KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Kazys Račkauskas

Automatizuoto grafinės vartotojo sąsajos mobiliuose įrenginiuose testavimo tyrimas

Magistro darbas

Vadovas: prof. E. Bareiša

KAUNAS, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Kazys Račkauskas

Automatizuoto grafinės vartotojo sąsajos mobiliuose įrenginiuose testavimo tyrimas

Magistro darbas

prof. E. Bareiša

2007-05-25

Recenzentas

Doc. D. Rubliauskas

Atliko

IFM 1/2 gr. stud.

K. Račkauskas

2007-05-25

KAUNAS, 2007

Research in Graphical User Interface Automated Testing for Mobile devices

SUMMARY

Mobile devices such as cell phones and personal digital assistance are widely used in new software products. Testing takes important place in software development process. Constraints of mobile devices (speed, amount of memory, energy, small screen, wide range of platforms) raise new problems for software development process including testing phase. Automated approach of software testing reduces testing time and increases testing range. It is important to distinguish graphic user interface as a special part of testing. The important part of testing is specification based testing. The aim of these master theses is to analyze automated testing of GUI for mobile devices, define testing tasks and enhance mobile device testing framework by providing means for test case generation from specification.

Automatizuoto grafinės vartotojo sąsajos mobiliuose įranginiuose testavimo tyrimas

SANTRAUKA

Naujuose programiniuose produktuose plačiai taikoma mobilioji įranga (mobilieji telefonai, delniniai kompiuteriai). Programinės įrangos kūrimo procese svarbią vietą užima testavimas. Dabartinių mobiliųjų įranginių apribojimai (darbo sparta, atminties kiekis, energija, ekrano dydis, platformų įvairumas) kelia naujas problemas programinės įrangos kūrimo procesui, tame tarpe ir testavimui. Testavimo proceso automatizavimas leidžia sumažinti bandymų trukmę, padidinti testavimo darbų apimtį. Programinės įrangos testavime mobiliems įrenginiams svarbu išskirti grafinės vartotojo sąsajos testavimą. Svarbi testavimo dalis – testavimas pagal specifikaciją.

Šiame darbe nagrinėjamas grafinės vartotojo sąsajos automatizuotas testavimas mobiliai įrangai, nustatomi tikslai, galimi keliai jiems pasiekti bei galimas testavimo aplinkos praplėtimas testų generavimui pagal specifikaciją.

TURINYS

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

ĮVADAS

Testavimas užima didelę dalį programos kūrimo cikle įdėtų pastangų, todėl suprantami ir šios veiklos nemaži kaštai. Nors pažangūs procesai ir įrankiai padėjo organizacijoms sumažinti programinių sistemų kūrimo laiką, tačiau dar nesugebėjo žymiai sumažinti laiko reikalingo šiom programom ištestuoti [1].

Naujuose programiniuose produktuose plačiai taikoma mobilioji įrangą (mobilieji telefonai, delniniai kompiuteriai). Programinės įrangos kūrimo procese svarbią vietą užima testavimas. Dabartinių mobiliųjų įranginių apribojimai (darbo sparta, atminties kiekis, energija, ekrano dydis, platformų įvairumas) kelia naujas problemas programinės įrangos kūrimo procesui, taip pat ir testavimui. Testavimo proceso automatizavimas leidžia sumažinti bandymų trukmę, padidinti testavimo darbų apimtį. Programinės įrangos testavime mobiliems įrenginiams svarbu išskirti grafinės vartotojo sąsajos testavimą.

Dažnai kuriama programa mobiliai įrangai yra orientuota į taikymą daugelyje mobiliųjų platformų. Tačiau plati mobiliųjų įrenginių įvairovė kelia tam tikrų problemų. Todėl programos kūrėjams tenka atsižvelgti į mobiliųjų įrenginių architektūros bei konfigūracijos skirtumus. Mobiliųjų įrenginių skirtumai komplikuoja grafinės vartotojo sąsajos testavimą. Viename įrenginyje puikiai veikianti vartotojo sąsaja gali būti visiškai nepriimtina kitame įrenginyje.

Kuriant bei testuojant programinę įranga naudojami mobiliųjų įrenginių emuliatoriai, kurie neperteikia visų mobiliųjų įrenginių galimybių, apribojimų bei darbo ypatumų. Todėl yra svarbu išbandyti programos grafinę vartotojo sąsają tiesiogiai mobiliajame įrenginyje ar skirtingų mobiliųjų įrenginių grupėje. Taigi vartotojo sąsaja turi atitikti specifikaciją visuose įranginiuose. Todėl ir automatinio testavimo įrankiai turėtų atlikti testavimą pagal specifikaciją.

Šiame darbe nagrinėjamas mobilios įrangos automatizuotas grafinės vartotojo sąsajos, grafinės vartotojo sąsajos specifikavimo būdai, nustatomi tikslai bei galimi keliai jiems pasiekt

1. AUTOMATIZUOTO GRAFINĖS VARTOTOJO SĄSAJOS MOBILIUOSE ĮRANGINIUOSE TESTAVIMO ANALIZĖ

1.1. Apibrėžimai

Grafinė vartotojo sąsaja – tai turinti hierarchinę struktūrą, grafiškai atvaizduota programos dalis, kuri priima vartotojo ir/arba sistemos sukeltus įvykius (*generated events*) iš fiksuotos įvykių aibės ir pateikia apibrėžtą (*determined*) grafinį rezultatą. Grafinę vartotojo sąsaja sudaro objektai – grafiniai komponentai. Kiekvienas grafinis komponentas turi fiksuotą savybių aibę. Kiekvienu laiko momentu grafinio komponento savybės turi diskrečią reikšmę. Komponentų savybių reikšmių aibė apibrėžia grafinės sąsajos būseną [3, 16].

Mobilieji įranginiai – tai ypatingai portatyvūs, savarankiškai apdorojantys informaciją bei naudojantys bevieles tinklo ryšio technologijas duomenų mainams įrenginiai. Įrenginys turi būti pakankamai mažas, kad tilptų žmogaus delne [17, 18]. Mobilusis įrenginys turi atitikti šiems reikalavimams:

- Kabeliai prie įrenginio jungiami tik pasikrovimui ir sinchronizacijai su kompiuteriu. Kitais atvejais darbas vyksta be kabelių.
- Įrenginys valdymas atliekamas viena ranka. Darbui su įrenginiu, jį nebūtina išdėstyti ant stalo.
- Įrenginys turi palaikyti bevielio tinklo ryšio technologiją arba jų grupę (Wi-Fi, WAP, IrDA, Bluetooth)

Testavimas – procesas skirtas nustatyti programinės įrangos teisingumą, pilnumą, saugumą ir kokybę [5].

Testavimo automatizavimas – programinės įrangos naudojimas testų vykdymui, gautos ir spėjamos reikšmės sulyginimui, prieš sąlygų nustatymui bei kitoms testavimo valdymo funkcijoms [6].

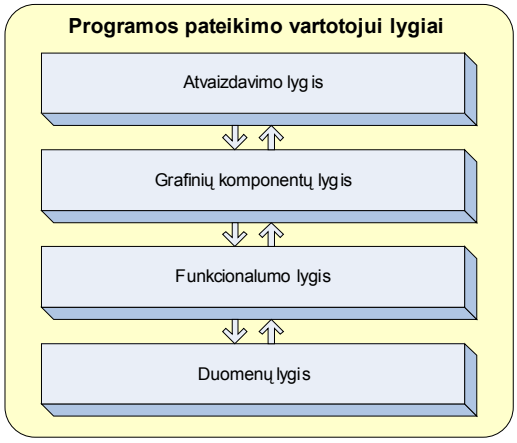
Modeliu pagrystąs testavimas – programinės įrangos testavimas testiniais atvejais gautais iš modelio kuris visiškai arba dalinai apibūdina testuojamą sistemą [7].

1.2. Grafinė vartotojo sąsaja ir jos komponentai

Egzistuoja keli požiūriai į grafinę vartotojo sąsają. Galima interpretuoti programos grafinę sąsaja kaip grafinį vaizdą, neišskiriant jame grafinių komponentų. Tačiau objektiškai orientuotame programavime turėtume grafinę vartotojo sąsają apibūdinti kaip hierarchiškai

išdėstytus grafinės vartotojo sąsajos komponentus (žiūrėti 1.1 pav.). Išskirsime testavimo įrankių ir programos grafinės sąsajos sąveikavimo lygmenys:

- Atvaizdavimo lygis – programos pateikimas ir interpretavimas kaip grafinio atvaizdavimo, neišskiriant jame grafinių komponentų.
- Grafinių komponentų lygis – iš programos grafinės sąsajos išskiriami grafiniai komponentai. Analizuojamos grafinių komponentų savybės.
- Funkcionalumo lygis – programos grafinė sąsaja analizuojama ne tik kaip grafinių komponentų aibė, bet ir kaip sąsaja, kurios pagalba galima kviesti bei vykdyti testuojamos programos funkcionalumą. Šiame lygyje gali būti vykdomas integravimo testavimas.
- Duomenų lygis – šiame lygyje sekamas ne tik grafinės vartotojo sąsajos kviečiamas bei vykdomas funkcionalumas, taip pat yra atliekamas ir vienetų testavimas.



1.1 pav. Programos pateikimo vartotojui lygiai

Nors delniniai/mobilieji įrenginiai skiriasi savo išvaizda bei forma, tačiau visų įrenginių vartotojo sąsajai yra būdingi bendri grafinės vartotojo sąsajos komponentai.

Reikalavimai grafiniams komponentams

Grafiniams komponentams taikomi šie reikalavimai:

- Grįžtamasis ryšys (*feedback*) – vartotojo informavimas apie naudojamus komponentus. Tam puikiai tinka spalvos pakeitimas, paryškinimas bei formos pasikeitimas.
- Pavadinimas (*naming*) – komponentu pavadinimai turi tiksliai atspindėti komponento veiksmą arba savybę. Reikia vengti šabloniniu pavadinimu, tokių

kaip „gerai“, „atlikti“, „taip“, „ne“. Iš komponento pavadinimo vartotojas turi suprasti, kokia yra komponento paskirtis. Taip pat į pavadinimus galima įtraukti daugtaškius „...“ kurie intuityviai informuoja vartotoją, kad suaktyvinus komponentą, jam bus pateiktas dialogas.

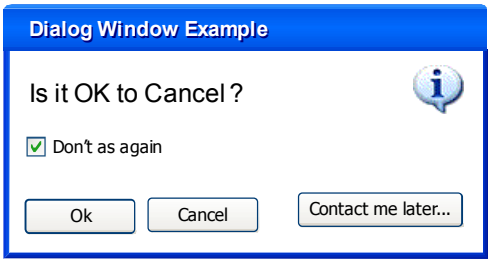
- Išdėstymas (*placement*) – svarbi yra komponentų išdėstymo ekrane tvarka. Numatyti komponentai turi būti pateikiami kairiau kitų komponentų. Mygtukai turi būti patiekiami lango apačioje, kadangi jų išdėstymas lango viršuje „paslėps“ nuo vartotojo lango turinį.
- Būsenos (*state*) – komponentų būsenų sekimas yra svarbus, siekiant efektyviai organizuoti grįžtamąjį ryšį su vartotoju. Komponentai, kurių panaudojimas operacijoje nėra leidžiamas ar apibrėžtas, turi būti uždrausti.

Konstrukcijos

Grafinės sąsajos konstrukcijos – tai grafinių komponentų rinkiniai. Panašiai kaip sakiniai sudaromi iš žodžių, grafinės konstrukcijos sudaromos iš grafinių komponentų. Apžvelgsime keletą pagrindinių grafinių konstrukcijų [18].

Dialogai

Dialogas – tai programos langas vaizduojamas programos viršuje (1.2 pav.). Šis langas iškviečiamas, kai reikalingas vartotojo dėmesys. Dialogo langų paskirtis yra pranešti vartotojui apie klaidą, informuoti apie programos būseną, užklausti vartotoją dėl tolimesnių nurodymų. Dažnai dialogo langas pateikia vartotojui du mygtukus, kurių pagalba jis gali sutikti su pasiūlymu arba jį atmesti. Šie mygtukai standartiškai yra [Gerai] (*OK*) ir [Atmesti] (*Cancel*). Tačiau ypač mobiliuose platformose, nerekomenduojama mygtukų suteikti šabloninius pavadinimus. Mygtukų pavadinimai turi atskleisti vartotojui savo funkcionalumą. Kadangi kartais programos užduodamas klausimas negali būti vienareikšmiškai interpretuotas, ir vartotojui bus neaišku, ką reikš atsakymas „Gerai“.



1.2 pav. Dialogo langas

Formos, Langai, Vedliai

Formos tai aibė teksto įvedimo bei atvaizdavimo laukų. Dažnai formos laukai yra grupuojami atskiromis kortelėmis. Siekiant patogesnio duomenų įvedimo, šitas procesas atliekamas vedlių (*wizards*). Kiekviename vedlio žingsnyje vartotojas įvedinėja tik vienos grupės reikšmes. Tačiau vedlys taip pat gali būti perteklinis ir nepatogus vartojime. Tai nutinka kai vedlio žingsnių yra daug, o vartotojui tenka įvesti tik keletą reikšmių.

Programose mobiliesiems įrenginiams duomenų įvedimas dažnai atliekamas vedlio. Taip yra todėl, kad į mažą mobilaus įrenginio ekraną netelpa visi reikalingi įvedimo laukai. Taip pat reikalavimai darbui su mobiliu įrenginių įneša savo apribojimus – vartotojas turi sugebėti atlikti darbą viena ranka [10]. Vedlys yra realizuojamas kaip programos langų seką.

Iškarpinė

Delniniai įrenginiai ir pranešėjai turi iškarpinės savybę (*clipboard*), kuri leidžia atlikti duomenų kopijavimo (*copy*), iškirpimo (*cut*) ir įterpimo (*paste*) operacijas. Iškarpinė yra paranki priemonė duomenų mainams tarp įvairių programų. Mobilieji telefonai neturi tiesioginio iškarpinės palaikymo. Mobilieji telefonai taip pat nepalaiko programų keitimo (*switching*).

1.3. Grafinės vartotojo sąsajos testavimas

Programinės įrangos kūrimo išlaidų mažinimas ir programinės įrangos kokybės gerinimas yra svarbūs programinės įrangos pramones uždaviniai. Todėl programų gamintojams yra svarbu turėti automatizuotus testavimo įrankius. Šiuolaikiniai automatizuoti testavimo įrankiai leidžia kurti testavimo skriptus, generuoti testavimo atvejus, atlikti vienetų testavimą, baltos ir juodos dėžės testavimą. Šių įrankių paskirtis yra pagreitinti programinės įrangos kūrimo procesą, aptikti daugiau defektų bei sumažinti programų kūrimo kaštus atliekant regresinį testavimą. Įprastai testavimo skriptai sudaromi rankiniu būdu arba naudojant įrašymo procedūrą. Testavimo skriptų sudarymas rankiniu būdu ir/arba įrašymas tai pat gali įnešti klaidų.

Grafinės sąsajos testavimas panašus į komponentų testavimą. Siekiant automatizuoti grafinės sąsajos testavimą, reikia remtis tokia pat logika kaip ir testuojant komponentus: t. y. Automatizuotam testavimo įrankiui pateikiama testuojama programa, gaunamas testavimo rezultatas – defektų sąrašas.

Šiuolaikinių programų vartotojai kviečia programos funkcionalumą per grafinę vartotojo sąsają. Tokiu būdu grafinės vartotojo sąsajos testavimas gali padengti programą. Programos mobiliajai įrangai taip pat pateikia vartotojui grafinę sąsają. Tačiau mobilieji įrenginiai įneša į grafinę sąsają tam tikrus apribojimus, su kuriais nesusiduria stacionarių kompiuterių vartotojai. Todėl grafinės sąsajos testavimas mobiliai įrangai yra sudėtingesnis nei stacionariems kompiuteriams.

Rankinis testavimas nėra efektyvus. Testavimo automatizavimas leidžia ženkliai padidinti testų skaičių. Tuo pačiu padidėjęs testų skaičius nereikalauja didesnių testavimo resursų.

Testuojant programas automatizuotu būdu, testuotojai naudoja skriptus su komandinėm eilutėm. Tokiu būdu galima patikrinti programos atskirų dalių funkcionalumo teisingumą, tačiau testavimas yra visiškai atskirtas nuo grafinės sąsajos. Dabartiniu metu yra paplitęs įrašymo-atkartojimo (*capture/playback*) grafinės sąsajos testavimo būdas. Pirmą kartą testavimas atliekamas rankiniu būdu. Testavimo aplinka įrašo testuotojo atliekamus veiksmus grafinėje sąsajoje: teksto įvedimą iš klaviatūros, palės klavišų paspaudimus bei paspaudimo koordinates. Vėliau, atliekant automatizuotą testavimą, šie įvykiai yra atkartojami. Taip atliekamas regresinis testavimas. Tačiau įrašymo-atkartojimo metodą taikantys testavimo įrankiai nesugeba atpažinti grafinių komponentų (žiūrėkite skyrių 1.2). Dabartinės grafinės sąsajos technologijos reikalauja rankinio testavimo skriptų sudarymo, redagavimo bei rezultatų analizavimo. Iš vienos pusės, grafinės sąsajos testavimo įrankiai suteikia vartotojas galingas priemones atlikti testavimą. Iš kitos pusės, testavimo aplinkos paruošimas reikalauja didelių pastangų [19, 20, 21, 22, 23].

Grafinės vartotojo sąsajos testavimas yra gyvybiškai svarbus todėl, kad jis vykdomas taikant galinio vartotojo požiūrį į programą. Tuo pačiu jis padengia programą, kadangi programos funkcionalumas atvaizduojamas ir valdomas iš grafinės vartotojo sąsajos.

Automatizuotas testavimas leidžia smarkiai sumažinti testavimo kaštus. Taip pat automatizavimas leidžia reguliariai atlikti, todėl defektai aptinkami anksčiau.

Egzistuoja keletas įrankių, skirtų grafinei vartotojo sąsajai testuoti. Tačiau įrankiai smarkiai priklausomi nuo platformos (šiuo metu dominuojanti stacionarių kompiuterių platforma yra Microsoft Windows). Pritaikyti vienai platformai, jie netinka kitoms. Dažniausiai įrankiai veikia skirtingose aplinkose ir naudojasi skirtingais skriptais. Testavimo skriptas, įrašytas vienoje platformoje, negali būti atkartotas kitoje platformoje. Taip pat skriptas, įrašytas vieno įrankio pagalba, negali būti atkartotas kitame įrankyje [25].

1.4. Esamų grafinės vartotojo sąsajos testavimo modelių įrankių ir metodų apžvalga

Yra keletas modelių kuriais norima automatizuoti tam tikrus grafinės vartotojo sąsajos testavimo aspektus.

1.4.1. Baigtiniais automatais pagrįstas testavimas

Automatizuotame testavime daugiausiai siūloma naudoti baigtinio automato medelius [1, 25, 26, 27, 28]. Pagrindinė idėja yra atvaizduoti vartotojo sąsajos elgseną kaip baigtinį automatą. Kiekvienas grafinės vartotojo sąsajos įvykis sukelia perėjimą iš vienos būsenos į kitą baigtiniame automate. Pereitų briaunų seka baigtiniame automate atvaizduoja testavimo atvejį. Baigtinio automato būsenos, testinių atvejų vykdymo metu, gali būti panaudotos vartotojo sąsajos būsenos verifikavimui. Tačiau didelėms grafinėms vartotojo sąsajoms tokie baigtiniai automatai turi išplečiamumo problemų. Buvo sukurtas vadinamas kintamas baigtinis automatas [29] kuriame įvedant kintamuosius į modelį yra sumažinamas abstrakčių būsenų skaičius. Praktiškai šie modeliai yra kuriami rankiniu būdu. Kai šie modeliai yra naudojami verifikavimui, tai konkrečių grafinės vartotojo sąsajos būsenų susiejimas su baigtinio automato būsenomis taip pat atliekamas rankiniu būdu.

