KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

EVALDAS BAGATAVIČIUS

MOBILIOS PROGRAMOS TRANSFORMAVIMAS IŠ VIENOS PLATFORMOS Į KITĄ

Magistro darbas

Vadovas

lekt. dr. Š. Packevičius

KAUNAS, 2013

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

EVALDAS BAGATAVIČIUS

MOBILIOS PROGRAMOS TRANSFORMAVIMAS IŠ VIENOS PLATFORMOS Į KITĄ

Magistro darbas

Vadovas

lekt. dr. Š. Packevičius

Recenzentas

doc. dr. R. Butkienė

KAUNAS, 2013

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLĖLIŲ SĄRAŠAS

SUMMARY

There is growth of mobile technologies and platforms providing for users so and developers of mobile applications need to take a larger market. There is some specificity of platforms, therefore developer needs a more knowledge or experts of mobile application developing where require a more resources, training, costs and it takes a time. One of the possible solutions to the problem, to make the tools which allow design and create mobile applications independent by platform keep the logic in design and development or testing phase. This is sufficient for developers to design or creates one mobile applications and using methods of Mobile Driven Architecture (MDA) and frameworks create transformations more applications many platforms.

In this research paper representing the tools developed based MDA to carry out transformations from Android to Windows Phone. To prove the importance of transformations performed research of transformation tools with certain comparison of metrics between the programs of separated implementation, these tools transformed programs and used universal tools like JavaScript or web-based software implementation.

SANTRAUKA

Mobilių technologijų populiarėjimas tarp vartotojų ir jų platformų įvairovė skatina mobilių programėlių kūrėjus užimti vis didesnę rinkos dalį. Kiekviena mobili platforma turi savo specifiką, todėl kūrėjams reikia vis daugiau žinių arba specialistų kuriant mobilias aplikacijas, tam reikalinga papildomų resursų, apmokymų,kaštų ir laiko. Vienas iš galimų problemos sprendimų, sukurti tam tikrus įrankius, kurie mobilių programėlių projektavimo ir kūrimo bei testavimo etape, leistų automatiškai suprojektuoti, suprogramuoti mobilias aplikacijas, nepriklausomai kokiai platformai išlaikant tos programėlės logiką. Tam pakaktų mobilių programų kūrėjams turėti vienai mobiliai platformai aprašytą modelį arba programėlę, ir iš jų remiantis MDA (*Model Dirven Architecture*) metodologijomis arba aprašytais karkasais atliktų transformacijas į reikiamą platformą.

Šiame darbe pateikimas MDA principais paremtos sukurtos priemonės , kurios, atlieka programų transformacijas iš Android į Windows Phone. Įrodant transformacijų svarbą, atliktas transformavimo priemonių tyrimas, įvedant tam tikras metrikas ir jų palyginimą tarp atskirai realizuotų programų, šių priemonių transformuotų programų ir naudojant universalias priemones kaip JavaScript arba žiniatinklio principu veikiančių programų.

1. ĮVADAS

Populiarėjant mobilioms technologijoms, yra poreikis turėti informaciją bei pramogas „delne“ su savimi. Tai rodo internete mobilių programėlių ir žaidimų poreikis bei jų pirkimas. Mobilių programėlių ir žaidimų kūrėjų tikslas – kuo daugiau vartotojui pasiūlyti paslaugų bei pramogų, bet susiduria su mobilių platformų įvairove. Kiekviena mobilioji platforma turi savo taikymo sritis (t. y. įrenginių gamintojus, kuriose įdiegiama platforma) ir tam tikrą vartotojų skaičių. Mobilių aplikacijų kūrėjai dažniausiai būna orientuoti į vieną platformą, norint išplėsti pardavimo rinką turi orientuotis į kitas platformas, kurios turi skirtingas taikymo metodikas. Mobilių programų kūrėjams reikia ieškoti būdų kaip mobilią programėlę pritaikyti įvairioms mobiliosioms aplinkoms.

Norima sukurti priemones ar įrankius, kurios naudotų bendrą programos modelį, pritaikytų kuriamą mobilią programėlę įvairioms platformoms, tinkamai išnaudotų tos platformos visas galimybes. Tai palengvintų mobilių programėlių gamintojams, kuriant vienai platformai ir išnaudojant jos galimybes, taikyti kitai platformai ir perimant tos platformos galimybes. Sistema bus svarbi mobilių programėlių ir žaidimų kūrėjams, kurie norės savo sukurtas programėles parduoti dideliai mobilių rinkos daliai, nepriklausomai, nuo pačių mobilių platformų skirtumo.

Kuriamos priemonės tikslas – kūrėjui žinant konkrečią mobilios platformos sistemą ir rašant jai programėlę ar žaidimą, galėtų programinėmis priemonėmis greitai pritaikyti skirtingai platformai, su mažomis tos platformos žiniomis. Reikalinga sukurti priemones, arba pritaikyti esamas technologijas, kurios esama mobilios programos kodą (arba modelį) konvertuotų į kitos mobilios sistemos programą. Priemonių veikimas turi pasižymėti:

- modelio sudarymu,
- logikos aprašymu modelio elementuose,
- transformacijų taisyklių užrašymas,
- modelio transformavimu į kodą,
- esamo kodo transformavimu į modelį.

2. ANALITINĖ DALIS

Šioje dalyje apžvelgsime sukurtas technologijas, kurių pagrindas yra programinės įrangos transformavimas iš vienos platformos į kitą. Pastaruoju metu populiarėja mobilių įrenginių naudojimas įskaitant ir naują sukurtą jiems programinę įrangą. Dauguma technologijų yra orientuotos į kompiuterines platformas, todėl ieškoma metodų kuriuos būtų galima pritaikyti ribotų išteklių, minimalios grafinės sąsajos įrenginiams programų kūrime ir transformavime.

2.1. Egzistuojantys sprendimai

2.1.1. Modeliais pagrįsta architektūra (MDA)

Modeliais pagrįsta architektūra (MDA angl. Model-Driven Architecture) – modeliavimo procese sukurtų modelių transformavimas į programos kodą nepriklausomai nuo technologijų architektūros. Problema atsirado, kai tą pati sistema projektuojama skirtingomis architektūromis arba norima reinžinerijos procesu atnaujinti pasenusią programinę įrangą naujai architektūrai. Taikant modeliavimą, ir modelių transformacija į skirtingas architektūras MDA skatina efektyvų modelių sistemų naudojimą, programinės įrangos kūrimo procesą ir tai palaiko geriausių reinžinerijos praktikas, kai kuriamos gimiringose sistemose. Tai leidžia pakartotinai naudoti esamus išteklius naujuose galimybėse, kuriuose įgyvendinamos verslo funkcionavimas laike, net ir kaip tikslius infrastruktūros vystymas. [1]

MDA pabrėžia modeliavimo naudą tarp įvairių abstrakcijos lygių, integracijos ir informacijos srautų modelių. Modeliai yra sukuriami po gero bendravimo tarp įvairių komandos narių, ir po modeliavimo užbaigimo, programinės įrangos kūrimo ir sistemų įjungimo. Sukurtas kodas, naudojant MDA yra nuoseklus ir prižiūrimas. [9]

Modeliai gali būti sukurti kaip pirmtakai į kuriamą fizinę sistemą, ar jų gali būti atkeliavusi iš esamos sistemos arba kuriamos sistemos kaip pagalba suprasti jos elgesį. [6]

MDA principai:

- Modelių atvaizdavimas gerai apibrėžtais žymėjimais – pagrindas suprasti sistemų sprendimus įmonės mastu.
- Kuriant sistemas gali būti organizuojama nustatančių transformacijų serijos aplink modelius ir tarp jų, organizuojančius į architektūros karkaso sluoksnius ir transformacijas.
- Formaliai pagrįsti aprašytus modelius, meta modelių rinkinyje, palengvinančių vaizdavimo integraciją ir transformaciją, tarp modelių, ir jų pagrindinių
- automatinių įrankių.

- Priimti ir platinti šio modelio platinimo tikslus, reikalaujančius gamybos standartus, atviresnę klientams ir skatinti konkurenciją tarp tiekėjų.

MDA transformavimo etapai:

- CIM (Skaičiavimams nepriklausomas modelis angl. Computing Independent Model), iš surinktų reikalavimų ir analizės sukonstruotas modelis.
- PIM (Platformai nepriklausomas modelis angl. Platform Independent Model), projektavimo modelis, kuris nepriklauso nuo specialių aplinkų, kaip programos kalbų, programinės įrangos, reinžinerijos proceso kuriant modelius, tinklo ir kt. skirti įgyvendinti programinę įrangą.
- PSM (Konkrečios platformos modelis angl. Platform Specific Model), detalus suprojektuotas modelis sudarytas iš įgyvendinimo aplinkų galimybių. [6]

2.1.2. Testais pagrįsta modelių transformacijos karkasas

Testais pagrįsta modelių transformacijos karkasas (The Test Based Model Transformation Framework) toliau (TBMT karkasas) aprašo konvertavimo taisykles reikalingas konvertuoti ir pasiūlyti konvertavimo profilius, ir konvertavimo algoritmus tarp modelių. Transformacija pagrįsta MDA principu, be to, dar pridedamas nepriklausomas testavimo modelis, pagal kurį palyginama pradinis kodas ir transformuotas. Modelio patikrinimas konvertuojamas per modelių transformavimo variklį, kuris sukuria ribotus modelius, aprašytus orakulo (Oracle) testais su kitom transformavimo taisyklėmis ir lyginami du modeliai su lyginamuoju algoritmu, pritaikytu su grafų konvertavimo metodais. [6]

Oracle testas aprašomas su konvertavimo taisyklėmis ir pagrindiniais modeliais taip pat konvertuojamas į prognozuojamą modelį. Kad būtų galima palyginti ir analizuoti sugeneruotą rezultatų modelį ir prognozuojamą modelį, labiau patikimas rezultatų modelis įgytas aprašant modelių palyginimo algoritmą naudojant grafų transformaciją , palyginat skirtumas tarp dviejų modelių.

Pagrindiniai modeliai yra plačiai klasifikuojami į tris modelius, panašiai kaip MDA: išteklių modelis, būsenų perėjimo diagrama ir taisyklių modelis, ir modelių elementai nustatantys individualius modelius kurie yra aprašyti. Pagrindinis modelis gali būti išreikštas kaip į PIM modelį.

Mobilios platformos pagrindinis modelis aprašomas meta modeliu konvertavime, ir aprašytas pagrindinis modelis arba prognozuojamas modelis per modelių transformacijos variklį. Pagrindinis modelis yra aprašomas su trims dalims įskaitant išteklių (source) modelį būnat mobilios programos komponentu, išreiškiamas būsenų perėjimo diagramų modelis, elgesių

specifikavimas ir taisyklių modelis, aprašantis taisyklių sąlygas, ir pritaikoma transformavimo taisyklėmis. Mobili aplikacija aprašoma savais procesų komponentuose valdymų įvykiais, išteklių meta modelis siekiant apibrėžti ypatybes ar ryšius tarp valdiklių. Tai įmanoma aprašyti statinę struktūra mobilios aplikacijos per išteklių modelių.

2.1.3. Simplian karkasas

Simplian karkasas leidžia paprasta programavimo sąsają konstruoti į programos sąsają ir kitus paprastus atvaizdavimus kaip duomenis ir įrankius, arba siųsti, priimti skirtingus bendravimo kanalus. Sukurtas C++ programos kodu ir platformai nepriklausomu XML failu, kuris gali konstruoti modeliavimo įrankį. Tikslas metodo yra reinžinerija platformai PIM modeliai generuojami PSM su vizualiais modelių procesais.

Transformacijų ypatybes sudaro: patikrinimas, išsaugojimas ir garantija, kuri patikrina ir reikalinga transformacijos procese. Transformacijos žingsnis S tikrina savybe P, kai sekanti sąlyga egzistuoja, jei savybė P buvo teisinga prieš žingsnį S, tai grąžins teisingą po S žingsnio vykdymo, jei P yra neteisinga žingsnis S nevyks. S saugo ypatybę P, kai sekanti sąlyga visada saugoma ir jei ypatybė P bus teisinga (neteisinga) prieš žingsnį S, tai gražins neteisingai (teisingai) po įvykdyto žingsnio S. Transformacija S garantuoja ypatybę P, kai sekanti sąlyga visada saugoma, jei ypatybė P bus teisinga prieš žingsnį S, tai grąžins teisingai po vykdymo žingsnio S ir jei P neteisinga žingsnis S keis ypatybę P teisingai. [5]

2.1.4. TERESA

TERESA (angl. Transformation Environnoment for inteRactivE System representAtions transformacijų aplinka sąveikaujanti su sistemų atstovybėms) – efektyvus vartotojo sąsajos įvairioms platformoms sudarytoms iš daugelių transformacijų įgyvendinimui [7]. TERESA sudaro šie lygiai:

- Užduočių ir objektų modelis. Šiame lygmenyje sukuriamas tik vartotojo tikslų modelis.
- Abstrakti vartotojo sąsaja. Aprašo sąveika tarp objektų.
- Konkreti vartotojo sąsaja. Specifiniai vartotojo sąveikos elementai palaikomi platformai.
- Baigtinė vartotojo sąsają. Galutinė programos versija.

Nustatyti galimus projektavimo sprendimus, kai palaikoma daugelį platformų, galima sukurti klasifikacijos ryšius tarp užduočių ir platformų. Tiksliau ryšiai aprašomi šiais atvejais:

- Tos pačios užduotys keliose platformose.

- Užduočių vaizdavimas vienai platformai.
- Priklausomybės tarp užduočių ir skirtingų platformų.
- Tą pati užduotis daugelyje platformų su kitais būdais:
    - Skirtingais pagrindiniais objektais.
    - Skirtingais sąsajos objektais.
    - Skirtingomis užduočių dekompozicijos.
    - Skirtingais laikinais ryšiais tarp užduočių

Pradedama aukšto lygio modeliavimo daugiaplatformė programa. Šioje fazėje, projektuotojai sukuria vieną modelį, kuris skirtas įmanomais naudojimo kontekstais ir įvairių platformų dalyvavime, įskaitant dominuojantį modelį padedantį identifikuoti visus objektus, kurie turi būti manipuliuojami, atliekant užduotis ir ryšius tarp objektų. Kiekvienai užduočiai galima nurodyti, kokias platformas palaikyti, taip pat galima parodyti nepriklausomybes tarp užduočių, kad būtų galima atlikti skirtingas transformacijas tarp platformų.

Sekantis etapas: kurti sistemos užduotis skirtingoms platformoms. Čia projektuotojai turi užduočių filtrus tarp rezultatų ir suformuojamas specifinis platformos modelis, sudarytas iš įvairių platformų užduočių modelių. Tada projektuojamas sistemos užduotys ir abstrakti vartotojo sąsają. Šios fazės tikslas – gaunamas abstraktus aprašymas vartotojo sąsajai ir aibė abstrakčių pateikčių, kurie yra identifikuojami per užduočių ryšių analizes.

Kitas žingsnis: transformacija iš abstrakčios į konkrečią sąsają, kiekviena sąveika surašomą į konkretų platformos įrenginį, t. y. iš abstraktaus vaizdavimo operacijos turi tinkamai susijusios su technologijų pabrėžimu jų loginiu vaizdavimu.

Galutinė fazė kodo generavimas, kur kodas sugeneruojamas pradedant konkrečia sąsają ir programinės įrangos aplinka. Tai gali būti pilnai atlikta automatiškai, kadangi visi projektavimo darbai yra atlikti. Pagrindiniai reikalavimai projektuojant ir kuriant su TERESA:

Mišri iniciatyva. Palaikyti skirtingus autonomijos vykdymo lygius iš užpildytų sprendimų į aukštus interaktyvius sprendimus kur projektuotojai gali pritaikyti, net radikaliai pakeisti sprendimus siūlomais sprendimais.

Modelių pagrindas. Daugelis platformų padidinimas įmanomas, tai gali būti geriau atlaisvinant per keletą abstrakcijų, kurie leis projektuotojams turėti loginį požiūrį palaikant veiklą.

Daugelių loginių lygių aprašomi per XML pagrindo kalbas.

- Viršus-apačia. Šis metodas yra pavyzdys grįžtamajai inžinerijai, kai sudaro įvairūs abstrakcijos lygiai, ir mes palaikome atvejus, kai projektuotojai nori pradėti nuo pradžios.

- Skirtingi įėjimo taškai. Metodas padeda išsamiai palaikyti įvairias galimybes, nurodančias užduotis/platformas klasifikuoti.
- Orientuota žiniatinklio (web-oriented). Metodas tinkantis generuoti vartotojo sąsajas žiniatinklyje taikant kitų tipų aplinkas kaip Java, Microsoft.

2.1.5. Nepriklausoma įrenginių žymėjimų kalba (Device-Independent Markup Language DIML)

Kalba skirta adresuoti problemas kurios ištinka transformuojat grafinę vartotojo sąsają. Kiekvieno įrenginio grafinės sąsajos elementai yra skirtingi. Sąsajos generuojamos transformacijų varikliu, kuris vartotojo sąsajas sugeneruotas vartotojo kombinuoja į daugelį skirtingų įrenginių su skirtingais parametrais. Ši kaba aprašo vartotojo sąsajas aukščiau nepriklausomai įrenginio, ir nurodo aprašus į kitą žymėjimo arba programavimo kalbą. Platformą susidaro trys dalys: namų serverio dalis, vaizduojamo įrenginio dalis, originali vartotojo sąsaja. Nepriklausomas įrenginys dažniausiai nurodomas kaip ekranas su grafinę sąsaja, neatsižvelgiant į įrenginio vidų, kuriame reikalingi tam tikri aprašymo formatai. [8]

Architektūra įtraukia mechanizmus, skirtus įrenginiams, prisijungti prie tinklo ir siusti būseną apie valdymą ir vartotojo sąsają. Ši architektūra taip pat nurodo enkapsuliacijos metodus grafinei sąsajai. Vartotojo sąsaja siunčiama iš serverio į įrenginį, kuris valdys ir naršys kaip klientinis įrenginys per jo esančius valdymo elementus. Kuriant sąsają apibrėžiamas gamintojo įrenginys ir įgyvendinimo kalba. Tai leidžia sukurti sąsają su nuorodomis kito įrenginio sąsajos elementais.

