KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Tadas Žilinskas

Žiniatinklio portalų kūrimo ir projektavimo metodika

Magistro darbas

Darbo vadovas
doc. dr. E. Kazanavičius

Kaunas, 2004

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. E. Kazanavičius
2004 05

Žiniatinklio portalų kūrimo ir projektavimo metodika

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. J. Mikelionienė
2004 05

Vadovas
doc. dr. E. Kazanavičius
2004 05

Recenzentas
doc. dr. A. Riškus
2004 05

Atliko
IFM-8/1 gr. stud.
T. Žilinskas
2004 05

Kaunas, 2004

SUMMARY

The nature of Web system development is significantly different from conventional software development. Good method for system development is needed. Analysis show, that main known system development methods are good at describing static web application aspects, but does not provide a good description for modeling web application behavioral characteristics.

This work uses model driven approach as one, suitable for dealing with web application development problems. The key aspect of model driven approach is specifying system domain model independent of any particular technology (J2EE, Microsoft .NET, etc.) and then generating platform specific model or its implementation. The UML is user-friendly, easy to use and is useful for describing system effectively. Specification expressed in terms of UML can be rendered into an XML document using the OMG's XMI DTD for UML, which makes UML suitable for model driven approach implementation.

In this work, method for specifying application behavior design using UML was presented. This method gives rules for specifying web application behavior model, suitable for its implementation code generation. The main components used in today's web application which implementation could be generated from the model were also described. It was shown, that it is possible to generate much of implementation, just using such a model representation in XMI.

The main idea presented in this work is that web application development should be model driven as much as possible. And web application behavior should be modeled too. The components used in developing each web application maybe very similar and could be generated from information specified in model (for example components working with relational databases could be generated just from database structure specified in UML). Such an approach forces to design applications before beginning its implementation, ensures that implementation is consistent with design, and exploits reuse of general components.

ĮVADAS

Šio darbo pagrindinis objektas yra žiniatinklio portalai ir jų kūrimas. Žiniatinklio portalai skiriasi nuo paprastos žiniatinklio svetainės tuo, kad portalai yra skirti ne tik tam tikros informacijos pateikimui, bet ir organizacijos tam tikros verslo logikos vykdymui. Kitaip sakant, žiniatinklio portalas yra žiniatinklio svetainė, kur vartotojo veiksmai – naršymas portale ir duomenų įvedimas veikia verslą.

Žiniatinklio portalų kūrimas yra besivystanti disciplina. Jų kūrimas skiriasi nuo tradicinės programinės įrangos kūrimo. Į portalų kūrimo procesą yra įtraukiami įvairių įgūdžių žmonės, tokie kaip išdėstymo projektuotojai, programuotojai, įvairialypės terpės ekspertai, rinkodaros specialistai. Išaugo vartotojų vaidmuo ir darosi sunku suprasti srities struktūrą [3]. Ypatingai svarbūs portaluose tapo estetiniai ir kognityviniai aspektai, tai kas tradicinėje programinės įrangos inžinerijoje nebuvo akcentuojama. Kūrimo procesas taip pat turi būti geriau struktūrizuotas, labiau palaipsninis, iteratyvus ir palaikymo fazė vaidina svarbesnę rolę portalų gyvavimo cikle nei tradicinės programinės įrangos gyvavimo cikle. Todėl portalų inžinerija vaidina svarbią rolę žiniatinkliui skirtų sistemų kūrime.

Per paskutinius metus buvo aprašyta daug portalų kūrimo metodų. Jie turi panašumų ir skirtumų. Dauguma iš šių metodų koncentruojasi į žiniatinklio portalų projektavimą ir tik kai kurie iš jų dengia daugiau aspektų, tokių kaip kūrimo ciklas, reikalavimų rinkimas, įgyvendinimas ir testavimas. Šiame darbe bus koncentruojamasi į portalų architektūros projektavimą. Nes nuo to, kaip gerai suprojektuota sistemos architektūra priklauso jos palaikomumas, kai atsiranda naujos bei kinta senos funkcijos. Architektūros projektavimas yra sudėtingas procesas. Jisai apima struktūros, elgesio, vidinių elementų tarpusavio sąveikos aprašymą.

Kiekvienos projektavimo metodikos pagrindas yra sistemos aprašymas naudojant eilę įvairių sistemos modelių. Metodikos skiriasi pagal tai, kokie modeliai sudaromi, kokie žymėjimai naudojami ir pagal tai kokie palaikantys įrankiai egzistuoja. Dauguma metodikų pakankamai gerai aprašo kaip sudaryti statinės portalų dalies modelius, tačiau veikimo logikos, kuri tampa labai svarbi portaluose, aprašymas labai ribotas. Taipogi tik kai kurios metodikos turi įrankius, kurie leidžia panaudoti modelius, komponentų kodo generavimui. O tai yra labai svarbu, norint pasiekti, kad modeliai būtų suderinti su realizacija ir būtų pasiektas maksimalus kūrimo našumas. Kitas dalykas, ko neakcentuoja ir nepalaiko nei viena metodika, tai pačios metodikos praplėtimo, papildymo naujomis sąvokomis.

Problema. Žiniatinklio portalų kūrimas reikalauja išskirtinai gero portalo architektūros projektavimo metodo, kuris leistų specifikuoti portalo elgseną, pasiekti gerą portalų palaikomumą,

gebėtų prisitaikyti prie nuolat atsirandančių technologijų ir tuo pačiu leistų pasiekti maksimalų kūrimo našumą. Egzistuojančios žiniatinklio aplikacijų kūrimo metodikos to neužtikrina.

Pagrindinis uždavinys. Pasiūlyti žiniatinklio portalų architektūros projektavimo metodiką, kuri padėtų minimizuoti problemas su kuriomis susiduriama žiniatinklio portalų kūrime. Kadangi pagrindinių portalo struktūros elementų ir jų ryšių projektavimas yra pakankamai gerai išnagrinėtas, šiame darbe bus nagrinėjami žiniatinklio veiklos procesų projektavimo aspektai.

Tikslai:

- Išanalizuoti žiniatinklio portalų kūrimo ypatybes;
- Susipažinti su egzistuojančiom žiniatinklio portalų kūrimo metodikom;
- Pasiūlyti savo metodą portalų veikimo logikos projektavimui;
- Sukurti programą ar intarpą pasiūlytos metodikos veikimo patikrinimui;
- Patikrinti metodikos veikimą eksperimentu.

Darbo struktūra.

- **Analitinė dalis.** Analitinėje dalyje analizuojamos žiniatinklio portalų ypatybės, egzistuojančios žiniatinklio projektavimo metodikos, UML galimybės, bei jos tinkamumas žiniatinklio portalų projektavimui bei modeliu valdomos metodikos realizacijai.
- **Teorinė dalis.** Čia pateikiama žiniatinklio portalų kūrimo metodika, aprašomi modeliai sudaromi žiniatinklio portalo veikimo logikos projektavimo procese. Numatomi būdai, kaip realizuoti modeliu valdomą metodologiją – t.y. tai, kad sudaryti modeliai tiktų generuoti bent dalį specifikuotų komponentų.
- **Eksperimentinė dalis.** Eksperimentinėje dalyje aprašomas įvykdytas eksperimentas. Tai atliktas tyrimas, ar naudojantis pasiūlyta metodika, sudarius portalo veikimo modelius, realizacijos komponentai gali būti sugeneruojami ir kokia dalis jų sugeneruojama. Tam sukurtas, įrankis leidžiantis valdyti UML diagramose išreikštą informaciją, ir generuoti komponentus aprašytus tose diagramose.

1 ANALITINĖ DALIS

1.1 Žiniatinklio portalai

1.1.1 Žiniatinklio portalo apibrėžimas

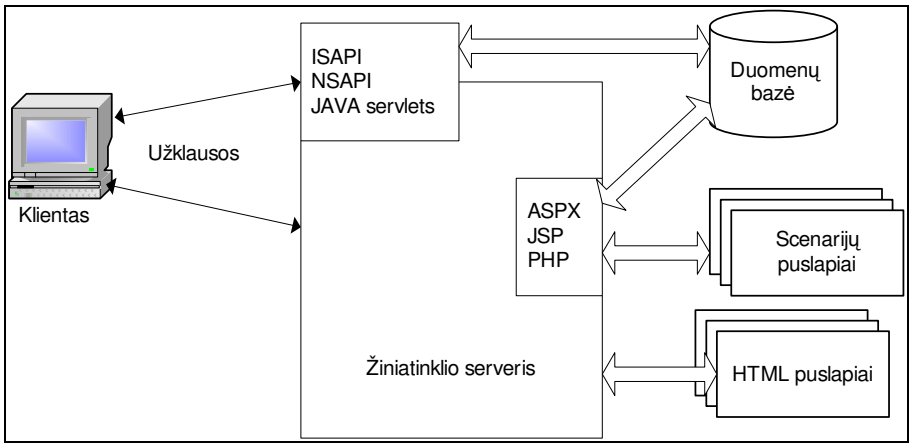
Žiniatinklio portalas – tai tradicinė žiniatinklio svetainė išplėsta galimybe atlikti verslo funkcijas. Jis leidžia vartotojams vykdyti verslo logiką naudojantis žiniatinklio naršykle. Kitaip sakant, žiniatinklio portalas yra svetainė, kur vartotojo veiksmai, įvedami duomenys veikia verslą. Tam, kad padaryti portalo turinį dinaminį ir tam, kad leisti sistemos vartotojams veikti verslo logiką, serveryje yra naudojamos įvairios technologijos.

1.1.2 Žiniatinklio portalams realizuoti naudojamos technologijos

Šiuo metu žiniatinklio portalų dinamiškumui realizuoti naudojamos dvi technologijos: sukompiliuoti moduliai ir interpretuojami scenarijai (angl. *scripts*) [15]. Sukompiliuotų modulių sprendimai yra panašūs į CGI modulius, kurie sukompiliuojami į užkraunamus ir serverio vykdomus modulius. Jie turi priėjimą prie informacijos apie užklausą, ir generuoja HTML kalba aprašytus puslapius, kurie yra siunčiami naršyklei. Vieni iš populiaresnių šios technologijos realizacijų yra Microsoft Internet Server API (ISAPI), Netscape Server API (NSAPI) ir Java servlets. Sukompiliuoti moduliai yra efektyvus sprendimas, tačiau jų pagrindinis trūkumas yra siejamas su kūrimu ir palaikymu. Šie moduliai paprastai apjungia verslo logiką su HTML puslapių teksto kūrimu. Kita problema – kiekvieną kartą, kai reikia modulį atnaujinti ar pataisyti, žiniatinklio portalas turi būti išjungiamas.

Kita technologijų kategorija yra scenarijų (angl. *scripted*) puslapiai. Kai sukompiliuoti moduliai atrodo kaip programa, kuri generuoja HTML, scenarijų puslapiai atrodo kaip HTML puslapiai, kurie vykdo tam tikrus veiksmus. Scenarijų puslapis yra failas žiniatinklio serverio failų sistemoje, kuris turi scenarijus, interpretuojamus serverio. Puslapis atrodo kaip paprastas HTML puslapis, tačiau turi tam tikrus žymėjimus, kuriuos interpretuoja serveris. Šį požiūrį realizuojančios technologijos: JavaServer Pages, Microsoft‘s Activer Server Pages ir PHP.

Ryšiai tarp žiniatinklio dinamiškumą realizuojančių komponentų ir tai padaryti leidžiančių technologijų ir serverio parodyti pav. 1. Duomenų bazė paveikslėlyje galėtų būti bet koks serverio resursas. Paveikslėlyje parodyta, kaip kompiliuoti moduliai perima užklausų apdorojimą ir patys veikia kaip maži serveriai.



Pav. 1 Žiniatinklio portalų dinamiškumą realizuojančios technologijos

Tuo tarpu scenarijų puslapius iškviečia serveris, jei nustato, kad puslapis tikrai turi interpretuojamus scenarijus. Kai serveris gauna užklausą vienam iš tokių puslapių, jis suranda tą puslapį ir perduodą jį serverio filtrui. Serveris interpretuoja puslapio scenarijus ir gaunamas HTML puslapis, kuris siunčiamas naršyklei.

Nors JavaServer Pages technologijos puslapiai turi scenarijus, tačiau pirmą kartą paleidus, jie yra sukompiliuojami. Tai duoda tam tikrą našumo pranašumą. Tačiau pagrindinis scenarijų puslapių privalumas yra ne jų greitis, tačiau jų kūrimo ir įdiegimo paprastumas. Paprastai, scenarijų puslapiai neatlieka visos veiklos logikos. Ji yra patalpinama komponentuose, kurie yra sukompiliuoti, laikomi atskirai ir puslapiuose yra tik iškviečiami.

1.1.3 Žiniatinklio portalų kūrimo proceso charakteristikos

Žiniatinklio portalų kūrimas smarkiai skiriasi nuo tradicinės programinės įrangos kūrimo. Šie skirtumai paprastai skirstomi į techninius ir organizacinius [2]. Techniniai siejami su naudojamomis technologijomis, organizaciniai su tuo, kokią naudą nori gauti organizacija iš portalo.

Pagrindiniai organizaciniai skirtumai:

- Klientų neapsisprendimas. Internetu pagrįstose sistemose, technologijos, verslo modeliai keičiasi taip greitai, kad probleminė sritis ne tik prastai suvokiama, bet ir pastoviai vystosi. Tai veda prie to, kad žiniatinklio portalų kūrimo procesas turi būti geriau struktūrizuotas, labiau palaipsninis, iteratyvus ir palaikymo fazė vaidina svarbesnę rolę portalo nei tradicinės programinės įrangos gyvavimo cikle.

- Verslo reikalavimų pasikeitimai. Ryšium su klientų neapsisprendimu, su pradėtom naudoti naujom technologijom, pasikeičia verslo procesai, todėl nestebina tai, kad reikalavimai keičiasi.

- Trumpesnis portalams kurti skiriamas laikas.

Techniniai skirtumai:

- Padidėjęs vartotojo sąsajos vaidmuo. Estetiniai ir kognityviniai aspektai portalų kūrime yra ypatingai svarbūs. To nėra tradicinės programinės įrangos kūrime. Taipogi tai, kad portalas prieinamas daugeliui žmonių, padidina kokybės aspekto svarbą. Nes trūkumai, kurie siejami su portalo naudojimu, našumu ir kt. aspektais labai gerai matomi ir todėl labiau problematiški.

- Išaugusi turinio svarba. Žiniatinklio portaluose turinys turi būti visada atnaujinimas, tai veda prie to, kad reikalingas efektyvus informacijos dizainas ir tinkamas turinio valdymas.

- Ryšys tarp verslo modelio ir architektūros. Techninė žiniatinklio portalo struktūra jungia sudėtingą verslo architektūrą (kurią paprastai lemia žymūs klientų verslo modelio pasikeitimai) su sudėtinga informacine architektūra. Ryšys tarp verslo architektūros ir techninio dizaino yra daug glaudesnis nei tradicinės programinės įrangos sistemose.

- Greitai besikeičiančios technologijos.

- Kuriant portalus paprastai dalyvauja labai įvairių profilių ir įgūdžių specialistai (turinio dizaineriai, menininkai, programuotojai ir kt.)

Visi šie skirtumai tarp tradicinės programinės įrangos ir portalų kūrimo lemia tai, kad portalų kūrimui valdyti reikalingos naujos arba modifikuotos tradicinės programinės įrangos kūrimo metodikos, kurios leistų atlikti tiek greitą realizaciją, tiek leistų kiek galima lengviau atlikti pakeitimus sistemoje.

1.2 Esamų metodologijų apžvalga

1.2.1 Hiperterpės projektavimo metodas (angl. Hypermedia Design Method)

Hiperterpės projektavimo metodas (HPM) yra vienas iš pirmųjų metodų, kuris buvo sukurtas aprašyti hiperterpės programų struktūrą ir sąveikas [5]. Jis yra paremtas E-R (angl. *Entity Relationship*) metodologija, bet praplečia esybės sąvoką ir įveda naujus primityvus, tokius kaip vienetai (angl. *unit*), kurie atitinka mazgus ir ryšius. HPM esybės turi vidinę struktūrą ir susietą naršymo semantiką, t.y. specifikaciją kaip galima naršyti ir kaip informacija vaizduojama. Jos sudaromos iš komponentų, komponentai – iš vienetų. Metodika apibrėžia tris ryšių tipus: struktūriniai, perspektyviniai (angl. *perspective*) ir programiniai. Struktūriniai ryšiai jungia komponentus, perspektyviniai – vienetus. Programiniai ryšiai jungia to pačio arba skirtingų tipų komponentus ir esybes.

Egzistuoja dvi skirtingos HPM esybių grupės: programos esybės ir dar taip vadinamos „kontūrais" (angl. *outlines*). Pastarosios leidžia prieiti prie taikomosios programos esybių ir siūlo naršymo pradžios taškus.

Garzotto, Mainetti ir Paollini išskiria sekančias, taikomųjų hiperterpės programų analizės dimensijas: turinys, struktūra, vaizdavimas, dinamika ir sąveika [5]. Turinys - tai informacijos dalys, o struktūra turinio organizavimas. Vaizdavimas apibrėžia kaip taikomosios programos turinys ir funkcijos yra parodomos vartotojui. HPM sąveika – tai funkcionalumas, kurį iškviečia vaizdavimo elementai. Kituose metoduose sąveika yra laikoma dinamikos ir vaizdavimo dalimi, kadangi ji yra abiejų faktorių kombinacija. HPM apibrėžia tokius ryšius: indekso ryšiai (angl. *index links*), nuoseklaus perėjimo ryšiai (angl. *guided tour*) ir ryšių rinkiniai. Indekso ryšiai jungia rinkinio mazgus su kiekvienu rinkinio nariu. Nuoseklaus perėjimo ryšiai jungia rinkinių mazgus į nuoseklią seką, kur kiekvienas narys sujungtas su sekančiu ir ankstyvesniu. Cikliniuose rinkiniuose paskutinis narys jungiamas prie pirmo. Ryšių rinkiniai yra indekso arba nuoseklaus perėjimo ryšiai, kurie leidžia pereiti rinkinių mazgus.

Vaizdo projektavimui, HPM apibrėžia dvi sąvokas: lizdo (angl. *slot*) ir rėmelio (angl. *frame*). Lizdas yra atominė informacijos dalis. Jis gali būti paprastas arba sudėtingas. Lizdai yra komponuojami rėmeliuose. Rėmelis yra vaizdavimo vienetas – tai kas rodoma vartotojui.

1.2.2 Ryšių valdymo metodologija (angl. Relationship Management Methology)

Ryšių valdymo metodologija (RVM) aprašo taikomųjų žiniatinklio programų projektavimą, kurį sudaro septyni žingsniai: esybių-ryšių projektavimas, dalių (angl. *slice*), naršymo struktūros, vartotojo sąsajos, protokolo konversijos (angl. *protocol conversion design*), vykdymo elgsenos projektavimas, konstravimas ir testavimas [14]. Žiniatinklio programos projektas yra vaizduojamas

ryšių valdymo duomenų modeliu (angl. *Relationship Management Data Model*), kuris yra paremtas esybių – ryšių modeliu.

Esybių ryšių projektavimo metu yra identifikuojamos esybės, atributai ir ryšiai, kurie tampa mazgais ir nuorodomis gautoje žiniatinklio taikomojoje programoje. Dalių projektavimas apima esybių atributų grupavimą atvaizdavimui. Tos dalys – tai „vaizdavimo vienetai", t.y. žiniatinklio puslapiai. Turinio ir vaizdavimo aspektų atskyrimas šiame žingsnyje nėra atliekamas. Naršymo struktūros projektavimas yra žingsnis, kuriame identifikuojami naršymo keliai. Protokolo konversijos projektavimo žingsnyje atliekamas RVM komponentų konvertavimas į fizinius realizacijos objektus. Vartotojo sąsajos projektavimo žingsnis apima kiekvieno diagramos elemento ekrano planų projektavimą.

Ši metodika esybių-ryšių projektavimo žingsnyje modelių sudarymui naudoja E-R notaciją, bei apibrėžia savas notacijas kitiems žingsniams.

