KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Milda Ridikaitė

Metaprojektavimo aspektų realizavimas pakartotinio naudojimo technologija grindžiamose informacinėse sistemose

(Implementation of meta-design aspects for information systems based on reuse technology)

Magistro darbas

Darbo vadovas

prof. habil. dr. A. Targamadzė

Kaunas, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Milda Ridikaitė

Metaprojektavimo aspektų realizavimas pakartotinio naudojimo technologija grindžiamose informacinėse sistemose

(Implementation of meta-design aspects for information systems based on reuse technology)

Magistro darbas

Recenzentas

prof. D. Rubliauskas

200707-05-29

Vadovas

prof. habil. dr. A. Targamadzė

200707-05-29

Atliko

IFM-1/2 gr. stud.

Milda Ridikaitė

2007-05-29

Kaunas, 2007

Turinys

Implementation of meta-design aspects for information systems based on reuse technology

Summary

The aim of this paper is to present symbiosis of reuse technology, component based design and meta-project aspects. Symbiosis occurs through software architectural model and set of reusable components designed to involve end-user to software development process as designer. Theoretical background of specified model is delivered in the beginning sections of this thesis. As experimental solution information system was introduced. System provides a pack of highly configurable and customizable components designed to achieve common tasks. Unexpected or rare tasks are solved by providing extra components, which are developed and integrated to entire system according predefined rules. Several complete definitions and application areas of those components are given. Paper also includes statistics about reusable component life cycle in different software projects: how often they are reused and for what purpose. Thesis is concluded with pros and cons of suggested model.

1 ĮVADAS

Jau daugiau nei dešimtmetį programinės įrangos kūrėjams suvokiama, kad programų sistemų kūrimas pagal klasikinį Krioklio modelį (*Waterfall model*) [46] yra teoriškai neįgyvendinamas. Visada atsiras vartotojas, kuris tik sistemos naudojimo metu pasiges jo įsivaizduojamos ir laukiamos funkcijos. Visada bus užsakovas, kuris po kelių metų, mėnesių ar dienų dėl įvairių priežasčių (pvz., pasikeitusios verslo veiklos, pasikeitusio veiklos taisyklių organizacijoje, dėl atsiradusių pokyčių visuomenėje, politikoje, ekonomikoje, aplinkoje ir pan.) turimą ar kuriamą programinę įrangą nuspręs tobulinti, atnaujinti sistemos funkcionalumą, rekonstruoti, reorganizuoti vidinę sistemos struktūrą, perskirstyti roles, atsakomybes sistemoje, modifikuoti formas, ataskaitas, duomenų saugojimo ir pateikimo struktūras ir pan. Visada bus už reikalavimų atsakingas analitikas, kuris kažkokią smulkmeną ar ne smulkmeną pamirš. Visada bus projektuotojas, kurio sprendimu nepasitenkins programuotojas.

Šiuolaikinių programinės įrangos pardavimų vadybininkų, projektuotojų, analitikų, programuotojų tikslas yra kiek galima daugiau ir anksčiau pritraukti galutinį sistemos vartotoją (*end-user*) [53] prie sistemos inicijavimo, reikalavimų išgavimo, projektavimo, programavimo, testavimo, pridavimo. Dėl šių siekių išsivystė tokios kraštutinės programinės įrangos kūrimo proceso valdymo metodikos kaip lankstusis programavimas (*Agile Programming*) [47]. Didelio dėmesio ypatingai susilaukė lanksčiojo programavimo metodikos šaka – ribinis programavimas (*eXtreme Programming, XP*) [48]. Ypatingai domimasi ir vystoma metamodeliavimo [49], metaduomenų modeliavimo ir apdorojimo [50], metaanalizių [51] procesų taikymų sritis, metakalbų [52] panaudojimas, metaprogramines įrangos kūrimas ir pan.

Šiame darbe pateikiamas pakartotinio panaudojimo komponentais grįstos architektūros sprendimas – informacinė sistema, kurios modeliavimas, projektavimas ir naudojimas yra tiesiogiai nukreiptas į **galutinį vartotoją kaip į sistemos dizainerį**. Eksperimentinis sprendimas (žr. skyrių 3 Eksperimentinė pakartotinio naudojimo komponentais grįsta informacinė sistema) remiamas pakartotinio panaudojimo technologija (žr. skyrių 2.2 Pakartotinis panaudojimas (Reuse)), komponentais grįsta architektūra (žr. skyrių 2.4 Komponentais grįsta architektūra), metaprojektavimo principais (žr. skyrių 2.3 Metaprojektavimas (meta-design)).

1.1 Tikslas

Magistrinio darbo tikslas yra išanalizuoti egzistuojančią pakartotinio panaudojimo technologiją ir komponentais grįstos architektūros metodologijas, susipažinti su metaprojektavimo koncepcija ir suprojektuoti informacinės sistemos architektūros koncepcinį modelį, kuris užtikrintų galutinio vartotojo įtraukimą į programinės įrangos kūrimo procesą projektuotojo teisėmis: programų sistemos realizacija galutiniam vartotojui padėtų tiesiogiai įtakoti sistemos planuojamą funkcionalumą, duomenis, išvaizdą, navigaciją, struktūrą.

1.2 Objektas

Magistrinio darbo tyrinėjimo objektas yra informacinės sistemos architektūros koncepcinis modelis, kuris įgalintų galutinį vartotoją susirinkti sistemą pagal poreikius iš programinės įrangos kūrėjų siūlomų pakartotinio panaudojimo komponentų aibės. Realizuotas modelis remiamas metaprojektavimo koncepcija, pakartotinio panaudojimo technologija, komponentais grįstos architektūros pritaikymu.

1.3 Uždaviniai

Gali būti išskirti tokie magistrinio darbo uždaviniai:

1. Išanalizuoti egzistuojančias metodologijas ir technologijas, kurios padėtų sukurti į galutinį vartotoją kaip į dizainerį orientuotą programų sistemą.
2. Pateikti koncepcinį architektūros modelį, jo veikimo principus, struktūrą.
3. Sukurti eksperimentinį sprendimą suprojektuotam modeliui realizuoti.
4. Pateikti kelių pakartotino panaudojimo komponentų specifikacijas, pabrėžiant šių komponentų vaidmenį galutinio vartotojo laisvės laipsniams projektavimo etape užtikrinti, konfigūravimo galimybes, pritaikomumo sritis.
5. Išskirti pasiūlyto modulio privalumus ir trūkumus.

2 TEORINIS SPRENDIMO PAGRINDAS

Šiame skyriuje pateikiamas magistrinio darbo teorinis pagrindas: pradedant nuo pakartotinio panaudojimo (*reuse*) koncepcijos, komponento apibrėžimo, komponentinio kūrimo (*component develepment*) ir komponentinės architektūros (*component-based architekture*) specifikacijos, baigiant metaprojektavimo (*meta-design*) paradigmų analize.

2.1 Terminai ir esminės sąvokos

- **Pakartotinis panaudojimas, atkartojimas, kodo atkartojimas (*reuse, software reuse, code reuse*)** – egzistuojančios programinės įrangos ar jos dalių naudojimas kuriant naują programinę įrangą [2] (šio termino kiti egzistuojantys apibrėžimai pateikiami skyriuje 2.2).

- **Metaprojektavimas *(meta-design)*** charakterizuoja veiklas, procesus ir tikslus sukurti naują aplinką, kuri leistų vartotojams elgtis kaip projektuotojais ir būti kūrybingais [38] (detaliau skyriuje 2.3 Metaprojektavimas (meta-design)).

- **Galutinis vartotojas *(end-user)*** – ekonomika ir komercija galutinį vartotoją apibūdina kaip asmenį, kuris naudoja produktą [53].

- **Pakartotinio panaudojimo komponentai *(reusable components)*** – tai vieną kartą sukurti programinės įrangos komponentai su galimybe juos panaudoti keliuose skirtinguose programinės įrangos projektuose, ar to paties projekto skirtingose funkcijose ar netgi tose pačiose funkcijose keletą kartų.

  Pakartotinio panaudojimo komponentai turi:

  - Aiškią veiklos logiką;
  - Plačias konfigūracijos galimybes;
  - Standartizuotą atvaizdavimo sluoksnį.

- **Karkasas (*framework*)** – šiame dokumente karkasas yra angliško termino „framework“ vertimas, kuriuo vadiname aprašytą pagalbinę struktūrą, kurios pagalba gali būti organizuojama ar kuriama kita programinė įranga.

- **JAVA** – objektiškai orientuota programavimo kalba, 1991 metais sukurta Džeimso Goslingo ir kitų „Sun Microsystems“ inžinierių. Kalba oficialiai paskelbta 1995 metų gegužės 23 d., o išleista tų pačių metų lapkritį. Java (pradžioje vadinta Oak) kalbos pirminis tikslas buvo pakeisti C++ kalbą [6].

- **„Tapestry“** – atviro kodo karkasas, skirtas kurti dinamines, patikimas, smarkiai plėtojamas interneto sistemas JAVA kalba.

2.2 Pakartotinis panaudojimas (Reuse)

Be apibrėžimo pateikto terminų ir esminių sąvokų skyrelyje, literatūroje sutinkama daug skirtingų pakartotinio panaudojimo apibrėžimų:

- Programinės įrangos pakartotinis panaudojimas yra egzistuojančios programinės įrangos patirties, žinių ar jos artefaktų panaudojimas naujoms sistemoms kurti [1].

- Programinės įrangos pakartotinis panaudojimas (*Software reuse*) yra programinės įrangos sistemų diegimo arba atnaujinimo procesas naudojant egzistuojančius programinės įrangos elementus (*software assets*). Programinės įrangos pakartotinis panaudojimas galimas programinės įrangos sistemos kontekste, taip pat panašių programinės įrangos sistemų kontekste ir netgi tarp gana skirtingų sistemų [18].

1968-aisiais NATO programinės įrangos projektavimo konferencijoje (NATO Software Engineering Conference) pirmą kartą buvo pasiūlyta sisteminė oficiali pakartotinio panaudojimo sąvoka. McIlroy, kuris pirmasis pasiūlė terminą „programinės įrangos pakartotinis panaudojimas“, norėjo iškeisti programinės įrangos kūrėjų komandą į programinės įrangos projektavimo industriją ir taip įveikti programinės įrangos krizę. Pranešėjas pasiūlė išeities kodo komponentų „nuo lentynos“ gamyklą ir įsivaizdavo sudėtingas sistemas sudarytas iš mažų sudedamųjų blokų, kuriuos galima įsigyti iš pakartotinio panaudojimo komponentų katalogų [17]. Ši svajonė dar toli gražu neišsipildė.

Vėliau, programinės įrangos pakartotinio panaudojimo koncepcija išsiplėtė. Buvo įvestas terminas „pakartotinio panaudojimo elementas“ (*reusable asset*). Pakartotinio panaudojimo elementas yra bet koks atkartojimo bibliotekos vienetas. Bibliotekoje gali būti projektavimo dokumentacija, specifikacijos elementai, išeities kodai, testų rinkiniai [17], programinės įrangos komponentai, objektai, programinės įrangos reikalavimų analizės/projektų modeliai, srities

architektūros dokumentai, modeliai, duomenų bazės schemos, vartotojo vadovai, testavimo planai, standartai [18], etc.

2.2.1 Pakartotinio panaudojimo nauda ir privalumai

Skirtinguose literatūros šaltiniuose pabrėžiami skirtingi pakartotinio panaudojimo metodikos privalumai. Viename šaltinyje [17] teigiama, kad atkartojimo nauda yra grįsta grynai ekonominiais aspektais ir gali būti skirstoma į dvi kategorijas:

- **Programinės įrangos kūrimo produktyvumo padidėjimas.** Produktyvumas padidėja [19], nes nebėra būtinybės programinę įrangą kurti nuo pradžios. Taip pat dažnai sumažėja kūrimo išlaidos dėl mažesnio poreikio kurti naują programinės įrangos išeities kodą.

- **Programinės įrangos kokybės padidėjimas.** Programinės įrangos kokybė išauga dėl pakartotinio panaudojimo komponentų, kurie jau paruošti naudojimui, ištestuoti, numatytos galimos klaidos, kurių pasitaiko naujų komponentų kūrimo metu. Taip pat sumažinamos palaikymo išlaidos.

Kitame literatūros šaltinyje [18] be aukščiau paminėtų privalumų, išvardinami dar ir šie:

- Mažesnės išlaidos;
- Sutrumpėję programinės įrangos kūrimo tvarkaraščiai;
- Mažina palaikomumo pastangas ir įkainius;
- Skatina standartizavimą;
- Didina perkelialumo galimybes;
- Įneša indėlį į plačiai paplitusių komponentų sandėlį;
- Didina atlikimo, vykdymo rodiklius.

2.2.2 Reikalavimai pakartotinio panaudojimo programinei įrangai

Išskiriami tokie reikalavimai [20] programinei įrangai, kuri gali būti vėliau sėkmingai panaudota:

- Programinė įranga turi būti naudojama, t.y., turi spręsti paplitusias problemas ir tenkinti dažnai pasitaikančius poreikius;

- Programinė įranga turi būti panaudoja iš naujo, t.y., geros kokybės, lengvai suprantama ir pritaikoma kitų programinės įrangos kūrėjų.

1995 metais buvo pasiūlytas pakartotinio panaudojimo modelis, kur pabrėžiamos tokios programinės įrangos savybės:

- **Pritaikomumas**. Pritaikomumas dažnai siejamas su lankstumu, t.y., kaip lengvai gali būti pritaikytas komponentas išpildyti pageidaujamam reikalavimui, kuris gali kažkiek skirtis nuo reikalavimo, kuriam komponentas buvo sukurtas. Pritaikomumas gali būti apibrėžiamas kaip moduliškumas, t.y., kaip lengvai egzistuojantis sprendimas gali būti padalintas į nepriklausomas dalines funkcijas, ir bendrumas, t.y., komponento nepriklausomumas nuo likusios programinės įrangos.

- **Suprantamumas**. Programinės įrangos atributai, kurie teikia vartotojams informaciją apie loginę elemento koncepciją ir jo pritaikomumą.

- **Perkeliamumas.** Programinės įrangos perkėlimo iš vienos kompiuterių sistemos į kitą arba iš vienos aplinkos į kitą lengvumas.

- **Pasitikėjimas.** Tikimybė, kad modulis, programa ar sistema naujoje aplinkoje bus sėkmingai pritaikyta, veiks be klaidų ir atitiks anksčiau apibrėžtą tikslą, kuriam buvo sukurta ir ištestuota.

2.2.3 Pakartotinio panaudojimo programinės įrangos kūrimą ribojantys faktoriai

Literatūroje išskiriami du pakartotinio panaudojimo programinės įrangos kūrimą ir panaudojimą ribojantys faktorių tipai: techniniai ir netechniniai. Autoriai nenusprendžia, kuri probleminė sritis yra svarbesnė ir labiau įtakoja procesą. Kai kurie specialistai teigia, kad netechniniai aspektai daro žymesnį poveikį [23, 24], kiti pripažįsta, kad egzistuoja netechniniai faktoriai, bet tiki, kad ateityje jų bus išvengta [21].

