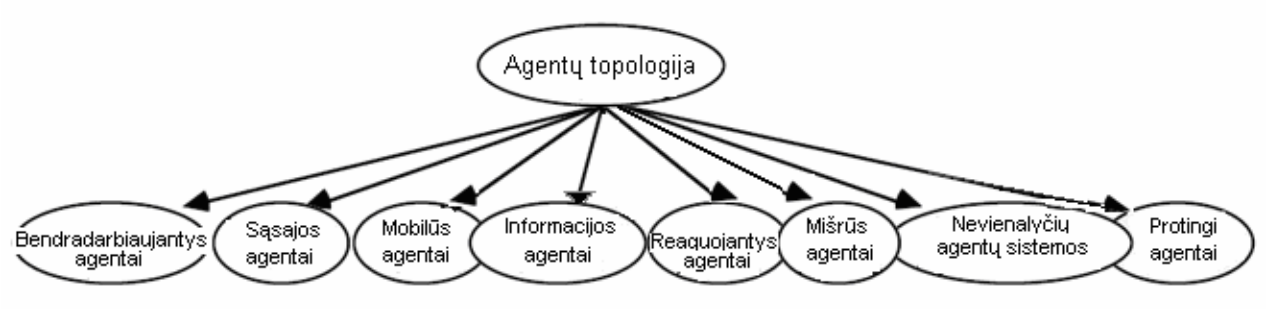


1 pav. Stan Franklin bei Art Graesser siūlomas agentų klasifikavimo modelis

Remiantis šiuo modeliu, savarankiškai veikiančius agentus galima skirstyti į biologinius agentus, robotus bei apskaičiuojančius agentus. Apskaičiuojantys agentai savo ruoštu gali būti suskirstyti į programinės įrangos agentus bei dirbtinio intelekto agentus (angl. *Artificial Life*

Agents). Programinės įrangos agentus galima detalizuoti į užduoties agentus (angl. *Task-specific Agents*), aplinkos agentus bei virusus. Stan Franklin bei Art Graesser nuomone, šis modelis apima pagrindines agentų klasifikacijas. Jei reikėtų agentus detalizuoti dar labiau, jie siūlo agentus klasifikuoti remiantis valdymo struktūromis (angl. *Control structures*), aplinkomis (duomenų bazė, failų sistema, tinklas, internetas) arba programavimo kalbomis, kuriomis jie yra parašyti.

Hyacinth S.Nwana savo darbe [4] daug labiau detalizuoja programinės įrangos agentus. Šiame darbe programinės įrangos agentai yra skirstomi į 8 tipus, kaip kad pavaizduota 2 pav.



2 pav. Hyacinth S.Nwana siūloma programinės įrangos agentų klasifikacija

- Bendradarbiaujantys agentai (angl. *Collaborative Agents*). Bendradarbiaujantys agentai siekdami tikslo veikia savarankiškai bei bendradarbiauja su kitais agentais. Šie agentai taip pat gali ir apsimokyti vykdydami užduotis, tačiau tai nėra viena iš jiems būdingų savybių. Bendradarbiaujantys agentai paprastai naudojami spręsti problemas, kurioms vieno agento jau nebeužtenka dėl sistemos dydžio ar sudėtingumo. Taip pat bendradarbiaujantys agentai naudojami paskirstytose sistemose.
- Sąsajos agentai (angl. *Interface Agents*). Sąsajos agentų tikslas yra bendradarbiauti su sistemos vartotoju ir jam asistuoti. Siekdami savo tikslo šie agentai veikia autonomiškai ir nuolat apsimoko. Tačiau reikėtų nepamiršti, kad bendradarbiavimas su sistemos vartotoju nėra bendradarbiavimas su kitu agentu ir nemaišyti šių agentų su bendradarbiaujančiais agentais. Kaip jau buvo minėta, pagrindinis šių agentų tikslas yra pagelbėti vartotojui, kuris tik pradėjo naudotis ar mokosi naudotis programine įranga. Sąsajos agentai stebi bei įsimena vartotojo atliekamus veiksmus, o vėliau pataria vartotojui kaip tam tikrus veiksmus atlikti paprasčiau ir greičiau.
- Mobilūs agentai (angl. *Mobile Agents*). Mobilūs agentai – tai programinės įrangos procesai, kurie keliauja tinklu, pavyzdžiui WWW, renka informaciją atitinkančią vartotojo pateiktą užduotį bei praneša vartotojui rezultatus. Mobilumas nėra svarbi ar esminė sąlyga būti agentu, tačiau šie procesai priskiriami prie agentų todėl, kad jie veikia

savarankiškai bei bendradarbiauja su kitomis sistemomis. Aišku, šis bendradarbiavimas skiriasi nuo bendradarbiaujančių agentų bendradarbiavimo. Mobilūs agentai gali keistis informacija su kitais agentais, tačiau jie neprivalo perduoti visos turimos informacijos, o naudoja tik minimalų jos kiekį, reikalingą gauti kitai informacijai.

- Informacijos agentai (angl. *Information Agents*). Informacijos agentai kartais dar vadinami interneto agentais. Šių agentų pagrindinis tikslas yra informacijos, esančios daugelyje paskirstytų resursų, paieška, valdymas bei palyginimas.
- Reaguojantys agentai (angl. *Reactive Agents*). Reaguojantys agentai yra atskira agentų grupė, kuri nevaldo savo aplinkos vidinių simboliškų modelių (angl. *Symbolic models*), tačiau jie reaguoja į sistemos, kurioje yra įdiegti, būsenas bei įvykius. Reikėtų pabrėžti, kad reaguojantys agentai yra santykinai nesudėtingi (angl. *Relatively simple*) ir su kitais agentais bendrauja elementariais metodais. Tačiau jei bandoma į šių agentų atliekamo darbo modelį pažvelgti globaliai, visas procesas gali atrodyti sudėtingas.
- Mišrūs agentai (angl. *Hybrid Agents*). Kiekvienas iš čia apžvelgtų agentų (bendradarbiaujantys, sąsajos, mobilūs, informacijos bei reaguojantys agentai) turi savo stipriąsias bei silpnąsias vietas ir tai lemia kada patogiau naudoti vieną ar kitą agentų tipą. Tačiau kartais realizuojant sistemą reikalingos savybės, kurios priklauso keletui skirtingų agentų. Tokiais atvejais galima bandyti apjungti keleto skirtingų agentų savybes bei realizavimo metodus tokiu būti realizuojant mišrius agentus.
- Nevienalyčių agentų sistemos (angl. *Heterogeneous Agents Systems*). Nevienalyčių agentų sistemos palaiko jose įdiegtus du ar daugiau skirtingų agentų. Šiose sistemose taip pat gali būti įdiegti ir mišrūs agentai.
- Protingi agentai (angl. *Smart Agents*). Šie agentai veikia autonomiškai, bendradarbiauja su kitais agentais bei nuolat apsimoko. Tai pats sudėtingiausias agentų tipas, kuris šiuo metu dar nėra pilnai realizuotas ir šioje srityje nuolat vyksta tyrimai.

2.2.2. Agentų ir objektų panašumai ir skirtumai

Agentai ir objektai tai tas pats ar jie skiriasi? Šis svarbus klausimas analizuojamas J.Odell darbe [6]. Dalis programuotojų agentus ir objektus laiko tuo pačiu. Kita dalis programuotojų agentus ir objektus laiko skirtingais dalykais, nepaisant to, kad jie turi daug bendro tarpusavy. Kaip bebūtų, tiek agentai, tiek ir objektai yra naudojami kartu kuriant agentines sistemas.

Vienas iš pagrindinių skirtumų tarp objektų ir agentų yra tai, kad agentai yra savarankiški. Kitaip sakant, kada ir kaip veikti nusprendžia pats agentas. Agentai gali ne tik reaguoti į tam tikrus metodų veiksmus, bet ir stebėti aplinka bei reaguoti į jos pokyčius. Tuo tarpu objektai yra pasyvūs. Objektas pradės veikti tik tuomet, kai jis bus iššaukiamas. Žinoma, nereikėtų pamiršti, kad UML ir Java jau pristatė įvykių-klausymo karkasus (angl. *Event-listening framework*) bei kitas priemones leidžiančias objektam būti bent šiek tiek aktyviems. Taigi, galima teigti, jog objektai šiuo metu turi dalį autonomiškumo, būdingo agentams.

Agentai yra interaktyvūs. Tai reiškia, kad agentai gali bendradarbiauti tiek su aplinka, tiek ir su kitais agentais. Agentai tarpusavyje bendrauja agentų bendravimo kalba - ACL (Agent Communication Language), kuri yra formali ir nedviprasmiška, o jos formatas ir turinys gali būti labai įvairus. Žinoma, objektai taip pat gali bendrauti vieni su kitais, tačiau vienas objektas paprastai bendrauja tik su vienu objektu, o jų siunčiamos žinutės yra tam tikros struktūros: tai parametrai, kurių eiliškumas, kiekis ir tipai yra griežtai apibrėžti iš anksto. Be to, agentai dažnai naudoja objektus kaip tarpinius elementus savo bendravime.

Objektai sistemose yra centralizuotai organizuoti, nes objektų metodus sužadina kiti sistemos komponentai. Tuo tarpu agentų aplinkos gali būti centralizuotos bei paskirstytos. Agentai puikiai susidoroja su centralizuotomis sistemomis, tačiau jie veikia ir paskirstytose sistemose atliekamuose skaičiavimuose.

OO (Object Oriented) kalbose objektai sukuriami naudojant klases ir po jų sukūrimo jie nebegali būti keičiami ar tapti atskira dalimi keleto klasių (išskyrus paveldėjimą). Tuo tarpu agentai yra daug lankstesni. Agentai gali atlikti keletą vaidmenų. Kurį vaidmenį agentas atlieka tuo metu apibrėžia tik skirtingas agento savybių rinkinys.

Visos objekto savybės yra apibrėžiamos objekto klasėje. Po to kai objektas jau sukurtas, jis nebegali keisti savo savybių. Tuo tarpu agentas gali keisti savo savybes. Pavyzdžiui jei agentas gali apsimokyti, priklausomai nuo to jis vėliau gali pakeisti savo veikimą. Jei agentas gali save keisti, jis gali dinamiškai pridėti sau savybių. Šios agentų savybės labai svarbios agentinėse sistemose žinomose dirbtinio intelekto pavadinimu.

Tiek agentai, tiek ir objektai gali būti aprašomi kaip nežymūs (angl. *Slim*) arba įtakingi (angl. *Fat*), smulkiai suskaidyti (angl. *Small grained*) arba stambiai suskaidyti (angl. *Large grained*). Taip pat didelėse sistemose sudarytose iš daugybės agentų ir objektų, kiekvienas jų gali būti vadinamas mažu palyginus su visa sistema. Tačiau, individualus agentas turi mažiau įtakos visai sistemai nei objektas. Jei sistemoje dingsta agentas, daugeliu atveju sistema ir toliau gali funkcionuoti. Tačiau jei sistemoje prarandamas objektas, sistemoje atsiranda išimtis (angl. *Exception is rised*)

Įvertinus agentų ir objektų panašumus ir skirtumus, galima teigti, kad naudinga kuriant sistemas pritaikyti abu šiuos būdus. Netgi galima kurti agregatus, kuri susideda iš objektų ir kitų agentų, bei atvirkščiai. Naudoti agentus objektinėse sistemose yra naudinga, nes [6]:

- Tai leidžia realizuoti stipriai paskirstytas sistemas,
- Suteikia galimybę kitaip pažvelgi į ir realizuoti statinę architektūrą,
- Susistemina metodus kaip realizuoti tuo pačiu metu bendradarbiaujančius objektus.

2.2.3. Agentais paremtos sistemos

Agentais paremtomis sistemomis vadinamos sistemos, kuriose vienas iš akcentų yra veikiantis agentas. Tokios sistemos paprastai turi vieną ar kelis jose veikiančius agentus, tačiau didesnes galimybes turi sistemos, kuriose veikia keli agentai. Tokios sistemos pasižymi žemiau pateiktomis savybėmis :

- Savarankiškumas (angl. *Autonomy)*: agentai apima tam tikrą sistemos būseną ir priima sprendimus kaip elgtis priklausomai nuo šios būsenos. Agentų apsisprendimą veikti lemia tik sistemos būsena be papildomo žmonių ar kitų įsikišimo.
- Reaktyvumas (angl. *Reactivity*): agentai yra *įdiegti aplinkoje* kurią *gali suprasti* bei savalaikiškai reaguoti į bet kokius šios sistemos pasikeitimus.
- Veiklumas (angl. *Pro-activeness*): agentai ne paprasčiausiai reaguoja į aplinkos pokyčius, o imasi iniciatyvos, kad išskirti aplinkos elgesį, kuris turi tikslą.
- Bendradarbiavimas (angl. *Social ability*): agentai bendradarbiauja su kitais agentais (kartais ir su žmonėmis) naudodami tam tikros rūšies agentų bendradarbiavimo kalbą ir šis bendradarbiavimas jiem padeda pasiekti tikslus.

Kuriant komponentines verslo, informacines ir programų sistemas paskutiniu metu yra rekomenduojama naudoti būtent tokias agentines technologijas. Informacinėse sistemose agentinės technologijos paprastai naudojamos informacijos paieškai, duomenų analizei bei naujų žinių formavimui.

Kaip jau buvo minėta, vienas svarbiausių agentinių technologijų skirtumas nuo tradicinių sistemų yra jų autonomiškumas. Agentas savarankiškai vykdo užduotis, be nurodymų ir komandų iš aplinkos. Aišku, vartotojas taip pat gali duoti komandas agentams, į kurias pastarieji atsižvelgs vykdydami savo užduotis [2].

Agentais paremtose sistemose agentų aplinka paprastai būna nevienalytė, paskirstyta arba kintanti [12]. Priklausomai nuo aplinkos, kurioje turi veikti agentas, pasirenkamas ir sistemos

realizavimo metodas. Paprastai įvykiais paremtas sistemas galima suskirstyti į įterptines sistemas (angl. *Embedded systems*) bei įvykiais pagrįstas sistemas (angl. *Event-based systems*).

Įterptinės sistemos. Remiantis Wikipedia [17], įterptinė sistema tai specialios paskirties sistema, kurioje kompiuteris yra visiškai įdiegtas į sistemą kurią jis valdo. Tokios įterptinės sistemos paprastai atlieka vieną ar keletą iš anksto apibrėžtų užduočių, kurioms keliami griežti reikalavimai.

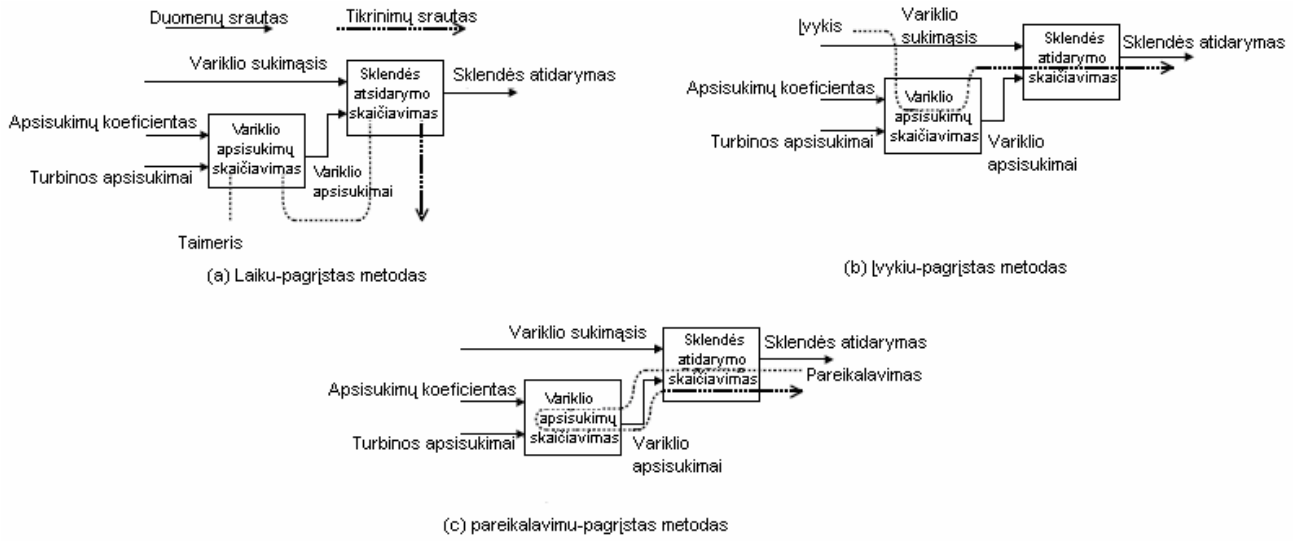
Dauguma įterptinių sistemų yra sudėtingos realaus laiko sistemos. Takanori Yokoyama savo darbe [14] išskiria dvi architektūras, pagal kurias yra realizuojamos tokios realaus laiko sistemos:

- Įvykio-sužadinimo architektūra (angl. *Event-triggered architekture*). Tokios architektūros sistema yra sudaryta iš modulių, kurie reaguoja į išorines įvestis dinamiškai ir iš karto.
- Laiko-sužadinimo architektūra (angl. *Time-triggered architekture*). Tokios architektūros sistema yra sudaryta iš modulių, kurie reaguoja į išorines įvestis ciklišk ai.

Įterptinėse sistemose egzistuoja trys pagrindiniai įvykių sužadinimo metodai [14]:

- laiku-pagrįstas (angl. *time-based*) įvykių sužadinimas (angl. *triggering*),
- įvykiu-pagrįstas (angl. *event-based*) įvykių sužadinimas
- pareikalavimu-pagrįstas (angl. *demand-based*) įvykių sužadinimas.

Laiku-pagrįstas metodas aktyvuoja operaciją ciklišk ai, įvykiu-pagrįstas metodas aktyvuoja operaciją priklausomai nuo išorinės įvesties, o pareikalavimu-pagrįstas metodas aktyvuoja operaciją, kai gauna tam nurodymą iš kitų objektų. 3 pav. pateikiamos diagramos, kuriose vaizduojama įvykių sužadinimas kiekvienu iš trijų būdų:



3 pav. Įvykių sužadinimas

Kaip matome iš aukščiau pateikto paveikslo, (a) atveju vykdymą sužadina taimeris, kuris nepriklauso nuo išorinių įvesčių. (b) atveju vykdymą sužadina tam tikra išorinė įvestis. (c) atveju vykdymą sužadina rezultatas, kurį gavo Throttle Opening Callculation blokas, priklausomai nuo į jį paduotų duomenų. Pareikalavimu-pagrįstas apdorojimas gali būti naudojamas tiek įvykio-sužadinimo architektūroj, tiek ir laiko-sužadinimo architektūroj. Nors populiariausia yra įvykio-sužadinimo architektūra, dažnai šios dvi architektūros, realizuojant programą, yra apjungiamos į mišrią architektūrą ir, priklausomai nuo situacijos, naudojamas reikiamas įvykių sužadinimo metodas. [14]

Įvykiais pagrįstos (angl. *Event-based*) sistemos. Per keletą paskutinių metų, įvykiais pagrįstos sistemos buvo pradėtos naudoti daugelyje sričių, tokių kaip įmonių valdymo sistemos, didelių kiekių duomenų platinimo sistemos, internete pritaikomos programos ar automatiniai skaičiavimai (angl. *Autonomic computing*). Įvykiais pagrįstos technologijos leidžia efektyviai konstruoti tokias sistemas bei valdyti komponentų bendravimą. [10].

Šiuo metu egzistuoja keletas įvykiais pagrįstų sistemų standartų, tokių kaip CORBA Notification Service, Data Distribution Service, bei Java Message SErvice. Nors šie standartai ir sulaukė dėmesio bei buvo naudojami, tačiau jie netapo paplitusiu ir vieningu standartu [10].

Įvykiais pagrįstos sistemos tai tokios sistemos, kuriose sudėtiniai moduliai (ar komponentai) neturi tiesioginio ryšio vieni su kitais. Tai reiškia, kad vienas metodas niekada nebandys iškviesti kito komponento metodo, nes vieni komponentai paprasčiausiai nežino apie kitų komponentų egzistavimą. [15]. Komponentai suaktyvinami naudojant įvykius arba žinutes.

2.3. Programinių agentų realizavimo technologijos pasirinkimas

Realizuojant įvykiais pagrįstas sistemas, įvykių apdorojimas paprastai vykdomas arba naudojant objektiškai orientuotas kalbas, arba aktyvias duomenų bazes. Jei kuriama sudėtinga paskirstyta sistema, įvykių apdorojimą rekomenduojama atlikti naudojant specialiai tam pritaikytas platformas bei architektūros karkasus.

2.3.1. Agentų realizavimas objektinio programavimo kalbomis

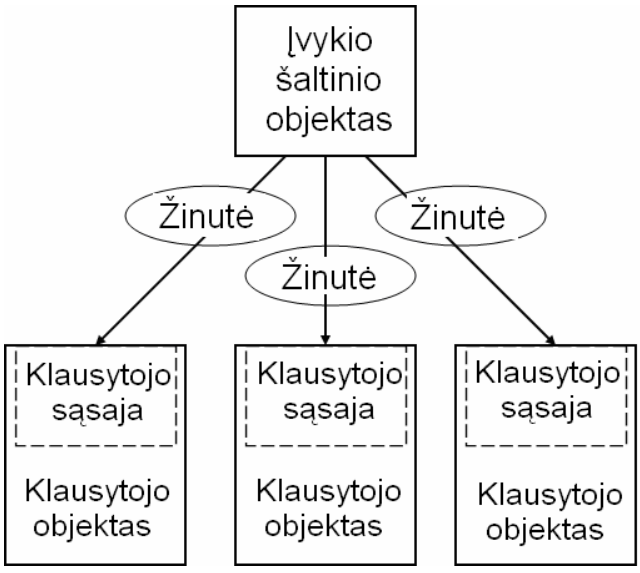
Šiuo metu egzistuoja daugybė programavimo kalbų bei platformų, skirtų kurti sistemas, pagrįstas agentais. Pavyzdžiui, IBM siūlo ABLE (Agent Building and Learning Environment). Siūlomos programavimo kalbos ar platformos paprastai būna skirtos kurti programinę įranga Java arba C kalbomis. Jei kuriama sistema nėra sudėtinga, galima naudoti objektiškai orientuotose

programavimo kalbose esančias specialiai tam pritaikytas klases. Tačiau jei sistema sudėtinga, tuomet geriau naudoti specialiai tam pritaikytas platformas bei architektūros karkasus.