1.4.2. Sąsajos dalinimas pagal užduotis

Kiti tyrėjai grafinės vartotojo sąsajos testavimo atvejų generavimui pristato baigtinio automato modelį, kuriame galimos būsenos, priklausomai nuo vartotojo atliekamų užduočių, suskirstomos į skirtingus baigtinius automatus [14][15]. Testų kūrėjas nustato užduotis kurias galima atlikti grafine vartotojo sąsaja ir kiekvienai tokiai užduočiai nustato taip vadinamą „pilnos sąveikos sekos“ (PSS) baigtinio automato modelį. Šitas PSS yra naudojamas testavimo atvejų generavimui. Šio modelio pagalba grafinę vartotojo sąsają sėkmingai suskaldoma į atskirus funkcinius vienetus, kuriuos galima testuoti atskirai. Tačiau naudojant šį metodą, baigtinio automato modelio kūrimui, reikia įdėti labai daug rankinio darbo.

1.4.3. Vartotojo elgsenos modelis

Testinių atvejų generavimui taip pat yra naudojami vartotojo elgsenos modeliai. Iš šių modelių geriausiai žinomas „pradedančio vartotojo“ modelis [29]. Šio metodo metu, testų kūrėjas pirmiausia rankiniu būdu sugeneruoja grafinės vartotojo sąsajos įvykių seką. Tada naudojant tam tikrą techniką sukurta seka yra modifikuojama ir pratęsiama, taip pamėgdžiojant nepatyrusį vartotoją [30, 31]. Tai yra grindžiama tuo kad naujas vartotojas

dažniausiai iš nepatyrimo pasirenka ne tiesiausius kelius užduočiai atlikti. Ši technika reikalauja nemažai rankinio darbo ir negali būti naudojama kitų tipų testavimo atvejams generuoti.

1.4.4. Įrašymo ir atkartojimo įrankiai

Visi aukščiau paminėti metodai sukurti turint omenį tik vieną automatizuoto grafinės vartotojo sąsajos testavimo aspektą. Šiems metodams trūksta pritaikomumo kitiems testavimo aspektams. Be to jos reikalauja nemažai rankinio darbo. Beje, remiantis šiais metodais nėra išleisto nei vieno komercinio produkto.

Dabar praktiškai naudojamos grafinės vartotojo sąsajos testavimo priemonės yra nepilnos, sukurtos kažkokiam konkrečiam tikslui, reikalaujančios daug rankinio darbo. Daugiausiai naudojami įrašymo/atkartojimo įrankiai, tokie kaip WinRunner, kurie suteikia labai mažai automatiškumo. Tai ypač pasireiškia kuriant testavimo atvejus, testavimo orakulus ir norint įvertinti testavimo padengimą. Testuotojas šiuos įrankius panaudoja per dvi fazes: įrašymo, o vėliau pakartojimo. Įrašyti testavimo atvejai gali vėliau būti automatiškai atkartoti modifikuotoje programos versijoje naudojant testavimo įrankio atkartojimo dalį. Kaip galima pastebėti šie įrankiai reikalauja labai daug rankinio darbo. Vystant vartotojo sąsają sukurti scenarijai nebetinka ir turi būti išmesti bei kuriami nauji.

1.4.5. Programuojami testavimo atvejai

Kiti grafinės sąsajos testavimo įrankiai reikalauja kiekvieno testavimo atvejo užprogramavimo. Tai tokie įrankiai kaip JFCUnit, Abbot, Pounder ar Jemmy Module. Testuotojas naudodamas šiuos įrankius turi užprogramuoti ne tik testavimo atvejus bet ir vartotojo sąsajos elgseną kurios tikisi.

1.5. Automatizuoto testavimo aplinkos

Šiame skyrelyje apžvelgsiu Andrej Ušaniov magistriniame darbe pristatytą testavimo aplinką [32]. Šioje aplinkoje sprendžiamos tokios problemos:

- Testavimų atvejų talpinimas ribotoje mobilaus įrenginio atmintyje
- Kodo perkodavimo problema. Neleidžiama pagal suprojektuotos klasės pavadinimą sukurti objekto, todėl negalima pasinaudoti atspindžio (*reflection*)[33] technologija.

Automatizuotam testų atlikimui siūloma naudoti specialią aplinką (*framework*) [34]. Testavimo aplinka atsakinga už testuojamos programos metodų kvietimą. Pagal pateiktą

navigacijos grafą bei scenarijus testavimo aplinka atlikinėja perėjimus tarp langų, perduodama testinius duomenis programai bei fiksuodama rezultatus.

Sprendimo būdas

Išorinėje duomenų bazėje yra saugomi programos navigacijos grafas, programos langų aprašymas, testavimo atvejai bei testavimo rezultatų istorija. Išorinės duomenų bazės panaudojimas leidžia išspręsti mobilaus įrenginio riboto atminties kiekio problemą. Iš duomenų saugyklos į testavimo aplinka perduodamas navigacijos grafas bei scenarijai. Kiekvienam navigacijos mazgui perduodami testavimo atvejų duomenys.

Atspindžio problema sprendžiama taikant programinius interfeisus (API). Šių interfeisų pagalba testavimo aplinkoje bus pasiekiami visi programos lango duomenų įvedimo/išvedimo laukai, jų reikšmės bei komandiniai mygtukai. Per programinę sąsaja gauti lango parametrai lyginami su suprojektuotais. Tokiu būdu bus nustatoma ar langas atitinka grafinės vartotojo sąsajos reikalavimus [35].

Automatizuoto testavimo aplinka – tai savarankiška programa, veikianti mobiliajame įrenginyje. Testavimo aplinka naudoja išorinę duomenų saugyklą testavimo duomenims gauti. Pagal paduotus duomenis: navigacijos grafą, scenarijus ir testinius duomenis, API pagalba kviečiami testuojamosios programinės įrangos metodai. Testavimo aplinka valdys duomenų perdavimą bei fiksuos rezultatus.

Atliekamas kiekvieno vaizduojamo lango bandymas. Tikrinami duomenų įvedimo laukai, leistinos įvesties simbolių aibės. Nustatoma ar visi laukai yra vaizduojami ekrane.

Pagal scenarijų bandoma atlikti perėjimą iš vieno lango į kitą, testavimo aplinka fiksuoja perduotus į metodą duomenis bei gautus rezultatus: loginės operacijos reikšmė bei atvaizduotas langas. Taip pat testavimo aplinka fiksuoja vėlinimą: laiko trukmė per kuria programa reaguoja į vartotojo veiksmus. Gauti rezultatai perduodami į testavimo orakului. Testavimo orakulas atlieka gautų bei laukiamų rezultatų palyginimą.

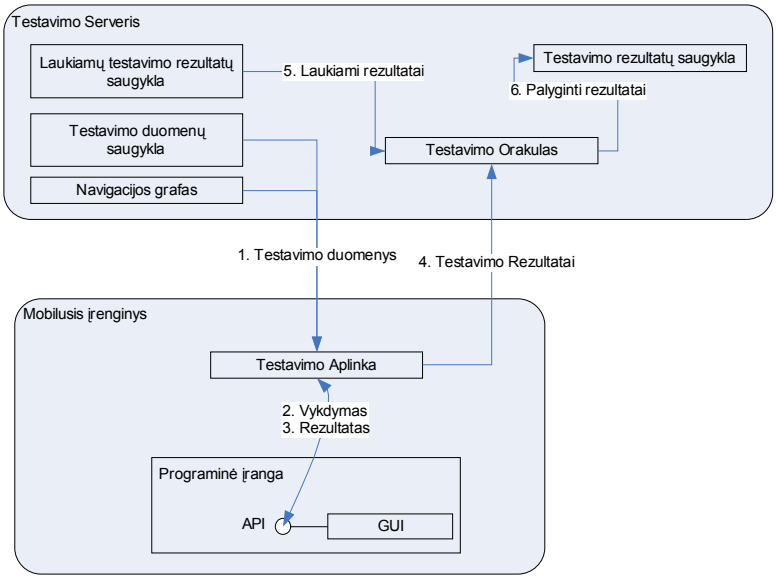
Testavimo aplinkos veikimas

Testavimo aplinka (*framework*) yra paskirstyta tarp mobilaus įrenginio ir serverio (1.3 pav.). Serverio duomenų saugykloje kaupiami visi testavimo duomenys: navigacijos grafas, testavimo scenarijai. Duomenų atskyrimas nuo mobilaus įrenginio atliekamas dėl kelių priežasčių:

- Mobilaus įrenginio atmintis yra ribota
- Keli mobilūs įrenginiai testuojami pagal bendrus scenarijus

- Kelių įrenginių testavimo rezultatų lyginimas

Mobiliajame įrenginyje veikianti testavimo aplinkos dalis priima testavimo duomenis iš serverio. Testavimo duomenys siunčiami dalimis. Pagal priimtus duomenis vykdomas testavimas.



1.3 pav. Testavimo aplinkos struktūrinė schema

Testavimo duomenų aprašymas

Žemiau pateiktos visos navigacijos grafo ir operacijų scenarijuose naudojamos XML etiketės (žiūrėti 1.1 lentelę).

Lentelė 1.1 Navigacijos grafo ir operacijų scenarijų aprašymo XML etiketės

	Grupė	**Pavadinimas**	**Pradžios etiketė**	**Pabaigos etiketė**
1.	Grafas	Grafas	`<graph>`	`</graph>`
2.		Pavadinimas	`<graphname>`	`</graphname>`
3.		Langų sąrašas	`<nodelist>`	`</nodelist>`
4.	Langas	Langas	`<node>`	`</node>`
5.		ID	`<nodeid>`	`</nodeid>`
6.		Pavadinimas	`<nodename>`	`</nodename>`
7.		Būsenų sąrašas	`<statelist>`	`</statelist>`
8.		Elementų sąrašas	`<itemlist>`	`</itemlist>`
9.		Komandų sąrašas	`<cmdlist>`	`</cmdlist>`
10.	Būsena	Būsena	`<state>`	`</state>`

	Grupė	Pavadinimas	Pradžios etiketė	Pabaigos etiketė
11.		ID	<stateid>	</stateid>
12.		Pavadinimas	<statename>	</statename>
13.		Aprašymas	<description>	</description>
14.	Elementas	Elementas	<item>	</item>
15.		ID	<itemid>	</itemid>
16.		Pavadinimas	<itemname>	</itemname>
17.		Tipas	<type>	</type>
18.		Numatyta reikšmė	<defaultvalue>	</defaultvalue>
19.		Apribojimų sąrašas	<constraintlist>	</constraintlist>
20.	Apribojimas	Apribojimas	<constraint>	</constraint>
21.		ID	<constraintid>	</constraintid>
22.		Tipas	<type>	</type>
23.		Reikšmė	<value>	</value>
24.	Komanda	Komanda	<cmd>	</cmd>
25.		ID	<cmdid>	</cmdid>
26.		Pavadinimas	<cmdname>	</cmdname>
27.		Parametrų sąrašas	<paramlist>	</paramlist>
28.	Parametras	Parametras	<param>	</param>
29.		ID	<paramid>	</paramid>
30.		Pavadinimas	<paramname>	</paramname>
31.		Tipas	<type>	</type>
32.	Scenarijus	Scenarijus	<scenario>	</scenario>
33.		ID	<scnrid>	</scnrid>
34.		Pavadinimas	<scnrname>	</scnrname>
35.		Žingsnių sąrašas	<steplist>	</steplist>
36.	Žingsnis	Žingsnis	<step>	</step>
37.		Indeksas	<index>	</index>
38.		Lango ID	<nodeid>	</nodeid>
39.		Komandos ID	<cmdid>	</cmdid>
40.		Parametrų sąrašas	<paramlist>	</paramlist>

	Grupė	Pavadinimas	Pradžios etiketė	Pabaigos etiketė
41.		Parametras	<param>	</param>
42.		Parametro ID	<paramid>	</paramid>
43.		Parametro reikšmė	<value>	</value>

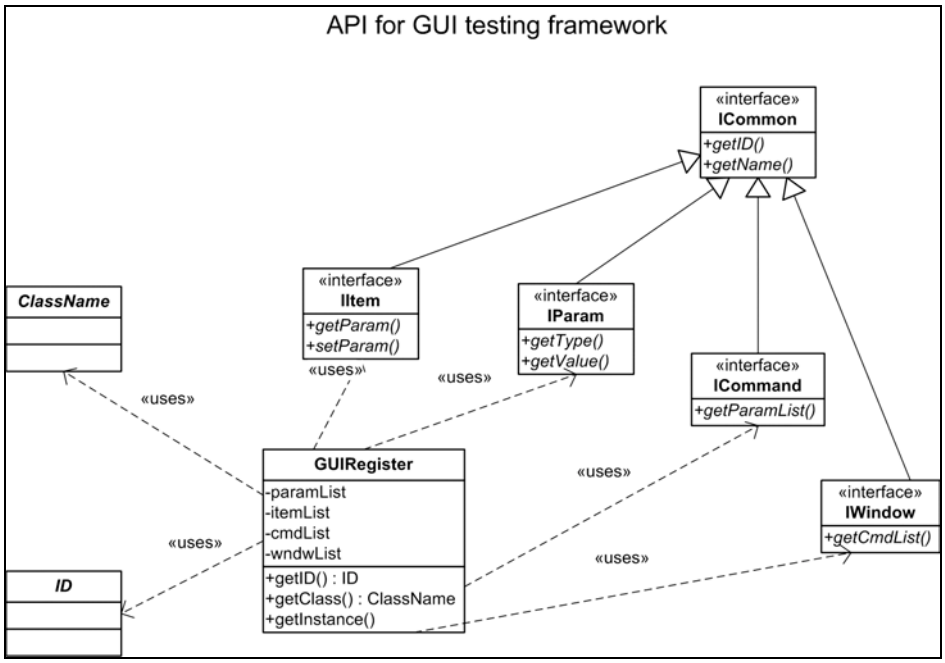
API

Tam, kad būtų galima atlikti grafinės sąsajos automatizuotą testavimą funkciniame lygyje, reikia kad jos grafiniai komponentai paveldėtų programinę sąsają (API) (1.4 pav.). Per API galime gauti šiuos parametrus:

- Kiekvieno lango identifikacijos kodą (*id*), pavadinimą (*name*), galimų komandų sąrašą (*cmdlist*).
- Kiekvienos komandos identifikacijos kodą (*id*), pavadinimą (*name*), parametrų sąrašą (*paramlist*)
- Kiekvieno parametro identifikacijos kodą (*id*), pavadinimą (*name*), tipą (*type*), reikšmę (*value*).

Pagrindinė problemą yra kodo perkodavimas, kurio negalime išvengti dėl mobiliųjų įrenginių apribojimų. Kodo perkodavimo metu visų klasių pavadinimai pakeičiami nesudarant vardų atitikimo lentelės. Tai sudaro problemą, jei mes norime kurti objektą pagal jo klasės pavadinimą.

Kodo perkodavimo problemai išvengti yra siūloma naudoti objektų identifikacijos kodo susiejimą su perkoduotu klasės pavadinimu. Testavimo aplinkos mobiliajame įrenginyje inicijavimo metu turi būti kietais suprogramuotas grafo objektų susijimas su jų ID. Kai mobiliajame įrenginyje bus vykdomas testavimo aplinkos inicijavimas, visų klasių pavadinimai jau bus perkoduoti. Todėl perkodavimo metu mes sužinosime naujus klasių pavadinimus. Objekto ID ir klasių susijimo lentelių apdorojimui bus naudojamas grafinių komponentų registras.



1.4 pav. Testavimo aplinkos programavimo sąsaja

1.6. Grafinės vartotojo sąsajos specifikavimo UML diagramomis apžvalga

Testavimas užima didelę dalį programos kūrimo cikle įdėtų pastangų, todėl suprantami ir šios veiklos nemaži kaštai. Nors pažangūs procesai ir įrankiai padėjo organizacijoms sumažinti programinių sistemų kūrimo laiką, tačiau dar nesugebėjo žymiai sumažinti laiko reikalingo šiom programom ištestuoti. Modelių naudojimas programų vystymą pakėlė į tam tikrą abstrakcijos lygį [1]. Tai kodėl nepanaudoti modelių ir testuojant? Šiame skyrelyje apžvelgsime keletą būdų kaip specifikuoti vartotojo sąsają naudojant UML priemones.

Grafinės vartotojo sąsajos specifikavimas UMLi priemonėmis

Normanas Patonas iš Mančesterio universiteto vartotojo sąsajos specifikavimui pasiūlė UML praplėtimą pavadintą UMLi [9]. UMLi notacijoje pateikiami šešių tipų vartotojo sąsajos komponentai:

- *FreeContainers* – tai klasė kurios jokia kita klasė negali turėti savyje. Ši klasė yra konteineris visiems kitiems grafiniams komponentams.
- *Containers* – grupuojami grafiniai komponentai.
- *Inputers* –informacijos įvesties grafiniai komponentai. Per šias klases gaunama vartotojo įvesta informacija.
- *Editors* – tai skirtas dvikrypčiam informacijos apsikeitimui. Šiais komponentais galima tiek įvesti tiek išvesti informaciją.
- *Displayers* – informacijos išvesties komponentai.

- *ActionInvokers* – perduodamos vartotojo instrukcijos.

Funkcionalumui išreikšti naudojamos UML notacijos panaudos atvejų ir veiklos diagramos. Tiesa, UMLi naudoja praplėstą veiklos diagramų versiją. Naudojant šią notaciją apsisaugojama nuo tendencijos, kad veiklos diagramos tampa labai sudėtingos aprašant gan nesudėtingas vartotojo sąsajas. Tačiau šio metodo naudojimui nėra išvystyta įrankių pakankamai įrankių. Tam galima naudoti, pvz. ArgoUMLi.

Grafinės vartoto sąsajos specifikavimas UML panaudos atvejais ir veiklos diagramomis

Pasiūlytame metode [8], panaudos atvejų diagrama susideda iš aktorių (vartotojai ir išorinės sistemos) bei panaudos atvejų. Tarp aktorių gali būti apibendrinimo (*generalization*) ryšys, o tarp panaudos atvejų - <<include>> ryšys bei apibendrinimo ryšys. Tarp aktorių ir panaudos atvejų yra tik paprasti asociatyviniai ryšiai.

Veiklos diagrama susideda iš būsenų, bei dviejų specialių būsenų t.y. pradinės ir galinės. Būsenos gali būti susietos pažymėtais perėjimais (rodyklėmis). Perėjimas gali išsišakoti, kur rombas pažymi išsišakojimo vietą.

Metodo taikymo žingsniai

a) Pirmiausiai, naudojant panaudos atvejų diagramą, padaroma laisvos formos aukšto lygio sistemos aprašymas. Į šia diagramą įeina aktoriai ir pagrindiniai panaudos atvejai.
b) Antra, kiekvieno panaudos atvejo elgsena, besinaudojant pateiktais apribojimais, aprašomas veiklos diagrama. Taip gaunamas formalesnė diagrama.
c) Panaudos atvejų ir veiklos diagramos yra transformuojamos į klasių diagramą.
d) Iš sugeneruotos klasių, bei ankščiau sukurtos panaudos atvejų diagramos generuojamas navigacijos grafas.
e) Generuoti kodo fragmentus iš prieš tai sugeneruotos klasių diagramos. Šie kodo fragmentai gali būti grafinės vartotojo sąsajos komponentų prototipai.