Netechniniai faktoriai, ribojantys programinės įrangos kūrimą pakartotiniam panaudojimui [21, 22]:

- **Ekonominės priežastys.** Pakartotiniam panaudojimui pasiekti reikalingos papildomos pastangos, todėl turi būti sukurti ir pritaikyti specifiniai ekonominiai modeliai, kurie padėtų matuoti ir paskirstyti investicijas.

- **Organizacinės priežastys.** Pakartotinio panaudojimo elementams išplatinti, surasti ir parduoti reikalingi specifiniai strateginiai sprendimai. Net ir geriausias pakartotinio panaudojimo komponentas bus nenaudingas, jei niekas apie jį nebus informuotas, jei jį surasti užtrunka laiko ar jis per brangus įsigyti.

- **Edukacinės priežastys.** Apmokyti personalą reikia ne tik apie tai, kaip kurti pakartotinio panaudojimo programinę įrangą, bet taip pat kaip konstruoti programinę įrangą panaudojant egzistuojančius elementus.

- **Psichologinės priežastys.** Asmuo, kuris nusprendė naudoti pakartotinio panaudojimo elementą, turi juo pasikliauti, t.y., tikėti, kad jis veiks taip, kaip yra pateikta to elemento aprašyme. Sindromas „nėra sukurta/išrasta“ yra labai populiarus psichologinis barjeras bandant pritaikyti pakartotinio panaudojimo produktus.

- **Vadybinės priežastys.** Vadybininkų pajėgos turi išskirti reikalingas roles pakartotinio panaudojimo programinės įrangos kūrimo projektuose ir užpildyti šias roles teisingais asmenimis.

Techniniai faktoriai, ribojantys programinės įrangos kūrimą pakartotiniam panaudojimui:

- Programinės įrangos kūrimo neišsivystymas kaip inžinerinės disciplinos:
    - nėra nusistovėjusių standartų, kurie padėtų pateikti/pristatyti pakartotinio panaudojimo komponentus;
    - neegzistuoja pakankamai architektūros standartų.
- Komponentų bibliotekų, kuriose pagal poreikius būtų galima rasti tinkamus elementus, trūkumas;
- Metodologijų, skirtų pakartotinio panaudojimo pritaikymui, trūkumas
- Tinkamų priemonių, palaikančių pakartotinio panaudojimo elementų apjungimą programinės įrangos kūrimo metu, trūkumas.

2.3 Metaprojektavimas (meta-design)

Metaprojektavimas [34] apibūdina tikslus, metodus ir procesus, skirtus sukurti naują erdvę ir aplinkas, kurios leistų „problemos savininkams“ (dar kitaip vadinamiems galutiniams vartotojams) elgtis kaip projektuotojais, savo poreikius atitinkančios programinės įrangos dizaineriais. Esminis metaprojektavimo tikslas yra sukurti socialines-technines aplinkas, kurios įgalintų vartotojus aktyviai įsitraukti į besitęsiantį programų sistemos vystymą, o ne prisitaikyti prie ribotų egzistuojančių sistemų [25].

Metaprojektavimas yra pagrįstas prielaida, kad ateityje nebus įmanoma pilnai numatyti visų panaudos atvejų ir programinės įrangos sprendžiamų problemų tik projektavimo metu, kai sistema yra kuriama. Sistemos naudojimo metu anksčiau ar vėliau programinės įrangos vartotojai pastebės sistemos teikiamų funkcijų neatitikimus jų poreikiams. Metaprojektavimas išplečia apribotos programų sistemos ribas suteikdama galutiniams vartotojams aktyvių bendraautorių teises, kurios įgalina vartotojus viršyti egzistuojančios sistemos funkcionalumą ir turinį. Kontrolė ir atsakomybė padalinamos visiems programų sistema suinteresuotiems asmenims [26].

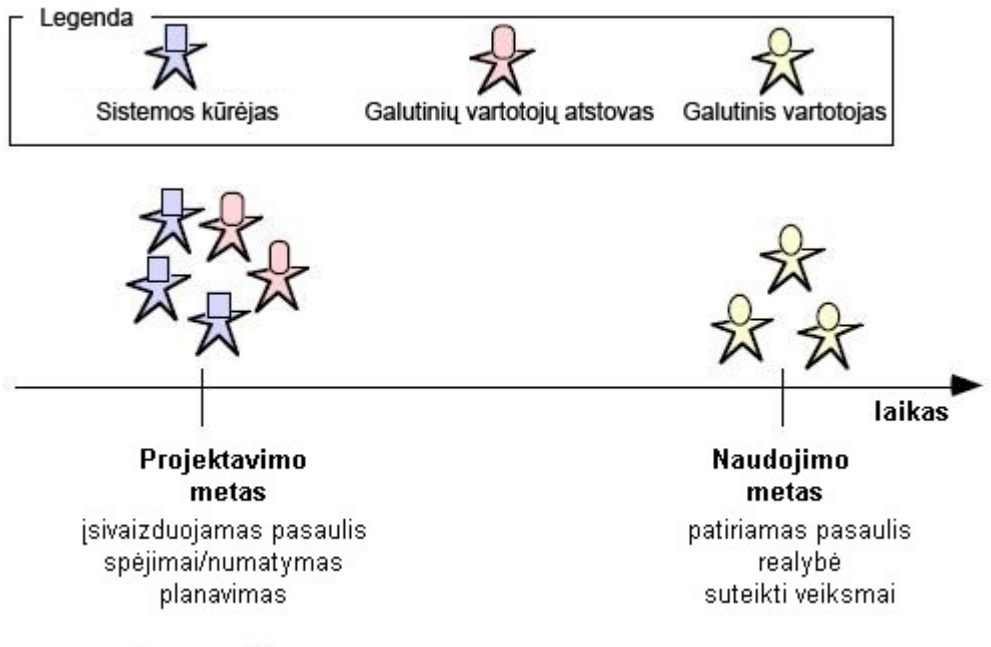
Jau dabar galima išvardinti keletą gerų galutinių vartotojų kooperacijos pavyzdžių, projektų, koncepcijų, kurie sėkmingai veikia ir intensyviai plėtojami:

- Atviro kodo programinės įrangos produktai [30];
- Bendru visų žiniatinklio vartotojų indėliu pagrįstos enciklopedijos, tokios kaip „Wikipedia“ [31];
- Skaitmeninės bibliotekos, tokios kaip DLESE [32].

Įsitraukdami į tokius projektus, žmonės ne tik naudojasi jiems suteikta programine įranga, bet taip pat įvairiais aspektais prisideda prie tos programinės įrangos funkcijų ir turinio tobulinimo, pildymo [33].

2.3.1 Projektavimo ir naudojimosi metas

Visi programinės įrangos konstravimo procesai gali būti susiskirstyti į dvi pagrindinės fazes: **projektavimo** ir **naudojimosi metą** (žr. 1 pav. Projektavimo metas ir naudojimo metas [35].). Projektavimo metu sistemos kūrėjai (su ar be galutinių vartotojų atstovų dalyvavimo) kuria aplinkas ir priemones jų įsivaizduojamam pasauliui, pagal tai, kaip jie įsivaizduoja vartotojo poreikius ir tikslus. Naudojimosi metu vartotojai naudojasi sistema, bet kadangi į jų poreikius ir tikslus buvo atsižvelgta tik projektavimo metu, dažnai užsakovai reikalauja modifikacijų, skirtų išpildyti atsiradusius naujus reikalavimus [35].

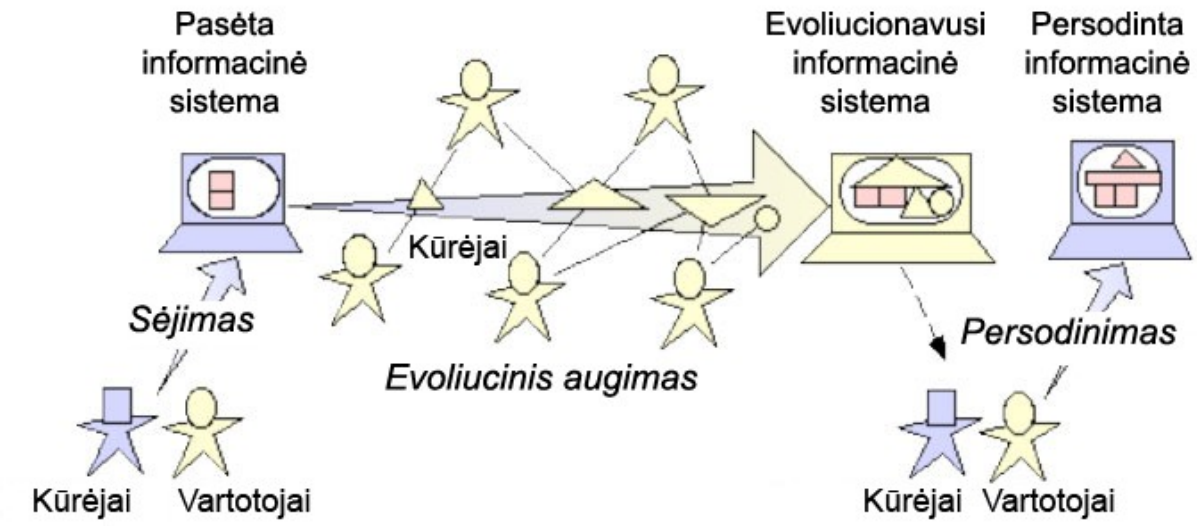


1 pav. Projektavimo metas ir naudojimo metas [35].

Norint numatyti netikėtas problemas naudojimosi metu, projektavimo metu sistemas reikia palikti nepilnai suprojektuotas tam tikroje stadijoje (*underdesigned*). Nepilnai suprojektuota (-as) (*underdesigned*) [36] šiame kontekste nereiškia, kad projektavimo komanda turi įdėti mažiau darbo ir pastangų, o kaip tik kūrimo procesas yra visiškai priešingas ir sudėtingesnis procesas nei pilnos, pagal specifikuotus reikalavimus išbaigtos sistemos sukūrimas. Pagrindinis nepilnai suprojektuotos programinės įrangos iššūkis nėra sukurti *ad hoc* sprendimą, bet išvystyti aplinkas, priemones, kurios leistų „problemų savininkams“ [37] patiems spręsti iškilusias problemas sistemos naudojimosi metu. Nepilnas projektavimas yra į metaprojektavimą orientuota veikla, kurios tikslas yra sukurti projektavimo erdves kitiems. Nepilnas projektavimas remiasi prielaida, kad sistemos prasmė, funkcionalumas ir turinys negali būti pilnai specifikuoti projektavimo metu net ir bendradarbiaujant sistemos projektuotojams ir galutinių vartotojų atstovams, tačiau visi šie sistemos elementai vystomi, konstruojami draugiškai galutinių vartotojų sistemos dizaino ir naudojimo ciklų metu.

2.3.2 Sėjimo, evoliucinio augimo ir persodinimo procesų modelis (*the seeding, evolutionary growth, and reseeding (SER) process model*)

Metaprojektavimo procesams palaikyti amerikiečiai G. Fischer, E. Giaccardi, Y. Ye, A.G. Sutcliffe ir britas N. Mehandjiev sukūrė sėjimo, evoliucinio augimo ir persodinimo procesų modelis. SEP modelis (žr. 2 pav. Sėjimo, evoliucinio augimo ir persodinimo procesų modelis [38].) yra aprašantysis ir siūlantysis modelis, skirtas didelėms besivystančioms sistemoms ir informacijos kaupykloms, teigiantis, kad sistemos, kurių vystymosi procesas užtrunka ilgai, privalo keistis tarp veiklų, neplanuojamos evoliucijos ir apgalvotos rekonstrukcijos ar tobulinimų periodų. SEP modelis skatina projektuotojus susikurti sistemų projektavimo koncepciją, grįstą metaprojektavimu, kuri vartotojams suteiktų projektuotojų roles ir neapribotų jų liekant tik pasyviais vartotojais [25].



2 pav. Sėjimo, evoliucinio augimo ir persodinimo procesų modelis [38].

Sėjimu vadinamas procesas, kurio metu sukuriama „sėkla“ – priemonė, aplinka, erdvė – turinti potencialą augti ir vystytis pagal specifinius programinės įrangos srities tikslus ir poreikius. Šios fazės metu „sėkla“ yra kuriama visų suinteresuotų asmenų bendru indėliu į projektavimo sprendimus [38].

Evoliuciniu augimu vadinami mažų apimčių programinės įrangos vystymosi pokyčiai. Šios fazės metu „pasodinta“ sistema vienu metu vaidina du svarbius vaidmenis: (1) teikia darbui resursus (informaciją, kuri buvo sukurta/sugeneruota/surinkta ankstesnio naudojimosi metu) ir (2) teikia darbui produktus, kur kiekvienas produktas suteikia galimybę pildyti sistemos resursus į „sistemą-sėklą“. Evoliucinio augimo fazės metu vartotoja susikoncentruoja labiau ties specifinės problemos sprendimu ir specifinės-problemos turinio kūrimu, nei ties pakartotinio panaudojimo informacijos vystymu. Dėl šios priežasties informacija, patalpinta šio etapo metu gali būti neintegrali su anksčiau patalpintąja „sistemoje-sėkloje“ [38].

Persodinimas yra apgalvotos pastangos evoliucinio augimo fazės metu patalpintos informacijos ir sukurtų produktų reorganizavimui, formalizavimui ir generalizavimui. Šio etapo metu gali būti atlikti drastiški sistemos pakeitimai [38].

2.3.3 Socialinis kūrybingumas

Sudėtingos projektavimo problemos reikalauja daugiau žinių nei jų gali turėti vienas žmogus. Taip pat labai dažnai reikalingą informaciją žino skirtingi užsakovo reikalavimus reprezentuojantys asmenys. Pagal metaprojektavimo principus, suinteresuotiems asmenims turi būti sukurtos priemonės, skirtos jų žinioms apie sistemą realizuoti tiesiogiai į programinę

įrangą, o ne per reikalavimų dokumentus ar architektūros specifikacijas. Jiems sukurtos priemonės nėra konkretūs konkrečių problemų sprendimai, bet socialinė-techninė aplinka, kuri užsakymo tarpininkus paverčia informuotais programinės įrangos kūrimo proceso dalyviais. Greitas ir vaizdinis grįžtamasis ryšys suteikia galimybes bendradarbiavimo metu realizuoti naujas idėjas ir produktus [25].