Agentų valdymas naudojant Java programavimo kalbą [7]. Įvykių apdorojimas naudojant 1.0 Java platformos versiją yra pagrįstas paveldėjimu. Siekiant, kad programa pagautų GUI komponentus, ji turi būti poklasis GUI komponentų ir perrašyti arba action() arba handleEvent() metodus. Jei vienas iš šių metodų grąžina teigiamą rezultatą (angl.*" True"*), įvykis nėra toliau apdorojamas, priešingu atveju įvykis kyla aukštyn GUI hierarchija iki būna įvykdytas arba pasiekia hierarchijos viršų. Naudojant tokį įvykių gaudymo modelį, programa iš esmės turi tik 2 įvykių-apdorojimo struktūrizavimo būdus:

- Kiekvienas individualus komponentas gali būti poklasis ir apdoroti savo tam tikro komponento įvykius. Tačiau šis sprendimas veda į klasių perteklių.
- Visi įvykiai visai hierarchijai gali būti apdorojami tam tikrame konteineryje (angl. *Container*), kuris perrašo action() arba handleEvent() metodus. Tačiau tokiu atveju perrašytuose metoduose turi būti sudėtingi sąlygų sakiniai, kurie užtikrintų teisingą įvykių apdorojimą.

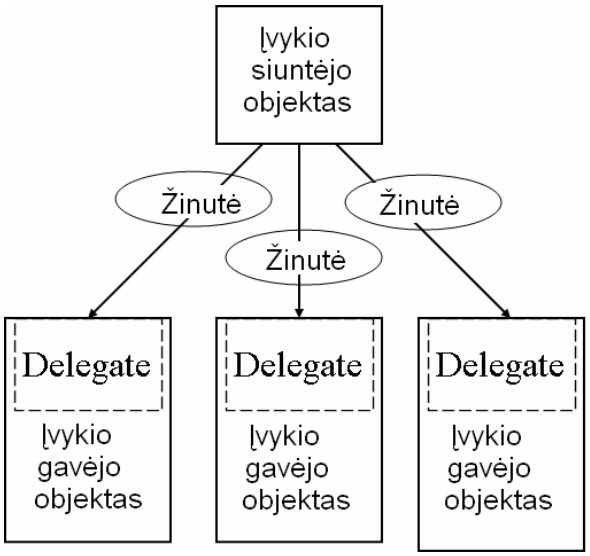
Žinoma, toks metodas veikiai mažuose apletuose su paprastomis sąsajomis, tačiau jis nėra tinkamas sudėtingesnėm Java programom. Siekiant suteikti galimybę kurti sudėtingas Java programas, 1.1 Java platformos versijoje buvo realizuotas Deleguoto įvykio modelis (angl. *Delegation event Model*). Modelio veikimo principas pavaizduotas 4 pav.



4 pav. Deleguoto įvykio modelio veikimo principas

Šiame modelyje įvykių tipai yra įdiegti klasių hierarchijoje, kurios viršus yra java.util.EventObject klasė. Įvykiai yra perduodami iš „šaltinio“ (angl. *Source*) objekto „klausytojo“ (angl. *Listener*) objektui. „Klausytojas“ yra objektas, kuriame realizuota specifinė EventListener sąsaja, kuris išplečia pagrindinį java.util.EventListener. EventListener sąsaja nusako vieną ar kelis metodus, kuriuos sužadins „įvykio šaltinis“, reaguodamas į įvykių tipus, prižiūrimus sąsajos. „Įvykio šaltinis“ (angl. *Event Source*) yra objektas, kuris sužadina arba „nugesina“ įvykius. Šis „šaltinis“ taip pat apibrėžia eilę įvykių, kuriems gali būti sukurti „klausytojai“. Paprastai „įvykių šaltinis“ yra GUI, tačiau juo gali būti ir kitas Java AWT (AWT – Java programavimo kalbos klasių biblioteka, tiekianti vartotojo sąsajos įrankį ir vadinama Abstract Windowing Toolkit) komponentas.

Agentų valdymas naudojant C# programavimo kalbą [8]. Įvykių apdorojimo modelis C# kalboje žinomas Delegate pavadinimu. Čia objektas, kuris sukelia įvykį vadinamas „įvykio siuntėju“ (angl. *Event sender*), o objektas kuris pagauna įvykį vadinamas „įvykio gavėju“ (angl. *Event receiver*). Įvykių ir Delegates veikimo principas pavaizduotas 5 pav.



5 pav. Įvykių ir Delegate principas

Įvykių bendravime, „įvykio siuntėjo“ klasė nežino kuris objektas ar metodas pagaus sukeltą įvykį. Todėl yra reikalingas tarpininkas tarp „įvykio siuntėjo“ ir „įvykio gavėjo“. .NET Framework šis tarpininkas žinomas Delegate pavadinimu. Delegate yra klasė, kuri saugo nuorodą į metodą. Priešingai nei kitos klasės, ši klasė turi savo parašą (angl. *Signature*) ir saugo nuorodas tik į tuos metodus, kurie atitinką jos parašą. Kad apibrėžti Delegate klasę, pakanka ją paskelbti. Būtent paskelbimo metu suteikiamas Delegate klasei suteikiamas parašas. Įdomi ir naudinga Delegate savybė yra ta, kad jis nieko nežino apie klasę, į kurios objektą saugo nuorodą. Vienintelis dalykas

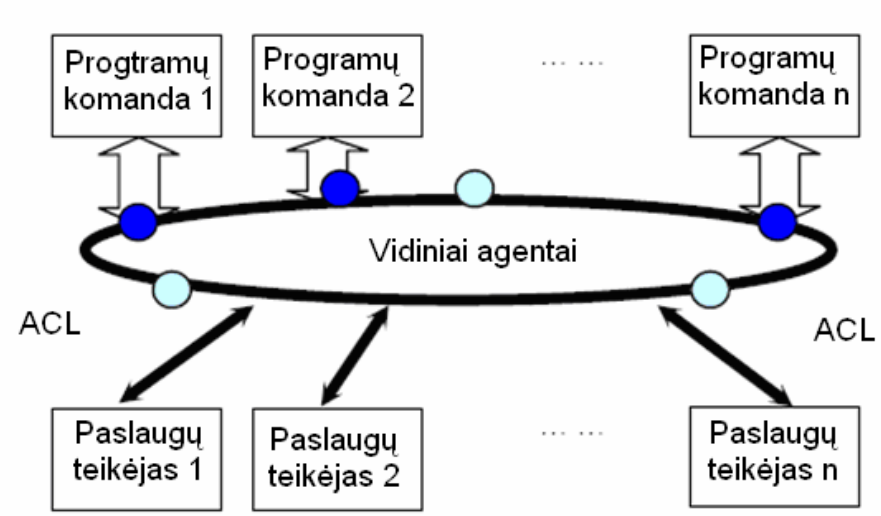
kuris rūpi Delegate klasei, yra kad metodo argumentų tipai bei grąžinama reikšmė atitiktų Delegate laukiamus. O tai padaro Delegate klasę labai tinkamą anoniminiam metodų iššaukimui.

Rekomenduojama architektūra sudėtingom paskirstytom sistemom. Nesvarbu, kuri programavimo kalba pasirenkama, daug ką lemia architektūra, pagal kurią realizuojama programinė įranga. Kuriant sudėtingas paskirstytas sistemas ir siekiant jas padaryti kiek įmanoma lankstesnes, naudojami skirtingi vidiniai agentai. Viena iš esminių agentinių sistemų savybių yra jos lankstumas. Tai reiškia, kad agentai turi būti dinamiškai įdedami, pašalinami arba reorganizuojami sistemoje kuomet to reikalauja problemos sprendimas ar aplinkos pakitimas. Paprastai, kad pasiekti šio lankstumo, yra naudojami keli vidiniai agentai. Priklausomai nuo to, kaip šie vidiniai agentai yra naudojami, sistemos architektūra vadinama paskirstyta arba centralizuota.

Chunsheng Li, Zili Zhang ir Chengqi Zhang savo darbe [3] siūlo sudėtingom paskirstytom sistemom naudoti žiedo architektūrą, kuri padeda išspręsti daugybę problemų, kylančių naudojant centralizuotą arba paskirstytą architektūros metodą. Tokioje architektūroje naudojami keli vidiniai agentai, kurių kiekvienas yra atsakingas už nepriklausomas programas (angl. *Application*). Kiekviena programos komanda, kad įvykdytų nepriklausomą užduotį, turi rinkinį glaudžiai bendradarbiaujančių nepriklausomų užklausų agentų (angl. *Requester agents*). Šioje platformoje agentai yra suskirstomi į tris kategorijas:

- Vidiniai agentai (angl. *Middle agents*)
- Paslaugų tiekimo agentai (angl. *Service provider agents*)
- Programų komandiniai užklausų agentai (angl. *Application team-based requester agents*)

Vidiniai agentai atsakingi už dinamišką paslaugų agentų bei programų komandinių agentų registraciją bei atšaukimą. Vidiniai agentai taip pat skirstomi į pagrindinius (angl. *Hosts*) bei dublikatus (angl. *duplicates*) priklausomai nuo jų atliekamų veiksmų sistemoje. Priklausomai nuo aplinkos pakitimų, šie agentai taip pat gali save padauginti arba atšaukti. Platforma, įgyvendinanti čia paminėtas agentų savybes, yra pagrįsta save organizuojančiais (angl. *Self-organized*) vidiniais agentais ir žiedo architektūros modeliu [3]. Tokios platformos karkasas (angl. *Framework*) pavaizduotas 6 pav.



6 pav. Platformos karkasas [3]

Šiame modelyje platforma yra suskirstyta į tris lygius,kurie pavadinti pagal tuose lygiuose veikiančių agentų pavadinimus:

- Programų komandų lygis (angl. *Application Team*). Šiame lygyje yra agentai, kurie atsakingi už specifinių problemų sprendimą. Agentai gali būti dinamiškai įtraukiami arba pašalinami iš šio lygio. Jei nauja programų komanda nori užklausti paslaugų iš paslaugų teikimo agentų, ji pirmiausia turi būti užregistruojama šiame lygyje. Tuomet komanda kreipiasi į atitinkamą vidinį agentą, kuris jai suteikia reikiamą informaciją. Jei reikia, ši komanda po to pašalinama iš programų komandų lygio. Agentai tarpusavyje bendrauja specialia agentų bendravimo kalba – ACL (Agent Communication Language).
- Vidinių agentų lygis. Vidiniai agentai šiame lygyje yra atsakingi už programų komandų bei paslaugų tiekimo agentų bendradarbiavimą ir susiejimą. Vidiniai agentai yra suskirstyti poromis, kur vienas agentas yra pagrindinis, o kitas – jo dublikatas. Abu poros agentai saugo tą pačią informaciją, tačiau toks suskirstymas sustiprina sistemą bei garantuoja jos stabilumą.
- Paslaugų tiekimo lygis (angl. *Service provider*). Šis lygis sudarytas iš daugybės agentų, kurių kiekvienas turi savo tikslą. Agentai keičia savo vidines būsenas priklausomai nuo aplinkos ir apie tai informuoja vidinius agentus.

2.3.2. Agentų realizavimas duomenų bazėse

Įprastos duomenų bazių valdymo sistemos (DBVS) yra pasyvios. Jose duomenys paprasčiausiai kuriami, naikinami, redaguojami ir pateikiami vartotojam. Pažangios DBVS turi išplėstą

funkcionalumą ir laikomos aktyviomis, t.y. DBVS tam tikrus veiksmus vykdo automatiškai reaguodama automatiškai į tam tikrus įvykius arba susidariusias sąlygas. Įprasto duomenų bazės palaiko paprastą duomenų suvaržymą, kuomet galima užtikrinti, kad pirminis raktas būtų visada unikalus ir t.t. Tačiau dažnai reikia ir sudėtingesnių duomenų apribojimų, pavyzdžiui, kad tam tikros DB reikšmės būtų ne didesnės nei tam tikrų DB saugomų reikšmių vidurkis. Tokius suvaržymus realizuoti SQL3 standartas numato trigerius, kitaip dar vadinamus aktyviomis taisyklėmis. Trigeriai – tai aktyvūs DBVS elementai, kurie aktyvuojami susidarius tam tikrom sąlygom ir užtikrina platų funkcionalumo spektrą, kuris apjungia duomenų darnos suvaržymų realizaciją, išvestinių duomenų tvarkymą, duomenų saugumą, duomenų evoliuciją ir peržiūrą, versijų kontrolę ir pan. [9].

Bendra trigerio struktūra atrodo taip:

<SQL3-trigger> ::= CREATE TRIGGER <trigger-name>
{AFTER ¦ BEFORE | INSTEAD OF} < trigger -event> ON <table-name>
[REFERENCING <references>]
[FOR EACH {ROW ¦ STATEMENT}]
[WHEN <SQL-statement>]
<SQL-procedure-statement>

Čia:
<trigger-event> ::= INSERT | DELETE ¦ UPDATE [OF <column-names>]
<references> ::= OLD [AS] <old-value-tuple-name> |
NEW [AS] <new-value-table-name> |
OLD_TABLE [AS] <old-value-table-name> |
NEW_TABLE [AS] <new-value-table-name>

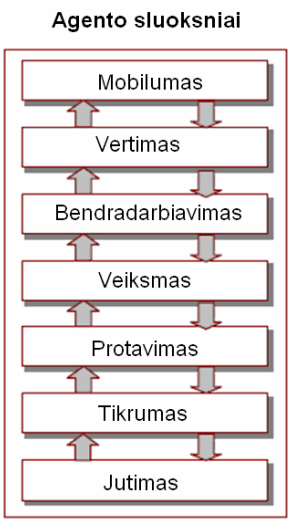
Žinoma, priklausomai nuo duomenų bazės, trigerių aprašymas gali skirtis, būti išplėstas papildomomis galimybėmis arba apribotas, tačiau trigerio principas vis tiek lieka tas pats. Trigeriai, reaguoja į DB lentelės atnaujinimą – įvykį UPDATE, DELTE arba INSERT – ir yra įvykdomi nurodytu momentu, kuris gali būti:

- AFTER – trigeris bus iškviečiamas **po** informacijos nurodytoje duomenų bazės lentelėje atnaujinimo.
- BEFORE – trigeris bus įvykdomas **prieš** informacijos atnaujinimą nurodytoje DB lentelėje, o tik po to šioje lentelėje bus atnaujinta informacija

- INSTEAD OF – trigeris bus įvykdomas **vietoj** informacijos atnaujinimo nurodytoje DB lentelėje. Pradiniai DB lentelės pakeitimai, sužadinę trigerį, šiuo atveju nebus įvykdomi.

REFERENCING trigeryje aprašo sąveiką tarp operacijos kintamųjų ir lentelių. Trigeris iškviečiamas kiekvienai paveiktai DB lentelės eilutei. Naudojant OLD [AS] galima gauti eilutės būseną prieš jos atnaujinimą, o NEW [AS] – jau atnaujintos eilutės būseną. Analogiškai ir su OLD_TABLE [AS] bei NEW_TABLE [AS], tik šiuo atveju gaunama senos DB lentelės būsena bei atnaujintos DB lentelės būsena. Iškviestas trigeris gali ne tik įvykdyti tam tikrą SQL sakinį ar jų rinkinį, bet ir iškviesti kitą duomenų bazės procedūra. Taip pat, siekiant išvengti įvairių konfliktų, trigeriai negali aprašyti daugiau nei vieną taisyklę.

Rekomenduojama architektūra. Kuriant sistemas, kuriose veikiančių agentų aplinka yra aktyvi duomenų bazė, patogiausia yra naudoti sluoksnių (angl. *Layer*) architektūrą , pavaizduotą 7 pav.. Tokia sluoksnių architektūra identifikuoja kiekvieno agento funkcijas ir gali būti naudojama tiek paprastiem, tiek ir sudėtingiem agentam realizuoti [12]. Šioje architektūroje, kiekvienas sluoksnis bendradarbiauja tik su gretimais, viršuje bei apačioje jo esančiais, sluoksniais išskyrus Mobilumo (angl. *Mobility*) ir Jutimo (angl. *Sensory*) sluoksnius. Mobilumo ir Jutimo sluoksniai, kad įvykdytų savo užduotis, bendrauja su kitais agentais ar aplinkomis. Būtent šios savybės ir yra pagrindinė priežastis, dėl ko reikėtų naudoti tokią architektūra: kiekvienas sluoksnis, išskyrus Mobilumo ir Jutimo, bendradarbiauja tik su savo gretimais kaimynais, o tai labai palengvina naujo funkcionalumo įdiegimą į sistemą.



7 pav. Agento sluoksnių architektūra

Agento sluoksniam būdingos funkcijos :

- Mobilumo sluoksnis (angl. *Mobility*) skirtas bendradarbiavimui tarp agentų.

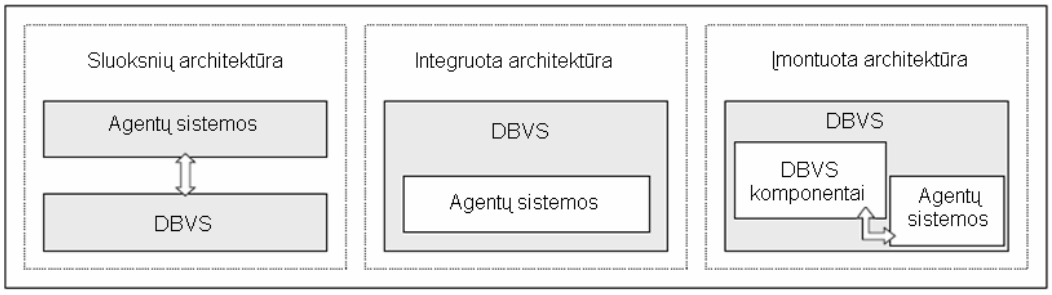
- Vertimo sluoksnis (angl. *Translation*). Šia sluoksnyje formuojamos žinutės, kurios išverčiamos į kitas kalbas arba semantikas
- Bendradarbiavimo sluoksnis (angl. *Collaboration*) apibrėžia, kaip agentai turi bendrauti tarpusavyje, nurodant ar ateinančios žinutės turi būti priimamos ar atmetamos.
- Veiksmo sluoksnis (angl. *Action*). Šiame sluoksnyje įgyvendinami Protaujančio sluoksnio veiksmai.
- Protaujantis sluoksnis (angl. *Reasoning*). Šiame sluoksnyje pasirenkamas sekantis veiksmas, kuris gali būti arba naujos paslaugos prašymas, arba atsakymas į gautą užklausą.
- Tikrumo sluoksnis (angl. *Believes*). Tai modeliai ir informacija apie agento veiksmus ir aplinką.
- Jutimo sluoksnis (angl. *Sensory*) gauna informaciją apie aplinką ir jos pokyčius.

Sluoksnių architektūroje egzistuoja du informacijos srautai:

- Nuo Mobilumo iki Jutimo srautas vykdomas, kai žinutė gaunama iš kito kurio nors agento.
- Nuo Jutimo iki Mobilumo srautas vykdomas, kuomet agentas naudoja informaciją, gautą iš jo aplinkos.

Norint apjungti agentus ir DBVS, siūloma naudoti vieną iš trijų integravimo architektūrų, pavaizduotų 8 pav. [12]:

- Sluoksnių architektūra (angl. *Layers*) yra plačiausiai naudojama, tačiau joje galima realizuoti daug mažiau funkcionalumo.
- Integruota architektūra (angl. *Integrated*) leidžia pasiekti didžiausią tarpininkavimo (angl. *Agency*) lygį, nes joje agentai pakeičia visus arba beveik visus DBVS komponentus. Deja, tokios architektūros realizacija yra labai sudėtinga.
- Įmontuota architektūra (angl. *Built-in*) leidžia panaudoti jau egzistuojančius DBVS komponentus.



8 pav. Agentų sistemų integravimo į DBVS architektūros

2.4. Analizės išvados

Šiame skyriuje buvo apžvelgti reikalavimai internetiniam portalam. Norint padaryti portalą kiek įmanoma lengviau valdomą bei automatizuotą, geriausia yra naudoti agentus. Kadangi portalai nėra laikomi didelėmis ar paskirstytomis sistemomis, agentų realizacijai geriausia naudoti objektiškai orientuotą programavimo kalbą arba duomenų bazę.