Architektūros komponentai:

- Vaizdavimo taisyklių sluoksnis (DRS): XML dokumentas aprašantis transformavimo politiką apie vaizdavimo stilių. Parenkami skirtingiems įrenginių ekranams bibliotekos ir to įrenginio pritaikymas.
- Vaizdavimo stiliaus sluoksnis (DSS): XML dokumentas aprašantis vartotojo sąsaja.
- Transformacijos variklis: tai branduolys su pagrindine funkciją, kuri konstruoja originalią grafinę sąsają pritaikant tam tikram įrenginiui. Jei toks įrenginys nesantis originaliam atvaizdavimui, naudojama nuodugni analizė aprašyta DSS DRS failuose.
- Sąsajos formavimo įrankis – tai programa skirta modeliuoti grafinę sąsają galutiniam vaizduokliui.
- Vaizdo generatorius – tai XML skaitymo įrankis, kuris atvaizduoją grafinę sąsają iš XML failų.

2.1.6. ArchMDE metodas

Pagrindinė idėja ArchMDE yra programinės įrangos nepriklausomumas programinės įrangos architektūros bet kokios platformos įgyvendinimo. Architektūra yra nepriklausoma nuo platformos nuostatų turi būti susijusi su PIM lygmeniu. Dėl šios priežasties, suskaidys PIM į du lygius: architektūros nepriklausomą modelį (AIM) ir architektūros specifinį modelį(ASM). ASM kombinuoja AIM specifikacijas su detalėmis, kurios aprašomos kai sistema naudoja praktinį architektūros stilių. [4]

ArchMDE metodas pasiūlo modelių rinkinį, kurios yra susijusios su ArchMDE vaizdu. Trys vaizdai aprašyti kaip verslo, architektūros ir platformos. Verslo vaizdas parodo dalį užpildytos specifikacijos, kuri nekeičia vienos architektūros stiliaus į kitą. Architektūros vaizdas sudarytas iš raktinio elementų aprašymo stilių. Tai leidžia naudoti ASM, pristatant skirtingas elementų dalis, naudojant specifinius architektūros stilius programai. Platformos vaizdas aprašo aibę techninių sąvokų, pristatant skirtingas dalis šios platformos teikiamos paslaugomis.

Kuriant ArchMDE pradedama reikalavimu modeliu (CIM), po to sujungiamas architektūros stiliaus modelis į programinės įrangos modelį (ASM) kuris apima funkcinį ir architektūrinius aspektus. Galiausiai, PSM yra gaunama iš ASM kurios programos kodas būna sugeneruotas. Modelio transformacijos procesas aprašo būdą, pradinio modelių elementai turi sutapti ir pereinami siekiant inicijuoti tikslinio modelio elementus. Transformacija charakterizuojama į:

- Transformavimo būdus: rankinė, automatinė, ir pusiau automatinė.
- Transformacija pritaikoma skirtingais elementų lygiais: metamodelis, modelis ir atskiras atvejis.
- Skirtingų tipų naudojami metodai, kuriant transformacijų taisykles. Kiekvienas metodas leidžia ištisines arba dvikryptes transformacijas.
- Naudojamos natūralios kalbos, įgyvendinat taisykles (imperatyvi, deklaruojama arba hibridinė).

2.2. Programų sistemų kokybinis palyginimas

2.2.1. Sprendimų paremtais MDA transformavimų komponentų palyginimas

Egzistuojantys sprendimai paminėti dalyje (2.1) yra paremti modeliais pagrįstos architektūros principais, kurie aprašyti (2.1.1) skyriuje. MDA transformavimo komponentai kaip konkretus platformos modelis (PSM), nepriklausomas nuo platformos modelis (PIM) bei nepriklausomas skaičiavimų modelis (CIM) kiekviename metode interpretuojami skirtingomis metodologijomis, objektais arba struktūromis. Lentelėje (2.1 lentelė) pateikimas transformavimo komponentų palyginimas.

2.1 lentelė: Sprendimų paremtais MDA transformavimo komponentų palyginimas

Metodas	***Klasikinė MDA***	**TBMT karkasas**	**Simplian karkasas**	**TERSA**	**DIML**	**ArchMDE**
Transformavimų komponentai	*Konkretus platformos modelis (PSM)*	Išteklių modelis	Grafinė sąsajos PSM	Užduočių ir objektų modelis Konkreti ir baigtinė vartotojo sąsaja.	Grafinė sąsajos PSM	Architektūros specifinis modelis (ASM)
	Nepriklausomas nuo platformos modelis (PIM)	Nepriklausomas testavimo modelis Pagrindinis meta-modelis	Grafinės sąsajos PIM XML pavidalu	Abstrakti vartotojo sąsaja.	Vaizdavimo stiliaus ir taisyklių sluoksnis (DSS) (DRS)•.	Architektūros nepriklausomas modelis (AIM)
	Nepriklausomas skaičiavimų modelis (CIM)	Būsenų ir taisyklių perėjimo modelis	*Neturi*	*Neturi*	*Neturi*	Reikalavimų modelis

Simplian, TERESA bei DIML metodai orientuoti į grafinės sąsajos transformavimą. Šiuose metoduose yra nuorodų į XML grafinės sąsajos formavimą. Šiuo atveju XML struktūros asocijuojasi su žiniatinklio sistemų veikimu. Galimas vienas iš būdų integruojant į šiuos metodus įterpiant papildomus žiniatinklio elementus, pavyzdžiui, kaip JavaScript, su kuriuo būtų galima aprašyti ne tik nepriklausomus nuo platformos grafinės sąsajos elementus, bet atlikti tam tikrą nepriklausomą skaičiavimų logiką (CIM).

Tiek ArchMDE ir TBMT karkasai paremti, iš anksto aprašytomis transformavimo taisyklėmis, kurios susijusius su konkrečiu atveju. Šiuo atveju šie metodai paremti konkrečiam atvejui taikant iš kuriamos programos specifikacijos aprašant nepriklausomas būsenas, bei

modelius ir iš jų pagal transformacijos aprašytus metodus sukurti konkrečius modelius, t. y. transformacija vyksta tik iš PIM į PSM.

Atliekant priemonių kūrimą ir tyrimą bus reikalinga įvertinti galimas įgyvendinimo problemas, palyginti sąnaudas kuriant programas be transformavimo priemonių, arba naudojant universalias žiniatinklio pagrindu veikiančias priemones.

2.3. Įgyvendinimo problemos

2.3.1. Orientacija į kompiuterines technologijas

MDA sprendimas daugiau orientuotos į plačių funkcijų ir didelio našumo architektūras. Todėl norint programas transformuoti ar modeliuoti į mobilius įrenginius, reikalingos esamų įrankių modifikacijos, kurios būtų orientuotos į ribotų funkcijų mobiliąsias architektūras.

2.3.2. Modeliavimo netaikymas projektuose

Dauguma programuotojų programuoja kodą iš „galvos“ naudojantis IDE. Jų kodo priklausomybė apibrėžiama paketais (bibliotekomis, modeliai). Visas projektavimas lieka tik lentoje, skaidrėse, ar komandos bendram suvokime. Kol šis požiūris bus adekvatus asmenis ir mažoms komandoms, ir galiausiai tai taps labai sudėtinga valdyti sprendimus kaip jie [sprendimai] didėja ir sudėtingėja, kaip sistema vystosi visą laiką, arba kai tos sistemos modeliavimo komandos nariai nebus tiesiogiai prieinami naujai komandai prižiūrinčiai sistemą. [1]

2.3.3. Modeliavimo nutolimas nuo realios užduoties

Šią problemą iškėlė Boudle. Minama, kad taikant MDA sistema projektuojama užduočių lygmenyje, į pačios sistemos funkcionalumą, bet neatsižvelgiama apie vartotojo sąsajos pateikimą. Dalis projektų, modeliuojant, ir renkant reikalavimus, vartotojas nemato visos sistemos kūrimo progreso, o galutinę versiją, kuri daugelį atvejų netinka jo poreikių. [1].

2.3.4. Programų transformacijų priemonių naudojimas mažina sistemos našumą

Naudojant karkasus, skirtus transformuoti modelius į programos kodą, naudojami bendri abstraktūs modeliai. Kad transformacija vyktų sėkmingai turi būti aprašytas bendras visų sistemų (arba programavimo kalbų) modelis, kurį transformuojant į skirtingas sistemas būtų atvaizduotas vienodai. Tai reiškia, naudoti esamas sistemų komponentus ar struktūras, o jei tą sistema tokio komponento neturi – jas papildyti karkase aprašytu kodu. Pavyzdžiui galime aprašyti modelyje

standartinę klase „class Integer“ ir ją panaudoti iš Java „Integer“ ir C# „Int32“ standartines klases , bet transformuojant iš modelio šią klasę į C++ tektų naudoti klasę aprašytą karkase. Šiuo atveju nebūtų C++ manipuliuojama su baziniu tipu „int“, ir kodas pasipildytų pertekliniu kodu. Sistemai didėjant perteklinis kodas lėtintų sistemų veikimą, bet šiuo atveju būtų išvengta klaidų ir kodas taptų vieningas

3. PROJEKTINĖ DALIS

Šioje dalyje aprašysime „Mobilių programų transformavimo iš vienos platformos į kitą priemonės“ (toliau Transformavimo priemonės) esminius architektūrinius sprendimus.

3.1. Reikalavimų specifikacija

3.1.1. Projektuojamos priemonės paskirtis ir tikslai

Modeliais pagrįsta mobilių programų kūrimas bei esamų mobilių programų išeities tekstų transformavimas į modelius, nepriklausomai nuo mobilios sistemos (programinės įrangos) ir architektūros.

Projektuojamos transformavimo priemonės tikslas – kūrėjui žinant konkrečią mobilios platformos sistemą ir rašant jai programėlę ar žaidimą, palengvintų programinės įrangos priemonėmis pritaikyti skirtingai platformai , su mažomis tos platformos žiniomis

3.1.2. Projekto kūrimo pagrindimas

Transformavimo priemonė bus svarbi mobilių programėlių ir žaidimų kūrėjams, kurie norės savo sukurtas programėles parduoti dideliai mobilių sistemų rinkos daliai, nepriklausomai, nuo pačių mobilių platformų skirtumo ir suderinamumo.

3.1.3. Suprojektuotos priemonės vartotojai

Būsimi transformavimo priemonės vartotojai:

- Mobilių programų projektuotojai – su šią priemone sudarys, mobilių programų modelius.
- Mobilių programų programuotojai – naudodamiesi transformavimo funkcijos kurti mobilias programėles nepriklausomai nuo platformos.
- Mobilių programų testuotojai-derintojai – šios sistemos pagalba galės rašyti testavimo atvejus, nepriklausomai platformai ir iš bandyti programėles konkrečioje platformoje.
- Mobilių programų diegėjai – transformavimo priemonės sukurtas programėles galės publikuoti el. erdvėje ir įdiegti vartotojams.

3.1.4. Apribojimai reikalavimams

3.1.4.1. Apribojimai sprendimui

Transformavimo priemonė turi tiksliai transformuoti modelius į programos kodą, laikantis UML bei transformacijų užrašymo taisyklių bei atliktus programuotojo pakeitimus programos kode, atvaizduoti juos modeliuose.

3.1.4.2. Diegimo aplinkos apribojimai.

Transformavimo priemonė turi turėti galimybę įdiegti tiek atviro tipo, tiek komercinio tipo operacinių sistemų aplinkomis, nurodant konkrečias diegimo programas tai operaciniai sistemai.

Transformavimo priemonė diegimas turi būti suprantamas, tos sistemos vartotojui, vadovaujantis diegimo vedliais (wizards), atvaizduojant numatytas pagrindines parinktis, ir leidžiant vartotojui pasirinkti tam tikrus, pagal jo sistemos poreikį.

3.1.4.3. Bendradarbiaujančių sistemų apribojimai.

Transformavimo priemonė turi veikti tiek atviros, tiek komercinės paskirties operacinėse sistemose. Norint išlaikyti suderinamumą ir universalumą, turi būti naudojamos universalios programavimo kalbos, kompiliatoriai. Jei to neįmanoma, vartotojui reikalaujant, kad programinė įrangos kodai būtų sutransliuoti į vartotojo architektūrą.

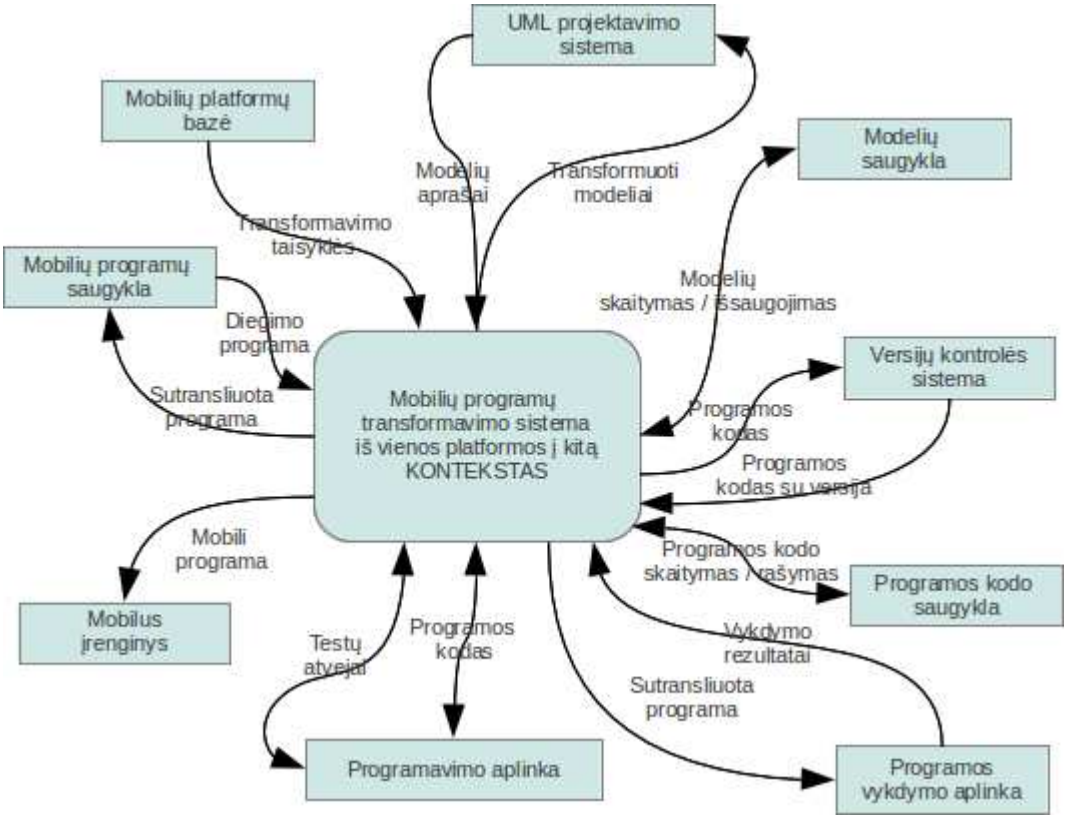
Transformavimo priemonė naudodama modeliais pagrįstą architektūrą (MDA), turi palaikyti bent kelių populiarių mobilių platformų operacinių sistemų ir komponentų transformavimą, taip pat suteikti galimybes, kurti sistemai įskiepius, kurie apjungtų vartotojų papildomas platformas.

3.1.5. Funkciniai reikalavimai

3.1.5.1. Veiklos kontekstas.

Transformavimo priemonė iš UML projektavimo sistemos surenka modelius, (arba susiformuoja modelį iš pradinės programos kodo saugyklos) ir juos išsaugo modelių saugykloje. Modelių saugykloje esantys modeliai nuskaitomi, ir redaguojami UML projektavimo sistemomis, arba tiesiog transformuojami, naudojant mobilios platformos transformavimo taisykles, į programos kodą, kuris išsaugojamas programos kodo saugykloje, ir versijų kontrolės sistemoje. Programos kodai iš versijų kontrolės sistemos ir kodo saugyklos įkeliami į programavimo aplinka, kurioje tobulinama programa, kuriami testiniai atvejai. Programos kodo pakeitimai išsaugojami saugyklose ir versijų kontrolės sistemose, sukuriamas atnaujintas modelis, remiantis

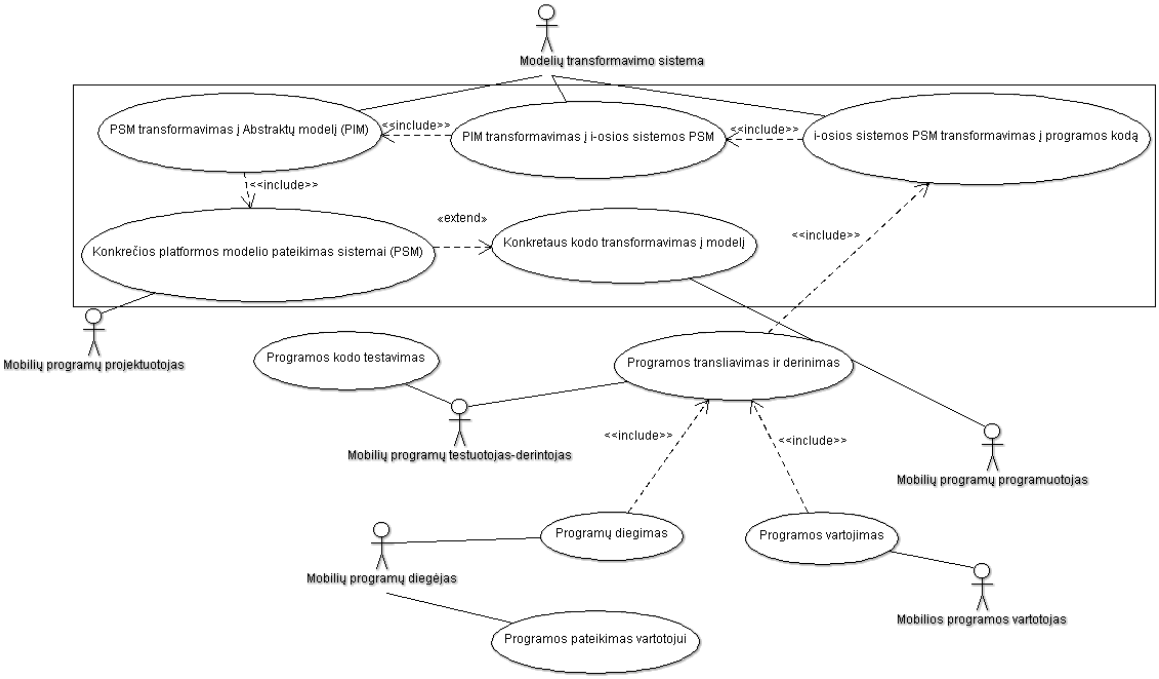
transformavimo taisyklėmis. Atnaujintas modelis išsaugojamas į modelių saugyklą. Jei įmanoma programos kodas transliuojamas į konkretaus įrenginio vykdomą bylą, jis testuojamas, programos vykdymo aplinkoje. Ištestuota ir sutransliuota programa išsaugojama mobilių programų saugykloje ir pateikiama vartotojui.