Vartotojo sąsajos langai suskirstyti į dvi dalis – apletai ir freimai. Apletas – tai toks komponentas kuris savyje gali turėti grafinių komponentų, tačiau jo ne negalima iškviesti kitų apletų ar freimų. Freimas – toks komponentas kuris savyje gali turėti grafinių komponentų ir iš jo gali būti iškviesti kiti apletai arba freimai.

Vartotojo sąsajos projektavimo taisyklės:

1. Kiekvienas aktorius panaudos atvejų diagramoje yra apletas arba freimas. Aktoriai vaizduojantys išorines sistemas nėra įtraukiami į vartotojo sąsajos projektavimą. Kas aktorius bus, apletas ar freimas, priklauso nuo to su keliais panaudos atvejais jis sąveikauja.
2. Apibendrinimo ryšys tarp aktorių p ir q reiškia, kad apletas/freimas p paveldi apletą/freimą q.
3. Kiekvienas panaudos atvejis atitinka apletą/freimą priklausantį freimui, atitinkančiam aktorių su kuriuo panaudos atvejis siejasi.
4. Apibendrinimo ryšys tarp aktorių u ir w reiškia, kad apletas/freimas u paveldi apletą/freimą w.
5. <<include>> ryšys tarp dviejų panaudos atvejų u ir w reiškia u įšvietimą iš w.
6. Kiekviena veiklos diagramos būsena gali būti arba terminalinė arba ne terminalinė. Terminalinė būsena yra pažymėta UML stereotipu ir reiškia grafinį išvesties komponentą. Kiekviena neterminalinė būsena neturi stereotipo ir yra detalizuojama kita veiklos diagrama. Neterminalinė būsena gali atitikti kitą panaudos atvejį arba ne.
7. Kiekvienas veiklos diagramos perėjimas gali būti pažymėtas UML sąlyga (*condition*) arba stereotipu ir sąlyga. Sąlygos vaizduoja vartotojo pasirinkimus arba verslo logiką (*business logic*).
8. Neterminalinėms būsenoms galima nustatyti <<include>> arba apibendrinimo ryšį su kita neterminaline būsena ir panaudos atveju. Šiuo atveju turime laikytis tokių taisyklių: (a) <<include>> ryšio atveju neterminalinė būsena yra freimas arba apletas kuriame yra grafiniai komponentai iš atitinkamos veiklos diagramos; (b) apibendrinimo ryšio atveju būsena taip pat yra freimas/apletas kuriame yra grafiniai komponentai iš atitinkamos veiklos diagramos, tačiau ir panaudos atvejis, kuris paveldi minėtą būsena, turi savyje tuos pačius komponentus.
9. Neterminalinė būsena neturinti atitinkamo panaudojimo atvejo panaudos atvejų diagramoje nėra nei apletas nei freimas, ir grafinės sąsajos komponentai atitinkamoje veiklos diagramoje yra susijusio panaudos atvejo freimo/apleto grafiniai komponentai.
10. Veiklos diagramų perėjimų sąlygos nėra įtraukiamos į klasių diagramą.

Panaudos atvejų ryšiai yra interpretuojami taip:

11. <<include>> tipo ryšys tarp panaudos atvejo u ir panaudos atvejo w reiškia, kad viena panaudos atvejo u veiklos diagramos neterminalinė būsena yra w.

12. Apibendrinimo ryšys tarp panaudos atvejo u ir panaudos atvejo w reiškia, kad panaudos atvejį w detalizuojanti veiklos diagrama turės kai kurias būsenas ir perėjimus iš veiklos diagramos detalizuojančios panaudos atvejį u. Be to, kai kurios veiklos diagramos u būsenos s veiklos diagramoje w gali būti pakeisti būsenomis s'. Veiklos diagrama w gali turėti ir naujų būsenų bei perėjimų.

Šiuo metodu naudojant standartinius UML įrankius galima aprašyti vartotojo sąsajos elgseną. Iš sugeneruotos klasių diagramos galima generuoti vartojo sąsajos kodo šabloną ir tuo pačiu testavimo navigacijos grafą.

1.7. Metamodelių specifikavimo XMI standartas

XMI – tai XML žymių kalba pagrystąs standartas metamodelių aprašymui. Šis standartas skirtas palengvinti informacijos apie metamodelius ir metaduomenis apsikeitimui ir naudojimui. Atskiru atveju tai skirtas padėti programuotojams naudoti UML modelio duomenis. Kadangi metodas pagrystąs XML duomenimis, nepriklauso kokiais įrankiais yra sukurtas modelis, svarbu, kad būtų galimybė transformuoti modelį XMI formatu [10, 11]. Taip pateiktus duomenis galima analizuoti naudojant programines priemones, tokias kaip XML DOM ar SAX. Nesudėtingoms transformacijoms gali užtekti XSLT transformacijų [12].

2. MOBILAUS DĖSTYTOJO PROGRAMINĖ ĮRANGA

Šioje dalyje apžvelgsime magistratūros studijų metu sukurtą programinę įrangą – mobilaus vadybininko darbo vietą.

2.1. Tikslas

Projekto tikslas yra remiantis kitų KTU studentų padarytais sprendimais praplėsti mobilaus dėstytojo programinę įrangą. Delninis kompiuteris ne tik padėtų dėstytojui lengviau atlikti visus rutininius darbus, bet ir vesti paskaitas, lengviau suprasti studentų poreikius bei produktyviau vesti paskaitą. Tuo pačiu, ši sistema, padėtų studentam lengviau aktyviai dalyvauti paskaitoje, perteikti dėstytojui tai ko jiem labiau reikia. Sistemos teikiamos funkcijos:

- Testų sudarymas ir vertinimas realiu laiku
- Klausimų apie dėstomą dalymą priėmimas (studentai galėtų klausimus pateikinėti naudodami delninius ar nešiojamus kompiuterius prijungtus prie bevielio tinklo)
- Paskaitos vertinimas (studentai, naudodami tuos pačius būdus kaip pateikdami klausimą, galėtų vertinti dėstytojo paskaitą realiu lauku)
- Transformuoti dėstytojų turimas informacines sistemas į suprantamas sistemai.
- Lankomumo žymėjimas, ataskaitos
- Atsiskaitymų rezultatų žymėjimas, jų ataskaitos

Dėstytojas, naudodamas delninį kompiuterį ir sukurtą sistemą, rutininiams darbams galės skirti kur kas mažiau laiko, galės gerinti paskaitų kokybę atsižvelgdamas į studentų poreikius. Procesas padės studentams galvoti apie dėstomą medžiagą, o dėstytojas paskaitos metu galės lengviau įvertinti studentų supratimo lygį

2.2. Sistemos galimybės

Išskiriamos dvi pagrindinės sistemos panaudojimo sritys:

- Dėstytojo mobili darbo vieta rutininiams darbams atlikti (lankomumo žymėjimas, pažymių įvedimas, ataskaitų gavimas)
- Paskaitų vedimo kokybės gerinimui (virtualiam klausimų pateikimui, apklausom ir pan.)

2.2.1. Lankomumo žymėjimas

Dėstytojas naudodamasis delniniu kompiuteriu galėtų lengvai žymėti studentų lankomumą, pavėlavusio studento pažymėjimas kaip dalyvaujančio užsiėmime būtų daug lengvesnis negu naudojant popierinį variantą [1]. Suvesta informacija būtų perduodama į centrinę duomenų bazę, o vėliau panaudojama ataskaitų ir suvestinių generavimui.

2.2.2. Rezultatų registravimas

Naudojant delninį kompiuterį būtų galima lengvai suvesti studentų pažymius. Dėstytojas galėtų suvedinėti pažymius laboratorinių metu, būdamas namie, vertindamas studentų darbus. Rezultatai iš kart ar vėliau būtų perkeliami į centrinę duomenų bazę.

2.2.3. Paskaitų tvarkaraščiai

Tvarkaraščius būtų galima peržiūrėti naudojant delninį kompiuterį.Kompiuteris galėtų pranešti apie artėjantį užsiėmimą. Galima būtų pažymėti, kad koks užsiėmimas nukeliamas ar dėstytojas vėluos ir visa tai būtų galima pateikti kitiems dėstytojams, personalui (kad galėtų kas nors pavaduoti dėstytoją, ar bent pranešti studentams apie užsiėmimo nukėlimą) [6].

2.2.4. Ataskaitų gavimas

Dėstytojas naudodamas delninį kompiuterį galėtų bet kuriuo metu gauti informaciją apie paskaitų tvarkaraščius, paskaitose turinčius dalyvauti studentus, jų rezultatus, bei įvairią statistiką.

2.2.5. Praktinių užsiėmimų vedimas

Kai kurie užsiėmimai būna vedami tokiu būdu, kad studentai dirba ir kai nori pristatyti savo darbą arba pasikonsultuoti su dėstytoju jie užsirašo į eilę nurodydami savo darbo vietą. Dėstytojas atsilaisvinęs tikrina sąrašą ir eina pas pirmą studentą norintį pristatyti darbą ar turinti klausimų. Jei studento toje vietoje nėra, pereinama prie kito studento sąraše. Gali būti vedami ir keli tokie užsiėmimai vienu metu ir gali studentams padėti keli dėstytojai. Problemos kyla kai studentai pakeičia darbo vietas ir nepavyksta atnaujinti sąrašo realiu laiku. [6]

Siūlomas sprendimas yra kiekvienam dėstytojui duoti naudoti delninį kompiuterį, kuriame jis galėtų matytį tą sąrašą su vietomis kur yra konkretus studentas. Studentams sukurti web puslapį kur jie galėtų save užregistruoti, kad laukia pagalbos ar nori pristatyti savo darbą. Studentai pakeitę darbo vietą galėtų lengvai pažymėti tai, neprarasdami savo

vietos sąraše. Taip pat sistema galėtų teikti informaciją: statistika apie studentų laukimo laiką, laiko trukmę, kuria dėstytojas dirbo su konkrečiu studentu. Nuspėjamus laukimo laikus studentams, sąrašo stebėjimą. [6].

Tokiam sprendimui realizuoti reiktų bevielio prisijungimo būdo dėstytojams (mobilus telefonas, W-LAN).

2.2.6. Medžiagos pateikimas

Delninius kompiuterius galima naudoti rodant pateiktis (ppt, pdf).Mobiliu įtaisu valdant kompiuterį rodantį prezentaciją, arba net rodant pateiktis iš nešiojamo kompiuterio [7]. Dėstytojui aiškinant paskaitą nereikėtų sėdėti prie kompiuterio valdant pateiktis. Taip būtų galima laisvai vaikščioti po auditoriją, laisviau ir įtaigiau naudoti neverbalinę kalbą.

2.2.7. Paskaitos vedimas

Klausimo paskelbimas

Dauguma studentų yra per drovus aktyviai dalyvauti paskaitoje komentuojant pateiktis, siūlant savo idėjas ar klausiant klausimus [4]. Studentai daug laisviau pateikinėtų savo idėjas jei tai galėtų atlikti tyliai ir anonimiškai. Vėliau dėstytojui atsakinėjant į virtualiai pateiktus klausimus studentai naujus iškilusius klausimus drąsiau pateiktų gyvai [3]. Kad būtų pirmiausia atsakinėjama į aktualiausius klausimus turėtų būti įdiegta balsavimo sistema. Studentas paskaitų metu matydami kitų studentų pateiktus klausimus galėtų balsuoti už jam aktualius klausimus taip suteikdamas jiem pirmenybę kitų klausimų atžvilgiu.

Paskaitos vertinimas

Studentai taip pat gali vertinti paskaitą balsuojant už ją. Dėstytojas mato realaus laiko vertinimų diagramą per pasirinktą laiko tarpą. Silpnas įvertinimas per trumpą laiko tarpą parodo, kad nagrinėjamą problemą reikėtų išaiškinti išsamiau. Tradicinėje mokymosi aplinkoje studentai galėtų kelti rankas ir reikalauti išsamesnio paaiškinimo, tačiau tai nutinka retai[4].

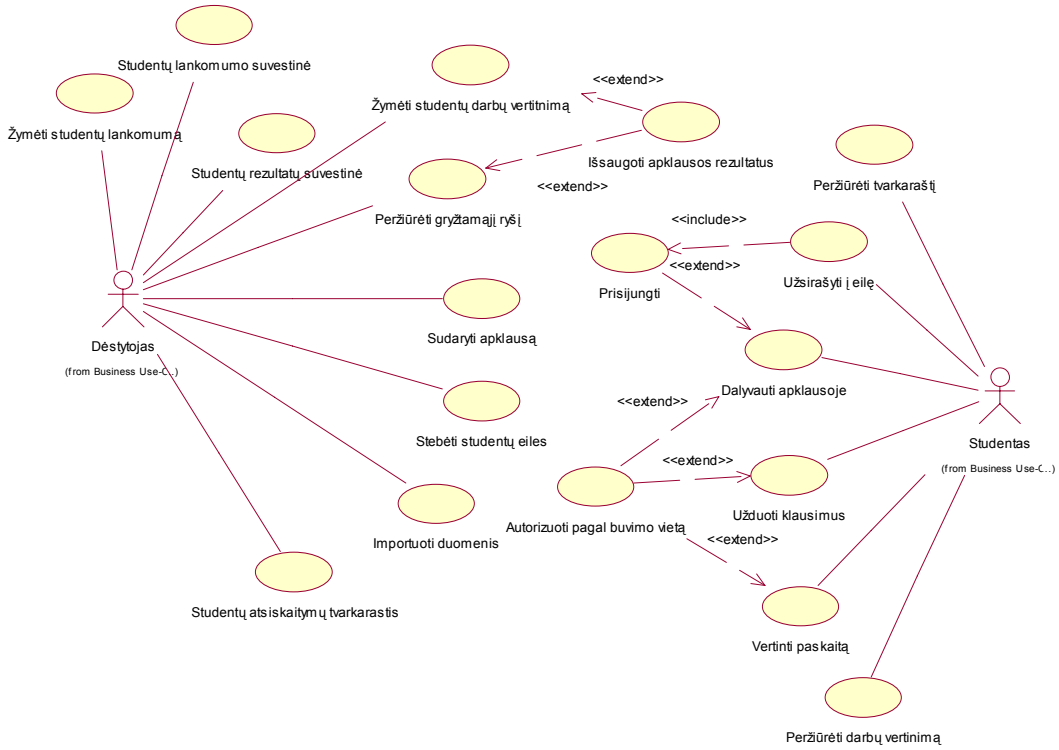
Apklausos

Dėstytojai negali žinoti ar studentai suprato dalyką taip kai jis norėjo. Todėl naudodamas delninį kompiuterį galėtų sukurti testą su pasirenkamais atsakymų variantais. Rezultatai gali būti peržiūrimi iškarto. Tai dėstytojui suteikia galimybę reaguoti, jei rezultatai

parodo, kad studentai dalyką suprato prastai. Šias apklausas būtų galima atlikti ir ne anoniminiu būdu, vėliau įtraukiant apklausų rezultatus į galutinį pažymį.

2.3. Sprendimo realizacija

Panaudos atvejų diagrama pateikta 2.1 paveiksle.



2.1 pav. Sistemos panaudos atvejų diagrama

Importuoti duomenis: Duomenys apie modulius, studentų sąrašai importuojami į duomenų bazę iš HTML, Xml, arba Exel.

Stebėti studentų eiles: Studentų eilės peržiūrėjimas ir koregavimas.

Sudaryti apklausą: Apima procesą kai yra sudaroma apklausa susidedanti iš įvairių klausimų kurie gali būti testo tipo arba ne. Pateikiamas atsakymų šablonas pagal kurį vėliau yra automatiškai tikrinami atsakymai ir pateikiami rezultatai.

Peržiūrėti grįžtamąjį ryšį: Apima studentų grįžtamojo ryšio peržiūrėjimą. T.y. studentų paskaitos vertinimą per pasirinktą laiko intervalą, apklausos rezultatų peržiūrėjimą ir studentų virtualiai pateiktų klausimų peržiūrėjimas.

Išsaugoti apklausos rezultatus: Pateiktos apklausos rezultatai gali būti panaudoti kaip galutinio semestro pažymio sudedamoji dalis. Atlikdamas testą studentas turi būti autentifikuotas.

Užsirašyti į eilę: Studentas užsirašo į eilę pagal kurią vėliau dėstytojas konsultuoja studentus laboratorinių ar praktinių darbų metu.

Prisijungti studentui prie paskaitos: Autentifikuojamas studentas dalyvaujantis paskaitoje ar praktiniame užsiėmime.

Vertinti paskaitą: Studentas dalyvaujantis paskaitoje savo nuožiūra dašimtbalėje sistemoje vertina paskaitos kokybę.

Užduoti klausimą: Studentas virtualiai pateikia dėstytojui klausimą. Jei norimas klausimas jau yra pateiktas kito studento, tai jis gali būti reitinguojamas.

Dalyvauti apklausoje: Dėstytojas pateikia paklausą. Studentas yra prisijungęs prie paskaitos, autentifikuotas arba ne, priklausomai nuo apklausos reikalavimo.

Registruoti atsiskaitymą: Sudaromas atsiskaitymų tvarkaraštis. Nustatomos darbų pristatymo paskutinės datos, nustatomos darbų gynimo datos.

Žymėti studentų lankomumą: Žymimi sąraše atvykę studentai į pasirinktą užsiėmimą.

Žymėti studentų darbų vertinimą: Žymimi studentų darbų vertinimai (laboratoriniai, kontroliniai darbai, egzaminas).

Studentų rezultatų suvestinė: Parodomas studentų rezultatų suvestinės.

Studentų lankomumo suvestinė: Parodomas studentų lankomumo suvestinės.

Vartotojų administravimas: Registruojami, šalinami, redaguojami sistemos vartotojai.

Peržiūrėti tvarkaraštį: Parodomas studento individualaus modulio atsiskaitymo tvarkaraštis. Pranešama apie artėjančius įvykius (paskaita, laboratorinis darbas, kontrolinis darbas)

Peržiūrėti darbų vertinimus: Parodomas studento modulio darbų vertinimų suvestinė.