1.2.3 Objektinis hiperterpės projektavimo metodas (angl. Object-Oriented Hypermedia Design Method)

Objektinis hiperterpės projektavimo metodas (OHPM) apima sekančias keturias veiklas:

- Konceptualus modeliavimas. Probleminės srities konceptualus modelis yra vaizduojamas klasių diagrama, taip kaip įprasta objektiniuose modeliuose. Naudojama UML notacija;
- Naršymo struktūros projektavimas. Įvedamos įvairios sąvokos aprašančios portalo naršymo ypatybes. Aprašymui naudojama speciali notacija;
- Abstraktus vartotojo sąsajos projektavimas. Abstraktus sąsajos modelis yra objektų, kuriuos supranta vartotojas, specifikacijos rezultatas. OHPM naudoja abstrakčius duomenų rodinius (angl. *Abstract Data Views*) statinių vartotojo sąsajos aspektų modeliavimui, o dinaminiai vartotojo sąsajos aspektai yra modeliuojami technika, paremta būsenų diagramomis (angl. *Statecharts*);
- Realizacijos projektavimas.

Šios veiklos yra vykdomos maišant palaipsninio, pasikartojančio ir prototipu-paremto projektavimo stilius [1]. Objektiniai modeliai yra sukuriami kiekviename žingsnyje tobulinant modelius sukurtus ankstesnėse iteracijose.

OOHDM-Web aplinka leidžia greitą hiperterpės programų prototipų kūrimą projektuojant su OHPM. Jis reikalauja vartotojo apibrėžtų naršymo konstruktų, tokių kaip naršymo klasės, naršymo kontekstai, vaizdavimo klasės ir priėjimo struktūros, konvertavimo į lenteles, kadangi šios

sąvokos nėra visiškai objektinės. OHPM naršymo ir sąsajos konstrukcijos yra susiejamos su funkcijų biblioteka cgi-Lua scenarijų kalba, tam naudojant Lua duomenų bazę. Remiantis lentelėmis ji generuoja puslapius dinamiškai. Projektuotojas, kuria puslapių šablonus naudodamas HTML kalbos konstrukcijas ir specialias komandas, kurias interpretuojama cgi-Lua scenarijų aplinka.

1.2.4 Išplėsta objekto ryšių metodologija (angl. Enchanced Object Relationship Methology)

Išplėsta objekto ryšių metodologija (IORM) yra apibrėžta kaip pasikartojantis procesas, praturtinantis objektinį modeliavimą tuo, kad ryšiai tarp objektų vaizduojami objektais [3]. Tai suteikia sekančius privalumus: ryšiai gali būti praplečiami, tampa semantiškai turtingesni, gali dalyvauti kituose ryšiuose ir gali būti pakartotinai panaudojami. Šis metodas siūlo kurti vartotojo sąsajos prototipą ankstyvoje projektavimo stadijoje.

Metodas yra pagrįstas trejais karkasais (angl. *framework*): klasės, kompozicijos ir vartotojo sąsajos. Klasės karkasas susideda iš pakartotino panaudojimo bibliotekos klasių aprašymų. Programos klasių identifikavimui IORM naudoja standartines objektines technikas. IORM išskiria du objektinių ryšių tipus: apibendrinimo ir vartotojo apibrėžtus. Pirmas ryšys turi iš anksto aprašytą susijusią semantiką, antras visiškai remiasi vartotojo specifikacija.

Kompozicijos karkasas susideda iš pakartotinio panaudojimo ryšių klasių aprašymo bibliotekos. Tai leidžia vartotojams pakartotinai panaudoti jau sukurtas ryšio klases ir, kai reikia, praplėsti jas naudojant paveldėjimą. Pagrindinių ryšių klasių semantika yra sekanti:

- Paprastas ryšys – bazinė ryšio klasė, kuri suteikia bazines ryšio savybes, įskaitant ir kūrimo, trynimo, kirtimo (angl. *traverse*) funkcijas;
- Naršymo ryšys – aprašo žiniatinklio naršymo nuorodų iškvietimus, įskaitant sukūrimo laiko ir istorijos išsaugojimą;
- Mazgas į mazgą – tai ryšys, kuris paveldėtas iš naršymo ryšio, suteikiant objekto į objektą ryšio funkcionalumą;
- Plėtimosi iki mazgo ryšys susieja objekto turinį su kitu objektu;
- Struktūros ryšys yra paveldėtas iš paprasto ryšio;
- Aibės ryšys – tai struktūrinis ryšys, kuris suteikia priėjimą prie objekto, esančio nerūšiuotame objektų rinkinyje.
- Sąrašo ryšys – tai kitas struktūrinis ryšys, kuris suteikia priėjimą prie objekto rūšiuotame objektų rinkinyje.

Paskutinis šio metodo žingsnis yra programos vartotojo sąsajos projektavimas naudojant vartotojo sąsajos bazinius elementus. Jame siūloma nustatyti vartotojo sąsajos langus ir vaizduojamus objektus, kurie turi atsirasti kiekviename lange. Nurodomi iš kokių klasių atributų sudaromi tie vaizdai, ir kokios klasių operacijos gali būti iškviečiamos.

Šiai projektavimo metodikai yra sukurtas ONTOS Studio įrankis, leidžiantis atlikti projektavimą naudojant IORM. Jis turi interaktyvią grafinę vartotojo sąsają, kuri generuoja C++ modelio realizaciją.

1.2.5 Scenarijais pagrįsta objektinė hiperterpės projektavimo metodologija (angl. Scenario-based Object-oriented Hypermedia Design Methodology)

Kitas sukurtas metodas yra scenarijais pagrįsta objektinė hiperterpės projektavimo metodologija (SPOHPM). Ji susideda iš šešių fazių: srities analizė, objektų modeliavimas, rodinių (angl. *views*) projektavimas, naršymo projektavimas, realizacijos projektavimas ir konstravimas. Ši metodologija turi panašumų su RVM, OHPM ir IORM. Ji skiriasi scenarijų naudojimu, kurie yra aprašomi scenarijų veiklos diagramose, ir yra pagrįsti įvykių, veiklų ir veiklos sekų primityvų, vartojimu. Scenarijai yra apibrėžiami probleminės srities analizės fazėje ir yra naudojami objektų modeliavimui. Rodinių projektavimas susideda iš objektinių rodinių aprašymo, sudaryto iš objektų klasių ar iš asociacijų ryšio tarp objekto klasių, nustatymo.

Naršymo projektavime naudojami scenarijai tam, kad nustatyti naršymo mazgus. Jie yra apibrėžiami panašiai kaip meniu mechanizmai, kurie leidžia vartotojams prieiti prie kitų hiperterpės dokumentų dalių. Priėjimo struktūros mazgai (PSM), kartu su objektiniais mazgais yra vadinami naršymo vienetais. Objektiniai rodiniai yra skirstomi į tris tipus: baziniai, asociacijos ir bendradarbiavimo. Šie rodiniai yra panašūs į kontekstus aprašytus OHPM. Bazinis rodinys yra generuojamas iš vieno objekto klasės. Asociacijos rodinys yra sudaromas iš asociacijos ryšio. Panašiai, bendradarbiavimo rodinys yra generuojamas iš bendradarbiavimo ryšio. Naršymo nuorodų identifikavimu užbaigiamas naršymo projektavimas.

Realizacijos projektavimo metu yra modeliuojama vartotojo sąsaja, puslapiai ir loginė duomenų bazės schema. Šis metodas pateikią aiškią žingsnių seką, kuri naudojasi scenarijais, gautais analizės fazės metu. Modeliavimui šioje fazėje naudojama sava notacija.

1.2.6 Žiniatinklio svetainių projektavimo metodas (angl. Web Site Design Method)

Žiniatinklio svetainių projektavimo metodas (ŽSPM) pasižymi į vartotoją orientuotu požiūriu, kuris apibrėžia informacijos objektus remdamasis žiniatinklio programų vartotojų reikalavimu informacijai [7]. ŽSPM susideda iš trejų pagrindinių fazių: vartotojų modeliavimo, konceptualaus ir realizacijos projektavimo.

Vartotojų modeliavimo fazėje potencialūs žiniatinklio svetainės vartotojai ar lankytojai yra identifikuojami ir klasifikuojami remiantis tuo, kuo jie domisi ir, kam jie teikia pirmenybę naršydami. Konceptualus projektavimas susideda iš dviejų žingsnių: objektų modeliavimo ir naršymo projektavimu. Objektų modelis sudaromas per tris žingsnius: verslo objektų modeliavimas, vartotojo objektų ir jų variacijų modeliavimas.

Naršymo modelis sudarytas iš tam tikro skaičiaus naršymo variantų, po vieną kiekvienam požiūriui, kuriuo remiantis vartotojai gali naršyti turimą informaciją. ŽSPM apibrėžia jį komponento ir nuorodos sąvokomis. Jis išskiria tris komponentų tipus: naršymo, informacijos ir išorinius.

Šio tipo naršymo projektavimas leidžia sukurti žiniatinklio programas, kurios turi labai sudėtingą hierarchinę struktūrą. Realizacijos projektavimo žingsnis užbaigia nuoseklaus ir efektyvaus konceptualaus modelio sukūrimą.

1.2.7 Naršymo ryšių analizė(angl. Relationship-Navigational Analysis)

Naršymo ryšių analizė (NRA) apibrėžia žingsnių seką, kuri bus naudojama žiniatinklio programų kūrimui, susikoncentruojant į analizę [8]. Ji yra ypatingai naudinga toms žiniatinklio programoms, kurios sukurtos liktinių sistemų pagrindu. Ji susideda iš penkių žingsnių:

- Programos vartotojų analizė, kurios tikslas yra identifikuoti programos vartotojus;
- Elementų analizė. Šiame žingsnyje yra išvardijami visi dominantys programos elementai. Šis sąrašas apima visokių tipų daiktus, dokumentus, formas, modelius ir t.t.;
- Ryšių ir metažinių analizė, kurios metu siekiama identifikuoti schemą, procesą, operaciją taip pat ir įvairių tipų ryšius;
- Naršymo analizė, kurios metu identifikuojama naršymo struktūra;
- Ryšių ir metažinių realizacijos analizė. Kūrėjas turi nuspręsti, kurie iš ryšių, identifikuotų trečiame žingsnyje bus realizuoti;

NRA duoda tik kelias nuorodas, kokius veiksmus atlikti kiekviename žingsnyje. Nėra siūlomos nei modeliavimo sąvokos, nei notacija.

1.2.8 Lankstus hiperterpės proceso modeliavimas (angl. Hypermedia Flexible Process Modeling).

Lankstus hiperterpės proceso modeliavimas (LHPM) yra Olsina pristatytas požiūris, kuris apima į analizę orientuoto aprašančiojo ir nurodančio proceso modeliavimo strategijas [19]. Jis aprašo egzistuojančius procesus, kartu suteikia nuorodas kaip planuoti ir valdyti hiperterpės projektą.

LHPM įtraukia funkcinius, metodologinius, informacinius, elgsenos ir organizacinius, hiperterpės kūrimo procesų vaizdus ar perspektyvas.

- Funkcinis vaizdas nurodo aibę fazių ir veiklų reikalingų tam, kad atlikti užduotis.
- Metodologinis vaizdas apibrėžia aibę specifinių procesų, kurie turėtų būti taikomi skirtingoms užduotims (pvz.: panaudojimo atvejais valdoma analizė, OHPM pagrįstas konceptualus ir naršymo modelis, t.t.). Specifinių procesų užduočių palaikymui yra pasirenkamas vienas ar daugiau metodų. Vieną specifinį metodą gali palaikyti vienas ar daugiau įrankių.
- Informacinis vaizdas parodo, kokie artefaktai, reikalingi užduotims, planuojami gauti.
- Elgsenos vaizdas reprezentuoja procesų modelio dinamiką. Daromi sprendimai susiję su užduočių, iteracijų pasibaigimo sąlygų, grįžtamojo ryšio ciklų ir t.t. sekomis ir sinchronizavimu.
- Organizacinis vaizdas apibrėžia aspektus tokius kaip rolės, komandų organizavimas, bendravimo mechanizmai, grupių dinamika ir t.t.

Sąrašas užduočių ir siūlomų užduočių dalių, nurodytų LHPM, hiperterpės programų kūrimui yra išvardintas žemiau. Jis apima sekančias technines, valdymo ir kognityvines užduotis:

- Programinės įrangos reikalavimų modeliavimas, toks kaip panaudojimo atvejų ar nefunkcinių reikalavimų modeliavimas, specialių terminų žodyno sudarymas;
- Projekto planavimas, t.y. projekto plano analizė ir specifikavimas;
- Konceptualus modeliavimas, kuris susideda iš probleminės srities analizės ir specifikavimo;
- Naršymo modeliavimas, kuris apima numatomų vartotojo užduočių analizę, naršymo klasių identifikavimą, naršymo schemos ir naršymo transformacijų specifikavimą;
- Abstraktus sąsajos modeliavimas, kuris remiasi vartotojo sąsajos modelio analize, apčiuopiamų sąsajos objektų specifikavimu, įvykiais ir transformacijomis;
- Projektavimo šablonų taikymas, t.y. naršymo, architektūros ir vartotojo sąsajos šablonų vartojimas;
- Įvairialypės terpės duomenų surinkimas ir redagavimas;

- Fizinis modeliavimas, kurį sudaro komponentų taikymas, greitas funkcinis prototipų sudarymas, evoliucinis prototipų sudarymas, eskizų ir maketų sudarymas, komponentų integravimas;
- Patikra ir atestavimas (angl. *validation, verification*);
- Kognityvinių kriterijų taikymas;
- Kokybės užtikrinimas, kokybės analizė ir tobulinimo strategijos, kokybės plano specifikavimas;
- Projekto koordinavimas ir valdymas, t.y. proceso, artefaktų ir resursų kontroliavimas ir valdymas;
- Dokumentavimas.

Tai yra platus inžinerija pagrįstas požiūris. Jis padengia visas esmines hiperterpės projekto fazes ir veiklas.

1.2.9 OO/Šablonų požiūris (angl. OO/Pattern Approach)

OO/Šablonų požiūris į hiperterpės aplikacijų projektavimą yra panašus į LHPM, kadangi jis siūlo išnaudoti objektinį projektavimą ir šablonų taikymą naršymo ir vaizdavimo projektavimui [9]. Jis skiriasi nuo LHPM, kadangi nepadengia viso programinės įrangos kūrimo ciklo, t.y. projekto valdymo, testavimo ir palaikymo aspektų. Šablonų vartojimas turi žinomus privalumus, tokius kaip tai, kad procesas yra gerai apibrėžtas, dokumentavimas gali būti pakartotinai panaudotinas ir palaikymas yra lengvas.

Šis metodas nurodo sekančius žingsnius: panaudojimo atvejų surinkimas, konceptualus projektavimas, bendradarbiavimo projektavimas, klasių apibrėžimas, naršymo projektavimas ir realizavimas. Nauji šio metodo aspektai lyginant su kitomis metodologijomis yra:

- Panaudojimo atvejų analizė skirtingiems vartotojų tipams;
- Sąveikų projektavimas pagrįstas apibrėžtais panaudojimo atvejais ir konceptualiu projektavimu;
- Naršymo projektavimas pagrįstas šablonais.

Jie apibrėžia sluoksniuotą hierarchijos šabloną, kurio tikslas yra sukurti intuityvią navigaciją hierarchiškai struktūrizuotai informacijai.

1.2.10 Inžinierinis Lowe-Hall požiūris (angl. Lowe-Hall‘s Engineering Approach)

Lowe ir Hall pateikė hiperterpės programų kūrimo proceso karkasą [10]. Jis apima srities analizę, produkto, procesų bei projekto modeliavimą, kūrimą ir dokumentavimą. Produkto

modeliavimas susideda iš modelio skirto galutiniam produktui pasirinkimo. Karkasas palaiko tris skirtingus požiūrius į produktą: programavimo kalba pagrįstas, modeliu pagrįstas ir koncentruotas į informaciją požiūriai.

- Programavimo kalba pagrįstame modelyje informacija ir informacijos struktūra yra įdėta į programavimo struktūrą;
- Ekranu pagrįstame modelyje puslapiai yra rankiniu būdu susiejami, kad gauti hiperterpės aplikaciją;
- Į informaciją koncentruotame modelyje informacija yra patalpinama duomenų bazėje.

Proceso modeliavimas susideda iš fazių, veiklų ir artefaktų kūrimui pasirinkimo, bei nustatymo, kaip jie yra integruojami į specifinį programos kūrimo procesą. Palaipsninis kūrimas ir prototipų konstravimas yra rekomenduojamas hiperterpės programų kūrimui.

Kūrimo proceso meto atliekamos veiklos yra: sistemos analizė, projektavimas, gamyba, patikra ir testavimas taip pat ir palaikymas. Kai kurioms veikloms susijusioms su hiperterpės programos analize ir projektavimu yra pristatoma aibė kūrimo technikų. Pavyzdžiui, siūloma naudoti RVM metodologiją struktūros projektavimui.

1.2.11 Notacijos naudojamos žiniatinklio portalų projektavime

Kai kurios žiniatinklio portalų kūrimo metodikos skiria dėmesį tik projektavimui ir notacijai. Tokios metodologijos yra: HPM, RVM, OHPM, SPOHPM ir ŽSPM. Kiti darbai skiria dėmesį tik notacijai: UML praplėtimas pasiūlytas J.Connallen [15]; darbas pristatytas Baumeister, Koch ir Mandel [20], kuris paremtas sąvokomis ir modeliais apibrėžtais jau egzistuojančių metodų, bei pristato UML išplėtimą – pilnai suderintą su UML standartu.

Struktūrizuota Hiperterpės Projektavimo Technika (angl. *Structured Hypermedia Design Techinque*) sutelkia dėmesį į savo notaciją, vaizduojančią projektavimo primityvus, kurie yra skirti statiniam žiniatinklio portalų modelio sudarymui. Šie projektavimo primityvai yra: svetainė, diagrama, puslapis, išdėstymas, forma, indeksas, meniu, nuoroda ir dinaminė nuoroda.

J. Conallen apibrėžia aibę UML stereotipų, skirtų žiniatinklio modeliavimui, bet nepristato projektavimo metodo [15]. Ji apima komponentų, klasių, metodų ir asociacijų stereotipus, tokius kaip serverio komponentas, kliento komponentas, serverio puslapis, kliento puslapis, forma, rėmelių aibė (angl. *frameset*), tikslas, serverio metodas, kliento metodas, nuorodos, nukreipimas, konstrukcija (angl. *build*), tikslinės nuorodos ir siutimas (angl. *submit*). Šie stereotipai yra tinkami

modeliuoti išdėstymo ir realizavimo aspektus, bet neapibrėžia žiniatinklio aplikacijos naršymo struktūros.

Baumeister, Koch ir Mandel darbas yra modeliu-pagrįstas požiūris, kurio modeliavimo technikos yra UML diagramos ir kurių grafinė reprezentacija tik naudoja UML notaciją [20]. UML yra išplėstas modeliuoti naršymo struktūrą ir vaizdavimą pagal UML išplėtimo mechanizmus. Šie mechanizmai yra paremti stereotipų apibrėžimu.

Kiti hiperterpės projektavimo metodai naudoja standartinę notaciją, tiktai konceptualiam projektavimui. Šios notacijos yra objektinės, kaip OMT ar UML, arba E-R paremtos. Jie apibrėžia jų notaciją ir grafines technikas kitiems žingsniams. Pavyzdžiui RVM atlieka E-R projektavimą pirmame žingsnyje, todėl naudoja E-R notaciją. Sekantiems žingsniams – naršymo ir vartotojo sąsajos projektavimui ji apibrėžia savo notaciją. Tuo tarpu ŽSPM nesiūlo jokios notacijos konceptualiam modeliui, tačiau naršymo objektų modeliui vaizduoti yra siūlomos E-R taip pat kaip OMT, kurios naudoja savo apibrėžtą grafinę notaciją naršymo objektų modeliui vaizdavimui.

Ankstyvose publikacijose OHPM metode siūloma naudoti OMT, tačiau vėliau naudojamas UML, bet tik konceptualiam modeliui [1]. OHPM yra nesuderintas su UML, kadangi jis naudoja savo notaciją taip vadinamiems perspektyvos atributams klasių diagramose ir siūlo kitas diagramų rūšis (naršymo konteksto schemos, sąrankos diagramos ir kt.) tam kad, aprašyti naršymo ir abstraktų sąsajos projektavimą.