3.1 pav.: Sistemos kontekstinė diagrama

3.1.5.2. Panaudos atvejų sąrašas

Pateiksime projektuojamos priemonės vidinius ir išorinius panaudos atvejų schemą (3.2 pav.).



3.2 pav.: Modelių transformavimo sistemos panaudos atvejų diagrama

Projektuojamos transformavimo priemonės pagrindinės funkcijos bus susijusius su programų arba modelių transformacijomis, tai sistemos projektavimui bus detalizuojami tik šie panaudos atvejai:

- Konkretaus kodo transformavimas į modelį PA (3.1 lentelė)
- Konkretaus modelio pateikimas sistemai (PSM) PA (3.2 lentelė)
- PSM transformavimas į abstraktų sistemos modelį (PIM) PA (3.3 lentelė)
- PIM transformavimas į i (-osios) sistemos PSM. PA (3.4 lentelė)
- i-(osios) sistemos PSM transformavimas į programos kodą PA (3.5 lentelė)

3.1 lentelė: Konkretaus kodo transformavimo į konkretų modelį PA specifikacija

Panaudos atvejis	1. **Konkretaus kodo transformavimas į modelį**
Tikslas	Transformuoti programos kodą ir logiką į UML modelį
Aktoriai	Mobilių programų programuotojas
Ryšiai su kitais panaudos atvejais	Nėra
Nefunkciniai reikalavimai	Programa turi veikti konkrečiai platformai
Prieš-sąlygos	Programos kodas rašomas žinant jos modelį
Sužadinimo sąlyga	Programuotojas pateikia sistemai kodą
Po-sąlyga	Programos kodas atvaizduotas UML modeliu su vykdymo logika.
Pagrindinis scenarijus	Tikrinama ar programos kodas yra be klaidų, ar programos kodas tinkamas atvaizduoti modeliui.
Alternatyvūs scenarijai	Vartotojas modifikuoją programos kodą (jei atsiranda klaidų, ar keičiasi programos modelis) ir pateikia sistemai iš naujo.

3.2 lentelė: Konkretaus modelio pateikimo sistemai PA specifikacija

Panaudos atvejis	2. **Konkretaus modelio pateikimas sistemai (PSM)**
Tikslas	Pateikti programos konkrečiai sistemai modelį.
Aktoriai	Mobilių programų projektuotojas
Ryšiai su kitais panaudos atvejais	Programos transformavimas į konkretų modelį
Nefunkciniai reikalavimai	Modelis turi būti teisingas atitinkantis UML aprašymo taisykles
Prieš-sąlygos	Modelis turi būti suprojektuotas UML taisyklėmis arba suprogramuotas programuotojo.
Sužadinimo sąlyga	Pateikiamas modelis sistemai
Po-sąlyga	Atliekama modelio transformacija
Pagrindinis scenarijus	• Analizuojamas modelis • Nustatoma modelio platforma • Transformuojamas modelis į abstraktų modelį • Transformuojamas abstraktus modelis į keletą sistemų modelių.
Alternatyvūs scenarijai	Patikslintas modelis pateikiamas sistemai iš naujo.

3.3 lentelė: PSM transformavimo į abstraktų sistemos modelį PA specifikacija

Panaudos atvejis	3. **PSM transformavimas į abstraktų sistemos modelį (PIM)**
Tikslas	Suvesti aprašytą modelį į abstraktų (subendrintą) konkrečios mobilios platformos modelį
Aktoriai	Modelių transformavimo sistema
Ryšiai su kitais panaudos atvejais	Įtraukti modelio pateikimą sistemai
Nefunkciniai reikalavimai	Pateiktas modelis turi atitikti konkrečios sistemos modelį
Prieš-sąlygos	Pateiktas modelis sistemai
Sužadinimo sąlyga	Modelio pateikimas sistemai
Po-sąlyga	Modelio transformaciją į konkretų abstraktų modelį
Pagrindinis scenarijus	Modelio transformavimas į konkrečiai abstraktų modelį
Alternatyvūs scenarijai	Transformacijos nutraukimas

3.4 lentelė: PIM transformavimas į i (-osios) sistemos PSM PA

Panaudos atvejis	4. **PIM transformavimas į *i* (-osios) sistemos PSM.**
Tikslas	Absoliučiai abstraktaus modelio transformavimas į keletą konkrečių platformų modelius
Aktoriai	Modelių transformavimo sistema
Ryšiai su kitais panaudos atvejais	Įtraukti PSM transformavimas į abstraktų sistemos modelį (PIM)
Nefunkciniai reikalavimai	nėra
Prieš-sąlygos	Absoliučiai abstraktus modelio transformacija
Sužadinimo sąlyga	Konkretaus modelio pateikimas sistemai
Po-sąlyga	Absoliučiai abstraktaus modelio transformavimas į abstrakčius sistemų modelius.
Pagrindinis scenarijus	Transformavimas absoliučiai abstraktaus modelio į abstrakčius sistemų modelius.
Alternatyvūs scenarijai	Transformavimo atšaukimas

3.5 lentelė: i (-osios) sistemos PSM transformavimo į programos kodą PA specifikacija

Panaudos atvejis	5. ***i*-(osios) sistemos PSM transformavimas į programos kodą**
Tikslas	Sugeneruotų modelių transformavimas į programos kodus
Aktoriai	Modelių transformavimo sistema
Ryšiai su kitais panaudos atvejais	Įtraukti PIM transformavimas į i (-osios) sistemos PSM.
Nefunkciniai reikalavimai	Programos kodai turi veikti visose platformose
Prieš-sąlygos	Abstrakčių modelių transformaciją į programos modelius
Sužadinimo sąlyga	Programos modelio pateikimas
Po-sąlyga	Programos kodų generavimas
Pagrindinis scenarijus	Modelių transformaciją į programos kodus
Alternatyvūs scenarijai	Transformacijos nutraukimas

3.1.5.3. Funkcinių reikalavimų sąrašas

Projektuojamos transformavimo priemonės funkcijos :

- Programos kodo transformavimas į UML modelį
- Konkretaus modelio pateikimas sistemai
- Konkretaus modelio transformavimas į abstraktų modelį, nepriklausoma nuo sistemos
- PIM transformavimas į i(-osios) sistemos PSM
- Žinomų sistemų modelių transformavimas į programos išeities kodą.
- Korektiško programos išeities kodo suformavimas transliavimui.

3.1.6. Nefunkciniai reikalavimai

Projektuojamos transformavimo priemonės nefunkcinių reikalavimų sąrašas:

- Komandinės eilutės sąsajos palaikymas.
- Priemonių veikimo užimtumo atvaizdavimas.
- Žinomų mobilių platformų palaikymas.
- Projektavimo sistemų palaikymas.
- Duomenų saugyklų palaikymas.
- Priemonių automatinis naujinimas.

3.1.7. Galimos problemos projektavimui ir diegimui

Problemos diegimo aplinkai:

- Projektuojant priemones, gali pasikeisti diegimo aplinka, ir gali iškilti problemų per projektuojant sistemą naujai diegimo aplinkai.

Įtaka jau įdiegtoms sistemoms:

- Transformavimo priemonės gali būti apkrautos platformų bibliotekomis, ir sudaryti tam tikras komponentų sąveikos klaidas, dėl kurių transformavimo priemonių našumas sumažės.

Kliudantys diegimo aplinkos apribojimai:

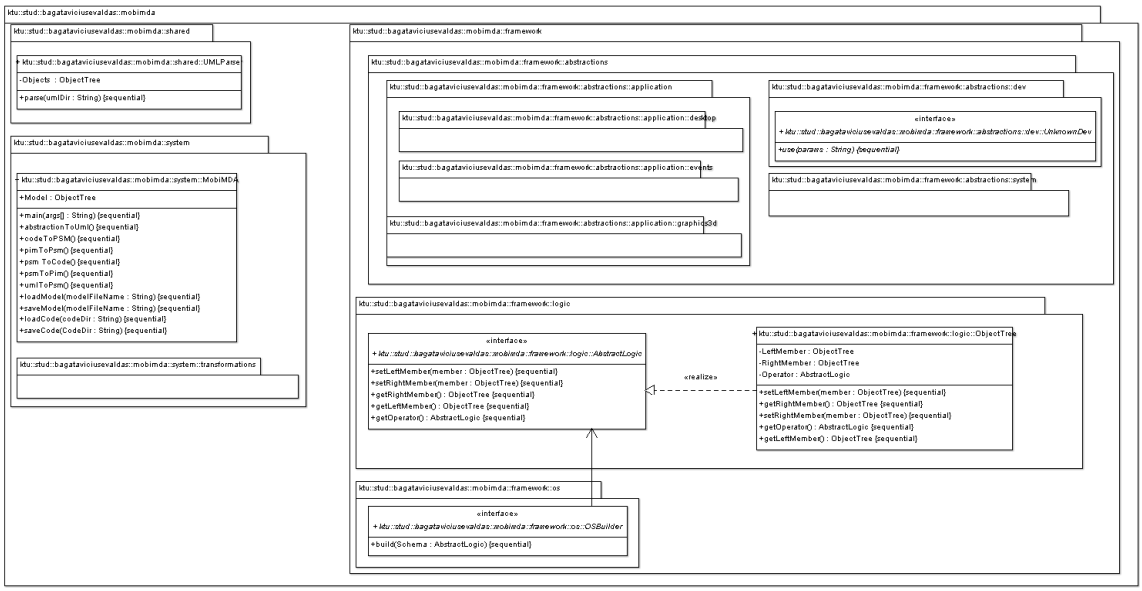
- Kadangi transformavimo priemonės apjungs, skirtingas platformas, yra galimybė pasikeisti sistemų platformoms šios sistemos projektavimo ir kūrimo eigoje arba pabaigoje, ir ši kuriama sistema taps pasenusi naujų platformų atžvilgiu.

3.2. Projektuojamos transformavimo priemonių architektūros specifikacija

3.2.1. Sistemos statinis vaizdas

3.2.1.1. Sistemos pagrindiniai komponentai

Sistemos pagrindinių paketų diagrama:



3.3 pav: Sistemos pagrindiniai komponentai

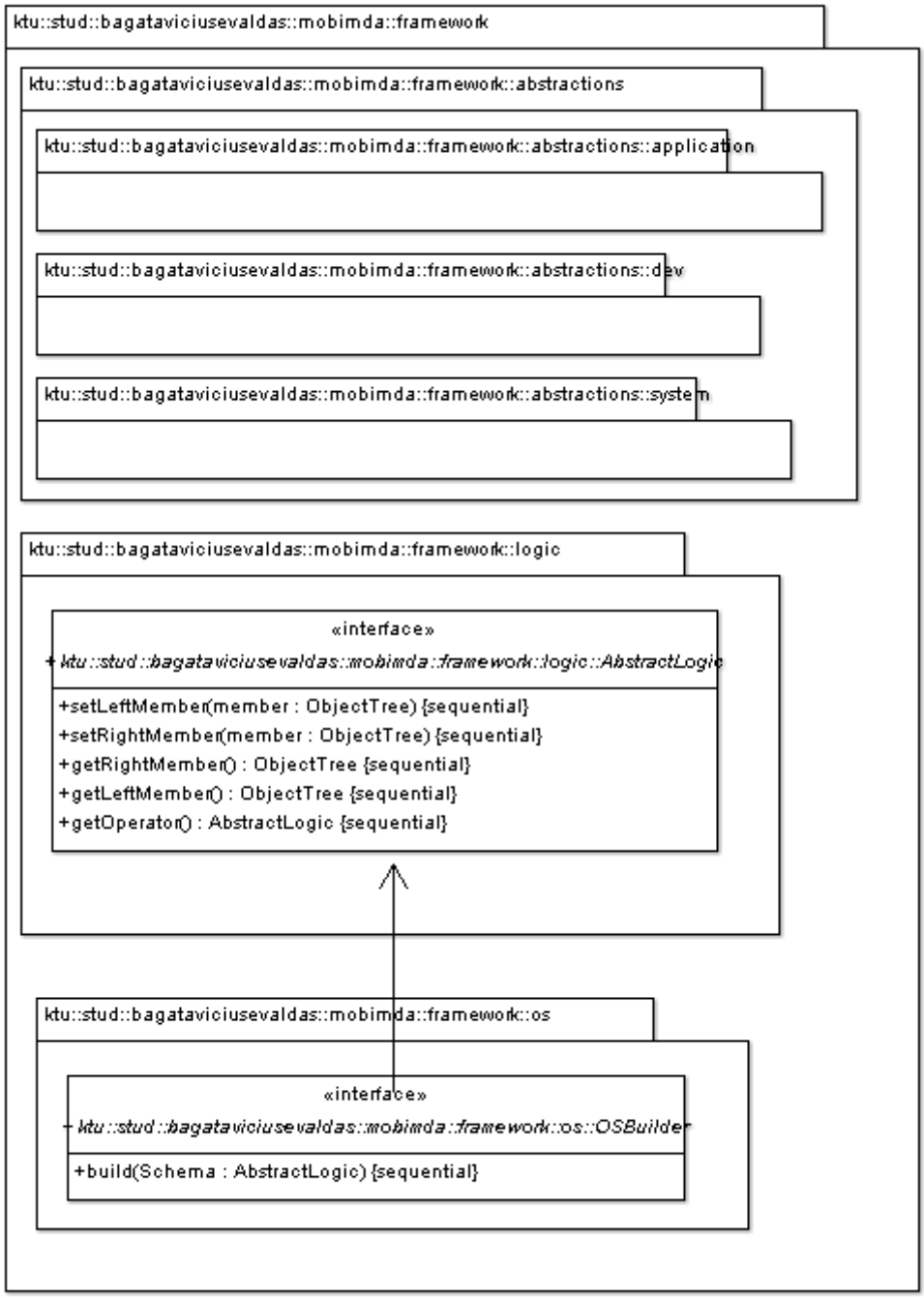
Sistemą sudaro šie pagrindiniai paketai:

- „framework“ - šiame pakete aprašomos abstrakčios struktūros kodas-modelis-kodas arba modelis-kodas-modelis transformacijoms.

- „libs“ - šis paketas skirtas, jei į sistemą reikės integruoti atskiras posistemes, kaip duomenų bazės, sistemos komponentai, kuriems bus reikalingos „adapter“ klasės.
- „shared“ paketas – aprašomas UML modelio analizavimo įrankiai.
- „system“ pakete vykdomasis sistemos paleidimo kodas.

3.2.1.2. Paketų detalizavimas

„framework“ paketo detalus aprašymas:

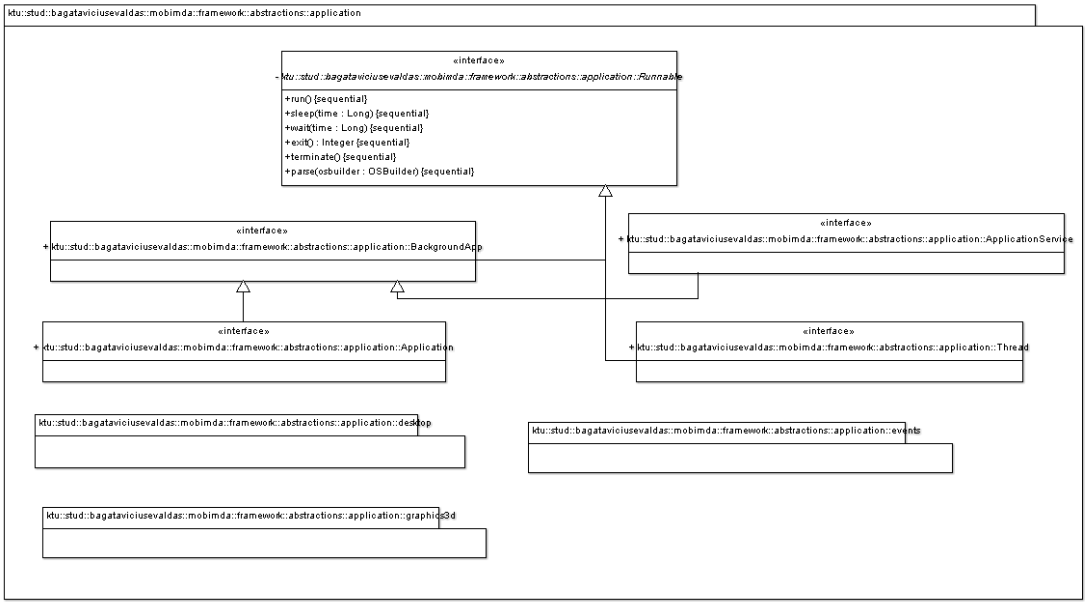


3.4 pav.: „framework“ paketų struktūra

Paketą „framework“ sudaro tokie paketai:

- „abstractions“ - kuriame aprašyti kokiais mobilių sistemų objektais manipuliuos transformavimo algoritmai:
- „abstractions::applications“ - mobilaus įrenginio paleidžiamos programos modeliais;
- „abstractions::dev“ - mobilaus įrenginio aparatinės įrangos įrenginiais;
- „abstractions::system“ - mobilaus įrenginio operacinės sistemos sąsajomis.
- „logic“ pakete saugomi modelio logikos klasės, kuriais sistema, sudarys objektų medį iš kurio transformuos modelius bei kodą.
- „os“ pakete -mobilių įrenginių modelių transformavimo į kodą transformacijų klasės pagal mobilios operacinę sistemą.

