KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Tomas Žemaitis

Nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams karkasas

Magistro darbas

Darbo vadovas
prof. E. Bareiša

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PRAGRAMŲ INŽINERIJOS KATEDRA

Tomas Žemaitis

Nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams karkasas

Magistro darbas

Vadovas
prof. E. Bareiša
2010-05

Atliko
IFM-4/2 gr. stud.
Tomas Žemaitis
2010-05

Recenzentas
dr. I. Lagzdinytė
2010-05

Kaunas, 2010

TURINYS

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

1. ĮVADAS

Šiomis dienomis labai greitai tobulėja mobilios technologijos. Į išmaniuosius įrenginius montuojamų procesorių taktinis dažnis jau pasiekė 1Ghz, ekranai tapo labai didelės raiškos, jautrūs lietimui bei pasižymintys labai kokybišku spalvų atkūrimu. Dėl daugybės į išmaniuosius telefonus montuojamų papildomų įtaisų, jų panaudojimo sritis vis plečiasi, jų populiarumas auga. Kartu su aparatūros tobulėjimu, tobulėja ir jiems skirta programinė įranga. Per paskutinius kelis metus pasirodė net trys naujos išmaniesiems telefonams skirtos operacinės sistemos.

Visos šios naujovės ir greitas jų tobulėjimas labai apsunkina programuotojų darbą. Labai sunku suspėti su visomis naujomis technologijomis. Kiekviena atsiradusi platforma atsineša ir savo programavimo principus. Norint, kad sukurta programinė įranga pasiektų kuo didesnę vartotojų auditoriją, būtina ją pateikti kelioms skirtingoms ir dažnai nesuderinamoms platformoms. Tai pareikalauja didesnių laiko sąnaudų, daugiau žmogiškųjų resursų kas sąlygoja ir didesnius programinės įrangos kūrimo kaštus. Taip pat, labai trūksta profesionalių specialistų gerai išmanančių šias visai neseniai rinkoje pasirodžiusias technologijas.

Tyrimo objektas – nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams kūrimo galimybės.

Tyrimo tikslas – realizuoti ir ištirti nuo platformos nepriklausomą programinės įrangos karkasą, palengvinantį programinės įrangos išmaniesiems telefonams kūrimą.

Tyrimo uždaviniai:

1. išanalizuoti nuo platformos nepriklausomos programinės įrangos kūrimo galimybes išmaniesiems telefonams, apžvelgti tokiai programinei įrangai kurti skirtų įrankių realizacijas.

2. suprojektuoti ir realizuoti priemonių rinkinį nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams kūrimui.

3. paliginti ir įvertinti sukurto įrankio panaudojimo galimybes bei ištirti jo efektyvumą.

Trumpai apžvelgsime darbo turinį sudarančius pagrindinius skirsnius:

Nuo platformos nepriklausomos programinės įrangos realizavimo metodų analizė. Aprašoma tyrimo sritis, objektas, problemos bei aktualumas. Išanalizuojami galimi problemos sprendimai, jų trūkumai bei privalumai.

Nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams kūrimo metodas. Detaliai aprašomi pasirinkti sprendimai nagrinėjamoms problemoms spręsti.

Nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams karkaso projektavimas remiantis siūlomu modeliu. Ši dalis apima sistemos komponentų projektavimą bei realizavimą. Pateikiami reikalavimai sistemai, sistemos statinis ir dinaminis vaizdai.

Eksperimentinis realizuoto karkaso tyrimas. Atliekamas sukurtos sistemos eksperimentinis tyrimas, įvertinamas jos efektyvumas. Ištiriamos efektyvumo pagerinimo galimybės. Suprojektuojami bei realizuojami pakeitimai.

Išvados. Šioje dalyje pateikiama apibendrinta informacija apie sukurtos sistemos funkcionalumą, informacija apie atliktų tyrimų ir eksperimentų rezultatus.

2. NUO PLATFORMOS NEPRIKLAUSOMOS PROGRAMINĖS ĮRANGOS REALIZAVIMO METODŲ ANALIZĖ

2.1. Tyrimo sritis, objektas ir problema

Pagrindinis ***darbo tikslas – realizuoti efektyvų įrankį (karkasą) nuo platformos nepriklausomai programinei įrangai, skirtai išmaniesiems telefonams, kurti. Įrankis turi palengvinanti, pagreitinti ir atpiginti programinės įrangos išmaniesiems telefonams kūrimo procesą.*** Naudojantis realizuotu karkasu turi būti galima realizuoti tiek grafinę vartotojo sąsają, tiek programinės įrangos funkcionalumą. Taigi, šio darbo ***tyrimo sritis - nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams kūrimo priemonių realizavimo galimybės***.

Šiomis dienomis labai greitai tobulėja mobilios technologijos: tiek ir patys išmaniųjų telefonų aparatai tiek ir jų programinė įranga. Išmaniųjų telefonų procesorių dažnis jau pasiekė 1GHz ribą. Taip pat labai ištobulėjo išmaniųjų telefonų ekranai, jie tapo labai didelės raiškos, jautrūs lietimui bei pasižymintys labai kokybišku spalvų atkūrimu. Labai patobulėjo ir mobiliojo interneto tinklo infrastruktūra. Šiuolaikiniai išmanieji telefonai turi ir 3G ir Wi-Fi tinklo adapterius, todėl jų savininkas nuolat gali turėti ryši su globaliuoju tinklu. Į naujausius išmaniuosius telefono įrenginius montuojama daugybė sensorių bei imtuvų: GPS imtuvas, akselerometras, artumo sensorius, giroskopas, geomagnetinis sensorius ir kiti. Šios naujos technologijos labai praplėtė mobiliųjų telefonų panaudojimo galimybes. Jie tapo lyg miniatiūriniai kompiuteriai, daugiafunkciniai įrenginiai, kurių pagalba galima patogiai naršyti Internete, klausytis muzikos, žiūrėti video medžiagą, fotografuoti bei atlikti kitas užduotis.

Taip išsivysčius aparatūrai, atsirado galimybė ir poreikis tobulesnės programinės įrangos panaudojimui. Per paskutinius kelis metus atsirado net trys naujos išmaniųjų įrenginių platformos: iPhone, Android ir Bada OS [1]. Šie rinkos naujokai labai aukštai iškėlė grafinės vartotojo sąsajos kartelę. Ji tapo ypatingai dinamiška, interaktyvi ir vartotojui patraukti. Beveik kiekviename žingsnyje atliekama vienokia ar kitokia animacija.

Visos šios naujovės ir greitas jų tobulėjimas labai apsunkina programuotojų darbą. Labai sunku suspėti su visomis naujomis technologijomis. Kiekviena atsiradusi platforma atsineša ir savo programavimo principus.

Dėl didelės konkurencijos ir daugybės skirtingų gamintojų standartai taip ir nenusistovėjo, todėl šiandien mes turime daugybę skirtingų išmaniųjų telefono platformų. Ši aplinkybė labai

apsunkina programinės įrangos išmaniesiems telefonams kūrimą. Šiuo metu išmaniuosiuose telefonuose naudojamų operacinių sistemų sąrašas pateiktas 1-oje lentelėje. Norint, kad sukurta programinė įranga pasiektų kuo didesnę vartotojų auditoriją, būtina ją pateikti kelioms skirtingoms ir dažnai nesuderinamoms platformoms. Tai pareikalauja didesnių laiko sąnaudų, daugiau žmogiškųjų resursų, kas sąlygoja ir didesnius programinės įrangos kūrimo kaštus. Be to, labai trūksta profesionalių specialistų, gerai išmanančių šias visai neseniai rinkoje pasirodžiusias technologijas.

1 lentelė. Operacinės sistemos, naudojamos išmaniuosiuose telefonuose.

Pavadinimas	Naudojama kalba	Trumpas aprašymas
Symbian OS	C/C++	Populiariausia ir plačiausiai naudojama operacinė sistema išmaniuosiuose telefonuose. Savo telefonuose ją naudoja tokie gamintojai kaip: BenQ, Fujitsu, LG, Mitsubishi, Motorola, Nokia, Samsung, Sharp ir Sonny Ericsson.
Windows Mobile (CE)	C/C++	Daugiausia naudojama Azijos rinkoje. 2007 metais išleistos dvi patobulintos versijos Mobile 6 Professional ir Mobile 6 Standart. Sistema kritikuojama dėl vartotojo sąsajos, kuri nėra optimizuota telefono valdymui pirštų prilietimais (Pritaikyta valdyti liečiamąja lazdele).
Android OS	Java	Šią operacinę sistemą sukūrė Google korporacija. Tai atviro kodo, kilusi iš Linux, greitai populiarėjanti ir daug žadanti.
iPhone	C/C++/ ObjectiveC	Naudojama išskirtinai tik iPhone išmaniajame telefone. Iki antros OS versijos nebuvo leidžiama instaliuoti trečios šalies programinės įrangos.
BlackBarry	Java	Sistema akcentuota į lengvą valdymą ir buvo projektuojama specialiai verslo poreikiams. Naujausios sistemos versijos jau turi ir pilną daugialypės terpės palaikymą.
Bada OS	C/C++	Pati naujausia, vis dar kuriama, išmaniesiems telefonams skirta operacinė sistema. Ją sukūrė Samsung korporacija.
Maemo	C/C++	Sukurta Nokia kompanijos „Internet Tablet" tipo išmaniesiems telefonams. Sistema sukurta Linux Debian operacinės sistemos pagrindu.

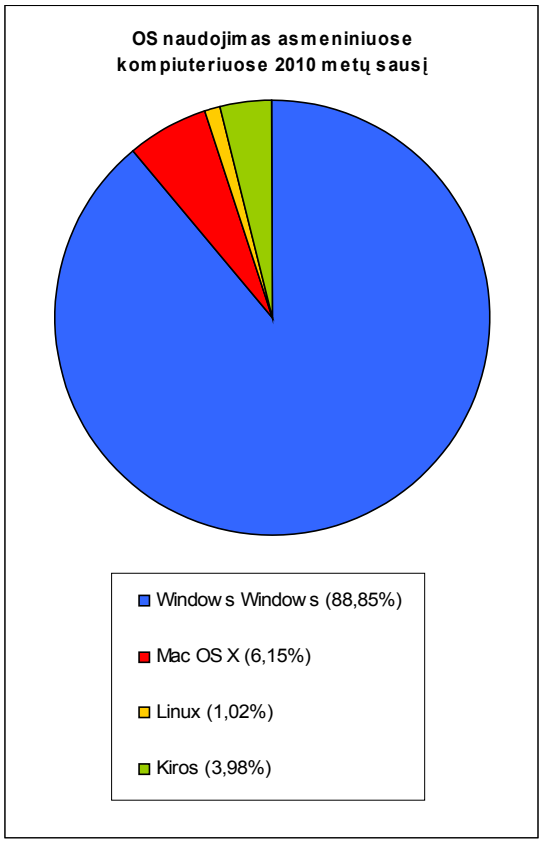
Programavimas asmeniniams kompiuteriams ir mobiliesiems telefonams turi keletą esminių skirtumų. Pirmiausia tai įtakoja skirtinga situacija rinkoje. Asmeniniuose kompiuteriuose naudojamų operacinių sistemų beveik 90% sudaro Microsoft Windows šeima. Naudojantis Microsoft .Net Framework ar JavaSE galima sukurti programinę įrangą, kuri veiks visose šios šeimos operacinėse sistemose. Tokiu būdu kuriamas produktas pasieks 90% procentų galimų vartotojų. Detalūs operacinių sistemų naudojimo asmeniniuose kompiuteriuose duomenys pateikti 1-ame paveikslėlyje.

Kaip matome 2-ame paveikslėlyje, išmaniuosiuose telefonuose operacinių sistemų paplitimo situacija kitokia. Čia nėra vienos dominuojančios platformos. Beveik kiekvienas išmaniųjų telefonų

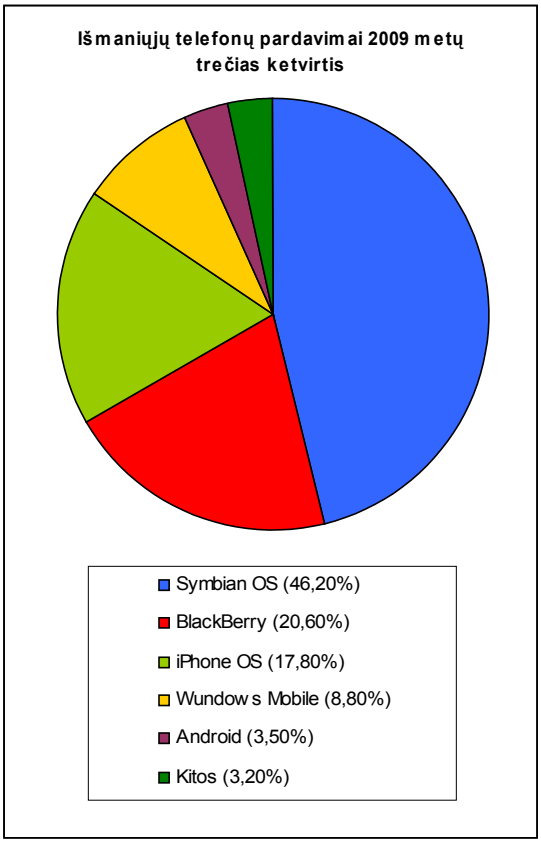
gamintojas kartu su aparatūra leidžia ir programinės įrangos komplektą – operacinę sistemą su taikomosiomis programomis tokiomis kaip kalendorius, adresų knygutė ir t.t.