SPOHPM yra scenarijais pagrįsta metodologija naudojanti savo notaciją scenarijų veiklos diagramoms, klasių struktūros diagramoms ir objektiniams vaizdams. Klasių struktūros diagrama yra grafinė, informacijos, patalpintos klasės atsakomybės bendradarbiavimo kortelėse (angl. *Class-Responsibility-Collaboration Cards*), reprezentacija. ŽSPM neriboja notacijos pasirinkimo: E-R ir OMT yra abu siūlomi naudoti verslo objektų, vartotojo objektų ir perspektyvos objektų modeliavimo žingsniams. Naršymo modeliui yra pasirenkama sava notacija.

1.2.12 Žiniatinklio portalų kūrimo metodų palyginimas

Programinės įrangos sistemų kūrimo metodų palyginimas yra sunki užduotis. Metodikų tikslai gali būti skirtingi, vieni bando paliesti daugelį kūrimo proceso aspektų, kiti bando giliai detalizuoti vieną ar du iš jų.

Čia bus palyginti žingsniai, kurie atliekami naudojant metodą, naudojamą modeliavimo techniką taip pat grafinę reprezentaciją ir notaciją pasirinktą šiam kūrimui. Kitas kriterijus naudotas palyginimui yra CASE įrankio palaikymas siūlomas kūrimui.

Procesų žingsniai ir fazės yra numeruoti. Šie numeriai yra naudojami trečiame ir ketvirtame stulpeliuose parodyti kokio tipo grafinė reprezentacija ar notacija yra siūloma kiekviename žingsnyje.

Galima pastebėti, kad nors žingsnių skaičius, technikos, notacijos taip pat kaip grafinė reprezentacija yra skirtingos, seka, kuria šie žingsniai yra atliekami yra panašūs. Pirma, programos srities modelis yra analizuojamas ir/arba projektuojamas, po to metodai susitelkia į struktūrą ir naršymą ir paskui jie tęsiasi vartotojo sąsajos projektavimu.

Lentelė Nr. 1 Žiniatinklio sistemų kūrimo metodai

	Procesai	**Modeliavimo technika**	**Grafinė reprezentacija**	**Notacija**	**Įrankio palaikymas**
HPM	1. Kūrimas plačiai 2. Kūrimas siaurai	E-R	1-2 E-R diagrama	E-R	
RVM	1. E-R projektavimas 2. Dalių projektavimas 3. Naršymo projektavimas 4. Konvertavimo protokolo projektavimas 5. Vartotojo sąsajos ekrano projektavimas 6. Vykdymo elgsenos projektavimas 7. Konstravimas ir testavimas	E-R	1. E-R diagrama 2. Dalių diagrama 3. RVM diagrama	1. E-R 2-3 nuosava	RMCase
IORM	1. Klasių karkasas 2. Kompozicijos karkasas 3. Grafinės vartotojo sąsajos (GVS) karkasas.	OO	1. Klasių diagrama 2. GVS dizainas	1. OMT	ONTOS Studio

OHPM	1. Konceptualus projektavimas 2. Naršymo projektavimas 3. Abstraktus vartotojo sąsajos projektavimas 4. Realizacijos projektavimas	OO	1. Klasių diagrama 2. Naršymo klasių + Konteksto schema 3. Abstraktaus Duomenų Rodinių (ADR), sąrankos diagrama + ADV diagrama	1. OMT/UML 2. Nuosava 3. ADV	OOHDM-Web
SPOHPM	1. Srities analizė 2. Objektų modeliavimas 3. Rodinių projektavimas 4. Naršymo projektavimas 5. Realizacijos projektavimas 6. Konstravimas	Scenarijai OO vaizdai	1. Scenarijų veiklos diagramos 2. Klasių struktūros diagramos 3. OO vaizdas 4. Naršymo nuorodų schema 5. Puslapio schema	1-5 nuosava	
ŽSPM	1. Vartotojų modeliavimas 2. Konceptualus projektavimas 2.1 Objektų modeliavimas 2.2 Naršymo projektavimas 3. Realizacijos projektavimas 4. Realizavimas	E-R/OO	1.E-R arba klasių diagrama 2. Naršymo lygmenys	1. E-R/OMT 2. Nuosava	
NRA	1. Programos vartotojų analizė 2. Elementų analizė 3. Ryšių ir metažinių analizė 4. Naršymo analizė 5. Ryšių ir metažinių realizacijos analizė				DHymE
LHPM	1. Reikalavimų modeliavimas 2. Projekto planavimas 3. Konceptualus	OO	3-5 OHPM	=OHPM	

	modeliavimas 4. Naršymo modeliavimas 5. Abstraktus sąsajos modeliavimas 6. Projektavimo šablonų panaudojimas 7. Įvairialypės terpės duomenų rinkimas/redagavimas 8. Fizinis modeliavimas/integravimas 9. Patikra/tikrinimas 10. Kognityvinių kriterijų išnaudojimas 11. Kokybės užtikrinimas 12. Projekto koordinavimas ir valdymas 13. Dokumentavimas				
OO/ Šablonų Požiūris	1. Panaudojimo atvejų analizė 2. Konceptualus projektavimas 3. Sąveikų projektavimas. 4. Klasių apibrėžimas 5. Šablonais pagrįstas naršymo projektavimas. 6. Realizavimas	OO	2. Klasių diagrama 3. Bendradarbiavimo diagrama		
Lowe-Hall požiūris	1. Srities analizė 2. Produkto modeliavimas 3. Procesų modeliavimas 4. Projekto planavimas 5. Kūrimas 5.1. Analizė 5.2. Projektavimas 5.3 Gamyba 5.4. Patikra ir testavimas 5.5. Pristatymas ir palaikymas 6. Dokumentavimas	RVM (siūloma)			

1.3 UML modeliavimo kalba

UML yra kalba, skirta sudėtingų sistemų struktūrai ir ryšiams vaizduoti. Ši kalba sukurta pagal OMG (angl. *Object Management Group*) užsakymą ir skirta objektinės metodologijos standartizavimui [12]. UML siūlo standartizuotą būdą aprašyti ne tik konceptualioms sistemos ypatybėms, tokioms kaip verslo procesai bei sistemų funkcijos, bet ir konkrečioms techninėms sistemų realizacijų ypatybėms.

UML yra sudėtinga ir palaiko daugiau skirtingų specifikavimo dalių, nei kitos modeliavimo kalbos, todėl gerai tinka itin sudėtingų sistemų modeliams kurti.

UML apima tokius sistemos specifikavimo aspektus:

- Objektinio modelio specifikavimas;
- Panaudojimo atvejų ir scenarijų specifikavimas;
- Elgsenos modelio ir būsenų diagramų specifikavimas;
- Įvairių rūšių elementų grupavimas;
- Užduočių kūrimo ir sinchronizavimo vaizdavimas;
- Fizinės topologijos modelių specifikavimas;
- Programos kodo sisteminimas.

Pagrindinės UML diagramos. Panaudojimo atvejų diagrama parodo ryšius tarp sistemos dalyvių ir vartojimo atvejų. Ši diagrama paprastai naudojama sistemos reikalavimams aprašyti. Panaudojimo atvejai – tai daugiau funkcinis, o ne objektinis sistemos aprašas.

Klasių diagrama - tai statinė modelio struktūra, dažniausiai vaizduojanti egzistuojančius elementus (pvz., klases ir tipus), jų vidinę struktūrą, ryšius tarp elementų. Klasių diagramoje nėra vaizduojama laikinė informacija, tačiau joje gali būti vaizduojami įvykiai kaip objektai, ar elementai, turintys laikinę informaciją.

Sekų diagrama rodo įvykių kitimą laike. Paprastai joje vaizduojama kaip objektai bendrauja tarpusavyje bėgant laikui.

Bendradarbiavimo diagrama vaizduoja skirtingų sistemos veikėjų vaidmenų santykius. Skirtingai nuo sekų diagramos, bendradarbiavimo diagrama rodo ryšius tarp objektų, turinčių skirtingus vaidmenis. Pagrindinis šių diagramų skirtumas, kad bendradarbiavimo diagramoje laikas nėra vaizduojamas kaip atskiras matmuo.

Veiksmų diagrama - tai būsenų mašinos atmaina. Šioje diagramoje būsenomis atvaizduojamas veiksmų vykdymas. Kai užbaigiamas vienas veiksmas, įgalinamas kito veiksmo vykdymas. Būsenų diagrama gali būti naudojama modelio elementų (pvz. objektų ar sąveikų) elgsenai aprašyti. Paprastai ši diagrama aprašo galimas būsenų sekas ir veiksmus, kuriais elementai pereina iš vienos būsenos į kitą per visą savo gyvavimo laiką. Šie veiksmai iškviečiami tam tikrų įvykių (pvz., signalų, operacijų kreipinių).

Išsidėstymo diagrama rodo sistemos struktūrą. Tai gali būti programinio kodo struktūra arba sistemoje naudojamų komponentų struktūra.

Išplėtimo mechanizmai. UML modeliavimo kalba taip pat turi sukurtus išplėtimo mechanizmus, kurie leidžia praplėsti UML naujomis sąvokomis, susietomis su konkrečia problemine sritimi. Tam yra naudojamos tokie UML kalbos elementai:

- Stereotipai yra naudojami norint įvesti naujus elementus;
- Apribojimai – tai taisyklės nusakančios sąlygas, kurios turi galioti modelio elementams;
- Žymėtosios reikšmės (angl. *tagged values*) yra modelių elementų savybės sudarytos iš rakto ir jo reikšmės.

Apsikeitimas diagramomis. Kita UML modeliavimo kalbos ypatybė yra tai, kad ji turi standartizuotą apsikeitimo diagramomis būdą XML formatu. Yra sukurtą daug UML įrankių sudaryti diagramoms ir daugumą jų palaiko šį UML diagramų apsikeitimo standartą.

Įrankiai. UML modeliavimo kalbos populiarumas lėmė kad buvo sukurtą daug įrankių palaikančių UML modelių sudarymą.

1.4 Modeliu valdoma architektūra (MVA)

1.4.1 MVA apibrėžimas

Programinės įrangos kūrimas kaip ir bet kokio kito darbo atlikimas reikalauja atlikti kuriamo produkto projektavimą. Projektavimas veda prie to, kad sistemos yra lengviau kuriamos, integruojamos ir lengviau palaikomos. Programinės įrangos projektavimas atliekamas sudarant eilę kuriamą sistemą aprašančių modelių.

Sistemos modelis yra sistemos bei jos aplinkos specifikacija arba aprašymas, atliekamas su tam tikru tikslu. Modelis yra išreiškiamas piešinių ir teksto kombinacija.

Modeliu valdoma architektūra (MVA) yra kitas OMG grupės standartizuotas sistemų kūrimo požiūris, kuriame didžiausią reikšmę vaidina modeliai. Modeliu valdomas tai reiškia, kad modeliai naudojami valdyti projektavimą, kūrimą, palaikymą [11].

MVA paradigmos esmė yra kūrimo procesas susidedantis iš sekančių žingsnių:

1. Patikimas reikalavimų kuriamai sistemas išsaugojimas;
2. UML diagramų, nepriklausančių nuo konkrečios realizacijai naudojamos technologijos, kūrimas taikymo sričiai. UML modelis aprašo pagrindines verslo funkcijas ir komponentus. Šis UML modelis vadinamas nepriklausomu nuo platformos modeliu;
3. UML diagramų, specifinių realizacijai naudojamų technologijų, kūrimas. Šis UML modelis turi technologijai specifinius elementus ir vadinamas specifiniu platformai modeliu;
4. Sistemos realizacijos išeities tekstų generavimas naudojant MVA įrankius. T.y. vietoj to, kad realizuoti visą sistemą remiantis UML modeliu, generuojama didžioji realizacijos dalis iš UML diagramų. Pavyzdžiui naudojant J2EE platformą, MVA įrankis sugeneruotų didžiąją dalį JSP ir EJB. Bereikėtų tik užpildyti sugeneruotų elementų detales, kurios negali būti aprašytos UML modeliu, tokius kaip veikimo logika.

1.4.2 MVA teikiami pranašumai

Organizacijos, kurios bendradarbiavo kuriant MVA tiki, kad jos taikymas suteiks tokius pranašumus:

1. Greitesnis kūrimo laikas. Kodo generavimas leidžia pasiekti tai, kad nereikia rašyti panašių komponentų daug kartų. Taip taupomas laikas, sugaištas kūrimui.
2. Architektūriniai pranašumai. Naudojant MVA, kuriamos sistemos modeliuojamos naudojant UML. Modeliuojama ne tik tarkim Java klasės, bet ir aukšto lygio dalykinės srities esybės. Tai verčia mąstyti apie architektūrą ir sistemos objektų modelį, vietoj to, kad iš karto pradėti rašyti realizacijos tekstą. Programinės inžinerijos principai įrodo, kad projektuojant sistemą sumažinama architektūrinių sistemos trūkumų tikimybė.
3. Pagerinama išeities tekstų priežiūra. Daugelis organizacijų turi problemų palaikant taikomųjų programų architektūros ir išeities tekstų nuoseklumą. Kai kurie kūrėjai naudoja gerai žinomus programavimo šablonus, kiti to nedaro. Naudojant MVA įrankius, tam, kad generuoti kodą, vietoj jų rašymo, kūrėjams suteikiama galimybė naudoti tuos pačius

projektavimo šablonus, kadangi išeities tekstai generuojami taip pat kiekvieną kartą. Tai yra svarbus pranašumas žiūrint iš priežiūros perspektyvos.

4. Padidintas pernešamumas tarp įvairių programavimo platformų. Jeigu reikia pakeisti realizavimui naudojamą platformą (pvz.: iš J2EE į .NET), nuo platformos nepriklausantis UML modelis gali būti pakartotinai panaudojamas.

1.5 Išvados

Yra sukurta daug žiniatinklio portalų projektavimo ir kūrimo metodikų, tačiau jos orientuotos į žiniatinklio struktūros modelių sudarymą ir neaprašo veikimo logikos, kuri yra labai svarbi portaluose, neužtikrina greito portalų kūrimo, neatsižvelgia į naujų atsirandančių technologijų portalų dinamiškumui realizuoti išnaudojimo galimybes ir neaptaria projektavimo aspektų gerai žiniatinklio sistemų priežiūrai, šių sistemų gyvavimo metu. Dauguma jų naudoja ir įveda įvairias savo sąvokas ir naujus žymėjimus, kas apsunkina tų metodikų įsisavinimą ir lemia tai, kad metodiką palaikančių įrankių arba visai nėra sukurta, arba tie, kurie sukurti labai riboti ir pritaikyti tik labai konkretiems atvejams. Tuo tarpu modeliu valdomos architektūros požiūris turėtų užtikrinti ir greitesnį portalų kūrimą ir lengvesnę portalo priežiūrą jo gyvavimo metu. UML modeliavimo kalba su savo išplėtimo mechanizmais, ją palaikančiais įrankiais bei standartizuotu duomenų apsikeitimu labai tinka realizuoti MVA požiūriui pritaikytam žiniatinklio portalų kūrimui. Išplėtimo mechanizmų pagalba galima apibrėžti žinomas portalo realizacijos sąvokas, kuriomis būtų operuojama sudarant portalų realizacijos modelius, aprašyti jų naudojimo ypatybes, taip, kad būtų galima generuoti jų realizacijos išeities tekstus. Apibrėžiamų sąvokų realizacijai būtų panaudojami patikrinti komponentai, todėl žymiai sumažėtų klaidų tikimybė. Diagramų sudarymui galima naudotis įvairiais UML diagramų sudarymo įrankiais, o standartizuotas apsikeitimas diagramomis leidžia realizuoti diagramose aprašytos specifikacijos realizacijos kodo generavimą.

2 TEORINĖ DALIS

Žiniatinklio portalų projektavimui siūlau naudoti metodiką paremtą MVA principais ir architektūros specifikavimui naudoti UML modeliavimo kalbą. Jos esmė yra portalo realizacijos ir svarbiausia portalo veiklos modelio sudarymas. Tam, kad parodyti, jog UML modeliavimo kalba tinka aprašyti portalo elgsenai, pirmiausiai pateiksiu formalų portalo elgsenos modelio aprašą, vėliau apibrėšiu, kokios tam galėtų būti naudojamos UML diagramos ir žymėjimai.

Sudarytas elgsenos modelis turėtų užtikrinti kiek įmanoma geresnį palaikomumą, portalo aiškumą ir turėtų numatyti komponentus, kurių realizacijos išeities tekstus būtų galima generuoti ir neprogramuoti kiekvienam portalui iš naujo.

2.1 Reikalavimai portalo architektūrai

Žiniatinklio portalai pasižymi kliento – serverio architektūra. Naršyklė prašo serverio tam tikro resurso, serveris identifikuoja portalą, kuriam ši užklausa skirta, ir inicijuoja jos apdorojimą. Kai vartotojas naršyklės pagalba naršo po portalą ar vykdo įvairius veiksmus, jis tai daro naudodamasis puslapyje esančiais komponentais. Taigi kiekviena užklausa gali būti iškviečiama iš įvairių vartotojo sąsajos komponentų. Užklausos apdorojimas - tai tam tikras portalo logikos vykdymas bei atsakymo naršyklei suformavimas. Portalo veiklą sudaro įvairūs objektai, atliekantys veiksmus su įvairiais duomenimis. Atsakymas naršyklei – tai puslapis, kurį generuoja tam tikri komponentai, kuriuos vadinsiu vartotojo sąsajos komponentais. Portalo gyvavimo metu dažniausiai keičiasi puslapių išvaizda, puslapiuose vaizduojamų duomenų struktūra bei duomenys, kurie surenkami iš vartotojų. Norint užtikrinti lengvą portalo palaikomumą, reikia, užtikrinti:

- kad, puslapius generuojantys komponentai patys duomenų, reikalingų jų generavimui, neformuotų, tačiau turėtų funkcijas, kurias naudojant tie duomenys būtų jiems perduodami;
- kad, vartotojo sąsajos komponentai su duomenimis, gaunamais iš vartotojų, neatliktų jokių veiksmų, tokių kaip jų išsaugojimas duomenų bazėje, o vietoj to turėtų turėti funkcijas, grąžinančias tuos duomenis ir jos būtų aprašytos architektūros modelyje;
- kad, modelyje būtų parodyti objektai ir veiksmai, kurie vykdomi su duomenimis, gaunamais iš vartotojo;
- kad, modelyje būtų pavaizduota, kokiai užklausai, kokie vartotojo sąsajos komponentai, užkraunami;
- kad, modelyje būtų pavaizduota, kokie objektai ir kokie veiksmai kviečiami, kad gauti duomenis, reikalingus užkrauti vartotojo sąsajos komponentui;

- kad metodai dirbantys su duomenų baze neatliktų kitų veiksmų išskyrus duomenų paėmimą ir išsaugojimą, nes portaluose naudojami duomenys yra dažnai imami iš duomenų bazės, o duomenų bazės struktūra turi tendenciją keistis.

Užtikrinus tai, kiekvienos portalo užklausos apdorojimą sudarytų:

1. Seka veiksmų, kurie atliekami su vartotojo sąsajos komponentų grąžinamais duomenimis, tame tarpe turi būti būtinai parodomi veiksmai su duomenų baze;
2. Vaizdavimo komponentų, kurie gali būti užkrauti kaip atsakas į užklausą, išrinkimas ir jiems reikalingus duomenis grąžinančių objektų ir jų veiksmų sekų atlikimas, tame tarpe būtinai turi būti parodyti veiksmai su duomenų baze, jei duomenys paimami iš jos;

Toliau pateiksiu formalų tokios portalo elgsenos modelio aprašymą naudojant z-notaciją.

2.2 Formalus portalo veiklos modelio aprašas

Visi duomenys, kuriais manipuliuojama portale yra tam tikro tipo ir turi tam tikrą pavadinimą. Todėl įvedamos tokios aibės:

[*DuomenųTipas*, *Pavadinimas*]

Kintamieji identifikuojami pavadinimu ir tipu.

KintamojoAprašas
pavadinimas: Pavadinimas
tipas: DuomenųTipas

Veiksmo, kurį gali atlikti tam tikras objektas, aprašymą sudaro pavadinimas bei įeinančių ir gražinamų duomenų aprašymas.