„framework::abstractions::application“ paketo aprašymas:



3.5 pav.: „aplication“ paketas

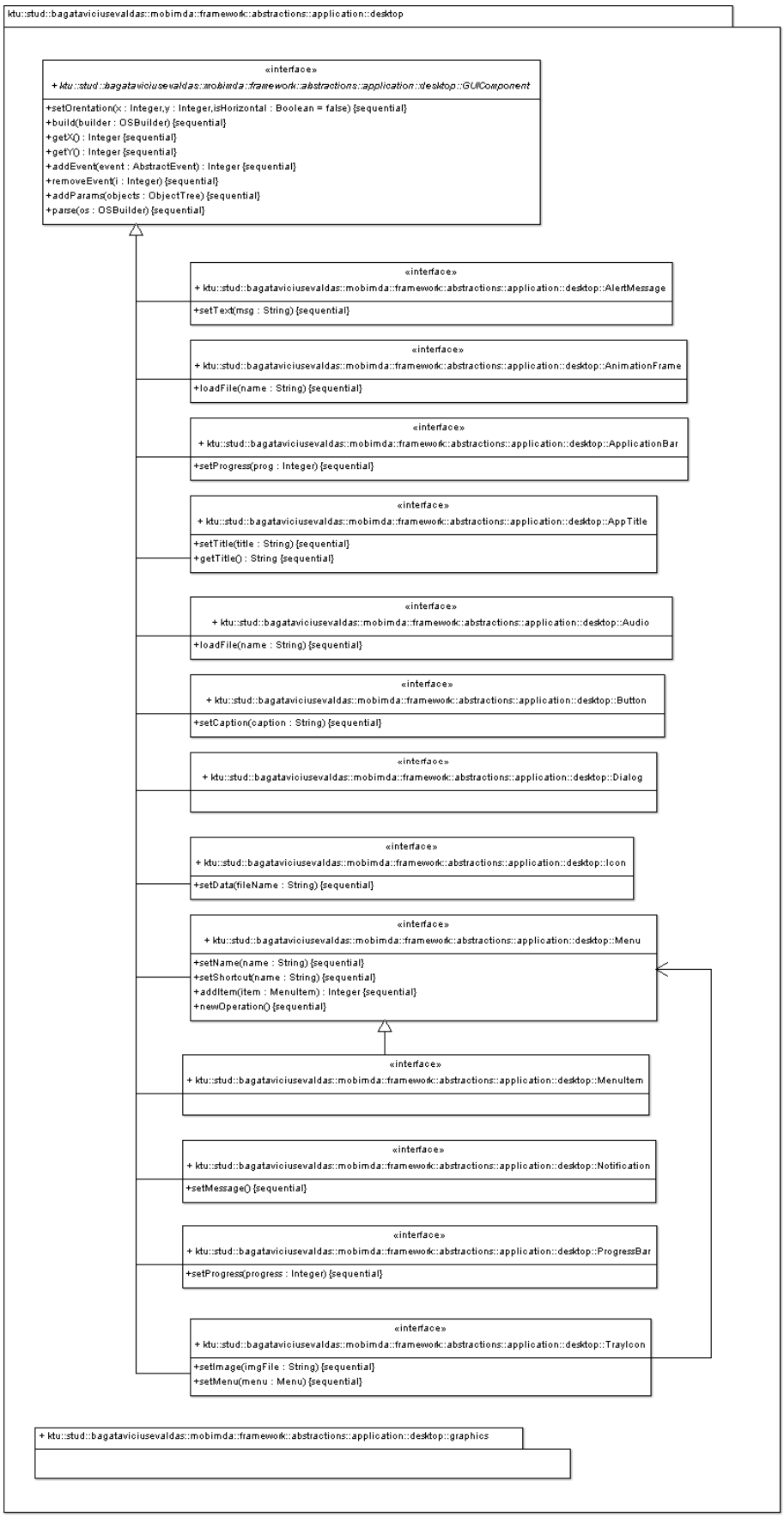
Paketą „application“ sudaro šios klasės (sąsajos):

Abstrakti sąsaja „Runnable“, kuri nurodo, kad bus formuojama per metodą „parse()“, mobilaus įrenginio vykdomoji programa. Mobilus įrenginys gali inicijuoti foninę programą „BackgroundApp“, programą kaip paslaugą „ApllicationService“ arba vartotojui taikomosios tipo programas „Application“.

Mobilaus įrenginio operacinė sistema, gali palaikyti gijas ir per „Thread“ sąsają, priklausomai nuo mobilios operacinės sistemos sukurti gijas.

Paketas taip pat sudarytas iš šių paketų kaip „desktop“, „events“, „graphics3d“ paketų, kuriuose esančios sąsajos, transformuojant modelis-kodas, sudarytų mobilaus įrenginio interaktyvią sąsają, bei išnaudotų erdvinės grafikos elementus.

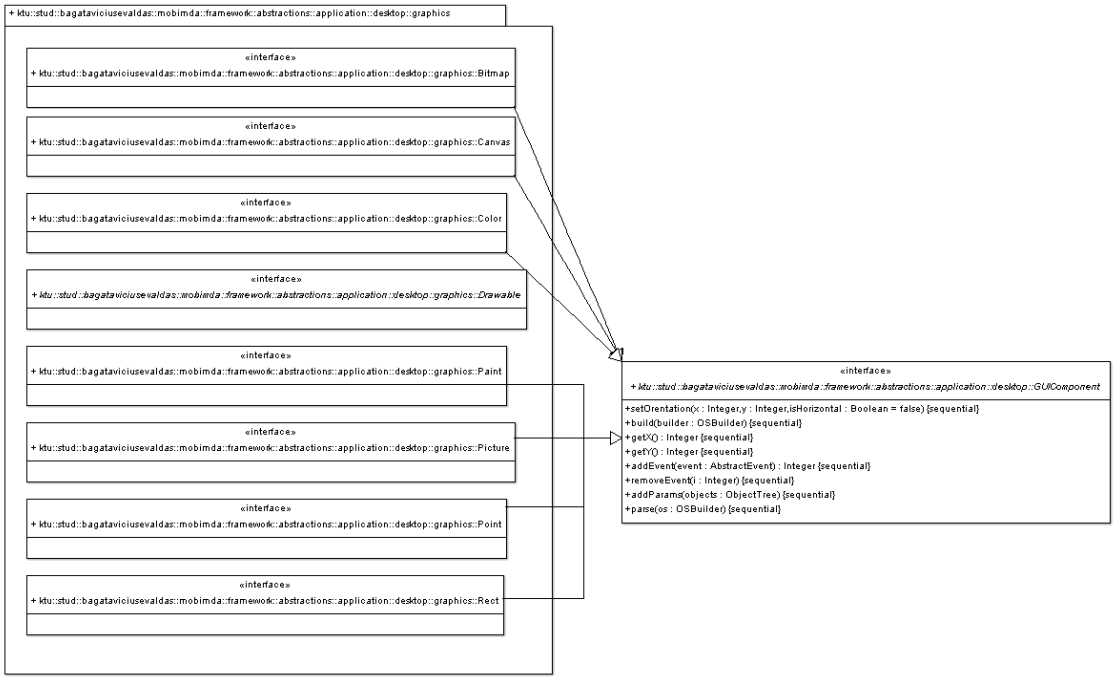
„framework::abstractions::application::desktop“ paketo aprašymas:



3.6 pav.: „desktop“ paketas

Paketas „desktop“ sudarytas iš pagrindinių dažniausiai naudojamų grafinės sąsajos elementų. Šios grafinės sąsajos elementai skirti, veikiant modelis-kodas transformacijai, transformatorius, pagal šias sąsajas sukuria iš modelio kodo fragmentus mobiliam įrenginiui. Jeigu pagrindinių grafinių sąsajos elementų neužtenka, „desktop“ turi paketą „graphics“, kuriame aprašomos dvimatės grafikos sąsajos.

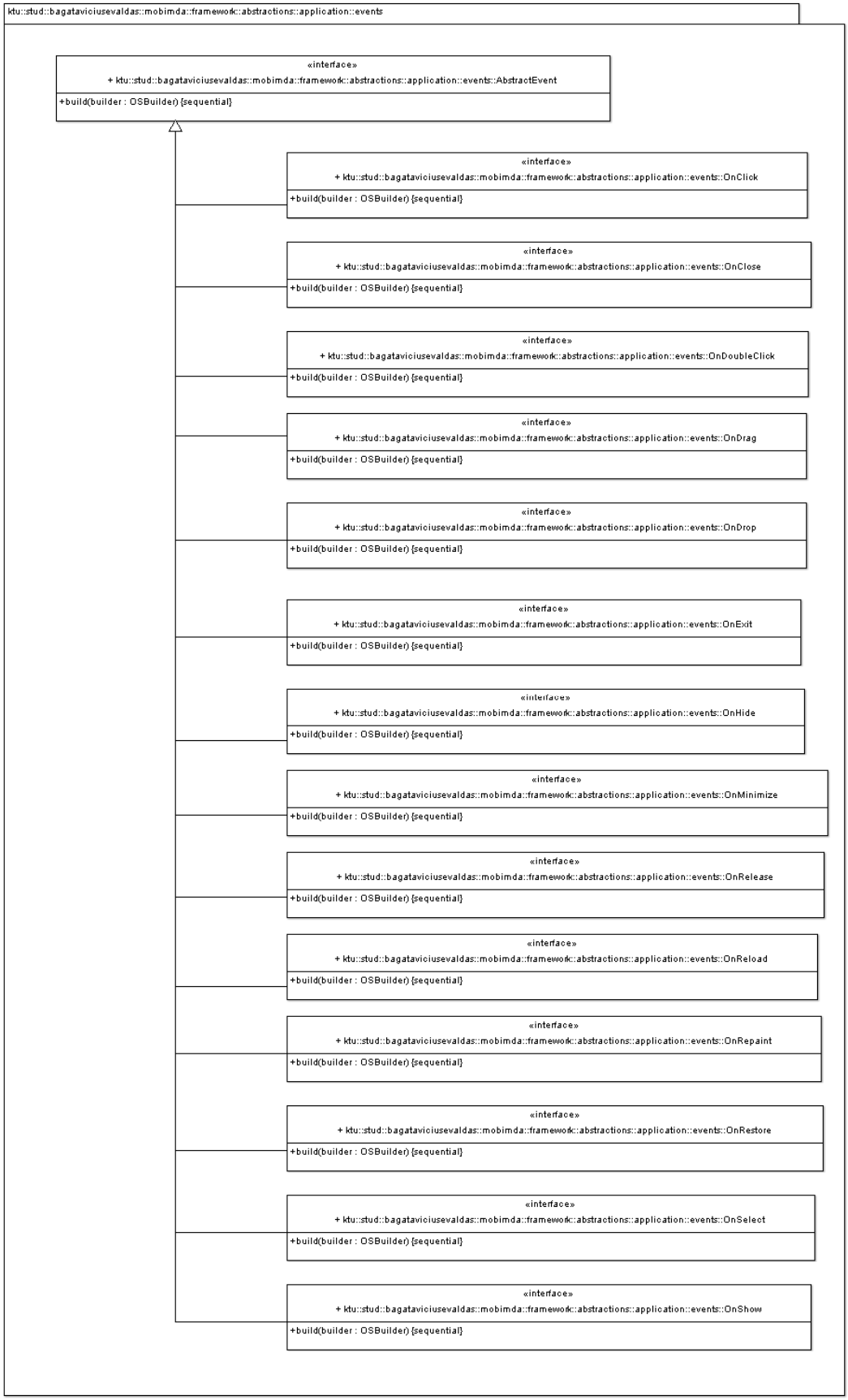
„framework::abstractions::application::desktop::graphics“ paketo aprašymas:



3.7 pav.: „graphics“ paketas

Paketas „graphics“ sudarytas iš elementarių dvimatės grafikos sąsajų, skirtų transformuoti, modelis-kodas transformacijomis. Pagrindinės „graphics“ sąsajos yra „Bitmap“, „Canvas“, „Color“, „Drawable“, „Paint“, „Picture“, „Point“ bei „Rect“.

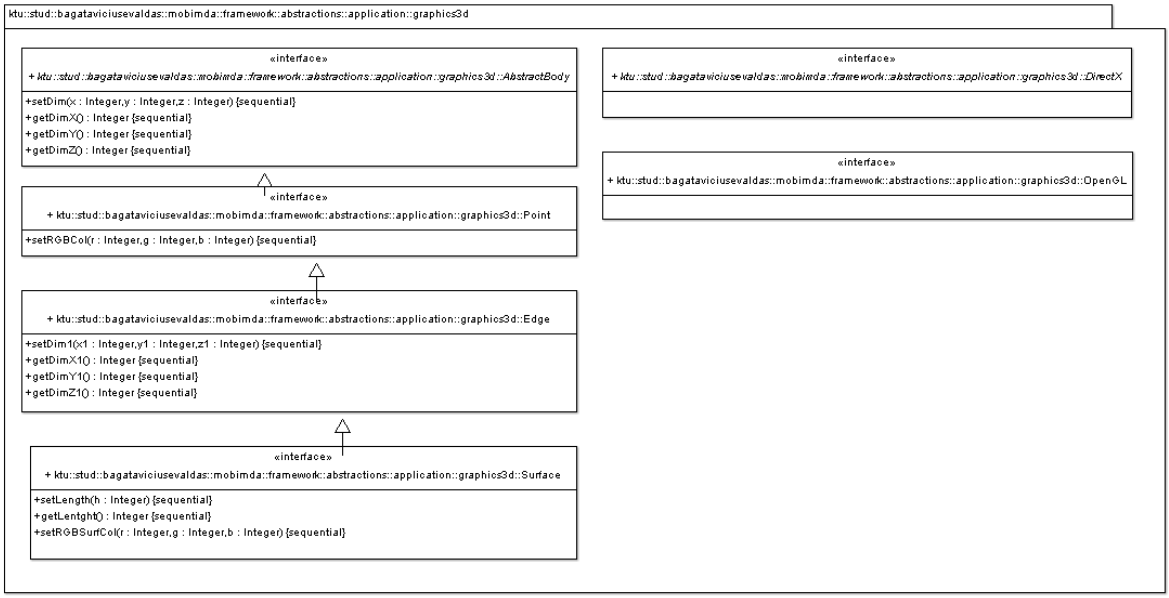
„framework::abstractions::application::events“ paketo aprašymas:



3.8 pav.: „events“ paketas

Paketas „events“ sudarytas iš sąsajų, kurios bus reikalingos formuoti įvykius, iš modelio į kodą. Šiuo atveju įvykis apibrėžiamas, į vartotojo atliktus veiksmus grafinėje sąsajoje.

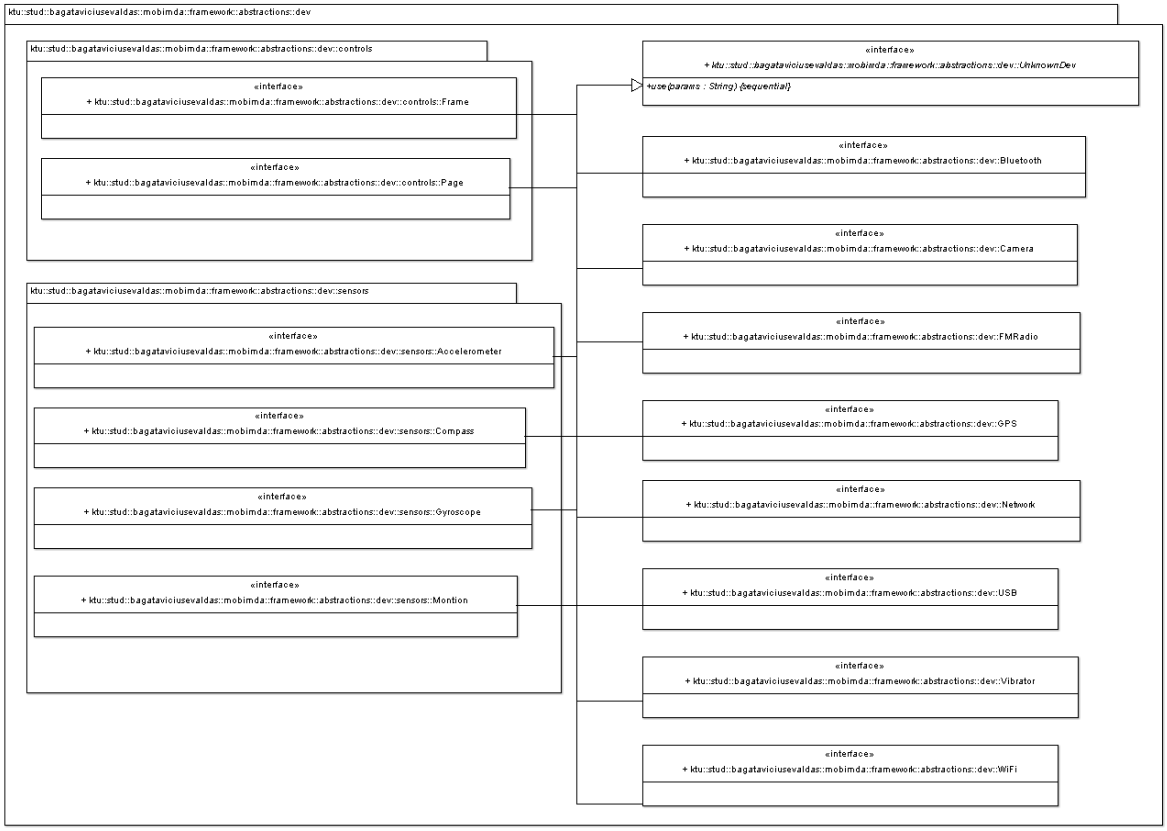
„framework::abstractions::application::graphics3d“ paketo aprašymas:



3.9 pav.: „graphics3d“ paketas

Paketas „graphics3d“ skirtas, sudaryti programos kodą, kuris transformacijų metu, pagal mobilios operacinės sistemos galimybes, formuotų erdvės efektus.