Tiek objektiškai orientuotoje kalboje, tiek ir duomenų bazėje galima kurti bei vykdyti reikalingas procedūras. Ne paslaptis, kad naudoti DB išsaugotas procedūras verta dėl trijų priežasčių: greitumo, saugumo, patikimumo. Duomenų bazėje išsaugotos procedūros veikia greitai, nes joms nereikalingas pakartotinis nagrinėjimas (angl. *Parsing*), optimizavimas ar kompiliavimas kiekvieno vykdymo metu, o tik vykdant patį pirmą kartą. Kadangi išsaugotos procedūros vykdomos SQL serveryje, jos sumažina laiką, reikalingą informacijai užkrauti kliento kompiuteryje. Taip pat naudojant išsaugotas procedūras vietoj SQL užklausų sumažėja tinklo apkrovimas, nes kompiuterio klientas serveriui siunčia tik procedūros pavadinimą (su reikalingais parametrais), o ne visą SQL užklausos tekstą. Išsaugotos procedūros taip pat gali būti naudojamos siekiant padidinti saugumą bei paslėpti esminius duomenų objektus, suteikiant vartotojui teises vien tik vykdyti procedūra. Išsaugotos procedūros taip pat gali padidinti programinės įrangos patikimumą. Pavyzdžiui, jei visi klientai naudoja tą pačią išsaugotą procedūrą atnaujinti duomenų bazei, tam reikalingas programinis kodas yra mažesnis ir daug paprasčiau lokalizuoti problemos vietą, jei ji iškyla. Taip pat išsaugotos procedūros labai naudingos atliekant pakeitimus duomenų bazėje. Pavyzdžiui, jei duomenų bazė buvo pakeista, užtenka pakeisti išsaugotą procedūrą, o programos, kuri naudos šios procedūros rezultatus perrašyti nereikia.

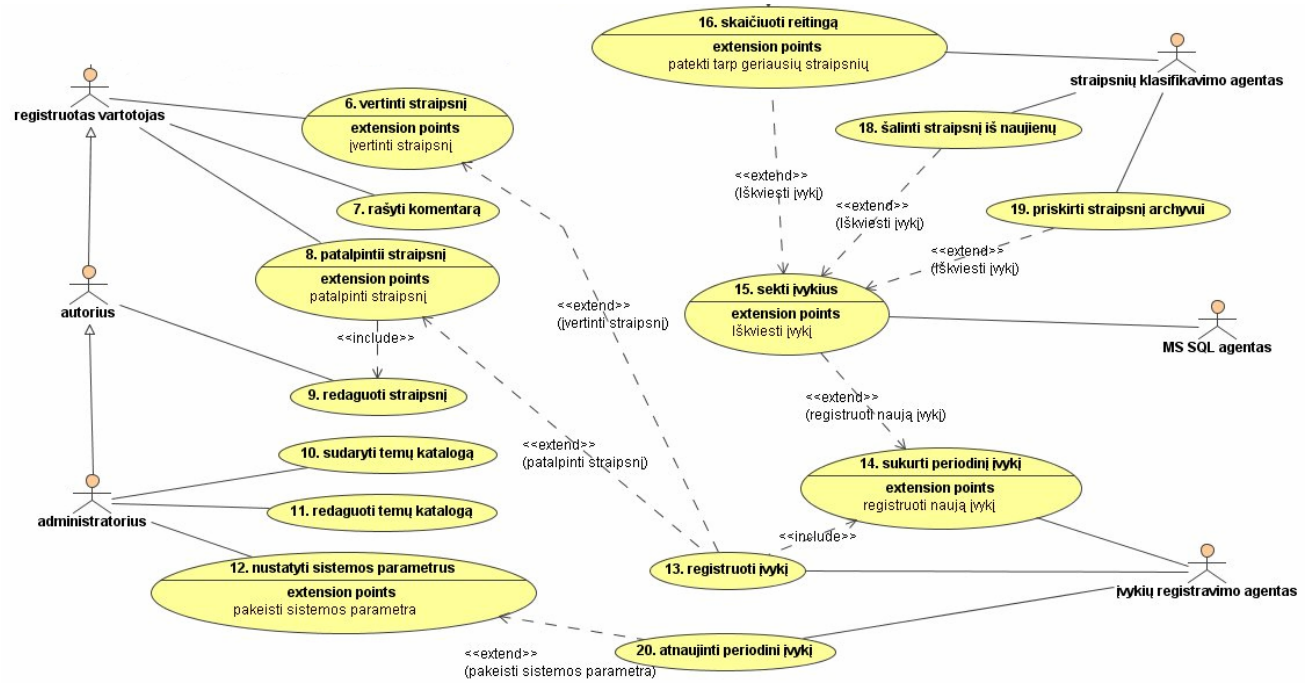
Renkantis geriausią ir patogiausią agentų realizavimo metodą, reikėtų atkreipti dėmesį į tai, kad MSSQL 2005 pasiūlė naujovę – CLR stored procedures. Ši naujovė leidžia sukurti procedūras naudojant vieną iš .NET programavimo kalbų, tokių kaip C# ar VisualBasic ir sukurtą procedūrą patalpinti duomenų bazėje kartu su kitomis duomenų bazės išsaugotomis procedūromis. Tokiu būdu sukurtos procedūros turės ne tik visas .NET programavimo galimybes, bet ir įgis DB išsaugotų procedūrų privalumus. Kadangi išsaugotų procedūrų veikimo greitis nenukenčia nuo to, kiek daug vartotojų naudojasi sistema, o pats sistemos veikimas neapkrauna vartotojų programinės sąsajos, šis realizacijos būdas labai tinkamas kuriant Web2 standartą atitinkančias programas. Žinoma, jei dalis funkcionalumo realizuojama duomenų bazėje, naudinga ir pačius agentus taip pat realizuoti duomenų bazėje, o ne programavimo kalboje.

3. Interneto portalo programinių agentų modelis ir kūrimo procesas

3.1. Sistemos agentų vaizdavimas reikalavimų modelyje

Šiuo metu agentų lygis tapo įprastu abstrakcijos lygiu aprašant agentais paremtas sistemas. Tačiau patvirtintų grafinių notacijų agentų vaizdavimui trūkumas kartais atbaido nuo agentų vaizdavimo. Būtent dėl to buvo kuriami UML išplėtimai skirti agentų vaizdavimui UML diagramose. Atsirado įvairių UML išplėtimų, vadinamų AUML, tačiau tai netapo standartu, ir daugelis CASE įrankių jų vis dar nepalaiko. Tačiau ir agentus galima atvaizduoti ir nenaudojant AUML, o remiantis tik standartiniais UML elementais.

Sistemos analizės dalyje paprastai siekiama ištirti ir išanalizuoti probleminę sritį bei nustatyti visus programinei įrangai keliamus reikalavimus. Ši analizės fazė bei jos metu paruoštos diagramos turi būti nepriklausomos nuo technologijos, kuria bus realizuojama sistema. Taip pat analizės dalyje visa sistema yra įsivaizduojama kaip juoda dėžė, bendraujanti su aplinka. Būtent dėl to agentais paremtų sistemų analizė gali atrodyti labai sudėtinga. Negana to, kai kurios agentų rūšys, tokios kaip asmeniniai asistentai ar informacijos agentai tiesiogiai bendradarbiauja su sistemos vartotojais ir jų atliekamas funkcionalumas būtinai turi būti nagrinėjamas analizės dalyje. Agentais paremtos panaudojimo atvejų diagramos yra tiesiog išplėstos standartinės panaudojimo atvejų diagramos. Jose atvaizduojami visi veikiantys agentai bei jų sąveika su kitais agentais, vartotojais, aplinka bei pačia sistema. Net jei agentai yra modeliuojami už juodos dėžės ribų, jie vis tiek yra sistemos dalis ir gali būti analizuojami atskirai siekiant juos plačiau išanalizuoti ir detaliau pavaizduoti jų atliekamą specialų funkcionalumą. Siekiant atvaizduoti agentais paremtas panaudojimo atvejų diagramas, rekomenduojama į sistemą įsivesti naują stereotipą *agentas*. Tai palengvintų ne tik tokių diagramų atvaizdavimą bei apdorojimą CASE įrankiais, bet ir leistų agentus traktuoti kaip paprastus aktorius. Jei agentai tiesiogiai bendrauja su aplinka, vartotojais ar kitais agentais, jų susiejimui su atliekamu scenarijumi yra naudojamas asociacijos ryšys. Agentų veikimas taip pat dažnai būna vaizduojamas <<*extend*>>, dar kitaip vadinamais išplėtimo, scenarijais. Agentų vaizdavimas panaudojimo atvejų diagramoje pateiktas 9 pav.



9 pav. Agentų vaizdavimas panaudojimo atvejų modelyje

Analizės dalyje aprašant panaudojimo atvejų modelį labai svarbu ne tik detalizuoti ir aprašyti sistemos vartotojus bei jų atliekamus scenarijus, bet ir agentus su jų atliekamais scenarijais. Tik išsami analizė ir agentų atliekamų veiksmų detalizavimas padeda pilnai suprasti kuriamos sistemos veikimą, išvengti pasikartojančių ar persikertančių agentų veiksmų, ciklų.

3.2. Agentų kūrimo procesas

Pilnai išsiaiškinus sistemos veikimą analizės dalyje, nustačius būsimus agentus bei aprašius ir išanalizavus jų atliekamus scenarijus, galima pereiti prie sekančio sistemos kūrimo žingsnio – sistemos architektūros sudarymo bei jos realizavimo.

Projektuojant kelių agentų sistemas agentų lygyje reikia modeliuoti ir agentais paremtos sistemos architektūrą bei sąveikas tarp agentų. Tai gali būti modeliuojama naudojant UML klasių diagramas, kuriose nurodomi agentų atliekami veiksmai sistemose. Kiekviena agento rolė yra asocijuojama su klase, kuri paprastai vadinama agento klase. Šios klasės pažymimos agentų stereotipu, kuris sutampa su stereotipu, naudojamu panaudojimo atvejų diagramoje. Asociacijos tarp agentų klasių nusako asociacijas tarp agentų, atliekančių skirtingus veiksmus. Agento klasėje gali būti aprašomi vienas ar keli agento atliekami veiksmai. Veiksmai paprastai aprašomi kaip vieši klasės metodai (angl. *Publi method*)

Realizuojant agentus duomenų bazėse, agentų veiksmai realizuojami naudojant trigerius bei procedūras. Nepaisant to, kad duomenų bazės trigeriai bei procedūros yra pakankamai galingas įrankis, daugelis nelabai patyrusių sistemos vartotojų jų vengia, nes juos kuriant neplanuotai ir atmestinai, galima sistemos realizaciją ne palengvinti, o dar labiau apsunkinti. Būtent dėl to rekomenduojama visus sistemos agentus gerai išanalizuoti bei nustatyti jų veiksmus sistemos analizės dalyje. Ir tik po to sumodeliavus sistemos architektūrą imtis realizuoti agentus duomenų bazėje.

Kiekvienas agento veiksmas turi būti sužadinamas tam tikro įvykio. DBVS šiuos įvykius atstoja duomenų įterpimas į DB lentelę, jų šalinimas ar atnaujinimas. Minėtiem pagrindiniam įvykiam – INSERT, DELETE ir UPDATE – yra kuriami trigeriai, kurie sistemoje realizuoja agentus. Taip pat agentą galima realizuoti ir naudojant standartinius duomenų bazėje egzistuojančius agentus. Šiuo atveju tai būtų standartinis MSSQL duomenų bazės agentas – MS SQL Agent. Šis agentas, priešingai nei sukurtieji pasinaudojus trigerius, reaguoja į laiko įvykius. Lengva ir paprasta MSSQL serverio pagalbinė sąsaja leidžia šį agentą redaguoti ir nurodyti laiką, kuomet agentas yra sužadinamas.

Agentų klasės duomenų bazėje realizuojamos kuriant procedūras kiekvienam agento atliekamam veiksmui. Kadangi duomenų bazės procedūros turi žymiai daugiau galimybių nei trigeriai, todėl jomis ir rekomenduojama realizuoti agentų atliekamus veiksmus, trigeriam paliekant tik mažą funkcionalumą – reikiamos procedūros iškvietimą reikiamu metu.

Suprantama, tokiu būdu realizuojant agentus, visa realizacija paslepiama duomenų bazės lygyje. Tai ne tik paspartina sistemos veikimą, nes neapkrauna vartotojo sąsajos, bet ir leidžia keisti agentų scenarijus nekeičiant pačios programos kodo. Tokia realizacija taip pat labai naudinga vėliau koreguojant sukurtąją informacinę sistemą, nes programuotojui nėra būtin žinoti programavimo kalbą, kuria buvo realizuota sistema. Užtenka žinoti SQL kalbą, kuri yra standartas ir būtent ja realizuojami trigeriai ir procedūros. Žinoma, jei agentų atliekami veiksmai ar tam tikri nustatymai turi būti prieinami ir matomi vartotojui, reikia kurti atskirą vartotojo sąsają. Jei vartotojam suteikiama galimybė nustatyti parametrus, kuriais remsis agentai, šiuos parametrus reikia saugoti atskiroje duomenų bazės lentelėje, o sužadinti agentai, prieš vykdydami tolesnius veiksmus, šiuos parametrus turėtų pasitikrinti.

Jei norima tam tikrus scenarijus vykdyti iš karto, tam geriau naudoti agentus realizuotus trigeriais. Jei scenarijų vykdymas turi būti atidėtas vėlesniam laikui, rekomenduojama scenarijus bei vykdymo laiką išsaugoti atskiroje duomenų bazės lentelėje ir jų vykdymą patikėti ne trigeriui, o agentui, reaguojančiam į laikinius įvykius. Šiuo atveju tam pasitarnauja duomenų bazėse jau

realizuoti agentai tokie kaip MS SQL Agent. Nurodytais laiko intervalais toks agentas turėtų peržiūrėti laukiančių įvykių sąrašą ir įvykdyti tuos, kuriem jau atėjo laikas būti įvykdytiems.

3.3. Agentų veikimo scenarijų analizė

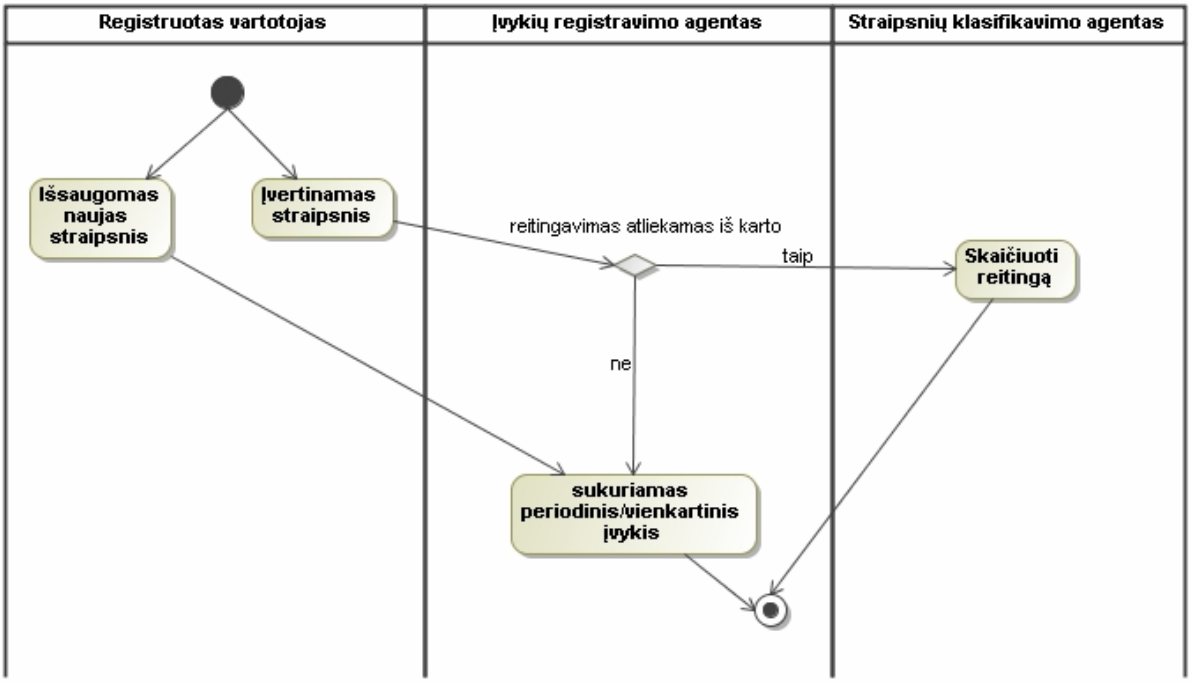
Bendru atveju, galima išskirti du agentų bendravimo ir jų veiksmų sužadinimo metodus:

- Agentas bendrauja su kitu agentu, o agento veiksmus sužadina kitas agentas.
- Agentas bendrauja su aplinka, o agento veiksmus sužadina aplinkos įvykis.

Paprastose informacinėse sistemose dažniausiai sutinkamas būtent antrasis agentų sužadinimo metodas. Aplinka, kurioje įvykęs įvykis ir sužadina agentą, paprastai priklauso nuo to, kokiu būdu agentai yra realizuoti. Jei agentai realizuojami duomenų bazėse trigerių pagalba, jų aplinka ir yra duomenų bazė, o sužadinantis įvykis gali būti informacijos įterpimas, atnaujinimas ar pašalinimas.

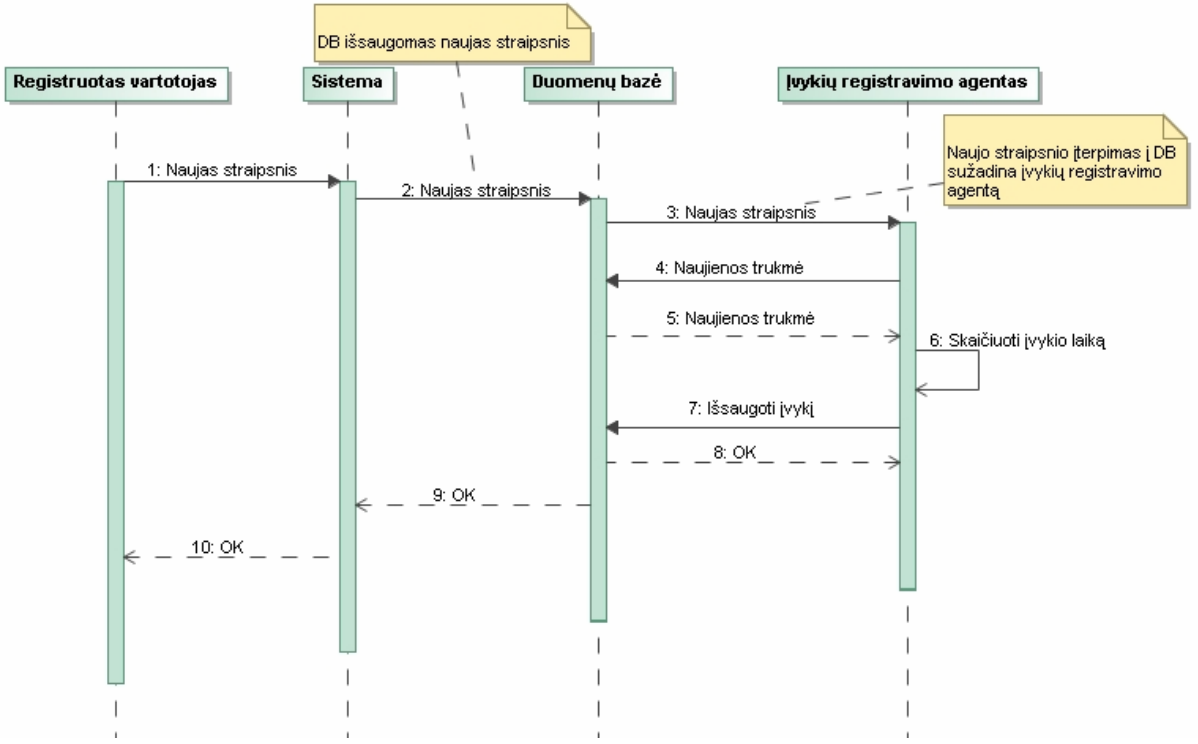
Prieš realizuojant agentus, visų pirma reikia ne tik gerai išsianalizuoti jų atliekamus veiksmus, bet ir jų veikimą bei bendradarbiavimą. Šiuo atveju patogiausia būtų naudoti veiklos ir sekų diagramas, kuriose būtų atvaizduoti agentai, jų bendravimas su aplinka ir kitais agentais bei atliekamų veiksmų eiliškumas bei perduodama informacija.