VeiksmoAprašas
pavadinimas: Pavadinimas
įeinantys_duomenys: iseq *DuomenųTipas*
grąžinami_duomenys: DuomenųTipas

Objekto aprašymą sudaro pavadinimas ir aibė veiksmų, kuriuos objektas gali atlikti. Kiekvienas objektas turi savo būseną, tačiau paprastumo dėlei to neparodau. Į objektų aibę įeina ir puslapius generuojantys komponentai, su veiksmais užpildančiais jų vaizduojamus duomenis ir veiksmais grąžinančiais iš vartotojo surinktus duomenis.

Objektas
pavadinimas: Pavadinimas
veiksmai: VeiksmoAprašas

Objekto veiksmo kvietimo aprašymą sudaro objektas, ir jo vykdomas veiksmas. Argumentai, kurie paduodami veiksmui, aprašomi seka kintamųjų, kurių tipai ir tvarka sutampa su veiksmo aprašymą sudarančių argumentų tipais.

ObjektoVeiksmoVykdymas
objektas: Objektas
veiksmas: VeiksmoAprašas
argumentai: iseq *KintamojoAprašas*
rezultatas: KintamojoAprašas

veiksmas objektas . veiksmai
i: 1 .. *# veiksmas . įeinantys_duomenys*
veiksmas . įeinantys_duomenys i = argumentai i . tipas
rezultatas . tipas = veiksmas . grąžinami_duomenys

Schema „*VeiksmųSeka*“ –tai tam tikru pavadinimu pavadinta objektų veiksmų seką, kur veiksmų iškvietimams paduodami objektai, gauti vykdant kitus veiksmus.

VeiksmųSeka
seka: iseq *ObjektoVeiksmoVykdymas*
pavadinimas: Pavadinimas

i: 1 .. *# seka* ran *seka i . argumentai* *x:* ran *seka x . rezultatas*

Užklausų elgsenai specifikuoti papildomai įvedamos tokios aibės:

- Aibė sąlygų, kurios gali padėti nuspręsti, kokias veiksmų sekas vykdyti;
- Aibė elementų identifikuojančių veiklos pradžią ir pabaigą;
- Užklausų aibė;
- Sąlygos mazgų aibė. Sąlygos mazgas įvedamas identifikuoti sąlygas, kuriomis remiantis būtų pasirenkama vykdymui tam tikra veiksmų seka;

[*SąlygosIšraiška*, *VeiklosPradžia*, *VeiklosPabaiga*, *SąlygosMazgas*, *Užklausa*]

Tam tikros veiksmų sekos vykdymo pasirinkimas identifikuojamas tokia schema:

PerėjimasIšSąlygosMazgoĮVeiklą
mazgas: SąlygosMazgas
veiklos_pavadinimas: Pavadinimas
sąlyga: SąlygosIšraiška

Schemos kintamasis „*veiklos_pavadinimas*“ identifikuoja veiksmų seką, kurią reikia iškviesti, jeigu sąlyga, užrašyta sąlygos išraiškoje, tenkinama.

Užklausos apdorojimas aprašomas įvairių veiksmų atlikimo seka bei jų ryšių aibėmis, schemoje „*UžklausosApdorojimas*“.

UžklausosApdorojimas
  pradžia: VeiklosPradžia
  pabaiga: VeiklosPabaiga
  pradžia_į_veiklą: VeiklosPradžia Pavadinimas
  pradžia_į_sąlygą: VeiklosPradžia SąlygosMazgas
  iš_veiklos_į_veiklą: Pavadinimas Pavadinimas
  iš_sąlygos_į_veiklą: PerėjimasIšSąlygosMazgoĮVeiklą
  iš_veiklos_į_sąlygą: Pavadinimas SąlygosMazgas
  iš_veiklos_į_pabaigą: Pavadinimas VeiklosPabaiga
  iš_sąlygos_į_sąlygą: SąlygosMazgas SąlygosMazgas
  veiklos: VeiksmųSeka

  ran *pradžia_į_veiklą v: veiklos v . pavadinimas*
  ran *iš_veiklos_į_veiklą v: veiklos v . pavadinimas*
  dom *iš_veiklos_į_veiklą v: veiklos v . pavadinimas*
  dom *iš_veiklos_į_sąlygą v: veiklos v . pavadinimas*
  dom *iš_veiklos_į_pabaigą v: veiklos v . pavadinimas*
    p: iš_sąlygos_į_veiklą p . veiklos_pavadinimas
      v: veiklos v . pavadinimas
    p: iš_sąlygos_į_veiklą p . mazgas
  = ran *pradžia_į_sąlygą* ran *iš_veiklos_į_sąlygą*
  dom *pradžia_į_veiklą* dom *pradžia_į_sąlygą* = *pradžia*
  pradžia_į_veiklą *pradžia_į_sąlygą* =
  pradžia_į_sąlygą *pradžia_į_veiklą* =
  ran *iš_veiklos_į_pabaigą* = *pabaiga*
    v1, v2: veiklos v1 . pavadinimas = v2 . pavadinimas v1 = v2
      ran *pradžia_į_sąlygą* ran *iš_veiklos_į_sąlygą*
     ran *iš_sąlygos_į_sąlygą*
  = *s: iš_sąlygos_į_veiklą s . mazgas* dom *iš_sąlygos_į_sąlygą*
      ran *pradžia_į_veiklą* ran *iš_veiklos_į_veiklą*
      s: iš_sąlygos_į_veiklą s . veiklos_pavadinimas
  = dom *iš_veiklos_į_veiklą* dom *iš_veiklos_į_sąlygą*
      dom *iš_veiklos_į_pabaigą*

Veiksmų pradžią identifikuoja schemos kintamasis „*pradžia*“.

Pradžia gali būti susijusi su viena veiksmų seka arba su sąlygos mazgu. Schemoje tai parodo ryšių „*pradžia_į_veikla*“ ir „*pradžia_į_sąlygą*“ aibės, kurios nurodo, koks veiksmas arba koks sąlygos mazgas vykdomas pradėjus užklausos apdorojimą. Pastebėtina, kad kiekviena pradžia gali būti susijusi tik su vienu veiksmu arba su vienu sąlygos mazgu.

Ryšių aibė „*iš_veiklos_į_veiklą“* nurodo ryšius tarp veiklų sekų. Tai yra aibė, kuri rodo, kokie veiksmai po kokio vykdomi.

Aibė „*iš_sąlygos_į_veiklą“* nurodo galimus perėjimus iš visų mazgų.

Aibė „*iš_veiklos_į_sąlygą“* nurodo visus perėjimus iš veiksmų sekos į sąlygos mazgą.

Aibė „*iš_veiklos_į_pabaigą“* nurodo veiksmų sekas, kurių vykdymu pasibaigia užklausos apdorojimas. Paprastai tai būna veiksmų sekos, užkraunančios tam tikrą puslapį generuojantį komponentą.

Portalo veiklos modelis gali būti aprašytas kaip užklausų ir jų apdorojimų aibė:

PortaloElgsena Užklausa UžklausosApdorojimas

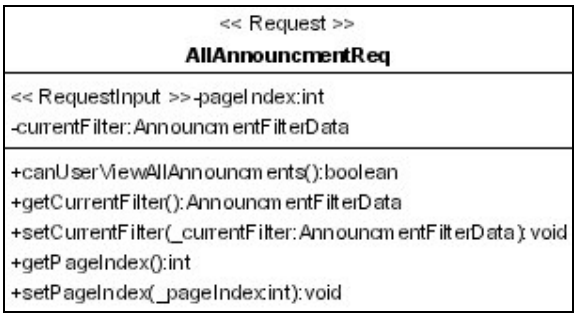
2.3 Portalo elgsenos aprašymas UML diagramomis

UML modeliavimo kalba sukurta sudėtingų sistemų architektūros specifikavimui. Kad pritaikyti MVA principus portalų projektavimui, reikia, kad elgseną būtų galima aprašyti UML diagramomis. Aptarsiu kaip, aukščiau aprašytas veiklos modelis, gali būti aprašytas naudojant UML diagramas ir žymėjimus.

2.3.1 Portalo apdorojamos užklausos

Portalo apdorojamos užklausos aprašomos UML klasių diagrama. Klasė - tai objektų abstrakcija. Ji apibūdina aibę objektų su vienodomis savybėmis (atributais), elgesiu (operacijomis), ryšiais su kitais objektais ir semantika. Klasės skirtos aprašymui supaprastinti, kad būtų galima jį taikyti ne konkrečiam objektui, o objektų aibei. Formalioje veiklos modelio specifikacijoje schema „*Objektas*“ tai supaprastintas klasės sąvokos UML kalboje aprašas.

Kiekvieną užklausą, kuri gali būti apdorojama portale, atspindi klasė diagramoje. Kadangi portalo modelyje gali būti aprašyta daug klasių, užklausos klases žymėsime stereotipu „*Request*“. Kadangi stereotipais remiantis bus atliekamas kodo generavimas, juos vadinsiu angliškais pavadinimais, kad išvengti problemų, kurios gali kilti naudojant įvarius įrankius. Užklausos taipogi gali turėti parametrus, kurie aprašomi užklausos klasės atributais su stereotipu – „*RequestInput*“. Užklausos parametrų perdavimas galėtų būti realizuotas naudojant HTTP GET metodą. Užklausa taip pat gali turėti bet kokį skaičių atributų ir metodų, kuriais operuojama užklausos apdorojimo metu.

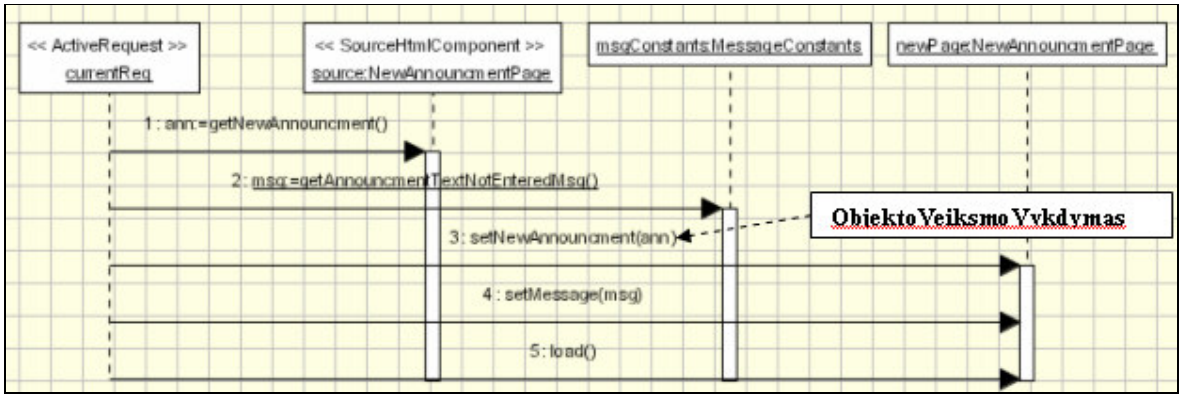


Pav. 2Užklausos pažymėjimo pavyzdys

2.3.2 Užklausų apdorojimas

Schema, kurią formaliame portalo veiklos modelyje pavadinome „*Objektas*“, UML modelyje taip pat vaizduojama klase. Klasės vardas atspindi schemos kintamąjį „*pavadinimas*“, o klasės metodai aibę „*veiksmai*“.

Formalaus aprašo schema „*VeiksmųSeka*“ UML modelyje vaizduojama sekų diagrama. Sekų diagramos naudojamos atvaizduoti sistemos objektų sąveiką ir pranešimus, kuriais keičiasi objektai. Svarbiausi sekų diagramų elementai yra objektai ir pranešimai, kuriais jie keičiasi. Objektas - tai elementas, turintis tam tikrą būseną ir elgesį, taip pat susijęs su kitais objektais. Kiekvienas objektas - tai tam tikros klasės egzempliorius. Pranešimus galima susieti su objektų metodais. Formalios specifikacijos schemą „*ObjektoVeiksmoVykdymas*“ galima aprašyti sekų diagramoje objektu ir pranešimu, kuris atitinka schemos kintamąjį „*veiksmas*“. Schemos kintamasis „*pavadinimas*“ atitinka sekų diagramos pavadinimą. Schemos kintamasis *„seka“* išreiškiamas sekų diagramos objektais ir pranešimų tarp jų apsikeitimais. Užklausų apdorojimo sekose dalyvauja, tų užklausų objektai, kurie pažymimi stereotipu „*ActiveRequest*“. Taip pat gali dalyvauti užklausą iškvietę vartotojo sąsajos komponentai, kurie aprašomi stereotipu „*SourceHtmlComponent*“.

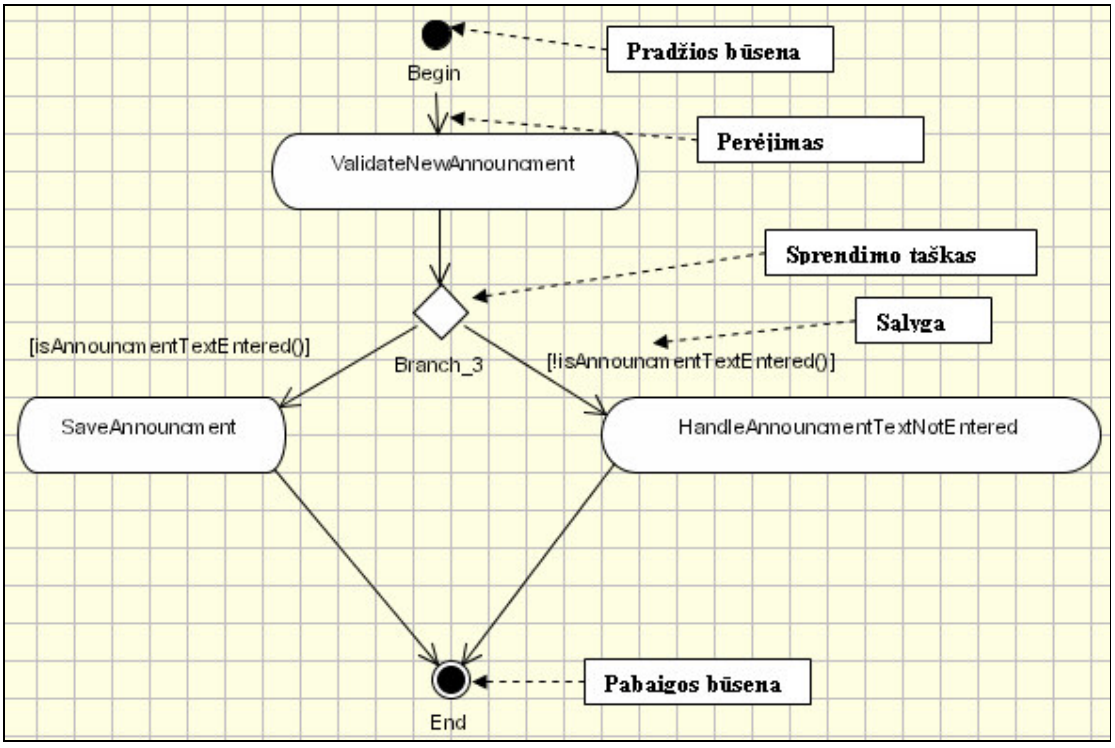


Pav. 3 Sekų diagramos pavyzdys (schemos „VeiksmųSeka“ UML atitikmuo)

Schema „*UžklausosApdorojimas*“ aprašoma veiklos diagrama. Veiklos diagramos vaizduoja veiksmus, jų atlikimo seką, perėjimus iš vieno veiksmo į kitą bei perėjimo sąlygas. „*ObjektoVeiksmoVykdymas*“ schemai aprašyti naudojami veiklų diagramos simboliai:

- Pradžios būsenos simbolis;
- Pabaigos būsenos simbolis;
- Veiksmo simbolis;
- Sprendimo taško simbolis;
- Perėjimas nuo vieno veiksmo prie kito simbolis.

Šios schemos kintamasis „*pradžia*“ gali būti pažymėtas veiklų diagramos simboliu „pradžios būsena“, sąlygos mazgai „sprendimo taško“ simboliu. Kintamieji „*pradžia_į_veiklą“*, „*pradžia_į_sąlygą“*, „*iš_veiklos_į_sąlygą“*, „*iš_veiklos_į_pabaigą“* gali būti atvaizduoti perėjimo simboliais.



Pav. 4 Užklausos apdorojimo veiklos diagrama (schemos „Užklausos apdorojimas“ atitikmuo)

Ryšys *Užklausa* *UžklausosApdorojimas* UML realizuojamas veiklos diagramai suteikus pavadinimą suformuotą iš užklausos pavadinimo ir žodžio „Processing“ (pvz.: jei užklausos

pavadinimas „NewsList“ tai jos apdorojimą aprašančios veiklos diagramos pavadinimas turėtų būti „NewsListPorcessing“).

2.4 Tipiniai portalo veikloje dalyvaujantys objektai

2.4.1 Duomenų saugojimo/skaitymo objektai

Pagrindinis žiniatinklio objektas yra informacija, kuri yra surenkama ir išsaugojama. Labai svarbu informacijos srautus identifikuoti ir specifikuoti modelyje. Tipiniai tokie objektai yra tie kurie dirba su duomenų baze.

2.4.1.1 Objektai skirti darbui su duomenų baze

Šiais laikais duomenų saugojimui dažniausiai naudojamos reliacinės duomenų bazės. Ne išimtis ir žiniatinklio sistemos. Darbas su reliacine duomenų baze pasižymi tam tikra specifika. T.y. bendravimui su duomenų baze naudojama kita kalba SQL. SQL kalba skirtingoms reliacinių duomenų bazių valdymo sistemoms skiriasi, tačiau pastebėtina, kad duomenų įrašymas, atnaujinimas bei trynimas – tai standartinės operacijos, kurių realizacija skiriasi tik tam tikrais elementais.

Duomenų bazės struktūros modelį taip pat patartina aprašyti UML modelyje. Daugelis UML diagramų sudarymo įrankių palaiko duomenų bazės modelio sudarymą – taip pat jos sukūrimą. Aš nenagrinėsiu principų, kaip sudarinėti duomenų bazės modelį, tačiau man svarbu programinė veiksmų su duomenų baze realizacija. Paprastai tiesiogiai manipuliuoti su iš duomenų bazės gautais duomenimis nepatogu, todėl iš tų duomenų yra konstruojami objektai, kuriuos patogu naudoti programuojant. Tokius objektus reikia aprašyti modelyje. Todėl apibrėžiami tokie baziniai duomenų modelio elementai:

- Duomenų bazė vaizduojama paketu, kurio stereotipas –„*Database*“;
- Duomenų bazių lentelių struktūra aprašoma klasėmis su stereotipais „*DataTable*“. Klasės turi visiškai atitikti duomenų bazės lenteles. T.y. kiekvienai duomenų bazės lentelei, kuria bus naudojamasi portale, turi būti sukurta klasė modelyje, duomenų bazės pakete. Tos klasės pavadinimas turi būti toks kaip lentelės ir ji turi turėti atributus tokius pačius kaip lentelės stulpeliai;
- Lentelės pirminį raktą sudarantys atributai nurodomi stereotipu „*PrimaryKey*“. Jie nurodomi tam, kad būtų galima realizuoti tose lentelėse saugomų duomenų redagavimą;
- Stulpeliai, kurių reikšmės kiekvienam naujam įrašui didinamos vienetu, nurodomos naudojant stereotipą „*AutoIncrement*“.

Duomenų bazėje saugomų duomenų aprašymo užtenka, kad sugeneruoti komponentus, kurie su tais duomenimis galėtų atlikti bazines operacijas: paėmimas, šalinimas, redagavimas. Komponentai, realizuojantys šias operacijas, modelyje turėtų būti identifikuojami klasėmis su stereotipu „*DBDataManager*“. Šio tipo klasės turėtų turėti metodus skirtus dirbti tik su duomenų baze. Svarbu pažymėti, kad šiuose komponentuose reiktų vengti bet kokio iš duomenų bazės gautų duomenų transformavimo ar skaičiavimų, nes tai padidintų sistemos sudėtingumą. Šie komponentai turėtų griežtai užsiimti darbu su duomenų baze. Metodai, skirti duomenims, modelyje pažymėtais stereotipu „*DataTable*“, įterpti į duomenų bazę, atnaujinti ar pašalinti iš jos, turėtų būti atitinkamai pažymimi stereotipu „*SqlInsert*“, „*SqlUpdate*“ ir „*SqlDelete*“. Šie metodai turėtų turėti vieną parametrą, kurio tipas turėtų būti „*DataTable*“ stereotipu pažymėta klasė.