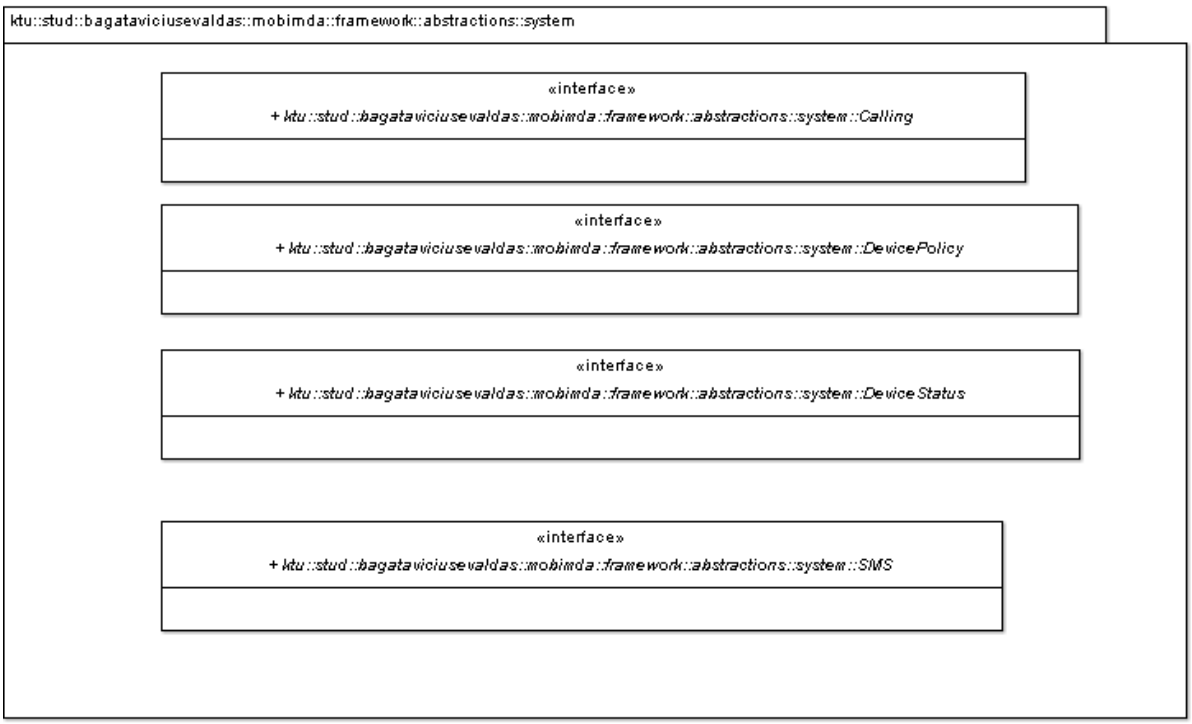
„framework::abstractions::dev“ paketo aprašymas



3.10 pav.: „dev“ paketas

Pakete „dev“ yra abstrakčios sąsajos, kurios sudarys mobilų programinį kodą, naudojantis mobilių įrenginių aprašymu.

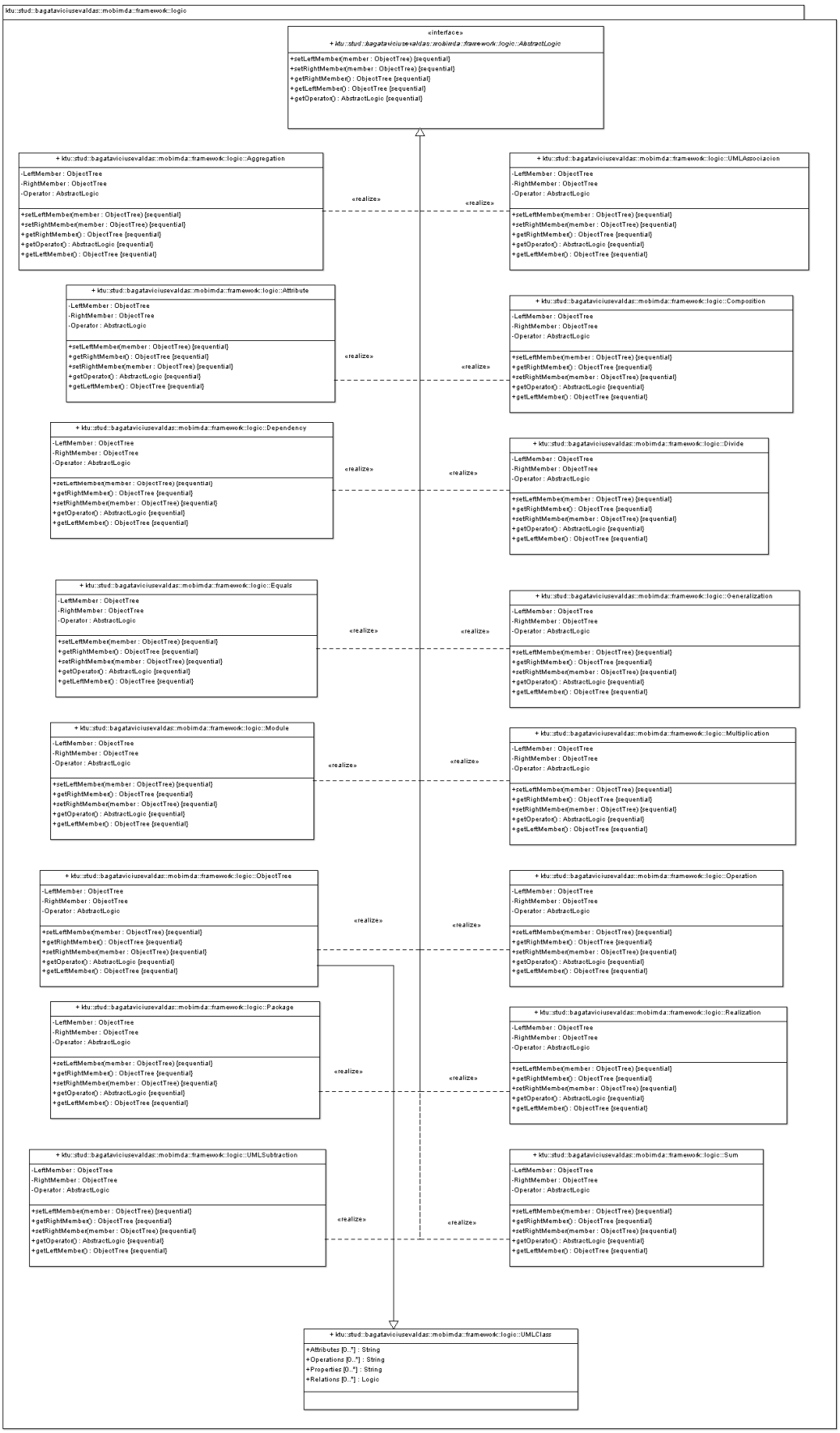
„framework::abstractions::system“ paketo aprašymas



3.11 pav.: „abstractions::system“ paketas

Pakete „abstractions::system“ aprašytos, mobilaus įrenginio vidinės posistemių sąsajos, skirtos formuoti mobilaus įrenginio programinį kodą.

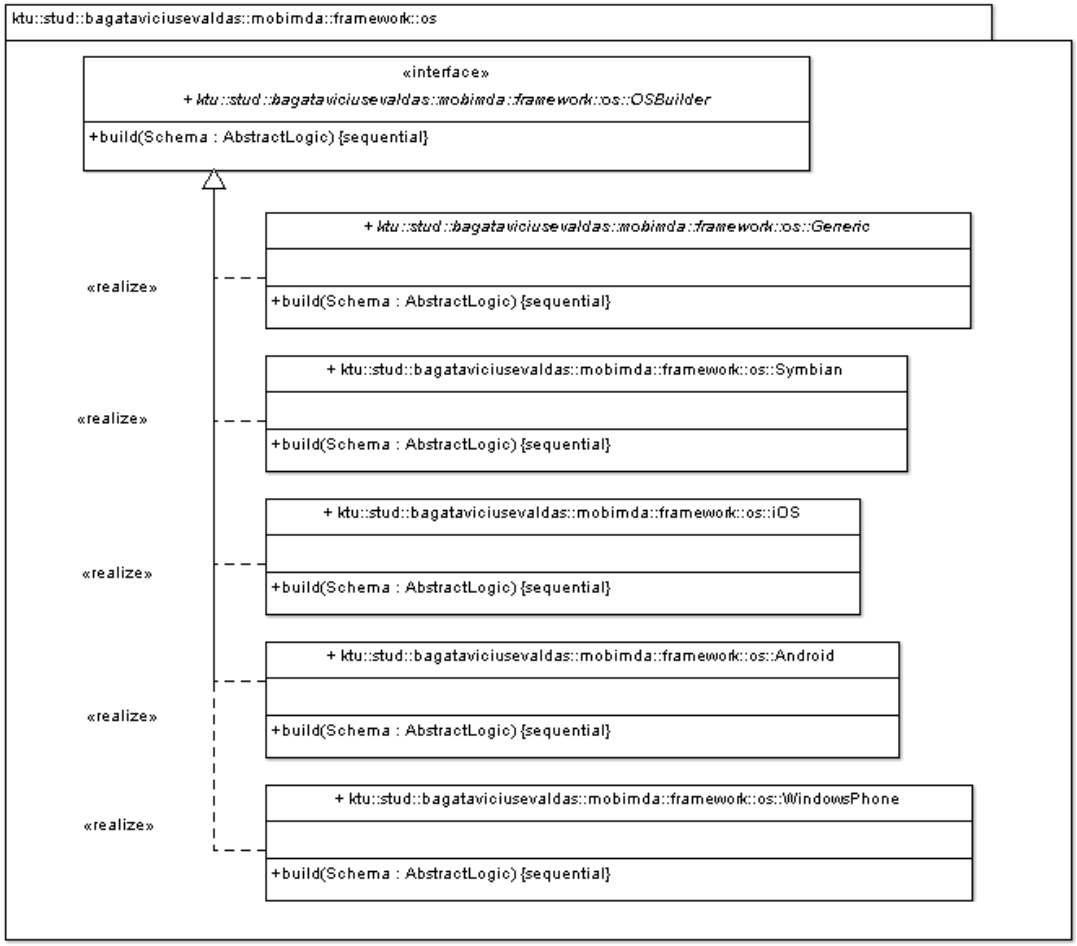
„framework::logic“ paketo aprašymas



3.12 pav.: „logic“ paketas

Paketas „logic“ skirtas, sudaryti objektų medžiui, iš UML diagramos bei kodo aprašo. Manipuliuojant objektų medžiu, sudaromos transformacijos tarp modelio ir kodo, išsaugojama ne tik modelis bet ir programinės įrangos logika.

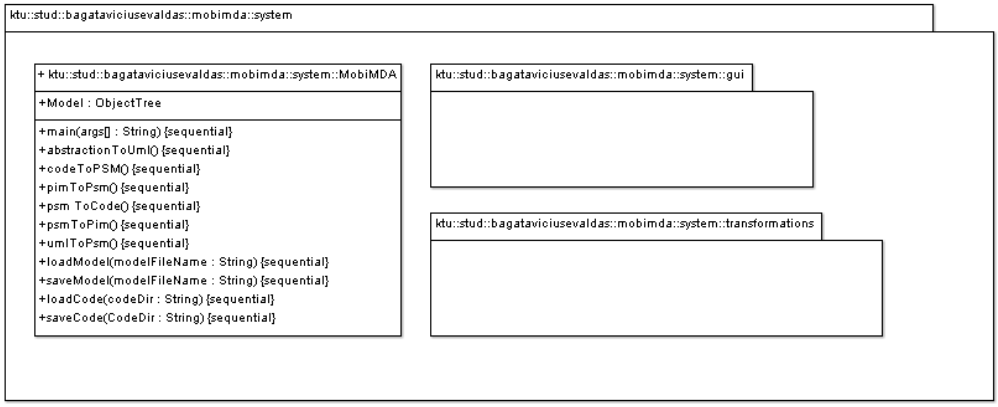
„framework::os“ paketo aprašymas



3.13 pav.: „os“ paketas

Pakete „os“ yra pagrindinės klasės, kurios pagal „factory pattern“ šabloną, generuos kodą arba modelį iš sudaryto sistemos objektų medžio.

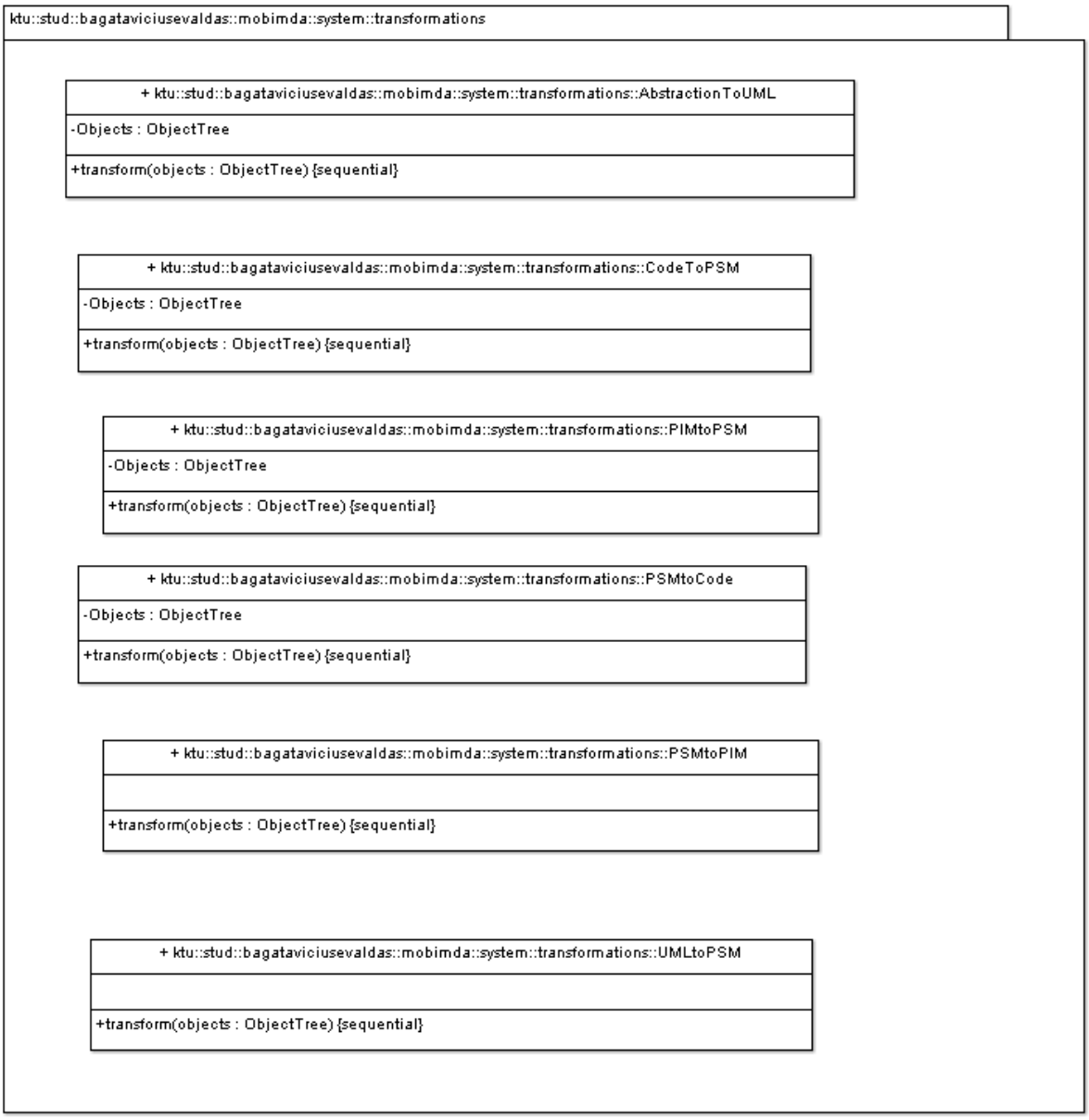
„system“ paketo aprašymas:



3.14 pav.: „system“ paketas

Pakete „system“ aprašyta MobiMDA transformavimo priemonės paleidimo klasės.

„system::transformations“ paketo aprašymas



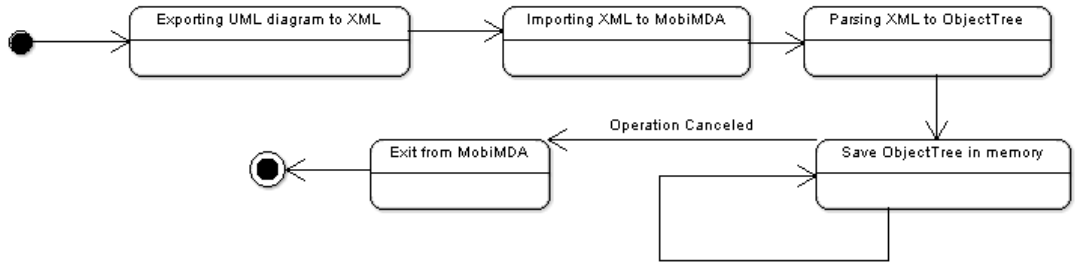
3.15 pav.: „transformations“ paketas

Pakete „transformations“ aprašytos galimo transformacijos per objektų medį.

3.2.2. Sistemos dinaminis vaizdas

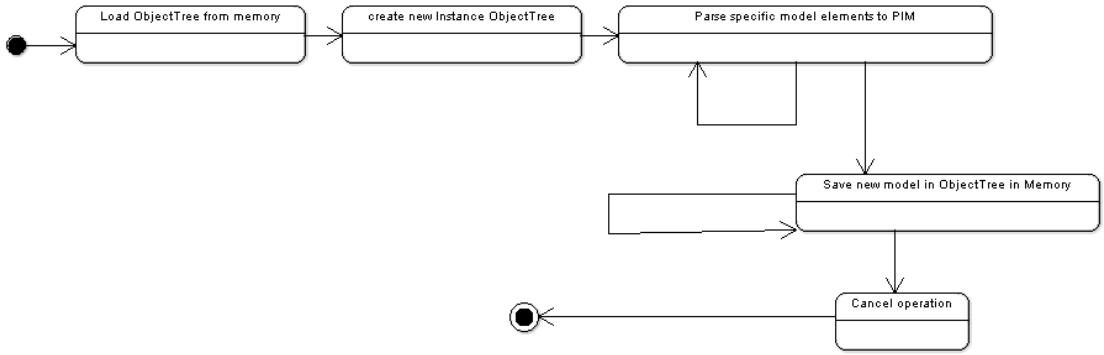
Šiame skyriuje pateiksime transformavimo priemonių sistemos dinaminį vaizdą. Kadangi transformavimo priemonė atlieka transformacijas tarp mobilių platformų bei modelių, dinaminis vaizdavimas žymiai glaustesnis, negu sistemos dinaminis modelis.