Pavyzdys. *Įvykio registravimas. Kuomet vartotojas įveda naują straipsnį ar vertinimą, Įvykių registravimo agentas tai užfiksuoja ir atitinkamai sukuria reikiamą periodinį/vienkartinį įvykį, nustatydamas laiką, kada tai turi įvykti. Įvykio registravimo veiklos diagrama pavaizduota* 10 *pav.*



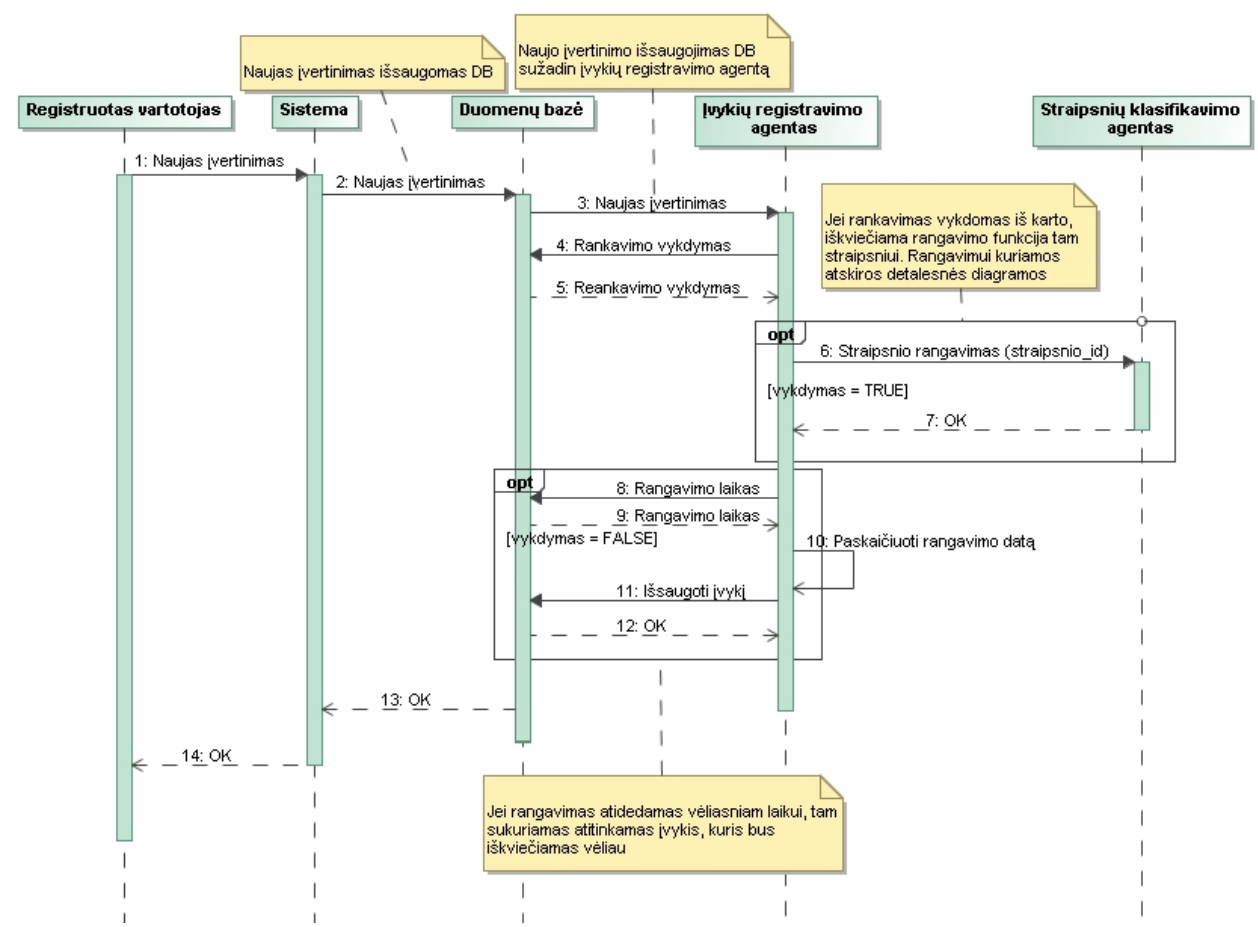
10 pav. Įvykio registravimo veiklos diagrama

Sekų diagramoje visa ši situacija atvaizduojama detaliau, nurodant ir tarp objektų siunčiamas/gaunamas žinutes bei jų eiliškumą. Kadangi įvykis užregistruojamas dviem atvejais (išsaugojus naują straipsnį arba įvertinus jau esamą straipsnį), abiem šiem veiksmam sukurkime po sekų diagramą. Įvykio registravimo sekų diagrama, kuomet įvykis fiksuojamas įvedus naują straipsnį, pavaizduota 11 pav.



11 pav. Įvykio registravimas, kuomet įvykį sužadina naujai įvestas straipsnis

Įvykio registravimo sekų diagrama, kuomet įvykis fiksuojamas išsaugojus naują įvertinimą, pavaizduota 12 pav.



12 pav. Įvykio registravimas, kuomet įvykį sužadina straipsnio vertinimas

Tam, kad išvengti pasikartojančių ar persikertančių agentų veiksmų, reikėtų pirma gerai išanalizuoti agentų atliekamus veiksmus, o tik tada pereiti prie pačio realizacijos proceso.

3.4. Agentų meta modelis

Šiuo metu egzistuoja keletas įvairių metodų, nusakančių kaip turi būti modeliuojami programiniai agentai. Vieni žinomesnių metodų yra AUML bei AOR.

AUML buvo kuriamas siekiant suteikti pastovias ir paplitusias notacijas agentų modeliavimui. AUML – tai UML išplėtimai skirti agentų vaizdavimui UML diagramose. Nors ir buvo sukurta daugelis tokių išplėtimų, jie netapo standartu ir daugelyje CASE įrankių yra nepalaikomi.

Kitas žinomas metodas, skirtas koncepciniam informacinių sistemų modeliavimui, yra AOR modeliavimas (angl. *Agent-Object-Relationship Modeling*) [5]. AOR modelio tikslas yra suteikti informaciją apie agentus, naudojamus kuriant agentais paremtas informacines sistemas. Visi sukurti sistemos modeliai turėtų būti nesunkiai paverčiami į atitinkamą duomenų bazės schemą. Tai reiškia, kad visi AOR meta-modelio elementai turi turėti formalias semantikas.

Tam, kad užfiksuoti semantinius dinamiškos informacinės sistemos aspektus, būtina suprasti skirtumą tarp agentų ir pasyvių objektų. Nors tiek agentai, tiek ir pasyvūs objektai egzistuoja sistemoje, tačiau tik agentai gali su ja sąveikauti. AOR modelyje esybė (angl. *Entity*) gali būti įvykis (angl. *Event*), veiksmas (angl. *Action*), reikalavimas (angl. *Claim*), perdavimas (angl. *Commitment*), agentas (angl. *Agent*) arba objektas (angl. *Object*). Tik agentai gali bendrauti, pajusti, veikti, vykdyti perdavimus ir patenkinti reikalavimus. Objektai nebendrauja, negali nieko pajusti, negali veikti, ir neturi jokių siuntimų ar reikalavimų. Tačiau būdami esybėmis, to paties tipo agentai ir objektai turi tuos pačius atributus, nusakančius jų savybes bei charakteristikas. Taigi, AOR modeliavime naudojamos tos pačios notacijos kaip ir ER modeliavime. Objektų tipai, tokie kaip straipsnis, vertinimas, komentaras atvaizduojami stačiakampiu, taip kaip esybių tipai ER modelyje, arba objektų klasės UML klasių diagramose. Objektai gali turėti asociacijos (angl. *Association*), apibendrinimo (angl. *Generalization*) arba agregavimo/kompozicijos (angl. *Aggregation/composition*) ryšius su kitais objektų tipais arba asociacijos ar agregavimo/kompozicijos ryšius su agentais. Asociacijos ryšiai yra specifikuojami taip pat kaip UML (pvz. 0..1, 1..*). Agentų tipas AOR diagramose yra vaizduojamas stačiakampiu suapvalintais kampais. Taip pat galima naudoti ikonas, kurios nurodo ar tai žmogus, grupė ar robotas.

Egzistuoja du pagrindiniai AOR modelių tipai: išorinis (angl. *External*) ir vidinis (angl. *Internal*). Išorinis AOR modelis apima išorinio stebėtojo, kuris stebi agentus ir jų bendravimą dalykinėje srityje, perspektyvą. Vidiniame AOR modelyje, apimamas vidinio (pirmo-asmens) stebėtojo žvilgsnis į modeliuojamą agentą. Dėl šio skirtumo siūloma tokia eiliškumo tvarka projekto vykdymui:

1) analizės dalyje sukuriamas išorinis AOR modelis, apimantis vieną ar daugiau esminių agentų
2) projektavimo fazėje kiekvienam esminiam agentui išorinis AOR modelis transformuojamas į vidinį remiantis agento perspektyva.
3) galiausiai vidinis AOR modelis įgyvendinamas pasirinkta kalba, kuria gali būti SQL, Java ar kitos programavimo kalbos.

Išorinis AOR modelis. Standartiškai, išoriniame AOR modelyje dalykinė sritis susideda iš šių elementų:

- agentai
- komunikaciniai (angl. *Communicative*) ir nekomunikaciniai (angl. *Non-communicative*) įvykiai

- ne veiksmo (angl. *Non-action*) įvykiai
- siuntimai/reikalavimai (angl. *Commitments/claims*) tarp dviejų agentų
- įprasti objektai
- įvairūs nustatyti ryšiai, tokie kaip siunčia (angl. *Sends*) ar atlieka (angl. *Does*).
- Įprasti ryšiai

Išorinis AOR modelis gali susidėti iš vienos ar daugiau šių diagramų:

- **Agentų diagramos (angl. *Agent diagrams*).** Nurodo agentus, susijusius objektus bei jų ryšius.
- **Sąveikos sandaros diagramos (angl. *Interaction Frame Diagrams*).** Pavaizduoja veiksmų įvykių tipus bei siuntimo/reikalavimo tipus tarp agentų.
- **Sąveikos sekų diagramos (angl. *Interaction Sequence Diagrams*).** Pavaizduoja sąveikos proceso atskirus atvejus.
- **Sąveikos modelių diagramos (angl. *Interaction Pattern Diagrams*).** Apibrėžiamas sąveikos tipas bei sąveikos taisyklės.

Vidinis AOR modelis. Standartiškai, vidiniame AOR modelyje dalykinė sritis susideda iš šių elementų:

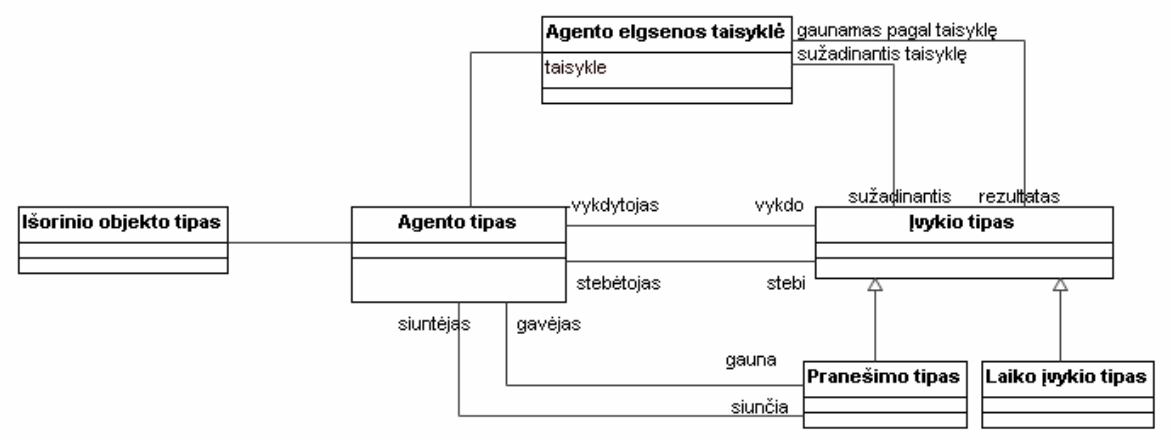
- kiti agentai
- veiksmai (angl. *Actions*)
- perdavimai kitiem agentam siekiant atlikti tam tikrus veiksmus
- įvykiai, kurių dauguma yra sukurti kitų agentų
- reikalavimai iš kitų agentų konkrečiame veiksme
- įprasti objektai
- įvairūs nustatyti ryšiai, tokie kaip yra-išsiųstas (angl. *isSentTo*) ar yra-pastebėtas (angl. *isPerceivedBy*)
- įprasti ryšiai/asociacijos

Vidinis AOR modelis gali susidėti iš vienos ar daugiau šių diagramų:

- **Reakcijos sandaros diagramos (angl. *Reaction Frame Diagrams*).** Pavaizduoja kitus agentus bei veiksmų ir įvykių tipus, nusakančius sąveikas su jais.
- **Reakcijos sekų diagramos (angl. *Reaction Sequence Diagrams*).** Atvaizduojami prototipiniai atskirų atvejų sąveikos procesai.
- **Reakcijos modelių diagramos (angl. *Reaction Pattern Diagrams*).** Dėmesys kreipiamas į agento elgesį esant tam tikrom taisyklėm.

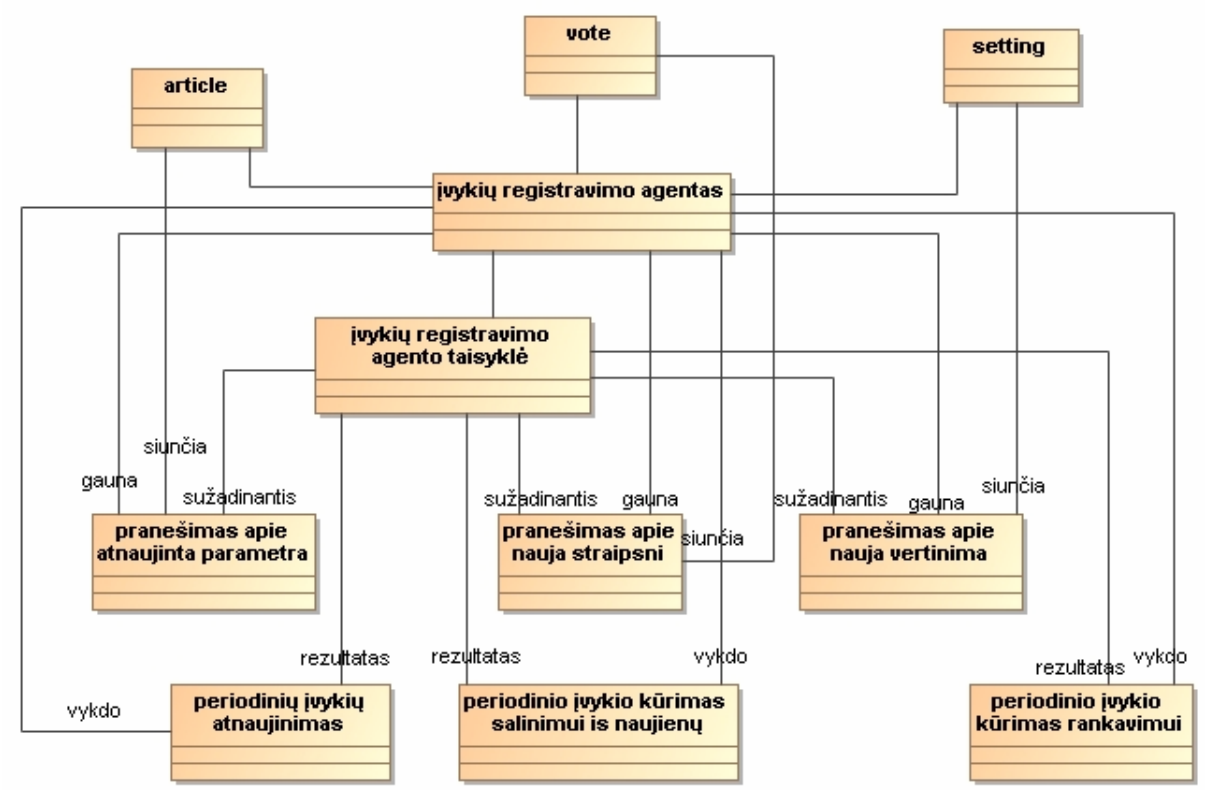
Kadangi šiuo metu nėra nusistovėjusio standarto agentų modeliavimui, dažnai apskritai yra vengiama juos modeliuoti. Programiniai agentai yra paliekami nuošalyje ir neatsispindi arba menkai atsispindi sistemos modeliuose. Tačiau tai yra viena didžiausių klaidų. Agentai turi būti modeliuojami bei atsispindėti sistemos modeliuose.

Remiantis AOR, galima būtų sudaryti tokį programinių agentų meta-modelį (13 paveikslas):

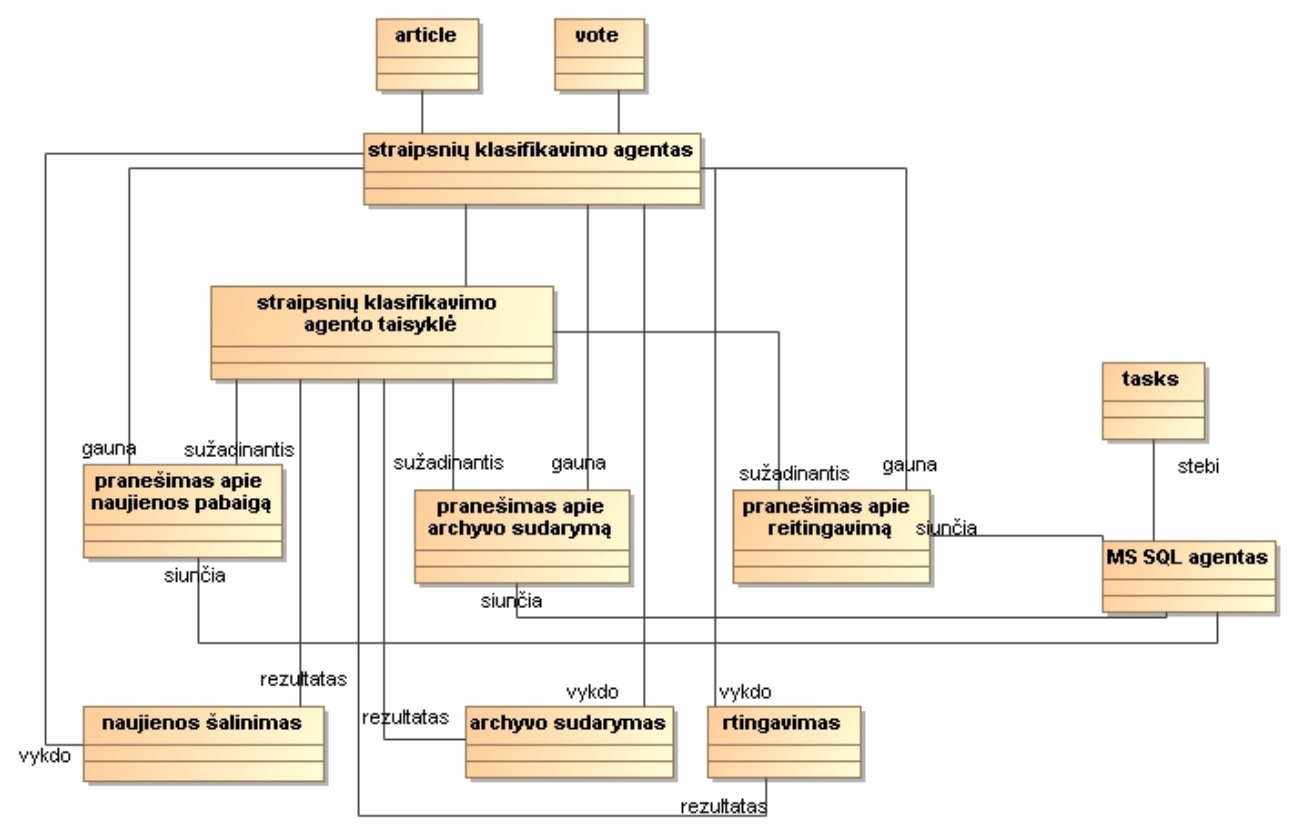


13 pav. Agentų meta-modelis

Remiantis 13 pav. pavaizduotu agentų meta-modeliu, galima sudaryti šio darbo eksperimentinėje dalyje realizuotų agentų modelius (14 pav., 15 pav.):



14 pav. Eksperimentinės sistemos įvykių registravimo agento modelis

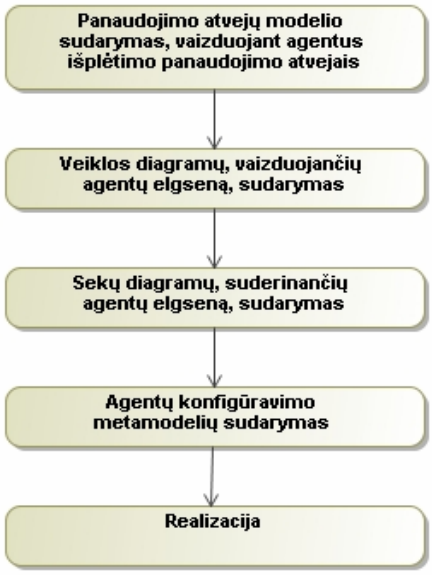


15 pav. Eksperimentinės sistemos straipsnių agentų modelis

Kuriant paprastas informacines sistemas, galima išsiversti ir be specifinių UML metodų agentų modeliavimui. Programinius agentus galima nesunkiai modeliuoti pasinaudojant ir standartinėmis UML notacijomis. Tai ne tik neapsunkintų pačio modeliavimo proceso, bet ir leistų agentų modeliavimą lengvai integruoti į praktikoje naudojamus kūrimo procesus. Taigi, tokiu atveju, agentų modeliavimas būtų lengviau pritaikytas didesniam vartotojų ratui.

Modeliuojant agentus naudojant standartines UML notacijas, daugiausia dėmesio reikėtų skirti panaudojimo atvejų, veiklos bei sekų diagramoms. Kuriant agentais paremtas sistemas, taip pat kaip ir kuriant paprastas informacines sistemas, pradžioje būtina gerai išanalizuoti dalykinę sritį. Sistemos dalykinės srities analizė pradedama nuo reikalavimų, keliamų sistemai išsiaiškinimo. Remiantis turimais reikalavimais yra sudaromas sistemos panaudojimo atvejų modelis. Šiame modelyje kartu su sistemos vartotojais yra atvaizduojami ir sistemoje veikiantys agentai. Aprašant sistemai keliamus reikalavimus, turi būti išanalizuoti ne tik vartotojų su sistema atliekami veiksmai bei sistemos galimybės, bet ir sistemos agentų atliekami veiksmai, jiem keliami reikalavimai.

Kuomet jau yra nustatyta, kokie agentai sąveikauja sistemoje, reikėtų pradėti detalesnę agentų analizę, leidžiančią užtikrinti, jog agentų veiksmai ne tik nepersikerta ar nesidubliuoja, bet ir yra efektyvūs konkrečiu atveju. Šiuo atveju, kiekvienam iš agentų veiksmų turėtų būti sudaromos veiklos diagramos, kurios po to turėtų būti dar labiau detalizuojamos braižant sekų diagramas. Agentų analizės metu būtina atkreipti dėmesį ne tik į agentų bendravimą su sistema, bet ir į agentų tarpusavio bendravimą žinutėmis. Taigi, sistemų kūrimo procese, agentam išskirti bei išanalizuoti reikalingi veiksmai, pavaizduoti 16 pav.