Išmanieji telefonai tai daugiafunkciniai įrenginiai. Juose įmontuota daugybė skirtingos aparatūros: foto kamera, GPS imtuvas, audio/video grotuvas, akselerometras, giroskopas, geomagnetinis sensorius ir kiti. Kiekvienai aparatūros daliai gamintojas pateigia programines sąsajas, išleisdamas savo gaminamam išmaniajam telefonui skirtą programų kūrimo įrankių komplektą (angl. Software Development Kit - SDK). Deja, skirtingų operacinių sistemų programinės sąsajos nesuderinamos. Taigi, norėdami kurti programinę įrangą tarkim Symbian OS sistemai, programuotojai turi naudotis Symbian OS SDK, Android operacinei sistemai – Android SDK ir t.t.

Tiesa, Sun Microsystems yra išleidusi JVM palaikančią J2ME daugumai įmantriuosiuose įrenginiuose naudojamų operacinių sistemų. Deja, joje nepateikiamos programinės aparatūros sąsajos. Tai žymiai susiaurina Java technologijos panaudojimo galimybes išmaniuosiuose telefonuose, todėl J2ME dažniausiai naudojama tik žaidimų kūrimui.



1 pav. Operacinių sistemų pasiskirstymas asmeniniuose kompiuteriuose 2010 metų sausis [2].



2 pav. Išmaniųjų telefonų pardavimai 2009 metų trečias ketvirtis [3].

Programinės įrangos mobiliesiems telefonams išleidimo grafikai, lyginant su asmeniniais kompiuteriais, labai trumpi. Dažniausiai tai būna nedidelės programos, kurios rinkai turi būti pristatytos per 3-4 mėnesius. Programuotojas tiesiog neturi pakankamai laiko programinės įrangos architektūrai apgalvoti. Daugelis mobiliųjų telefonų platformų paremtos vienu iš populiarių dizaino šablonų. Tačiau skubėdami programuotojai nesilaiko šablonų standartų ir sumaišo visus komponentus vienoje klasėje. Programos struktūra tampa labai prastos kokybės ir darosi vis sudėtingesnė su kiekvienu nauju pakeitimu. Tai apsunkina vėlesnius jos tobulinimus ir taisymą, kas paprastai yra būtina dėl labai greitai kintančios aplinkos. Išauga programinės įrangos palaikymo kaštai.

Išmaniųjų telefonų resursai, lyginant su asmeniniais kompiuteriais, vis dar lieka labai riboti. Šiuos ribojimus lemia du pagrindiniai faktoriai:

- Įrenginio gabaritai;
- Nuolatinio energijos šaltinio nebuvimo;

Pirmasis faktorius tampa vis mažiau aktualus. Pasitelkus nano technologijas, aparatūros gabaritai ženkliai sumažėjo. Bet antrasis faktorius vis dar kelia dideles problemas. Didelės raiškos ekranams reikalingas daug didesnis skaičiavimų kiekis, todėl papildomai apkraunamas procesorius. Didelio dažnio procesoriams reikalingos daug didesnės energijos sąnaudos. Stengiantis neapkrauti procesoriaus papildomais skaičiavimais, išmaniuosiuose telefonuose nuo seno naudojamos žemo lygio programavimo technologijos.

Dėl palyginti žemo lygio programinių sąsajų, programinės įrangos kūrimas Symbian OS [4] yra varginanti, sunki ir kelianti daug problemų užduotis, reikalaujanti patyrimo ir profesionalumo. Deja, profesionalių ir patyrusių Symbian operacinę sistemą išmanančių programuotojų labai trūksta. Tiesa, tendencijos po truputi keičiasi. Pastaraisiais metais, gamintojai bando supaprastinti programines sąsajas. Microsoft išleido .Net Compact Framework [5], kurio programinės sąsajos kur kas aukštesnio lygio. Android operacinės sistemos programinės sąsajos realizuotos Java programavimo kalba, kas taip pat ženkliai supaprastina programinės įrangos kūrimą šiai operaciniai sistemai.

Apibendrinkime ***problemas susijusias su programinės įrangos išmaniesiems telefonams kūrimu***:

- Didelė skirtingų platformų įvairovė, vieningų standartų nebuvimas;
- Sudėtingos programinės sąsajos;
- Specialistų trūkumas;

- Trumpi programinės įrangos gamybos terminai.

Šias problemas išspręstų nuo platformos nepriklausanti metodika, turinti paprastas aukšto lygio programavimo sąsajas. Šiame darbe pateiksime metodą, kuris pasižymi aukščiau minėtomis savybėmis: nepriklausomumu nuo platformos ir aukšto lygio programavimo sąsajomis. Remiantis metodu buvo sukurtas nepriklausomas nuo platformos programinės įrangos išmaniesiems telefonams karkasas (toliau karkasas). Norint įrodyti sprendimo veikimą, sukurto Karkaso pagalba buvo realizuota programinė įranga ir išbandyta skirtingų platformų išmaniuosiuose įrenginiuose.

2.2. Egzistuojančių sprendimų analizė

Egzistuojančių analogiškos problemos sprendimų analizė leidžia suprasti kas jau padaryta, kokie metodai naudojami, kokios egzistuoja jų realizacijos, kokie šių metodų ir jais remiantis realizuotų sistemų privalumai bei trūkumai.

Literatūroje minimi trys pagrindiniai metodai nepriklausomumui nuo platformos įgyvendinti:

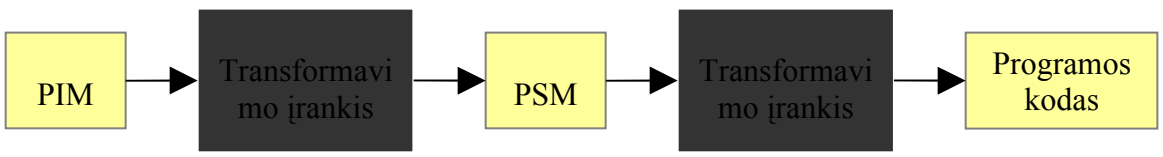
- Platformai specifinio kodo generavimas remiantis nuo platformos nepriklausomu formaliu modeliu;
- Nepriklausomo nuo platformos kodo interpretavimas;
- Platformoms bendros aukštesnio lygio programinės sąsajos realizavimas platformai specifinėmis priemonėmis.

Aptarsime kiekvieną metodą detaliau.

2.2.1. Platformai specifinio kodo generavimas remiantis nuo platformos nepriklausomu formaliu modeliu

Sričiai specifinis modeliavimas (DSM) [6] - kylanti technologija. Technologijos pagrindinė idėja – platformai būdingi bendri ir specifiniai bruožai, aprašomi sričiai specifine modeliavimo kalba (DSML), kuri sukurta remiantis konkrečia sritimi. Naudojantis DSML, kuriama programa aprašoma aukštesniame abstrakcijos lygyje, kuris yra bendras visai sričiai. Galutinis produktas automatiškai sugeneruojamas pagal aprašytą modelį. Generavimą atlieka DSM procesorius, realizuotas konkrečiai platformai.

Modeliu paremta architektūra (MDA) siūlo būdą atskirti esminę, nuo platformos nepriklausomą informaciją, nuo platformai specifinių konstrukcijų ir prielaidų. Remiantis šiuo metodu, programinės įrangos veikimas aprašomas aukšto lygio nuo platformos nepriklausomu modeliu (PIM). Tam dažniausiai naudojami tokie formalūs metodai kaip unifikuota modeliavimo kalba (UML). Vėliau šis modelis transformuojamas į platformai specifinį modelį (PSM). Galiausiai platformai specifinis modelis transformuojamas į programos kodą [7]. Modelių transformavimo procesas pateiktas 3-ame paveikslėlyje.



3 pav. MDA principas.

Norint pritaikyti MDA konkrečiai sričiai, nuo platformos nepriklausomi modeliai aprašomi panaudojant DSML. Vėliau šie modeliai DSM procesoriaus pagalba transformuojami į programos kodą, skirtą konkrečiai platformai. Šio metodo pliusai bei minusai pateikti 2-oje lentelėje.

2 lentelė. MDA technologijos pliusai bei minusai.

Pliusai
► Išvengiama kodavimo etapo – iš aukštos abstrakcijos modelių kodas sugeneruojamas automatiškai.
Minusai
► "Idealistiška" iniciatyva**.** MDA sumanytas kaip tiesioginės inžinerijos metodas, kuris atlieka DSML transformaciją į konkrečią realizaciją. Tai sutampa su OMG vizija, kad MDA turi leisti sumodeliuoti visą srities sudėtingumą pasitelkus UML (ar kitą modeliavimo kalbą) su vėlesniu transformavimu į galinį produktą. Tai sąlygoja, kad galinis produktas ateityje jau nebegali būti keičiamas. Tačiau realiame pasaulyje keičiasi tiek aplinka, tiek reikalavimai, todėl programinė įranga taip pat turi kisti.
► Neišbaigti standartai – MDA metodologija pagrįsta įvairiais techniniais standartais, kurie dar tik specifikuojami (pvz. xtUML) arba dar nepateikti plačiam naudojimui (QVT transformavimo variklis).
► Norint efektyviai panaudoti kodo generavimą, būtina griežtai apibrėžti dalykinę sritį. Todėl šiuo metodu realizuotas įrankis efektyviai gali būti pritaikytas tik labai siauroje srityje.
► Specifiniai gebėjimai. Norint praktiškai panaudoti MDA, reikia turėti žinių ir patirties šioje srityje.

Tai dar jauna ir neišbaigta metodologija retai kada taikoma praktikoje. Tačiau jau yra sukurti šią metodologiją įgyvendinantys įrankiai programinei įrangai išmaniesiems telefonams kurti [8][9].

2.2.2. Nepriklausomo nuo platformos kodo interpretavimas

Metodo įgyvendinimo metu realizuojama programa – interpretatorius. Šios programos pagalba vykdomos (interpretuojamos) kitos kalbos. Interpretuojamas kodas transliuojamas iš aukšto lygio programavimo kalbos į mašininį kodą vykdymo metu. Tai kartojama kiekvieną kartą vykdant programą. Tuo tarpu kompiliuojamas kodas iš aukšto lygio programavimo kalbos transliuojamas į mašininį kodą dar prieš vykdant programą ir tai atliekama tik vieną kartą. Interpretatoriaus realizavimo metu pasirenkama ar bus kuriama nauja programavimo kalba su savita sintakse, ar naudojama jau egzistuojanti kalba, su gerai žinoma ir griežtais standartais apibrėžta sintakse. Dažniausiai pasirenkama naudoti jau egzistuojančios kalbos sintaksę. Tai pigiau, reikia mažiau resursų vartotojų apmokymui ir dažniausiai pilnai pakanka jau egzistuojančios kalbos funkcionalumo. Naują interpretuojamą kalbą kurti tikslinga tik tuo atveju, jei interpretatorius taikomas labai specifinėje dalykinėje srityje, kuriai aprašyti nepakanka egzistuojančių kalbų priemonių.

Didžiausias interpretuojamo kodo privalumas tai, kad jį galima vykdyti daugelyje platformų, tuo tarpų kompiliatoriumi sukompiliuota programa gali būti vykdoma tik tam tikruose gimininguose

kompiuteriuose ir/ar operacinėse sistemose. Interpretuojama kalba leidžia žymiai lengviau naudoti tipų konversiją. Pavyzdžiui, daugelyje interpretuojamų kalbų tas pats kintamasis programos vykdymo metu gali įgyti skirtingo tipo reikšmes.

Klaidų, ypač loginių, paieška naudojant interpretatorius lengvesnė, sutaupoma laiko. Atlikus pakeitimą iškarto galime matyti jo įtaką programai (nėra kompiliavimo fazės). Deja, dėl kompiliatoriaus nebuvimo interpretuojamos programos gali turėti su sintaksės analize susijusių klaidų, kurių sukompiliuotose programose nebūna (programa nesukompiliuojama, kol tokios klaidos nepašalinamos).

Interpretatorius galima suskirstyti į tris grupes:

- Interpretatoriai tiesiogiai vykdantys skripto kodą, t.y. interpretatoriui skaitant skriptą ir aptikus atitinkamą konstrukciją iš karto įvykdomi ją atitinkantys skaičiavimai.
- Interpretatoriai nuskaitytą kodą perverčiantys (kompiliuojantys) į patogesnę vykdymui formą (tarpinis kodas) ir iškart jį įvykdantys;
- Interpretatoriai interpretuojantys artimą mašininiam tarpinį kodą (angl. bytecode), kuris sukonstruotas iš anksto.