2.4 Komponentais grįsta architektūra

Komponentais pagrįstas programavimas (*Component Based Development (CBD)*) – viena iš gana populiarių frazių (*buzz words*) programinės įrangos kūrime. Kaip ir šios frazės pirmtakų, taip ir šios koncepcijos idėja yra kurti gerą programinę įrangą kiek galima pigiau ir patikimiau; kaip ir ankstesnių panašaus pobūdžio sprendimų viltys viršijo tikrąją naudą; kaip ir iš kitų metodologijų, taip ir iš šios yra ko pasimokyti ir pamažu gerinti programinės įrangos kūrimo kaip inžinerinės srities suvokimą [39].

3 pav. "Skambūs žodžiai" [39].

Komponentais grįstame projektavime yra išskiriamos trys atskiros veiklos:

- **Produkto konstravimas** – greitas produktų konstravimas iš konfigūruojamų komponentų.
- **Komponentų projektavimas** – specifinių komponentų realizacija.
- **Rinkinio architektūra** – sąsajų tarp komponentų specifikavimas.

2.4.1 Komponentais grįstos architektūros privalumai ir trūkumai

Komponentais grįsta architektūra turi daug privalumų: **efektyvesnis sudėtingumo valdymas**, **sumažintas produkto pateikimo rinkai laikas**, **padidėjęs produktyvumas**, **pagerinta programinės įrangos kokybė**, **aukštesnis nuoseklumo lygis**, **platesnės panaudojamumo ribos** [40]. Tačiau ta pati architektūra turi keletą trūkumų ir rizikų, kurios gali statyti į pavojų šios metodikos sėkmę:

- **Laikas ir pastangos reikalingos komponentų kūrimui.** Vieni iš pagrindinių faktorių, kodėl programuotojų komandos gali vengti komponentais grįstos architektūros sprendimų, yra laikas ir pastangos reikalingos pakartotinio panaudojimo elementų sukūrimui [41, 42].

- **Neaiškūs ir nevienareikšmiški reikalavimai.** Bendrai paėmus reikalavimų valdymas yra labai sudėtinga ir svarbi programinės įrangos kūrimo proceso fazė. Šio etapo tikslas yra apibrėžti nuoseklius ir pilnus reikalavimus komponentui. Pakartotinio panaudojimo komponentai pagal apibrėžimą yra elementai, kuriuos planuojama panaudoti kitose sistemose, kurių reikalavimai komponento kūrimo metu yra nežinomi ir dažniausiai negali būti nuspėjami, prognozuojami. Ši taisyklė galioja ir funkciniams, ir nefunkciniams reikalavimams. Dėl šių priežasčių yra sunku teisingai nustatyti reikalavimus, o tuo labiau pagal turimus reikalavimus sėkmingai suprojektuoti ir sukurti komponentus [43, 44].

- **Konfliktas tarp naudojimo ir pakartotinio panaudojimo.** Norint, kad komponentas būtų plačiai pakartotinai panaudojamas, jis turi būti kiek galima bendresnis, keičiamo dydžio ir lengvai pritaikomas. Dėl šių priežasčių komponentas tampa sudėtingesnis (vadinasi ir sunkiau naudojamas) ir reikalauja daugiau skaičiavimo resursų (vadinasi jo panaudojimas yra brangesnis). Poreikis pakartotiniam panaudojimui skatina projektuoti abstraktesniame lygyje, dėl to vėlgi sumažinamas pradinis komponento lankstumas ir prarandama komponento savybė būti lengvai įdiegiamu, nors tokiu būdu ir pasiekiamas paprastumas [41, 42].

- **Komponento palaikymo kaštai.** Nors visos programų sistemos palaikymo kaštai komponentais grįstos architektūros pagalba gali būti sumažinti, vis dėlto komponento palaikymo kaštai gali būti labai dideli, nes komponentai turi išpildyti skirtingus reikalavimus skirtingoms sistemoms, veikiančioms skirtingose aplinkose, kurios turi

skirtingus palaikomumo reikalavimus ir tikriausiai teikia skirtingus palaikymo lygius [41].

- **Patikimumas ir jautrumas pokyčiams.** Dėl priežasties, jog komponentai ir sistemos turi atskirus gyvavimo ciklus ir skirtingus reikalavimus, egzistuoja rizika, kad komponentas nepilnai tenkins specifinius tam tikros programinės įrangos reikalavimus arba turės savybių, apie kurias nebus informuotas programinės įrangos kūrėjas. Kai programų sistemos kūrėjai nusprendžia atlikti tam tikrus pakeitimus programinės įrangos aplinkoje (pvz., operacinės sistemos atnaujinimas, kitų komponentų atnaujinimas ar kiti sistemos pakeitimai), jie turi pripažinti, jog nėra tikri, jog pakartotinai naudojamas komponentas nesukels nenumatytų problemų ar klaidų [45].

2.5 Komponentais grįstos architektūros JAVA karkasai

Šiame poskyryje bus apžvelgiami trys JAVA karkasai palaikantys komponentais grįstą architektūrą žiniatinklio programinės įrangos kūrime JAVA technologijų pagalba: „Wicket“ [3], „JavaServer Faces“ [4] ir „Tapestry“ [5]. JAVA karkasų analizė buvo atlikta priimant sprendimą, kurį produktą pasirinkti ir adaptuoti informacinės sistemos modelio realizacijai.

2.5.1 „Wicket“

„Wicket“ yra komponentais pagrįstas internetinių sistemų kūrimo karkasas, skirtas JAVA programavimo kalbai. 2005 metų birželio mėnesį buvo išleista 1.0 versiją, todėl galime teigti, kad „Wicket“ yra gana „jaunas“ karkasas palyginus su kitais („Tapestry“, „JavaServer Faces“).

Pasak karkaso kūrėjų, „Wicket“ privalumai lyginant su kitais siūlomais šablonais yra šie:

- Aiškus būsenų valdymas (Transparent state management);
- „Wicket“ puslapiai gali būti suformuoti, atvaizduoti ir vėliau pakoreguoti naudojant standartinį WYSIWYG (What you see is what you get) HTML dizaino priemones;
- Dinaminis turinio apdorojimas ir formų valdymas yra pilnai atliekamas JAVA kode;
- Nėra XML konfigūracijos failų.

Pagrindinės „Wicket“ savybės:

- „Swing“ [7] tipo objektiškai orientuotas komponentais grįstas modelis;

- Kūrimo lengvumas;
- Lygių atskyrimas (atskiri HTML ir JAVA failai);
- Saugumas;
- Skaidrus mygtuko „Atgal“ palaikymas;
- **Pakartotinio panaudojimo komponentai;**
- Paprastas, lankstus, lokalizuotas formų validavimas;
- Visos operacijos gali būti plečiamos „Factory“ metodų pagalba;
- Atskiriami modeliai;
- „Border“ komponentai;
- Palaikomos visos HTML savybės;
- Atributų manipuliavimas;
- Automatinė konversija;
- Dinaminiai paveiksliukai;
- Navigacijos nuorodos skirstymui puslapiais;
- Medžio komponentas;
- Lokalizacija;
- Pavyzdžiai.

„Wicket“ kaip ir „Tapestry“ naudoja specialią HTML žymę identifikuoti komponentus, o iš „Echo“ [8] paėmė „pirmos-klasės“ komponentų modelį.

Naujų karkasų kūrimas yra visada lengvesnis remiantis kitų klaidomis ir priimtais sprendimais. Sena patarlė sako: „Kam išradinėti ratą iš naujo?“, o „Wicket“ į tai atsako: „norime sukurti apvalesnį“.

2.5.2 „JavaServer Faces“

„JavaServer Faces“ (JSF) – tai „Sun Developer Network“ komandos sukurta JAVA technologija interneto sprendimams, kuri apima:

- API(***a**pplication **p**rogram **i**nterface)* rinkinį, kuris reprezentuoja UI (*user interface*) komponentus ir tų komponentų būsenų valdymą, įvykių priežiūrą, įeinančių duomenų validavimą, puslapių navigaciją ir internacionalizavimo palaikymą ir prieinamumą

- JSP (**J**ava**S**erver **P**ages) standartinių žymių biblioteką, skirtą JavaServer Faces sąsajų susiejimui su JSP puslpiais.

Pirmiausia, JSF programa yra servlet/JSP programa. „JavaServer Faces“ yra pagrįsta įvykiais ir jų valdymu. Kaip teigiama [9], JSF įgalina programuotojus interneto komponentus naudoti interneto puslapiuose ir fiksuoti vartotojo veiksmų sukeltus įvykius. Tikimasi (2003), kad netolimoje ateityje daugelis JAVA priemonių palaikys šią technologiją. Portalų, svetainių ir kitų žiniatinklio sistemų kūrimas bus toks pat paprastas kaip ir „Swing“ [7] programėlių rašymas (2003). Šaltinyje [10] netgi teigiama, kad „JavaServer Faces“ karkasas gali tapti tiesioginiu konkurentu „Microsoft .Net“ WebForm'oms. Ar tokios prognozės išsipildys, sužinosime po kelių metų.

JSF karkaso kūrėjai tikėjosi sustabdyti palaipsniui vis didėjantį JAVA interneto technologijų karkasų kūrimą (2002 metais jų buvo jau 35) (žr. 4 pav. Ką siūlo JSF? [11]).

4 pav. Ką siūlo JSF? [11]

Nors 2002 metais šiai naujai technologijai buvo žadama šviesi ateitis, tačiau 2006 metais JSF dar nebuvo apvertusi JAVA programuotojų pasaulio aukštyn kojom. Vis dar yra teigiama, kad JSF yra gana sudėtingas standartas, retai taikomas didesniems projektams [12], nors buvo įdiegtas į keletą kitų karkasų, sukurtos „Plug-in“ priemonės pagrindinėms JAVA kalbą palaikančioms IDE aplinkoms, „JSFcentral.com“ portale pateikiami keli realizuoti produktai [13], skatinama šią technologiją naudoti su AJAX [14].

2.5.3 „Tapestry“

„Tapestry“ yra atviro kodo karkasas, skirtas kurti dinamines, patikimas, smarkiai plėtojamas interneto sistemas JAVA kalba. Kadangi „Tapestry“ komplektuojamas ir kuriamas iš standartinių JAVA Servlet API (***a****pplication* ***p****rogram* ***i****nterface)*, šis karkasas veikia bet kuriame servlet talpyklos ar taikomųjų programų serveryje („Jakarta Tomcat“, „Jetty“, etc.). „Tapestry“ suskirsto interneto sistemą į puslapių aibę, kiekvienas puslapis konstruojamas iš eibės komponentų. Toks sistemos suskirstymas užtikrina nuoseklią struktūrą, kuri lemia galimybes konstruoti URL, saugoti pastovias būsenas (persistent state storage) kliento arba serverio pusėje, validuoti vartotojo įvedamų duomenų srautą, suteikti lokalizacijos / internacionalizacijos savybes ir lengvai valdyti išimtis (exceptions).

„Tapestry“ programinės įrangos kūrimas apima HTML šablonų kūrimą naudojant gryną HTML, sukurtų šablonų kombinavimas su JAVA kodu naudojant XML apibūdinamuosius failus. „Tapestry“ siūlo sistemą suvokti kaip objektų ir tų objektų metodų ir parametrų rinkinį, bet tikrai ne kaip URL ir užklausų aibę.

„Tapestry“ karkasas skatina grynojo objektiškai orientuoto programavimo taikymą žiniatinklio portalams ir kitoms sistemoms. Karkasas specialiai suprojektuotas taip, kad nesunkiai būtų kuriami nauji komponentai, nes būtent komponentų kūrimas yra sistemos plėtojimo rutina.

Karkaso pradinėje versijoje pateikiama apie 50 pradinių standartinių komponentų: nuo paprastų išeities komponentų iki sudėtingų duomenų lentelių ir medžio struktūros navigatorių.

„Tapestry“ architektūra suprojektuota taip, kad kuriama sistema nuo mažos programėlės gali būti vystoma iki masyvaus kelių šimtų puslapių portalo, kuriamo didelės programuotojų komandos.

Pagrindinės „Tapestry“ savybės:

- Paprastumas;

- Nuoseklumas (kas veikia puslapiuose, tas veikia komponentuose, kas veikia mažose programėlėse, tas veiks ir didesnėse, skirtingi programuotojai panašioms problemoms turėtų rasti panašius sprendimus);
- Efektyvumas;
- Grįžtamasis ryšys.

2.5.4 Kokybinis karkasų palyginimas

Labai detalus ir išsamus dviejų (JSF ir „Tapestry“) karkasų palyginimas pagrįstas praktika yra pateiktas „TheServerSide“ portale [15]. Analizės išvadose Phil Zoio teigia, kad liko sužavėtas „Tapestry“ galimybės, lankstumu, brandumu, standartinių komponentų įvairove, tačiau pripažįsta, kad programuotojams vidutiniokams šis karkasas gali pasirodyti sudėtingas; tuo tarpu nors ir JSF pasižymi neįtikėtinomis lankstumo, pritaikymo galimybėmis, autorius drįsta abejoti technologijos sėkme ateityje, dėl JSP šablonų naudojimo nepatrauklumo (autorius tikisi, kad jie bus pakeisti naujose versijose kažkuo panašiu į „Tapestry“ XML savybių failus), dėl komponentų kūrimo sudėtingumo.

Taip pat smagi karkasų analizė pateikiama Matt Raible prezentacijoje [11]. Tiesa, joje paminimi tik JSF ir „Tapestry“.

Deja, pilnos ir detalios „Tapestry“ ir JSF palyginimo su „Wicket“ analizės literatūros šaltinių nepavyko rasti.

Egzistuoja gana informatyvus interviu su kiekvieno karkaso autoriumi. Visiems kūrėjams yra pateikiami tie patys klausimai. [16]. Projektų vadovai atsakydami į klausimus pabrėžia karkaso privalumus, išdėsto, kokioje taikymo srityje jų sukurtas produktas nelabai tinkamas, įvertina kitus karkasus, atskleidžia numatytas naujas karkasų galimybes planuojamose naujose versijose. Apklausą rėmė ir įvykdė „ServerSide JAVA Symposium“ ir „Virtuas“.

Peržvelgus egzistuojančias analizes ir literatūros šaltinius, sudaryta karkasų privalumų ir trūkumų lentelė (žr. 1 lentelė. Karkasų „Wicket“, „Tapestry“, „JSF“ privalumai ir trūkumai.).

1 lentelė. Karkasų „Wicket“, „Tapestry“, „JSF“ privalumai ir trūkumai.