16 pav. Agentų analizės bei realizacijos veiksmai sistemų kūrimo procese

4. Eksperimentinė portalo su programiniais agentais realizacija

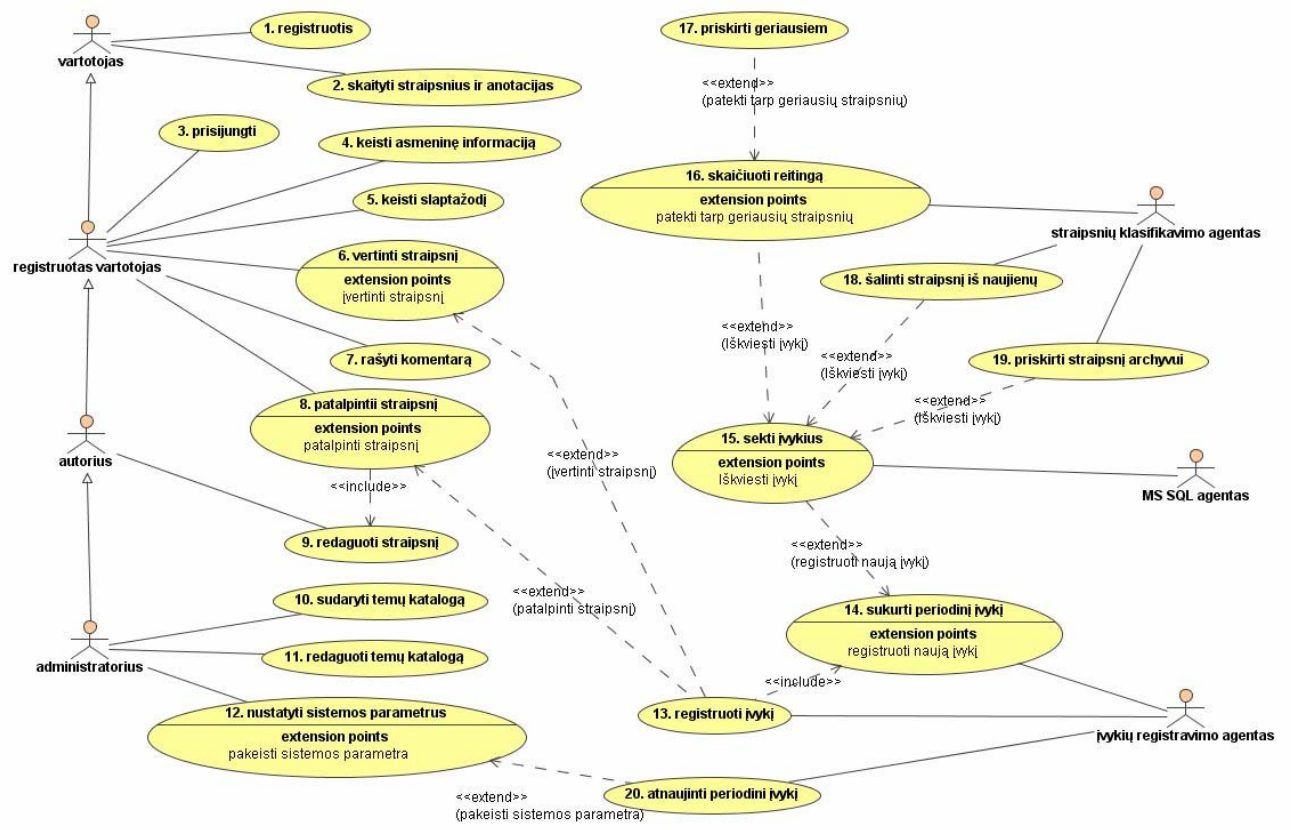
Tam, kad būtų galima pademonstruoti agentų veikimą, buvo sukurta eksperimentinė sistema. Sistema buvo realizuota naudojant MS .NET 2.0 *Framework* programinę įrangą ir MS SQL Server. Realizacijos metu buvo:

- Sukurta lengvai naudojama sistema, paremta internetine sąsaja
- Realizuota sistemos reakcija į periodinius įvykius (nurodytais laiko tarpais sistema pati perrūšiuos naujienų ir archyvo pakatalogius)
- Realizuotas straipsnių reitingavimas, ir nuo reitingų priklausantis „Top“ straipsnių sąrašas (priklausomai nuo vartotojų vertinimų, straipsniai įgauna tam tikrą reitingą; straipsniai su didžiausiu reitingu atvaizduojami „Top“ sąraše/pakatalogyje)
- Realizuotos pagrindinės sistemos vartotojo funkcijos:
    - Prisijungimas
    - Registracija
    - Straipsnių įkėlimas
    - Straipsnių redagavimas
    - Straipsnių vertinimas
    - Straipsnių komentavimas
    - Straipsnių peržiūra

Kuriant šią sistemą, pagrindinis dėmesys buvo skirtas tam, kaip sėkmingai užtikrinti laiko įvykių generavimą ir sistemos reakciją į laiko bei kitus įvykius. Buvo sukurtas bendresnis įvykių fiksavimo ir sistemos reakcijos į įvykius automatizavimo modelis, kurį projektuotojai gali pritaikyti kuriant panašias sistemas.

4.1. Eksperimentinio portalo reikalavimai

Ribas tarp sistemos ir vartotojo nusako panaudojimo atvejų modelis, pateiktas 17pav.



17 pav. Sistemos panaudojimo atvejų modelis

1. PANAUDOJIMO ATVEJIS: Registruotis	
Vartotojas/aktorius:	Vartotojas
Aprašas:	Procesas, kuomet neregistruotas vartotojas prisiregistruoja prie sistemos, taip įgaudamas daugiau teisių naudotis sistema.
Prieš – sąlyga:	Vartotojas yra neregistruotas sistemoje
Sužadinimo sąlyga:	Naujas vartotojas nori prisiregistruoti sistemoje
Po – sąlyga:	Sistemoje užregistruojamas naujas vartotojas ir jam suteikiama daugiau teisių naudotis sistema.
2. PANAUDOJIMO ATVEJIS: Skaityti straipsnius ir anotacijas	
Vartotojas/aktorius:	Vartotojas, registruotas vartotojas, autorius, administratorius
Aprašas:	Procesas, kuomet sistemos vartotojai (registruoti ir neregistruoti) gali peržiūrėti sistemoje patalpintus straipsnius.
Prieš – sąlyga:	-

Sužadinimo sąlyga:	Vartotojas pasirenka straipsnį, kurį nori peržiūrėti
Po – sąlyga:	-
3. PANAUDOJIMO ATVEJIS: Prisijungti	
Vartotojas/aktorius:	Registruotas vartotojas, autorius, administratorius
Aprašas:	Procesas, kurio metu vartotojas prisijungia prie sistemos naudodamasis jam suteiktu vartotojo vardu bei slaptažodžiu
Prieš – sąlyga:	Vartotojas turi būti registruotas sistemoje Vartotojas tuo metu negali būti jau prisijungęs prie sistemos
Sužadinimo sąlyga:	Vartotojas nori prisijungti prie sistemos
Po – sąlyga:	Vartotojas prijungiamas prie sistemos. Vartotojui suteikiamos papildomos galimybės naudotis sistema.
4. PANAUDOJIMO ATVEJIS: Keisti asmeninę informaciją	
Vartotojas/aktorius:	Registruotas vartotojas, autorius, administratorius
Aprašas:	Procesas, kurio metu vartotojas gali pasikeisti savo asmeninę informaciją, kurią nurodė registracijos prie sistemos metu.
Prieš – sąlyga:	Vartotojas turi būti registruotas sistemoje Vartotojas turi būti prisijungęs prie sistemos
Sužadinimo sąlyga:	Vartotojas nori keisti asmeninę informaciją
Po – sąlyga:	Atnaujinama vartotojo asmeninė informacija.
5. PANAUDOJIMO ATVEJIS: Keisti slaptažodį	
Vartotojas/aktorius:	Registruotas vartotojas, autorius, administratorius
Aprašas:	Procesas, kurio metu vartotojas pasikeičia savo prisijungimo slaptažodį
Prieš – sąlyga:	Vartotojas turi būti registruotas sistemoje Vartotojas turi būti prisijungęs prie sistemos
Sužadinimo sąlyga:	Vartotojas nori pasikeisti slaptažodį
Po – sąlyga:	Vartotojo slaptažodis pakeičiamas į naujai nurodytąjį.
6. PANAUDOJIMO ATVEJIS: Vertinti straipsnį	
Vartotojas/aktorius:	Registruotas vartotojas, autorius, administratorius
Aprašas:	Procesas, kurio metu galima įvertinti straipsnį, nurodant balą (vertinimo skalė yra balai nuo 1 iki 5)
Prieš – sąlyga:	Vartotojas yra prisijungęs prie sistemos Vartotojas nėra to straipsnio autorius

	Vartotojas dar nėra įvertinęs to straipsnio
Sužadinimo sąlyga:	Vartotojas nurodo vertinamo balą ir spaudžia mygtuką „Vertinti“
Po – sąlyga:	Sistemoje užfiksuojamas įvertinimo balas
	Sistemoje užfiksuojama kuris vartotojas įvertino straipsnį
	Sistemoje užregistruojamas vertinimo įvykis.
7. PANAUDOJIMO ATVEJIS: Rašyti komentarą	
Vartotojas/aktorius:	Registruotas vartotojas, autorius, administratorius
Aprašas:	Procesas, kurio metu galima parašyti komentarą prie straipsnio
Prieš – sąlyga:	Vartotojas yra prisijungęs prie sistemos
Sužadinimo sąlyga:	Vartotojas parašo komentarą ir spaudžia mygtuką „Komentuoti“
Po – sąlyga:	Sistemoje užfiksuojamas straipsnio komentaras
	Sistemoje užfiksuojama, kuris vartotojas komentavo straipsnį
8. PANAUDOJIMO ATVEJIS: Patalpinti straipsnį	
Vartotojas/aktorius:	Registruotas vartotojas, autorius, administratorius
Aprašas:	Procesas, kurio metu vartotojas į sistemą patalpina savo straipsnį
Prieš – sąlyga:	Vartotojas turi būti prisijungęs prie sistemos
Sužadinimo sąlyga:	Vartotojas nurodo straipsnį, parašo jo anotaciją bei nurodo temos pakatalogį ir spaudžia mygtuką „Publikuoti“
Po – sąlyga:	Straipsnis ir su juo susijusi informacija išsaugoma sistemoje
	Sistemoje užregistruojamas straipsnio patalpinimo įvykis.
9. PANAUDOJIMO ATVEJIS: Redaguoti straipsnį	
Vartotojas/aktorius:	Autorius
Aprašas:	Procesas, kurio metu vartotojas gali pakoreguoti savo pateiktą straipsnį, jo anotaciją bei pakeisti temos pakatalogį, kuriame publikuojamas straipsnis
Prieš – sąlyga:	Vartotojas turi būti prisijungęs prie sistemos
	Vartotojas turi būti straipsnio autorius
Sužadinimo sąlyga:	Vartotojas paspaudžia ant pasirinkto savo straipsnio
Po – sąlyga:	Sistemoje atnaujinama informacija apie straipsnį
10 PANAUDOJIMO ATVEJIS: Sudaryti temų katalogą	
Vartotojas/aktorius:	Administratorius
Aprašas:	Procesas, kurio metu sudaromas temų, pagal kurias bus grupuojami

	straipsniai, katalogas
Prieš – sąlyga:	Įvedama nauja tema
Sužadinimo sąlyga:	Administratorius įveda naują temą ir paspaudžia saugojimo mygtuką
Po – sąlyga:	Temų katalogas pasipildo nauja tema
11 PANAUDOJIMO ATVEJIS: Redaguoti temų katalogą	
Vartotojas/aktorius:	Administratorius
Aprašas:	Procesas, kurio metu pakoreguojama temos informacija
Prieš – sąlyga:	Tema turi jau būti įvesta į sistemą
Sužadinimo sąlyga:	Pasirenkama tema redagavimui
Po – sąlyga:	Atnaujinama temos informacija
12. PANAUDOJIMO ATVEJIS: Nustatyti sistemos parametrus	
Vartotojas/aktorius:	Administratorius
Aprašas:	Procesas, kurio metu administratorius nurodo/koreguoja sistemos parametrus
Prieš – sąlyga:	Vartotojas turi būti prisijungęs prie sistemos Vartotojas turi turėti administratoriaus teises
Sužadinimo sąlyga:	Paspaudžiamas mygtukas „Redaguoti parametrus“
Po – sąlyga:	Sistemoje išsaugomi parametrų pakeitimai Iškviečiamas periodinių įvykių atnaujinimas pagal pakoreguotą informaciją.
13. PANAUDOJIMO ATVEJIS: Registruoti įvykį	
Vartotojas/aktorius:	Įvykių registravimo agentas
Aprašas:	Įkėlus naują straipsnį, ar įvertinus straipsnį, sistemoje užregistruojamas įvykis
Prieš – sąlyga:	-
Sužadinimo sąlyga:	Į sistemą įkeltas naujas straipsnis arba įvertintas jau esantis straipsnis
Po – sąlyga:	Iškviečiamas periodinio įvykio sukūrimas
14. PANAUDOJIMO ATVEJIS: Sukurti periodinį įvykį	
Vartotojas/aktorius:	Įvykių registravimo agentas
Aprašas:	Procesas, kurio metu sukuriamas periodinis įvykis, kuris nusako, kokiu metu kurią funkciją reikia įvykdyti.
Prieš – sąlyga:	-

Sužadinimo sąlyga:	Įvedamas/pakoreguojamas sistemos parametras, nusakantis, kas kiek laiko bus sudaromas archyvas
Po – sąlyga:	Sistemoje užfiksuotas periodinis įvykis
15. PANAUDOJIMO ATVEJIS: Sekti įvykius	
Vartotojas/aktorius:	MS SQL agentas
Aprašas:	Procesas kurio metu agentas peržiūri įvykių sąrašą ir pradeda vykdyti įvykį, jei atėjo tam laikas.
Prieš – sąlyga:	MS SQL agentas turi būti startuotas MS SQL serveryje
Sužadinimo sąlyga:	MS SQL agentas peržiūri įvykius periodiškai
Po – sąlyga:	Iškviečiama reikiamo periodinio įvykio funkcija
16. PANAUDOJIMO ATVEJIS: Skaičiuoti reitingą	
Vartotojas/aktorius:	Straipsnių klasifikavimo agentas
Aprašas:	Procesas, kurio metu perskaičiuojamas straipsnio reitingas
Prieš – sąlyga:	-
Sužadinimo sąlyga:	Registruotas vartotojas įvertino straipsnį (jei nustatyta, kad perskaičiuojama iškart po įvertinimo) arba perskaičiavimas iššaukiamas periodinio įvykio
Po – sąlyga:	Sistemoje pakoreguotas straipsnio vertinimas Iškviečiamas top sąrašo atnaujinimas
17. PANAUDOJIMO ATVEJIS: Priskirti geriausiem	
Vartotojas/aktorius:	Straipsnių klasifikavimo agentas
Aprašas:	Procesas, kurio metu perrūšiuojamas top sąrašas
Prieš – sąlyga:	Buvo įvertintas straipsnis
Sužadinimo sąlyga:	Skaičiuojamas straipsnio reitingas
Po – sąlyga:	Atnaujintas top sąrašas
18. PANAUDOJIMO ATVEJIS: Šalinti straipsnį iš naujienų	
Vartotojas/aktorius:	Straipsnių klasifikavimo agentas
Aprašas:	Procesas, kurio metu straipsnis pašalinamas iš naujienų
Prieš – sąlyga:	Straipsnis buvo pažymėtas kaip naujiena
Sužadinimo sąlyga:	Periodinis įvykis
Po – sąlyga:	Straipsnis sistemoje pašalintas iš naujienų

19. PANAUDOJIMO ATVEJIS: Priskirti straipsnį archyvui	
Vartotojas/aktorius:	Straipsnių klasifikavimo agentas
Aprašas:	Procesas, kurio metu straipsnis priskiriamas archyvui
Prieš – sąlyga:	Straipsnis turi nepriklausyti archyvui
Sužadinimo sąlyga:	Periodinis įvykis
Po – sąlyga:	Straipsnis sistemoje patalpintas į archyvą
20. PANAUDOJIMO ATVEJIS: Atnaujinti periodinį įvykį	
Vartotojas/aktorius:	Įvykių registravimo agentas
Aprašas:	Procesas, kurio metu atnaujinama informacija periodinių įvykių sąraše
Prieš – sąlyga:	-
Sužadinimo sąlyga:	Pakoreguotas sistemos parametras
Po – sąlyga:	Periodiniai įvykiai pakoreguojami taip, kad atitiktų naujai nustatytą parametrą.

Kiekvienam panaudojimo atvejui buvo keliami funkciniai reikalavimai. Be to, visai sistemai buvo keliami ir nefunkciniai reikalavimai. Realizuota sistema tenkina jai iškeltus tiek funkcinius, tiek ir nefunkcinius reikalavimus.

4.1.1. Funkciniai reikalavimai

Žemiau pateikti sistemai iškelti funkciniai reikalavimai:

Reikalavimas #: 1		Įvykis/panaudojimo atvejis #:	1/1
Aprašymas	Sistemoje turi būti galimybė prisiregistruoti naujiems vartotojams		
Pagrindimas	Nuolatiniai sistemos vartotojai turi turėti daugiau teisių nei anonimai (neregistruoti vartotojai). Registruotas vartotojas įgauna daugiau teisių naudojantis sistema.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Vartotojas pats pateikia apie save reikalingą informaciją registracijos metu.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai:	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Reikalavimas #:	2	Įvykis/panaudojimo atvejis #:	2/1
Aprašymas	Sistemos vartotojam turi būti suteikiami unikalūs prisijungimo vardai		
Pagrindimas	Sistemos vartotoją reikia identifikuoti pagal jo vartotojo vardą.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Naujas sistemos vartotojas negali pasirinkti tokio vartotojo vardo, kuris jau yra suteiktas kitam vartotojui		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai:	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #:	3	Įvykis/panaudojimo atvejis #:	1/2
Aprašymas	Sistemoje turi būti galimybė peržiūrėti patalpintus straipsnius, kartu su jų anotacijom, vertinimais ir komentarais.		
Pagrindimas	Visi vartotojai turi turėti galimybę peržiūrėti sistemoje esančius straipsnius. Tai viena pagrindinių sistemos funkcijų.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Straipsnius gali peržiūrėti visi vartotojai, tiek ir registruoti, tiek ir anonimai.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Įtakoja visus sistemos reikalavimus	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #:	4	Įvykis/panaudojimo atvejis #:	1/3
Aprašymas	Sistema turi suteikti galimybę registruotam vartotojui prisijungti		
Pagrindimas	Prisijungęs prie sistemos vartotojas įgauna daugiau galimybių naudotis sistema. Pvz., rašyti komentarus ar vertinti straipsnį		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Prisijungti prie sistemos gali tik registruoti vartotojai, kuriems suteiktas prisijungimo vardas ir slaptažodis.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra

Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 5		Įvykis/panaudojimo atvejis #:	1/4
Aprašymas	Sistema turi suteikti galimybę registruotam vartotojui pasikeisti asmeninę informaciją		
Pagrindimas	Vartotojo asmeninė informacija laikui bėgant gali kisti, todėl turėtų būti realizuota galimybė ją atnaujinti		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Pasikeisti asmeninę informaciją nurodytą registracijos metu gali visi registruoti vartotojai, būdami prisijungę prie sistemos		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 6		Įvykis/panaudojimo atvejis #:	1/5
Aprašymas	Sistema turi suteikti galimybę registruotam vartotojui pasikeisti slaptažodį		
Pagrindimas	Saugumo sumetimas vartotojui turi būti leidžiama pasikeisti slaptažodį		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Slaptažodį pasikeisti gali tik registruoti vartotojai, prisijungę prie sistemos. Slaptažodžio pakeitimo metu būtina nurodyti tiek seną, tiek ir naują slaptažodžius.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 7		Įvykis/panaudojimo atvejis #:	1/6
Aprašymas	Registruotiems vartotojams turi būti suteikta galimybė įvertinti kito autoriaus straipsnį.		
Pagrindimas	Viena svarbiausių sistemos funkcijų. Straipsniai reitinguojami priklausomai nuo vartotojų vertinimų.		
Šaltinis	Užsakovas		

Tinkamumo kriterijus	Vartotojas gali vertinti kiekvieną straipsnį tik po vieną kartą. Autorius negali įvertinti savo straipsnio.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 8		Įvykis/panaudojimo atvejis #:	1/7
Aprašymas	Registruotiem vartotojam turi būti suteikta galimybė komentuoti straipsnį		
Pagrindimas	Vartotojam suteikiama galimybė pareikšti savo nuomonę apie straipsnį.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Komentarus gali rašyti tik prie sistemos prisijungęs vartotojas.		
Užsakovo patenkinimas	3	Užsakovo nepatenkinimas	4
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 9		Įvykis/panaudojimo atvejis #:	1/8
Aprašymas	Vartotojams turi būti suteikta galimybė pateikti straipsnius sistemai		
Pagrindimas	Tai viena svarbiausių sistemos funkcijų. Sistemoje bus publikuojami tik vartotojų pateikti straipsniai.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Straipsnius į sistemą įkelia registruoti jos vartotojai – straipsnių autoriai		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 10		Įvykis/panaudojimo atvejis #:	1/9
Aprašymas	Straipsnių autoriams turėtų būti suteikta galimybė redaguoti savo straipsnį		
Pagrindimas	Vartotojams suteikiama galimybė paredaguoti straipsnį, jei buvo pastebėta jame esant klaidų.		
Šaltinis	Užsakovas		