Pirmojo tipo interpretatoriai patys lėčiausi ir praktikoje beveik nebenaudojami. Antrojo ir trečiojo tipo interpretatoriuose skripto kompiliavimas į tarpinę formą leidžia interpretavimą pagreitinti nuo kelių iki kelių dešimčių kartų, nes visa sintaksinė kodo analizė, konstantų apskaičiavimas ir pan., gali būti įvykdyti prieš vykdant programą. Antrojo tipo interpretatorius pirmą kartą vykdant programą analizuoja jos sintaksę ir transliuoja ją į tarpinę formą, kuri vėliau interpretuojama. Kiekvieno sekančio vykdymo metu sintaksė nebeanalizuojama, tiktai interpretuojama išsaugota tarpinė kodo forma. Trečiojo tipo interpretatoriui kaip įėjimo duomenys pateikiamas iš anksto sukompiliuotas tarpinis kodas. Tarpinis kodas - tai plokščia, nuosekliai išdėstytų operacijų seka, artima mašininiam kodui.

Dinaminis kompiliavimas (angl. Just in Time Compilation arba JIT) – artima interpretavimui technologija, interpretavimo ir kompiliavimo hibridas [10]. JIT naudojamas tarpinio kodo vykdymo pagreitinimui. Tarpinio kodo vykdymo metu dinaminis kompiliatorius sukompiliuoja dalį ar visą programos kodą į mašininį kodą, tam kad būtų padidintas programos vykdymo greitis. Kompiliavimas gali būti atliktas failui, funkcijai ar bet kuriai kitai pasirinktai kodo daliai. Kompiliavimas atliekamas prieš pat kodo vykdymą.

Panaudojus JIT galima pasiekti daug geresnį programos vykdymo greitį lyginant su tradiciniu interpretatoriumi. Papildomai, kai kuriais atvejais JIT panaudojimas gali pasiūlyti aukštesnį

vykdymo greitį nei statinis programų kompiliavimas, nes daugelis kodo optimizavimų įmanomas tik vykdant kodą, pvz. kompiliavimas gali būti optimizuotas atsižvelgiant į procesorių ir operacinę sistemą, kurioje vykdoma programa (naudojant statinį kompiliavimą, mes tiksliai nežinome, kokioje aplinkoje bus vykdoma programa).

JIT panaudojimas sąlygoja nežymų uždelsimą programos paleidimo metu. Laiko nuostoliai atsiranda dėl laiko reikalingo nuskaityti tarpini kodą ir jį sukompiliuoti. Kuo daugiau optimizavimo aplieka JIT, tuo kokybiškesnis mašininis kodas sugeneruojamas, tačiau taip pat padidėja ir laikas, reikalingas programos paleidimui. Dėl šios priežasties turi būti rastas kompromisas tarp kompiliavimo trukmės ir sugeneruoto kodo kokybės.

Šio nepriklausomumą nuo platformos užtikrinančio metodo pliusai bei minusai pateikti 3-oje lentelėje.

3 lentelė. Kalbos interpretavimo pliusai bei minusai.

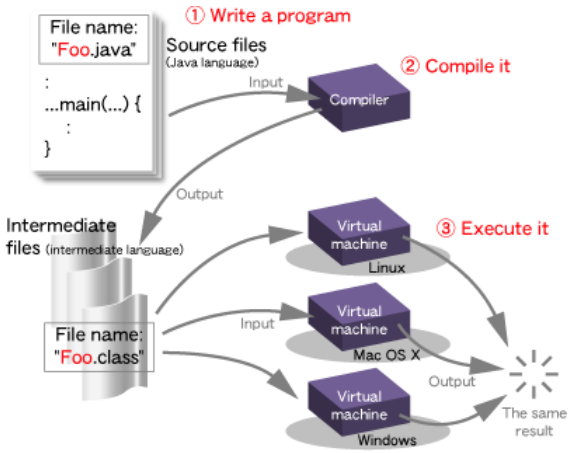
Pliusai
► Didesnis saugumas. Interpretatorius gali apsaugoti nuo daugelio programuotojo paliktų, neigiamai veikiančių vykdymo aplinką, klaidų, tokių kaip dinaminės atminties neatlaisvinimas ir panašiai. ► Interpretuojamos kalbos paprastesnės, todėl sąlygoja lengvesnį ir greitesnį programų kūrimą, pakanka žemesnės kvalifikacijos specialistų.
Minusai
► Lėtesnis programos vykdymas. Interpretuojamos kalbos visuomet vykdomos lėčiau nei programos sukompiliuotos į mašininį kodą. ► Programos ne visada veikia teisingai, nors ir parašytos teisingai – dėl galimų interpretatoriaus defektų net ir teisingas interpretuojamos kalbos kodas gali butų klaidingai interpretuotas. ► Skirtingos interpretatoriaus versijos turi būti realizuotas visoms palaikomoms platformoms.

2.2.2.1. Java platforma

Geriausiai žinoma ir plačiausiai naudojama šiuo metodu realizuota technologija – Java platforma, kurią 1995 metais sukūrė Sun Microsystems.

Kuriant Java programavimo technologiją buvo keliami šie pagrindiniai tikslai:

- paprasta, objektiškai orientuota kalba ir pažįstama sintaksė;
- griežtai apibrėžta ir saugi technologija;
- portatyvi nepriklausoma nuo platformos technologija;
- greitas kodo vykdymas;
- interpretuojama ir dinaminė kalba;



4 pav. Java technologija.

Sun Microsystems kurdama Java programavimo kalbą, sintaksę perėmė iš C/C++ kalbų, bet supaprastino objektinį modelį bei sumažino žemo lygio priemones, tokias kaip atminties valdymas ir pan. Tokiu būdu, kalba tapo paprastesnė, ją išmokti ir pradėti naudoti reikia mažiau laiko resursų. Kuriant programas šia kalba įveliama mažiau klaidų.

Java Technologijos principas pateiktas 4-ame paveikslėlyje. Java kalba parašyta programa kompiliuojama į tarpinį kodą (angl. Java bytecode), kuris interpretuojamas virtualioje mašinoje (JVM). JVM realizuojama kiekvienai platformai atskirai.

Specifinių platformai funkcijų, tokių kaip grafikos atvaizdavimas, gijų valdymas, prieiga prie tinklo, panaudojimui realizuotas bendras būdas, standartizavus tam skirtas bibliotekas.

Beveik visose Java virtualiosiose mašinos versijose realizuoti dinaminiai kompiliatoriai (JIT). Sun‘s HotSpot Java virtualioje mašinoje programos tarpinis kodas pirmiausia interpretuojamas, o JVM realizuotos priemonės stebi, kurios tarpinio kodo sekos vykdomos dažniausiai. Šios dalys naudojant JIT sukompiliuojamos į mašininį kodą, kuris vykdomas tiesiogiai aparatūroje. Toms tarpinio kodo dalims, kurios vykdomos vos kelis kartus, kompiliavimas netaikomas, todėl sutaupoma laiko, skirto kompiliavimui atlikti. Dažnai vykdomam tarpinio kodo kompiliavimui sugaištas laikas atsiperka dėl greitesnio mašininio kodo vykdymo nei tarpinio kodo interpretavimo [11].

Kita Java virtualios mašinos versija turi realizuotą JIT, kuris veikia dviem režimais, taikančiais skirtingas kodo optimizavimo strategijas, – kliento ir serverio. Kliento režime atliekamas tik minimalios tarpinio kodo dalies kompiliavimas ir optimizavimas tam, kad būtų sumažintas programos paleidimo laikas. Serverio režime didelė tarpinio kodo dalis kompiliuojama į mašininį kodą ir optimizuojama tam, kad kiek įmanoma būtų padidintas programos vykdymo greitis.

Antroji Java technologijos versija buvo išleista kelioms skirtingų tipų platformoms:

- J2SE (Java Standard Edition) – skirta naudojimui asmeniniuose kompiuteriuose;
- J2EE (Java Enterprice Edition) – skirta verslo programinei įrangai (serveriams ir panašiai);
- J2ME (Java Micro Edition) – skirta mobiliesiems telefonams.

Plačiau apžvelgsime J2ME. J2ME buvo sukurta spręsti problemas, atsirandančias programinę įrangą kuriant mažiems įrenginiams (pvz. mobiliems telefonams). Tam tikslui Sun panaudojo standartinės Java klasių bibliotekų poaibį, kuris tiktų tokiai ribotai aplinkai ir jo pagalba būtų galima kurti programinę įrangą mažiems įrenginiams su ribota atmintimi, ekranu ir galia.

J2ME technologija paremta trimis elementais:

- konfigūracija – apibrėžia pagrindinių bibliotekų rinkinį ir virtualios mašinos galimybes plačiai įrenginių grupei;
- profilis – apibrėžia programinių sąsajų rinkinį, kurį palaiko siauresnė įrenginių grupė;
- papildomi paketai – specifinės programinės sąsajos.

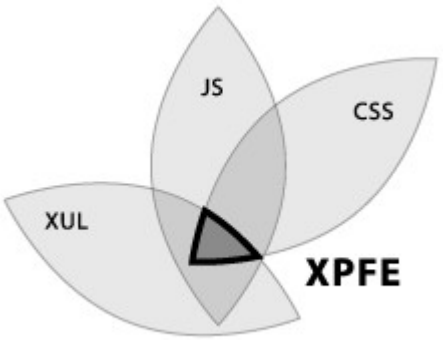
Šiuo metu J2ME platforma yra padalinta į dvi pagrindines konfigūracijas: viena skirta mažiems mobiliems įrenginiams (CLDC), antroji skirta daugiau galimybių turintiems mobiliems įrenginiams, tokiems kaip išmanieji telefonai (CDC) [13].

Sun Microsystems pateikia tik aukšto lygio programinių sąsajų, skirtų išmaniųjų telefonų aparatūros resursams pasiekti, aprašus, bet jų nerealizuoja. Šis darbas paliekamas trečios šalies kompanijoms. Dėl to dažniausiai J2ME telefonai turi tik standartines programines sąsajas, o sąsajos išmaniajame telefone įdiegtai aparatūrai pasiekti viešai nepaskelbiamos. Tai labai apriboja J2ME technologijos panaudojimą, ir remiantis šia technologija dažniausiai kuriami tik kompiuteriniai žaidimai, skirti mobiliesiems telefonams.

2.2.2.2. Mozilla programų karkasas (Mozilla Application Framework)

Dar vienas įrankis realizuotas remiantis šiuo metodu ir vertas paminėti - Mozilla Application Framework. Mozilla projektas buvo pradėtas 1998 metais, kurio tikslas buvo sukurti naršyklę, pakeisiančia Netscape‘s Communicator. Šią naršyklę buvo planuojama išleisti trims platformoms: Mac, Windows ir Unix operacinėms sistemoms. Kiekvienai platformai skirtos naršyklės versijos kūrimui ir palaikymui buvo suformuotos atskiros programuotojų komandos. Projekto vadybininkai pastebėjo, kad kurti kelioms skirtingoms platformoms skirtą programinę įrangą ir naudoti platformai specifinę technologiją labai brangu ir ilgai trunka. Todėl buvo nuspręstą sukurti nuo platformos nepriklausomą įrankį, kodiniu pavadinimu XPFE (Mozilla‘s Cross-Platform Front End), kurio pagalba būtų galima sukurti bendrą, visose platformose veikiančią programinę įrangą.

XPFE naudoja kelis jau egzistuojančius interneto technologijų standartus, tokius kaip CSS (Cascading Style Sheets), JavaScript ir XML (naudojantis XML standartu buvo sukurta nauja kalba, pavadinta XUL – XML based User-interface Language). Kitaip tariant, Mozilla kūrėjai panaudojo pasiteisinusias ir laiko patikrintas DHTML technologijas, skirtas interneto naršyklėje vykdomoms programoms aprašyti, ir adaptavo jas greitesnių, lankstesnių ir vykdomų už interneto naršyklės ribų programų kūrimui. XPFE galima apibūdinti kaip internetinių technologijų uniją. 5-tas paveikslėlis iliustruoja kaip kartu sudėtos trys minėtos internetinės technologijos suformuoja XPEE karkasą [14].



5 pav. Internetinės technologijos XPEE karkase.

XPFE naudojamos internetinės technologijos skirtos:

- JavaScript - aprašyti programos funkcionalumą;
- CSS - aprašyti programos grafinę išvaizdą (angl. Look and Feel);
- XUL – aprašyti programos vartotojo sąsajos struktūrą.