Konkrečios platformos modelio pateikimas sistemai (PSM) būsenų diagrama:



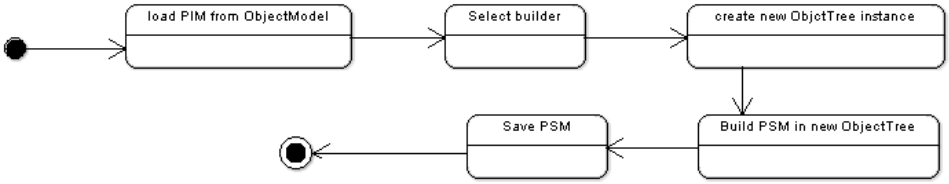
3.16 pav.: PSM įkėlimas, būsenų diagrama

PSM transformavimas į abstraktų modelį (PIM) būsenų diagrama



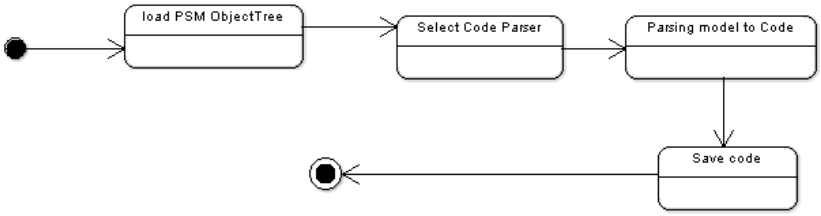
3.17 pav.: PSM-PIM būsenų diagrama

PIM transformavimas į i-osios sistemos PSM būsenų diagrama:



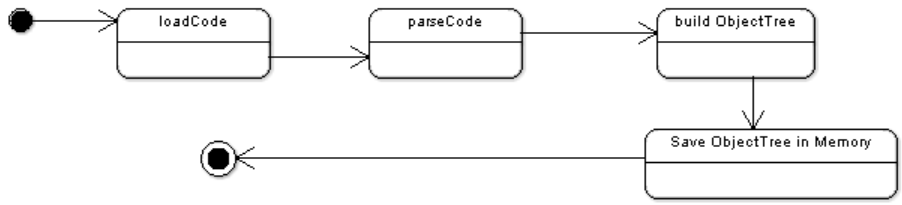
3.18 pav.: PIM-PSM būsenų diagrama

i-osios sistemos PSM transformavimas į programos kodą būsenų diagrama:



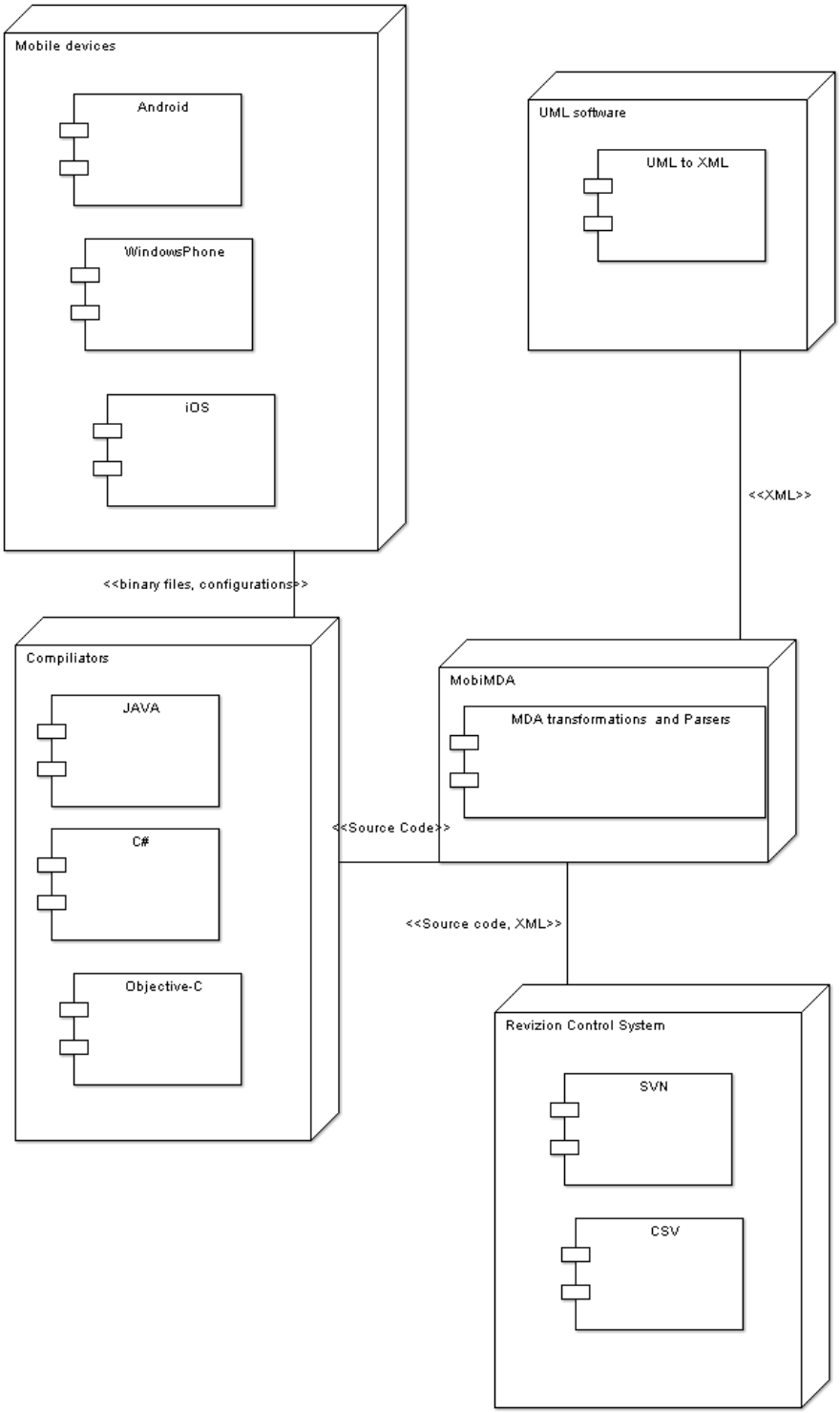
3.19 pav.: PSM-kodas būsenų diagrama

Konkretus kodo transformavimas į modelį būsenų diagrama :



3.20 pav.: Kodas-modelis būsenų diagrama

3.2.3. Transformavimo priemonių sistemos išdėstymo vaizdas



3.21 pav.: Sistemos išdėstymo vaizdas

3.2.4. Sistemos pradiniai duomenys

Sistemos pradiniai duomenys:

- XML byla (-os) kuriose saugomas UML klasių aprašai. XML byla turi atitikti OMG specifikaciją.
- Transformuojamos programos išeities kodų rinkinys. Programos išeities kodai turi būti tinkami, be sintaksės klaidų, vykdomi, laikantis tos programavimo kalbos specifikos.
- Programos dvejetainės vykdomos bylos

3.2.5. Sistemos rezultatai

Sistemos rezultatai:

- UML modelio modifikacijos XML bylose, atitinkančias OMG specifikacijas.
- Programos išeities kodų rinkiniai, keletai mobilių platformų, skirti sukompiliuoti ir paleisti programinę įrangą.

3.3. Testavimas

Ši dalis aprašo sistemos „Mobilių programų transformavimo iš vienos platformos į kitą sistemos“ (toliau MPTIVIK Sistema) testavimo planą bei sistemos testavimo strategijas ir politiką. Šios dalies aprašas skirtas sistemos, programuotojams-testuotojams, kuriuo remiantis turiniu, aprašyti sistemai testinius atvejus, nurodytomis automatinių testavimo generavimo priemonėmis, atlikti kokybišką sistemos testavimą, efektyviai ištaisyti sistemos klaidas, nepažeidžiant bendro sistemos funkcionalumo.

3.3.1. Testavimo planas

3.3.1.1. Testavimo tikslai ir objektai

- Užtikrinti, kad testavimo atvejai būtų skirti ne tik programinės įrangos klaidoms aptikti, bet ir atitiktų testuojamos programinės įrangos veikimo logiką bei specifikaciją.
- Pasiekti mažiausią programinės įrangos klaidų skaičių – aprašyti sistemai testinius atvejus, kurie apimtų mažiausią testavimo vienetą.
- Sumažinti kuriamos sistemos klaidų atsiradimo tikimybę vartotojo atžvilgiu – aprašyti visus galimus testavimo atvejus, kurie pasiektų, kiek įmanoma, didžiausią sistemos testuojamo vieneto kodo padengimą (coverage).

- Užtikrinti kokybišką testavimą, dubliuojant kuriamos sistemos išeities kodą, įtraukiant (emuliuojant) dirbtines klaidas (mutation testing), kurios parodytų ar testuojamas sistemos vienetas kokybiškai (visapusiškai) ištestuotas.
- Užtikrinti, kad atliktas programinės įrangos pakeitimas atitiktų esamus testavimo atvejus, jei to neįmanoma, atnaujinti pačius testavimo atvejus.
- Užtikrinti, kad ištaisyta sistemoje klaida nesugriaus sistemos funkcionalumo, neprieštaraus aprašytos sistemos specifikacijai ar jos architektūrai, išskyrus tik tuos atvejus, jei bus įrodyta klaida pačioje sistemos specifikacijoje.

3.3.1.2. Testavimo apimtis

- Sistemos funkcionalumo testavimas:
    - Testavimas derinimo metu.
    - Testavimas sistemos išeities kode:
        - Paketinis-komponentinis testavimas
        - Objektinis testavimas
        - Funkcinis testavimas
    - Išorinis funkcionalumo testavimas (Juodos dėžės)
        - Kodas – PSM testavimas
        - PSM-PIM testavimas
        - PIM – PSM testavimas
        - PSM – kodas testavimas
- Sistemos greitaveikos testavimas
- Sistemos funkcijų įgyvendinimo testavimas
    - Sudėtingesnių aplikacijų generavimas Android platformai
    - Sudėtingesnių aplikacijų generavimas Windows Phone platformai
- Sistemos veikimo testavimas skirtingose sistemose
    - Sistemos veikimas Windows Aplinkoje
    - Sistemos veikimas Linux aplinkoje

3.3.1.3. Pagrindiniai apribojimai

- Testavimo ir programavimo darbus atlieka vienas žmogus. Dėl to gali būti neapspręsti, kiti programinės įrangos testavimo faktoriai ir nukentėti kuriamos sistemos kokybė. Taip pat galimi programavimo darbų vėlavimai atliekant testavimą.

- Sistemos testavimui naudojami nemokami (freeware) arba atviro kodo (open source) programiniai paketai. Kuriant kokybiškus testinius atvejus šie paketai gali turėti apribojimų arba nepilnai įgyvendinta funkcionalumą, negu komerciniai.
- Testuojant kuriamą sistemą, bus nežinoma trečių šalių komponentų kokybė, ypač tų komponentų kurie yra uždari, ir jų išeities kodai nėra pasiekiami.

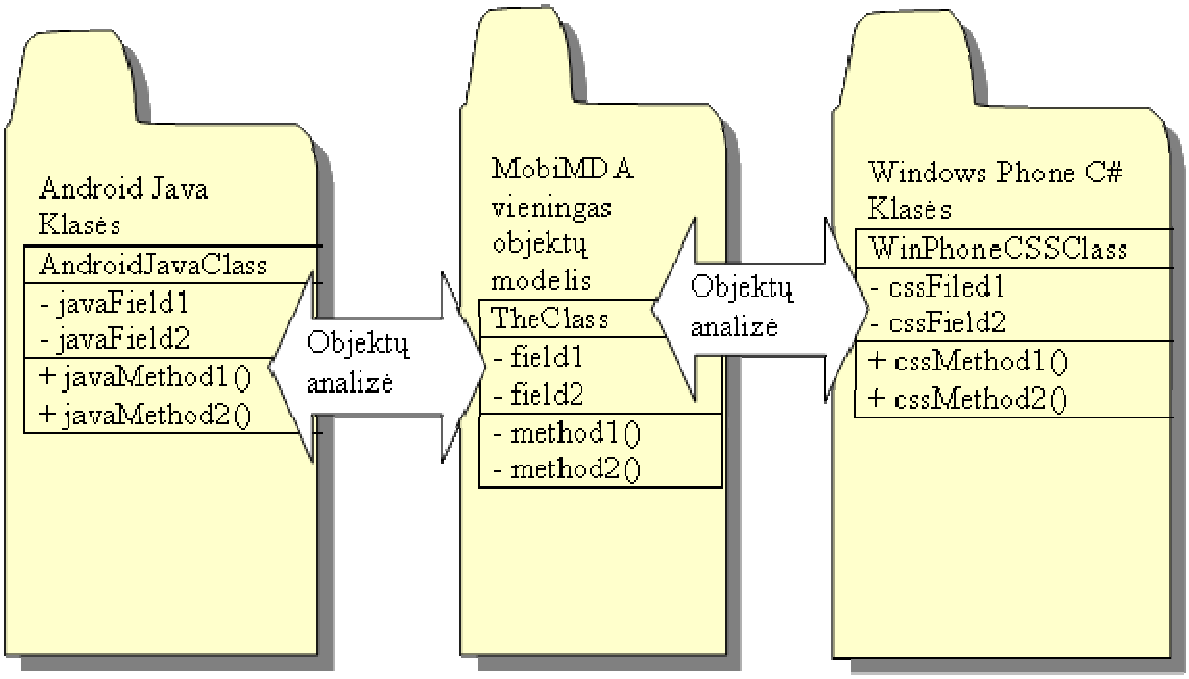
4. TYRIMO DALIS

4.1. Transformavimo priemonės (MobiMDA) veikimo metodas

Kadangi, norint transformuoti programėlę skirtą Android platformai į Windows Phone aplinką, remsimės MDA etapais [1] (PSM (Android) → PIM → PSM(Windows Phone)).

Pradiniai duomenys: Mobili aplikacija su išeitis kodu ir skaitmeniniais failais skirta konkrečiai mobiliai platformai.

Naudojamos duomenų struktūros: *Vieningas objektinio modelio aprašymas* (4.1 pav.). Tai supaprastintas, pasirinkta (šiuo atveju JAVA) programavimo kalba aprašyta objektų ir jų ryšių struktūra ir taisyklės. Pavyzdžiui, aprašomas objektas klasė kurios tipas gali būti (klasė, sąsaja) ir ji gali turėti laukus ir metodus (kurie gali būti vieši, privatūs, apriboti) , bei parametrus (statinė, abstrakti) ir ryšius (paveldėjimas, realizacija). Šis vieningas objektinis modelis turi būti vieningas ir nepriklausomas programavimo kalbai ar platformai, tam kad vykstant transformacijai, nepriklausomai kokioje aplinkoje yra programėlė, būtų vieningai atvaizduota.

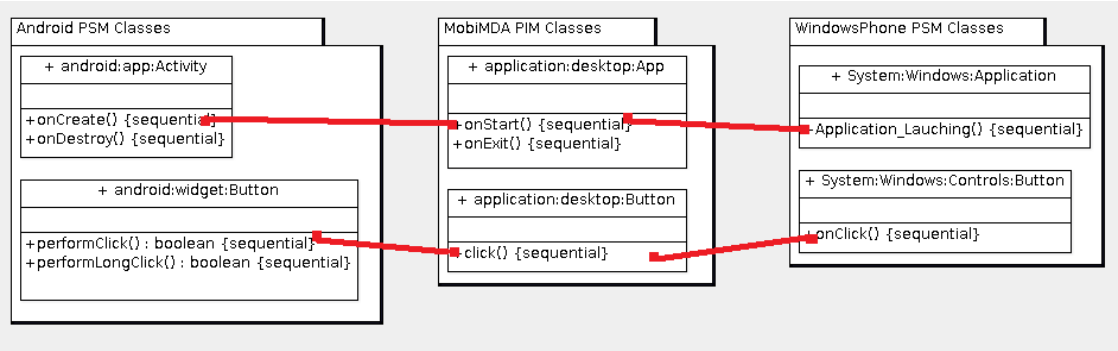


4.1 pav.: MobiMDA vieningas objektų modelis

Reflektavimo bibliotekos – skirtos analizuoti pradinės programėlės binarinius ir išeities kodo failus ir sudaryti vieningą objektinį modelį. Pasinaudojant šiuo modelio struktūra ir perduodant CASE priemonėms, atvaizduoti UML tos programėlės klasių diagramą.

PSM → PIM bibliotekų susiejimo matrica. Kiekviena programėlė turės specifines klases kurios bus susietos su tą platformą, bet bendrai bus suvokiamos kaip atominis vienetas, nepriklausantis nuo platformos. Pavyzdžiui, Android aplikacija turės klasę android::os:App, kuri reikš kaip programėlės pagrindinį įėjimo tašką. Tokią klasę galima pavadinti nepriklausomai nuo kitų platformų kaip „Application“. Taip pat galima susieti mygtukų, formų klases, bei įvykius kurios iškviečia platforma.

PIM → PSM Bibliotekų susieta matrica. Tai matrica, kuriose bendrines arba atomines klases pakeičiame į tos platformos specifines klases. Pavyzdžiui aprašyta „Application“ klasė gali būti susiejama kaip „System.Windows.Application“ klase. Bendras PSM → PIM → PSM klasių ir metodų susiejimas pavaizduotas 4.2 pav.