Tinkamumo kriterijus	Straipsnius redaguoti gali tik jų autoriai		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 11		Įvykis/panaudojimo atvejis #:	1/10
Aprašymas	Sistemoje turi būti galimybė sudaryti temų katalogą.		
Pagrindimas	Visi straipsniai sistemoje bus suskirstyti pagal temas.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Administratorius turi turėti galimybę sukurti temų sąrašą.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 12		Įvykis/panaudojimo atvejis #:	1/11
Aprašymas	Sistemoje turi būti galimybė redaguoti temų katalogą.		
Pagrindimas	Turi būti realizuota galimybė pakeisti jau sudarytas temas, jei to reikia		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Administratorius turi turėti galimybę redaguoti temų sąrašą.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 13		Įvykis/panaudojimo atvejis #:	1/12
Aprašymas	Sistemos administratorius turi turėti galimybę nustatyti sistemos parametrus.		
Pagrindimas	Vienas svarbiausių sistemos reikalavimų. Administratorius tik turi nurodyti reikiamus parametrus (pvz., archyvo sudarymo periodą), o su jais susijusius veiksmus atliks pati sistema.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Parametrus sistemai gali nustatyti tik sistemos administratorius.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5

Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 14		Įvykis/panaudojimo atvejis #:	1/13
Aprašymas	Straipsnio patalpinimas arba naujo straipsnio įvertinimas turi sužadinti naują įvykį, kurį užregistruos sistema.		
Pagrindimas	Užregistravus įvykį, sukuriamas reikiamas periodinis įvykis. Šiuo veikimo principu pagrįstas sistemos automatizavimas		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Sistema turi sureaguoti į naujo straipsnio ar naujo vertinimo įrašymą į duomenų bazę ir sukurti reikiamus periodinius įvykius.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 15		Įvykis/panaudojimo atvejis #:	1/14
Aprašymas	Sistemos agentas turi sukurti periodinius įvykius.		
Pagrindimas	Periodiniai įvykiai nurodytu metus iškvies reikiamas funkcijas, kuriomis realizuotas automatinis sistemos valdymas		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Periodiniai įvykiai sukuriami naujai patalpintam straipsniui bei vertinimui, jei reitingo skaičiavimas yra atidėtas vėlesniam laikui		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 16		Įvykis/panaudojimo atvejis #:	1/15
Aprašymas	MSSQL agentas turi nuolat peržiūrėti periodinius įvykius ir įvykdyti juos, jei atėjo tam laikas		
Pagrindimas	Periodiniai įvykiai nurodytu metus iškvies reikiamas funkcijas, kuriomis realizuotas automatinis sistemos valdymas		
Šaltinis	Užsakovas		

Tinkamumo kriterijus	Siekiant išvengti galimų klaidų, įvykių sekimui naudojamas vidinis MS SQL agentas, kuris periodiškai peržiūri būsimų įvykių sąrašą		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 17		Įvykis/panaudojimo atvejis #:	1/16
Aprašymas	Įvedus straipsnio vertinimą, sistema turi perskaičiuoti straipsnio reitingą arba iškart, arba nustatytu paros metu.		
Pagrindimas	Straipsnio reitingas priklauso nuo vartotojų vertinimų		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Straipsnio reitingas apskaičiuojamas pagal formulę, įvertinant kiek vartotojų balsavo ir kokius balus skyrė tam straipsniui.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 18		Įvykis/panaudojimo atvejis #:	1/17
Aprašymas	Geriausiai įvertinti straipsniai turi patekti į „Top“ sąrašą		
Pagrindimas	Tai vienas pagrindinių sistemos reikalavimų. Tam tikras kiekis geriausiai vartotojų įvertintų straipsnių sudaro „Top“ sąrašą		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Perskaičiavus straipsnių reitingus sistema turi, jei reikia, atnaujinti „Top“ sąrašą.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 19		Įvykis/panaudojimo atvejis #:	1/18
Aprašymas	Praėjus nustatytam laikui, naujai įvestas straipsnis turi būti pašalintas iš naujienų skilties		

Pagrindimas	Vienas svarbiausių sistemai keliamų reikalavimų. Visi nauji straipsniai turi patekti į naujienų skiltį ir joje išbūti nustatytą laiko tarpą.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Laiko tarpą, kurį straipsniai išbus naujienose, nurodo sistemos administratorius per parametrus. Straipsnių vaizdavimu naujienų skiltyje ir iš jos pašalinimu rūpinasi pati sistema.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 20		Įvykis/panaudojimo atvejis #:	1/19
Aprašymas	Sistemoje išbuvę tam tikrą laiko tarpą straipsniai patenka į archyvą.		
Pagrindimas	Vienas svarbiausių sistemai keliamų reikalavimų. Visi straipsniai, praėjus tam tikram laiko tarpui, perkeliami į archyvą.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Laiko tarpą, po kurio straipsniai bus perkeliami į archyvą, nurodo sistemos administratorius per parametrus. Straipsnių perkėlimu į archyvą rūpinasi pati sistema.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 21		Įvykis/panaudojimo atvejis #:	1/20
Aprašymas	Pakeitus sistemos parametrus, susiję periodiniai įvykiai taip pat turi būti atnaujinami.		
Pagrindimas	Vienas svarbiausių reikalavimų, kuris užtikrina teisingą automatinį įvykių valdymą		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Administratoriui pakeitus laikinius sistemos parametrus, turi būti perskaičiuojami ir dar neįvykę periodiniai įvykiai taip, kad atitiktų naujus nustatymus		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5

Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

4.1.2. Nefunkciniai reikalavimai

Žemiau pateikti sistemai iškelti nefunkciniai reikalavimai.

Reikalavimai sistemos išvaizdai (Look and feel)

Reikalavimas #:	22	Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Sistemoje spalvos turi būti suderintos tarpusavyje		
Pagrindimas	Reikalinga tam, kad nevargintų sistemos vartotojų		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Sistemos spalvos turi būti suderintos pagal nustatytus spalvų derinimo kriterijus		
Užsakovo patenkinimas	4	Užsakovo nepatenkinimas	4
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #:	23	Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Vartotojo interfeisas turi vienodai atrodyti naudojant įvairias interneto naršykles.		
Pagrindimas	Šiuo metu labiausiai paplitusios interneto naršyklės yra Internet Explorer, Firefox; informacijos atvaizdavimas šiose naršyklėse tam tikrais atvejais skiriasi.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Informacijos atvaizdavimas peržiūrint portalą šiomis naršyklėmis turi nesiskirti.		
Užsakovo patenkinimas	3	Užsakovo nepatenkinimas	3
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #:	24	Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Vartotojo sąsajoje neturi būti daug grafinės informacijos.		

Pagrindimas	Įvairi grafinė informacija, pvz. didelio formato paveiksliukai, pailgi portalo užkrovimo laiką. Tokiu atveju, vartotojams, besinaudojantiems lėtu interneto ryšiu, tektų ilgai laukti, iki užsikraus portalas bei jame esanti informacija.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Įvairūs paveiksliukai turi būti naudojami siekiant paryškinti antraštes, o ne kaip vartotojo interfeiso pagrindas.		
Užsakovo patenkinimas	2	Užsakovo nepatenkinimas	2
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Reikalavimai panaudojamumui (Usability)

Reikalavimas #: 25		Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Galimybė naudotis portalu naudojant tiek pelę, tiek ir vien klaviatūros pagalba.		
Pagrindimas	Dalis vartotojų įpratę naudotis vien tik klaviatūra, net ir peržiūrėdami internetinius portalus. Pelė tam tikrais atvejais labai lėtina darbo greitį.		
Šaltinis	Visi portalo valdymo mygtukai turi būti pasiekiami tiek naudojantis pele, tiek ir naudojantis klaviatūra („Tab“ klavišas).		
Tinkamumo kriterijus	Užsakovas		
Užsakovo patenkinimas	2	Užsakovo nepatenkinimas	2
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		
Reikalavimas #: 26		Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Sistema turi būti paprasta, vartotojams turi būti keliami minimalūs kompiuterinių žinių reikalavimai		
Pagrindimas	Vartotojai sistema naudosis per interneto naršyklę		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Sistemos prieiga per interneto naršyklę nereikalauja ypatingų kompiuterinių žinių		

Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Reikalavimas #:	27	Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Sistemos vartotojo interfeisas turi būti toks, kad vartotojas galėtų intuityviai nujausti, kaip elgtis sus sistema.		
Pagrindimas	Vartotojai mieliau naudojasi sistemomis, kurios yra jiem įprastos išvaizdos bei valdymo.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Vartotojas ir be pagalbos dokumento turi sugebėti naudotis pagrindinėmis programos funkcijomis.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Reikalavimas #:	28	Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Maža vartotojo klaidos kaina		
Pagrindimas	Vartotojo sukeltos klaidos neturėtų sugadinti sistemos ar kitaip pakenkti sistemai ar joje esantiems duomenims.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Jei vartotojas padaro klaidą, sistema turi jį apie tai informuoti, ir, jei įmanoma, leisti tą klaidą ištaisyti. Sistema bent minimaliai turi numatyti galimas vartotojo klaidas.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Reikalavimai vykdymo charakteristikoms (Performance)

Reikalavimas #:	29	Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Sistema turi atitikti minimalius darbo vietai keliamus greičio		

	reikalavimus		
Pagrindimas	Reikalinga tam, kad darbas su sistema būtų efektyvus		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Portalo puslapio užkrovimo laikas esant lėtam interneto ryšiui, neskaitant paieškos, neturėtų užtrukti ilgiau nei 15sec.		
Užsakovo patenkinimas	3	Užsakovo nepatenkinimas	3
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Reikalavimai sistemos priežiūrai (Maintainability and portability)

Reikalavimas #:	30	Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Sistema turi būti atnaujinama pasikeitus technologijoms, kurios blogai paveiktų sistemos kokybės charakteristikas		
Pagrindimas	Reikalinga, pritaikant sistemą prie pasikeitusios aplinkos		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Sistema tenkina funkcinius ir nefunkcinius reikalavimus		
Užsakovo patenkinimas	3	Užsakovo nepatenkinimas	3
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Reikalavimai saugumui (Security)

Reikalavimas #:	31	Įvykis/panaudojimo atvejis #:	1/1-16
Aprašymas	Vartotojams turi būti užtikrintos atitinkamos sistemos prieigos teisės		
Pagrindimas	Reikalinga tam, kad užtikrintų tinkamą sistemos naudojimą		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Vartotojai, naudojantis sistema, turi prieiti tik prie jiem suteiktų funkcijų.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		

Istorija:	Užregistruotas 2006-03-08

Kultūriniai-politiniai reikalavimai

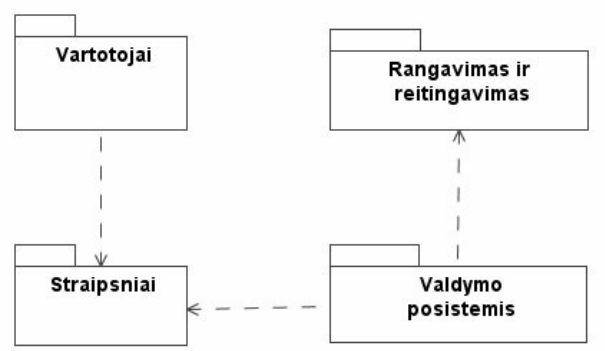
Reikalavimas #: 32		Įvykis/panaudojimo atvejis #:	1/1-16
Aprašymas	Sistema turi būti pritaikyta Lietuvos ir Europos specifikai		
Pagrindimas	Sistema naudosis Lietuvos gyventojai		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Sistemoje nėra nusižengimų galiojantiems papročiams, taisyklėms, etikai ir pan.		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

Teisiniai reikalavimai

Reikalavimas #: 33		Įvykis/panaudojimo atvejis #:	-/-
Aprašymas	Sistema turi tenkinti Lietuvoje galiojančius programų kūrimo įstatymus.		
Pagrindimas	Sistema naudosis Lietuvos gyventojai.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Sistema turi nepažeisti įstatymų		
Užsakovo patenkinimas	5	Užsakovo nepatenkinimas	5
Priklausomybės:	Nėra	Konfliktai	Nėra
Papildoma medžiaga:	Nėra		
Istorija:	Užregistruotas 2006-03-08		

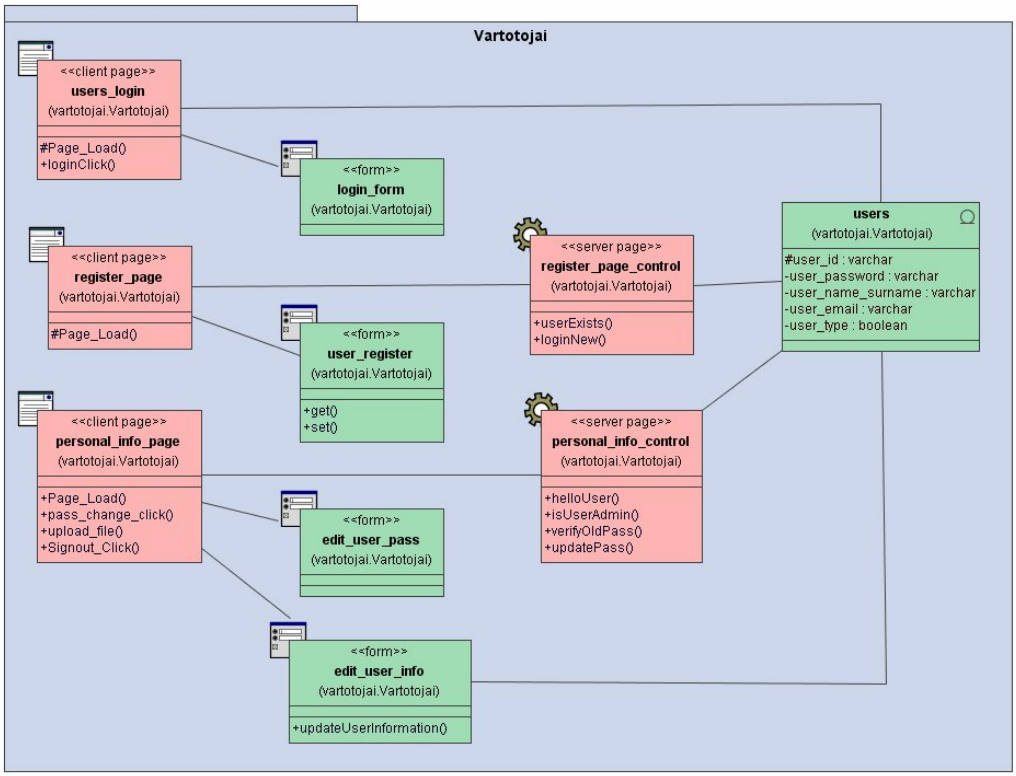
4.2. Portalo architektūra

Sistemos suskaidymas į paketus pavaizduotas 18 pav.



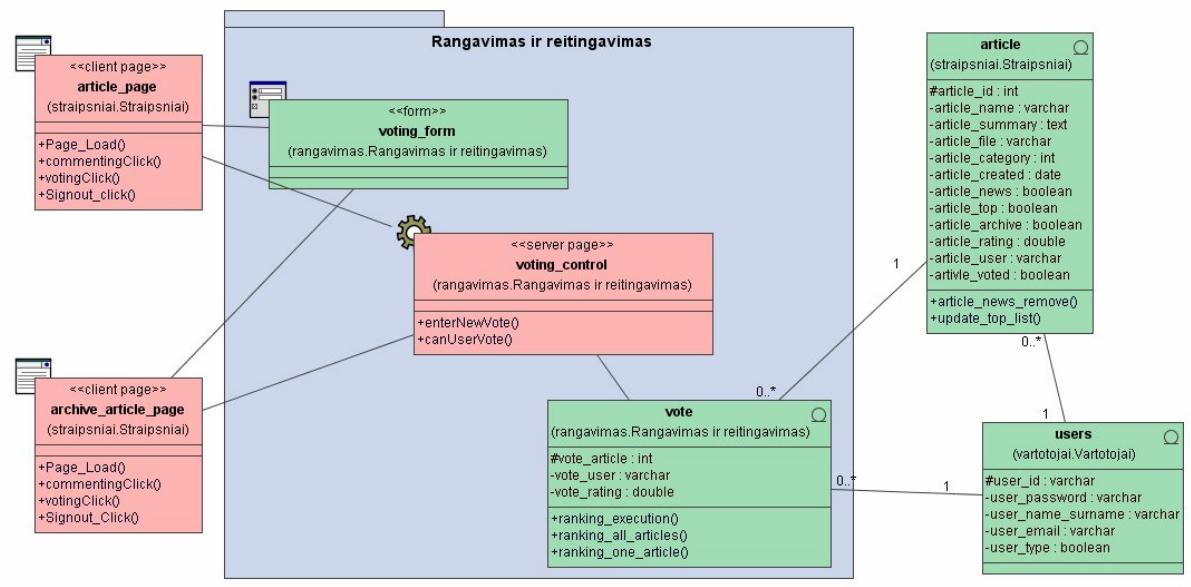
18 pav. Sistemos paketai

- Paketas „Vartotojai“. Paketas atsakingas už vartotojų registraciją ir prisijungimą prie sistemos. Paketo „Vartotojai“ klasių diagrama pateikta 19 pav.



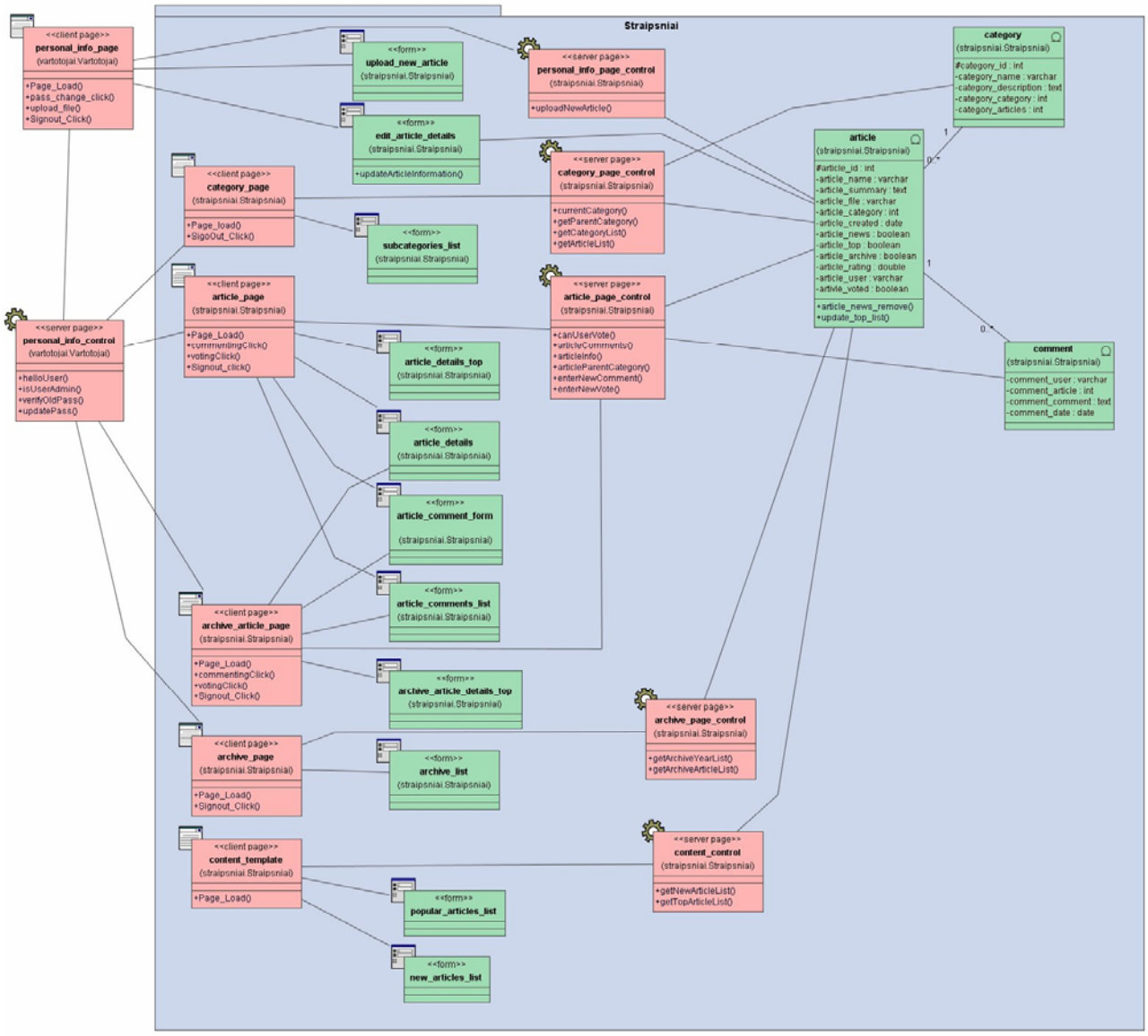
19 pav. Paketo „Vartotojai“ klasių diagrama

- Paketas „Rangavimas ir reitingavimas“ atsakingas už straipsnių reitingo skaičiavimą. Paketo „Rangavimas ir reitingavimas“ klasių diagrama pavaizduota 20 pav.