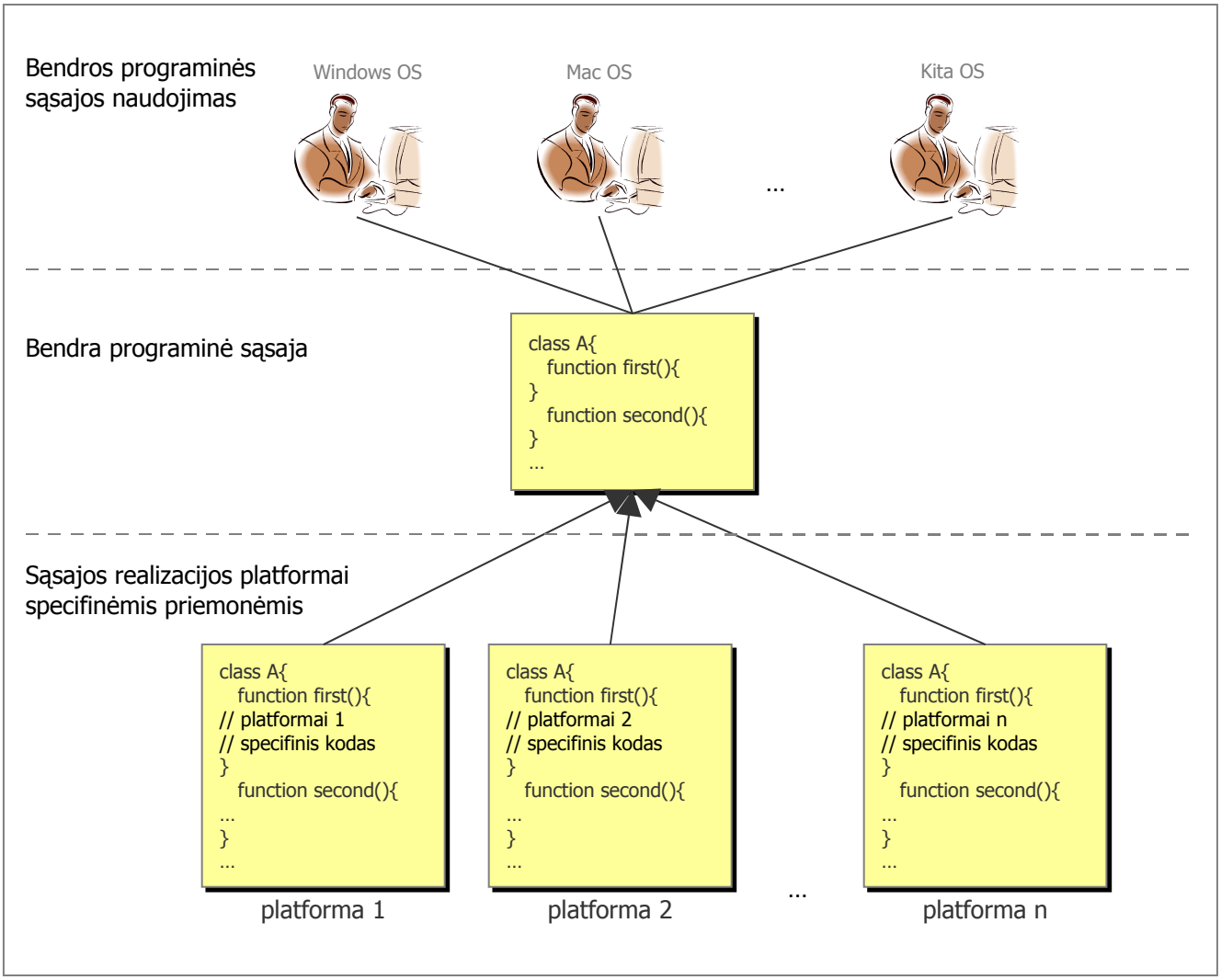
Vietoj platformai specifinių programavimo kalbų tokių kaip C ar C++ naudojimo kuriant programinę įrangą, XPFE naudoja gerai žinomus interneto technologijų standartus. Dėl šios priežasties, žmonės, išmanantys tinklalapių kūrimą, gali labai greitai išmokti ir panaudoti XPFE nuo platformos nepriklausomai programinei įrangai kurti.

XPFE taip pat naudoja XPCOM/XPConnect technologijas, leidžiančias JavaScript pagalba pasiekti ir panaudoti C ir C++ bibliotekas. Tokiu būdu galima nesunkiai praplėsti karkaso funkcionalumą pagal savo poreikius.

2.2.3. Platformoms bendros aukštesnio lygio programinės sąsajos realizavimas platformai specifinėmis priemonėmis

Tai vienas populiariausių ir dažniausiai naudojamų nuo platformos nepriklausomų įrankių realizavimo metodų. Šiandienos rinkoje egzistuoja daugybė jau sukurtų ir sėkmingai naudojamų įrankių, užtikrinančių nepriklausomumą nuo platformos populiariausiose operacinėse sistemose,

tokiose kaip Windows, Linux, Moc X OS ir t.t. Šis metodas dažniausiai realizuojamas žemesnio lygio kalbomis tokiomis kaip C++, tam, kad būtų išlaikytas greitas programos vykdymas. Metodo realizacijos principas pateiktas 6-ame paveikslėlyje.



6 pav. Bendros aukštesnio lygio programinės sąsajos panaudojimas.

Šio metodo pliusai bei minusai pateikti 4-oje lentelėje.

4 lentelė. Bendros programinės sąsajos realizavimo pliusai bei minusai.

Pliusai
► Greitas programų vykdymas. Kodas galiausiai sukompiliuojamas į mašininį kodą, todėl programos vykdymo greitis beveik neatsilieka nuo platformai specifinėmis priemonėmis realizuotų programų. ► Paprastas naudojimas. Aukštesnio lygio sąsajos suprojektuojamos taip, kad jas būtų paprasta ir patogu naudoti.

Minusai
► Ribojanti programavimo kalba. Bendra sąsaja gali būti realizuota tik tose platformose, kurios palaiko tą pačią programavimo kalbą. ► Daug pasikartojančio kodo. Kiekviena platforma turi savo aukštesnio lygio sąsajos realizaciją. Norint įtraukti naują aukštesnio lygio sąsajos funkcionalumą, tenka realizuoti jį kiekvienoje palaikomoje platformoje.

2.2.3.1. Qt nuo platformos nepriklausomų programų karkasas (Qt Cross-Platform Application & UI Framework)

Qt Cross-Platform Application & UI Framework - tai turbūt vienas sėkmingiausių nuo platformos nepriklausomai programinei įrangai kurti skirtas įrankis. Naudojantis šiuo įrankiu, sukurtos tokios visiems žinomos taikomosios programos kaip: Google Earth, Opera, Skype, VLC Media Player ir daugelis kitų. Pirmoji Qt karkaso versija buvo išleista 1994 metais TrollTech kompanijos. Ši versija buvo skirta tik dvejoms platformoms – Unix ir Windows. 2003 metais TrollTech išleido Qt karkaso versiją ir Mac OS X platformai. 2008 metais TrollTech buvo parduota Nokia korporacijai. Nokia planuoja Qt karkasą naudoti kaip pagrindinę programų kūrimo priemonę savo gaminamuose išmaniuosiuose telefonuose [15]. Maeno (Linux) platformoje Nokia jau dabar naudoja Qt karkasą. Greitu laiku planuojama S60 platformoje naudojamas Avkon klasių bibliotekas pakeisti Qt karkaso bibliotekomis.

Šiuo metu oficialios Qt karkaso versijos yra išleistos šioms platformoms:

- Linux/X11;
- Mac OS X;
- Windows;
- Embedded Linux;
- Windows CE;
- Symbian OS;
- Maemo.

Nokia korporacijai nupirkus TrollTech, Qt karkasas buvo paskelbtas atviro kodo projektu. Todėl atsirado ir neoficialių kitoms platformoms skirtų Qt karkaso versijų:

- Qt for OpenSolaris;
- Qt for Haiku OS;
- Qt for OS/2;
- Qt-iPhone;

- Android-Lighthouse;
- Qt for webOS;

Pirmosiose Qt versijose buvo bandoma simuliuoti platformai būdingą grafinės vartotojo sąsajos elementų išvaizdą. Tačiau tai nepasitvirtino ir vėliau buvo pereita prie kiekvienai platformai savitų programinių sąsajų, grafinei vartotojo sąsajai atvaizduoti, naudojimo [16].

Qt karkase taip pat realizuotas meta objektų kompiliatorius (MOC). Šis įrankis interpretuoja Qt išeities kode naudojamas makro komandas ir sugeneruoja papildomą C++ kodą. Tokių būdu praplečiamos standartinės C++ kalbos galimybės.

3. NUO PLATFORMOS NEPRIKLAUSOMOS PROGRAMINĖS ĮRANGOS IŠMNIESIEMS TELEFONAMS KŪRIMO METODAS

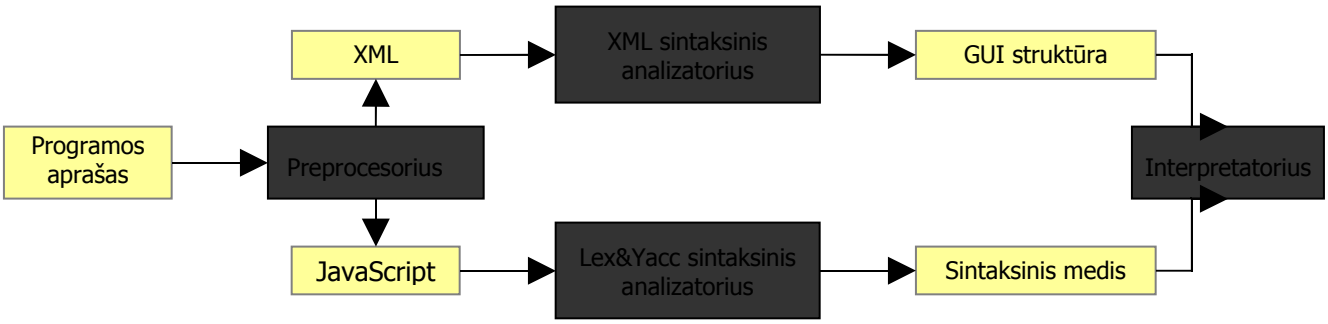
Nepriklausomumo nuo platformos problemai spręsti buvo pasirinkta aukštesnio lygio nuo platformos nepriklausanti kalba ir interpretatorius. Tokį pasirinkimą lėmę faktoriai:

- *Lankstumas.* Lyginant su išeities kodo generavimu, kodo interpretavimas lankstesnis, platesnės pritaikymo galimybės. Kodo generatorius gali būti efektyviai panaudotas tik siauroje ir griežtai apibrėžtoje dalykinėje srityje, tokioje kaip programos grafinę sąsają (tai realizuota daugelyje šiuolaikinėse programinės įrangos kūrimo aplinkose, pvz. Microsoft Visual Studio). Tačiau programos funkcionalumo generavimas daug sudėtingesnis. Egzistuoja daugybė skirtingų galimų programos funkcionalumo realizacijos variantų ir sunku juos visus įvertinti. Dažniausiai, atlikus generavimą, vis tiek tenka rankinių būdu modifikuoti sugeneruotą kodą.
- *Interpretuojamos kalbos paprastumas.* Interpretuojama kalba gali būti daug paprastesnė lyginant su žemo lygio kalbomis, dažniausiai naudojamomis išmaniųjų telefonų programinei įrangai kurti, pavyzdžiui C/C++. Dėl paprastumo ji greičiau įsisavinama ir išmokstama, todėl nereikia aukštos kvalifikacijos programuotoju.
- *Galimybė sumažinti kritinių klaidų kiekį.* Naudojant šį sprendimą galima eliminuoti kai kurių defektų atsiradimo tikimybę. Atminties valdymo klaidos (gana dažnos programuojant C/C++ kalbomis) yra kritinės išmaniesiems telefonams skirtoje programinėje įrangoje, dėl labai ribotų atminties resursų. Atminties valdymo užduotį perleidus interpretatoriui, tokių klaidų tikimybė sumažinama iki minimumo.

Interpretuojamoms programoms aprašyti nuspręsta naudoti jau egzistuojančius ir populiarius standartus, naudojamus interneto tinklalapių kūrimui: JavaScript ir XML. Dėl didelio šių dienų interneto populiarumo, internetines technologijas išmanančių specialistų labai daug, todėl šių technologijų panaudojimas išmaniųjų telefonų programinei įrangai kurti panaikina specialistų trūkumo problemą.

Realizuojamas įrankis analizuoja XML ir JavaScript standartais aprašytą programą. Pirmasis komponentas, preprocesorius, atskiria XML notacijas nuo JavaScript kodo, bei nukreipia į atitinkamą gramatinį analizatorių. XML gramatinis analizatorius realizuotas taip, kad atpažintų iš anksto apibrėžtus vartotojo sąsajos elementus. Aptikęs kurį nors iš aibėje apibrėžtų elementų, sukuria atitinkamą vartotojo sąsajos komponentą. JavaScript gramatinis analizatorius analizuoja

kodą ir sukonstruoja sintaksinį medį, kurį vėliau vykdo ir atitinkamus veiksmus atlieka realizuotas interpretatorius. Visas šis procesas pavaizduotas 7-ame paveikslėlyje.



7 pav. Skripto apdorojimas realizuoto karkaso priemonėmis.

3.1. Grafinė vartotojo sąsajos aprašymas

Dalis grafinės vartotojo sąsajos struktūros viso programos vykdymo metu išlieka statinė. Kadangi siekiama kuo labiau supaprastinti programų išmaniesiems telefonams kūrimą, nuspręsta atskirti priemones statiniai ir dinaminiai programos daliai aprašyti [17]. Programos vartotojo sąsajos struktūrai (statinė programos dalis) aprašyti pasirinkta XML notacija. Pasirinkimo argumentai:

- Technologija apibrėžta griežtais standartais ir plačiai naudojamos;
- XML formatas lengvai skaitomas tiek žmogui, tiek mašinai. Net su formatu nesusipažinęs žmogus gali nesunkiai jį perprasti, be to realizuoti efektyvūs XML skaitymo algoritmai.
- XML notacija labai lengva adaptuosi savo poreikiams aprašant savitus elementus su specifiniais atributais.