	Privalumai	**Trūkumai**
Karkasas „Wicket“	Saugumas, paprastumas, lankstumas, plečiamumas, pakartotinio panaudojimo komponentų palaikymas, visiškai nepriklausomi komponentai, „švarūs“ HTML šablonai, palaikomumas, perimtos visos gerosios „Tapestry“ savybės ir padengtos blogosios.	„Jaunumas“, XML atsisakymas, nepateikia navigacijos ir būsenų specifikacijos, netinkamas dideliems tekstinės informacijos kaupimo portalams.
Karkasas „Tapestry“	Pakartotinio panaudojimo komponentų palaikymas, brandumas, pragmatiškumas, gausus rinkinys standartinių komponentų, naujų komponentų kūrimas yra paprastesnis nei JSF, informatyvus klaidų raportavimas, kodo produktyvumas, patogesnis šablonų kūrimas, HTML šablonai lengvai tobulinami dizainerių, formų validavimas.	Sunkiai įsisavinamas, menka ir „sausa“ dokumentacija, mažai pavyzdžių, nelabai tinkamas smulkiems projektams.
Karkasas „JavaServer Faces“	Lankstumas, neribotos pritaikymo galimybės, greitumas, „turtingas“ navigacijos šablonas, naudojimo lengvumas, logikos valdymas per įvykius, yra sukurti specialūs įrankiai.	Abstraktumas, „jaunumas“, komponentų kūrimo sudėtingumas, JSP šablonai, yra saugumo „skylių“, netinkamas dideliems tekstinės informacijos kaupimo portalams, paslėptas HTML kodas, reikalauja daug konfigūracinių veiksmų.

3 EKSPERIMENTINĖ PAKARTOTINIO NAUDOJIMO KOMPONENTAIS GRĮSTA INFORMACINĖ SISTEMA

Sistemos esminis principas yra suteikti galutiniam vartotojui galimybę kiek galima daugiau prisidėti prie sistemos projektavimo. Šiam tikslui buvo sukurtas programų sistemos modelis, kuris susideda iš:

sistemos branduolio,

standartinių sistemos komponentų rinkinio ir

priklausomai nuo užsakovo ir projekto poreikių specifinių komponentų rinkinio.

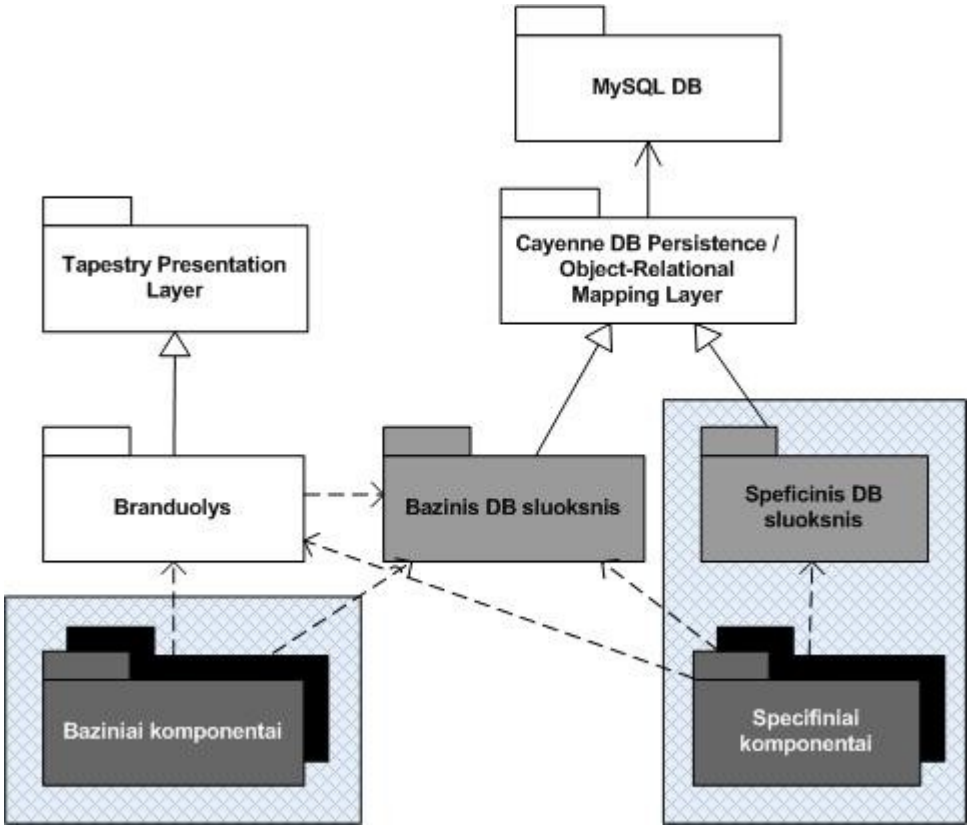
Sistemos branduolys metaprojektavimo terminais gali būti vadinamas „sėkla“, t.y., pagrindas, kuris yra bet kurios kitos būsimos sistemos pamatas, kurio veiklos logika, funkcijos, struktūra ir vartotojo sąsaja orientuota į vartotojo veiksmus iš jam siūlomų egzistuojančių standartinių komponentų bibliotekos susirinkti pageidaujamą programų sistemą (sistemos surinkimo procesas metaprojektavimo terminologijoje vadinamas evoliuciniu procesu).

Eksperimento metu buvo realizuota podiplominių studijų informacinė sistema, skirta Kauno medicinos universiteto podiplominių studijų centrui ir Vilniaus universiteto Medicinos fakulteto podiplominių studijų skyriui. Užsakovo sistema yra didelė didelių duomenų kiekių talpykla. Panašūs veiksmai (sukurti, trinti, redaguoti, registruoti, įtraukti, perkelti ir pan.) atliekami su skirtingais duomenų tipais. Tokia pritaikymo sritis padėjo analizuoti komponentais grįstos architektūros tinkamumą informacinėms sistemos realizuoti, taikyti ir adaptuoti egzistuojančius pakartotinio panaudojimo komponentus, stebėti jų evoliucijos procesą palaipsniui juos pritaikant, fiksuoti, kokiais atvejais buvo kuriami nauji komponentai, kokiomis situacijomis ir kokie komponentai tapo labiausiai pritaikomi, lanksčiausi, universaliausi sistemos kontekste.

Detalesnis sistemos modelio aprašas pateikiamas tolimesniuose skyriuose.

3.1 Sistemos architektūrinis modelis

Kaip jau paminėta šio skyriaus įžangoje, sistemą sudaro universalus branduolys ir komponentai skirti konkrečioms užduotims atlikti (komponentų aibė gali būti skirstoma į standartinių komponentų aibę ir specifinių projektų komponentų aibę). Branduolio architektūra ir vartotojo sąsaja parengta taip, kad suteiktų galutiniam vartotojui galimybę pilnai formuoti sistemos struktūrą, navigaciją, susirinkti sistemos funkcionalumą iš egzistuojančių komponentų aibės, pagal komponento panaudos atvejus keisti tų komponentų parametrus. Sistemos architektūros modelis pavaizduotas žemiau paveikslėlyje.



5 pav. Sistemos architektūros modelis: tamsiai pilka pažymėti komponentų paketai, pilkšva spalva pažymėtas duomenų bazės sluoksnis, languota sritimi pažymėta sistemos dalis, kuri kinta priklausomai nuo projekto poreikių – komponentų aibė pildoma naujais komponentais, praplečiami egzistuojantys.

3.2 Sistemos branduolys

Šiame skyrelyje pateikiamas detalesnis sistemos branduolio sudėties aprašymas. Sistemos branduolys susideda iš šių esminių posistemių:

- **Vartotojų bei grupių valdymo posistemė.** Administratorius bei vartotojų administravimo teises turintys vartotojai gali peržiūrėti sistemoje esančius vartotojus, keisti jų asmeninę informaciją, metaduomenis bei priskirti ar šalinti juos iš grupės. Administratorius bei grupių administravimo teises turintys vartotojai gali peržiūrėti sistemoje esančias grupes, keisti grupės duomenis, grupės teises, metaduomenis, priskirti ar pašalinti grupei priklausančius vartotojus, o taip pat priskirti ar pašalinti grupei priklausančias kitas grupes (vidines grupes). Vieną ar kitą grupę administruojantis vartotojas turi galimybę siųsti grupinius laiškus visiems toje grupėje registruotiems vartotojams.

- **Puslapių kūrimo, pozicijos keitimo puslapių hierarchijoje bei navigacijos posistemė.** Puslapiai kuriami nepririšant jų prie konkrečios portalo vietos, taigi jie yra nepriklausomi ir tik sukurti būna nepasiekiami. Tam, kad puslapis būtų pasiekiamas, reikia įkelti nuorodą į puslapį navigacinio meniu komponentą arba nuorodą iš kito puslapio, naudojant vidines nuorodas. Tokiu būdu, į tą patį puslapį galime nukreipti ne iš vienos, bet iš daugelio vietų. Puslapius kurti ir šalinti gali puslapių kūrimo teises turintys vartotojai. Kuriant puslapį, leidžiama pasirinkti komponentų atvaizdavimo puslapyje schemą. Pavyzdžiui: galima pasirinkti, kad komponentai puslapyje būtų atvaizduoti dviem stulpeliais. Puslapio konfigūracijoje galima nustatyti, kad būtų rodoma paskutinė puslapio atnaujinimo data. Ji yra atnaujinama, kai atnaujinamas nors vienas tame puslapyje esantis komponentas.

- **Puslapių turinio redagavimas.** Puslapių turinį sudaro komponentai. Puslapyje gali būti neribotas kiekis net ir vienos rūšies komponentų. Komponentų padėtį puslapyje galima keisti. perstumiant juos aukštyn arba žemyn. Puslapius gali redaguoti puslapių redagavimo teises turintys vartotojai.

- **Teisių sistema komponentams bei kitiems sistemos objektams.** Kiekvienas sukurtas komponentas ar puslapis pagal nutylėjimą turi tam tikras priėjimo teises - tik sukūrus komponentą ar puslapį, jis tampa matomas bet kuriam sistemos vartotojui. Norint jo matomumą apriboti, reikia atitinkamai pakoreguoti priėjimo teises. Priėjimo teises yra

priskiriamos konkrečioms vartotojų grupėms. Atskirai vartotojams priėjimo teisių keisti negalima, tad, esant būtinybei, tai realizuojama, sukuriant grupę ir jai priskiriant atitinkamą vartotoją. Kiekvienas komponentas sistemoje gali turėti unikalias teises. Šias teises apibrėžia komponento kūrėjas.

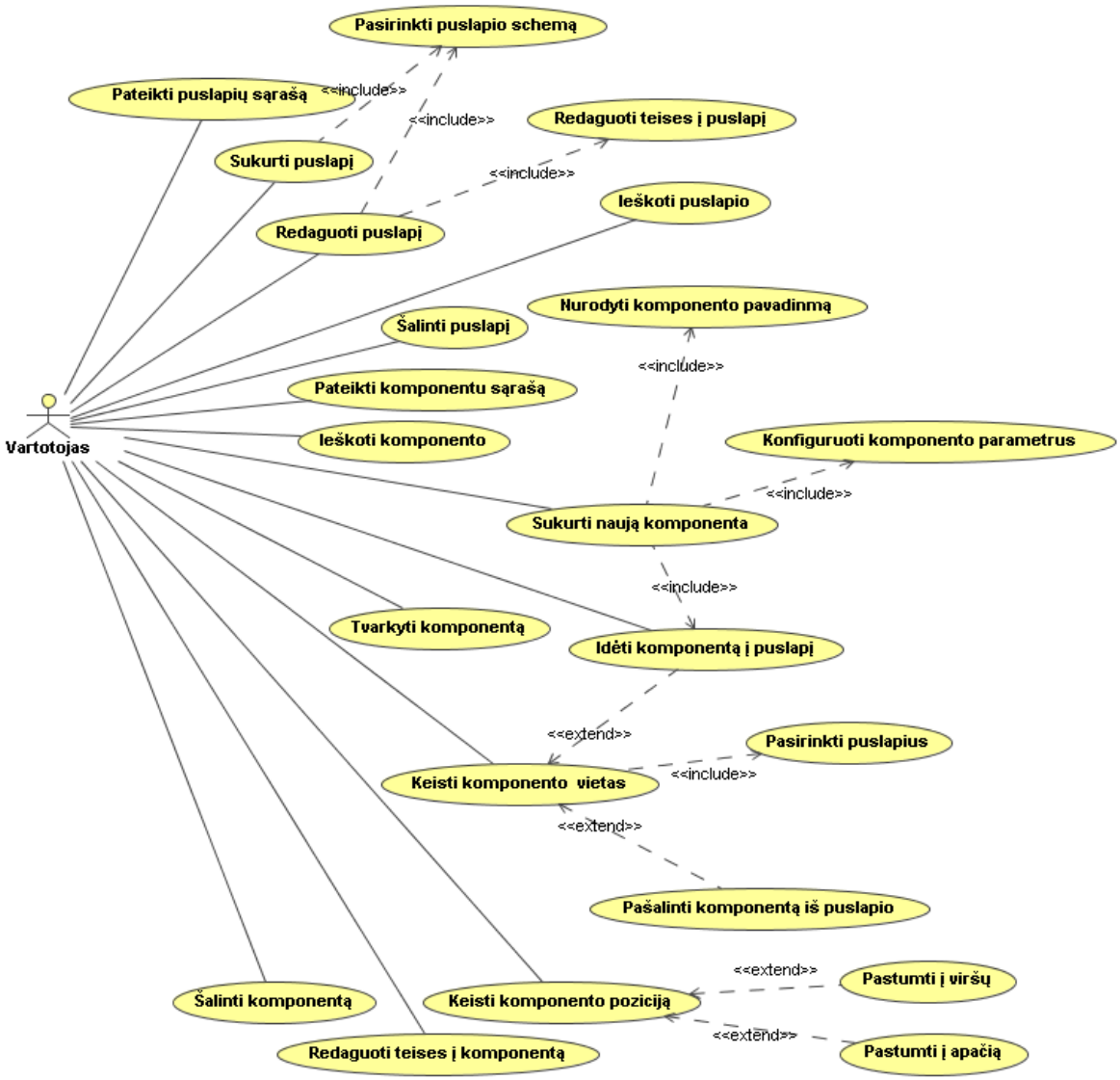
- **Komponentų kūrimas, įdėjimas, pozicijos keitimas, šalinimas.** Komponentai, kaip ir puslapiai, gali būti kuriami nepriklausomai nuo jų atvaizdavimo vietos. Tą patį komponentą galima atvaizduoti keliuose puslapiuose, kelis kartus tame pačiame puslapyje bei nesusiejant su konkrečiu puslapiu, tai yra atvaizduoti tik komponentą (tai galima padaryti, naudojant navigacinį meniu arba vidines nuorodas). Komponentus kurti ir šalinti gali komponentų kūrimo teises turintys vartotojai.

- **Komponentų tvarkymas ir konfigūravimas.** Kiekvienas komponentas yra unikalus ir turi savo parametrus. Keičiant parametrus, gali keistis komponentų funkcionalumas, atvaizdavimo būdas ar turinys. Priklausomai nuo pageidaujamo rezultato vartotojas, turintis konkretaus komponento konfigūravimo teises, gali keisti parametrus savo nuožiūra. Komponentas gali turėti savo duomenų tvarkymo puslapį. Pavyzdžiui, naujienų komponento tvarkymo puslapyje galime sukurti naujas bei šalinti senas naujienas, keisti naujienų duomenis bei turinį. Tvarkyti komponentus gali tvarkymo teises konkrečiam komponentui turintis vartotojas.