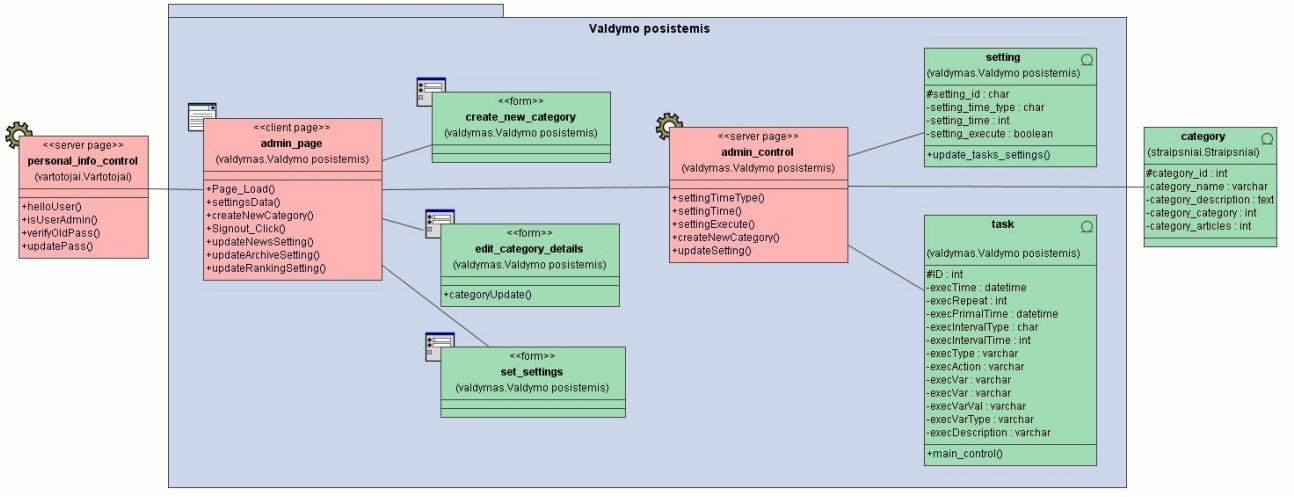
20 pav. Paketo „Rangavimas ir reitingavimas“ klasių diagrama

- Paketas „Straipsniai“ atsakingas už straipsnių saugojimą, atvaizdavimą, peržiūrą ir komentavimą. Paketo klasių diagrama pavaizduota 21 pav.



21 pav. Paketo „Straipsniai“ klasių diagrama

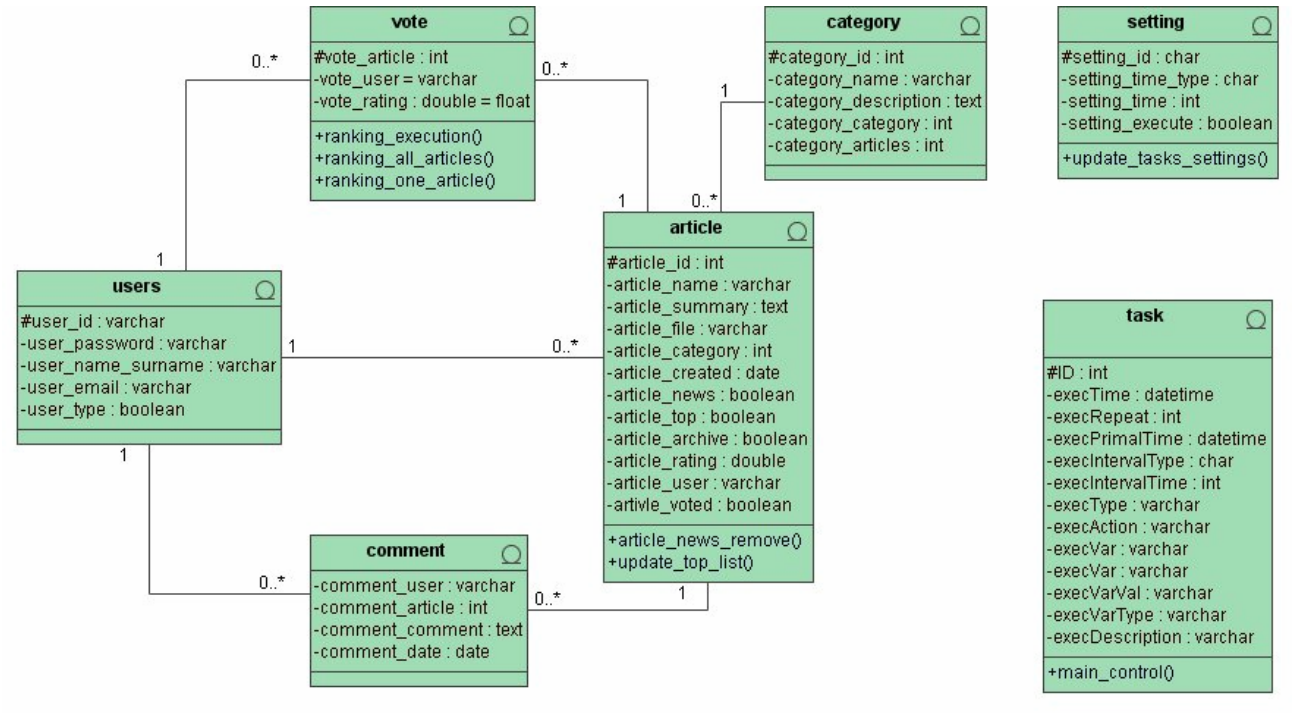
- Paketas „Savaiminio atsinaujinimo posistemis“ atsakingas už temų katalogo koregavimą ir savarankišką informacijos perrūšiavimą. Visa informacija, reikalinga atlikti šiam perrūšiavimui, saugoma duomenų bazėje, settings ir task lentelėse. Informaciją parametrų lentelėse gali pakoreguoti tik sistemos administratorius. Paketo klasių diagrama pavaizduota 22 pav.22 pav.



22 pav. Paketo „Savaiminio atsinaujinimo posistemis“ klasių diagrama

4.3. Duomenų modelis

Duomenų modelis pateikiamas 23 pav.

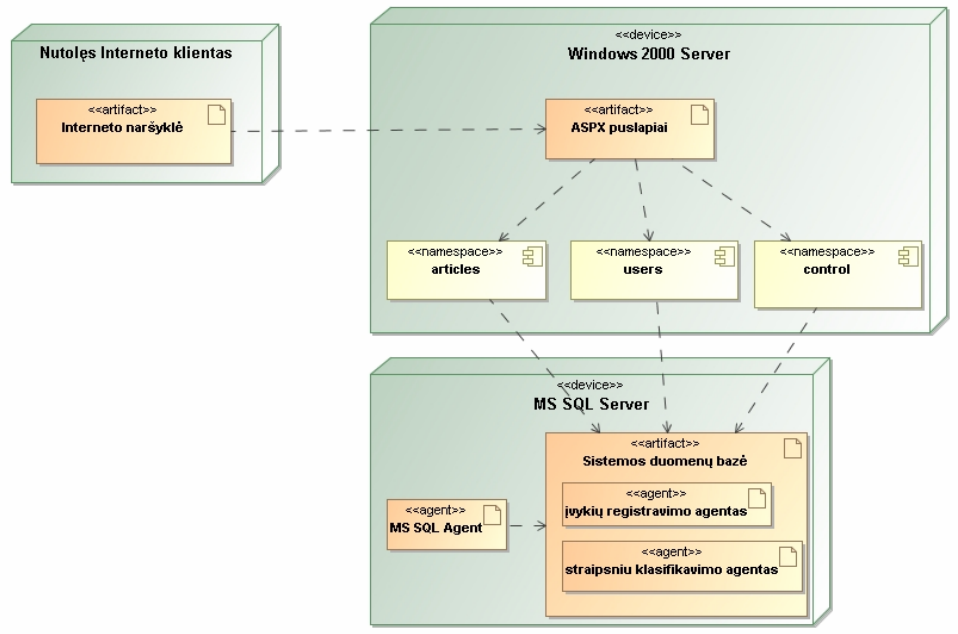


23 pav. Duomenų modelis

4.4. Realizacijos modelis

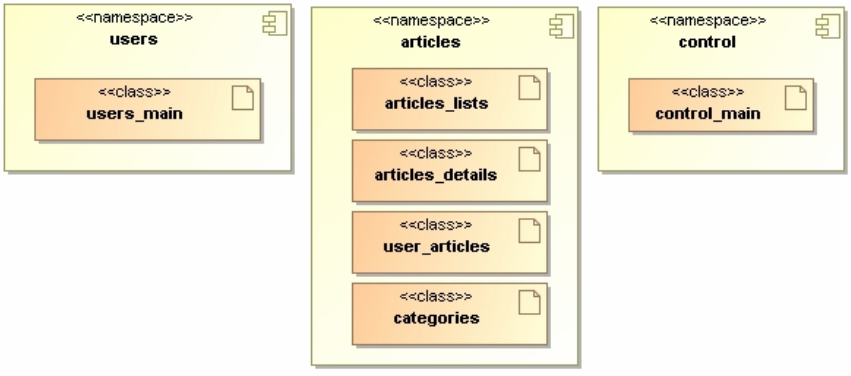
Siekiant sistemą padaryti lengviau plečiamą ir modifikuojamą, sistema buvo realizuota pagrindines funkcijas, reikalingas portalo veikimui, aprašant kiek įmanoma labiau viena nuo kitos

nepriklausomomis C# kalbos klasėmis, o funkcijas, susijusias su agentų veikimu, iškeliant į duomenų bazę. Sistemos išdėstymo modelis pateikiamas 24 pav.



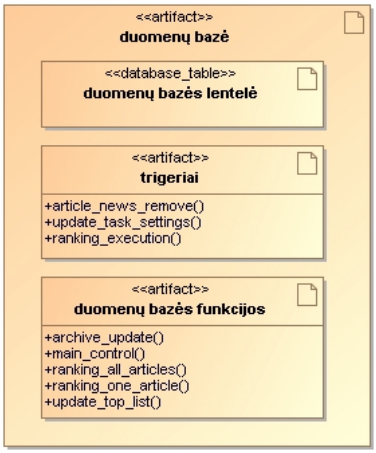
24 pav. Sistemos išdėstymo modelis

Pagrindinės klasės, realizuotos C# kalboje pateikiamos 25 pav. Namespace *users* sudarytas iš vienos klasės *users_main*. Klasėje realizuotos visos portalo funkcijos, susijusios su vartotojais: vartotojų registracija, prisijungimas ir pan. Namespace *articles* sudarytas iš trijų klasių: *articles_lists, articles_details* ir *categories*. *Articles_list* klasėje realiuzotos funkcijos, susijusios su straipsnių suradimu, pavyzdžiui Top sąrašas, naujienų sąrašas. Klasėje *articles_details* realizuotos funkcijos, susijusios su konkretaus straipsnio informacija. Tai straipsnio detalės, komentarai, vertinimas. Klasėje *categories* realizuotos funkcijos, susijusios su kategorijų kūrimu ir valdymu. Namespace *control* sudarytas iš klasės *control_main*. Šioje klasėje realizuotos funkcijos, susijusios su sistemos parametrų valdymu.



25 pav. Pagrindinės sistemos klasės

Su straipsnių rangavimu bei sistemos agentais susijusios funkcijos yra realizuotos duomenų bazėje. Funkcijos, realizuotos išorinėmis bei vidinėmis duomenų bazės funkcijomis, pavaizduotos 26 pav.



26 pav.duomenų bazėje realizuotos funkcijos

Funkcijų, realizuotų duomenų bazėje, aprašymas:

- *archive_update* – procedūra skirta straipsnių archyvo atnaujinimui
- *main_control* – procedūra, atsakinga už periodinių įvykių valdymą. Funkcijos metu peržiūrima užduočių lentelė ir įvykdoma užduotis, kuriai atėjo laikas būti įvykdytai. Užduoties vykdymo metu arba iškviečiama kita nurodyta procedūra (pvz., *ranking_all_articles*), arba įvykdomas nurodytas SQL sakinys (pvz., SQL sakinys nurodytam straipsniui šalinti iš naujienų). Už procedūros *main_control* vykdymą atsakingas MS SQL Agentas, kuris realizuotas MS SQL Serveryje ir šią procedūra pakartotinai iškviečia jam nustatytu metu.
- *ranking_all_articles* – procedūra, atsakinga už straipsnių reitingo perskaičiavimą. Procedūra kiekvienam straipsniui, kurio reitingas nėra perskaičiuotas, iškviečia funkciją *ranking_one_article* ir pasibaigus skaičiavimams iškviečia straipsnių Top sąrašo perrūšiavimo funkciją *update_top_list*.
- *ranking_one_article* – išorinė funkcija, parašyta .NET C# kalba ir įtraukta į duomenų bazę. Funkcija atsakinga už nurodyto straipsnio reitingo perskaičiavimą.
- *update_top_list* – funkcija kuri atnaujina Top straipsnių sąrašą.

4.5. Veikimo aprašymas

Visą realizuotą eksperimentinę sistemą galima būtų suskirstyti į dvi dideles dalis. Pirmoji dalis – tai informacijos portalas, turintis visus pagrindinius reikalavimus keliamus internetiniam portalam. Portale realizuotas straipsnių talpinimas per peržiūra, jų vertinimas ir komentavimas. Taip pat realizuotos keturių lygių vartotojų teisės:

- Neregistruotas vartotojas,
- Registruotas vartotojas
- Autorius
- Administratorius

Šis portalas buvo realizuotas naudojantis .NET C# kalba ir MSSQL duomenų baze. Vartotojo sąsaja su sistema buvo realizuota naudojant ASP. Toks portalas yra gana paprastas, nereikalaujantis nei naujausių .NET ar MSSQL Server programinės įrangos versijų, nei daug žinių programavimo srityje. Tačiau tam, kad parodyti, kaip galima palengvinti portalo administratoriaus darbą ir išplėsti tokį paprastą internetinį portalą, jame buvo realizuotas automatinių įvykių valdymas. Ši technologija jau reikalauja ne tik naujausių .NET bei MSSQL Server programinės įrangos versijų, bet ir žinių programavimo srityje.

Sistemos realizacijos metu, visos pagrindinės funkcijos, susijusios su automatiniu sistemos valdymu, buvo realizuotos duomenų bazėje kaip trigeriai ar procedūros. Tam, kad būtų galima sekti laiką ir įvykius, kurie turi įvykti nurodytu metu, visų dar neįvykusių įvykių aprašymai yra saugomi duomenų bazės lentelėje „task“. Šioje lentelėje saugomi ne tik laikai, kada įvykiai turi įvykti, bet ir nurodymai, ką reikia atlikti. Šie nurodymai gali būti dviejų tipų:

- Sql sakinys, kuris bus įvykdomas atėjus numatytam laikui
- Procedūros pavadinimas, kuri bus iškviesta atėjus numatytam laikui. Procedūros, savo ruožtu, gali būti dviejų tipų:
    - Vidinė procedūra, parašyta duomenų bazėje naudojant SQL kalbą
    - Išorinė procedūra, parašyta naudojant .NET C# programavimo kalbą ir vėliau įtraukta į duomenų bazę.

Už naujų užduočių įrašymą į duomenų bazės lentelę „task“, atsakingi sistemos agentai, realizuoti trigeriais duomenų bazės lentelėse. Šie agentai, naujos informacijos įvedimo metu, sukuria naują įvykį, jo aprašymą patalpindami duomenų bazės lentelėje „task“. Kadangi MSSQL 2005 versijoje buvo patobulintas MSSQL agentas, šis agentas, žinoma, buvo panaudotas ir realizuojamoje

sistemoje. MSSQL agentas atsakingas ne tik už periodinį „task“ duomenų lentelės peržiūrėjimą ir analizavimą, bet ir nustačius, kad atėjo laikas vykdyti kažkurią iš užduočių, jos įvykdymą.

Kadangi visos funkcijos, realizuotos duomenų bazėje vyksta žymiai greičiau, nei realizuotos kuria nors programavimo kalba už duomenų bazės ribų, jos nesulėtina portalo veikimo, kuomet portalu naudojasi vartotojai. Be to, vartotojų reitingų perskaičiavimą taip pat naudinga realizuoti pačioje duomenų bazėje ir dėl dar vieno atvejo: jei reitingų skaičiavimo funkcija yra labai sudėtinga ar reikalaujanti daugelio vartotojų rezultatų palyginimo, labai naudinga tokius skaičiavimus atidėti vėlesniam laikui, ir vykdyti, tarkim naktį. Šiuo atveju automatinis įvykių valdymas tampa tikru išsigelbėjimu, nes sistemos administratoriui visai nereikia tuo rūpintis, o tik nurodyti, kuriuo paros metu įvykdyti perskaičiavimą.

Žemiau pateikiama keletas sukurto eksperimentinio portalo langų, iliustruojančių sistemos veikimą bei valdymą.

27 pav. Pagrindinis sistemos langas

28 pav. Sistemos kategorijų ir jose esančių straipsnių peržiūra

29 pav. Straipsnio informacija, matoma registruotam sistemos vartotojui

30 pav. Registruotam vartotojui suteikiamos papildomos funkcijos darbui su sistema.

31 pav. Sistemos administratoriui suteikiamos funkcijos darbui su sistema

32 pav. Sistemos parametrų nustatymas

5. Metodikos tyrimas

5.1. Taikymo galimybių tyrimas

Daugeliu paprastų sistemų atveju agentai nėra būtini. Tačiau jei yra pageidavimas tam tikrų įvykių vykdymą atidėti vėlesniam laikui, agentai tampa tikrai reikalingi. Be to, jei sistemą norime padaryti kiek įmanoma lankstesnę ir kuo labiau konfigūruojamą, agentai ne tik palengvina sistemos kūrėjų darbą, bet ir leidžia sistemai veikti greičiau.

Elementarus pavyzdys, kaip kad galimybė vartotojui įvertinti sistemoje esantį straipsnį. Žinoma, šį funkcionalumą sistemoje įmanoma realizuoti dviem būdais: su agentais ir be jų. Jei sistemoje straipsnio vertinimas realizuojamas be agentų, galimas dvejopas sistemos realizavimas:

1) Sistema įrašo naują vertinimą į duomenų bazę; sistema nuskaito esamus vertinimus iš duomenų bazės; sistema perskaičiuoja straipsnio reitingą; sistema atnaujina straipsnio reitingą;
2) Sistema įrašo naują vertinimą į duomenų bazę; sistema iškviečia duomenų bazėje esančią procedūrą, kuri atsakinga už straipsnio reitingo perskaičiavimą ir atnaujinimą.

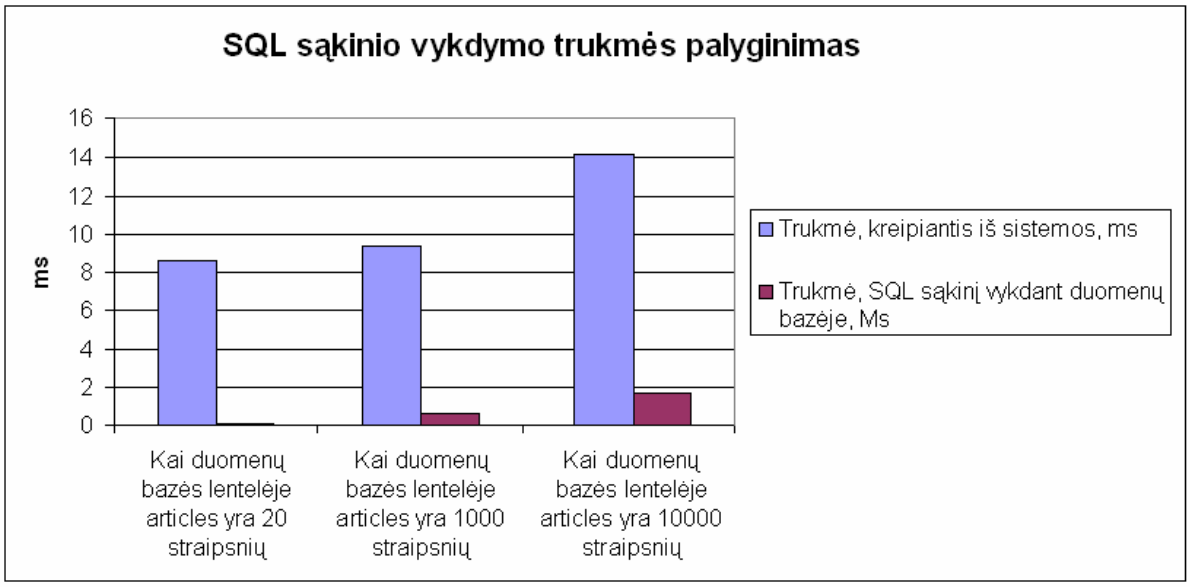
Pirmu atveju, sistema atlieka tris, antru atveju – du kreipinius į duomenų bazę. Ne paslaptis, kad šie kreipimaisi užima laiko, kuris priklauso ir nuo siunčiamos užklausos ilgumo bei sudėtingumo. Natūraliai kyla klausimas, kaip tokiu atveju elgiasi sistema, kuri yra realizuota remiantis agentais? Šiuo atveju galimas vienas kelias: sistema įrašo informaciją į duomenų bazę. Duomenų bazėje realizuotas agentas-trigeris iškviečia duomenų bazės procedūrą, kuri rūpinasi reitingo perskaičiavimu ir atnaujinimu. Negana to, agentai suteikia galimybę tam tikrus veiksmus atidėti vėlesniam laikui. Taigi, agentas-trigeris, priklausomai nuo sistemoje nurodytų parametrų gali arba įvykdyti reitingo perskaičiavimą tuoj pat, arba atidėti jį nurodytam laikui. Būtent toks tam tikrų įvykių vykdymo atidėjimas naudingas tada, jei atliekami sudėtingi skaičiavimai, kurie ne tik apkrauna sistemą, bet ir skaičiavimam atlikti reikalauja kitos informacijos iš nurodyto laiko intervalo. Be to, net jei atliekami nesudėtingi skaičiavimai, kurie užima panašiai laiko tiek skaičiavimus atliekant sistemoje, tiek ir duomenų bazėje, realizavus sistemą pasinaudojus agentais, yra sutaupoma šiek tiek laiko, kuris būtų sueikvotas sistemai papildomai kreipiantis į duomenų bazę. Be to, kiekvienas skaičiavimas atliekamas sistemos apkrauna vartotojo sąsają, priešingai nei skaičiavimai atliekami duomenų bazės lygyje.