XML formato gramatinis analizatorius realizuotas trečios šalies atviro kodo projekto „The Expat XML Parser“ pagrindu. Šiame gramatiniame analizatoriuje realizuota SAX (Simple API for XML) programinė sąsaja. Tai įvykiais paremta sąsaja. Vartotojas aprašo funkcijas, kurias sąsajos realizacija iškviečia įvykus atitinkamam įvykiui XML notacijos analizės metu. Programinės sąsajos raportuojami įvykiai:

- XML tekstinis elementas;
- XML elemento pradžia;
- XML elemento pabaiga;
- XML apdorojimo instrukcijos;
- XML komentaras.

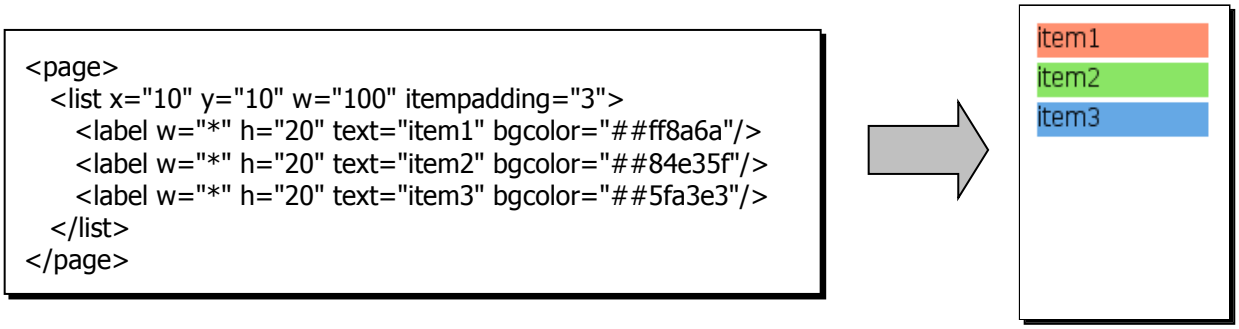
Ši sąsaja labai patogi mūsų sprendimui realizuoti. Įvykus įvykiui „XML elemento pradžia“, sukuriamas naujas, XML elementą atitinkantis, vartotojo sąsajos komponentas ir priskiriami jam

XML elemento atributuose nurodyti parametrai. Įvykus įvykiui „XML elemento pabaiga“, sukurtas komponentas įtraukiamas į bendrą vartotojo sąsajos struktūrą. Šios sąsajos privalumas, lyginant su DOM sąsaja, – efektyvesnis resursų panaudojimas, kas labai aktualu kuriant programinę įrangą išmaniesiems telefonams [18]:

- Dėl įvykiais paremtos realizacijos, SAX sąsaja dažniausiai veikia greičiau, nei DOM;
- Nereikia viso struktūrinio medžio laikyti atmintyje (naudojant DOM sąsają, prieš atliekant bet kokią analizę, visa struktūra jau turi būti nuskaityta) [19];

Kiekvieno programos lango struktūra aprašoma atskirame dokumente. Vartotojo sąsajos struktūros aprašymas pradedamas šakniniu elementu – „Page“. Šis elementas atstoja programos langą. Šiame elemente talpinami visi kiti grafinės vartotojo sąsajos komponentus aprašantys elementai. Vartotojo grafinės sąsajos aprašymui naudojama bazinė pagrindinių komponentų aibė: komponentas teksto atvaizdavimui, komponentas paveikslėlio apvaizdavimui, komponentas fiksuojantis paspaudimus, komponentas kitų komponentų grupavimui. Sudėtingesni komponentai realizuojami apjungiant šiuos bazinius komponentus.

XML elementų atributais nurodomi grafinės vartotojo sąsajos komponentų parametrai, tokie kaip: pozicija, dimensijos, spalva ir panašiai. Grafinės vartotojo sąsajos aprašymo pavyzdys pateiktas 8-ame paveikslėlyje.



8 pav. Grafinės vartotojo sąsajos aprašymas XML formatu.

3.2. Programos logikos aprašymas

JavaScript gramatinis analizatorius realizuojamas Yacc/Lex generatoriaus pagalba. Šios sistemos užduotis yra palengvinti kompiliatorių, interpretatorių ir transliatorių kūrimą, automatizuojant programų leksikinę ir sintaksinę analizę. *Lex&Yacc* sistema yra sudaryta iš dviejų atskirų, bet bendradarbiaujančių dalių. Kiekvieną iš jų aptarsime detaliau.

Lex tai leksikinių analizatorių generatorius. Iš vartotojo pateiktos specifikacijos Lex pagalba yra sugeneruojama programa-analizatorius *yylex* (9-as paveikslėlis). Specifikacijoje aprašomos

reguliarios išraiškos ir jas atitinkantys programos fragmentai, parašyti bet kuria bendros paskirties kalba (pvz., C++) [20].

9 pav. Lex pagalba generuojamas leksikinis analizatorius.

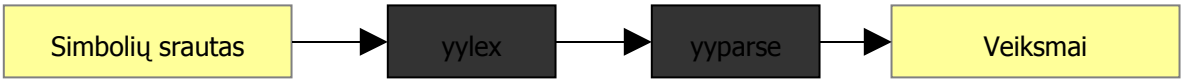
Sugeneruota programa-analizatorius *yylex* analizuoja įėjimo simbolių srautą, išskaido jį į eilutes (žymes), atitinkančias specifikacijoje nurodytas reguliarias išraiškas, bei atlieka specifikacijoje nurodytus, išraišką atitinkančius, veiksmus.

Yacc tai sintaksinių analizatorių generatorius. Iš vartotojo pateiktų gramatinių taisyklių Yacc pagalba sugeneruojama programa yyparse (10-as paveikslėlis).

10 pav. Yacc pagalba generuojamas sintaksinis analizatorius.

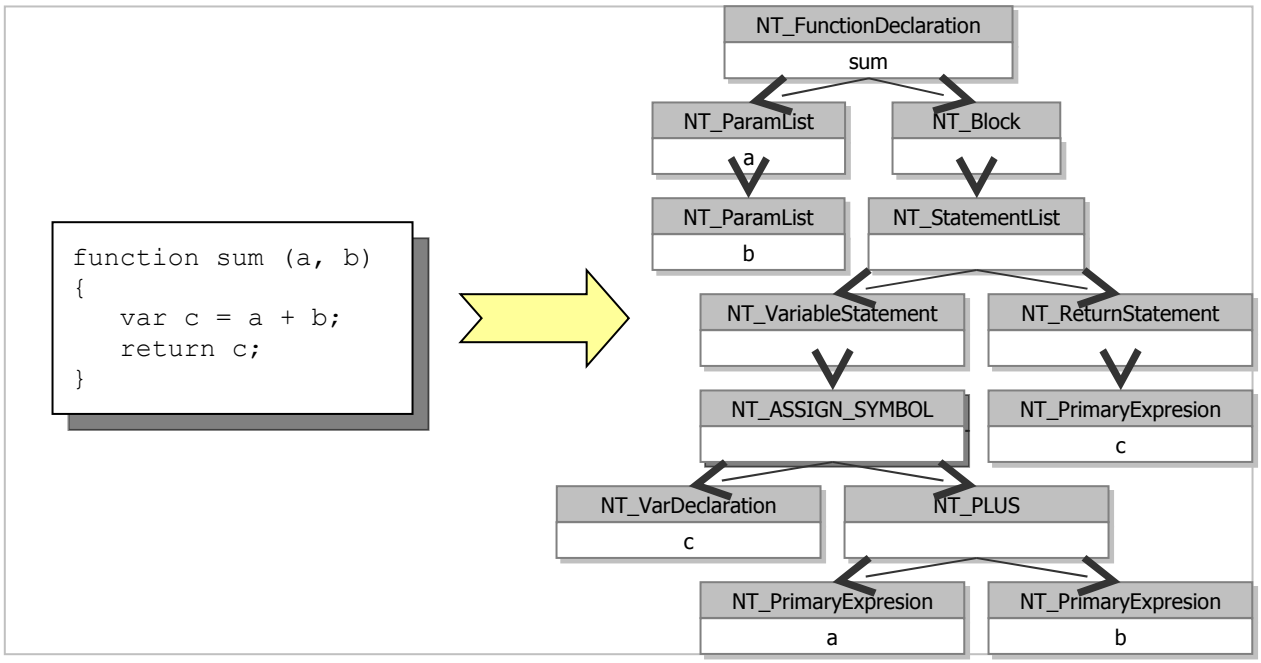
Yacc įėjimo duomenys yra seka gramatinių taisyklių ir jas atitinkantys programos fragmentai, parašyti bet kuria bendros paskirties kalba (pvz., C++). Sugeneruota yyparse programa (sintaksinis analizatorius), skaito įėjimo žymių srautą, ir pritaiko atitinkamą gramatinę taisyklę. Kuomet gramatinėmis taisyklėmis aprašyta išraiška yra atpažįstama, įvykdomas tą išraišką atitinkantis programos fragmentas [20].

Savo karkaso realizacijoje mes naudojame abiem generatoriais sugeneruotų programų yylex ir yyparse kombinaciją. Programa yylex naudojama tik įėjimo simbolių srauto suskaidymui į reguliariosiomis išraiškomis aprašytas žymes. Tuomet yyparse analizuoja gautų žymių seką ir atlieka JavaScript gramatinėmis taisyklėmis aprašytus veiksmus. Bendras abiejų programų panaudojimas pavaizduotas 11-ame paveikslėlyje.



11 pav. yylex ir yyparse programų kombinacijos panaudojimas.

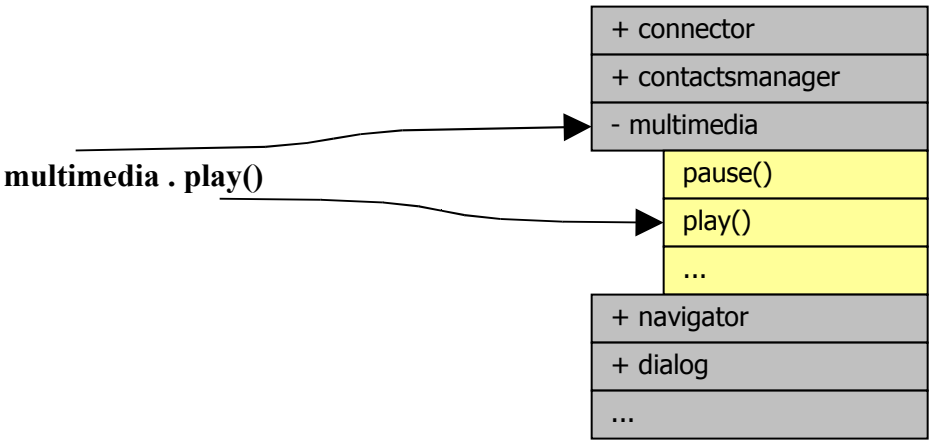
Generatorių pagalba sugeneruota programa naudojama sintaksinio medžio konstravimui. Medžio viršūnės duomenų struktūra turi du laukus: viršūnės tipą (privalomas laukas) ir duomenis (neprivalomas laukas). 12-ame paveikslėlyje pateikta JavaScript kodu aprašyta funkcija ir ją atitinkančio gramatinio analizatoriaus pagalba sukonstruoto medžio pavyzdys.



12 pav. Gramatinio analizatoriaus sukonstruotas medis, atitinkantis JavaScript kodu aprašytą funkciją.

3.3. Išmaniojo telefono resursų valdymas skripto pagalba

Išmaniojo telefono įrenginių (pavyzdžiui GPS imtuvas) ir resursų (pavyzdžiui failų sistema, ar adresų knygutė) valdymui kiekvienoje platformoje realizuotos savitos sąsajos, todėl skiriasi jų naudojimas. Šių sąsajų panaudojimui, realizuojamos aukštesnio lygio, platformoms bendros sąsajos. Šios sąsajos pasiekiamos papildomomis (ne standartinėmis) JavaScript funkcijomis, kurios suskirstytos į tam tikras grupes: daugialypės terpės funkcijos, funkcijos informacijos siuntimui iš ir į Internetą, trumpųjų žinučių siuntimo funkcijos, adresų knygutės valdymo funkcijos ir t.t. Kreipinys į šias funkcijas JavaScript kode susideda iš dviejų dalių, atskirtų tašku. Pirmoji dalis nurodo komponentą (funkcijų grupę), o antroji – funkcijos pavadinimą (13-as paveikslėlis).



13 pav. Papildomų JavaScript funkcijų pavadinimų sandara.

4. NUO PLATFORMOS NEPRIKLAUSOMOS PROGRAMINĖS ĮRANGOS IŠMANIESIEMS TELEFONAMS KARKASO PROJEKTAVIMAS REMENTIS SIŪLOMU METODU

4.1. Reikalavimai sistemai

Realizuojama sistema turi:

- galėti atlikti skaičiavimus aprašytus JavaScript kalba;
- galėti sukurti vartotojo sąsajos struktūrą pagal pateiktą aprašą XML notacija;
- turėti programines sąsajas pagrindinių išmaniojo telefono funkcijų valdymui.

Toliau smulkiau detalizuosime sistemai keliamus funkcinius reikalavimus.