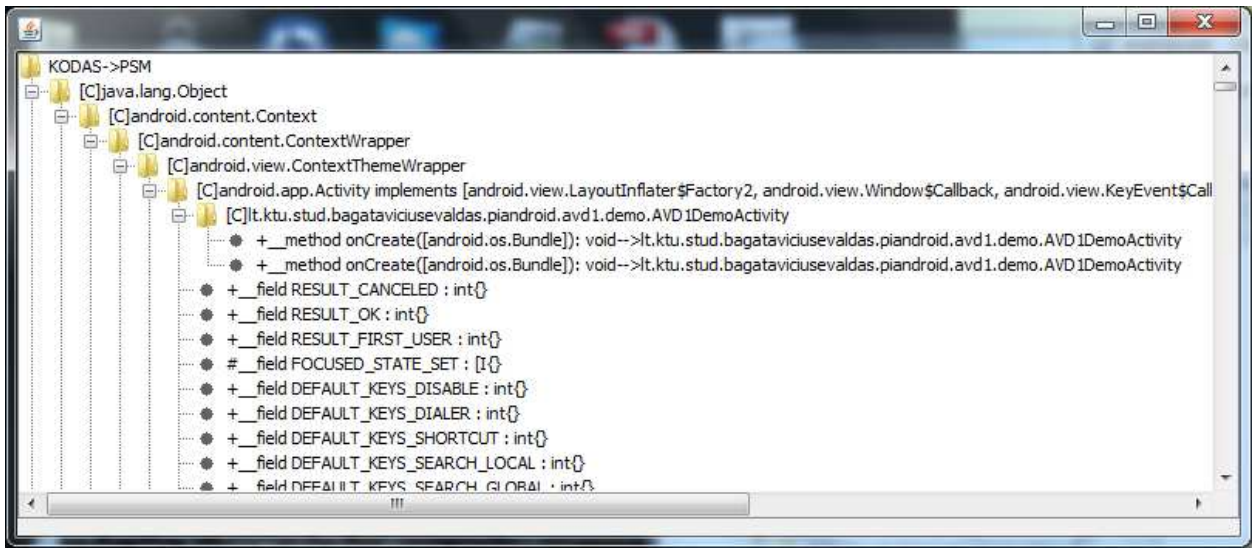


4.2 pav.: PSM->PIM->PSM klasių ir metodų susiejimas

Išeities kodo generavimo bibliotekos. Pagal platforma atrenkama programavimo kalba ir jos konstrukcijos bei tos programavimo kalbos bibliotekos, skirtos išvesti, po atliktų transformacijų, išeities kodą.

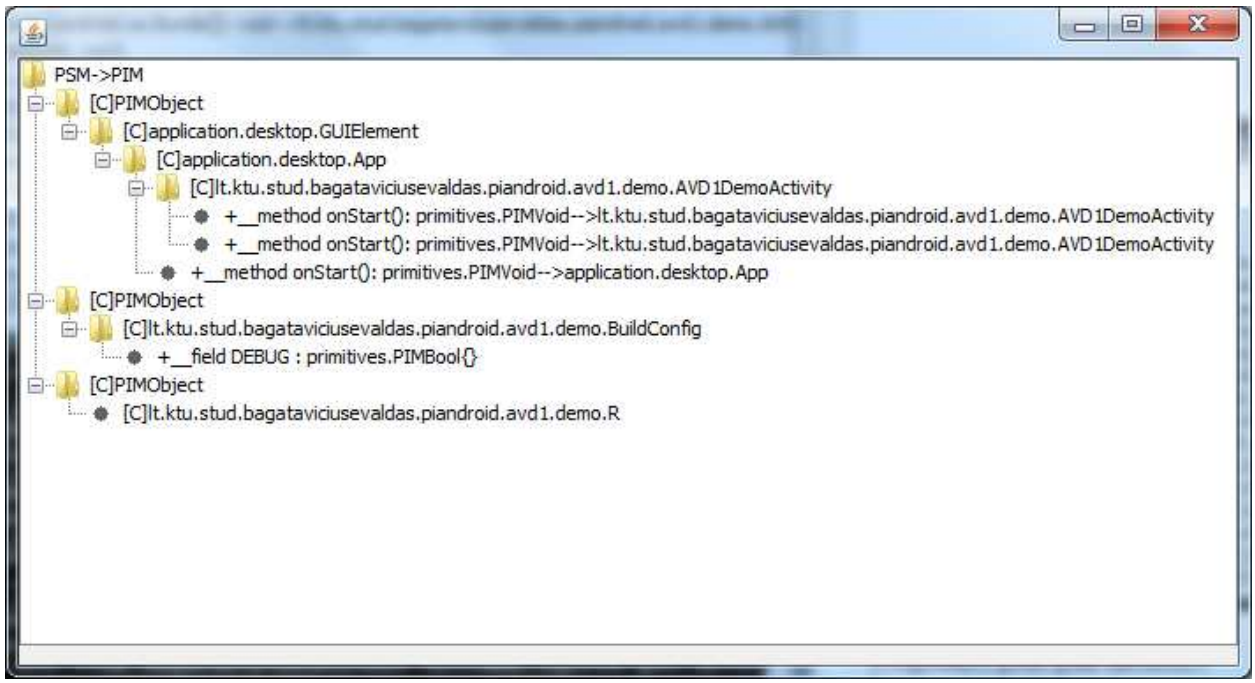
Veikimas. Veikimo schema pavaizduota 4.6 pav.

Pradinė fazė: pradinės programos kodo analizė, konkretaus modelio sudarymas (PIM) naudojant reflekcija. Analizuojamas programėlės kodas. Sudaromas objektų sąrašas ir ryšiai tarp jų. Šiame modelyje sudaromas specifinis programėlės modelis (panašiai naudojantis UML, priemonėmis, sudaroma pilna klasių diagrama 4.3 pav.).



4.3 pav.: Transformacijos kodas – PSM klasių diagrama

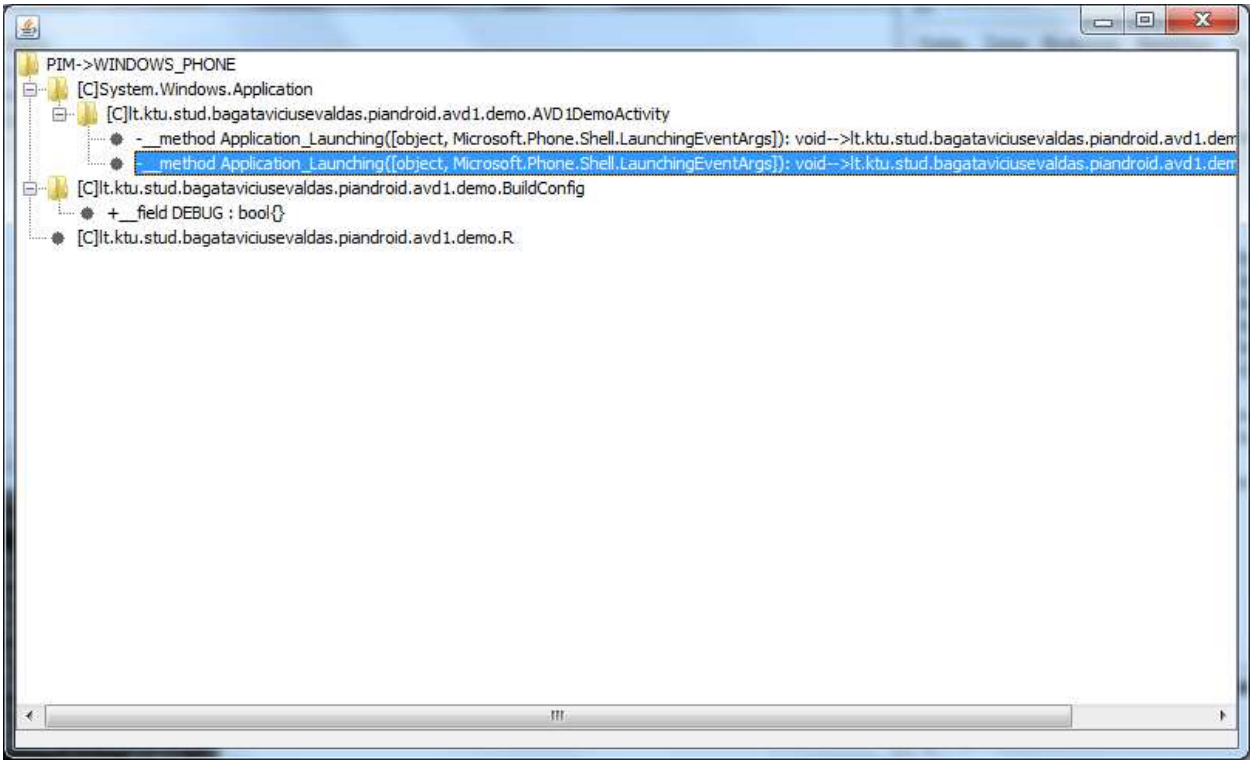
Antra fazė: konkretaus modelio vertimas į nepriklausomą modelį (PIM). Vartotojo aprašytos klasės pradinėje programėlėje netransformuojamos, jos priskiriamos prie (CIM). Transformuojami tik tos platformos specifinės instrukcijos naudojantis PSM → PIM susiejimo matrica. Gauname supaprastinta vieningą (bendrą) programėlės objektų struktūrą, kurią perduodant programinės įrangos kūrėjams butų realizuota bet kuria programavimo kalba, nepriklausoma nuo platformos (4.4 pav.).



4.4 pav.: Transformacijos PSM (Android) – PIM klasių diagrama

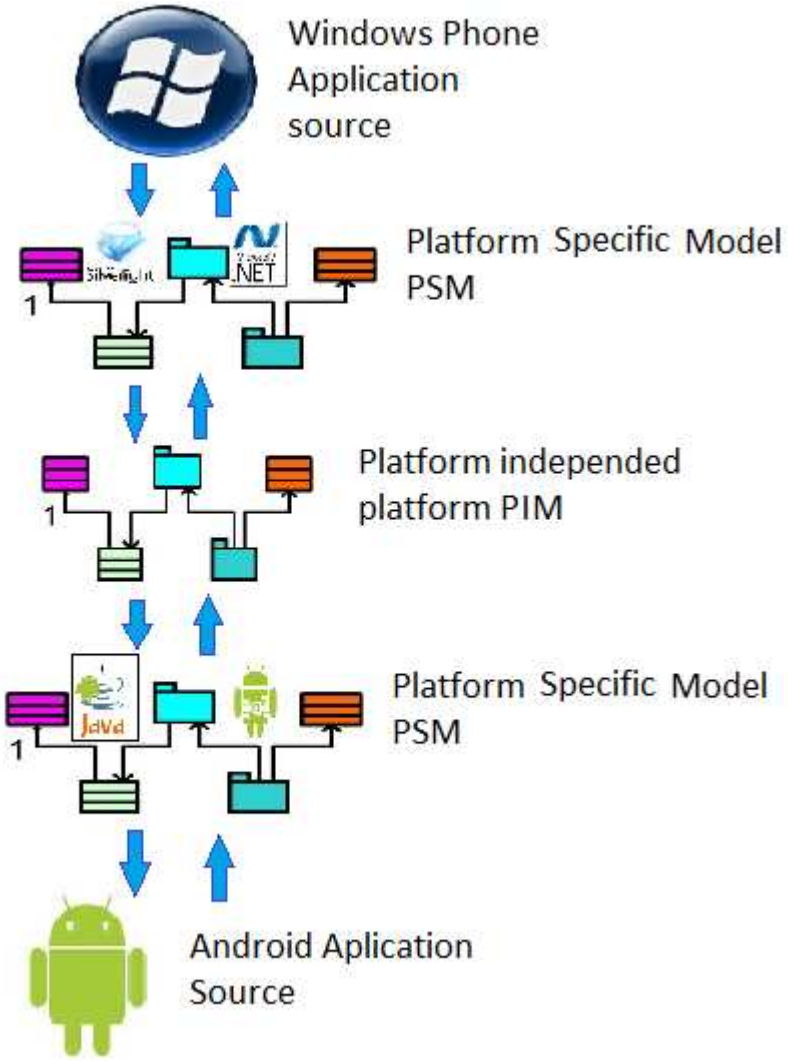
Trečia fazė: nepriklausomo modelio transformavimas į kitos sistemos specifinį modelį. Išlaikoma vartotojo aprašytų klasių struktūra ir logika (CIM), kuri nepriklauso nuo platformos.

Likęs modelis kuris priklauso tos platformos specifinių instrukcijų per PIM → PSM susiejimo matricą verčiamas į tos platformos instrukcijas (PSM), (4.5 pav.).



4.5 pav.: Transformacijos PIM – PSM (Windows Phone) klasių diagrama

Ketvirta fazė: specifinio modelio išeities kodo generavimas. Parenkama tos platformos programavimo kalbos bibliotekos ir sugeneruojamas išeities kodas.



4.6 pav.: MobiMDA transformavimo schema

5. EKSPERIMENTINĖ DALIS

Šiame skyriuje atliksime kodo transformavimo priemonių eksperimentinį tyrimą šiais aspektais:

- Programų realizavimas atskirai bei vykdant transformacijas realizavimo sąnaudų palyginimas.
- Programų vykdymo laiko palyginimas naudojant transformuotas programas bei žiniatinklio pagrindu veikiančias programas.

Programų pavyzdžiai ir išeities tekstai pateikiami (9. Priedai) skyriuje.

5.1. Realizuotų ir transformuotų programų kūrimo sąnaudų palyginimas

Pateiksime programos, kuri atlieka duomenų rikiavimą burbuliuko metodu realizavimo sąnaudas. Įvertinsime kodo metrikas, ir laiko sąnaudas (minutėmis) realizuojant šią programą šiais aspektais:

- Realizuojant programą Android aplinkoje
- Realizuojant programą Windows Phone Aplinkoje
- Atliekant transformaciją iš Android į Windows Phone, ir papildant kodą iki galutinės realizacijos.

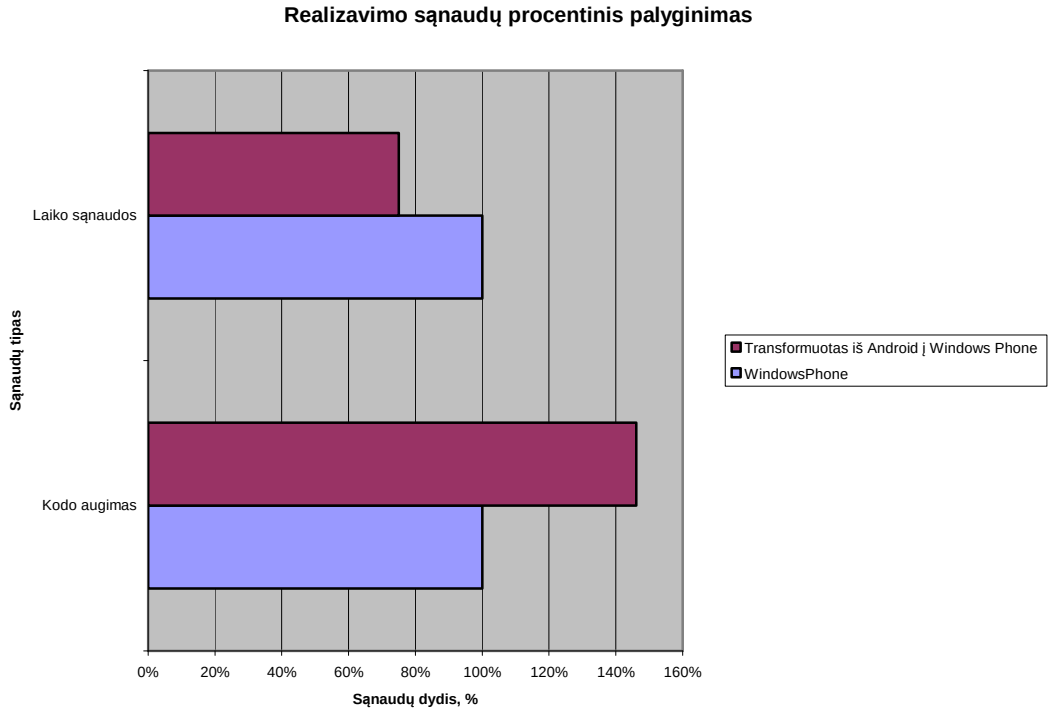
Gautos realizavimo sąnaudos pateiktos 5.1 lentelėje.

5.1 lentelė: Programos realizavimo sąnaudos, be (su) transformavimo priemonėmis

Aplinka	Android	WindwsPhone	Transformacija Android -> Windows Phone			Skirtumas trap WindowsPhone ir transformuotos programos
			Po transformacijos	Papildymas	Papildymo skirtumas su transformuotu kodu	
Kodo eilučių skaičius	92	131	124	190	66	59
Klasių skaičius	8	1	3	4	1	3
Metodų skaičius	2	2	2	2	0	0
Realizavimo laikas	15:00 min	25:00 min	0:06 min	15:00 min		-10:00 min

Pagal lentelės duomenis, jei tą pačią programą reikėtų realizuoti dvejoms platformoms, atskirai, be transformacijų, tai abejų programų realizavimas truktų 40 minučių. Tuo tarpu naudojant transformavimo priemones – 30 minučių (neįvertinat transformavimo laiko, kadangi jis pakankamai mažas). Bet transformuota ir papildyta programa, įvertinat pagal kodo metrikas išauga. Šį augimą lemia atskirtos vartotojo aprašytos klasės kurios įtraukimas kaip

nepriklausomas skaičiavimo modelis (CIM). Sąnaudų procentinis palyginimas laiko ir kodo atvaizduotas 5.1 pav.



5.1 pav.: Programos realizacijai sąnaudų procentinis palyginimas naudojant transformavimo priemones

Iš grafiko matome, kad transformuotos programos realizavimo laiko sąnaudos sumažėja 25 procentais, o kodas išauga 0,5 karto daugiau daugiau.

5.2. Transformuotos programos veikimo laikas, lyginant su žiniatinklio pagrindu veikiančiomis programomis

Forstner, Jong-Won Ko, Tai-Yeon Ku ir kitų autorių pasiūlytuose metoduose yra nuorodos naudoti standartizuotas XML bei žiniatinklio komponentus. Naudojant JavaScript realizavimo komponentus, galima realizuoti analogiškas programas, kurios veiks kaip žiniatinklio pagrindu. Šiuo atveju realizuojant tokias programos nebūtų reikalinga transformacija ir iš čia būtų galima spręsti komponentų ar programų kūrimą, žiniatinklio atvaizdavimui, mobiliems įrenginiams.

Turime programas, kurios atlieką rikiavimą burbuliuko metodu. Galimos realizacijos:

- Rikiavimas naudojant JavaScript realizuotą programą ProgJS (37 kodo eilutės, 2 metodai, 1 klasė)
- Rikiavimas naudojant Android realizuotą programą ProgA (92 kodo eilutės, 2 metodai, 8 klasės)

- Rikiavimas naudojant Windows Phone realizuotą programą ProgWP (131 kodo eilutės, 2 metodai, 1 klasė).
- Rikiavimas naudojant transformuotą programą iš Android į Windows Phone ProgATWP (190 kodo eilučių, 2 metodai ir 4 klasės).