Atlikime paprastą eksperimentą ir pasižiūrėkime, kiek laiko užtrunka įvykdyti paprastą SQL sakinį duomenų bazėje, ir kiek užtrunka jį įvykdyti iš sistemos (1 lentelė):

1 lentelė. SQL sakinio vykdymo trukmės palyginimas

SELECT count(*) FROM *articles*	**Trukmė, kreipiantis iš sistemos, ms**	**Trukmė, SQL sakinį vykdant duomenų bazėje, Ms**
Kai duomenų bazės lentelėje *articles* yra 20 straipsnių	8,6	0,1
Kai duomenų bazės lentelėje *articles* yra 1000 straipsnių	9,4	0,6
Kai duomenų bazės lentelėje *articles* yra 10000 straipsnių	14,2	1,7

Grafikas, nubraižytas pagal 1 lentelėje pateiktus rezultatus, pavaizduotas 33 pav.



33 pav. SQL sakinio vykdymo trukmės palyginimas

Tiek lentelėje, tiek ir diagramoje pavaizduoti rezultatai – tai atliktų skaičiavimų aritmetinis vidurkis.

Aritmetinis vidurkis (angl. *arithmetic mean*) – tai vidurkis, skaičiuojamas sudedant visas kiekybinio kintamojo reikšmes ir padalijant šią sumą iš reikšmių skaičiaus **[1]**:

Aritmetinis vidurkis - $$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Vis dėl to, vidurkis ne visada atspindi tikrąją situaciją, kuriai įtakos turi ir rezultatų skirtumas, priklausantis nuo aplinkos bei sistemos apkrovimo tuo momentu. Todėl reikėtų įvertinti ir standartinį nuokrypį, kuris nusako kaip plačiai yra pasklidusios reikšmės, rodo kiek vidutiniškai reikšmės nukrypsta nuo vidurkio.

Standartinis nuokrypis (angl. *standard deviation*) – tai tiriamojo požymio reikšmių sklaidos apibūdinimas, apibrėžiamas kaip požymio įgyjamų reikšmių ir vidurkio skirtumų kvadratų sumos vidurkis (įprasta žymėti *s* arba *SD*). Matematiniais terminais apibrėžiant standartinio nuokrypio sąvoką, galima būtų pasakyti, jog tai yra variacijos kvadratinė šaknis **[1]**:

Variacija - $V= \sum(x_i - x)^2/n-1$

Standartinis nuokrypis $SD = \sqrt{\sum(x_i - x)^2/n-1}$

Kadangi labiausiai realios sistemos situaciją atspindi tyrimai atlikti su didesne duomenų baze, standartinį nuokrypį paskaičiuokim tik tam atvejui, kai duomenų bazės lentelę sudaro 10000 straipsnių. Standartinis nuokrypis atliekant kreipinį iš sistemos yra 12,2 ms, o atliekant duomenų bazėje 3,8 ms

Kaip matome, skirtumas tarp SQL sakinio įvykdymo laikų, kuomet jis vykdomas tiesiogiai duomenų bazėje ir kreipiantis iš programoms, visada egzistuoja. Žinoma, šis laikas gali kisti priklausomai nuo serverio charakteristikų bei apkrovimo skirtingose aplinkose. Tačiau palyginus trukmes galima daryti išvadą, jog tą patį SQL sakinį vykdant kreipiniu iš sistemos, didžiausią laiko dalį užima sistemos kreipimasis į duomenų bazę bei rezultato grąžinimas sistemai. Ir šis laikas, atlikto bandymo metu prie 10000 duomenų kiekio lentelėj, buvo apytiksliai lygus 12,5 ms. Įvertinus standartinį nuokrypį matome, kaip stipriai laikus gali įtakoti sistemos apkrovimas bei joje vykstantys įvairūs procesai. Natūralu, jog kuriant dinamiškas ir greitas informacines sistemas, tokių, atrodytų nereikalingų, laiko gaišimų reikėtų kaip įmanoma labiau vengti. O tai padaryti lengviausia mažinant sistemos kreipinių į duomenų bazę skaičių. Būtent tai padaryti mums ir padeda sistemos agentai, realizuoti duomenų bazėse trigerių ir procedūrų pagalba.

5.2. Kūrimo ir modifikavimo galimybių tyrimas

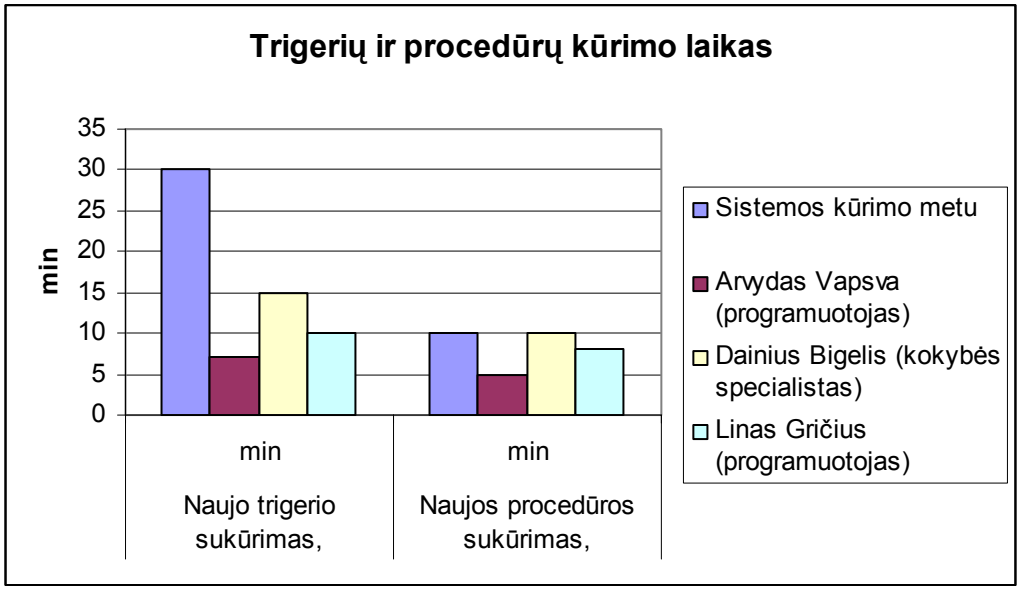
Vienas svarbiausių faktorių, kuriamose sistemose yra galimybė jas vėliau plėsti bei modifikuoti. Šiuo atveju, kai sistema sukurta naudojantis .NET technologijomis ir MSSQL duomenų baze bei joje realizuotais sistemos automatiniais agentais, sistemos plėtimas bei modifikavimas yra apribojamas šių technologijų galimybėmis.

Sistemos funkcionalumo plėtimas/sukūrimas. Sistemos funkcionalumo keitimui arba plėtimui reikalingos .NET C#, SQL bei pagrindinės HTML ir CSS žinios. Sistemos funkcionalumo kūrimo bei keitimo metu turės būti keičiamos arba sukuriamos naujos klasės, realizuojamos arba redaguojamos duomenų bazės procedūros, keičiama arba naujai sukuriama vartotojo sąsaja. Būtent funkcionalumo keitimo ar naujo funkcionalumo realizavimo paprastumą sunku įvertinti todėl, kad tai nulemia norimo funkcionalumo sudėtingumas. Žinoma, jei funkcionalumas realizuojamas .NET C# kalba, labai pasitarnauja tai, jog naudojama programinė įranga yra Microsoft kompanijos, todėl vartotojam prieinama ne tik daug pagalbinės informacijos, bet ir .NET realizuoti pagalbininkai (angl. *Wizard*). Jei sistemos išplėtimas apsiriboja keletu naujų elementų vartotojo sąsajoje įdiegimu, tai galima padaryti per keletą minučių pasinaudojus pagalbininkais. Jei kuriamas visiškai naujas funkcionalumas .NET C# kalba arba rašomos SQL procedūros, laiko sąnaudas įvertinti tampa sudėtinga ir šis procesą galima prilyginti projekto realizacijos trukmės planavimui.

Naujų agentų sukūrimas. Tokiu atveju, kai agentai yra kuriami naudojantis duomenų baze, naujų agentų kūrimui yra būtinos SQL ir T-SQL žinios. Mano siūlomu agentų realizavimo būdu, agentai tik surašo būsimus veiksmus į duomenų bazės lentelę, o veiksmų vykdymu reikiamu metu rūpinasi MS SQL Agent, jau realizuotas MS SQL serverio sistemoje. Tokiu atveju, naujų agentų kūrimas apsiriboja trigerių bei procedūrų sukūrimu. Žemiau pateiktoje 2 lentelėje pavaizduota, kiek laiko užtrunka sukurti paprastą trigerį ir procedūrą tiek programuotojam, tiek programavimo žinių pagrindus teturintiem vartotojam. Kad būtų lengviau išmatuoti laikus, buvo laikoma, kad naujas trigeris ir procedūra kuriami paprasti ir naudojant MS SQL Server administravimo programos pateikiamais šablonais. Trigeryje realizuojamas naujos užduoties įterpimas į užduočių lentelę (eksperimento dalyviui kaip pavyzdys pateikiamas panašus jau realizuotas trigeris), o procedūroje realizuojamas paprastas SQL sakinys.

2 lentelė. Naujo trigerių ir procedūrų sukūrimo bei redagavimo trukmės palyginimas

		Naujo trigerio sukūrimas, min.	**Naujos procedūros sukūrimas, min.**
Sistemos kūrimo metu	Birutė Meilutytė	30	15
Sistemos modifikavimo metu	Arvydas Vapsva (programuotojas)	7	5
	Dainius Bigelis (kokybės specialistas)	15	10
	Linas Gričius (programuotojas)	10	8



34 pav. Trigerių ir procedūrų kūrimo laikų palyginimas

Kaip matome iš rezultatų, trigerį, įterpiantį užduotis į užduočių lentelę, ilgiausiai kurti reikia tik pirmą kartą. Taip yra todėl, jog trigerio metu didžiausias dėmesys skiriamas užduoties vykdymo laikui apskaičiuoti, kuris priklauso nuo administratoriaus parinktų nustatymų. Kuriant tokį trigerį pakartotinai, galima remtis jau anksčiau parašytu laiko apskaičiavimo algoritmu, o tai labai palengvina kūrimą. Tuo tarpu paprastos procedūros kūrimas užtrunka panašiai laiko visais atvejais. Žinoma, procedūrų sudėtingumas realybėje gali labai kisti, o tai ir nulems jų kūrimo trukmę.

Taip pat reikėtų paminėti, jog tokiu būdu realizavus sistemą, agentus galima kurti ir modifikuoti neliečiant pačios sistemos bei .NET C# kalba parašytų klasių ar vartotojo sąsajos. Šis teiginys taip pat galioja modifikuojant ir realizuojant agentų atliekamus veiksmus, kurie yra realizuoti SQL vidinėmis bei išorinėmis procedūromis. Kadangi programos kodas yra tiesiogiai neliečiamas, labai sumažėja tikimybė pakenkti sistemos funkcionalumui bei klaidų atsiradimui sistemos atnaujinimo metu. O tai vienas svarbesnių reikalavimų sistemos kokybės atžvilgiu.

5.3. Metodikos įvertinimas

Analizuojant galimybę kurti save palaikančias internetines informacines sistemas, buvo apžvelgta keletas priemonių, tinkamų kurti tokiom sistemom. Kadangi šiuo metu vienas svarbiausių reikalavimų internetinėm sistemom, tokiom kaip žinių portalai, yra jų greitis ir lankstumas realizavimui naudinga rinktis agentus, realizuotus duomenų bazėje. Agentai įgyvendinami duomenų

bazėse juos realizuojant trigeriais bei vidinėmis duomenų bazės funkcijomis. Kaip parodė atliktas taikymo galimybių tyrimas, toks realizacijos metodas suteikia galimybę sutaupyti sistemos vykdymo laiko, nes išvengiama daugkartinių sistemos kreipinių į duomenų bazę.

Vienas didesnių šios metodikos trūkumų yra tas, kad sistemos kūrėjai gali įvelti klaidų kuriant agentus. Jei sistemos veikimas ir agentai nėra pakankamai gerai išanalizuoti, sukurti trigeriai gali persikirsti su savo atliekamais veiksmais ar dubliuoti vienas kitą. Taip pat, skaitant atsiliepimus įvairiuose forumuose apie trigerių ir duomenų bazių vidinių funkcijų naudojimą, teko susidurti su dvejopa nuomone. Dauguma programuotojų sutinka, jog tai greičiau veikiantis realizacijos metodas, tačiau dėl to, ar šiuo būdu realizuoti sistemą yra greičiau bei lengviau palaikyti jau esamą, nuomonės tarp programuotojų išsiskiria. Kol vieni teigia, jog naudojant paprastą objektinę programavimo kalbą realizuoti sistemas yra greičiau ir paprasčiau, be to lengviau ką nors keisti ateity, kiti akcentuoja, jog realizavus pagrindinius akcentus duomenų bazėje, vėliau informaciją reikia redaguoti tik vienoje procedūros vietoje, o ne ieškoti tų pačių SQL sakinių visame programos kode. Visgi reikia pripažinti, jog dabartinių duomenų bazių galimybės vis dar yra naujovė, o naujovėm įsitvirtinti programavimo srityje ne visada būna lengva.

Kuriant nedideles informacines sistemas, agentų veiksmus galima lengvai išanalizuoti paprastomis UML diagramomis ir juos realizuoti duomenų bazėse. Tokiu būdu ne tik sistemos analizė ir projektavimas užims trumpiau laiko, bet ir realizuotos sistemos veikimas bus tikrai greitas. Vienas iš pavojų, kylančių realizuojant dideles sistemas – klaidų įvėlimas. Todėl būtina sistemą gerai išanalizuoti, protingai atlikti dekompoziciją bei kurti moduliniu būdu. Taigi, reikia gero projekto, kuriame atsispindėtų viskas – pradedant elementariais sistemos veiksmais, baigiant agentų atliekamais veiksmais bei jų tarpusavio bendravimu. Jei sistemoje egzistuoja daug agentų, tikslinga juos projektuoti grupuojant į pogrupius, lygiai taip pat, kaip kad didelę sistemą yra tikslinga išskaidyti į posistemius. Kai kas iš programuotojų teigia, jos pačių trigerių naudojimas sistemoje yra trūkumas, nes trigeriai yra nematomi, realizuojami paslėpti duomenų bazėje ir tai dažniausias klaidų sistemoje šaltinis. Tačiau jei agentai yra gerai suprojektuojami, žinomi jų scenarijai, šių klaidų programuojant galima ir išvengti.

6. Išvados

1. Portalų reikalavimų analizė parodė, kad portalai turi būti saugūs, o reikiama informacija juose turi būti lengvai pasiekiama ir randama. Tačiau vienas svarbiausių portalam keliamų reikalavimų yra tas, kad informacija juose turi būti pastoviai patikrinama ir atnaujinama.
2. Siekiant sukurti šiuolaikišką, išoriniams ir vidiniams vartotojams patogų informacinį portalą, išanalizuotos aktyvių, save administruojančių sistemų kūrimo galimybės. Prieita išvados, kad geriausia šiam tikslui fiksuoti į sistemą patenkančius įvykius ir sukurti į juos reaguojančius programų komponentus – agentus, kurie atliktų dalį sistemos administratoriaus užduočių: perskirstytų informaciją pagal įvertinimus ir laiko faktorius.
3. Siekiant išbandyti naujas MS SQL Server siūlomas galimybes dinamiškam įvykių apdorojimui, realizacijai buvo pasirinkta Microsoft kompanijos siūloma programinė įranga - .NET C# programavimo kalba bei MS SQL Server. Programinių agentų kūrimui buvo nuspręsta naudoti duomenų bazę ir agentus realizuoti trigerių pagalba, nes tai paspartina programinės įrangos veikimą ir neapkrauna vartotojo sąsajos. O tai labai svarbu siekiant kurti greitas informacines sistemas, atitinkančias Web 2 keliamus tikslus.
4. Literatūros analizės pagrindu sudaryta agentų projektavimo metodika ir pritaikyta standartiniams UML elementams, kadangi tuomet ji lengvai bus integruojama į praktikoje naudojamus kūrimo procesus ir pritaikoma didesniam vartotojų ratui.
5. Eksperimentinės sistemos tyrimas parodė, kad naudojant SQL procedūras pasiekiamas didesnis greitis negu realizuojant viską sistemoje, o agentų atnaujinimui ar papildymui sugaištama bent dvigubai mažiau laiko.
6. Atlikus metodikos tyrimą galima teigti, jog pagrindiniai šios metodikos privalumai yra greitis ir paprastas sistemos keičiamumas. Tačiau norint taikyti šią metodiką kuriant dideles ir sudėtingas informacines sistemas, agentų sąveikų valdymas taptų sunkesnis ir reikėtų papildomų automatizavimo priemonių jų suderinamumui užtikrinti.

7. Terminų ir santraukų žodynas

angl. - terminas anglų kalba

DB – duomenų bazė

DBVS – duomenų bazių valdymo sistemos

GUI – grafinė vartotojo sąsaja

min. - minutės

ms – milisekundės

8. Literatūros sąrašas

[1] Bačinskas, A; Janilionis, V; ir Jokimaitis, A. *Tikimybių teorija ir statistika.* Kaunas, 2005.

[2] Lupeikienė, A. *Agentinių technologijų panaudojimo reikalavimai kuriant komponentines verslo, informacines ir programų sistemas*. Kaunas, 2003, VI-16-VI-22 p.

[3] Chunsheng Li; Zili Zhang; ir Chengqi Zhang. A Platform for Dynamic Organization of Agents in Agent-Based Systems. *International Conference on Intelegent Agent Technology*, 2004. ISBN: 0-7695-2101-0.

[4] Hyacinth S.Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 1996, Nr. 11 (3).

[5] Gerd Wagner. The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior**.** *Information Systems,* 2003, Nr. 28. ISSN:0306-4379.

[6] James Odell: Objects and Agents Compared. *Journal of Object Technology*, 2002, Nr. 1. ISSN 1660-1769.

[7] *Java AWT: Delegation Event Model*. [žiūrėta 2007-03-18]. Prieiga per internetą < http://java.sun.com/j2se/1.3/docs/guide/awt/designspec/events.html >

[8] *Msdn*. [žiūrėta 2007-03-18]. Prieiga per internetą < http://msdn2.microsoft.com/en-us/library/17sde2xt.aspx >

[9] Vasilecas, O. ECA taisyklių realizacija aktyviose duomenų bazių valdymo sistemose. Informacinės technologijos, 2002, 71-77 p.

[10] Peter Pietzuch; Gero Muhl; ir Ludger Fiege. Distributed Event-Based Systems: An Emerging Community. *IEEE Distributed Systems Online*, 2007, Nr. 2, ISSN 0702-o2002.

[11] *Portal Implementation Issues and Challenges* [žiūrėta 2005-11-13]. Prieiga per internetą: < http://www.asis.org/Bulletin/Oct-04/morgan.html >

[12] Rogerio Luis de Carvalho Costa; Sergio Lifschitz ir Marcos Antonio Vaz Salles. Index Self-Tuning with Agent-based Databases. *CLEI Electronic Journal – Special Issue of Best Papers presented at CLEI'2002,* 2003, Nr. 1.

[13] Stan Franklin; Art Graesser. Is it an Agent, or just a Program: A Taxonomy for Autonomous Agents. *Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag*, 1996.

[14] Takanori Yokoyama. An Aspect-Oriented Development Method for Embedded Control Systems with Time-Triggered and Event-Triggered Processing. *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, 2005, ISSN:1080-1812

[15] Tarak Modi. *Event-Driven Architecture: Achieving Architectural Agility*. [žiūrėta 2007-02-25]. Prieiga per internetą: < http://www.soa-pipeline.com/trends/164302146 >

[16] *Web-Based Portal Computer-Human Interface Guidelines.* [žiūrėta 2005-11-06]. Prieiga per internetą < http://hf.tc.faa.gov/technotes/dot-faa-ct-tn04-23.pdf >

[17] *Wikipedia*. [žiūrėta 2007-02-25]. Prieiga per internetą < http://en.wikipedia.org/wiki/Embedded_system >

[18] *Wikipedia*. [žiūrėta 2007-02-25]. Prieiga per internetą < http://en.wikipedia.org/wiki/Web_2 >

Software agents development methodology based on DBMS

SUMMARY

During the last few years, amount of information, published in the internet, grew up exponentially. All that information is distributed during many different and not connected information systems, and it became difficult to find needed information easily. That's why we need powerful but easy controlled Web information systems that would allow us to find information in more easy way. However, such systems often require maintenance and information updates from system administrators. The purpose of this work is to increase dynamism, flexibility and automation degree of information systems and to ease maintenance of such systems by implementing configurable events control mechanisms, which would allow system administrator to set up and update automatically executed actions.

On purpose to make information systems configurable and automatic as much as possible, it is best to use agents. It is not a secret, that to use DBMS based agents is worth because of three reasons: rapidity, security and reliability. Procedures implemented in DBMS are executed very quickly because they do not need to be parsed, compiled or optimized every time – only during first time execution. They also do not increase loading time for web pages. Those features are very important for implementing smart internet information systems that pretend to satisfy Web2 standard requirements.