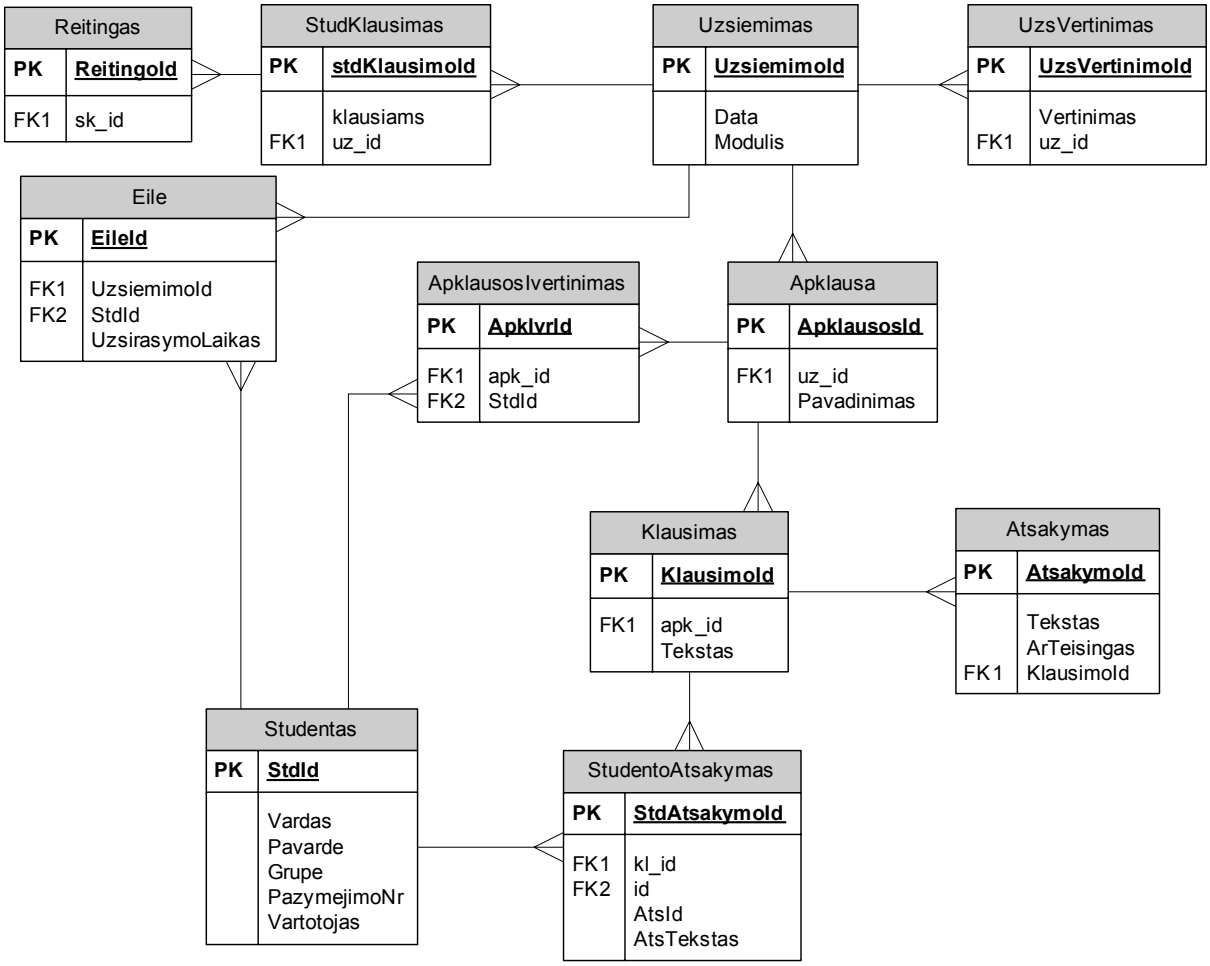
Prisijunkti: Vartotojas prisijungia prie sistemos nurodydamas savo prisijungimo vardą ir slaptažodį.

Registruoti užsiėmimą: Sudaromas užsiėmimų tvarkaraštis. Pažymima kada vyks paskaitos, kada laboratoriniai darbai, kada kontroliniai darbai. Nustatomos darbų pristatymo paskutinės datos.

2.4. Duomenų vaizdas

Duomenų bazės valdymo sistemai yra pasirinkta MSDE (*Microsoft SQL Server 2000 Desktop Engine*) duomenų bazės valdymo sistema. Duomenų bazės modelis pateiktas

2.2 paveiksle.



2.2 pav. Duomenų bazės modelis.

Duomenų bazės modelyje esančių esybių aprašymai pateikti 2 lentelėje.

Lentelė 2: Duomenų bazės modelio esybės

Esybė	Aprašymas
Studentas	Saugoma informacija apie studentus (jų priklausomybės grupėms).
StudKlausimas	Šioje lentelėje saugomi paskaitos metu studentų virtualiai užduoti klausimai
Reitingas	Šioje lentelėje saugomas studentų klausimui suteiktas reitingas.
Eile	Saugom informacija apie studentus užsirašiusius į eilę.
Apklausa	Saugoma informacija apie apklausą.
Klausimas	Saugomi apklausos klausimai.
Atsakymas	Saugomi galimi atsakymai į apklausos klausimą.
StudentoAtsakymas	Saugomas studento atsakymas į klausimą.

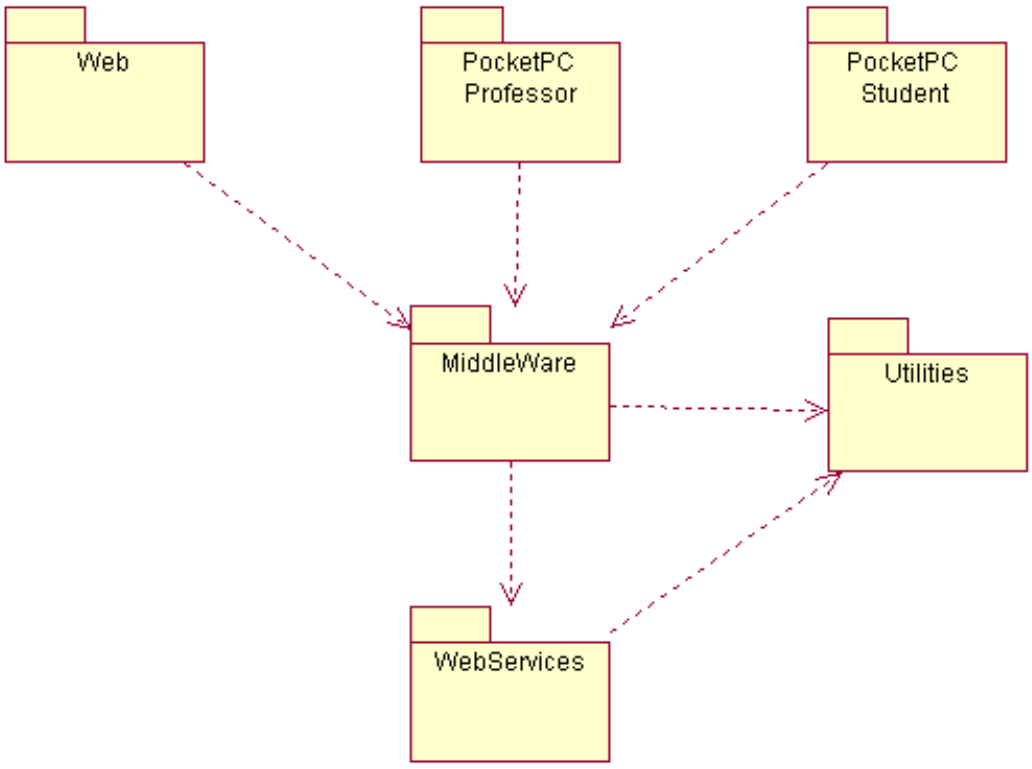
Užsiėmimai	Saugoma informacija apie modulių užsiėmimus (jų datos).

2.5. Architektūra

Šiame skyrelyje pateikiamas sistemos išskaidymas į posistemes ir paketus. Detalesnis sistemos aprašymas.

Klasių diagramos

Sistema susideda iš tokių pagrindinių paketų: Web, PocetPC Professor, Pocket PC Student, MiddleWare, Utilities, WebServices. Sistemos paketų diagrama pavaizduota 2.3 paveiksle.

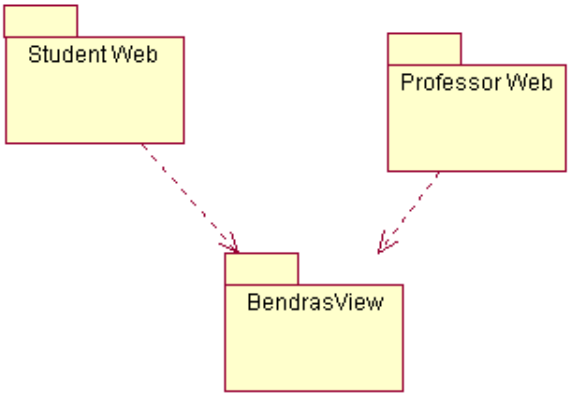


2.3 pav. Sistemos suskirstymas į paketus.

Paketas Web

Pakete WEB pateikiamos klasės skirtos bendravimui su vartotoju per internetą. Jame yra tik sistemos vaizdavimo lygio klasės (duomenų išvedimo langai ir įvedimo langai). Šiame pakete esančios klasės visoms funkcijoms atlikti naudoja klases esančias MiddleWare pakete. Pakete pateikiamos klasės realizuojančios studento ir vartotojo sąsajas, jos yra detaliau suskirstytos į žemesnio lygio paketus. Pakete „Bendras“ pateikiamos klasės skirtos bendram naudojimui visiems kitiems paketams. Pakete pateikiamos apibendrintos klasės puslapio

vaizdavimui, prisijungimo langui ir ataskaitų formavimui. Paketo WEB suskirstymas į paketus pateiktas 2.4 pav.



2.4 pav.: Paketo Web sudėtis

Paketas PocketPc Professor

Pakete PocketPc Professor pateikiamos klasės skirtos bendravimui su vartotoju delniniame kompiuteryje. Jame yra tik sistemos vaizdavimo lygio klasės (duomenų išvedimo langai ir įvedimo langai). Šiame pakete esančios klasės visoms funkcijoms atlikti naudoja klases esančias MiddleWare pakete. Pakete pateikiamos klasės realizuojančios dėstytojo vartotojo sąsają.

Paketas PocketPc Student

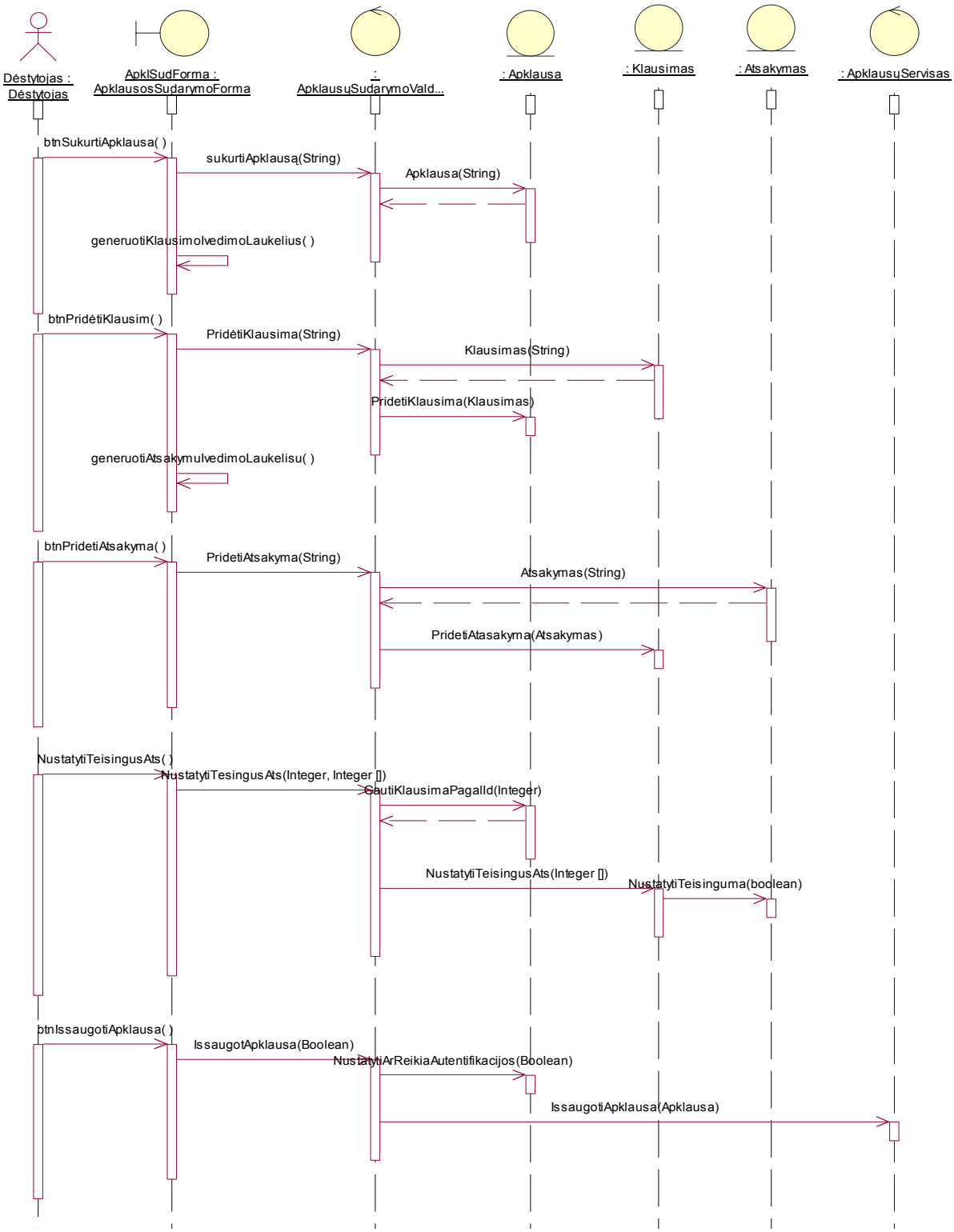
Pakete PocketPc Student pateikiamos klasės skirtos bendravimui su vartotoju delniniame kompiuteryje. Jame yra tik sistemos vaizdavimo lygio klasės (duomenų išvedimo langai ir įvedimo langai). Šiame pakete esančios klasės visoms funkcijoms atlikti naudoja klases esančias MiddleWare pakete. Pakete pateikiamos klasės realizuojančios studento vartotojo sąsają .

Paketas MiddleWare

Vartotojo sąsaja realizuojama delniniame kompiuteryje ir per internetinę prieigą. Kadangi abi vartotojo sąsajos atlieka tą patį funkcionalumą tai norint išvengti kodo dubliavimo ir palengvinti palaikymą bus kuriamas šis paketas. Šio paketo klasės palaiko sistemos būseną, atlieka reikalingus skaičiavimus vartotojo sąsajos pusėje. Šis paketas atitinkamai pagal vartotojo sąsajos tipus suskirstytas į du sudėtinius paketus.

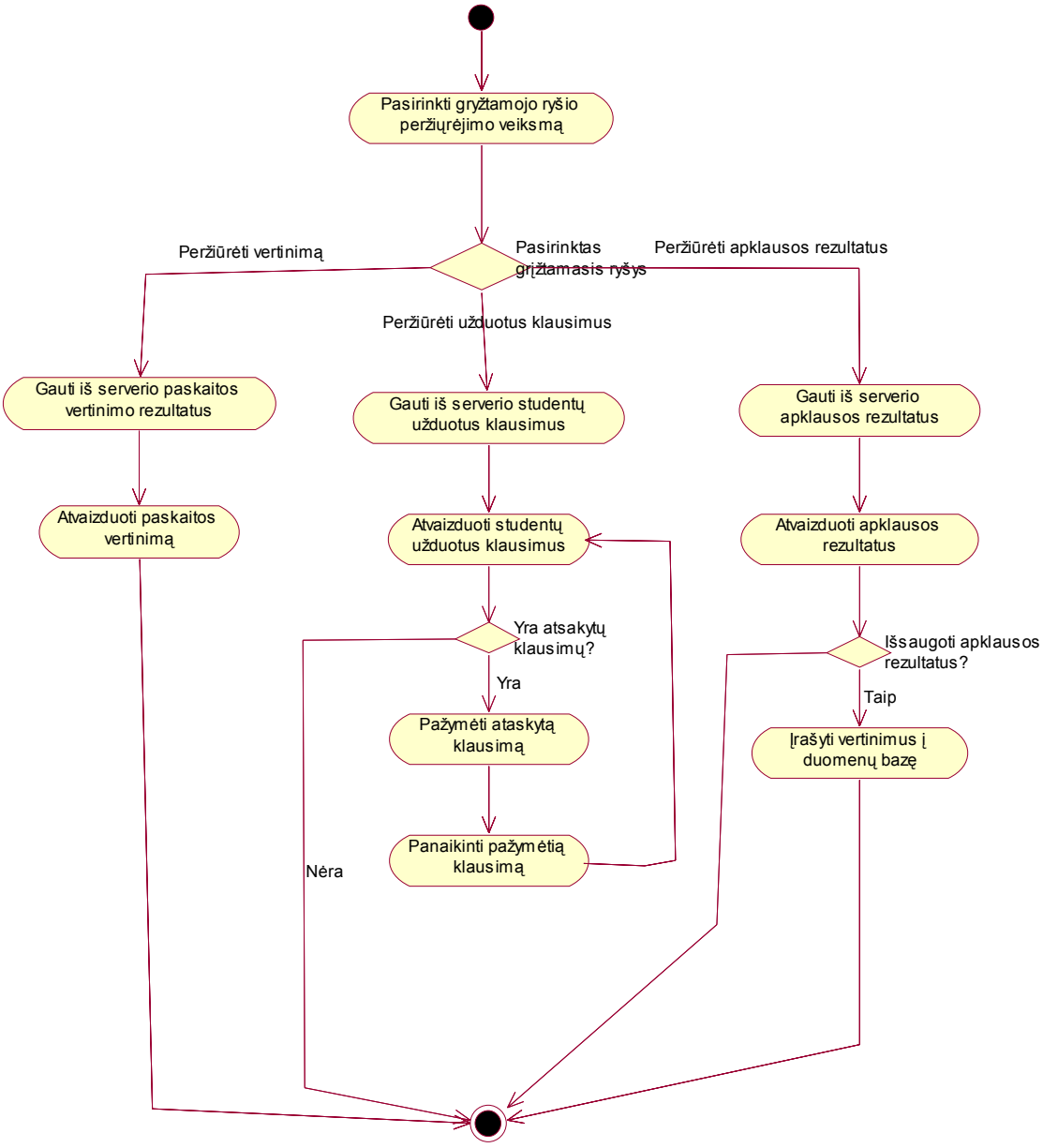
Dinaminis vaizdas

Projekto dinaminiam vaizdui perteikti buvo naudojamos veiklos ir sekų diagramos. Apklausos sudarymo veiklos diagrama patekta žemiau esančiame paveiksle (2.5 pav.)