Kadangi JavaScript programa ProgJS yra pakankamai nedidelė bet jos vykdymo laikas Android ir Windows Phone įrenginiuose palyginat su tų platformų realizuotomis programomis žymiai skiriasi rikiavimo elementų skaičiumi.

5.2 lentelė: ProgJS veikimo laiko palyginimas skirtinguose įrenginiuose su jų programų realizacijomis ir transformacijomis

	Android	
	ProgJS rikiavimas 10 tūkst. elemntų laikas, ms	ProgA, rikiavimas 1 mln. elementų laikas, ms
Bandymas 1	13893	262
Bandymas 2	12278	277
Bandymas 3	12171	267
Bandymas 4	11979	265
Bandymas 5	12091	270
Vidurkis, ms	**12482**	**268**

	Windows Phone įrenginys/emuliatorius		
	ProgJS, rikiavimas 10 tūkst. elementų laikas, ms	ProgWP rikiavimas 1 mln elementų laikas, ms	ProgATWP rikiavimas 1 mln. elementų laikas, ms
Bandymas 1	17249	11	12
Bandymas 2	17229	11	18
Bandymas 3	17108	12	11
Bandymas 4	16770	11	11
Bandymas 5	17073	12	11
Vidurkis, ms	**17086**	**11**	**13**

Iš pateiktų lentelėse duomenų, ProgJS negali būti PIM arba CIM programų kūrimo sluoksnis dėl našumo. Dėl to šiuo metu yra aktualiausia, transformuotos programos našumas nuo atskiros realizacijos programos. Palyginus programą ProgATWP ir ProgWP, transformuotos programos vykdymo laikas yra nežymiai ilgesnis, negu atskiros realizacijos programos. Šį laiką gali lemti papildomai išaugęs transformuotos programos kodo kiekis.

6. IŠVADOS

- Atliekant esamų sprendimų analizę, pagrindiniai programų transformacijų įrankiai yra paremti MDA transformavimo principais, kurie yra lengvai suvokiami realizavime. Taip pat esamuose sprendimuose yra gairių į esamas žiniatinklio technologijas, kaip XML, skirtus grafinės sąsajos atvaizdavimui bei būsenų atvaizdavimui, į juos įterpiant pagalbines priemones kaip JavaScript. Pasinaudojant šiomis gairėmis atliktas tyrimas, patikrinant ar žiniatinklio pagrindu veikiančios programos, ar jų realizavimo įrankiai leidžia efektyviai kurti mobilias programas, išvengiant transformavimo procesų, tam turint nepriklausomą skaičiavimo ir atvaizdavimo modelį (CIM ir PIM).
- MDA transformavimo pagrindu realizuotos priemonės, kurios atlieka programų transformaciją iš Android į Windows Phone. Realizuojant šias priemones buvo naudojamos vieningos duomenų struktūros, ir Android bei Windows Phone (PSM) komponentų susiejimas su vieningu nepriklausomo modeliu (PIM). Realizacijos metu yra iškilusios problemos atskiriant, nepriklausomą skaičiavimų modelį (CIM). Todėl atliekant tyrimą, transformavus programą reikalingas papildomas transformuotos programos papildymas programos išeities kodu, kad veiktų galutinai transformuota programa.
- Atlikus eksperimentą, transformuojant programas ir jas papildant, kodo eilučių skaičius išauga 0,5 karto daugiau, negu realizuojant analogišką programą atskirai. Bet transformuota programa realizuojama greičiau sutaupant 25 % realizavimo laiko. Programa realizuota žiniatinklio pagrindu, naudojant JavaScript realizaciją, veikia žymiai lėčiau, negu tą platformą realizuotos bei transformuotos programos, t. y. JavaScript programa yra 50 kartų lėtesnė. Todėl šiuo atveju parodo, kad programų transformavimas yra aktualus. Lyginant atskiros realizacijos programas su tos pačios analogiška programos transformacija, dėl transformacijų programos išaugęs kodas, veikia lėčiau, bet tai nėra žymiai didelis, lyginat su kitomis realizavimo priemonėmis.
- Apibendrinant šių priemonių realizavimą ir tyrimą, norint išvengti transformacijos metu programos kodo augimo ir išspręsti CIM skaičiavimo modelio transformavimo problemas, šiuo atveju reikėtų, pagrinde atlikti papildomus tiriamuosius darbus, kurie realizuotų papildomus įrankius arba karkasus, kurie vykdytų tik vienpusius PIM į PSM transformacijas. Tokiu atveju programos kodas bus universalus, lengvai prižiūrimas, ir nereikalaus papildomų žinių, rašant atskiroms platformoms programas, ir atliekant jų transformavimą, papildyti jas iki galutinės realizacijos.

7. LITERATŪRA

1. Boundle, D., "(User) Interface-Driven Architecture vs Model Driven Architecture". [interaktyvus]. 2009-12-01 [žiūrėta 2011-10-27] Prieiga per internetą <http://www.dougboude.com/blog/1/2009/12/Interface-Driven-Architecture-vs-Model-Driven-Architecture.cfm>.
2. Brown, A., "An introduction to Model Driven Architecture". Developer Works [interaktyvus]. 2004-02-17 [žiūrėta 2013-04-30]. Prieiga per internetą <http://www.ibm.com/developerworks/rational/library/3100.html>.
3. Choi, Y.; Yang, J.-S.; Jeong, J., "Application framework for multi platform mobile application software development," Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on , vol.01, no., pp.208,213, 15-18 Feb. 2009
4. Elleuch, N.; Khalfallah, A.; Ben Ahmed, S., "Software Architecture in Model Driven Architecture," Computational Intelligence and Intelligent Informatics, 2007. ISCIII '07. International Symposium on , vol., no., pp.219,223, 28-30 March 2007
5. Forstner, B.; Lengyel, L.; Levendovszky, T.; Mezei, G.; Kelenyi, I.; Charaf, H., "Model-based system development for embedded mobile platforms," Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006. Fourth and Third International Workshop on , vol., no., pp.10 pp.,52, 30-30 March 2006
6. Jong-Won Ko; Sung-Ho Sim; Young-Jae Song, "Test Based Model Transformation Framework for Mobile Application," Information Science and Applications (ICISA), 2011 International Conference on , vol., no., pp.1,7, 26-29 April 2011
7. Mori, G.; Paterno, F.; Santoro, C., "Design and development of multidevice user interfaces through multiple logical descriptions," Software Engineering, IEEE Transactions on , vol.30, no.8, pp.507,520, Aug. 2004
8. Tai-Yeon Ku; Dong-Hwan Park; Kyeong-Deok Moon, "Device-independent markup language," Computer and Information Science, 2005. Fourth Annual ACIS International Conference on , vol., no., pp.508,512, 2005

9. SINGH, Abhishek Pratap; JAIN, Akansha. Model-Driven Architecture with GMF. DevX [interaktyvus]. 2010 [žiūrėta 2011-10-27]. Prieiga per internetą <http://www.devx.com/architect/Article/43366/1954>.

8. SANTRUMPŲ IR TERMINŲ ŽODYNAS

- **AIM**

  Architektūrai nepriklausomas modelis (angl. *Architecture Independed Model*).

- **ASM**

  Architektūros specifinis modelis (angl. *Architecture Specific Model*).

- **ATL**

  Atlas transformavimo kalba (angl. ATLAS Transformation Language).

- **CASE įrankiai**

  Įrankiai aprašantys programinės įrangos kūrimo procesą.

- **CIM**

  Skaičiavimams nepriklausomas modelis (*angl. Computing Independent Model*).

- **Diegimo vedlys (wizard)**

  Sistemos diegimo metodas, kai vartotojui pateikiami klausimai bei paaiškinimai sistemai įdiegti ir nustatyti.

- **DRS**

  Vaizdavimo taisyklių sluoksnis (angl. *Display Rule Sheet*).

- **DSS**

  Vaizdavimo stilių sluoksnis (angl. *Display Style Sheet*).

- **Duomenų saugykla**

  Tinklinė arba fizinė sistemos dalis, skirta saugoti duomenis (duomenų bazė, kompiuterio diskas, ftp serveris t. t.)

- **Java virtulai mašina (JVM)**

  Programinė įranga, vykdanti Java kalba aprašyta programinę įrangą.

- **Juodos dėžės testavimas**

  Testavimo metodas, kuriuo testuojamos programinės įrangos dalys atžvengiant į jų parametrus bet ne į išeities kodą.

- **Komponentas**

  Sistemos biblioteka, kuri turi sąsaja, leidžianti sąveikauti su kitomis sistemomis.

- **MDA**

  (*Model Driven Architecture*) žr. Modeliais pagrįsta architektūra

- **Mobilus įrenginys**

  Tai nedidelių išmatavimų įrenginys, prietaisas skirtas išsinešti, turintis interneto arba telefoninio ryšio sąsają, atliekantis kasdienines minimalias vartotojų poreikius.

- **Modeliais pagrįsta architektūra**

  Sistemos funkcionavimo aprašymas taikant modeliavimo kalbas, nepriklausomas nuo programavimo kalbos ar architektūros

- **Modelių transformacija**

  Procesas, nuskaitantis programos modelį, ir pagal jį sudaro naują abstraktų arba konkretų modelį bei programos išeities tekstus.

- **Objektų medis**

  Kuriamos sistemos objektai aprašantys modelio struktūrą ir logiką.

- **OO**

  Objektiškai orientuota (angl. Object Oriented).

- **Operacinė sistema**

  Speciali programinė įranga, užtikrinanti vartotojo sąsają ir kompiuterio techninės įrangos, taikomųjų programų bei duomenų valdymą.

- **PIM**

  (Platform Independet Model) – modeliais aprašyta sistema neprikasoma nuo architektūros.

- **Platforma**

  Mobilaus įrenginio operacinė sistema, ar jos komponentai, fizinės detalės skirtos mobiliam įrenginiui funkcionuoti.

- **Programos (sistemos) biblioteka**

  Sistemos arba programos elementas leidžiantis išplėsti funkcionalumą.

- **Programos išeities tekstai (kodas)**

  programavimo kalbos tekstas (kodas) skirtas žmogui, suprasti kokius veiksmus vykdys kompiuteris, tiek pat skirtas kompiuteriui, transliuoti programos kodą į kompiuterio vykdomą (mašininį) kodą.

- **Programos transliavimas (kompiliavimas)**

  Programos išeities tekstų (kodo) vertimas į skaitmeninį (mašininį) kodą.

- **Projektavimo (modeliavimo) sistema**

  Sistema skirta sukurti programų arba sistemų specifikaciją remiantis projektais (modeliais).

- **PSM**

  Tam tikrai platformai aprašytas konkretus modelis

- **Reinžinerija**

  Atvirkščias procesas, kai esama sistema kuriama iš naujo naujai architektūrai, dažniausiai žinant jos funkcijas, bet nežinant jos išeities kodo.

- **Sistemos konfigūracija**

  Duomenų rinkinys skirtas sistemai nustatyti ir valdyti.

- **Sistemos sutrikimas (pakibimas)**

  Sistemos būsena, kai sistema atlieka veiksmus neribotu laiku ir tampa neprieinama vartotojui.

- **Specifikacija**

  Dokumentas aprašantis sistemos projektą ir jos atliekamas funkcijas.

- **UML**

Vieninga modeliavimo kalba (*Unified Modeling Language*), modeliavimo ir specifikacijų kūrimo kalba, sukurta OMG grupės, skirta specifikuoti, atvaizduoti ir konstruoti objektiškai orientuotų programų dokumentus.

- **XML**

(*Extensible Markup Language*) yra W3C rekomenduojama bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba.

9. PRIEDAI

9.1. JavaScript programos išeities kodas

```
<script>
var a = [];


 function genArr() {
 var arr = [];
for (var i = 0, l = 10000; i < l; i++) {
    arr.push(Math.round(Math.random() * l))
}

return arr;
}

function bubbleSort(a)
{
    var swapped,  t1 = 0, t2 = 0;
      a = genArr();
      t1 = new Date().getTime();
    do {
        swapped = false;
        for (var i=0; i < a.length-1; i++) {
            if (a[i] > a[i+1]) {
                var temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                swapped = true;
            }
        }
    } while (swapped);

      t2 = new Date().getTime();
      alert(t2-t1);
}
</script>

<html>

<input type="button" value="sort" onclick="bubbleSort(a);" >
```

9.2. Android programos išeities kodas

```
package lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo;


import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class AVD1DemoActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        int b = 12;
        int c = 20;
        final int j = b + c;
        setContentView(R.layout.main);
```

```
        final TextView tv = (TextView) findViewById(R.id.text1);

        final Button button = (Button) findViewById(R.id.button1);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                  tv.setText("sorting");

                  boolean swapped;
                  long t1 = 0, t2 =0;
                  int a[] = getArr();

                  t1 = System.currentTimeMillis();

                  do {
                        swapped = false;
                        for (int i = 0; i < a.length - 1; i ++) {
                              if (a[i] > a[i+1]) {
                                    int temp = a[i];
                                    a[i+1] = temp;
                              swapped = true;
                              }
                        }
                  } while (swapped);
                  t2 = System.currentTimeMillis();
               tv.setText(Long.toString(t2-t1));
            }
        });
    }


    private int [] getArr() {
      int arr [] = new int[1000000];

      for (int i = 0; i < arr.length ; i++) {
         arr[i] = (int) Math.round(Math.random() * 1000);
      }

      return arr;
    }
}

/* AUTO-GENERATED FILE.  DO NOT MODIFY.

 *

 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int button1=0x7f050000;
        public static final int text1=0x7f050001;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

```
/** Automatically generated file. DO NOT MODIFY */
package lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo;
public final class BuildConfig {
   public final static boolean DEBUG = true;
}
```

9.3. Windows Phone išeities kodas

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;

namespace Rikiavimas_original
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            textBlock1.Text = " sorting ";
            bool swapped;
            DateTime  t1 , t2 ;
            int[] a = null;
            a = genArr();

            t1 = DateTime.Now;

           do
            {
                swapped = false;
                for (int i = 0; i < a.Length - 1; i++)
                {
                    if (a[i] > a[i + 1])
                    {
                        int temp = a[i];
                        a[i + 1] = temp;
                        swapped = true;
                    }
                }
            } while (swapped);

            t2 = DateTime.Now;

            textBlock1.Text = t2.Subtract(t1).ToString();
        }

        private int[] genArr()
        {
            int [] arr = new int[1000000];

            for (int i = 0; i < arr.Length; i++)
            {
```

```
                arr[i] = (new Random(10000)).Next();
            }

            return arr;
        }
    }
}
<phone:PhoneApplicationPage
    x:Class="Rikiavimas_original.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION"
style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="page name" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <TextBlock Height="41" HorizontalAlignment="Left"
Margin="37,22,0,0" Name="textBlock1" Text="laikas" VerticalAlignment="Top"
Width="216" />

            <Button Content="Rikiuoti" Height="72" HorizontalAlignment="Left"
Margin="12,69,0,0" Name="button1" VerticalAlignment="Top" Width="160"
Click="button1_Click" />
        </Grid>
    </Grid>

    <!--Sample code showing usage of ApplicationBar-->
    <!--<phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
            <shell:ApplicationBarIconButton
IconUri="/Images/appbar_button1.png" Text="Button 1"/>
            <shell:ApplicationBarIconButton
IconUri="/Images/appbar_button2.png" Text="Button 2"/>
            <shell:ApplicationBar.MenuItems>
                <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
                <shell:ApplicationBarMenuItem Text="MenuItem 2"/>
            </shell:ApplicationBar.MenuItems>
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>-->


</phone:PhoneApplicationPage>
```

```
<phone:PhoneApplicationPage
    x:Class="Rikiavimas_original.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION"
Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="page name" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <TextBlock Height="41" HorizontalAlignment="Left"
Margin="37,22,0,0" Name="textBlock1" Text="laikas" VerticalAlignment="Top"
Width="216" />

            <Button Content="Rikiuoti" Height="72" HorizontalAlignment="Left"
Margin="12,69,0,0" Name="button1" VerticalAlignment="Top" Width="160"
Click="button1_Click" />

        </Grid>
    </Grid>

    <!--Sample code showing usage of ApplicationBar-->
    <!--<phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
            <shell:ApplicationBarIconButton
IconUri="/Images/appbar_button1.png" Text="Button 1"/>
            <shell:ApplicationBarIconButton
IconUri="/Images/appbar_button2.png" Text="Button 2"/>
            <shell:ApplicationBar.MenuItems>
                <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
                <shell:ApplicationBarMenuItem Text="MenuItem 2"/>
            </shell:ApplicationBar.MenuItems>
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>-->

</phone:PhoneApplicationPage>
```

9.4. Iš Android į Windows Phone transformuota programa

```
namespace lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo

{
    public partial class R
```

```
    {
        public R()
        {

        }

    }
}
<Application
    x:Class="lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo.R"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone">

    <Application.Resources>
    </Application.Resources>
    <Application.ApplicationLifetimeObjects>
        <shell:PhoneApplicationService
        />
    </Application.ApplicationLifetimeObjects>
</Application>


namespace lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo

{

    public partial class BuildConfig
    {
        public static bool DEBUG;

        public BuildConfig()
        {

        }

    }
}

<Application
    x:Class="lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo.BuildConfig"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone">
    <Application.Resources>
    </Application.Resources>

    <Application.ApplicationLifetimeObjects>
        <shell:PhoneApplicationService
        />
    </Application.ApplicationLifetimeObjects>
</Application>
namespace lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo
{
    public partial class AVD1DemoActivity: System.Windows.Application
    {
        public AVD1DemoActivity()
        {

        }
```

```
        private int[] genArr()
        {
        }

       private void Application_Launching(object argObject0,
Microsoft.Phone.Shell.LaunchingEventArgs argLaunchingEventArgs1)
        {
        }

    }
}
<Application

x:Class="lt.ktu.stud.bagataviciusevaldas.piandroid.avd1.demo.AVD1DemoActivity
"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone">


    <Application.Resources>
    </Application.Resources>

    <Application.ApplicationLifetimeObjects>
        <shell:PhoneApplicationService
            Launching=Application_Launching
        />
    </Application.ApplicationLifetimeObjects>
</Application>
```