4.1.1. Reikalavimai programos funkcionalumo realizavimo galimybėms naudojantis karkasu

JavaScipt skriptų rašymo kalba vartotojas turi galėti aprašyti kuriamos programos atliekamus skaičiavimus. Aprašymui turi būti naudojama standartinė JacaScript sintaksė, dalis standartinių funkcijų, bei rinkinys naujai apibrėžtų funkcijų. Skaičiavimų realizavimui turi būti galima naudoti šį JavaScript standarto struktūrų poaibį:

1. veiksmai su sveikais skaitmenimis:
    a. Aritmetinės operacijos (suma, atimtis, dalyba, daugyba);
    b. palyginimo operacijos (lygu, nelygu, daugiau arba lygu, mažiau arba lygu);
    c. loginės operacijos (ir, arba, ne);
2. veiksmai su eilutės tipo kintamaisiais:
    a. sumos operacija;
    b. palyginimo operacijos (lygu, nelygu, daugiau arba lygu, mažiau arba lygu);
    c. loginės operacijos (ir, arba, ne);
3. veiksmai su masyvais;
4. veiksmai su sąrašais;
5. sąlygos sakiniai:
    a. if () konstrukcija;
    b. if () else konstrukcija;
6. ciklai:

a. for ciklas;
b. while ciklas;

7. funkcijų deklaravimas.

4.1.2. Reikalavimai vartotojo sąsajos aprašymo galimybėms naudojantis karkasu

Vartotojo sąsaja turi būti sukuriama pagal XML formatu aprašytas struktūras. Turi būti palaikomi visi pagrindiniai vartotojo sąsajos komponentai:

- Button – komponentas, galintis atvaizduoti tekstą ir paveikslėlį, bei reaguojantis į paspaudimus. Užfiksavus paspaudimą, įvykdomas aprašytas veiksmas.
- Label – komponentas atvaizduojantis tekstą;
- Image – komponentas atvaizduojantis paveikslėlį;
- Container – komponentas galintis savyje laikyti kitus komponentus;
- List – komponentas galintis savyje laikyti kitus komponentus ir juos išrikiuojantis nurodyta tvarka (horizontaliai arba vertikaliai);
- Input – komponentas informacijos įvedimui. Realizuojami skirtingi komponento tipai: komponentas slepiantis įvestą informaciją (slaptažodžio įvedimui), komponentas skaičių įvedimui, komponentas teksto įvedimui.
- SoftKey – nematomas komponentas, fiksuojantis klaviatūros klavišų paspaudimus. Užfiksavus aprašyto klavišo paspaudimą, įvykdomas nurodytas veiksmas. Veiksmas aprašomas JavaScript sintakse.
- Page – šakninis vartotojo sąsajos komponentas, atstojantis langą.
- Menu – komponentas atvaizduojantis meniu. Komponentas savyje laiko kitus komponentus – meniu punktus. Iškviečiamas paspaudus vieną iš telefono funkcinių klavišų (kairįjį arba dešinįjį).
- MenuItem – komponentas atstojantis meniu punktą. Komponentas atvaizduoja tekstą ir reaguoja į paspaudimus.

Komponentų atributai, tokie, kaip spalva, pozicija, dimensijos ir panašiai, nurodomi XML elemento atributais.

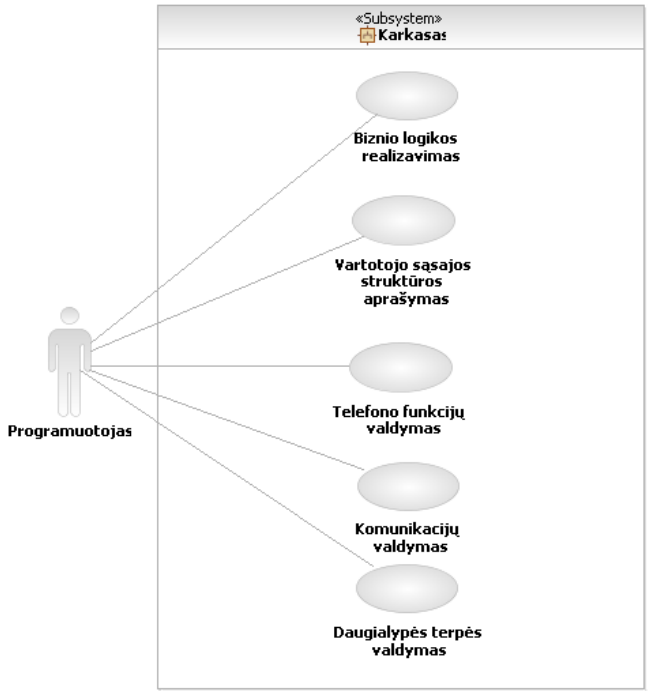
4.1.3. Reikalavimai išmaniojo telefono funkcijų programinėms sąsajoms

Turi būti realizuotos programinės sąsajos visoms pagrindinėms išmaniojo telefono funkcijoms pasiekti. 5-oje lentelėje pateikiamas sąsajų sąrašas bei reikalavimai joms.

5 lentelė. Reikalavimai programinių sąsajų funkcionalumui.

Sąsaja	Reikalavimai
Daugialypės terpės sąsaja	Realizuojamos priemonės audio bei video medžiagos atkūrimui. Turi būti realizuotos funkcijos: atkūrimui pradėti, atkūrimui sustabdyti, atkūrimui pratęsti, įrašo pozicijai gauti, įrašo pozicijai nustatyti, įrašo trukmei gauti.
Interneto sąsaja	Turi būti realizuotos priemonės informacijos parsisiuntimui bei išsiuntimui internetu nurodytu adresu. Realizuotos priemonės interneto puslapiu atidarymui bei atvaizdavimui. Galimybė naudotis tiek 3G, tiek Wi-Fi technologijomis.
SMS/MMS sąsaja	Galimybė išsiųsti trumpąsias bei daugialypės terpės žinutes, nurodytais telefono numeriais. Galimybė perskaityti žinutes, esančias „Inbox" kataloge. Galimybė filtruoti įeinančias žinutes pagal gavėją.
Adresų knygutės sąsaja	Galimybė nuskaityti duomenis, esančius adresų knygutėje. Galimybė redaguoti duomenis, esančius adresų knygutėje. Galimybė įrašyti naujus duomenis adresų knygutėje.
Įrenginio parametrų sąsaja	Galimybė sužinoti unikalų įrenginio kodą. Galimybė fiksuoti baterijos elemento pokyčius. Galimybė sužinoti telefono kalbą. Galimybė sužinoti ryšio stiprumą. Galimybė sužinoti tinklo tipą. Galimybė sužinoti ryšio operatorių. Galimybė skambinti nurodytu numeriu.
Failų sistemos sąsaja	Galimybė sukurti bylą/direktoriją. Galimybė pašalinti bylą/direktoriją. Galimybė kopijuoti bylą/direktoriją. Galimybė perkelti failą/direktoriją į naują vietą. Galimybė pervadinti bylą/direktoriją. Galimybė sužinoti visos ir laisvos atminties kiekį. Galimybė sužinoti bylos dydį. Galimybė redaguoti failų turinį.

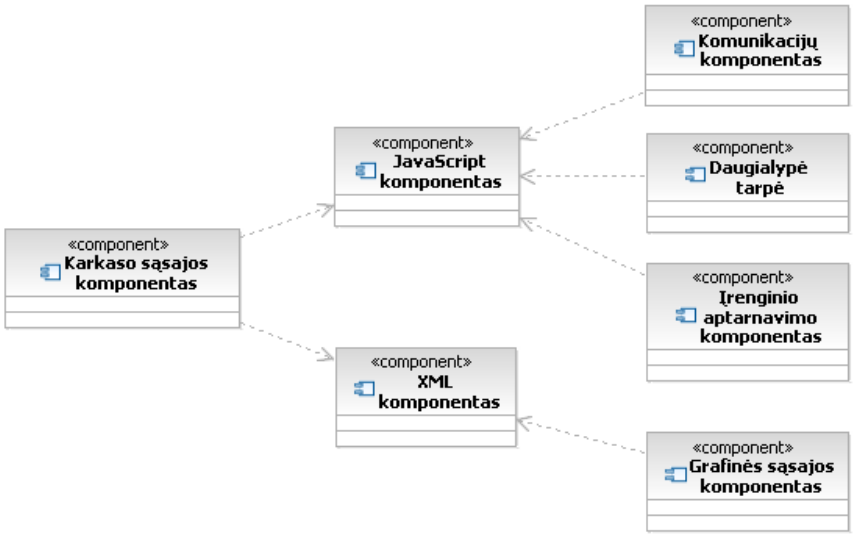
Atsižvelgdami į keliamus reikalavimus, apibrėžiame sistemos panaudos atvejus. Nedetalizuota sistemos panaudos atvejų diagrama pateikta 14 paveikslėlyje. Norėdami supaprastinti diagramą, pateikiame tik apibendrintus sistemos panaudos atvejus.



14 pav. Supaprastinta sistemos panaudos atvejų diagrama.

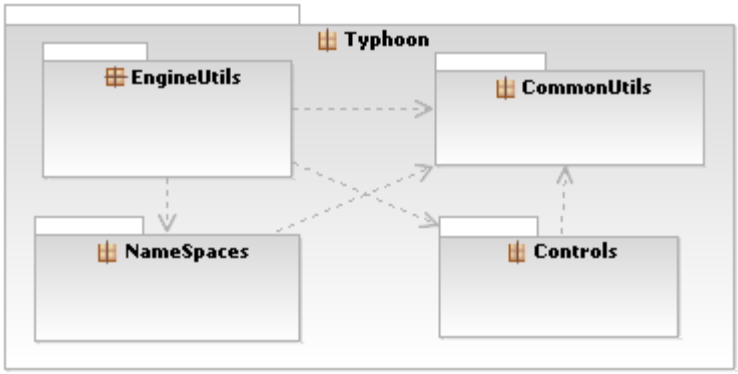
4.2. Statinis sistemos vaizdas

Sistemos funkcionalumui realizuoti sukursime 7 komponentus. Komponentų diagrama pateikta 15-ame paveikslėlyje. Karkaso sąsajos komponentas yra atsakingas už programos vykdymą, atsakomąją reakciją į vartotojo veiksmus. Komponentas taip pat valdo navigaciją tarp programos langų. JavaScript komponentas atsakingas už skripto analizavimą ir juo aprašytų skaičiavimų vykdymą. XML komponentas atsakingas už grafinės vartotojo sąsajos struktūros sukūrimą. Komunikacijų komponente realizuotos priemonės globaliam tinklui pasiekti, išsiųsti bei parsisiųsti informaciją tiek Wi-Fi, tiek 3G tinklais, siųsti SMS/MMS žinutes bei elektroninį paštą. Daugialypės terpės komponentas skirtas garso/vaizdo įrašų atkūrimui valdyti. Įrenginio aptarnavimo komponente realizuotos priemonės failų sistemos valdymui, įrenginio parametrų nuskaitymui. Grafinės sąsajos komponente pateikiamos grafinės vartotojo sąsajos komponentų realizacijos.



15 pav. Sistemos komponentų diagrama.

Šiuos komponentus suskirstyme į 4 paketus. Paketų diagrama pateikta 16-ame paveikslėlyje. Pakete EngineUtils realizuoti karkaso sąsajos, skriptų analizavimo ir vykdymo komponentai. Šis paketas turi ryšius su visais kitais sistemos paketais. CommonUtils pakete realizuotos skripte naudojamos duomenų struktūros: objektai, masyvai ir kolekcijos. NameSpaces pakete realizuotos programinės sąsajos telefono funkcijoms pasiekti. Paketas turi ryši su CommonUtils paketu. Controls pakete realizuoti grafinės vartotojo sąsajos komponentai. Paketas turi ryši su CommonUtils paketu. Toliau pateiksime detalesnę informaciją apie kiekviename pakete realizuotas klases.

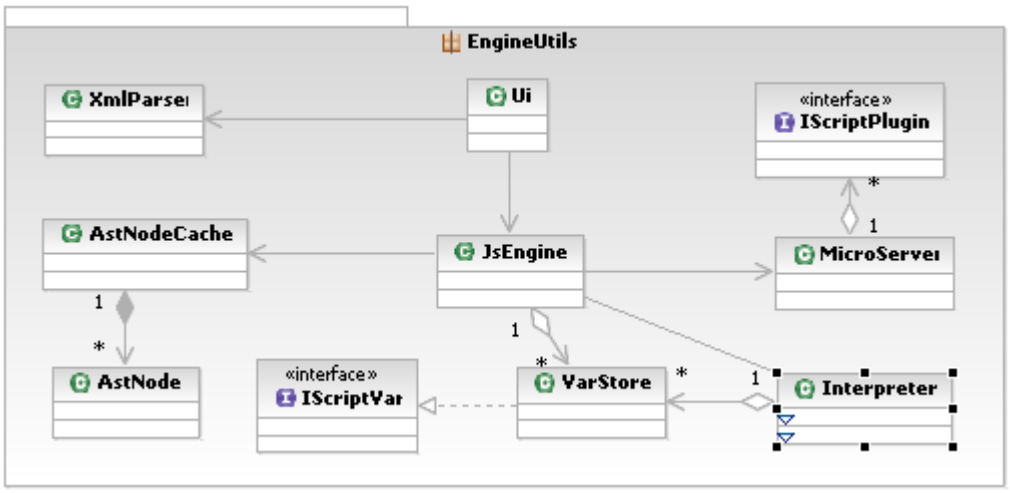


16 pav. Komponentų suskirstymas į paketus.