2.4.1.2 Vartotojo seanso metu saugojami duomenys

Duomenys, kurie laikomi tik vartotojo seanso metu, yra labai dažnai naudojami portaluose. Pavyzdžiui vartotojas prisijungia prie portalo įvesdamas prisijungimo vardą ir slaptažodį. Reikia, kad visi sekantys vartotojo kviečiami puslapiai tai žinotų. Tokių duomenų išsaugojimą ir apsikeitimą palaiko visi serveriai. Dažnai tokių duomenų srautai portaluose nemodeliuojami ir neidentifikuojami, tačiau labai dažnai jie dalyvauja atliekant įvairius veiksmus. Todėl norint užtikrinti portalo vykdomų procesų aiškumą, reikia juos specifikuoti. Tokius duomenis aprašysime klasėmis, kurias pažymėsime stereotipu „*SessionDataManager*“. Ši klasė turėtų turėti tik atributus, ir jų reikšmių priskyrimo ir paėmimo metodus.

2.4.2 Vartotojo sąsajos objektai

Pagrindinė portalo funkcija yra teikti ir rinkti informaciją. Informacijos pateikimas turi būti vaizdingas, patogus ir pastoviai atnaujinamas. Tai veda prie to, kad informacijos pateikimo aspektai portalo gyvavimo metu turi tendenciją keistis. Todėl būtina užtikrinti, kad, HTML generavimą aprašantys komponentai, būtų kiek galima aiškesni. Portalų išskirtina ypatybė yra tai, kad vartotojo sąsaja yra aprašoma HTML kalba, kuri buvo sukurta statinės informacijos vaizdavimui. Tačiau šiais laikais portalų turinys generuojamas dinamiškai ir tam realizuoti yra naudojamos įvairios technikos ir priemonės. Populiariausia yra scenarijų technologija, kuri paremta tuo, kad puslapio tekste programinis kodas yra išskiriamas specialiomis žymėmis ir paskui vykdomas serverio. Tai leidžia įvairias funkcijas tokias kaip duomenų paėmimas, jų transformavimas ir atnaujinimas aprašyti tuose pačiuose puslapiuose. Toks sumaišymas veda prie to, kad scenarijų puslapių išeities tekstai tampa labai sudėtingi ir nevaizdūs, kas apsunkina palaikymą. Todėl siūloma, kad komponentai, užsiimantys HTML generavimu neatliktų jokių kitų veiksmų išskyrus informacijos vaizdavimą ir surinkimą. Tuo iš kur gaunami vaizduojami duomenys, kur dedami surinkti duomenys jie visiškai

nesirūpina. Kokie duomenys bus reikalingi tam, kad sugeneruoti HTML, bei kokie duomenys gali būti įvesti, būtinai turi būti parodyti modelyje.

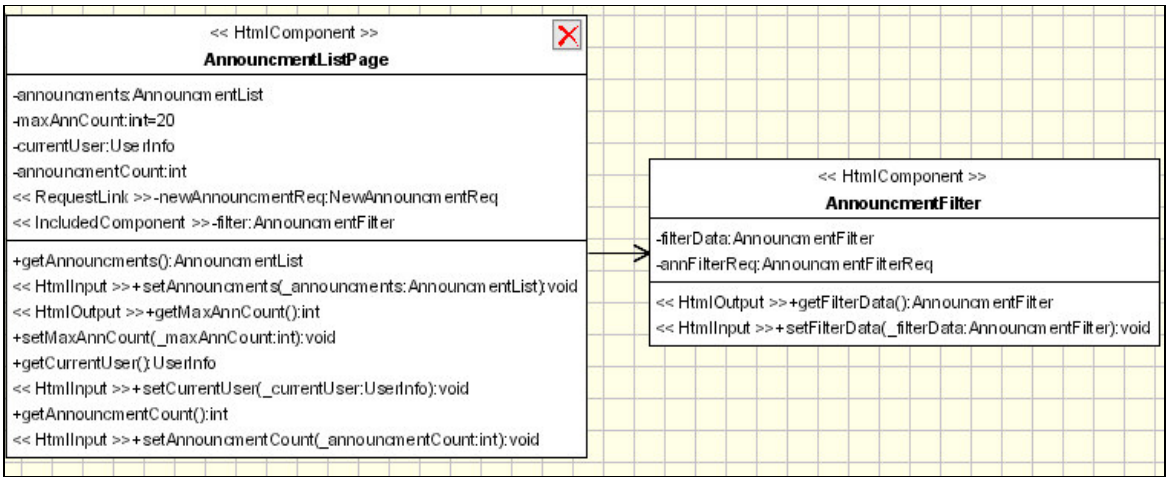
HTML generuojantys komponentai modelyje vaizduojami klasės elementu su stereotipu „*HtmlComponent*“. Komponentui reikalingi duomenys, bei duomenys, kuriuos komponentas grąžina, vaizduojami metodais. Užklausos, kurias galima iškviesti iš tų komponentų, aprašomos nurodant asociacijos ryšį tarp komponento ir užklausos tipo objekto. Taip pat ši klasė turėtų turėti metodą pavadinimu „*load“*, kuris būtų naudojamas veiklose nurodant komponento užkrovimą.

HTML generavimui reikalingi duomenys nurodomi metodų, pažymėtų stereotipu „*HtmlInput*“, parametrais.

Duomenys, kurie gali būti įvedami ir surinkti komponente, aprašomi metodais su stereotipu „*HtmlOutput*“. Šių metodų grąžinamo parametro tipas nurodo, kokius duomenis komponentas grąžina.

Kiekvienas HTML generuojantis komponentas gali būti sudarytas iš kitų, tokių pačių, komponentų. Tokiu atveju šie komponentai aprašomi atributais, su stereotipu „*IncludedComponent*“.

Užklausos, kurios gali būti atliekamos iš puslapio aprašomi vartotojo sąsajos klasės atributais su stereotipu „*RequestLink*“.



Pav. 5 Vartotojo sąsajos komponento UML specifikacijos pavyzdys

2.4.3 Valdiklis

Tai portalo komponentas, kuris atlieka pagrindines užklausų apdorojimo valdymo funkcijas: jis turi identifikuoti gautą užklausą, iškviesti jos apdorojimą ir išsaugoti informaciją apie tai, koks paskutinis varotojo sąsajos komponentas buvo užkrautas. Valdiklis portalo modelyje turėtų būti vaizduojamas klasės simboliu, su stereotipu „*RequestControler*“.

2.5 Portalo veiklos realizacijos išeities teksto generavimo aspektai

Kadangi portalo elgsenos modelio sudaryme nenaudojamos jokios platformai specifinės sąvokos, tai nuo realizacijos naudojamos platformos nepriklauso. Greito portalų kūrimo užtikrinimui, ir MVA principų įgyvendinimui, reikia, kad pagal sudarytą portalo veiklos modelį, būtų galima generuoti specifinį tam tikrai platformai, modelį ir realizaciją. Tai yra įmanoma padaryti, kadangi mūsų elgsenos aprašyme naudojami elementų atitikmenys yra naudojami beveik visose programavimo kalbose naudojamose žiniatinklio portalų kūrime. Tai yra kiekvienoje programavimo kalboje yra galimybė aprašyti objektus ar modulius, sudarytus iš funkcijų, kurias jis gali atlikti.

Pagal sudarytą portalo elgsenos modelį galima generuoti:

1. Valdiklį, kuris gautų visas portalo HTTP užklausas, turėtų sąrašą visų užklausų ir iškviestu jų apdorojimą. Valdiklyje galėtų būti realizuotas bendras mechanizmas, netikėtų klaidų apdorojimui;
2. Užklausų objektus. Vartotojas naršo po portalą naudodamasis interneto naršykle, įvesdamas puslapio URL adresą, pasirinkdamas nuorodas ar mygtukus. Bet kokiu atveju, kokį puslapį nori gauti vartotojas identifikuoja URL. Todėl kiekviena užklausa turėtų turėti funkcijas, suformuojančias tą URL. Šis URL turėtų nurodyti valdiklio funkcijas atliekančio komponento adresą, su parametrais identifikuojančiais užklausą. Šios funkcijos turėtų būti kviečiamos iš vartotojo sąsajos komponentų generuojant puslapius. Taipogi užklausos tipo objektai turėtų turėti metodą, iškviečiantį užklausos apdorojimo vykdymą. Šis metodas pagal URL duomenis nuspręstų ar užklausos veiksmai turėtų būti vykdomi ir įvykdytų aprašytus veiksmus jei taip.
3. Seanso metu reikalingų duomenų saugojimu užsiimančius objektus;
4. Užklausos vykdymą sudarančių objektų kvietimo sekas.

2.6 Išvados

Šioje darbo dalyje buvo pateiktas formalus portalo veiklos modelio, tinkamo automatizuoti tam tikrų portalo realizacijos komponentų kūrimą, aprašas, buvo aprašyta kaip tokį modelį būtų galima sudaryti naudojant UML kalbą. Taip pat buvo sudarytos tam tikrų tipinių portaluose dalyvaujančių komponentų projektavimo taisyklės, kurios leistų išnaudoti MVA principus. Toliau eksperimentinėje dalyje pademonstruosim, kokie komponentai ir kaip gali būti generuojami. Svarbu yra tai, kad aprašyti tipiniai portalo komponentai yra ne visi komponentai, kurių realizacija galėtų būti generuojama. Aprašas tokių komponentų, galėtų ir turėtų būti plečiamas, portalų

projektavimo metu. T.y. portalų projektuotojas, turėtų stengtis projektavimui naudoti aprašytą metodiką. Tačiau, tuo pačiu metu, jis galėtų identifikuoti ir kitus komponentus, kurių kūrimas nėra automatizuotas tačiau galėtų būti automatizuotas, aprašant juos modelyje. Pabaigus portalo realizaciją ir patikrinus, kad tie komponentai veikia, būtų galima realizuoti tų komponentų generavimą pagal UML modelį.

3 METODIKOS PATIKRINIMO EKSPERIMENTAS

3.1 Eksperimento tikslas

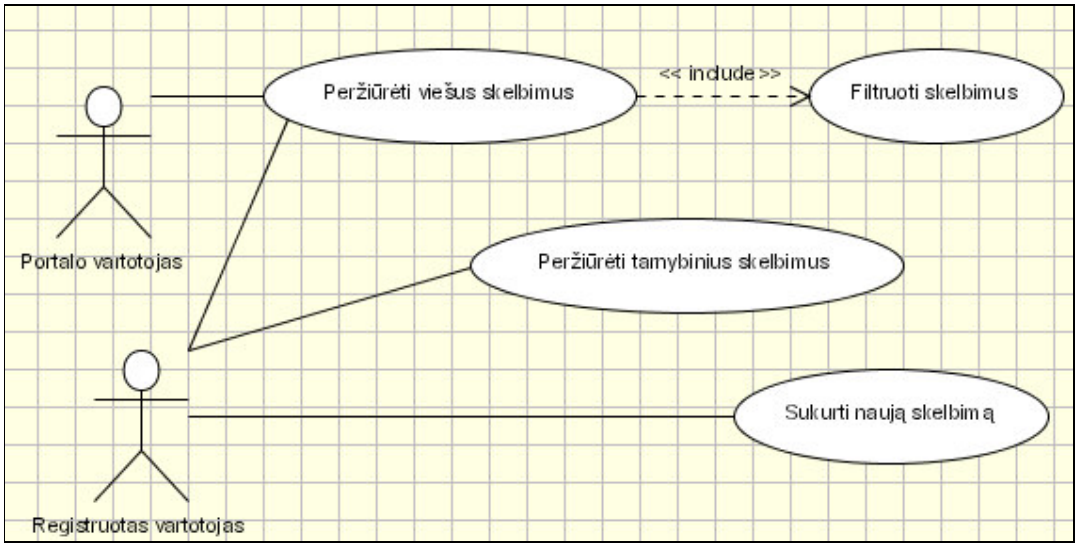
Šiame darbe aprašytos metodikos esmė yra portalo veiklos aprašymas ir modeliu valdomos architektūros principų realizacija. Eksperimento tikslas buvo sudaryti tam tikro portalo veiklos modelį UML diagramomis ir ištirti to modelio XMI aprašo transformavimo į tam tikrą realizaciją galimybes.

3.2 Sudarytas portalo modelis

3.2.1 Portalo funkcijos

Eksperimentui atlikti buvo aprašyta portalo dalis, su tokiomis funkcijomis:

- paprasti vartotojai gali peržiūrėti viešus skelbimus;
- registruoti vartotojai gali peržiūrėti visus skelbimus, bei sukurti naujus;
- galima atlikti skelbimų filtravimą.

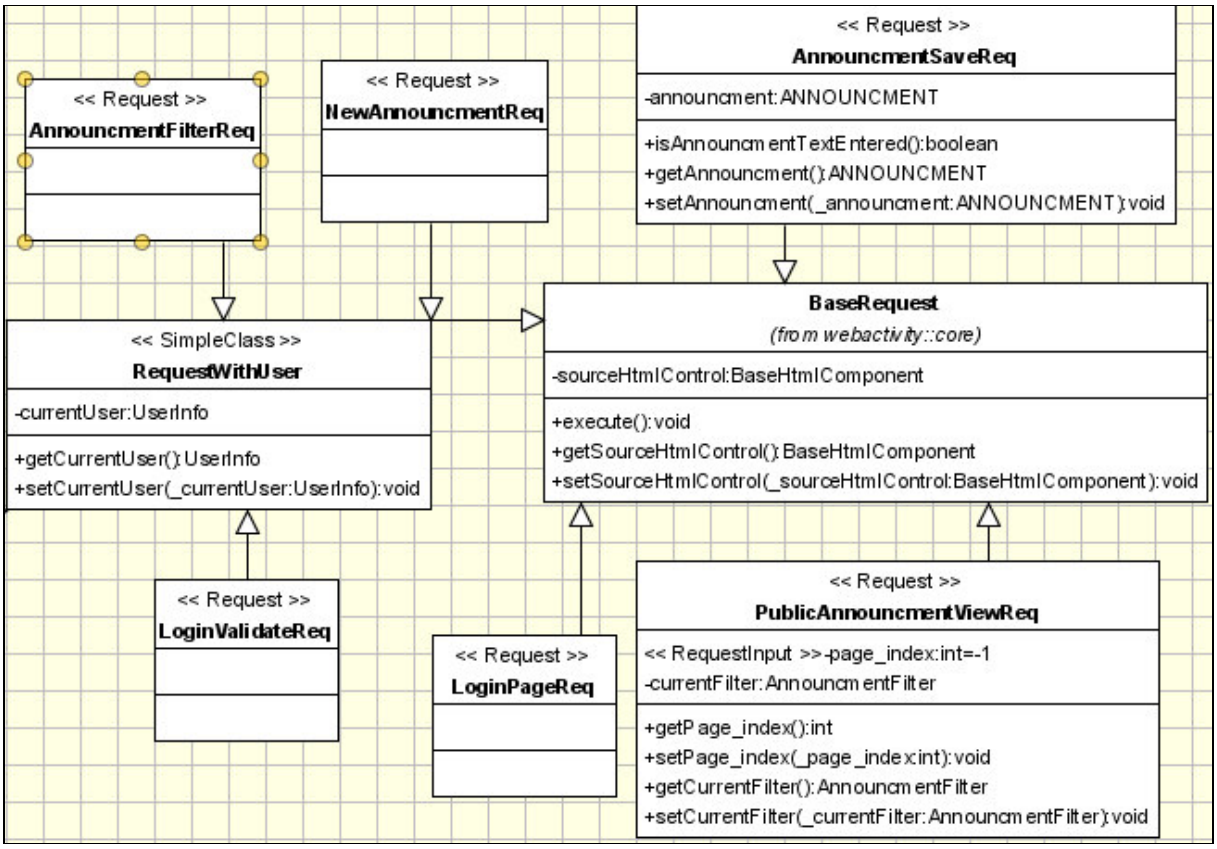


Pav. 6 Portalo funkcijos

Eksperimentui atlikti buvo sudarytas portalo veiklos modelis. Šiame modelyje buvo panaudotos visos metodikoje aprašytos sąvokos.

3.2.2 Portalo užklausų modelis

Portalo funkcijas realizuojanti užklausų aibė parodyta Pav. 7.

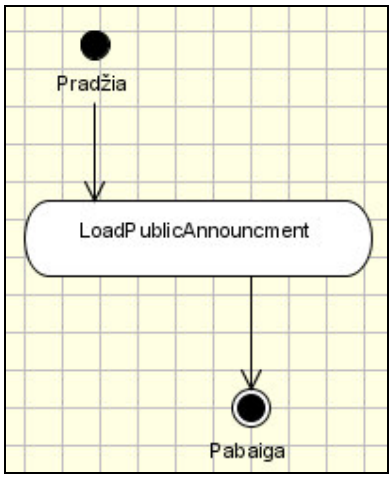


Pav. 7 Portalo užklausų diagrama

Toliau aptarsiu visas užklausas ir jų apdorojimo veiklos modelius.

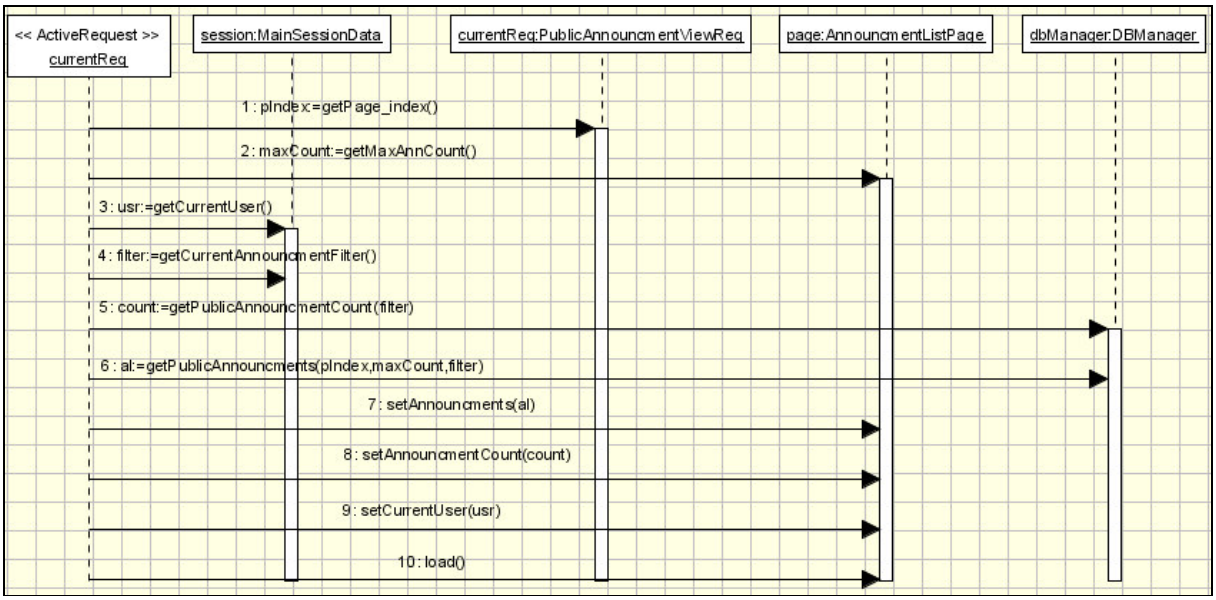
3.2.2.1 Užklausa, kurios pavadinimas „*PublicAnnouncmentViewReq*“

Tai užklausa, kurios rezultatas – puslapis su viešais skelbimais. Jį gali iškviesti kiekvienas portalo vartotojas iš vartotojo sąsajos komponento pavadinimu „*LoginPage“* puslapio. Jos apdorojimo veiklos diagrama parodyta Pav. 8.



Pav. 8 „PublicAnnouncmentViewReq“ užklausos apdorojimo diagrama

Užklausos apdorojimas susideda tik iš veiksmų sekos pavadinimu „*LoadPublicAnnouncment*“ iškvietimo.