2.5 pav. Apklausos sudarymo sekų diagrama

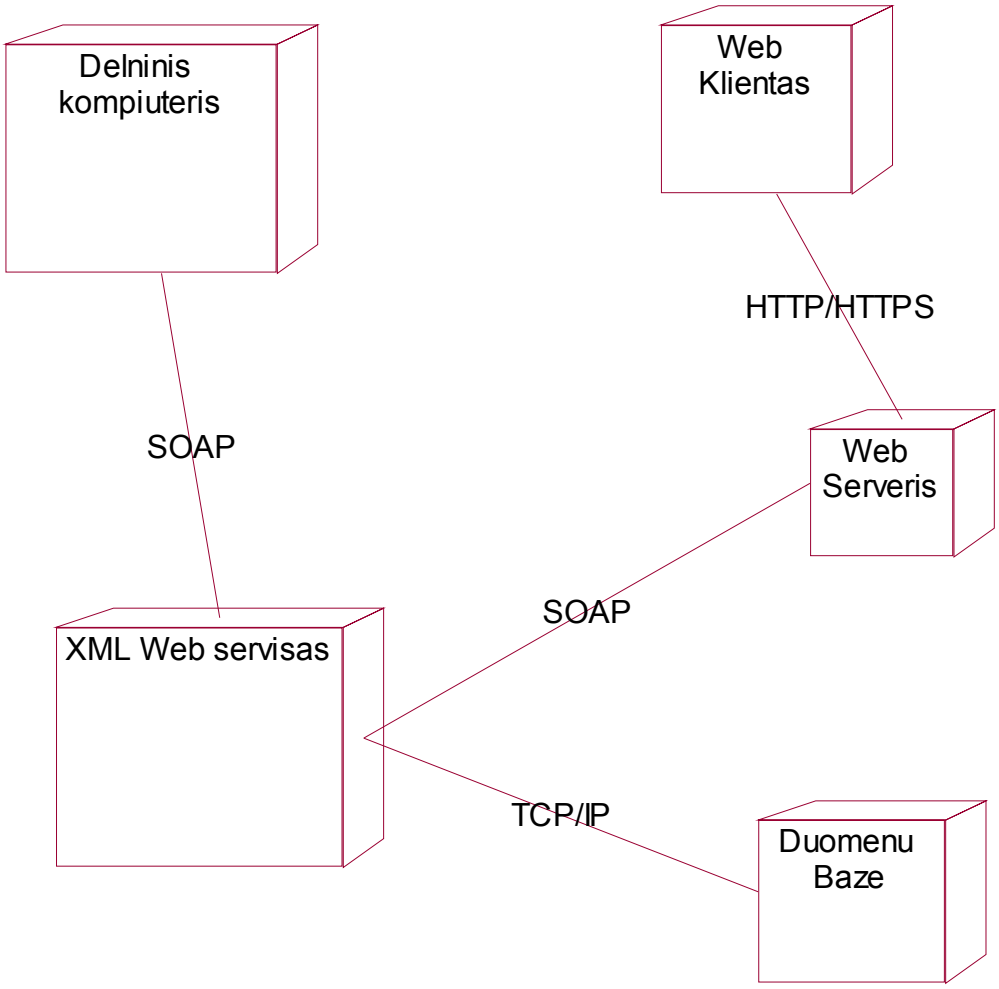
Grįžtamojo ryšio peržiūrėjimo veiklos diagrama pateikta 2.6 paveiksle.



2.6 pav. Grįžtamojo ryšio peržiūrėjimo veiklos diagrama

Išdėstymo diagrama

Sistemos išdėstymo (*deployment*) diagrama pateikta 2.7 paveiksle.



2.7 pav. Sistemos išdėstymo diagrama

Duomenų Baze

Duomenų bazė diegiama Microsoft Windows 2003 Server kompiuteryje. Duomenų bazės valdymo sistemai naudojama Microsoft SQL Server Desktop Engine.

Minimalus CPU: 133 MHz

Minimalus RAM kiekis: 128 MB

Minimalus Disko dydis 2 GB

Serveriui reikia tinklo palaikymas. Microsoft SQL Server 2000 Desktop Engine teikia paslaugas naudojant TCP/IP protokolą.

XML Web servisas

XML WEB servisas diegiamas Microsoft Windows 2003 Server kompiuteryje. Jam įdiegti dar naudojama Microsoft Infomation Services 6.0 web serverio versija.

Minimalus CPU: 133 MHz
Minimalus RAM kiekis: 128 MB
Minimalus Disko dydis 2 GB

Serveriui reikia tinklo palaikymas. XML Web servisas teikia paslaugas naudojant SOAP protokolą, kuris yra perduodamas naudojant HTTP protokolą, o HTTP naudoja TCP/IP protokolą perduoti duomenis žemame lygyje.

XML WEB servisas realizuojamas .NET platformoje ir naudoja Microsoft .NET Framework virtualia mašiną.

Kompiuteryje turi būti įdiegta Microsoft .NET Framework 2.0.

Jame diegiami komponentai iš paketo VeiklosPaslaugos.

3. TESTAVIMO SKRIPTŲ GENERAVIMO METODIKA

Šioje dalyje apžvelgsime siūlomą grafinės vartotojo sąsajos mobiliems įrenginiams testavimo metodą.

Magistratūros studijų metu buvo vykdomas programinės įrangos mobiliai įrangai kūrimo projektas. Sukūrus projektą buvo numatytos programų kūrimo procesą gerinančios galimybės. Šių galimybių realizavimui siūlomas grafinės sąsajos mobiliems įrenginiams testavimo metodo apžvelgto 1.5 praplėtimas.

3.1. Problemos

Testavimo aplinkoje pateiktoje 1.5 skyrelyje išspręstos tokios mobiliems įranginiams adresuojamos problemos kaip maža mobilių įrenginių atmintis, bei atspindžio panaudojimo problema obfuskavus programos kodą. Tačiau naudojant pateiktą testavimo aplinką testavimo scenarijus bei navigacijos grafą reikia sudarinėti rankiniu būdu. Nėra numatyta jokių priemonių, kad šį procesą automatizuoti. Toks procesas užima daug laiko ir dėl žmogiškojo faktoriaus gali įnešti daug klaidų. Atsiradus pakeitimų vartotojo sąsajoje reikia keisti ir navigacijos grafą ir scenarijų. O tai papildomai vėl įneša daug rankinio darbo, o kartu ir klaidų. Taigi, keičiant programos specifikaciją keičiasi realizacija, todėl natūralu būtų testavimo scenarijus bei navigacijos grafą generuoti iš grafinės vartotojo sąsajos specifikacijos. Problema yra ta, kad nėra vienareikšmių priemonių grafinei vartotojo sąsajai specifikuoti. Yra tik keletas apribotų sprendimų (žiūrėti 1.7 skyrelį).

Testavimo aplinkos, pateiktos 1.5 skyriuje, navigacijos grafo bei scenarijaus aprašyme numatytu aprašymu neina tikrinti lango pagal jo būseną. Aprašomas tik būsenų sąrašas, tačiau nėra duomenų kuo jos skiriasi. Norint aprašyti langą kitoje būsenoje reikalinga atskira viršūnė navigacijos grafe, o tai nėra patogu, be to atsiranda duomenų dubliavimas.

3.2. Tikslai

Mobilios programinės įrangos grafinės testavimo aplinkai siūlomi tokie patobulinimai:

- Grafinės vartotojos sąsajos specifikavimo UML diagramomis metodas.
- Grafinės vartotojo sąsajos specifikacijos transformavimas į klasių diagramą.
- Grafinės vartotojo sąsajos apribojimų aprašymas OCL priemonėmis.
- Navigacijos grafo generavimas iš klasių ir panaudos atvejų diagramų.
- Scenarijų generavimas iš sekų diagramų.

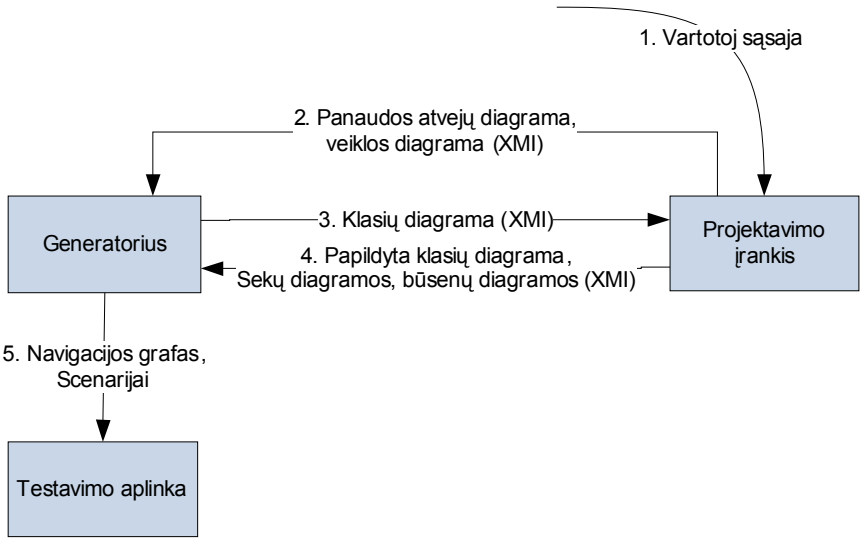
Šiam tikslui reikia sukurti atskirą programą, kuri generuotų navigacijos grafą bei scenarijus. O sugeneruoti duomenys būtų pateikiami testavimo aplinkai. UML diagramos gali būti sukuriamos, bet kokiu įrankius, kuris palaiko XMI eksportavimą.

3.3. Siūlomas sprendimas

Aukščiau išvardintiems tikslams įgyvendinti reikalingos tokios priemonės ir metodai:

- Projektavimo priemonė palaikanti UML diagramas ir galinti eksportuoti modelį XMI formatu.
- Papildoma programinė įranga kuri pagal 1.6 skyriuje aprašytą metodika galėtų transformuoti panaudos atvejų ir veiklos diagramas į klasių diagramą.
- Generatoriaus kuris:
    - Kuriame būtų OCL analizatorius ir iš aprašytų išraiškų galėtų sugeneruoti apribojimus
    - Iš pateiktų klasių ir panaudos atvejų diagramų generuotų navigacijos grafą.
    - Iš sekų diagramų generuotų testavimo scenarijus.

Duomenų mainų schema pateikta 3.1 paveiksle.



3.1 pav.: Duomenų mainų schema.

Duomenų mainų schemos aprašymas:

1. Projektuotojas (arba testuotojas) projektavimo įrankiu sukuria panaudos atvejų ir veiklų diagramas atitinkančias projektuojamą vartotojo sąsają.
2. Panaudos atvejų ir veiklos diagramos yra eksportuojamos XMI formatu ir pateikiamos generatoriui.

3. Generatorius iš pateiktų panaudos atvejų ir veiklos diagramų sugeneruoja klasių diagramą ir grąžina ją XMI formatu.

4. Projektavimo įrankiu papildoma klasių diagrama komandomis bei apribojimais. Sukuriamos būsenų diagramos. Testavimo scenarijai aprašomi sekų diagramomis. Šios diagramos yra eksportuojamos XMI formatu ir pateikiamos generatoriui.

5. Generatorius iš pateiktų diagramų sugeneruoja navigacijos grafą bei scenarijus.

Lango tikrinimo pagal būsenas problemai išspręsti reikia praplėsti lango aprašymą. Tai plačiau aptarsime kitame skyrelyje

3.4. Grafinės vartotojo sąsajos specifikavimas panaudos atvejų ir veiklos diagramomis

Grafinės vartotojo sąsajos specifikavimui siūloma naudoti panaudos atvejų ir veiklos diagramas. Panaudos atvejų diagramomis aprašant ryšius tarp programinių langų. Panaudos atvejis atitinka programos langą. Ryšys <<include>> galimas perėjimas iš vieno lango į kitą.

Metodo taikymo žingsniai

a) Pirmiausiai, naudojant panaudos atvejų diagramą, padaroma laisvos formos aukšto lygio sistemos aprašymas. Į šia diagramą įeina aktoriai ir pagrindiniai panaudos atvejai.

b) Antra, kiekvieno panaudos atvejo elgsena, besinaudojant pateiktais apribojimais, aprašomas veiklos diagrama. Taip gaunamas formalesnė diagrama.

c) Panaudos atvejų ir veiklos diagramos yra transformuojamos į klasių diagramą.

d) Iš sugeneruotos klasių, bei ankščiau sukurtos panaudos atvejų diagramos generuojamas navigacijos grafas.

e) Generuoti kodo fragmentus iš prieš tai sugeneruotos klasių diagramos. Šie kodo fragmentai gali būti grafinės vartotojo sąsajos komponentų prototipai.

Vartotojo sąsajos projektavimo taisyklės pateiktos 1.6 skyriuje. Formalus metodo aprašas pateiktas [8].

3.5. Grafinės vartotojo sąsajos specifikacijos transformavimas į klasių diagramą

Iš sukurtos vartotojo sąsajos specifikacijos būtina sugeneruoti klasių diagramą. Tam reikalinga papildoma programinė įranga. Grafinės vartotojo sąsajos specifikacija kuriama projektavimo priemonėmis (*case tools*). Sukurta specifikacija turi būti eksportuota XMI formatu į XML failą. Papildoma programinė įranga turi atitikti tokius kriterijus:

- Priimti UML diagramas XMI formatu.
- Transformuoti panaudos atvejų ir veiklos diagramas pagal tam tikras taisykles į klasių diagramą.
- Išsaugoti gautą klasių diagramą XMI formatu, kad projektavimo priemonės galėtų suprasti.

Transformacija atliekama pagal tokias taisykles:

1. Nagrinėjama panaudos atvejų diagrama.
2. Kiekvienam aktoriui ir panaudos atvejui sukurti to paties vardo klases kurios paveldi lango klasę. Klasėms sukurtoms iš aktorių pridedamas <<app>> stereotipas, o klasėms sukurtoms iš panaudos atvejų pridedamas <<form>> stereotipas.
3. Surasti apibendrinimo ryšius tarp panaudos atvejų ir apibendrinimo ryšius tarp aktorių. Sudėti apibendrinimo ryšius tarp atitinkamų klasių.
4. Kiekvienam panaudos atvejui nagrinėti atitinkamas veiklos diagramas.
5. Rasti visas būsenas ir perėjimus turinčius stereotipus. Sukurti klasėse laukus, kurių tipas - stereotipo pavadinimus, o vardas – atitinkamai būsenos arba perėjimo vardas.
6. Jei yra nenagrinėtų panaudos atvejų grįžti į 4 puntą.

Tokiu būdu sukuriama klasių diagrama kurioje klasės turinčios stereotipą <<form>> atitinka programos langus, o klasė turinti stereotipą <<app>> atitinka programos startavimo klasę, o visos kitos klasės atitinkančios formoje esančius grafinius vartotojo sąsajos komponentus. Dar viena svarbi klasė esanti klasių diagramoje – Form (pastaba: priklausomai nuo platformos ji gali vadintis ir kitaip, Java aplinkoje *JApplet*, .NET aplinkoje "*Form*"). Ši klasė turi du metodus, kuriuos pavedi visos kitos klasės. Šie metodai skirti parodyti kada forma yra iškviečiama ir parodoma ir kada uždaroma. Formos uždarymo metodas pažymimas stereotipu <<showForm>>, formos uždarymo metodas - <<closeForm>>. Ši informacija bus vėliau reikalinga nagrinėjant sekų diagramą.

3.6. Klasių diagramos papildymas komandomis ir apribojimais

Navigacijos grafo viršūnių generavimui iš klasių diagramos būtina papildyti klasių aprašymą komandomis ir apribojimais. Komandos aprašomos kaip metodai klasių diagramoje. Metodai žymintys lango komandas pažymimi stereotipu <<cmd>>.

Langų elementų apribojimams surašyti tinka OCL kalba, pvz.:

```
context someWindow inv:
self.someGUIElemetn.someProperty <= value
```

Lango būsena gali kisti nuo vartotojo atliekamų veiksmų bei norimos verslo logikos. Kiekvieno lango būsenų kitimui aprašyti yra naudojama būsenų diagrama. Svarbi sąlyga – perėjimams tarp būsenų naudoti lango komandas. Tam parašyti naudojama OCL funkcija *inOclState(būsena),* pvz.:

```
context someWindow inv:
oclInState(someState) implies someGUIElement.somePropert = value
```

OCL išraiškoms įvertinti reikalingas OCL analizatorius [13].

3.7. Navigacijos grafo ir scenarijaus aprašymų praplėtimas

Naudojant 1.5 skyriuje pateiktą navigacijos grafo ir scenarijų aprašymą nėra galimybės patikrinti kokia lango būsena turi būti kiekviename žingsnyje. Šiai problemai spręsti siūlau praplėsti navigacijos grafo ir scenarijų aprašymus. Pirmiausia siūloma praplėsti apribojimų grafiniams vartotojo sąsajos elementams aprašymą.

Esamas apribojimų aprašymo formatas:

```
<constraint>
<constraintid>...</ constraintid >
<type>...</type>
<value>...</value>
</constraint>
```

Kiekvieną apribojimą vienareikšmiškai nustato unikalus kodas – apribojimo ID, kuris žymimas etiketėmis `<constraintid>` ir `</constraintid >`. Apribojimo tipas žymimas etiketėmis `<type>` ir `</type>`. Apribojimo reikšmė žymima etiketėmis `<value>` ir `</value>`. Į kiekvieną apribojimo aprašymą galima įdėti etiketę `<stateid>` ir `</stateid>`. Taip bus aišku kuriai lango būsenai esant bus taikomas kuris apribojimas. Jei langas turi tik vieną būseną, arba apribojimas nesikeičia keičiantis būsenoms tada `<stateid>` turi būti paliekamas tuščias taip nurodant, kad apribojimas taikomas visais atvejais.

Naujas apribojimo aprašymo formatas:

```
<constraint>
<constraintid>...</ constraintid >
```

```
<type>...</type>
<value>...</value>
<stateId>...</stateid>
</constraint>
```

Norint tikrinti lango elementus pagal būsenas reikia žinoti kokia kiekviename žingsnyje turi būti lango būseną. Todėl reikia praplėsti ir scenarijaus žingsnio aprašymo formatą.

Esamas scenarijaus žingsnio aprašymo formatas:

```
<step>
      <index>...</index>
      <nodeid>...</nodeid>
      <cmdid>...</cmdid>
      <paramlist>
      <param>
                  <paramid>...</paramid>
                  <value>...</value>
      </param>
      ...
      </paramlist>
</step>
```

Kur kiekvienas žingsnis žymimas etiketėmis `<step>` ir `</step>`. Kiekvienas scenarijaus žingsnis yra indeksuojamas. Indeksavimui žymėti naudojamos etiketės `<index>` ir `</index>`. Žingsniui nurodomi mazgo/lango ID, žymimas etiketėmis `<nodeid>` ir `<nodeid>`, mazgo komandos ID, žymimas etiketėmis `<cmdid>` ir `</cmdid>`, komandai perduodamų parametrų sąrašas, žymimas etiketėmis `<paramlist>` ir `</paramlist>`. Kiekvienas parametrui žymėti naudojamos etiketės `<param>` ir `</param>`. Parametrui nurodomas ID, žymimas etiketėmis `<paramid>` ir `</paramid>`, ir reikšmė, žymima etiketėmis `<value>` ir `</value>`.Taigi būtina pateikti ne tik koks langas turi būti matomas, bet ir kokioje tas langas būsenoje. Tam reikia pridėti etiketę `<stateid>` ir `</stateid>`.

3.8. Navigacijos grafo generavimas iš klasių diagramos

Navigacijos grafo generavimui pasitelkiamos ankstesniuose žingsniuose sukurtos panaudos atvejų diagrama ir klasių diagrama, būsenų diagrama, bei klasių diagramoje esantis OCL apribojimų aprašymas. Panaudos atvejų diagrama naudojama langų hierarchijai aprašyti. Iš klasių diagramos klasių generuojamos grafo viršūnių aprašymai. Iš būsenų diagramos gaunamas grafo mazgai būsenų aprašymas. OCL apribojimų pagalba aprašomi grafo mazgai

elementų apribojimai. Lentelėje 3.1 pateikiamas navigacijos grafo elementų atitikimas UML diagramų elementams. Aprašysime tik terminalines etiketes, tas kuriose laikoma informacija.

Lentelė 3.1 Navigacijos grafo XML etikečių atitikimas UML diagramų elementams.