4.2.1. Paketas EngineUtils

Paketo klasių diagrama pateikta 17-ame paveikslėlyje. Tai pagrindinis sistemos paketas, naudojantis visus kitus sistemoje realizuotus paketus. Pakete realizuotos klasės, atsakingos už skriptų nuskaitymą, analizę bei apdorojimą, vartotojo įvesties apdorojimą. Visos (išskyrus Ui klasę)

paketo klasės yra realizuotos naudojantis tik standartinėmis C/C++ bibliotekomis, todėl yra bendros visoms palaikomoms platformoms. Toliau detaliau aprašysime pakete realizuotas klases.



17 pav. Paketo EngineUtils klasių diagrama.

Ui klasė apjungia visas kitas pakete esančias klases. Šioje klasėje pradedamos vykdyti visos karkaso funkcijos: apdorojamos vartotojo įvestys (klaviatūros ar ekrano paspaudimai), nuskaitomi pradiniai duomenys (skriptai) ir nukreipiami atitinkamiems analizatoriams. Šioje klasėje taip pat saugoma ir sukonstruota vartotojo sąsajos struktūra. Toliau detaliau aprašysime pakete realizuotas klases.

XmlParser – klasė, atliekanti XML formato gramatinę analizę bei sukonstruoja XML formatu aprašytą vartotojo sąsajos struktūrą.

JsEngine – klasė, atsakinga už JavaScript nuskaitymą, analizę ir sintaksinio medžio sukonstravimą.

AstNodeCache – klasė, reprezentuojanti sukonstruotą sintaksinį medį.

AstNod – klasė, reprezentuojanti sintaksinio medžio lapą (Node).

MicroServer – klasė, kurios pagalba iš JavaScript pasiekiamos realizuotos programinės sąsajos telefono funkcijoms vykdyti.

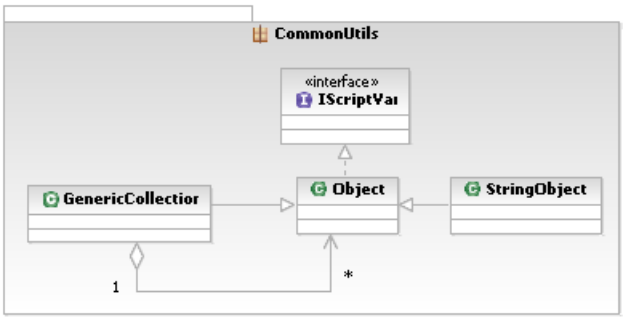
VarStore – klasė, reprezentuojanti skripto kintamąjį. Tai objektas, kuriame saugoma kintamojo informacija: tipas, reikšmė, rodyklių į kintamąjį kiekis.

Interpreter – klasė, vykdanti skriptu aprašytus skaičiavimus.

4.2.2. Paketas CommonUtils

Paketo klasių diagrama pateikta 18-ame paveikslėlyje. Pakete realizuotos sistemos duomenų struktūros. Visos pakete esančios klasės realizuotos naudojant tik standartines C/C++ bibliotekas,

todėl šios klasės bendros visoms palaikomoms platformoms. Toliau detaliau aprašysime pakete realizuotas klases.



18 pav. Paketo CommonUtils klasių diagrama.

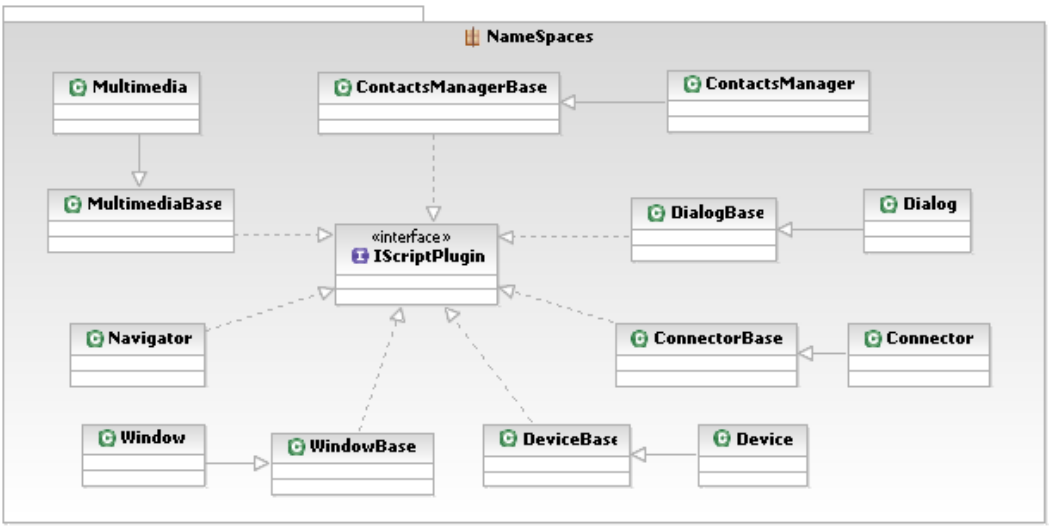
Object – klasė, sauganti informaciją apie objektą (pvz. nuskaitytą XML elementą): jo vardą, tipą, bei atributus – raktų/reikšmių poras.

GenericCollection – objektas, laikantis savyje kitus objektus.

StringObject – objektas, turintis tik vieną eilutės tipo reikšmę (XML text tipo elementas).

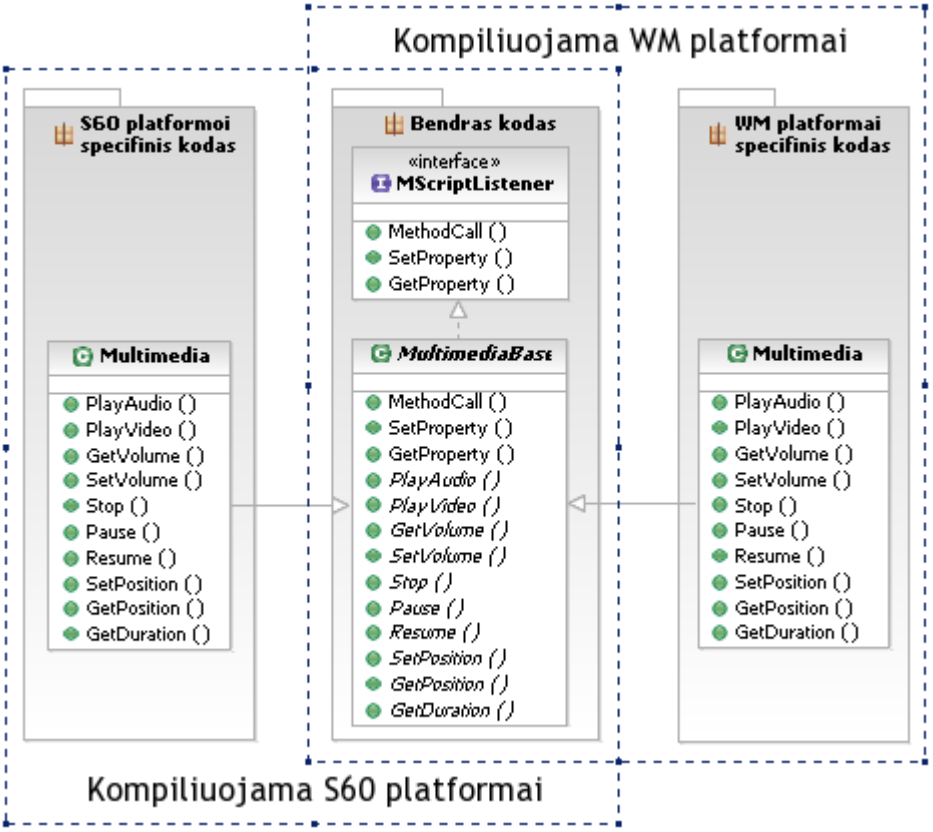
4.2.3. Paketas NameSpaces

Pakete pateikiamos programinės sąsajos išmaniojo telefono funkcijoms, tokioms kaip prisijungimas prie interneto, garso/vaizdo įrašų atkūrimas. Taip pat realizuotos funkcijos laikmačių valdymui, modalinių dialogų iškvietimui. Paketo klasių diagrama pateikta 19-ame paveikslėlyje.



19 pav. Paketo NameSpaces klasių diagrama.

Kiekvieną sąsają atitinkanti klasė padalinta į dvi dalis (pvz. *MultimediaBase* ir *Multimedia*). Pirmojoje dalyje (*MultimediaBase*) realizuojama nepriklausanti nuo platformos logika. Antrojoje (*Multimedia*) realizuojama platformai specifinė logika. Tokio realizavimo pavyzdys pateiktas 20-ame paveikslėlyje.



20 pav. Platformoms bendros sąsajos realizacija.

Toliau detaliau aprašysime pakete realizuotas klases.

Multimedia – klasė, realizuojanti programinę sąsają garso/vaizdo įrašų atkūrimui.

Navigator – klasė, pateikianti funkcijas navigacijai tarp skirtingų programos langų.

Window klasėje realizuotos funkcijos laikmačių (angl. timer, periodic timer) valdymui.

Device – klasė, realizuojanti programinę sąsają telefono nustatymams skaityti bei valdyti. Taip pat realizuotos priemonės failų sistemos valdymui.

Connector klasėje realizuotos sąsajos SMS/MMS servisų valdymui, informacijos išsiuntimui/parsiuntimui internetu, GPS imtuvo valdymui.

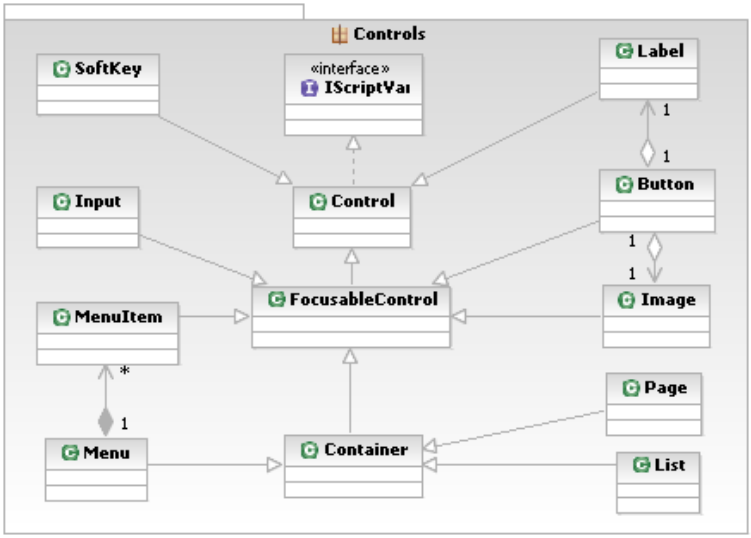
Dialog klasėje realizuotos funkcijos platformai specifinių modalinių dialogų iškvietimui.

ContactsManager klasėje realizuotos sąsajos telefono adresų knygutės valdymui.

4.2.4. Paketas Controls

Pakete realizuoti vartotojo sąsajos komponentai. Paketo klasių diagrama pateikta 21-ame paveikslėlyje. Komponentai realizuoti naudojantis standartinėmis C/C++ bibliotekomis, todėl jų realizacijos bendros visoms palaikomoms platformoms. Platformai specifinis tik komponentų

piešimas, kurį atlieka *Ui* klasė, esanti *EngineUtils* pakete. Toliau detaliau aprašysime pakete realizuotas klases.



21 pav. Paketo Controls klasių diagrama.

Button klasėje realizuotas komponentas, galintis atvaizduoti tekstą ir paveikslėlį, reaguojantis į paspaudimus. Užfiksavus paspaudimą, įvykdomas nurodytas veiksmas.

Label klasėje realizuotas komponentas, galintis atvaizduoti tekstą.

Image klasėje realizuotas komponentas, galintis atvaizduojanti paveikslėlį.

Container klasėje realizuotas komponentas, galintis savyje laikyti kitus komponentus.

List klasėje realizuotas komponentas, galintis savyje laikyti kitus komponentus ir juos išrikiuojantis nurodyta tvarka (horizontaliai arba vertikaliai).

Input klasėje realizuotas komponentas informacijos įvedimui. Realizuoti skirtingi komponento tipai: komponentas slepiantis įvestą informaciją (slaptažodžio įvedimui), komponentas skaičių įvedimui, komponentas teksto įvedimui.

SoftKey klasėje realizuotas nematomas komponentas, fiksuojantis klaviatūros klavišų paspaudimus. Užfiksavus aprašyto klavišo paspaudimą, įvykdomi nurodyti skaičiavimai. Skaičiavimai aprašomi JavaScript sintakse.

Page klasėje realizuotas šaknimis vartotojo sąsajos komponentas, atstojantis langą.