Visos šios posistemės suteikia neribotas galimybes ne tik į sistemos projektavimo etapą įtraukti vartotojus, bet tuo pačiu jiems suteikti teises į tam tikrų komponentų panaudojimą, taip pat tie patys galutiniai vartotojai gali kurti priemones valdyti žemesnio lygio vartotojus, skirti jiems teises į tam tikrus komponentus, puslapius, sistemos dalis. Galutinio vartotojo panaudos atvejai, skirti sistemos struktūros projektavimui, pateikti sekančiame skyriuje.

3.3 Puslapių ir komponentų valdymo posistemio panaudos atvejai

Šiame skyriuje pateikiami puslapių ir komponentų valdymo posistemio panaudos atvejai. Detalizuojamas būtent šis posistemis, nes jo teikiamos funkcijos įgalina vartotoją formuoti sistemą: kurti sistemos puslapius, sekcijas, formuoti puslapio turinį, rinktis skirtingus komponentus, juos sistemoje pakartotinai naudoti, kiek nori kartų, keisti jų paskirtį sistemoje nustatant atitinkamus komponento parametrus, keisti komponentų vietą, komponentų teises žemesnio lygio vartotojams. Panaudos atvejai detalizuojami žemiau pateikiamoje diagramoje (žr. 10 pav. Puslapių ir komponentų valdymo modulio informacijos srautai.). Kiekvienas panaudos atvejis detalizuojamas panaudos atvejų lentelėje (žr. 2 lentelė. Puslapių ir komponentų valdymo posistemio funkcionalumas.).



6 pav. Puslapių ir komponentų valdymo posistemio panaudos atvejų diagrama.

2 lentelė. Puslapių ir komponentų valdymo posistemio funkcionalumas.

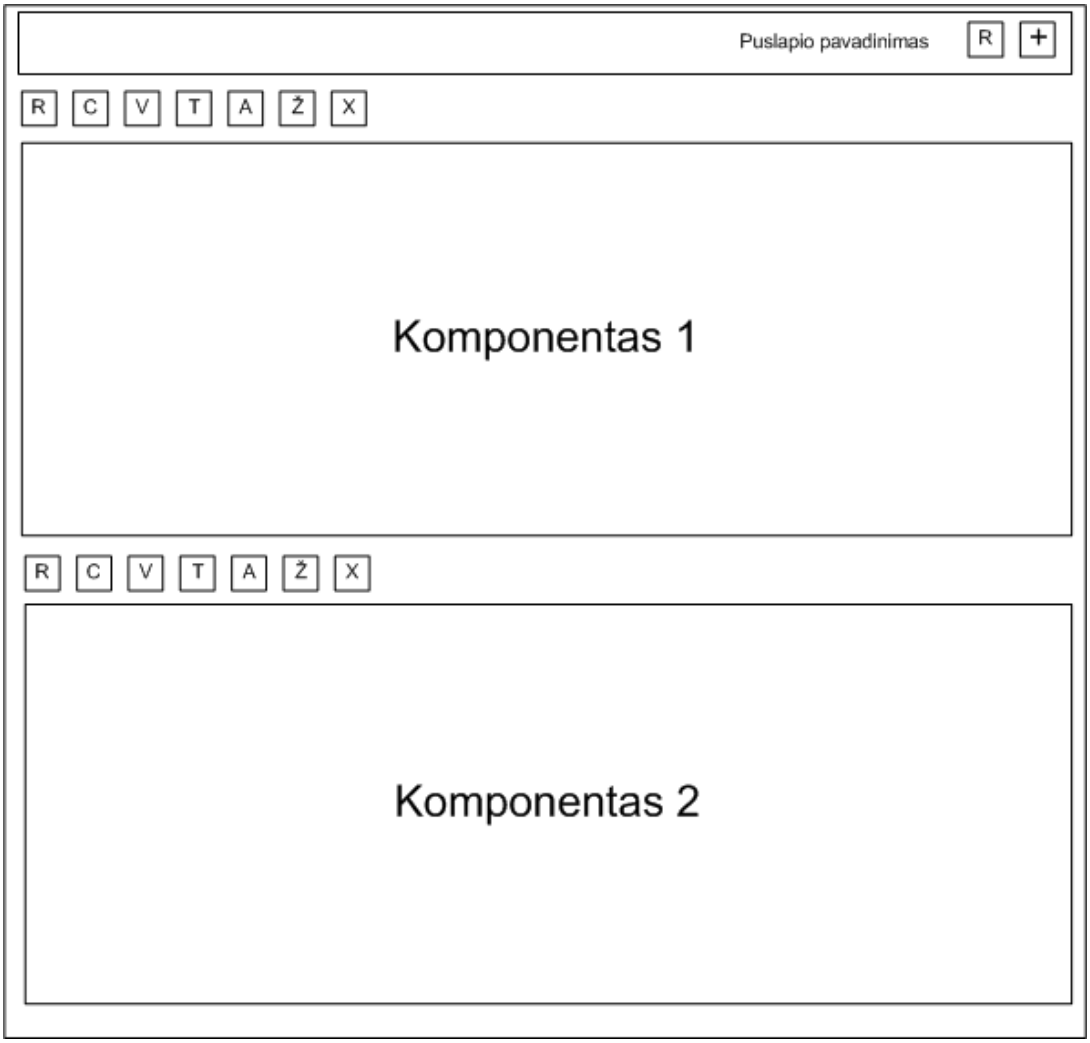
Funkcija	**Funkcijos aprašymas**
Peržiūrėti puslapių sąrašą	Vartotojas gali peržiūrėti puslapių sąrašą. Gali rikiuoti puslapius pagal puslapio vardą, vartotoją, kuris sukūrė puslapį. Sąrašas rodomas puslapiais jeigu duomenų yra daug. Prie kiekvieno puslapio pateiktas valdymo meniu.
Sukurti puslapį	Vartotojas kuria naują puslapį. Kuriant puslapį reikia įvesti pavadinimą, komponentų išdėstymo schemą bei kitus atributus.
Pasirinkti puslapio schemą	Vartotojas pasirenka norimą komponentų išdėstymo schemą puslapyje.
Redaguoti puslapį	Vartotojas keičia pageidaujamą puslapį. Jam pateikiami keičiami duomenys. Jo teisė rinktis, kurį lauką jis keičia, ar palieka senas reikšmes.
Redaguoti teises į puslapį	Vartotojas redaguoja teises į nurodytą puslapį. Galima pakeisti skirtingų grupių priėjimo prie nurodyto puslapio teises.
Ieškoti puslapio	Vartotojas gali ieškoti puslapio. Yra įvedama reikšmė ir paieška atliekama visuose laukuose, kurie yra pateikti puslapių sąraše – pavadinimas, kūrėjas.
Šalinti puslapį	Šalinamas nurodytas puslapis.
Peržiūrėti komponentų sąrašą	Vartotojas gali peržiūrėti komponentų sąrašą. Gali rikiuoti komponentus pagal pavadinimą, tipą ir vartotoją, kuris sukūrė puslapį. Sąrašas rodomas puslapiais jeigu duomenų yra daug. Prie kiekvieno komponento pateiktas valdymo meniu.
Ieškoti komponento	Vartotojas gali ieškoti komponento. Yra įvedama reikšmė ir paieška atliekama visuose laukuose, kurie yra pateikti komponentų sąraše – pavadinimas, tipas, kūrėjas.
Sukurti naują komponentą	Pasirinktame puslapyje vartotojas kuria komponentą. Komponentą pradedamas kurti pasirenkant komponento tipą. Kuriant komponentą atsidaro komponento konfigūravimo puslapis. Jame vartotojas nurodo komponento parametrus bei jo pavadinimą. Sukūrus komponentą, jis atsiranda puslapio viršuje.
Nurodyti komponento pavadinimą	Vartotojas įrašo komponento pavadinimą. Kiekvienas komponentas privalo jį turėti.
Konfigūruoti	Vartotojas konfigūruoja komponento parametrus. Kiekvienas

komponento parametrus	komponentas gali turėti savo individualius parametrus.
Įdėti komponentą į puslapį	Komponentas įdedamas į puslapį. Jis atsiranda pačiame puslapio viršuje.
Tvarkyti komponentą	Atidaromas komponento tvarkymo puslapis. Kiekvienas komponentas turi individualų puslapį. Pvz.: naujienų komponente sukuriamos šiandien įvykusios naujienos.
Keisti komponento vietas	Komponentas įdedamas arba išimamas nurodytuose puslapiuose.
Pasirinkti puslapius	Vartotojas pasirenka puslapius iš pateikto sąrašo.
Pašalinti komponentą iš puslapio	Vartotojas pašalina pasirinktą komponentą iš puslapio. Pašalina naudodamasis komponento valdymo meniu esančiomis funkcijomis.
Keisti komponento pozicija	Vartotojas keičia komponento pozicija puslapyje. Komponentą galima stumti į viršų arba apačią.
Pastumti i viršų	Komponentas stumiamas į apačią.
Pastumti į apačią	Komponentas stumiamas į viršų.
Redaguoti teises į komponentą	Vartotojas redaguoja teises į nurodytą komponentą. Galima pakeisti skirtingų grupių priėjimo prie nurodyto komponento teises.
Šalinti komponentą	Šalinamas nurodytas komponentas.

3.4 Komponentų integracijos į sistemą vartotojo sąsaja

Kiekvienas komponentas iš bibliotekos yra išorine struktūra analogiškas bet kuriam kitam, todėl gali būti patalpintas bet kurioje sistemos vietoje. Komponentai gali būti panaudoti pakartotinai neribojamą skaičių kartų, taip pat jų paskirtis sistemoje gali skirtis priklausomai nuo komponento konfigūracijos.

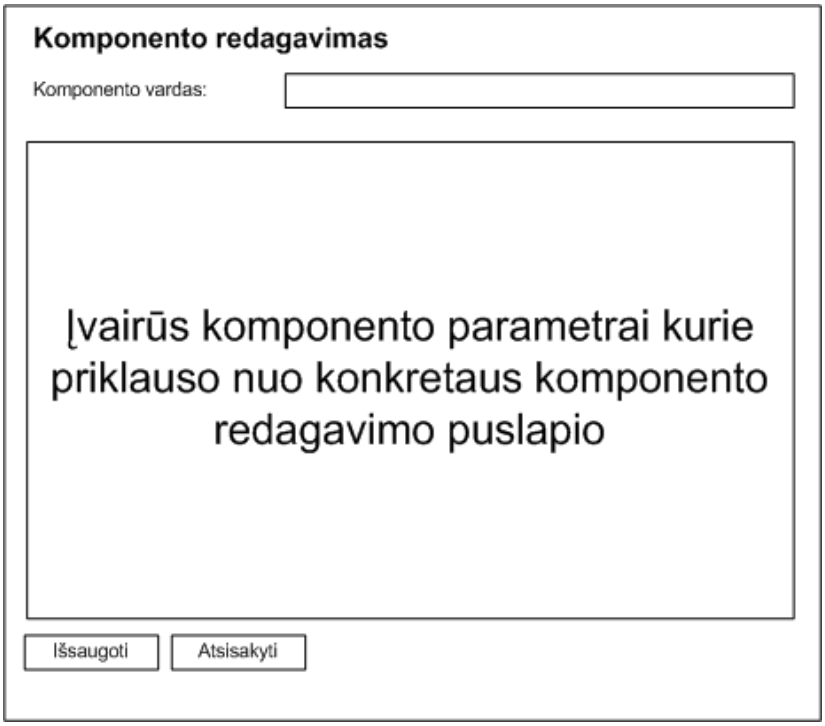
Komponentai talpinami į informacinės sistemos pasirinktas vietas vienas po kito arba keliais stulpeliais. Galutinis vartotojas renkasi komponento pridėjimo funkciją. Atsivertusiame lange vartotojas nurodo arba jau kitoje vietoje sistemoje sukurtą komponentą iš pateikto sąrašo, arba kuria naują komponentą ir identifikuoja jo tipą.



7 pav. Principinis puslapio vaizdas.

Paveikslėlyje (žr. 7 pav. Principinis puslapio vaizdas.) pateiktas galimas jau sukurtas informacinės sistemos sekcijos vaizdas. Viršuje matyti sekcijos redagavimo mygtukai: R –

mygtukas, kurį paspaudus būtų atverčiama esamo sekcijos redagavimo forma, + - mygtukas, kurį paspaudus būtų atverčiama komponento įdėjimo forma. Žemiau sekcijoje matome 2 komponentus – „Komponentas 1“ ir „Komponentas 2“. Tai gali būti bet kurie iš informacinės sistemos komponentų. Virš kiekvieno komponento matome komponento redagavimo meniu, kuriame pateiktas visas rinkinys valdymo mygtukų. R – mygtukas, kurį paspaudus atidaromas komponento turinio puslapis, C – mygtukas, kurį paspaudus atidaroma komponento redagavimo forma, V – atidaroma komponento vietų forma, T – teisių forma, A – mygtukas, kurį paspaudus komponentas stumiamas per viena poziciją aukštyn, Ž – mygtukas, kuri paspaudus komponentas stumiamas per vieną poziciją žemyn, X – komponento pašalinimo iš puslapio mygtukas.



8 pav. Abstrakti komponento redagavimo forma.

Paveikslėlyje (žr. 8 pav. Abstrakti komponento redagavimo forma.) pateikta abstrakti komponento redagavimo forma. Šioje formoje nustatomas komponento pavadinimas bei nustatoma jo konfigūracija. Konfigūracijos laukai skiriasi priklausomai nuo komponento. Už šios formos atvaizdavimą atsakingas konkretus komponentas.

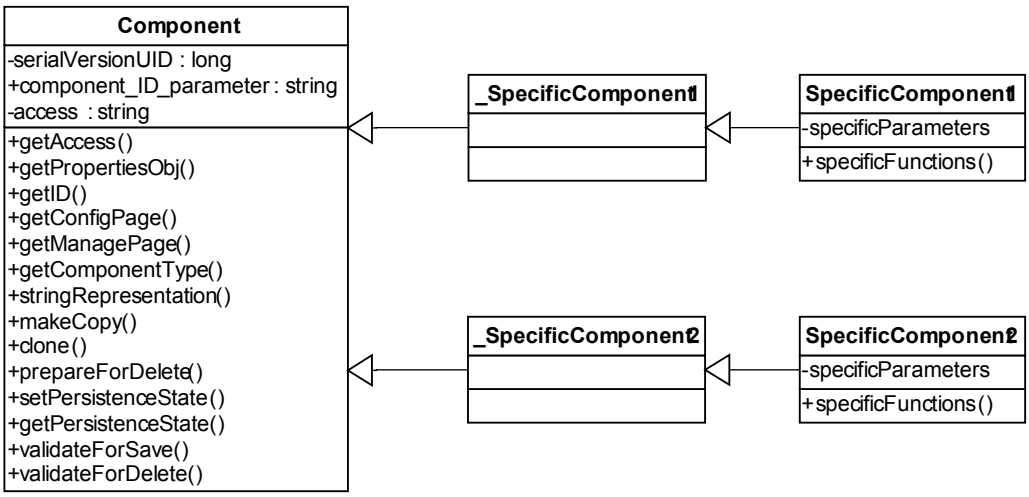
Tokiu principu galutinis vartotojas projektuoja sistemą, dėlioja komponentus, nustato jų parametru, identifikuoja teises, formuoja struktūrą.