Pav. 9 Veiksmų seka „LoadPublicAnnouncment“

Užkraunant viešus skelbimus, dalyvauja tokie objektai:

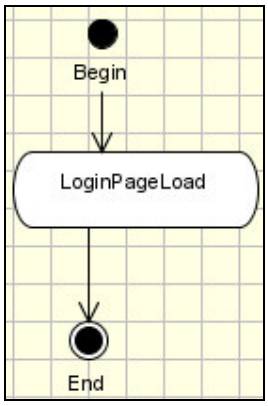
- „*MainSessionData*“ – tipo objektas, kuriame saugoma informacija apie prisijungusį vartotoją;

- „*DBManager“* – tipo objektas, turintis metodus viešų skelbimų paėmimui iš duomenų bazės;
- bei „*AnnouncmentListPage*“ – objektas, realizuojantis vartotojo sąsajos generavimo funkcijas.

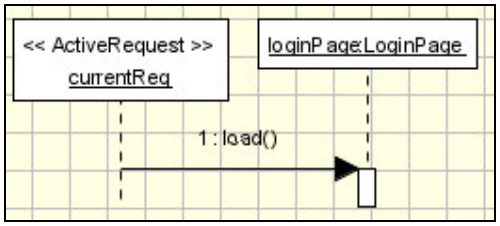
Objektas „*currentReq“* simbolizuoja apdorojamą užklausos objektą.

3.2.2.2 Užklausa, kurios pavadinimas „*LoginPageReq*“

Tai užklausa užkraunanti prisijungimo puslapį. Tai atlieka veiksmų seka pavaizduota Pav. 11.



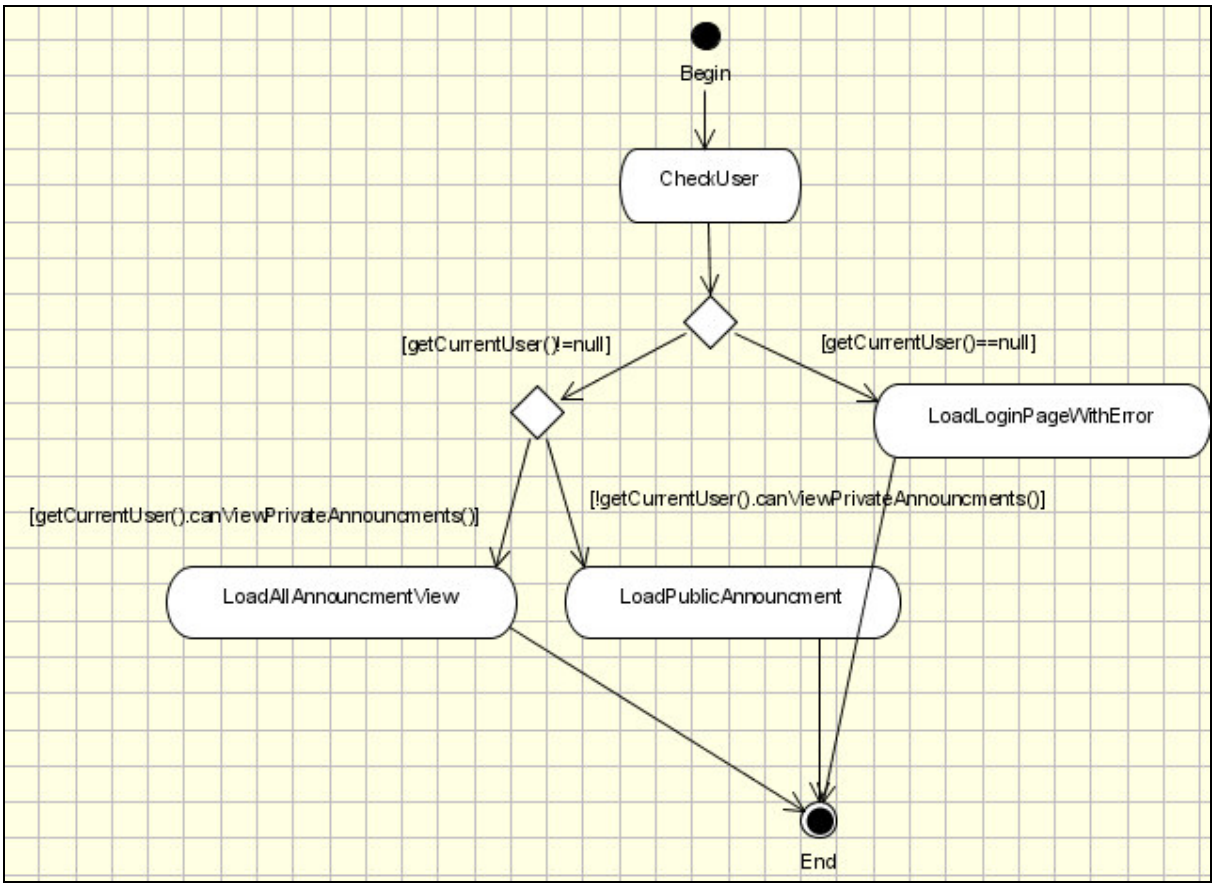
Pav. 10 Užklausos „LoginPageReq“ apdorojimo diagrama



Pav. 11 Veiksmų seka „LoginPageLoad“

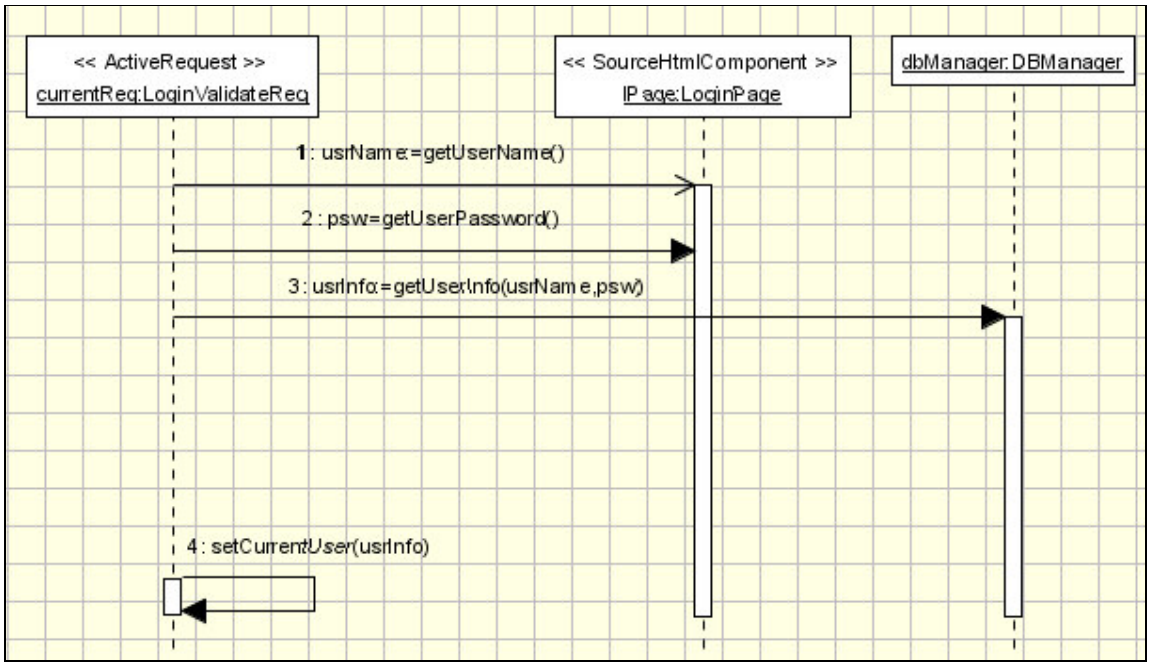
3.2.2.3 Užklausa, kurios pavadinimas „*LoginValidateReq*“

„*LoginValidateReq*“ užklausa iškviečiama iš prisijungimo lango, tam, kad patikrinti prisijungimo vardą ir slaptažodį. Jos apdorojimo veiklos diagrama parodyta Pav. 12.



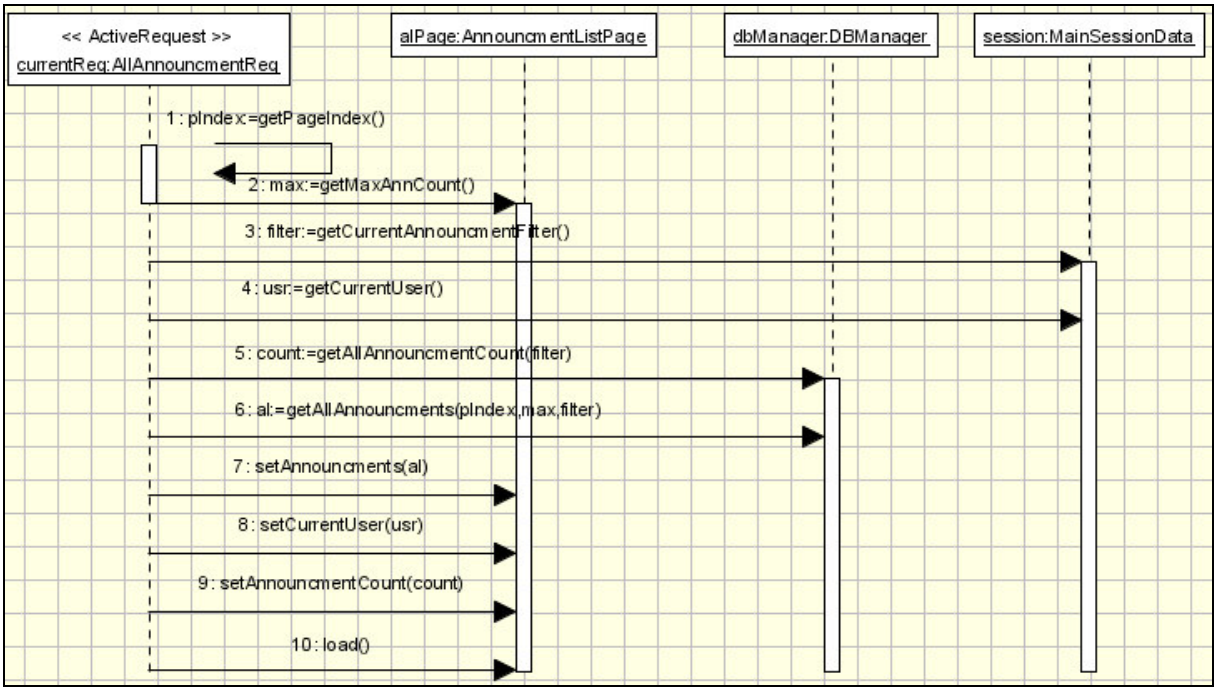
Pav. 12 Užklausos „LoginValidateReq“ apdorojimo diagrama

Pirmiausia įvykdoma veiksmų seka pavadinimu „*CheckUser*“. Joje aprašyta kaip, pasinaudojant vartotojo sąsajos komponento metodais, gaunama vartotojo suvesta prisijungimo informacija, bei kaip ji palyginama su duomenų bazėje esančia.



Pav. 13 Veiksmų seka „CheckUser“

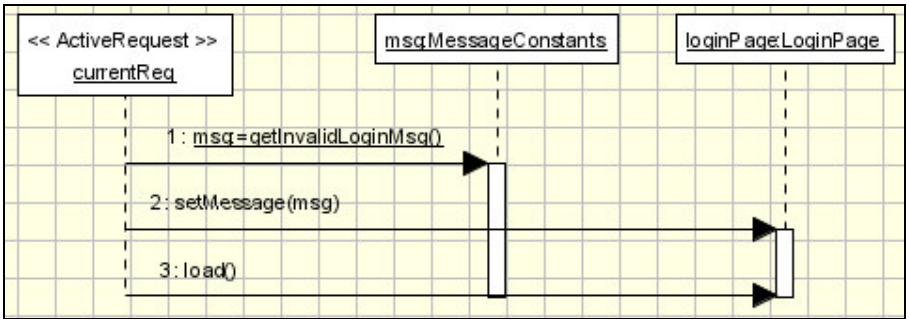
Jeigu loginė išraiškos „getCurrentUser!=null“ ir „getCurrentUser().canViewPrivateAnnouncments()“ tenkinamos, tada kviečiama veiksmų seka pavadinimu „*LoadAllAnnouncmentView*“.



Pav. 14 Veiksmų seka „LoadAllAnnouncmentView“

Jeigu loginė išraiškos „getCurrentUser!=null“ ir „!getCurrentUser().canViewPrivateAnnouncments()“ tenkinamos, tada kviečiama veiksmų seka pavadinimu „*LoadPublicAnnouncment*“ (Pav. 9).

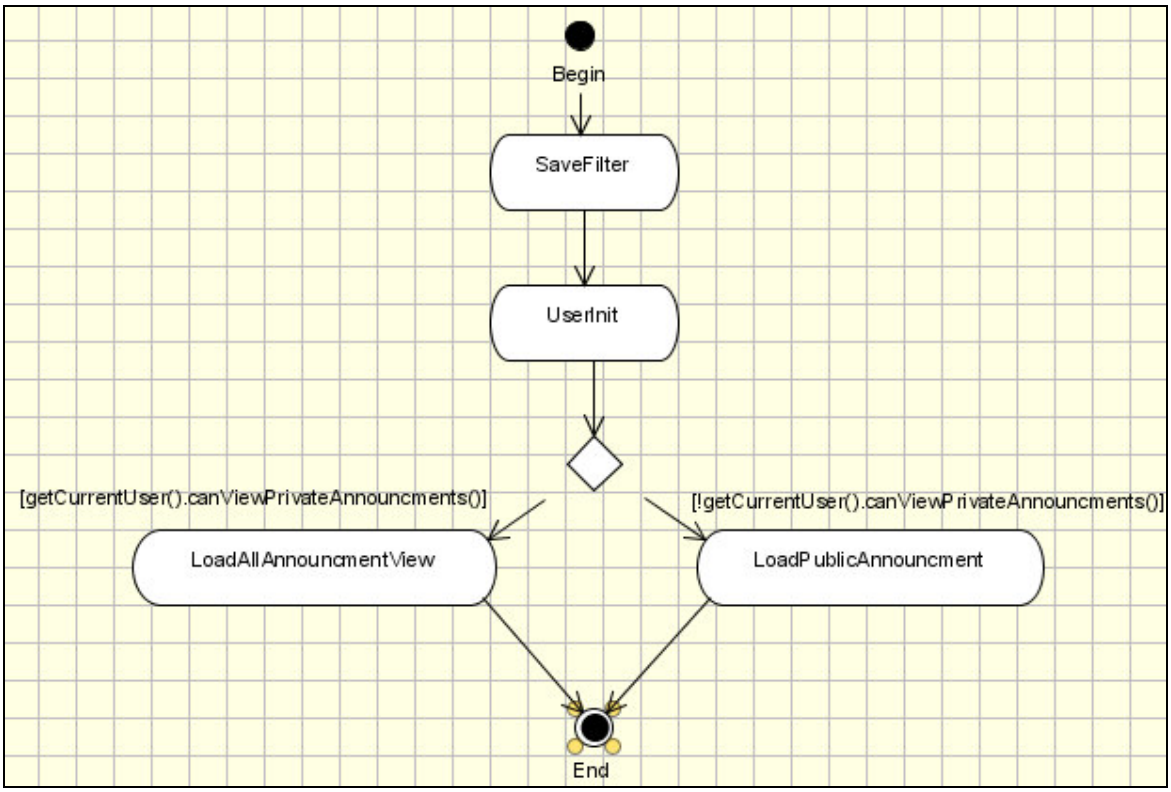
Jeigu loginė išraiškos „getCurrentUser==null“ rezultatas teigiamas tada vykdoma veiksmų seka „*LoadLoginPageWithError*“.



Pav. 15 Veiksmų seka „LoadLoginPageWithError“

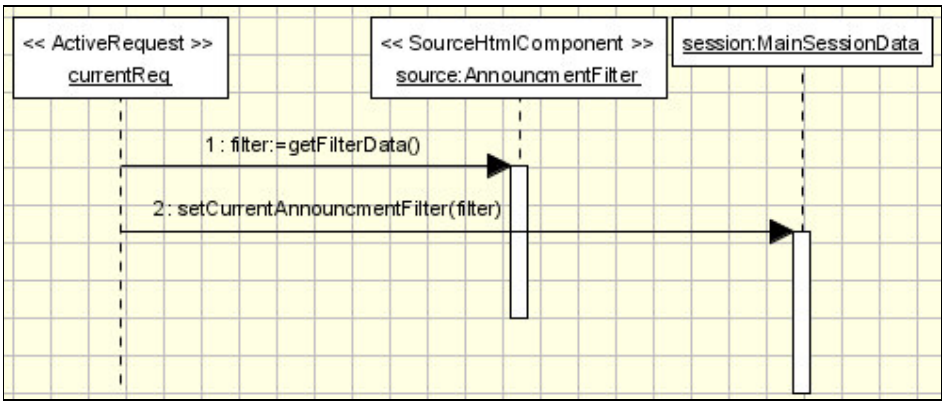
3.2.2.4 Užklausa, kurios pavadinimas „*AnnouncmentFilterReq*“

Ši užklausa iškviečiama iš „*AnnouncmentFilter*“ komponento, kai norima atlikti puslapių filtravimą. Jos vykdymas, tai filtro išsaugojimas sesijoje (veiksmų seka „*SaveFilter*“) ir viešų arba visų skelbimų atfiltruotų pagal filtro kriterijus parodymas.



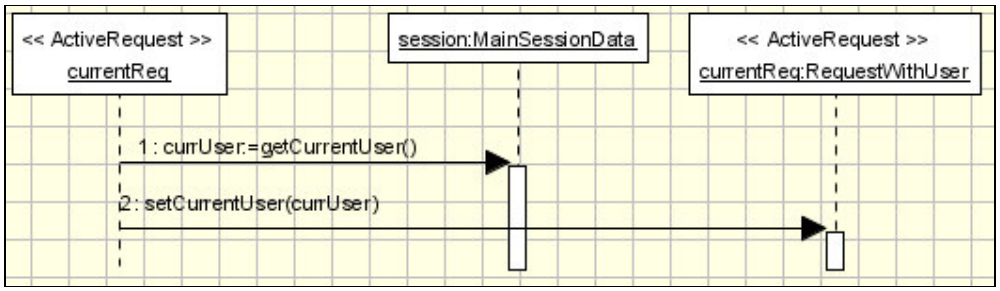
Pav. 16 Užklausos „AnnouncmentFilterReq“ apdorojimo diagrama

„*SaveFilter*“ veiksmų seka išsaugo filtro duomenis, grąžintus „*AnnouncmentFilter*“ komponento, vartotojo sesijoje.



Pav. 17 Veiksmų seka „SaveFilter“

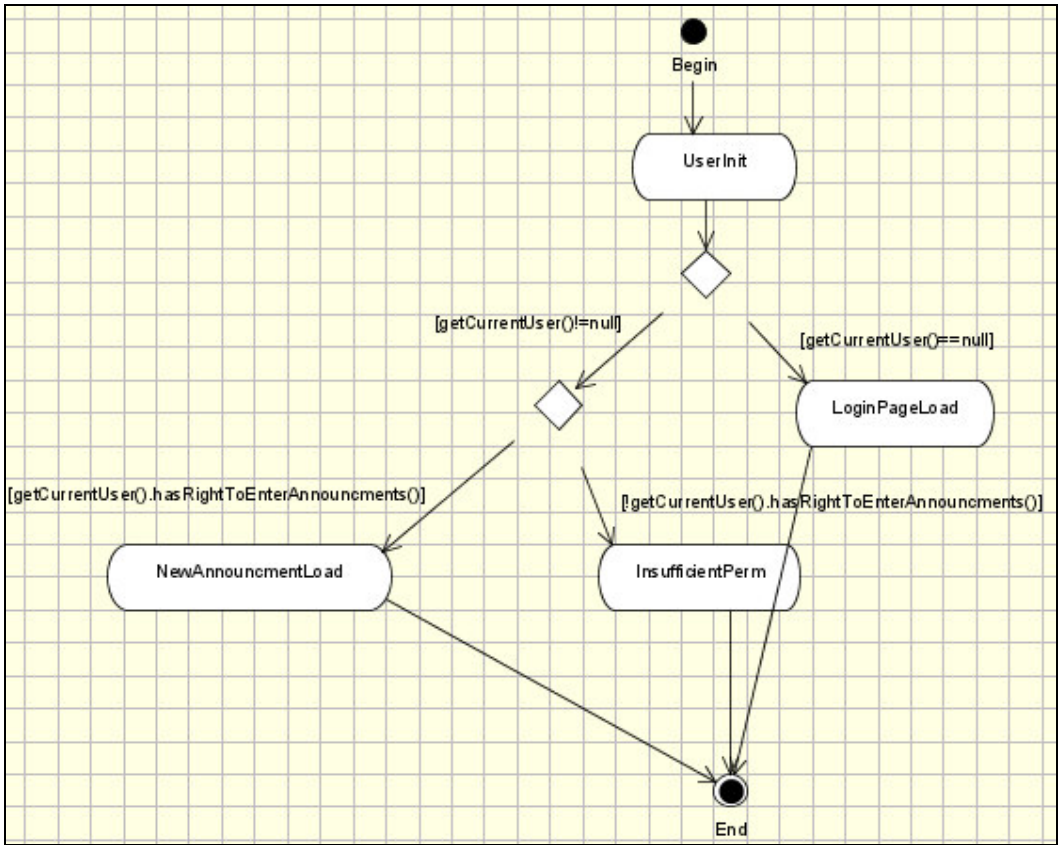
Veiksmų sekoje „*UserInit*“ parodyta, kad vartotojo informacija paimama, iš klasės pavadinimu„*MainSessionData*“.