Navigacijos grafo elementas	UML diagrama	Iš kur imama reikšmė
<graphname>	Panaudos atvejų	Diagramos pavadinimas
<nodeid>	-	Generuojamas
<nodename>	Klasių diagrama	Klasės turinčios stereotipą <<form>> vardas.
<stateid>	OCL aprašymas/Būsenų diagrama	Generuojamas
<statmename>	Būsenų diagrama	Būsenos pavadinimas
<itemid>	-	Generuojamas
<itemname>	Klasių diagrama	Klasės, atitinkančios grafinės vartotojo sąsajos langą, elementas.
<type>	Klasių diagrama	Elemento tipas aprašytas klasių diagramoje
<defaultvalue>	Klasių diagrama	Pradinė reikšmė aprašyta klasių diagramoje.
<constraintid>	-	Generuojama
<type>	OCL aprašymas	Priklausomai nuo apribojimo tipo generuojama reikšmė.
<value>	OCL aprašymas	Iš OCL aprašymo gaunama reikšmė.
<cmdid>	-	Generuojamas
<cmdname>	Klasių diagrama	Metodo, turinčio stereotipą <<cmd>>, pavadinimas
<paramid>	-	Generuojamas
<paramname>	Klasių diagrama	Metodo parametro pavadinimas
<type>	Klasių diagrama	Metodo parametro tipas

3.9. Scenarijų generavimas iš sekų diagramos

Kaip aptarėme 1.6 skyrelyje, vartotojo veiksmų sekai aprašyti geriausiai tinka UML sekų (*sequence*) diagrama. Vartotojo veiksmai aprašomi kaip eilė operacijų, aprašytų klasių diagramoje, iškvietimų. Kiekvienas scenarijaus žingsnis atitinka metodo kvietimą sekų diagramoje. Tačiau tik tų metodų kurie klasių diagramoje yra pažymėti stereotipu `<<cmd>>` arba `<<closeForm>>`. Būtina stebėti kiekvieną formą dalyvaujančią sekų diagramoje ir pagal kviečiamus metodus registruoti būsenų pasikeitimą. Generuojant scenarijaus skriptą būtina saugoti kiekvienos formos esamą būseną po kiekvieno metodo kvietimo. Šios būsenos surašomos į scenarijaus skriptą kaip esamos formos būsenos. Lentelėje 3.2 pateikiamas scenarijaus XML etikečių atitikimas UML diagramų elementams.

Lentelė 3.2 Scenarijaus skripto XML etikečių atitikimas UML diagramų elementams.

Navigacijos grafo elementas	UML diagrama	Iš kur imama reikšmė
`<scnrId>`	-	Generuojamas
`<scnrname>`	Sekų diagrama	Sekų diagramos pavadinimas
`<index>`	-	Generuojamas
`<nodeid>`	Sekų diagrama	Formos, į kurią ateina įvykio kvietimas, id.
`<cmdid>`	Sekų diagrama	Metodo, atitinkančio įvykio kvietimą sekų diagramoje, id
`<paramid>`	Sekų diagrama	Metodo, atitinkančio įvykio kvietimą sekų diagramoje, parametro id
`<value>`	Sekų diagrama	Parametro nurodyto įvykio kvietime reikšmė

Pagal šią metodiką sugeneruotuose skriptuose yra pakankamai informacijos patikrinti lango būseną. Žinant kokia turi būti būsena ir turint būsenos apribojimus galima tikrinti grafinius komponentus ir nustatyti ar jie pavaizduoti gerai. Kokia turi būti būsena mes žinome iš scenarijaus skripto žingsnio aprašymo, o lango apribojimus žinome iš navigacijos grafo viršūnės aprašymo.

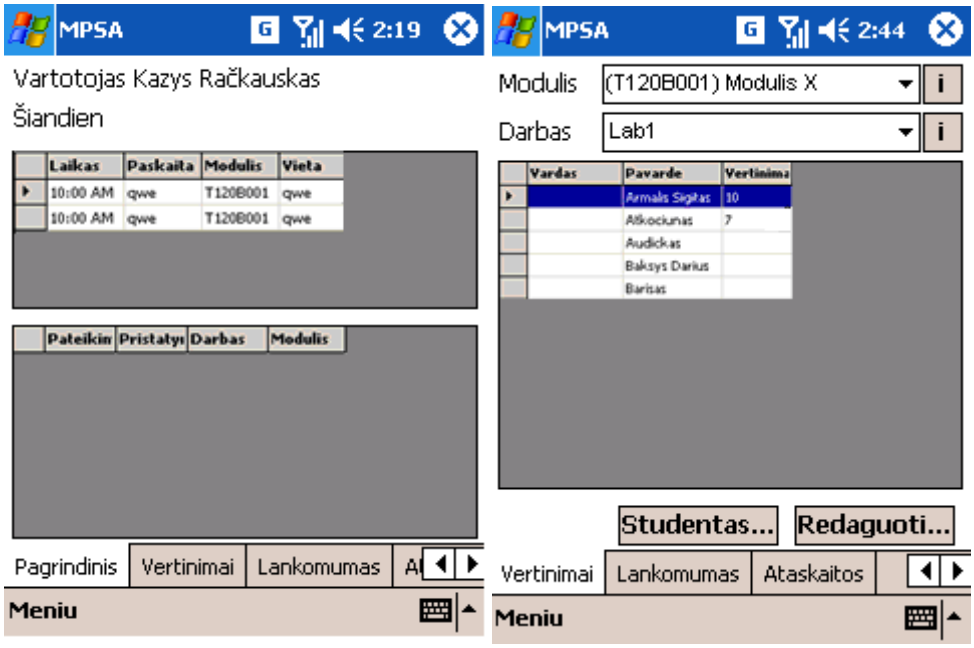
4. TESTAVIMO SKRIPTŲ GENERAVIMAS

Šioje dalyje pateiksime 3 skyriuje aptartų metodų eksperimentus. Eksperimentai buvo atliekami su „Mobilaus dėstytojo programine įranga“ (žiūrėti 2 skyrių). Sistemos dalis eksperimentui buvo pasirenkama pagal tokius kriterijus:

- Turi būti navigacija tarp langų.
- Nuo vartotojo veiksmų turi kisti bent vieno lango būsenos. Kintant būsenoms turi kisti apribojimai grafiniams elementams.

4.1. Nagrinėjamo grafinės vartotojo sąsajos aprašymas:

Vartotojo pagrindinis langas turi meniu iš kurio gali iškviesti kitus langus. Šis langas sudarytas iš keturių lapų (*tabs*). Lapai atitinka skirtingas lango būsenas. Esant skirtingoms būsenoms matomi skirtingi grafinės vartotojo sąsajos elementai. Lango vaizdas pateiktas žemiau esančiame paveiksle (4.1 pav.)



4.1 pav. Programos langų pavyzdžiai

Langas turi trijų punktų meniu:

Darbo pabaiga – uždaroma programa.

Sinchronizuoti – atidaromas programos sinchronizacijos langas.

Nustatymai - atidaromas programos nustatymų langas.

Šiame lange yra keturi lapai (*tabs*):

Pagrindinis – rodomi esamos dienos paskaitų ir laboratorinių darbų laikai.

Vertinimai – rezultatų rodymo/surašymo langas.

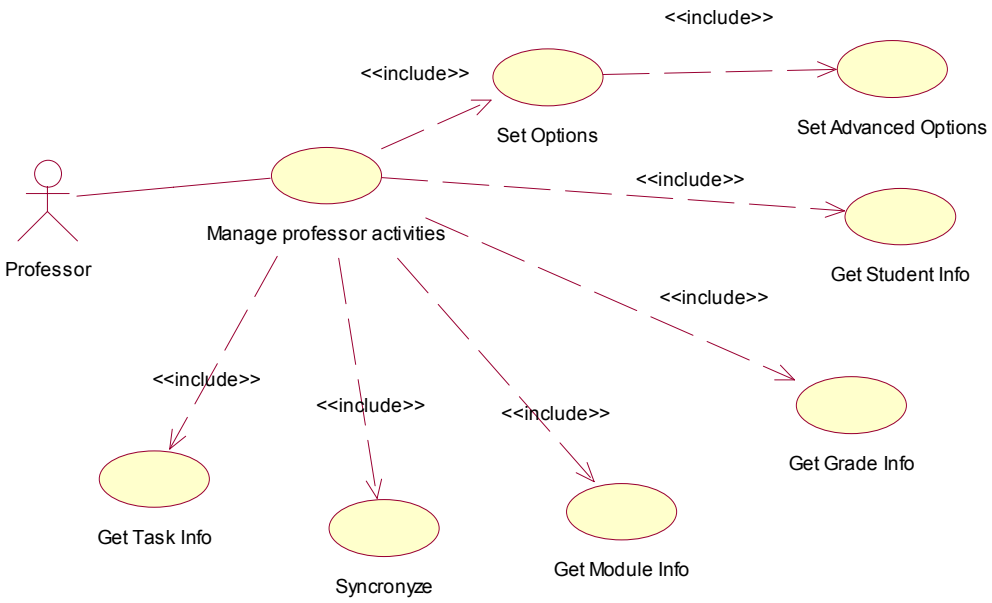
Lankomumas – lankomumo rodymo/surašymo langas.

Ataskaitos – ataskaitų langas.

Iš kai kurių lapų, paspaudus mygtukus galima patekti į naujus langus.

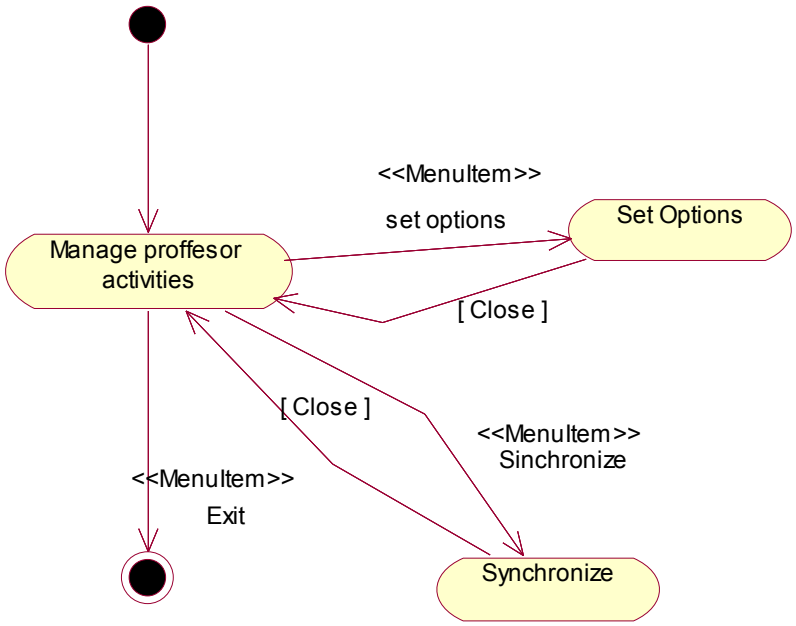
4.2. Grafinės vartotojos sąsajos specifikavimas

Programos navigacijos schema aprašoma panaudos atvejų diagrama 3.4 skyriuje pateiktu metodu. Programos langų navigacijos schema pateikta 4.2 paveiksle.



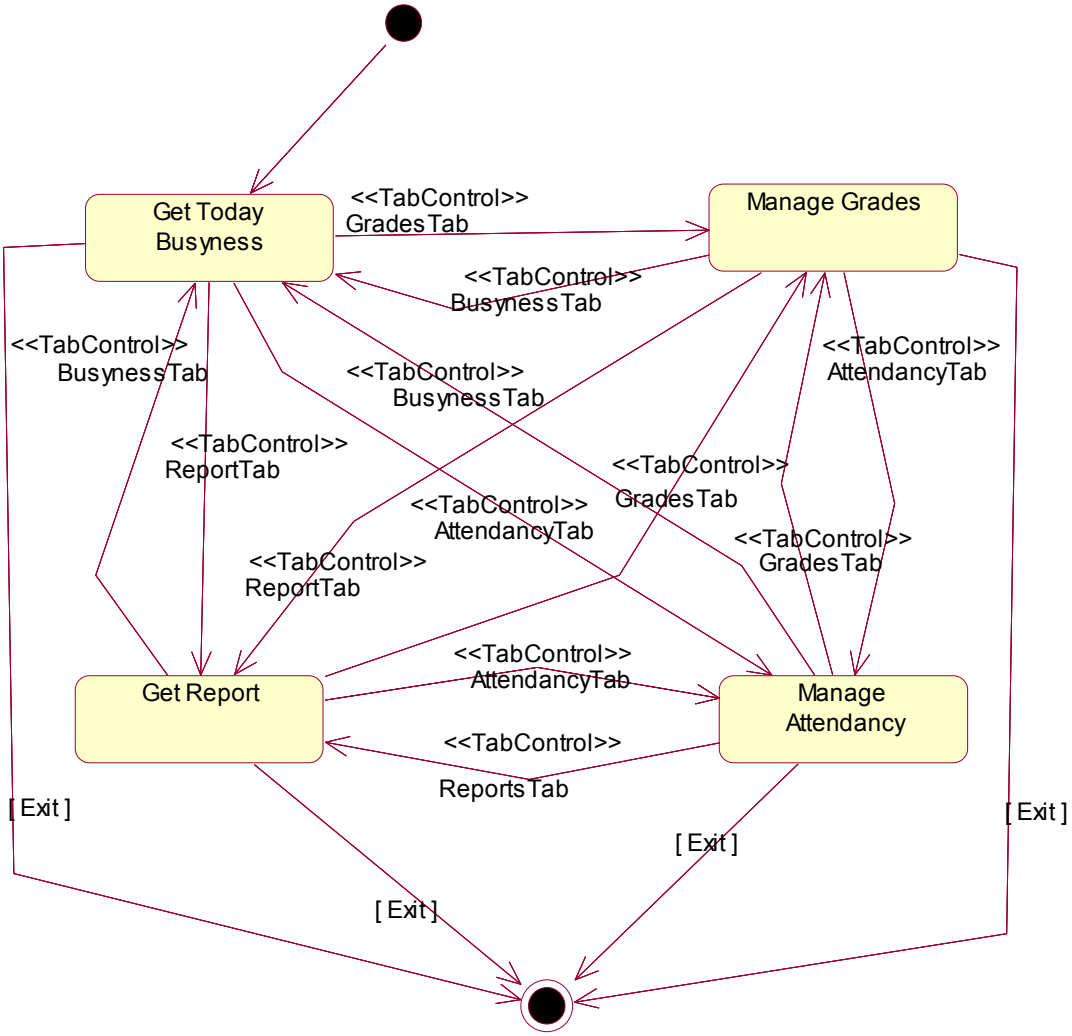
4.2 pav. Programos langų navigacijos schema.

Sumodeliuosime lango „*Manage professor activities*“ elgseną veiklos diagramomis.Šią formą menamai galime suskirstyti į dvi dalis – veiksmai su meniu ir veiksmai lango lapuose (*tabs*) esančiais elementais. Šių dviejų dalių funkcionalumas nepersikerta ir yra vienas nuo kito nepriklausomi. Lango navigaciją po meniu elgsena pateikta 4.3 paveiksle.



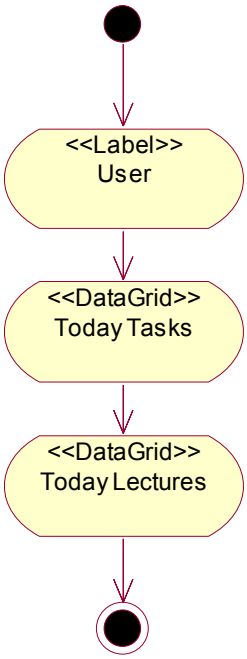
4.3 pav. Lango „*Manage professor activities*“ navigacijos po menu elgsena.

Kiekvieno lango lapo (*tab*) funkcionalumas pasiekiamas atskirai, todėl kiekvieno jų elgsena bus aprašyta atskiroje diagramoje. Bendra lango „*Manage professor activities*“ elgsena pateikta žemiau esančioje diagramoje (4.4 pav.). Šitoje diagramoje parodoma kaip keičiasi būsena šokinėjant per lapus (*tabs*).



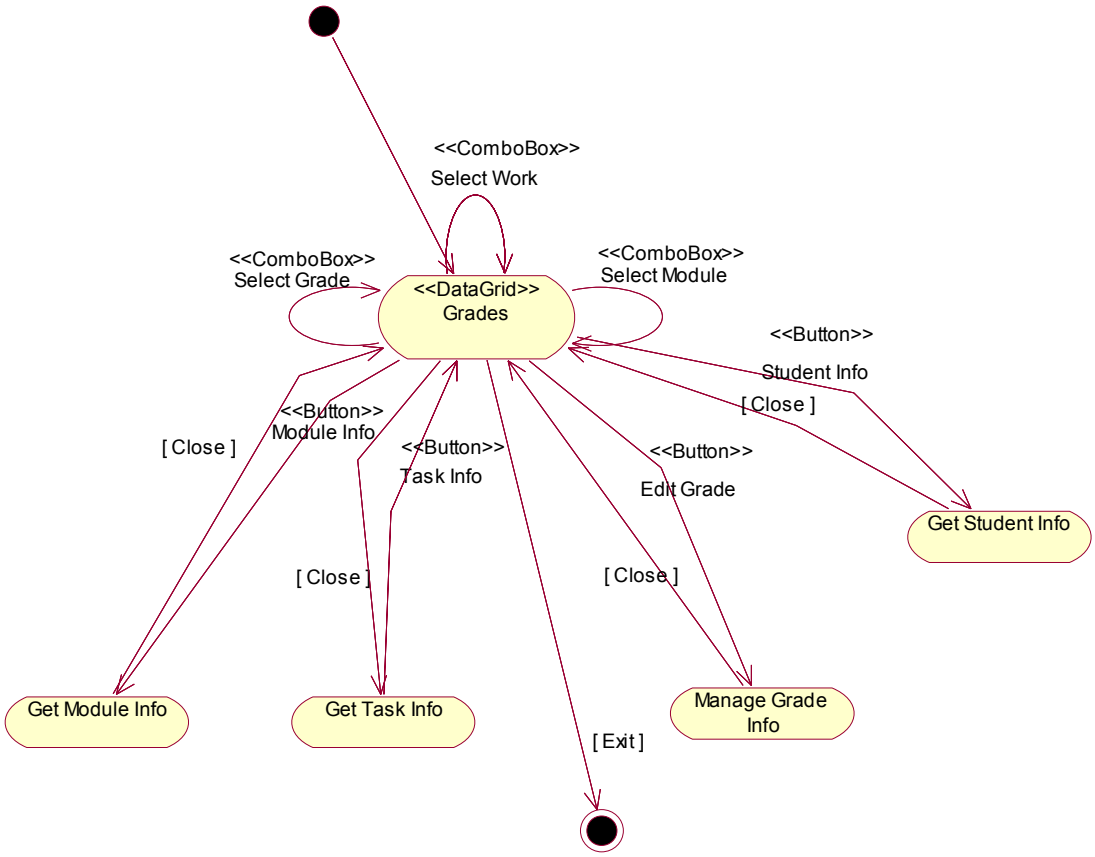
4.4 pav. Lango "*Manage professors activities*" elgsenos aprašymas

Kiekvieno lapo elgsena aprašoma atskirose diagramose. Lape „*Get Today Busyness*" duomenys pateikiami užkraunant langą (4.5 pav).



4.5 pav. Lapo "*Get Today Busyness*" elgsenos aprašymas

Sudėtingesnis yra „*Manage Grades*" lapas. Iš šio lango galima patekti į „*Get Student Info*", „*Manage Grade Info*", „*Get Module Info*" ir „*Get Task Info*". Informacija šioje formoje įvedama išsiskleidžiančiais sąrašais (*combo box)*. Šioje formoje įvedama tokia informacija – modulis, darbas ir pažymys. Šio lapo (*tab*) elgseną vaizduojanti diagrama pateikta 4.6 paveiksle.