Tolimesniame skyriuje aprašomas techninis komponentais grįstos architektūros sprendimo modelis.

3.5 Techninis komponentais grįstos architektūros modelis

Šiame skyriuje pateikiama komponentų klasių diagrama ir puslapių ir komponentų duomenų srautų diagrama. Komponentų pakartotinio panaudojimo universalumas realizuotas objektinio programavimo principais: su bet kuriuo komponentu yra elgiamasi analogiškai.

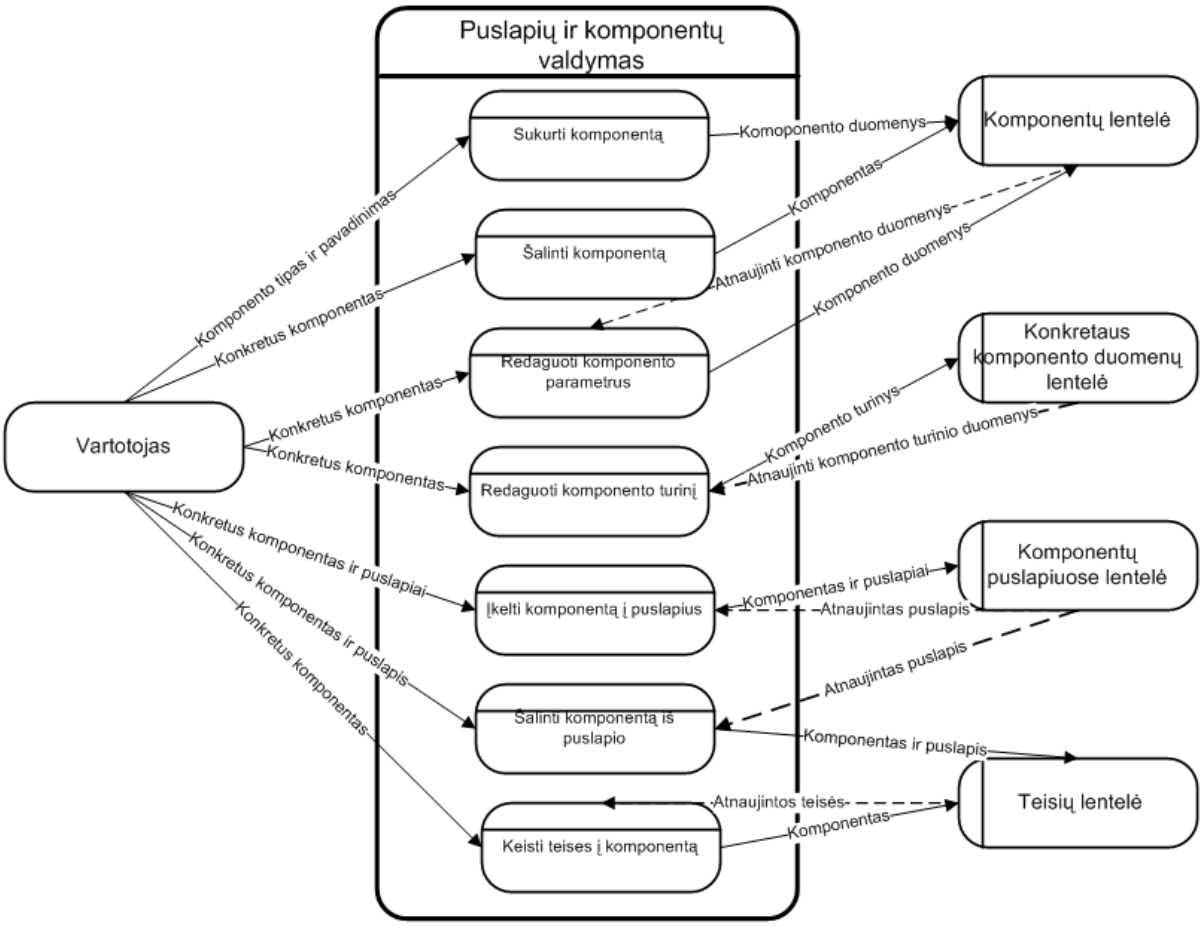
3.5.1 Komponentų klasių diagrama



9 pav. Komponentų dalies klasių diagrama.

Kiekvienas komponentas sukurtais šiai sistemai gali turėti rinkinį unikalių atributų. Visi komponentai su savo aprašais yra saugomi sistemos komponentų saugykloje. Komponento administruojamas bei kuriamas turinys yra saugomas atskiruose komponento lentelėse. Taigi kiekvienas komponentas turintis arba valdantis turinį privalo turėti ir savo turinio lenteles. Komponento duomenys taip pat yra ir sistemos duomenys. Kiekvienas komponentas yra atsakingas už savo duomenų valdymą, tvarkymą bei atvaizdavimą.

3.5.2 Puslapių ir komponentų duomenų srautų diagrama



10 pav. Puslapių ir komponentų valdymo modulio informacijos srautai.

Vartotojas puslapių ir komponentų valdymo modulio pagalba gali kurti, modifikuoti komponentus, įkelti juos į norimus puslapius, keisti priėjimo teises tiek prie komponentų, tiek prie puslapių.

3.6 Sistemos projektavimas funkciniu ir komponentiniu požiūriu

Projektavimo etape informacinė sistema buvo įsivaizduojamas, kaip į paslaugas orientuota sistema, t.y., visas sistemos funkcionalumas suskaidytas į funkcijų (funkcijų grupių) aibę, kurias turi atlikti programinė įranga:

- Kvalifikacijos kėlimo kursų ir grupių valdymas (kūrimas, redagavimas, šalinimas);
- Kursantų registracija;

- Kursanto paraiškų teikimas į pageidaujamų kursų grupes;
- Asmeninių paraiškų peržiūra;
- Kursantų paraiškų dviejų pakopų tvirtinimas/atmetimas;
- Kelialapių formavimas kursantams (peržiūra, spausdinimas);
- Atvykusių kursantų fiksavimas;
- Pažymėjimų kursantams formavimas (peržiūra, spausdinimas);
- Asmeninių pasiekimų peržiūra;
- Kursantų duomenų valdymas (redagavimas);
- Įstaigų, dalyvaujančių tęstinėse studijose, ir jų darbuotojų duomenų valdymas;
- Kursus teikiančių organizacijų ir jų darbuotojų duomenų valdymas;
- Kursus teikiančių organizacijų padalinių ir jų darbuotojų duomenų valdymas.

Visos išvardintos tęstinių studijų administravimo ir teikimo portalo funkcijos (funkcijų grupės) sistemos projektavimo atžvilgiu tapo komponentais (tam tikros funkcijos tapo realizuojamos per vieno ir to paties pakartotinio panaudojimo komponento pritaikymą). Tolimesniuose skyriuose pateikiama kelių komponentų projektavimo ir realizavimo specifika.

3.7 Konkrečių realizacijos komponentų aprašai

Šiame skyriuje pateikiami keli konkretūs komponentai, jų paskirtis, konfigūravimo galimybės.

3.7.1 Registracijos komponentas

Atlikus minimalią registracijos į portalus ir kitas sistemas proceso analizę, nustatyta, kad registracijos procesas susideda iš dviejų esminių funkcijų:

- Reikiamų duomenų surinkimas registracijos metu;
- Vartotojo įtraukimas į tam tikras teises po registracijos įgaunančių vartotojų grupę (-es).

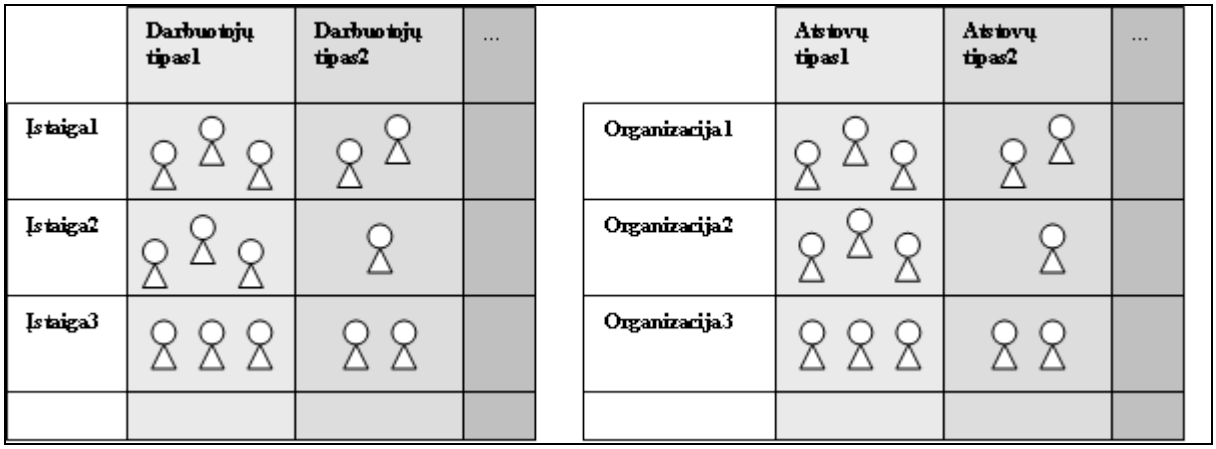
Dėl šių priežasčių registracijos komponento konfigūracijos metu turėtų būti suteikta galimybė nurodyti vartotojų grupes, į kurias registracijos metu įtraukiamas vartotojas, ir suformuoti registracijos formą iš pageidaujamų duomenų. Registracijos formos laukai yra lanksčiai konfigūruojami naudojant anketų ir klausimynų kūrimo principą.

Komponento pakartotinis panaudojimas yra palyginus paprastas. Šio komponento įsikėlimui į pageidaujamą sistemos vietą nereikalinga programuotojo pagalba. Su esminėmis komponento veiklos taisyklėmis susipažinęs sistemos administratorius ar kitas galutinis sistemos vartotojas gavęs panašias teises gali susikurti registraciją į visas egzistuojančias ar naujai sukurtas portalo vartotojų grupes.

3.7.2 Vartotojų grupių valdymas pagal pasirinktą tipą

Realizuojant funkcijų grupes „Įstaigų, dalyvaujančių tęstinėse studijose, ir jų darbuotojų duomenų valdymas“, „Kursus teikiančių organizacijų ir jų darbuotojų duomenų valdymas“ ir „Kursus teikiančių organizacijų padalinių ir jų darbuotojų duomenų valdymas“ buvo pastebėta, kad visos šios operacijos yra analogiškos, tik skiriasi apdorojami duomenys: skirtingo tipo vartotojų grupės (įstaiga, institucija, padalinys).

Dėl šios priežasties buvo sukurtas vartotojų grupių valdymo pagal pasirinktą tipą komponentas. Komponento konfigūracijos metu nurodomas grupių, kurių duomenys pageidaujama valdyti, tipą (įstaigos, institucijos, padaliniai) ir vartotojų grupes, kurioms, be jau pasirinkto tipo grupės, priklauso dominantys vartotojai. Grafiškai vartotojų priklausomybė grupėms pavaizduota diagramoje (žr. 11 pav. Pavyzdinių vartotojų grupių pagal tipą ir vartotojų pagal atsakomybes grupių sąryšis.).



11 pav. Pavyzdinių vartotojų grupių pagal tipą ir vartotojų pagal atsakomybes grupių sąryšis.

Pažvelgus į pavyzdį (žr. 11 pav. Pavyzdinių vartotojų grupių pagal tipą ir vartotojų pagal atsakomybes grupių sąryšis.) galima teigti, jog komponento konfigūracijos metu vartotojui suteikiama teisė rinktis grupės tipą (įstaiga, organizacija) ir dominančių vartotojų atsakomybių grupę(-es) toje grupėje (darbuotojų tipas1, darbuotojų tipas2 arba atstovų tipas1, atstovų tipas2).

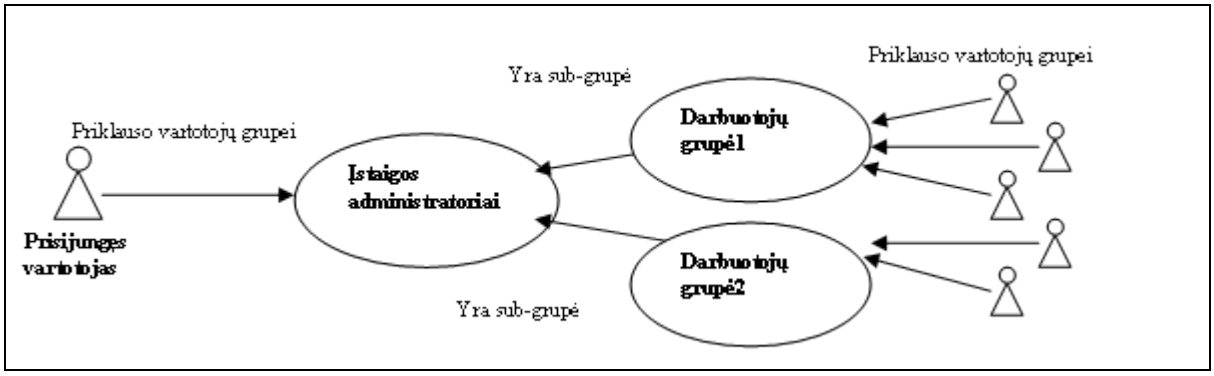
Vidinė komponento veiklos logika realizuoja grupių pagal pasirinktą tipą sąrašo pateikimą, kiekvienos iš grupių duomenų redagavimą, sąrašo papildymą, nebereikalingų grupių šalinimą. Taip pat pateikta galimybė valdyti kiekvienos grupės vartotojus: sukurti naujus, papildyti jau egzistuojančiais vartotojais iš kitų grupių, šalinti vartotojus iš grupės.

Šio pakartotinio komponento architektūros pagalba be papildomų programuotojo darbo laiko resursų buvo realizuotos iš karto trys informacinės sistemos funkcijos: tęstinių studijų teikimo ir valdymo sistemos galutinis vartotojas, turintis administratoriaus teises, trijuose skirtinguose puslapiuose patalpinto tą patį komponentą su skirtingais konfigūraciniais parametrais.