Pav. 18 Veiksmų seka „UserInit“

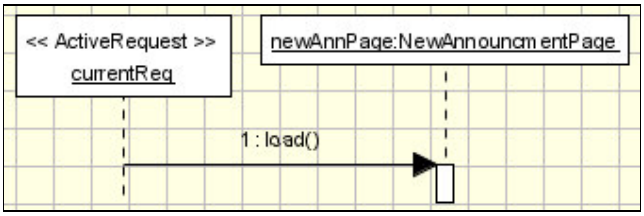
3.2.2.5 Užklausa, kurios pavadinimas „*NewAnnouncmentReq*“

Ši užklausa gali būti iškviečiama iš „*AnnouncmentListPage*“ vartotojo sąsajos komponento, kai vartotojas nori įvesti naują skelbimą. Tokiu atveju yra patikrinama ar vartotojas turi teisę įvedinėti skelbimus, ir jei taip pateikiamas skelbimo įvedimo langas.



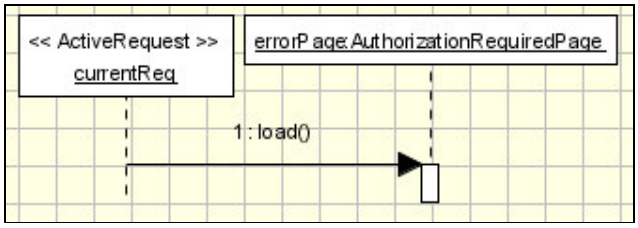
Pav. 19 Užklausos „NewAnnouncmentReq“ apdorojimo diagrama

Veiksmų seka, kuri atliekama, kai vartotojas yra prisijungęs ir turi teisę įvesti skelbimus pateikta Pav. 20.



Pav. 20 Veiksmų seka „NewAnnouncmentLoad“

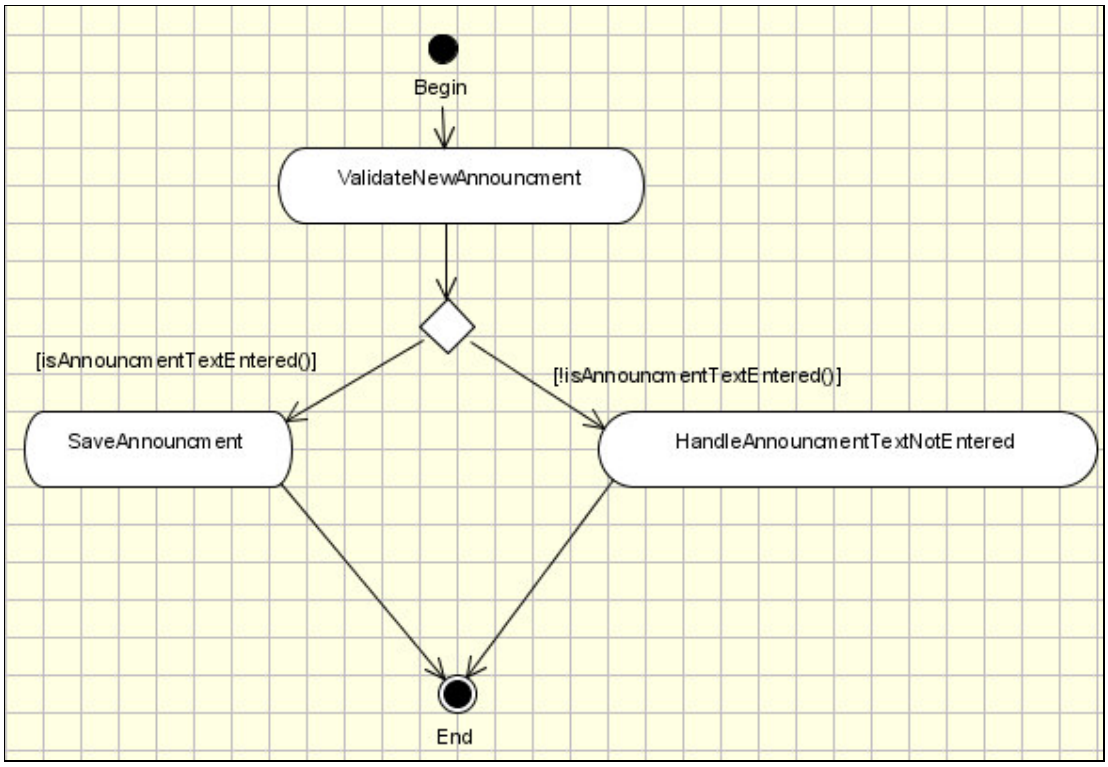
Veiksmų seka, kuri atliekama, kai vartotojas neturi teisių įvesti naują skelbimą, pateikta Pav. 21.



Pav. 21 Veiksmų seka „InsufficientPerm“

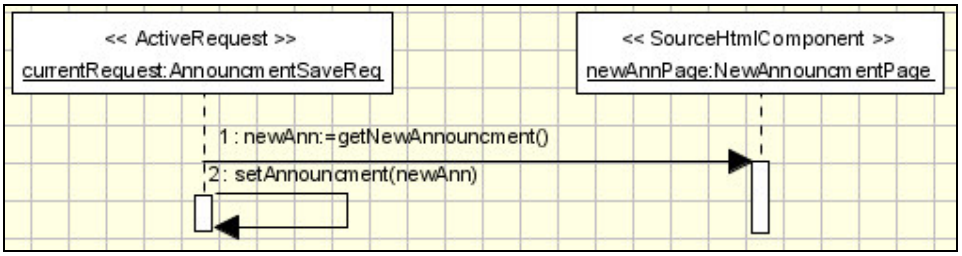
3.2.2.6 Užklausa, kurios pavadinimas „*AnnouncmentSaveReq*“

Ši užklausa iškviečiama iš „*NewAnnouncmentPage*“ komponento. Šios užklausos apdorojimas susideda iš patikrinimo ar skelbimas netuščias ir, jei netuščias, jo išsaugojimo.



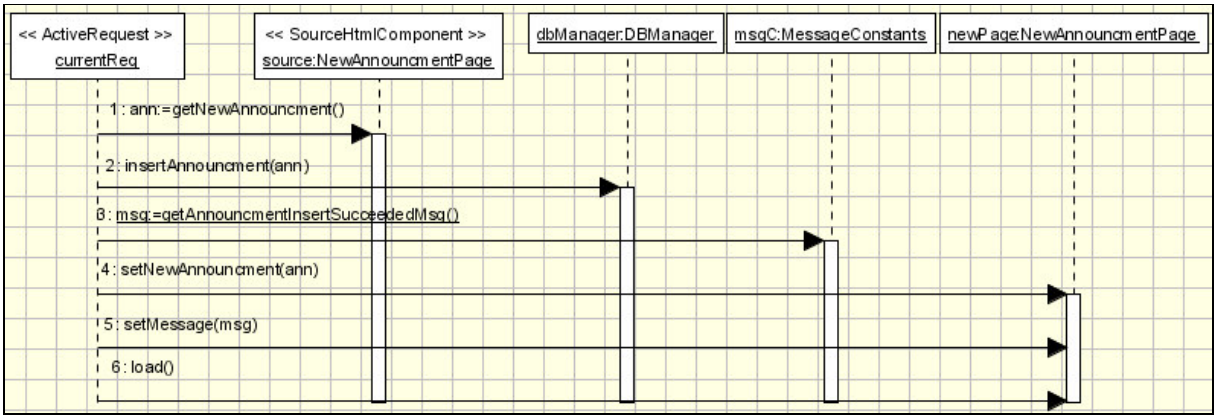
Pav. 22 Užklausos „AnnouncmentSaveReq“ apdorojimo diagrama

Seka, kurios pavadinimas „*ValidateNewAnnouncment*“, įvykdoma tik gavus užklausą naujo skelbimo įvedimui.



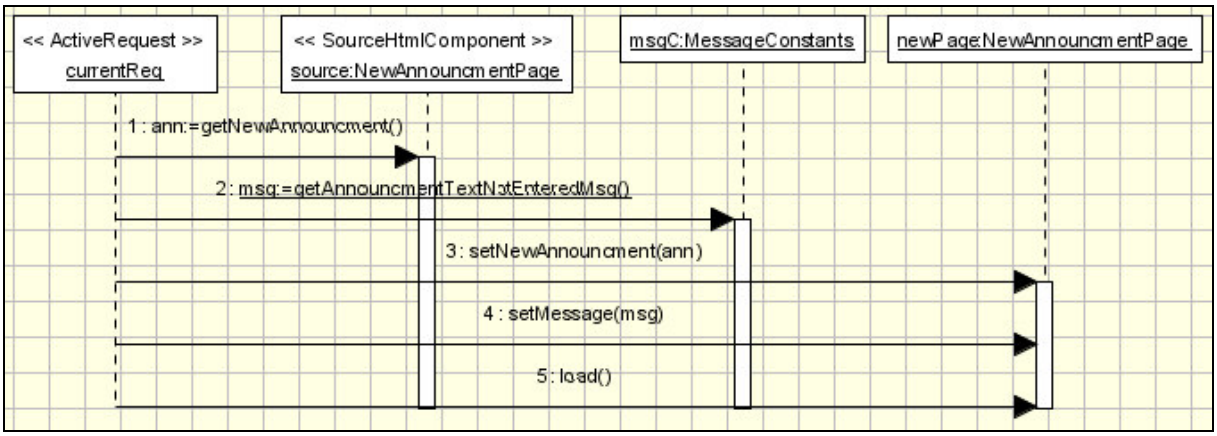
Pav. 23Veiksmų seka „ValidateNewAnnouncment“

Kai įvestas skelbimas validus, atliekama veiksmų seka parodyta Pav. 24.



Pav. 24 Veiksmų seka „SaveAnnouncment“

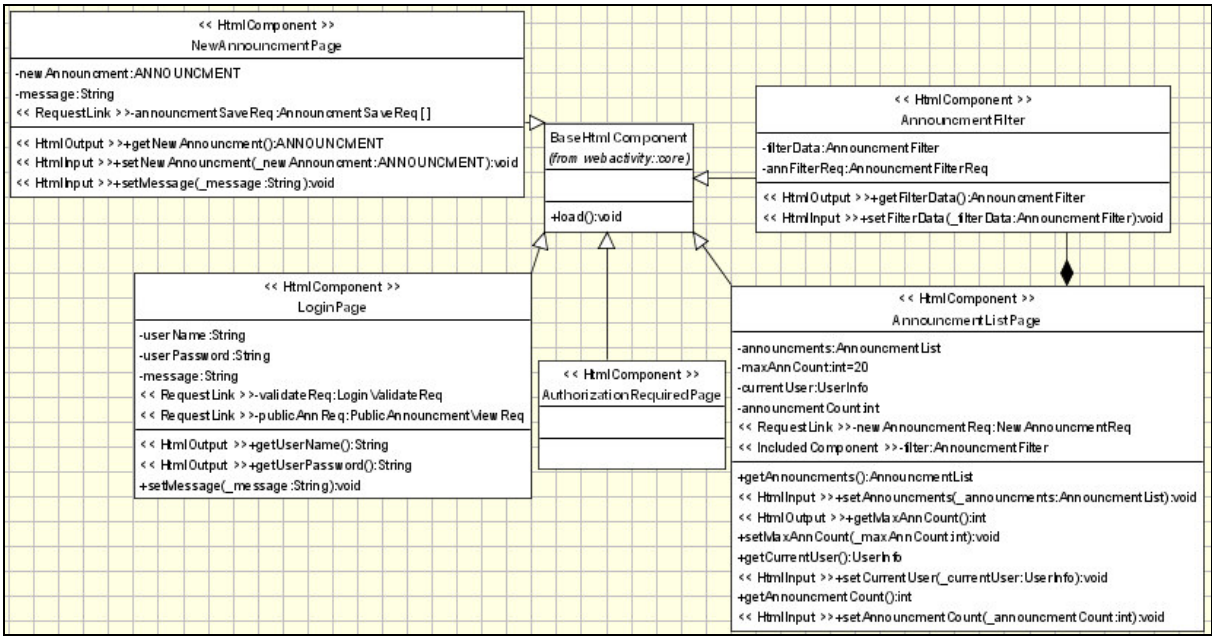
Kai įvestas skelbimas neteisingas, vartotojui parodomas pranešimas. Tai parodyta veiksmų sekoje „*HandleAnnouncmentTextNotEntered*“.



Pav. 25 Veiksmų seka „HandleAnnouncmentTextNotEntered“

3.2.3 Vartotojo sąsajos komponentai

Buvo sudaryta tokia, portalo veikloje dalyvaujančių vartotojo sąsajos komponentų, diagrama.



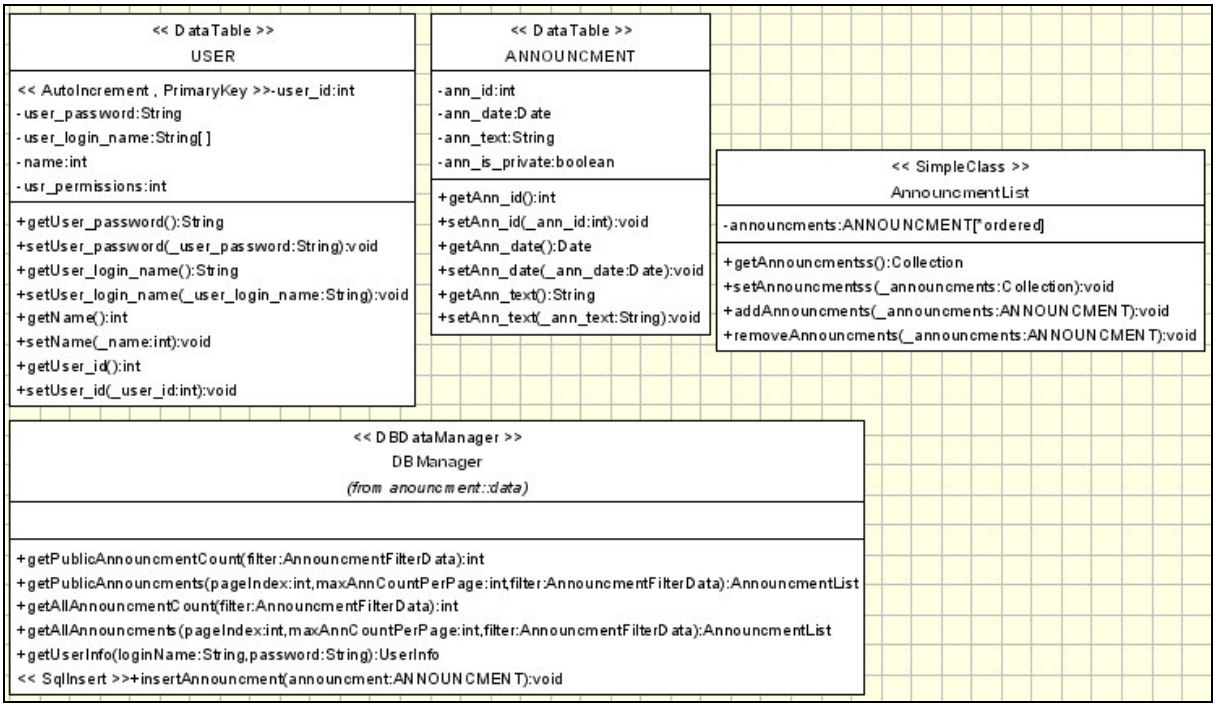
Pav. 26 Vartotojo sąsajos komponentų klasių diagrama

3.2.4 Duomenų bazės komponentai

Darbui su duomenų baze realizuoti aprašyti tokie objektai:

- „USER“ ir „ANNOUNCMENT“– klasės, simbolizuojančios duomenų bazės lenteles. Šios klasės priskirtos paketui, kurio pavadinimas „*portal_db*“, ir kuris pažymėtas stereotipu „*Database*“.

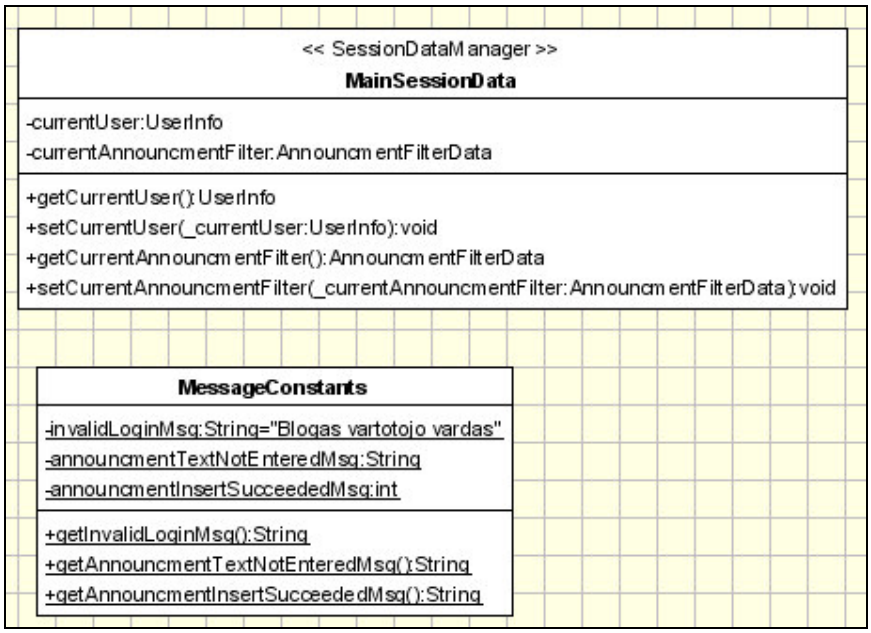
- „DBManager“ – klasė vaizduojanti veiksmus su duomenų baze atliekantį komponentą.



Pav. 27 Duomenų bazės komponentų diagrama

3.2.5 Kiti duomenų šaltiniai

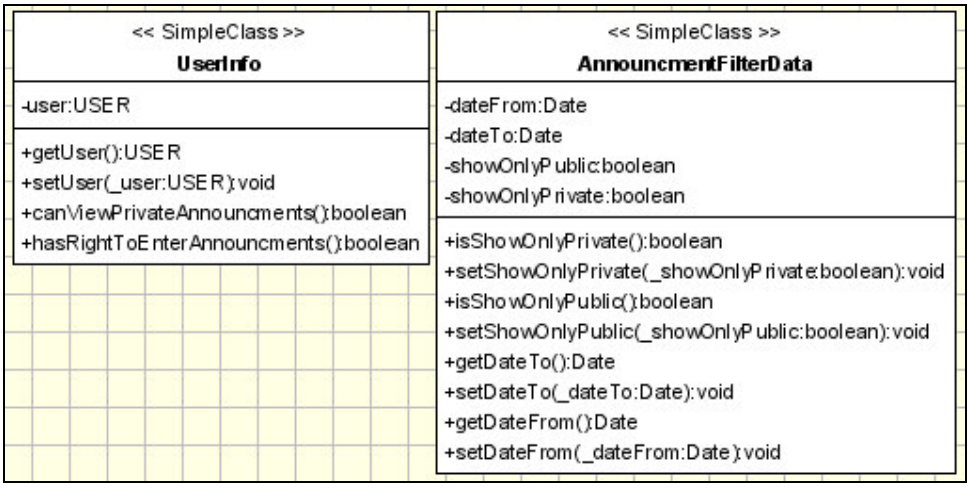
Kiti portalo veikloje dalyvaujantys ir veiksmus su duomenimis atliekami objektai: „*MainSessionData*“ –objektas, atliekantis su veiksmais saugomais vartotojo seanso metu; „*MessageConstants*“ – pranešimų konstantas turintis objektas.