4.6 pav. Lapo "*Manage grade Info*" elgsenos aprašymas

Kitų langų elgsenos apsirašomos panašiai todėl čia nedetalizuosime.

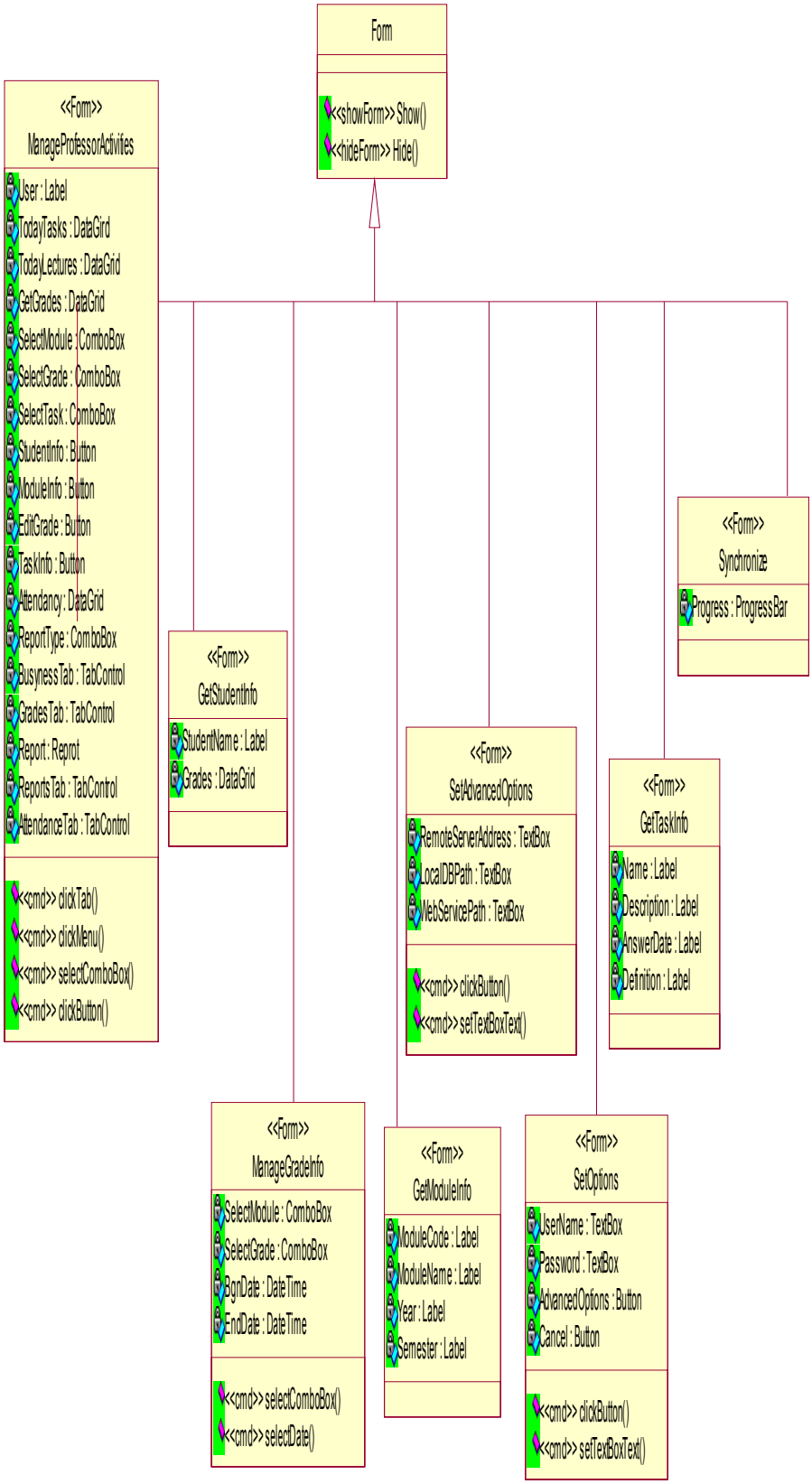
Klasių diagramos generavimas

Klasių diagrama generuojama iš sukurtų panaudos atvejų ir veiklos diagramų. Generavimas vyksta tokiu algoritmu:

1. Einamuoju pažymimas pirmas nenagrinėtas panaudos atvejis turintis asociatyvinį ryšį su aktoriumi.
2. Sukuriama klasė tokiu pat pavadinimu kaip ir einamasis panaudos atvejis. Klasė pažymima stereotipu <<form>>.
3. Sukurta klasė pažymima einamąja.
4. Panaudos atvejį detalizuojanti veiklos diagrama pažymima einamąja.
5. Iš einamosios veiklos diagramos išskiriamos visos veiklos/būsenos bei perėjimai (*transition)* pažymėti stereotipais.
6. Pagal kiekvieną išskirtą būseną ir perėjimą einamojoje klasėje sukuriamas atributas. Atributas sukuriamas pagal tokias taisykles:

    a. Atributo tipui suteikiamas stereotipo vardas.
    b. Atributo vardui suteikiamas būsenos/perėjimo vardas.
7. Iš einamosios veiklos diagramos išskiriamos stereotipu nepažymėtos veiklos.
8. Kiekviena veikla, kurių pavadinimai nesutampa su nei vienu panaudos atveju, paeiliui pažymima einamąja ir vykdomas 5 – 8 punktai.
9. Visom veiklom, kurių pavadinimas sutampa su kažkuriuo panaudos atveju sukuriamos klasės tokiu pat pavadinimu kaip ir veikla. Klasė pažymima stereotipu <<form>>. Sukurta klasė pažymima einamąja. Paminėtos veiklos detalizuojanti veiklos diagrama pažymima einamąja ir vykdomi 5-9 punktai.
10. Algoritamas baigiamas kai išnagrinėjamos visos veiklos diagramos.

Tiriamajai vartotojo sąsajai sugeneruota klasių diagrama pateikta 4.7 paveiksle.



4.7. pav. Sugeneruota klasių diagrama.

Kitas žingsnis – pilnai aprašyti kiekvieną formą. Pilnas formos aprašymas susideda iš:

- Komandų aprašymu.
- Būsenų diagramos, jei forma turi daugiau nei vieną būseną.
- OCL apribojimų formos elementams aprašymo.

Komandų aprašymas

Aprašysime formą *„ManageProfessorsActivities“*. Formoje galima atlikti tokius veiksmus:

- Pasirinkti kitą formos lapą (*tab*).
- Paspausti mygtuką.
- Iš iškrentančio sąrašo (*combo box*) pasirinkti reikšmę.
- Paspausti meniu punktą.

Šiems veiksmams įvykdyti sukurtos tokios komandos:

clickTab(string tabName) – imituojamas formos lapo (*tab*), kurio vardas *tabName*, pasirinkimas.

clickMenu(string menuItemName) – imituojamas meniu punkto, kurio vardas *menuItemName*, paspaudimas.

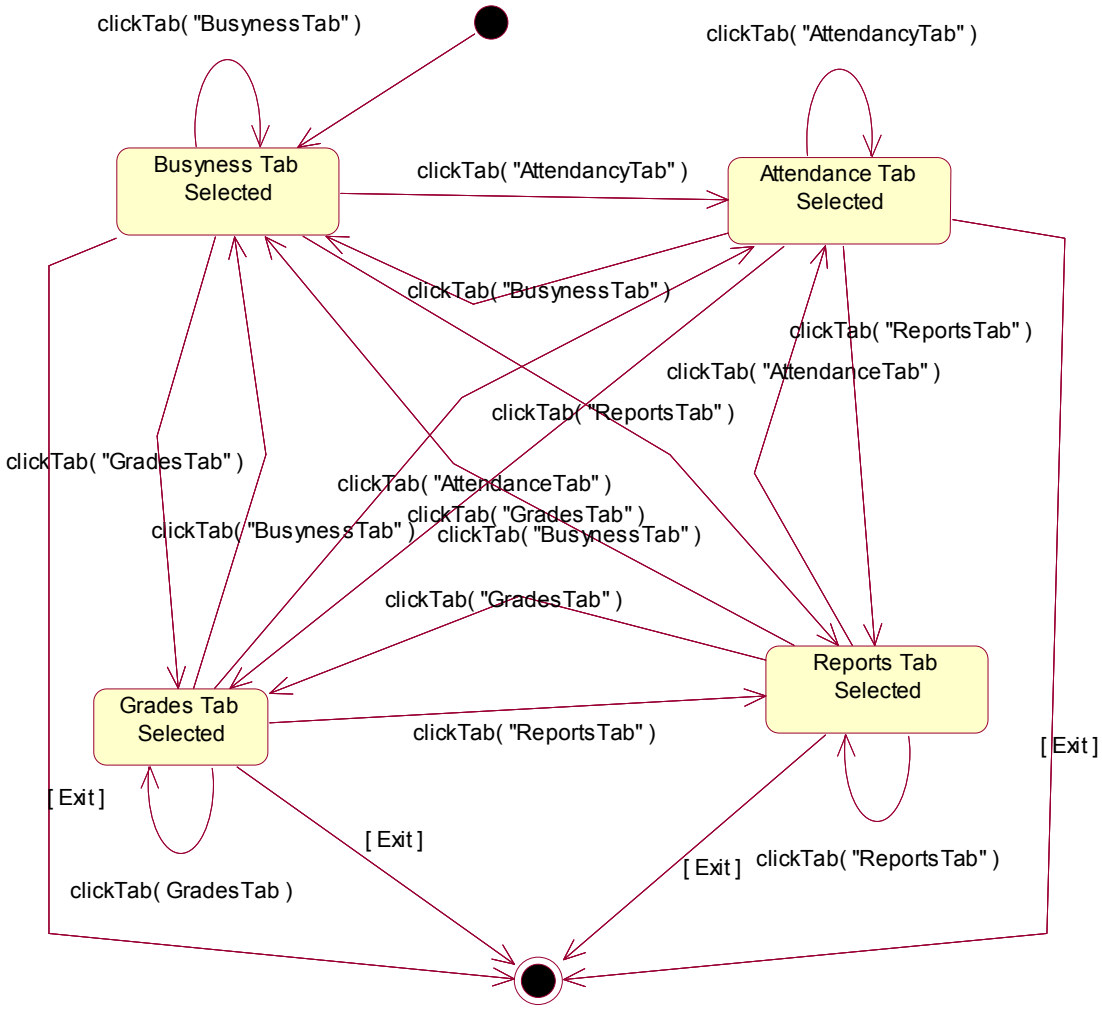
selectComboBox(string comboBoxName, string value) – imituojamas sąrašo elemento, kurio vardas *comboBoxName,* sąrašo punkto, kurio reikšmė – *value,* pasirinkimas.

Būsenų diagrama

Forma *„ManageProfessorsActivities“* turi 4 būsenas:

- paspaustas lapas (*tab*) „Pagrindinis“,
- paspaustas lapas (*tab*) „Vertinimai“,
- paspaustas lapas (*tab*) „Lankomumas“,
- paspaustas lapas (*tab*) „Ataskaita“.

Formos *„ManageProfessorsActivities“* būsenų diagrama pateikta 4.8 paveiksle.



4.8. pav. Formos „*ManageProfessorsActivities*“ būsenų diagrama.

OCL apribojimų surašymas

„*ManageProfessorsActivities*“ formos būsenų apribojimas aprašomi OCL sintakse. Aprašoma kurie lementai turi matytis ir kurie ne, esant konkrečiai būsenai.

```
context ManageProfessorsActivities
inv: oclInState(BusynessTabSelected) implies BusynessTab.Visible = true
inv: oclInState(BusynessTabSelected) implies ReportsTab.Visible = false
inv: oclInState(BusynessTabSelected) implies AttendancyTab.Visible = false
inv: oclInState(BusynessTabSelected) implies GradesTab.Visible = false

inv: oclInState(ReportsTabSelected) implies BusynessTab.Visible = false
inv: oclInState(ReportsTabSelected) implies ReportsTab.Visible = false
inv: oclInState(ReportsTabSelected) implies AttendancyTab.Visible = false
inv: oclInState(ReportsTabSelected) implies GradesTab.Visible = false

inv: oclInState(AttendenceTabSelected) implies BusynessTab.Visible = true
```

```
inv: oclInState(AttendenceTabSelected) implies ReportsTab.Visible = false
inv: oclInState(AttendenceTabSelected) implies AttendancyTab.Visible= false
inv: oclInState(AttendenceTabSelected) implies GradesTab.Visible = false

inv: oclInState(GradesTabSelected) implies BusynessTab.Visible = true
inv: oclInState(GradesTabSelected) implies ReportsTab.Visible = false
inv: oclInState(GradesTabSelected) implies AttendancyTab.Visible = false
inv: oclInState(GradesTabSelected) implies GradesTab.Visible = false
```

4.3. Formos skripto generavimas

Formos scriptas generuojamas iš klasių diagramos, ją aprašančios būsenų diagramos bei ją aprašančio OCL kodo. Veiksmai atliekami tokia:

1. Sugeneruojamas identifikatorius (*id*) ir įrašomas į formos aprašą.
2. Į mazgo vardo lauką (*nodename*) įrašomas klasės pavadinimas.
3. Generuojamas būsenų sąrašas jei nagrinėjama klasė turi ją aprašančią būsenų diagramą.
    a. Sugeneruojamas identifikatorius (*id*).
    b. Į būsenos pavadinimo (*statename*) elementą įrašomas būsenos pavadinimas iš būsenų diagramos.
4. Jei nagrinėjama klasė turi atributų, iš jų generuojamas formos elementų sąrašas. Suk kiekvienu atributu atliekami tokie veiksmai:
    a. Sugeneruojamas elemento identifikatorius (*id*).
    b. Elemento vardui (*itemname*) priskiriamas nagrinėjamo atributo vardas.
    c. Elemento tipui (*type*) priskiriamas nagrinėjamo atributo tipas.
    d. Pradinei reikšmei (*defaultvalue*) priskiriama atributo pradinė reikšmė.
    e. Apribojimų sąrašas gaunamas OCL analizatoriaus pagalba iš OCL apribojimų aprašymo.
        i. Nagrinėjama kiekviena būsena. Jei nagrinėjamas elementas yra būsenos aprašyme, generuojamas apribojimas.
5. Jei nagrinėjama klasė turi operacijų, iš jų generuojamas formos komandų sąrašas. Suk kiekviena operacija atliekami tokie veiksmai:
    a. Sugeneruojamas elemento identifikatorius (*id*).
    b. Komandos vardui (*cmdname*) priskiriamas nagrinėjamos operacijos vardas.
    c. Parametrų sąrašas gaunamas iš operacijos parametrų sąrašo. Su kiekvienu operacijos parametru atliekami tokie veiksmai:

i. Nagrinėjamas kiekviena parametras. Sugeneruojamas parametro identifikatorius (*id*). Parametro vardui priskiriamas operacijos parametro vardas, parametro tipui priskiriamas operacijos parametro tipas.

Iš formos „*ManageProfessorActivities*" sugeneruotas kodas pateiktas priede A.

4.4. Grafo generavimas

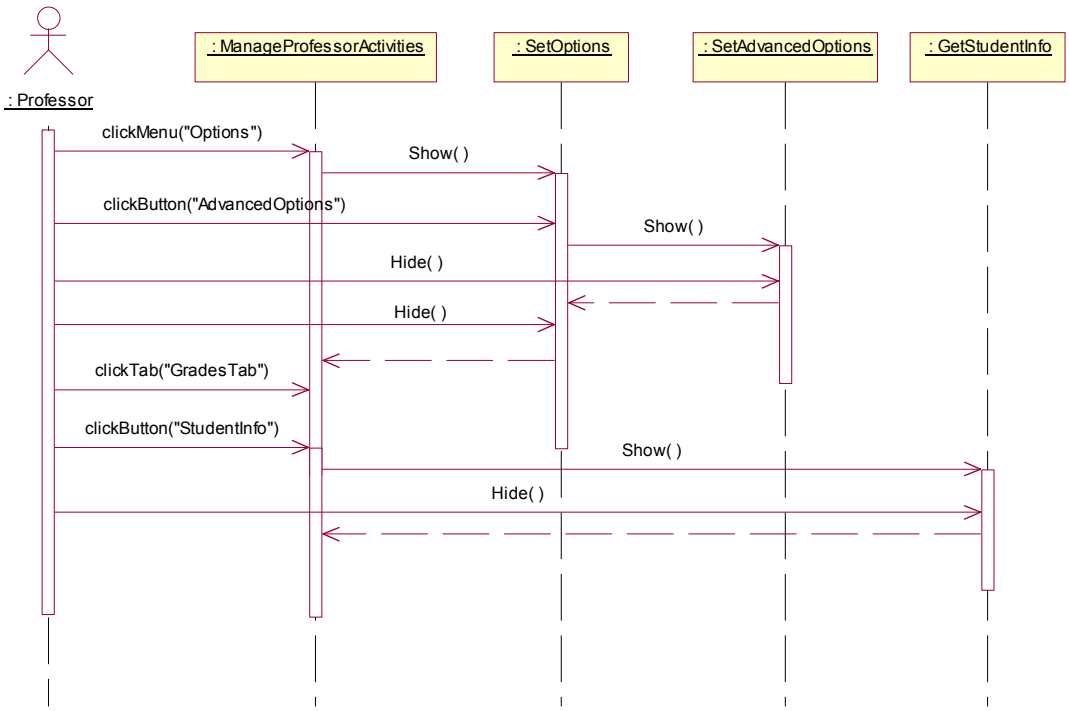
Navigacijos grafo ryšiai tarp mazgų generuojami iš panaudos atvejų diagramos. Grafo mazgai – sugeneruotas formos skriptas (žiūrėti skyrelį „Formos skripto generavimas"). Naudojamas paieškos gilyn algoritmas.

Grafas generuojamas pagal tokį algoritmą:

1. Pirmiausia nagrinėjami panaudos atvejai kurie turi asociatyvinį ryšį su aktoriumi.
2. Imamas pirmas nenagrinėtas panaudos atvejis turintis asociatyvinį ryšį su aktoriumi. Šis panaudos atvejis pažymimas einamuoju.
3. Į grafo mazgų sąrašą įrašomas einamąjį panaudos atveją atitinkančios formos skriptas.
4. Jei einamasis panaudos atvejis turi `<<include>>` ryšių su kitais panaudos atvejais – imamas pirmas nenagrinėtas panaudos atvejis turintis `<<include>>` ryšį su einamuoju panaudos atveju. Šis panaudos atvejis pažymimas einamuoju ir vykdomas 3 punktas. Jei nagrinėjamas panaudos atvejis neturi `<<include>>` ryšių, einamuoju panaudos atveju pažymimas tėvinis panaudos atvejis ir vykdomas 4 panaudos atvejis iš naujo. Jei tėvinio panaudos atvejo nėra – baigiamas grafo generavimas.

4.5. Scenarijų generavimas

Testavimo scenarijai generuojami iš sekų diagramos. Vaikščiojimo tarp langų sekų diagrama pateikta 4.10 paveiksle.



4.9. pav. Veiksmų seką aprašanti sekų diagrama.