3.7.3 Vartotojų valdymas pagal asociaciją

Labai dažna informacinių sistemų funkcija yra tam tikrų sistemos vartotojų duomenų valdymas: įstaigos administratorius nori matyti įstaigos darbuotojų sąrašus, organizacijos administratorius – organizacijos personalą, visos sistemos administratorius – visus sistemos vartotojus ir pan. Konkrečiai tęstinių studijų administravimo ir teikimo portale yra poreikis matyti kursantų sąrašą, redaguoti jų duomenis. Dėl šios priežasties buvo nuspręsta sukurti komponentą vartotojų valdymui per asociaciją. Asociacija šiuo atveju reiškia, jog komponentas atrenka visus vartotojus iš komponentą peržiūrinčio vartotojo pogrupių. Šiuo atveju komponentas nėra konfigūruojamas. Reikiamą informaciją jis gauna iš komponentą

peržiūrinčio registruoto sistemos vartotojo (kokiai vartotojų grupei jis priklauso, kokie pogrupiai priklauso tai grupei). Grafiškai vartotojų priklausomybė grupėms ir grupių tarpusavio ryšiai pavaizduoti diagramoje (žr. 12 pav. Pavyzdinis vartotojas su asociacija į jo sub-grupių vartotojus.).



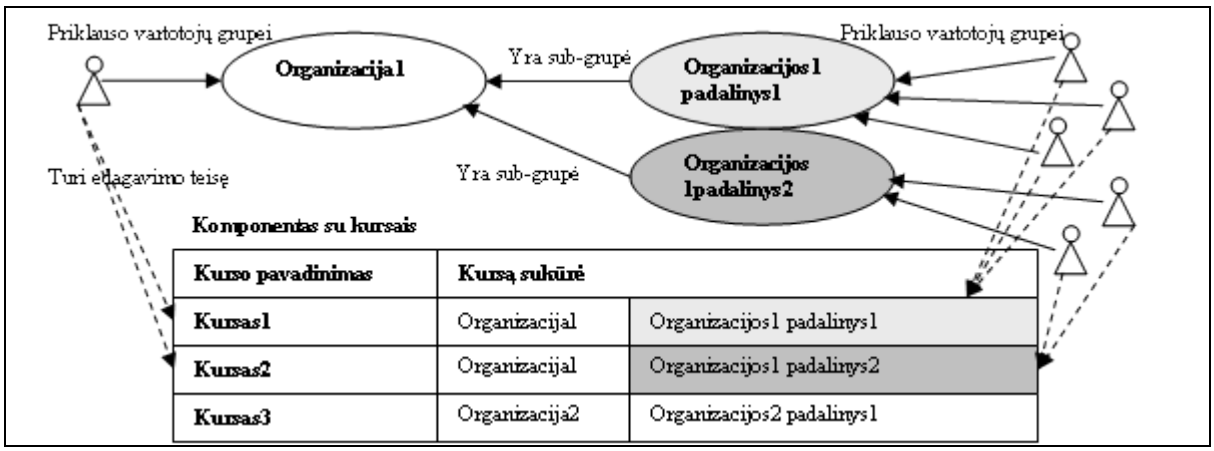
12 pav. Pavyzdinis vartotojas su asociacija į jo sub-grupių vartotojus.

Pagal diagramą (žr. 12 pav. Pavyzdinis vartotojas su asociacija į jo sub-grupių vartotojus.) prisijungęs vartotojas komponento pagalba bus įgalintas matyti visus savo vartotojų grupės pogrupių vartotojus ir pilnai valdyti jų duomenis.

Šio komponento pagalba buvo realizuota funkcija „Kursantų duomenų valdymas (redagavimas)“ ir tikimasi šį komponentą pritaikyti panašiose funkcijose toliau plėtojant šią informacinę sistemą (pavyzdžiui, organizacijos administratorius norės valdyti visus organizacijos darbuotojus), o taip pat kuriant kitas sistemas ateityje.

3.7.4 Kursų ir grupių valdymo komponentas

Tęstinių studijų teikimo ir valdymo informacinės sistemos projektavimo metu buvo sukurtas dažnai mokymo(si) proceso organizavime reikalingas komponentas kursų ir jų grupių valdymui. Šis komponentas įgalina sistemos administratorių ar kitą atsakingą galutinį vartotoją pilnai valdyti informaciją apie kursus ir jų grupes nepriklausomai nuo jų skaičiaus, tipo. Komponento veikimo logika realizuota taip, kad sistemos vartotojai mato visus publikuotus (publikavimą galima stabdyti) kursus, bet redaguoja ir šalina tik savo organizacijos ar organizacijos padalinių (priklausomai nuo vartotojo priklausomybės vartotojų grupėms) sukurtus kursus. Veikimo logika pavaizduota diagramoje (žr. 13 pav. Vartotojo asociatyvios teisės kursų redagavime.).



13 pav. Vartotojo asociatyvios teisės kursų redagavime.

4 Komponentų gyvavimo ciklo tyrimas

Šiame skyriuje pateikiamas eksperimente aprašytos (3 Eksperimentinė pakartotinio naudojimo komponentais grįsta informacinė sistema) sistemos komponentai, tų komponentų pritaikymas skirtinguose projektuose, kiek kokių komponentų buvo pakartotinai pritaikyta skirtinguose projektuose, kokią paskirtį jie ten atliko.

4.1 Komponentų tipai ir jų paskirtis

Lentelėje (3 lentelė. Komponentų tipai ir jų paskirtis.) pateikiami sistemos komponentų tipai ir trumpi tų komponentų bendrinės paskirties aprašymai. Paryškinti komponentai išsamiau buvo aprašyti šio dokumento kitame skyriuje (žr. 3.7 Konkrečių realizacijos komponentų aprašai).

3 lentelė. Komponentų tipai ir jų paskirtis.

Nr.	Komponento pavadinimas	Komponento paskirtis
1.	**Anketų komponentas**	Komponentas skirtas klausimynams, anketoms kurti. Komponento dizaineriui yra suteikiama galimybė kurti anketas/klausimynus iš įvairių tipų klausimų, rinkti ir stebėti tų anketų užpildymo statistiką, ją eksportuoti į bylą.
2.	**Apklausų komponentas**	Komponentas skirtas rengti balsavimus portale. Juo naudojantis portalo administratorius turės galimybę apklausi svetainės lankytojus. Vizualiai matyti rezultatus.
3.	**Forumo komponentas**	Forumo komponentas savo funkcijomis ir išvaizda primena vieną populiariausių „phpBB“ sistemų. Forumą sudaro diskusijos, kurios susideda iš temų. Temose yra skelbiamos žinutės, kuriomis reiškiama dalyvių nuomonė konkrečia tema. Temoms gali būti suteikiami kelių lygių prioritetai, o norint nutraukti diskusijas, tema gali būti užrakinama. Rašant žinutę galima prisegti failą, kurį vėliau visi forumo lankytojai gali atsisiųsti.

4.	**Grupių valdymo pagal tipą komponentas**	Sistemos vartotojų teisių paskirstymas yra paremtas vartotojų grupėmis. Projekto „Medas“ eigoje atsirado poreikis atskirai valdyti tam tikras vartotojų grupes pagal tipą, su jomis atlikti analogiškus veiksmus: papildyti grupėmis, ištrinti grupes, redaguoti tų grupių duomenis, papildyti vartotojais, ištrinti, redaguoti jų duomenis.
5.	**Kursų katalogo komponentas**	Komponentas skirtas kursų ir jų grupių valdymui. Atsakingam vartotojui suteikiamos teisės nustatyti kurso pradžios, pabaigos, registracijos pradžios, pabaigos datas, sukurti kursų grupes ir pan. Taip pat šiam komponentui yra sukurtas teisių valdymas: institucijos valdo tik savo kursus, aukštesnio lygio organizacijos gali valdyti joms priklausančių institucijų kursus ir pan.
6.	**Meniu komponentas**	Komponentas skirtas praplėsti standartinio meniu, esančio turinio valdymo sistemoje, galimybes. Kadangi toks meniu komponentas gali būti patalpintas viename ar keliuose puslapiuose, jis sistemai suteikią galimybes realizuoti pačias sudėtingiausias hierarchines bei tinklines portalo navigacijos struktūras, pvz., portale kurti institucijų ar net asmenų individualias svetaines su savo meniu, kurie tarpusavyje gali sietis per nuorodas arba bendrus puslapius.
7.	**Naujienų komponentas**	Labai lankstus konfigūracijos atžvilgiu komponentas. Komponentas, skirtas automatizuotiems naujienų pranešimams rodyti. Kuriami naujienų pranešimai, nurodant įvykio laiką bei pranešimo skelbimo pradžią ir pabaigą (jei aktualus naujienos senaties terminas). Yra galimybė archyvuoti bei komentuoti naujienas.
8.	**Paprasto paragrafo komponentas**	Pagrindinis komponentas statinei informacijai įvesti. Komponentas supranta ne tik paprastą tekstą, bet ir HTML kalbą. Į komponentą yra integruota „JavaScript“ kalba paremta „WYSIWYG“ principu (ką matai, tą ir turi), atviro kodo redaktorius „FCKEditor“, kuriuo nesunkiai ir patogiai redaguojamas komponento atvaizduojamos informacijos turinys. Taip pat naudojantis šiuo komponentu bus galima įdėti paveiksliukus, failus bei kitus resursus. Komponentas teiks galimybę kurti medžiagą „ką matai, tą gauni“ režimu.
9.	**Paveiksliukų galerijos komponentas**	Komponentas skirtas sukelti paveiksliukų galerijas. Bus galima kurti daug albumų. Galimybė lengvai naršyti albumuose esančias nuotraukas. Taip pat bus

		galima pareikšti savo nuomonę nuotraukų atžvilgiu – jas komentuoti, už jas balsuoti.
10.	**Registracijos į pasirinktas vartotojų grupes komponentas**	Sistemos vartotojų teisių paskirstymas yra paremtas vartotojų grupėmis. Dėl šios priežasties atsiranda poreikis registruoti pageidaujančius vartotojus į šias grupes. Šiam tikslui buvo suprojektuotas ir sukurtas registracijos komponentas, kuris realizuoja funkcijas gautų paraiškų tvirtinimą, atmetimą, ištrynimą, vartotojo duomenų redagavimą, vartotojų importavimą iš skirtingų formatų bylų, surinktų paraiškų eksportavimą į bylą.
11.	**Vartotojų valdymo pagal asociaciją komponentas**	Sistemos vartotojų teisių paskirstymas yra paremtas vartotojų grupėmis. Aukštesnio lygio vartotojai turi teises į žemesnio lygio vartotojų valdymą. Šiam tikslui buvo sukurtas vartotojų valdymo pagal asociaciją komponentas.

4.2 Komponentų pritaikymo dažnis

Žemiau pateikiama komponentų lentelė (žr. 4 lentelė. Komponentų pakartotinis pritaikymas skirtinguose projektuose.), kurioje išvardinti sistemos komponentų tipai ir jų pakartotinis pritaikymas skirtinguose projektuose. Nurodytas kiekis komponentų, kiek jų buvo įkeltą į tą sistemą iki 2007 metų gegužės 25 dienos.

4 lentelė. Komponentų pakartotinis pritaikymas skirtinguose projektuose.

NR.	Projektas / Komponentas	VIPT	MEDAS	EVETE	EQUAL - GREITKELIS	LieDM
1.	**Anketų komponentas**	1	-	-	6	11
2.	**Apklausų komponentas**	5	-	-	2	1
3.	**Forumo komponentas**	76	1	3	4	3
4.	**Grupių valdymo pagal tipą komponentas**	-	4	-	-	-
5.	**Kursų katalogo komponentas**	-	1	-	-	1
6.	**Meniu komponentas**	94	-	3	12	3
7.	**Naujienų komponentas**	83	1	4	3	4
8.	**Paprasto paragrafo komponentas**	1549	29	94	99	57
9.	**Paveiksliukų galerijos komponentas**	74	-	21	2	1
10.	**Registracijos į pasirinktas vartotojų grupes komponentas**	1	1	31	-	1
11.	**Vartotojų valdymo pagal asociaciją komponentas**	1	1	-	-	-

4.3 Komponentų vystymosi ciklas ir taikymai

4.3.1 Anketų komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“ atlikti tam tikrų regionų apklausas.

Taikymai: komponentas buvo išplėstas ir pritaikytas projektuose „Equal-Greitkelis“ ir „LieDM“, kuriuose buvo poreikis apklausti tam tikras tikslines grupes, surinkti statistiką, atlikti analizę. Besivystant projektams yra tikimybė, kad šio komponento taikymų skaičius išaugs.

Planuojami atnaujinimai: klausimus bus galima importuoti iš kitų tam tikrus standartus palaikančių sistemą, pavienių atsakytų anketų eksportavimas.

4.3.2 Apklausų komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“ išgauti portalo lankytojų nuomonę tam tikrais trumpais klausimais pateikiant atsakymų variantus.

Taikymai: komponentas buvo pakartotinai panaudotas projektuose „Equal-Greitkelis“ ir „LieDM“ tais pačiais tikslais.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

4.3.3 Forumo komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“ regionų kaimelių gyventojams ir prieigos taškų priežiūros personalo tarpusavio bendravimui, bendrųproblemų sprendimui.

Taikymai: šis komponentas yra taikomas visuose projektuose kaip bendravimo ir dokumentų talpinimo ir aptarimo priemonė. Komponentas buvo atnaujintas galimybe registruotis į dominančias temas ir gauti pranešimus elektroniniu paštu apie naujas žinutes.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

4.3.4 Grupių valdymo pagal tipą komponentas

Sukūrimo tikslas: komponentas buvo sukurtas biomedicinos podiplominių studijų sistemai, kurioje yra keletas skirtingų vartotojų grupių tipų tokių kaip įstaigos, organizacijos,

departamentai. Buvo suprojektuotas pakartotinio panaudojimo komponentas, kurio pagalba konfigūracijos metu nurodomas valdomos grupės tipas (arba kelios), tam tikri papildomi parametrai ir komponentas leidžia valdyti pasirinkto tipo grupių duomenis, vartotojus ir pan.

Taikymai: šis komponentas buvo pritaikytas kelis kartus tame pačiame projekte keturioms skirtingoms funkcijų grupėms realizuoti.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

4.3.5 Kursų katalogo komponentas

Sukūrimo tikslas: kursų katalogo komponentas buvo sukurtas biomedicinos podiplominių studijų sistemai, Kauno medicinos universiteto ir Vilniaus universiteto Medicinos fakulteto organizuojamų kursų pateikimui vartotojų registracijai.

Taikymai: kursų katalogas buvo pakartotinai panaudotas ir adaptuotas projekte „LieDM“. Dėl institucinių skirtumų ir poreikių skirtumų, komponentas turėjo būti smarkiai pakeistas ir papildytas.