Pav. 28 Kitų veiksmuose dalyvaujančių duomenų šaltiniai

3.2.6 Kitų klasių, dalyvaujančių portalo veikloje, diagrama

Klasė „*UserInfo*“ – turi veiksmus vartotojo teisių patikrinimui. „*AnnouncmentFilterData*“ aprašo duomenis, naudojamus skelbimų filtravimui.

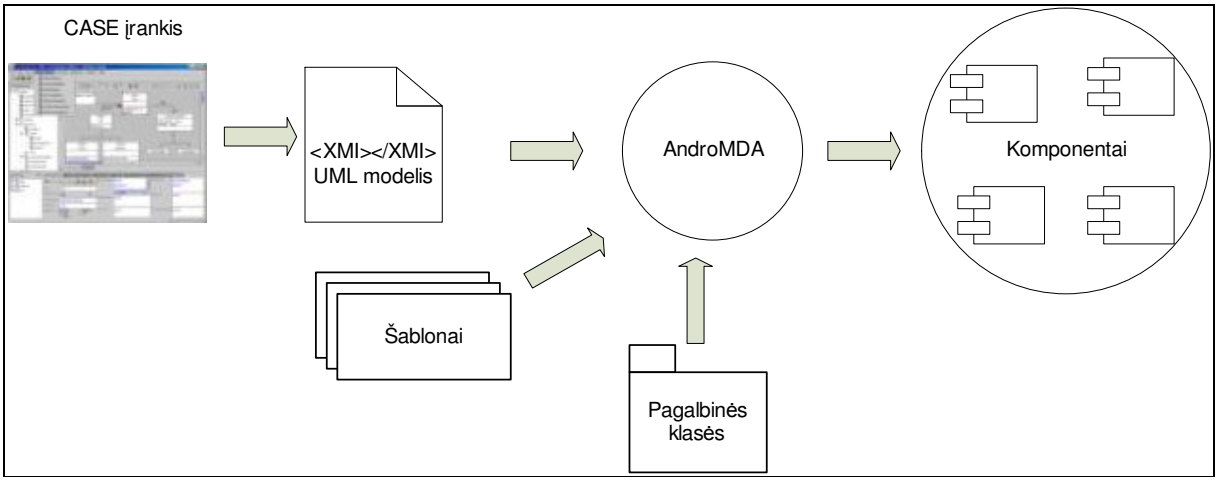


Pav. 29 Portalo veiklai realizuoti reikalingų klasių aprašas

3.3 Įrankiai, naudoti UML diagramų transformavimui į realizacijos komponentus

Kad patikrinti, jog metodikoje aprašytos elgsenos modelio sudarymo taisyklės tinka kodo generavimui, buvo sukurta programa, kuri pagal UML modelio duomenis, pateiktus XMI formatu, sugeneruoja pagrindinius veikloje dalyvaujančius komponentus. UML modelio sudarymui buvo panaudotas Poseidon CE įrankis. Ši programa buvo pasirinkta todėl, kad ji savo modelio duomenų saugojimui naudoja XMI 1.1 standartą, o ne eksportuoja kaip dauguma kitų.

Programos realizacijai buvo panaudotas AndroMDA paketas. AndorMDA – tai atviro kodo sistema padedanti realizuoti MVA principus. AndroMDA veikimo principas parodytas Pav. 30.



Pav. 30 AndroMDA veikimo principas

Portalo realizacijai buvo pasirinkta J2EE platforma ir buvo sudaryti tokie šablonai:

- Stereotipu „*Request*“ pažymėtoms modelio klasėms sudaryti šablonai, pagal kuriuos generuojamos užklausų tipo komponentus realizuojančios Java klasės. Šiose klasėse pilnai realizuojamas užklausų apdorojimas aprašytas diagramomis;
- Stereotipu „*RequestControler*“ pažymėtoms klasėms sudarytas šablonas pagal kurį, generuojama valdiklio funkcijas realizuojantis Java servlet komponentas. Šis valdiklis atpažįsta visas užklausas aprašytas modelyje ir iškviečia jų apdorojimą;
- Stereotipu „*HtmlComponent*“ pažymėtoms klasėms sudaryti šablonai, pagal kuriuos generuojamos Java klasės, turinčios visus aprašytus atributus ir metodus, taip pat ir vartotojo sąsajos komponento užkrovimo metodą. Puslapių generavimui sukuriami JSP failai;
- Stereotipu „*SessionDataManager*“ pažymėtoms modelio klasėms realizaciją generuojantys šablonai;

- Stereotipu „*Database“* pažymėtu modelio paketams buvo sukurti šablonai, pagal kuriuos sukuriamas *Torque* duomenų bazės schemos aprašas. *Torque* - atviro kodo sistema, realizuojanti sąsają tarp Java objektų ir releacinių duomenų bazių;

- Stereotipu „*DBDataManager“* pažymėtoms modelio klasėms buvo sukurtas šablonas, pagal kurį generuojamos Java klasės, realizuojančios sąsają su duomenų baze, naudojantis *Torque* priemonėmis.

3.4 Pagal UML diagramas sugeneruoti komponentai

3.4.1 „*Request*‘ stereotipu pažymėtų klasių realizacija

Kiekvienai portalo modelio klasei su stereotipu „*Request*“ sugeneruota po dvi Java klases:

1. Abstrakčios klasės, kurių pavadinimas gautas prie atitinkamos užklausos pavadinimo pridėjus žodį -„Abstract“. Šiose klasės kuriose pilnai realizuoti tokie metodai:

    a. *execute* – metodas, kuriame realizuoti visi veiksmų sekų kvietimai, kurie aprašyti UML diagramose. Kiekvienai veiksmų sekai realizuoti buvo sugeneruota po procedūrą;

    b. Kiekvienoje užklausos apdorojime nurodytai veiksmų sekai sugeneruoti tuos veiksmus atliekantys metodai su pavadinimais tokiais kaip ir atitinkamų veiksmų sekų pavadinimai. Šie metodai kviečiami iš *execute* metodo;

    c. Atributų nustatymo / grąžinimo metodai.

2. Klasės, kurių pavadinimai tokie pat kaip ir užklausos klasės modelyje, ir kurios paveldi abstrakčių klasių funkcijas. Šiose klasėse sugeneruoti tušti metodai, kurių realizacija negalėjo būti sugeneruota.

3.4.2 *„RequestControler“* stereotipu pažymėtos klasės realizacija

Šiuo stereotipu pažymėtai klasei buvo sugeneruotas pilnai funkcionuojantis Java servlet komponentas, atliekantis užklausų valdymą. T.y. užklausos identifikavimą ir jos vykdymo iškvietimą.

3.4.3 „*HtmlComponent*“ stereotipu pažymėtų klasių realizacija

Kiekvienai šiuo stereotipu pažymėtai portalo modelio klasei buvo sugeneruota:

1. JSP failas – puslapio generavimo komponentas, kuris iškviečiamas *load* metodo kvietimo metu, ir kuris turi nuorodą į ją iškvietusią klasę;

2. Abstrakčios klasės, kurių pavadinimas gautas prie pavadinimo pridėjus žodį - „Abstract“. Šiose klasės kuriose pilnai realizuoti tokie metodai:

    a. *load* – metodas, realizuojantis JSP puslapio generavimo komponento iškvietimą;

    b. Atributų nustatymo / grąžinimo metodai.

3. Klasės, kurios paveldi sugeneruotų abstrakčių klasių funkcijas. Šiose klasėse sugeneruoti tušti metodai, kurių realizacija negalėjo būti sugeneruota. Jie skirti užpildyti bet kokiems duomenims, kurių gali prireikti generuojant atitinkamą puslapį.

3.4.4 „*SessionDataManager*“ stereotipu pažymėtos klasės realizacija

Sugeneruota pilnai funkcionuojanti klasė realizuojanti visas aprašytas funkcijas ir sauganti kiekvieno vartotojo duomenis.

3.4.5 *„Database*“ stereotipu pažymėto paketo transformacija

Pagal „*Database*“ stereotipu pažymėtame pakete esančių klasių aprašą buvo sugeneruotas *Torque* duomenų bazės schemos aprašas. Iš kurio vėliau *Torque* priemonėmis sugeneruoti duomenų bazės kūrimo scenarijai, Java klasės atitinkančios „*DataTable*“ pažymėtų klasių struktūrą, tačiau turinčios metodus duomenims išsaugoti ir atnaujinti, bei klasės turinčios metodus šių tipo duomenų paėmimui iš duomenų bazės.

3.4.6 „*DBDataManager“* stereotipu pažymėtos klasės realizacija

Pagal šią modelio klasė buvo sugeneruotos dvi Java klasės:

1. Abstrakti klasė, kurios pavadinimas gautas prie atitinkamos modelio klasės pavadinimo pridėjus žodį -„Abstract“. Šioje klasės pilnai realizuotas metodas, pažymėtas *„SqlInsert“* stereotipu , kurio pavadinimas „*insertAnnouncment“*;

2. Klasės, kuri paveldi sugeneruotos abstrakčios klasės funkcijas. Šiose klasėse sugeneruoti tušti metodai, kurių realizacija negalėjo būti sugeneruota;

3.5 Išvados

Eksperimento metu buvo siekiama patikrinti ar pagal sudarytą portalo veiklos UML modelio aprašymą XMI, galima sugeneruoti komponentus, realizuojančius tą veiklą. Buvo aprašytas skelbimų portalo veikimo modelis, sudarytas to modelio transformavimas į realizaciją naudojant J2EE platformą. Eksperimento metu buvo nustatyta, kad specifikuotas portalo veikimas visiškai pilnai transformuojamas į realizaciją. Buvo parodyta, kad dauguma komponentų bei jų funkcijų gali būti arba dalinai arba pilnai sugeneruota vien iš UML aprašo. Buvo sugeneruotos tokios pilnai veikiančios funkcijos ir komponentai:

- klasių atributų reikšmių keitimo funkcijos;
- duomenų išsaugojimo duomenų bazėje, bei jų atnaujinimo bei trynimo funkcijos ;
- vartotojo sąsajos komponentų užkrovimo funkcijos;
- užklausų apdorojimą realizuojančios funkcijos;
- užklausų atpažinimo funkcijos;
- užklausos iškvietimo URL generuojančios funkcijos;
- vartotojo seanso duomenų saugojimo – paėmimo komponentai.

IŠVADOS

1. Darbe buvo atlikta žiniatinklio portalų kūrimo ypatybių analizė. Buvo nustatyta, kad žiniatinklio portalų kūrimas skiriasi nuo tradicinės programinės įrangos kūrimo. Šie skirtumai skirstomi į techninius (padidėjęs vartotojo sąsajos vaidmuo, išaugusi turinio svarba, glaudesnis ryšys tarp verslo modelio ir architektūros) ir organizacinius (klientų neapsisprendimas, verslo reikalavimų pasikeitimai, trumpesnis kūrimo laikas).

2. Todėl buvo atlikta egzistuojančių metodikų analizė. Analizės metu nustatyta, kad dauguma žiniatinklio portalo kūrimo metodikų gerai aprašo struktūrinius portalo modelius, tačiau mažai dėmesio skiriama portalo veiklos modelio sudarymui, kuris yra labai svarbus žiniatinklio portalų kūrime. Be to dažniausiai metodikos aprašo, kaip sudaryti portalo architektūros modelius, tačiau visiškai neapibrėžia, kaip susieti tuos modelius su realizacija, kas reikalinga, norint užtikrinti didesnį portalų kūrimo našumą.

3. Buvo nuspręsta, kad norint, užtikrinti gerą portalų palaikomumą ir greitą jų kūrimą, reikalinga įgyvendinti MVA principus.

4. UML modeliavimo kalba su savo išplėtimo mechanizmais, ją palaikančiais įrankiais, bei standartizuotu modelių apsikeitimu labai tinka realizuoti MVA požiūrį įgyvendinančiam žiniatinklio portalų kūrimui**.** Todėl buvo pasiūlytas portalo veiklos modelio sudarymo taisyklės, leidžiančios ne tik specifikuoti pagrindinius portalo komponentus bei jų elgseną portalo funkcijų atlikimo metu, bet ir generuoti tą elgseną realizuojančius komponentus.

5. Tai pat buvo aprašyti tipiniai portalų kūrime dalyvaujantys komponentai, bei jų naudojimo veiklos modelyje principai, leidžiantys generuoti ir dalį tų komponentų realizacijos pagal UML modelį.

6. Eksperimentinėje dalyje buvo sudarytas portalo veiklos modelis pagal aprašytą metodiką ir atliktas modelio realizacijos generavimas į J2EE platformos komponentus. Buvo parodyta, kad daugelio komponentų realizacijos generavimui užtenka modelyje esančios informacijos. Tie komponentai tai užklausos, užklausų valdiklis, užklausų apdorojimo sekos, bei komponentai atliekantys pagrindinius veiksmus sus duomenų baze.

7. Tas, kad portalo veikla aprašyta modelyje ir pagal ją generuojama realizacija ne tik padidina sistemos aiškumą, užtikrina kūrimo našumą, bet ir leidžia lengvai pakeisti tam tikrų užklausų apdorojimo seką neprogramuojant, o tiesiog keičiant modelį.

LITERATŪRA

[1] Schwabe D.; Rossi G. An Object Oriented Approach to Web-Based Application Design [interaktyvus]. 1998 [žiūrėta 2004-03-01]. Prieiga per internetą: <http://www.telemidia.puc-rio.br/oohdm/oohdm.html>

[2] Lowe D.; Henderson-Sellers B. Characteristics of Web Development Processes [interaktyvus]. International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet, 2001 [žiūrėta 2003-05-24]. Prieiga per internetą: <http://www.ssgrr.it/en/ssgrr2001/papers>

[3] Koch N. A Comparative Study Of Methods For Hypermedia Development [interaktyvus]. Technical Report 9905, Ludwig-Maximilians-Universität München, 1999 [žiūrėta 2003-04-23]. Prieiga per internetą:

<http://www.pst.informatik.uni-muenchen.de/personen/kochn/techrep/hypdev.pdf >

[4] Kuhnke C.; Schneeberger J.; Turk A. A Schema-Based Approach to Web Engineering [interaktyvus]. 2000 [žiūrėta 2003-05-03]. Prieiga per internetą: <http://www.eds.schema.de/doku/html-deu/schemapu/resources/pdf/publik/HICSS-ST1.pdf>

[5] Garzotto F.; Paolini P.; Schwabe D. HDM - A Model-Based Approach to Hypertext Application Design [interaktyvus]. ACM Transactions on Information Systems, Vol. 11, No. 1. 1995 [žiūrėta 2003-04-23] . Prieiga per internetą: <http://www-is.informatik.uni-oldenburg.de/~dibo/teaching/mm/pages/HDM.html>

[6] Isakowitz T.; Kamis A.; Koufaris M. The Extended RMM Methodology for Web Publishing [interaktyvus]. Center for Research on Information Systems, 1998 [žiūrėta 2003-04-23]. Priega per internetą: <http://jmis.bentley.edu/rmm/papers/RMM-Extended.pdf>

[7] De Troyer O.; Leune C. WSDM: A user-centered design method for Web sites [interaktyvus]. 1997 [žiūrėta 2003-05-05]. Prieiga per internetą: <http://www7.scu.edu.au/programme/fullpapers/1853/com1853.htm>

[8] Bieber M.; Galnares R.; Lu Q. Web engineering and flexible hypermedia [interaktyvus], 1998 [žiūrėta 2003-05-15]. Prieiga per internetą: <http://wwwis.win.tue.nl/ah98/Bieber.html>

[9] Thomson J.; Greer J.; Cooke J. Algorithmically detectable design patterns for hypermedia collections [interaktyvus]. 1998 [žiūrėta 2003-05-06]. Prieiga per internetą: <http://wwwis.win.tue.nl/ah98/Hardman.html>

[10] Lowe D.; Hall W. Hypermedia & the Web: An engineering approach. John Wiley & Sons, 1999. 626 p.

[11] OMG (Object Management Group). MDA Guide Version 1.0.1 [interaktyvus]. 2003 birželis. Prieiga per internetą: <http://www.omg.org/mda/>

[12] OMG (Object Management Group). OMG Unified Modeling Language Specification, Version 1.5 [interaktyvus]. 2003 kovas [žiūrėta 2004-04-05]. Prieiga per internetą: <http://www.omg.org/mda/>

[13] OMG (Object Management Group). OMG XML Metadata Interchange (XMI) Specification, v1.2 [interaktyvus]. 2002 vasaris [žiūrėta 2004-04-05]. Prieiga per internetą: <http://www.omg.org/mda/>

[14] Isakowitz T., Edward A. Stohr; Balasubramanian P. RMM: A Methodology for Structured Hypermedia Design [interaktyvus]. 1995 [žiūrėta 2003-04-23]. Prieiga per internetą: <http://jmis.bentley.edu/rmm/papers/rmd.pdf>

[15] Conallen J. Building Web Applications with UML Second Editon. Addison Wesley, 2002, 496 p.

[16] Henderson-Sellers B. Who needs an object-oriented methodology anyway? Journal of Object Oriented Programming, 1995.

[17] Sinan Si A. Learning UML, O'Reilly, 2003. 252 p.

[18] Avison D.; Fitzgerald G. Information systems development: methodologies, techniques and tools, Higher Education, 2002. 416 p.

[19] Olsina L. Building a Web-based information system applying the hypermedia flexible process modeling strategy [interaktyvus]. 1998 [žiūrėta 2003-04-05]. Prieiga per internetą: <http://gidis.ing.unlpam.edu.ar/personas/olsinal/AISWork_HT98.html>

[20] Baumeister H.; Koch N.; Mandel L. Towards a UML extension for hypermedia design [interaktyvus]. 1999 [žiūrėta 2003-05-05]. Prieiga per internetą: <http://www.pst.informatik.uni-muenchen.de/projekte/forsoft/pubs/uml99.pdf>

TERMINŲ IR SANTRUMPŲ ŽODYNAS

Santrumpa	Paaiškinimas
CGI	Common Gateway Interface
cgi-Lua	Scenarijų kalba
EJB	Enterprise Java Beens
E-R	Esybių ryšių notacija
HPM	Hiperterpės projektavimo metodas
HTML	Hyper Text Markup Language
IORM	Išplėsta objekto ryšių metodologija
JSP	Java Server Pages
LHPM	Lankstus hiperterpės proceso modeliavimas
Modeliavimas	Programinės įrangos projektavimo proceso dalis, kurioje kuriamas sistemos architektūros (loginės, fizinės, komunikavimo) modelis
Modelis	Elementų ir jų sąryšių visuma, apibūdinanti programinę įrangą tam tikru aspektu
MVA	Modeliu valdoma architektūra
Notacija	Standartizuotas žymėjimas
NRA	Naršymo ryšių analizė
OHPM	Objektinis hiperterpės projektavimo metodas
OMG	Object Management Group
OMT	Object Modeling Technique
OOHDM-Web	Įrankis realizuojantis OHPM metodologijos principus
Paketas	Programinės įrangos elementų (programų, dokumentų, bibliotekų ir t.t.) visuma, sudaranti ekslotacijai paruoštą programinės įrangos produktą
Realizacija	Programos užrašymas tam tikra programavimo kalba ir jos suderinimas bei vykdomųjų elementų (failų) sukūrimas

RVM	Ryšių valdymo metodologija
Scenarijus	Vykdomasis specialių komandų rinkinys (vykdomas tiesiogiai secializuotų interpretatorių) tekstiniame formate.
Specifikacija	Detalus aprašymas.
SPOHPM	Scenarijais pagrįsta objektinė hiperterpės projektavimo metodologija
SQL	Structured Query Language
Stereotipas	Modelio elementų pažymėjimas, norint nurodyti elemento semantiką
Šablonas	Elemento griaučiai, formavimo taisyklės ir pan.
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
ŽSPM	Žiniatinklio svetainių projektavimo metodas