Analizuojant sekų diagramą atliekami tokie veiksmai:

1. Pagal būsenų diagramas nustatomos sekų diagramoje dalyvaujančių langų pradinės būsenos. Šios būsenos nustatomos kaip einamosios langų būsenos.
2. Nagrinėjama iš viršaus į apačią po vieną komandos kvietimą. Imama pirma nenagrinėta komanda.
3. Komanda įtraukiama į scenarijaus žingsnį jei klasių diagramoje ji turi stereotipą `<<cmd>>` arba `<<hideForm>>`.
    a. Iš formos skripto gaunama komandos identifikacija (*id*) ir įrašoma į žingsnio aprašymą.
    b. Iš formos skripto gaunami komandos parametrų identifikacijos (*id*) ir įrašomi į žingsnio aprašymą.
    c. Iš komandos kvietimo aprašymo gaunamos parametrų reikšmės (*value*) ir įtraukiamos į žingsnio aprašymą.
    d. Iš formos skripto gaunama einamosios būsenos identifikacija (*id*) ir įrašoma žingsnio aprašyme.
4. Žingsnis įrašomas į scenarijų.
5. Patikrinama ar komandos kvietimas nepakeičia lango būsenos. Jei keičia, nauja būsena nustatoma einamąją.

6. Jei dar yra nenagrinėtų komandų grįžtame į punktą 2.

Iš 4.10 paveiksle pateiktos sekų diagramos sugeneruotas scenarijus pateiktas priede B.

5. IŠVADOS

1. Norint sėkmingai testuoti grafinę vartotojo sąsają būtina jau projektavimo metu numatyti testavimo priemones .
2. Automatizuotas testavimas leidžia atlikti regresinį testavimą.
3. Automatizuotas vienas testavimo aspektas nebūtinai pagerina testavimo efektyvumą, priešingai tai kartais gali net žymiai prailginti testavimo kaštus.
4. Specifikacija pagrįstas testavimo atvejų generavimas padidina automatinio testavimo efektyvumą ir praktiškumą.
5. UML pagrįstas vartotojo sąsajos specifikavimas suteikia priemones automatizuotam testavimo skriptų generavimui.
6. Generuojant testavimo skriptus iš specifikacijos išvengiama daug problemų kurios gali atsirasti dėl reikalavimų bei specifikacijos pakeitimo. Tereikia pakeisti specifikaciją ir pergeneruoti skriptus iš naujo.

6. LITERATŪRA

1. Finite State Machine [Interaktyvus]. [Žiūrėta 2007 05 14]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Finite_state_machine
2. Marlon V., Leduc J., Hasling B., Subramanyan R., Kazmeier J.. Automation of GUI Testing Using a Model-driven Approach // International Conference on Software Engineering Proceedings of the 2006 international workshop on Automation of software test.[Šangai, Kinija, 2006 Gegužės 3-3], p. 9-14.
3. Memon A.. An event-flow model of GUI-based applications for testing. Merilendo universitetas, Kompiuterių mokslo departamentas. [2007 sausio 02].
4. Handheld device [Interaktyvus]. [Žiūrėta 2007 05 14]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Mobile_device
5. Software testing [Interaktyvus]. [Žiūrėta 2007 05 14]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Software_testing
6. Test Automation [Interaktyvus]. [Žiūrėta 2007 05 14]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Test_automation
7. Model-based testing [Interaktyvus]. [Žiūrėta 2007 05 14]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Model-based_testing.
8. Designing GUI components from UML Use Cases // Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the [2005 Balandžio 4-7] p. 210-214.
9. Pinheiro da Silva P., Norman W. Paton. User Interface Modelingi n UMLi [Interaktyvus]. [Žiūrėta 2007 05 10]. Prieiga per Internetą: http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/mags/so/&toc=comp/mags/so/2003/04/s4toc.xml&DOI=10.1109/MS.2003.1207457
10. XMI. [Interaktyvus]. [Žiūrėta 2007 05 06]. Prieiga per Internetą: http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci213944,00.html
11. XML Metadata Interchange (XMI) Specification [Interaktyvus]. [Žiūrėta 2007 05 Prieiga per Internetą: http://www.omg.org/docs/formal/03-05-02.pdf
12. Extensible Markup Language [Interaktyvus]. [žiūrėta 2007-05-15]. Prieiga per Internetą: http://www.w3.org/XML/.

13. Tcherchango A., Analysis of the Metamodeling Semantics for OCL. Masters thesis, Dresden University of Technology, Department of Computer Science, 2007 Lapkritis.

14. White L, Almezen H. Generating test cases for GUI responsibilities using complete interaction sequences. Proceedings of the International Symposium on Software Reliability Engineering, 8–11 October 2000. IEEE Computer Society Press:
Piscataway, NJ, 2000; 110–121.

15. White L, Almezen H, Alzeidi N. User-based testing of GUI sequences and their interaction. Proceedings of the
International Symposium on Software Reliability Engineering, 8–11 November 2001. IEEE Computer Society Press:
Piscataway, NJ, 2001; 54–63.

16. Memon, A., Soffa, M., Pollack, M. Coverage Criteria for GUI Testing// 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9): tarptautinės konferencijos pranešimų medžiaga, [2001 m. spalio 14], p.

17. Chin. Mobile technologies and interactive applications [Interaktyvus]. [žiūrėta 2007-05-14]. Prieiga per internetą: http://www.chin.gc.ca/English/Digital_Content/Tip_Sheets/Wireless/glossary.html.

18. Weiss, S. Handheld usability. London: John Wiley & Sons Ltd, 2002. ISBN: 0-470-84446-9.

19. Berber, S. Automated Testing for Embedded Devices [Interaktyvus]. [žiūrėta 2007-03-15]. Prieiga per internetą: http://www.perftestplus.com.

20. Dustin, E. Lessons in Test Automation: A Manager's Guide to Avoiding Pitfalls When Automating Testing [Interaktyvus]. [žiūrėta 2007-05-14]. Prieiga per internetą: http://www.informit.com/articles/article.asp?p=21467&rl=1.

21. Herbert, M. The practical organization of automated software testing [Interaktyvus]. [žiūrėta 2007-05-01]. Prieiga per internetą: http://www.automated-testing.com/PATfinal.htm.

22. Linz, T., Daidl, M. How to automate testing of graphical user interfaces [Interaktyvus]. [žiūrėta 2007-02-19]. Prieiga per internetą: http://www.imbus.de/forschung/pie24306/gui/aquis-full_paper-1.3.shtml.

23. Zambelich, K. Totally data-driven automated testing [Interaktyvus]. [žiūrėta 2007-02-18]. Prieiga per internetą: http://www.sqa-test.com/articles.html.

24. Javamobiles. List of JVM for PDA [Interaktyvus]. [žiūrėta 2007-04-15]. Prieiga per internetą: http://www.javamobiles.com/jvm.html.

25. Clarke JM. Automated test generation from a behavioural model. Proceedings of Pacific Northwest Software Quality Conference, May 1998. PNSQC: Portland, OR, 1998.

26. Chow TS. Testing software design modelled by finite-state machines. IEEE Transactions on Software Engineering 1978; 4(3):178–187.

27. Esmelioglu S, Apfelbaum L. Automated test generation, execution, and reporting. Proceedings of Pacific Northwest

Software Quality Conference, October 1997. IEEE Press: Piscataway, NJ, 1997.

28. Bernhard PJ. A reduced test suite for protocol conformance testing. ACM Transactions on Software Engineering and Methodology 1994; 3(3):201–220.

29. Shehady RK, Siewiorek DP. A method to automate user interface testing using variable finite state machines. Proceedings of the 27th Annual International Symposium on Fault-Tolerant Computing (FTCS'97), June 1997. IEEE Computer Society Press: Piscataway, NJ, 1997; 80–88.

30. Fogel LJ, Owens AJ, Walsh MJ. Artificial intelligence through a simulation of evolution. *Biophysics and Cybernetic Systems: Proceedings of the 2nd Cybernetic Sciences Symposium*, Maxfield M, Callahan A, Fogel LJ (eds.). Spartan Books: Washington, DC, 1965; 131–155.

31. Fogel LJ, Owens AJ, Walsh MJ. *Artificial Intelligence Through Simulated volution*. Wiley: New York, 1966.

32. Ušaniov, A. Mobiliųjų įranginių grafinės vartotojo sąsajos automatizuotas testavimas// Magistro tezės, Kauno Technologijos Universitetas, Informatikos fakultetas, programų inžinerijos katedra. [Kaunas, 2005 m. birželio 11], p.

33. Ancona, M., Cazzola, W. Implementing the essence of reflection: a reflective run-time environment// Symposium on Applied Computing archive. Proceedings of the 2004 ACM symposium on Applied computing: tarptautinės konferencijos pranešimų medžiaga, [2004 m.], p. 1503-1507.

34. Satoh, I. A Testing Framework for Mobile Computing Software// IEEE Transactions on Software Engineering. ISSN 2003, p. 1112-1121.

35 Ušaniov, A., Packevičius, Š. Grafinės vartotojo sąsajos automatizuotas testavimas mobiliems įrenginiams su apribojimais// Informacinės Technologijos: konferencijos pranešimų medžiaga: tarptautinės konferencijos pranešimų medžiaga, [Kaunas, 2005 m. balandžio 29], p. 180-184.

7. TERMINŲ IR SANTRUMPŲ ŽODYNAS

#	Trumpinys	Apibrėžimas
1.	MSDE	Microsoft SQL Server 2000 Desktop Engine
2.	JRE	Java Runtime Environment
3.	PDA	Personal Digital Assistant
4.	DB	Database
5.	DBMS	Database Management System
6.	J2ME	Java Micro Edition
7.	JVM	Java Virtual Machine
8.	MIDP	Mobile Information Device Profile
9.	CLDC	Connected Limited Device Configuration
10.	XML	eXtensible Markup Language
11.	SAX	Simple API for XML
12.	DOM	Document Object Model
13.	XMI	XML Metadata Interchange
14.	OCL	Object Constraint Language
15.	API	Application programming interface
16.	Etiketė	XML node

8. PRIEDAI

A. Navigacijos grafo skriptas

```
<?xml version="1.0" encoding="utf-8" ?>
<graph>
    <graphname>profesoriaus grafas</graphname>
    <nodelist>
        <node>
            <nodeId>1</nodeId>
            <nodename>ManageProfessorActivities</nodename>
            <statelist>
                <state>
                    <stateid>1</stateid>
                    <stetename>BusynessTabSelected</stetename>
                    <description>Mantomas pagrindinis lapas</description>
                </state>
                <stete>
                    <steteid>2</steteid>
                    <statename>AttendanceTabSelected</statename>
                    <description>Matomas lankomumo skirtukas</description>
                </stete>
                <stete>
                    <steteid>3</steteid>
                    <statename>GradesTabSelected</statename>
                    <description>Matomas vertinimų skirtukas</description>
                </stete>
                <stete>
                    <steteid>4</steteid>
                    <statename>ReportsTabSelected</statename>
                    <description>Matomas ataskaitų skirtukas</description>
                </stete>
            </statelist>
            <itemlist>
                <item>
                    <itemid>1</itemid>
                    <itemname>BusynessTab</itemname>
                    <type>TabControl</type>
                    <defaultvalue></defaultvalue>
                    <constraintlist>
                        <constraint>
                            <constraintid>1</constraintid>
                            <stateid>1</stateid>
                            <type>Visible</type>
                            <value>true</value>
                        </constraint>
                        <constraint>
                            <constraintid>2</constraintid>
                            <stateid>2</stateid>
                            <type>Visible</type>
```

```
            <value>false</value>
        </constraint>
        <constraint>
            <constraintid>3</constraintid>
            <stateid>3</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
        <constraint>
            <constraintid>4</constraintid>
            <stateid>4</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
    </constraintlist>
</item>
<item>
    <itemid>2</itemid>
    <itemname>AttendanceTab</itemname>
    <type>TabControl</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
        <constraint>
            <constraintid>4</constraintid>
            <stateid>1</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
        <constraint>
            <constraintid>5</constraintid>
            <stateid>2</stateid>
            <type>Visible</type>
            <value>true</value>
        </constraint>
        <constraint>
            <constraintid>6</constraintid>
            <stateid>3</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
        <constraint>
            <constraintid>7</constraintid>
            <stateid>4</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
    </constraintlist>
</item>
<item>
    <itemid>3</itemid>
    <itemname>GradesTab</itemname>
    <type>TabControl</type>
```

```
    <defaultvalue></defaultvalue>
    <constraintlist>
        <constraint>
            <constraintid>8</constraintid>
            <stateid>1</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
        <constraint>
            <constraintid>9</constraintid>
            <stateid>2</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
        <constraint>
            <constraintid>10</constraintid>
            <stateid>3</stateid>
            <type>Visible</type>
            <value>true</value>
        </constraint>
        <constraint>
            <constraintid>11</constraintid>
            <stateid>4</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
    </constraintlist>
</item>
<item>
    <itemid>4</itemid>
    <itemname>ReportsTab</itemname>
    <type>TabControl</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
        <constraint>
            <constraintid>12</constraintid>
            <stateid>1</stateid>
            <type>Visible</type>
            <value>true</value>
        </constraint>
        <constraint>
            <constraintid>13</constraintid>
            <stateid>2</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
        <constraint>
            <constraintid>14</constraintid>
            <stateid>3</stateid>
            <type>Visible</type>
            <value>false</value>
        </constraint>
```

```
        <constraint>
            <constraintid>15</constraintid>
            <stateid>4</stateid>
            <type>Visible</type>
            <value>true</value>
        </constraint>
    </constraintlist>
</item>

<item>
    <itemid>6</itemid>
    <itemname>User</itemname>
    <type>Lable</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
</constraintlist>
</item>
<item>
    <itemid>7</itemid>
    <itemname>TodayTasks</itemname>
    <type>DataGrid</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
    <itemid>8</itemid>
    <itemname>TodayLectures</itemname>
    <type>DataGrid</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
    <itemid>9</itemid>
    <itemname>GetGrades</itemname>
    <type>DataGrid</type>
    <defaultvalue></defaultvalue>
    </constraintlist>
</item>
<item>
<itemid>10</itemid>
    <itemname>SelectModule</itemname>
    <type>ComboBox</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
<itemid>11</itemid>
    <itemname>SelectGrade</itemname>
    <type>ComboBox</type>
```

```
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
<itemid>12</itemid>
    <itemname>SelectTask</itemname>
    <type>ComboBox</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
<itemid>13</itemid>
    <itemname>StudentInfo</itemname>
    <type>Button</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
<itemid>14</itemid>
    <itemname>ModuleInfo</itemname>
    <type>Button</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
<itemid>16</itemid>
    <itemname>AttendancyGrid</itemname>
    <type>DataGrid</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
<itemid>17</itemid>
    <itemname>ReportType</itemname>
    <type>ComboBox</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
<item>
<itemid>18</itemid>
    <itemname>Report</itemname>
    <type>Report</type>
    <defaultvalue></defaultvalue>
    <constraintlist>
    </constraintlist>
</item>
```

```
    </itemlist>
    <cmdlist>
        <cmd>
            <cmdid>1</cmdid>
            <cmdname>AttendanceClick</cmdname>
            <paramlist></paramlist>
        </cmd>
        <cmd>
            <cmdid>2</cmdid>
            <cmdname>MainClick</cmdname>
            <paramlist></paramlist>
        </cmd>
        <cmd>
            <cmdid>3</cmdid>
            <cmdname>menuItemClick</cmdname>
            <paramlist>
                <param>
                    <paramid>1</paramid>
                    <paramname>menuItemName</paramname>
                    <type>String</type>
                </param>
            </paramlist>
        </cmd>
        <cmd>
            <cmdid>4</cmdid>
            <cmdname>close</cmdname>
            <paramlist></paramlist>
        </cmd>
    </cmdlist>
</node>
<node>
    <nodeid>11</nodeid>
    <nodename>Options</nodename>
    <statelist></statelist>
    <itemlist>
        <item>
            <itemid>1</itemid>
            <itemname>textBoxLogin</itemname>
            <itemtype>TextBox</itemtype>
            <defaultvalue></defaultvalue>
            <constraintlist>
                <constraint>
                    <constraintid>1</constraintid>
                    <type>maxTextLength</type>
                    <value>20</value>
                    <stateid></stateid>
                </constraint>
            </constraintlist>
        </item>
        <item>
            <itemid>2</itemid>
```

```
        <itemname>textBoxPassword</itemname>
        <itemtype>TextBox</itemtype>
        <defaultvalue></defaultvalue>
        <constraintlist>
            <constraint>
                <constraintid>1</constraintid>
                <type>maxTextLength</type>
                <value>20</value>
                <stateid></stateid>
            </constraint>
        </constraintlist>
    </item>
</itemlist>
<cmdlist>
    <cmd>
        <cmdId>1</cmdId>
        <cmdname>close</cmdname>
        <paramlist></paramlist>
    </cmd>
    <cmd>
        <cmdId>2</cmdId>
        <cmdname>clickTab</cmdname>
        <paramlist>
            <param>
                <paramid>1</paramid>
                <paramname>tabName</paramname>
                <type>String</type>
            <param>
        </paramlist>
    </cmd>
    <cmd>
        <cmdId>3</cmdId>
        <cmdname>clickMenu</cmdname>
        <paramlist>
            <param>
                <paramid>2</paramid>
                <paramname>menuItemName</paramname>
                <type>String</type>
            <param>
        </paramlist>
    </cmd>
    <cmd>
        <cmdId>4</cmdId>
        <cmdname>selectComboBox</cmdname>
        <paramlist>
            <param>
                <paramid>3</paramid>
                <paramname>comboBoxName</paramname>
                <type>String</type>
            <param>
            <param>
                <paramid>4</paramid>
```

```
                        <paramname>value</paramname>
                        <type>String</type>
                    <param>
                </paramlist>
            </cmd>
            <cmd>
                <cmdId>5</cmdId>
                <cmdname>clickButton</cmdname>
                <paramlist>
                    <param>
                        <paramid>5</paramid>
                        <paramname>buttonName</paramname>
                        <type>String</type>
                    <param>
                </paramlist>
            </cmd>
        </cmdlist>
    </node>
</nodelist>
</graph>
```

B. Scenarijaus skriptas

```
<scenario>
  <scnid>1</scnid>
  <scnrname>TikrintiBusenas</scnrname>
  <steplist>
    <step>
      <index>1</index>
      <nodeid>1</nodeid>
      <stateid>1</stateid>
      <cmdid>3</cmdid>
      <paramlist>
        <paramid>2</paramid>
        <value>Options</value>
      </paramlist>
    </step>
    <step>
      <index>2</index>
      <nodeid>11</nodeid>
      <stateid></stateid>
      <cmdid>7</cmdid>
      <paramlist>
        <pram>
          <paramid>6</paramid>
          <value>AdvancedOptions</value>
        </pram>
      </paramlist>
    </step>
    <step>
      <index>3</index>
      <nodeid>111</nodeid>
      <stateid></stateid>
      <cmdid>10</cmdid>
      <paramlist></paramlist>
    </step>
    <step>
      <index>4</index>
      <nodeid>11</nodeid>
      <stateid></stateid>
      <cmdid>12</cmdid>
      <paramlist></paramlist>
    </step>
    <step>
      <index>5</index>
      <nodeid>1</nodeid>
      <stateid>1</stateid>
      <cmdid>2</cmdid>
      <paramlist>
        <param>
          <paramid>1</paramid>
          <value>GradesTab</value>
        </param>
      </paramlist>
    </step>
    <step>
      <index>6</index>
      <nodeid>1</nodeid>
      <stateid>3</stateid>
      <cmdid>5</cmdid>
      <paramlist>
        <param>
          <paramid>5</paramid>
          <value>StudentInfo</value>
        </param>
      </paramlist>
    </step>
    <step>
      <index>7</index>
      <nodeid>16</nodeid>
      <stateid></stateid>
      <cmdid>8</cmdid>
      <paramlist></paramlist>
    </step>
  </steplist>
</scenario>
```