Planuojami atnaujinimai: komponentas bus patobulintas pagal projekto „Liedm“ poreikius dėl poreikio apjungti kelias skirtingas sistemas, t.y., registruojant studentus į vieną sistemą, bus inicijuojama registracija ir į kitas kursus teikiančias sistemas.

4.3.6 Meniu komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“, esant poreikiui praplėsti standartinę medžio struktūros tinklalapių navigaciją.

Taikymai: meniu komponentas yra lengvai naudojamas ir labai dažnai pakartotinai pritaikomas skirtinguose projektuose.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

4.3.7 Naujienų komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“ portalo naujienoms skelbti, kaupti, valdyti.

Taikymai: šis komponentas yra plačiai taikomas visuose projektuose skirtingais tikslais: naujienų, renginių, įvykių skelbimui ir kaupimui.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

4.3.8 Paprasto paragrafo komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“ įvairiems tikslams realizuoti: talpinti tekstinę ir vaizdinę informaciją, ją laisvai redaguoti bet kuriuo metu, suteikti galimybę aukštesnio lygio vartotojams talpinti skubius pranešimus, keisti, papildyti tinklalapio duomenis.

Taikymai: paprastas komponentas yra dažniausiai pakartotinai panaudojamas komponentas, poreikis jį naudoti yra kiekviename projekte.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

4.3.9 Paveiksliukų galerijos komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“ įvairių piešinių ir nuotraukų konkursų organizavimui, taip pat renginių nuotraukų talpinimui.

Taikymai: paveiksliukų galerijos komponentas pakartotinai naudojamas visuose projektuose, nes funkcijos, kurias komponentas suteikia yra dažnai projektų reikalavimų specifikacijose.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

4.3.10 Registracijos į pasirinktas vartotojų grupes komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“ konferencijos dalyvių registracijai.

Taikymai: „LieDM“ projekte komponentas tapo kursų katalogo komponento sudėtine dalimi. Projekte „Evete“ komponentas buvo pritaikytas studentų registracijai į mažus mokymosi vienetus, kurso sudėtines dalis.

Planuojami atnaujinimai: komponentas bus papildytas vartotojų importavimo iš bylos galimybe. Kadangi komponentas yra plačiai pritaikomas ir dažnai naudojamas, numatoma, jog ateityje bus dar tobulinamas ir plečiamas.

4.3.11 Vartotojų valdymo pagal asociaciją komponentas

Sukūrimo tikslas: komponentas buvo sukurtas projektui „VIPT (viešieji interneto prieigos taškai)“, kai atsirado poreikis kiekvienai kaimeliui valdyti savo registruotus vartotojus, kiekvienam apskrities koordinatoriui – visų apskrities kaimelių vartotojus, projekto koordinatoriams – visus sistemos vartotojus.

Taikymai: panašus reikalavimas iškilo ir podiplominių studijų informacinėje sistemoje, kur kiekviena institucija gali valdyti savo darbuotojus, kiekvienas departamentas – savo darbuotojus, o kursų teikėjai – visus registruotus vartotojus.

Planuojami atnaujinimai: komponento atnaujinti neplanuojama.

5 MODELIO TRŪKUMŲ IR PRIVALUMŲ APIBENDRINIMAS. IŠVADOS IR REKOMENDACIJOS

Darbe pateikta pakartotinio panaudojimo technologijų, komponentais grįsto programavimo ir metaprojektavimo aspektų apžvalga. Eksperimentinėje dokumento dalyje pristatytas architektūros modelis, kuris realizuotas remiantis paminėtomis koncepcijomis. Realizacijos kūrimo ir pritaikymo metu pirmas žingsnis buvo sukurti „sistemą-sėklą“, kuri vėliau tampa pakartotinio panaudojimo komponentų konteineriu. Turint sistemos branduolį, kuris sukurtas priimti bet kokias funkcijas teikiančius komponentus, suprojektuotus ir suprogramuotus pagal nustatytą standartą, tolimesnis procesas yra pakartotinai pritaikomų elementų bibliotekos kūrimas. Eksperimento metu paaiškėjo, jog kuriant pakartotinio panaudojimo komponentus būtina:

- Numatyti komponento panaudojimo kontekstą (jeigu komponentas yra naudingas tik vienoje vienintelėje sistemos vietoje, gal planuoti apie jos pritaikomumą nėra prasminga);
- Detaliai apsirašyti komponento veiklos taisykles (nes pasitaiko tokių atvejų, kai pats kūrėjas nebeatsimena, kaip komponentas elgiasi su duomenimis, kokia jo paskirtis);
- Aiškiai apibrėžti konfigūruojamų parametrų reikšmę komponentui;
- Aiškiai nustatyti, kokių funkcijų, duomenų ir rezultatų gali tikėtis vartotojas įtraukdamas komponentą į sistemą;
- Žinoti, kad komponentas gali būti vadinamu pakartotinio panaudojimo komponentu tik tada, kai jis yra pritaikytas daugiau nei vieną kartą.

Kadangi šios sistemos modelis jau buvo pritaikytas ne viename projekte, galima išskirti realią pakartotinio panaudojimo komponentų teikiamą naudą:

- Padeda centralizuoti kuriamos sistemos resursus;
- Supaprastina sudėtingos logikos ir atvaizdavimo paketų realizavimo, integravimo ir testavimo procesus;
- Vieną kartą sukūrus komponentą, jis gali būti pritaikytas neribotą skaičių kartų;

- Perkėlus sistemos projektavimo (sistemos navigacija, struktūra, duomenys, nustatymai ir pan.) ir komponentų konfigūravimo atsakomybes skirtingoms galutinių vartotojų grupėms mažinamos programuotojų darbo laiko sąnaudos.

Pakartotinio panaudojimo komponentų architektūros projektavimo ir realizavimo sunkumai:

- Sunku nustatyti ir numatyti optimalų komponento universalumo lygį;
- Kartais kas tinka viskam, iš tiesų netinka niekam, todėl pakartotinis komponento teisingas panaudojimas yra įmanomas tik tada, jei sukurtas komponentas nors kartą jau buvo sėkmingai panaudotas;
- Metaprojektavimu grįstos sistemos gali būti aktyviai naudojamos ir plečiamos tik tuo atveju, kai galutinių vartotojų grupė yra socialiai aktyvi ir techniškai kompetentinga.

Tolimesniems modelio vystymams praverstų tokios rekomendacijos:

- Komponentų kūrimo standartai turi būti aprašyti ir publikuoti visiems kūrėjų komandos nariams;
- Specifiniai sistemos komponentai sukurti specifiniams projektams turėtų turėti atskirą talpyklą;
- Perduodant atsakomybes galutiniam vartotojui labai svarbu jam suteikti ne tik priemones kaip jomis naudotis, bet labai svarbūs vartotojo vadovo dokumentai, nuolatinė parama ir konsultacijos.
- Galutinio vartotojo motyvacijos skatinimas prisiimti projektuotojo atsakomybes sistemoje.

6 Literatūra

1. W. B. Frakes, K. Kang, Software Reuse Research: Status and Future, 2005.
2. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Code_reuse [2007.05.14].
3. Apache Software Foundation, Wicket. Prieiga internete: http://www.wicketframework.org/ [2007.02.11].
4. Sun Developer Network, JavaServer Faces Technology. Prieiga internete: http://java.sun.com/javaee/javaserverfaces/ [2007.01.02].
5. Apache Software Foundation, Tapestry. Prieiga internete: http://tapestry.apache.org/ [2007.01.11].
6. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://lt.wikipedia.org/wiki/Java_%28kalba%29 [2007.01.11].
7. Wikimedia Foundation, Inc., Swing (Java). Prieiga internete: http://en.wikipedia.org/wiki/Swing_(Java) [2007.01.11].
8. NextApp, Inc., Echo (version 1.x). Prieiga internete: http://www.nextapp.com/platform/echo1/echo/ [2007.01.11].
9. B. Kurniawan, Introducing JavaServer Faces. Prieiga internete: http://www.onjava.com/pub/a/onjava/2003/07/30/jsf_intro.html [2007.01.11].
10. Apache Software Foundation, Wicket. Prieiga internete: http://www.wicketframework.org/ [2007.01.02].
11. M. Raibe, Comparing web frameworks: Struts, Spring MVC, WebWork, Tapestry & JSF. Prieiga internete: http://www.chariotsolutions.com/slides/spring-forward-2006-web-frameworks.pdf [2007.01.11].
12. K. Pitty, Why JavaServer Faces?. Prieiga internete: http://squizlog.keithpitty.org/archives/000447.html [2007.01.11].

13. Virtua, Inc., JSFcentral community. Prieiga internete: http://www.jsfcentral.com/ [2007.01.11].

14. E. Burns, J. Hookom, A. Winer, Evolving JavaServer™ Faces Technology: AJAX Done Right, 2006. Prieiga internete: http://developers.sun.com/learning/javaoneonline/2006/webtier/TS-1161.pdf [2007.01.11].

15. P. Zoio, JavaServer Faces vs Tapestry, A Head-to-head comparison, 2005. Prieiga internete: http://www.theserverside.com/tt/articles/article.tss?l=JSFTapestry [2007.01.11].

16. ServerSide JAVA Symposium, Virtuas, Java Web Framework: Sweet Spots. Prieiga internete: http://www.virtuas.com/files/JavaWebFrameworkSweetSpots.pdf [2007.01.11].

17. M. Smolarova, P. Navrat, Slovak University of Technology, Bratislava, Slovakia, Software Reuse: Principles, Patterns, Prospects, Journal of Computing and Information Technology, 1997.

18. J. Sodhi, P. Sodhi, Software Reuse: Domain Analysis and Design Process, Computing McGraw-Hill, 1999.

19. B. H. C. Cheng, J. J. Jeng, Formal methods applied to reuse. In 5th Annual Workshop On Software Reuse WISR'92, 1992.

20. H. Milli, F. Milli, A. Milli, Reusing software: Issues and research directions. IEEE Trans. On Software Engineering. 21 (6): 529-561, 1995.

21. B. Meyer. Object-Oriented Software Construction. Prentice Hall, 1988.

22. R. Prieto-Diaz. Status report: Software reusability. IEEE Software, 10 (3): 61-66, 1993.

23. W. Tracz, Third International Conference on Software Reuse. Summary, ACM SIGSOFT, 20 (2):21-25, 1995.

24. R. Prieto Diaz. Making software reuse work: An implementation model, ACM Sigsoft, 16(3): 61-68, 1991.

25. G. Fischer, E. Giaccardi, Y. Ye, A.G. Sutcliffe, N. Mehandjiev. Meta-Design: A Manifesto for End-User Development, Communications of ACM (A Special issue on End User Development), 2004.

26. G. Fischer. Meta-Design: Expanding Boundaries and Redistributing Control in Design, Proceedings of the Interact'2007 Conference, Rio de Janeiro, Brazil, September (in press), 2007.

27. N.Bevan. Quality in use for all. User interfaces for all. Stephanidis, C (ed), Lawrence Erlbaum, 1999. Prieiga internete: http://www.usabilitynet.org/papers/quality_in_use_for_all.pdf [2007.01.20].

28. G.Gledec. Evaluating Web Site Quality. 2005. Prieiga internete: http://www.carnet.hr/CUC/program/papers/abs/b1_gledec_abs.pdf

29. J.Offutt. Web Software Applications Quality Attributes. Quality Engineering in Software Technology (CONQUEST 2002), pages 187-198, Nuremberg, Germany, September 2002. Prieiga internete: http://ise.gmu.edu/faculty/ofut/rsrch/papers/conquest02.pdf

30. E. S. Raymond, B. Young. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly & Associates, Sebastopol,CA, 2001.

31. Wikipedia. The Free Encyclopedia, 2006. Prieiga internete: http://wikipedia.org/

32. M. Wright, M. Marlino, T. Sumner. Meta-Design of a Community Digital Library, D-Lib Magazine, Volume 8, Number 5, 2002. Prieiga internete: http://www.dlib.org/dlib/may02/wright/05wright.html

33. C. Scaffidi, M. Shaw, B. Myers. Estimating the Numbers of End Users and End User Programmers. In Proceedings of 2005 IEEE Symposiumon Visual Languages and Human-Centric Computing (VL/HCC'05), Dallas,Texas, September, 2005, pp. 207-214.

34. G. Fischer, E. Giaccardi. Meta-Design: A Framework for the Future of End User Development. In H. Lieberman, F. Paternò, & V. Wulf (Eds.), End User Development: Empowering People to Flexibly Employ Advanced Information and Communication Technology, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2006. pp. 427-457.

35. A. Henderson, M. Kyng. There's No Place Like Home: Continuing Design in Use. In J. Greenbaum, & M. Kyng (Eds.), Design at Work: Cooperative Design of Computer Systems, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1991, pp. 219-240.

36. S. Brand. How Buildings Learn: What Happens After They're Built, Penguin Books, New York, 1995.

37. G. Fischer. Beyond 'Couch Potatoes': From Consumers to Designers and Active Contributors, in FirstMonday (Peer-Reviewed Journal on the Internet), 2002. Prieiga internete: http://firstmonday.org/issues/issue7_12/fischer/ .

38. E. Giaccardi, G. Fischer. Creativity and Evolution: A Metadesign Perspective, Sixth International Conference of the European Academy of Design (EAD06), 2006.

39. A. C. Wills. Component Based Development, TriReme International Ltd, 1999. Prieiga internete: http://www.trireme.com .

40. A. W. Brown. Large-Scale Component-Based Development, Upper Saddle River, NJ: Prentice Hall, 2000.

41. I. Crnkovic, M. Larsson. A Case Study: Demands on Component-Based Development,. Proc. 22nd Int. Conf. Software Engineering, Limerick, Ireland, ACM Press, 2000.

42. C. Szyperski. Component Software Beyond Object-Oriented Programming, Reading, MA: Addison-Wesley, 1998, pp. 46.56.

43. N. A. Maiden, C. Ncube. Acquiring COTS Software Selection Requirements,. IEEE Software, Vol. 15, No. 2, 1998, pp. 46.56.

44. A. V. Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective, Proc. 22nd Int. Conf. Software Engineering, Limerick, Ireland, ACM Press, 2000.

45. M. Larsson. Applying Configuration Management Techniques to Component-Based Systems. Licentiate Thesis Dissertation 2000-007, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2000.

46. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Waterfall_model [2007.05.21].

47. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Agile_software_development [2007.05.21].

48. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Extreme_programming [2007.05.21].

49. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Metamodel [2007.05.21].

50. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Metadata [2007.05.21].

51. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Metaanalysis [2007.05.21].

52. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/Metalanguage [2007.05.21].

53. Vikipedija, Laisvoji enciklopedija. Prieiga internete: http://en.wikipedia.org/wiki/End_user [2007.05.23].