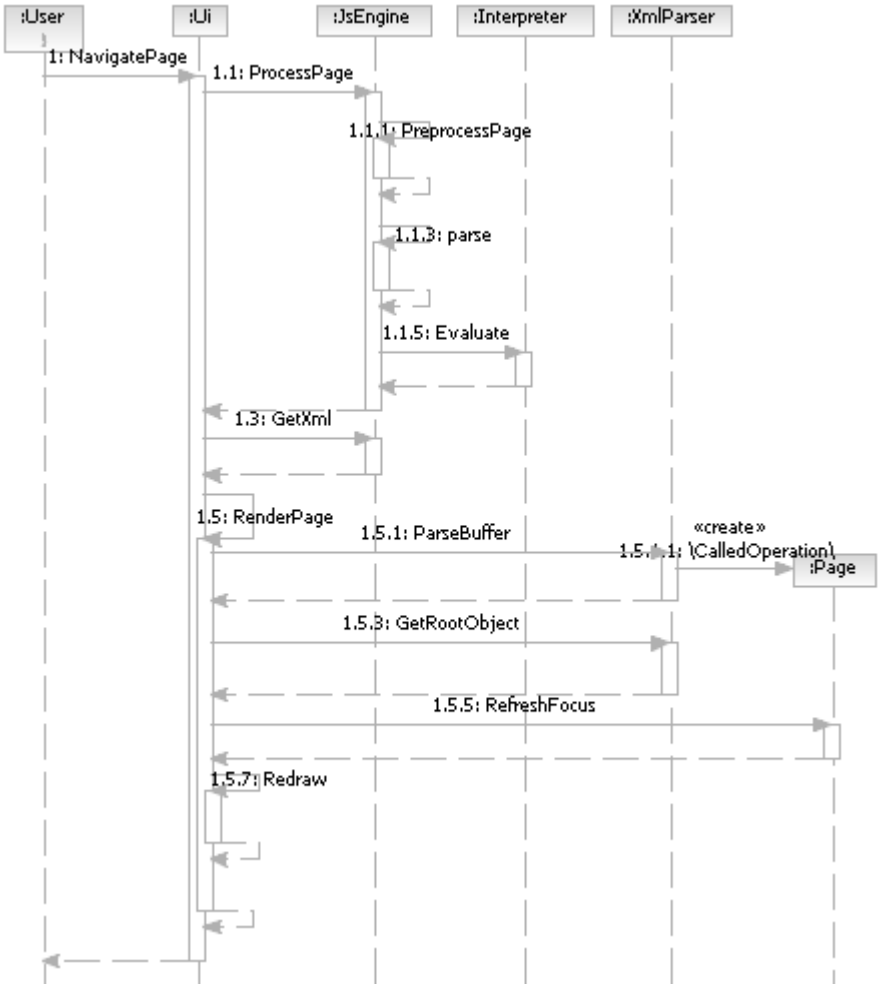
Menu klasėje realizuotas komponentas, atvaizduojantis meniu. Komponentas savyje laiko kitus komponentus – meniu punktus. Iškviečiamas paspaudus vieną iš telefono funkcinių klavišų (kairįjį arba dešinįjį).

MenuItem klasėje realizuotas komponentas, atstojantis meniu punktą. Komponentas atvaizduoja tekstą ir reaguoja į paspaudimus.

4.3. Dinaminis sistemos vaizdas

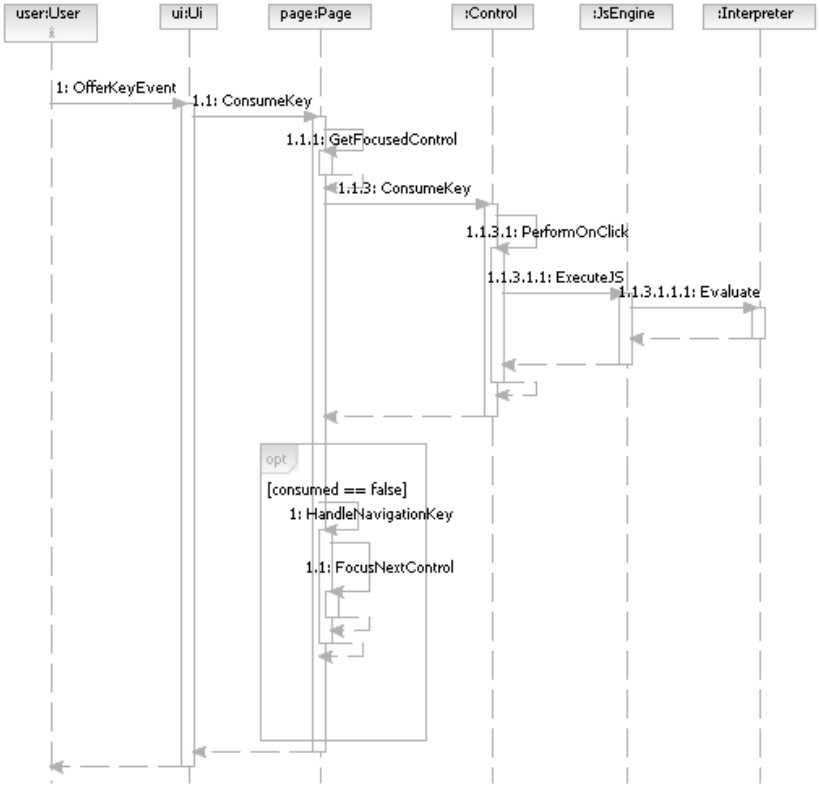
Ankstesniame skyriuje pateiktose klasių diagramose matome tik statinį sistemos vaizdą, tai yra jos struktūrą. Tačiau, norint suprasti sistemos funkcionavimą, būtinas ir dinaminis sistemos vaizdas. Šioje dalyje pademonstruosime komponentų sąveiką tarpusavyje. Pateiksime aukštos abstrakcijos pagrindines sistemos veiklas, atspindinčias sekų diagramas.

22-ame paveikslėlyje pateikta sekų diagrama vaizduoja naujo programos „puslapio“ apdorojimo veiklos eigą: nuo kodo nuskaitymo iki vartotojo grafinės sąsajos atvaizdavimo. Ši veiksmų seka tinka tiek pirmo, tiek ir kiekvieno sekančio programos lango aktyvavimui. Diagramoje matome, kad vartotojo veiksmus apdoroja *Ui* klasės tipo objektas. Vartotojas kreipiasi į *Ui* objektą, nurodydamas failą, kuriame aprašyta programos lango struktūra ir funkcionalumas. Nuskaičius failo duomenis, kreipiamasi į *JsEngine* objektą, kuris failo turinį padalina į dvi atskiras dalis: JavaScript kodą ir XML struktūras. Gramatinis analizatorius iš JavaScript kodo sukonstruoja sintaksinį medį, kuris perduodamas *Interpreter* objektui. *Interpreter* objektas įvykdo aprašytus veiksmus. Pasibaigus skripto vykdymui, *Ui* objektas kreipiasi į *XmlParser* objektą. *XmlParser* objektas pagal duotas XML struktūras sukuria vartotojo sąsajos komponentus. Atlikus visus veiksmus, siunčiamas grafinės sąsajos lango perpiešimo signalas.



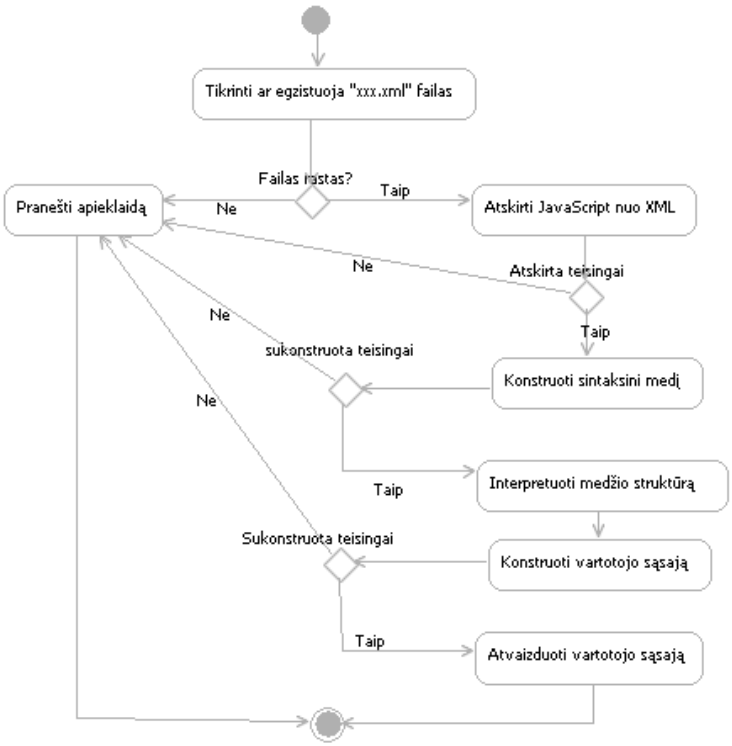
22 pav. Sekų diagrama, vaizduojanti parėjimą iš vieno programos lango į kitą.

23-ame paveikslėlyje pateikta sekų diagrama iliustruoja sistemos veiksmų seką, vykdomą apdorojant klaviatūros klavišo paspaudimo signalą. Signalas pirmiausia užregistruojamas *Ui* objekte, kuris signalą perduoda *Page* objektui. *Page* objektas suranda šiuo metu aktyvų vartotojo sąsajos komponentą ir jam perduoda klavišo paspaudimo signalą. Komponentas signalą „sunaudoja“, jei turi aprašytas šio mygtuko paspaudimo instrukcijas. Jei komponentas klavišo paspaudimo „nesunaudoja“, tikrinama, ar signalas atitinka kurį nors iš klaviatūros navigacijos klavišų. Jei taip, pagal apibrėžtą tvarką aktyvuojamas sekantis vartotojo sąsajos struktūroje esantis komponentas.



23 pav. Klaviatūros klavišo paspaudimo apdorojimo sekų diagrama.

24-ame paveikslėlyje pateikta veiklos diagrama iliustruoja veiklas, vykdomas naujo programos „puslapio“ apdorojimo metu.



24 pav. Naujo skripto puslapio apdorojimo veiklos diagrama.

4.4. Sistemos testavimas

Atliekant kuriamos programinės įrangos testavimą, buvo naudojami keli testavimo metodai. Pasirinkti toliau išvardinti testavimo metodai:

Vienetų testavimas. Atskirų vienetų (karkaso komponentų) testavimui naudojamas struktūrinis testavimo metodas. Testavimo metu yra atsižvelgiama į algoritmų struktūrą ir tikrinamas vieneto veikimas, bent vieną kartą vykdant kiekvieną jo sakinį.

Integracijos testavimas. Realizavus naują sistemos komponentą, jis prijungiamas prie jau veikiančios sistemos ir atliekamas pakartotinis sistemos integruotumo testas. Naudojama Top-Down strategija. Tokiu būdu anksčiausiai suintegruoti komponentai labiausiai ištestuoti. Šis testavimo metodas taip pat užtikrina, kad bet kuriuo momentu užsakovui galima pateikti veikiančią programos versiją.

Priėmimo testavimas. Testavimas atliekamas peržiūrint specifikaciją ir patikrinant ar programa turi reikalavimuose aprašytas savybes. Esant nesutarimams ar konfliktams, galutinis sprendimas priimamas remiantis reikalavimų specifikacija. Aptikus neatitikimus, sudaromas taisytinų programos savybių dokumentas, kuriame aprašomos klaidos ir jų taisymo metodai.

Regresinis testavimas. Realizavus naujas karkaso funkcijas, programinė įranga pakartotinai testuojama su ankščiau paruoštais ir ankstesniuose etapuose vykdytais testais. Taip pat sukuriami nauji testai naujų programinės įrangos galimybių funkcionavimo teisingumui patikrinti. Seni testai, neatitinkantys naujų programinės įrangos galimybių, nebenaudojami.

Stresinis testavimas. Karkasas testuojamas imituojant didelį kiekį telefono klavišų paspaudimų ir tikrinama, ar sistema spėja juos visus apdoroti, ar neįvyksta rimtų klaidų sistemoje.

Automatinis testavimas. Panaudojus realizuotą karkasą, sukuriama automatiškai testus vykdanti paprogramė. Kiekvieno testo rezultatai išsaugojami rezultatų byloje xml formatu. Baigus automatinį testavimą, rezultatų failas importuojamas į „TestLink“ sistemą. Šioje sistemoje patogiai galima matyti nepavykusius testus.

5. EKSPERIMENTINIS REALIZUOTO KARKASO TYRIMAS

5.1. Sukurtos sistemos kokybės tyrimas

Karkaso pagalba sukurta programinė įranga buvo išbandyta Symbian ir Windows Mobile operacinėse sistemose. Abiejose platformose karkaso pagalba sukurtos programinės įrangos vartotojo grafinė sąsaja atrodė ir reagavo į vartotojo veiksmus identiškai. Funkcionalumas taip pat nesiskyrė. Buvo išmatuota, kad Windows Mobile operacinėje sistemoje programa vykdoma lėčiau. Pastebėta, kad šį greičių skirtumą įtakojo skirtingos kompiliatorių, naudotų karkaso kompiliavimui Symbian ir Windows Mobile operacinėms sistemoms, versijos. Taip pat įtaką daro ir skirtinga standartinių C/C++ bibliotekų realizacija (pvz., pastebėta, kad Microsoft Windows Mobile operacinėje sistemoje atminties išskyrimo operacija vykdoma lėčiau, nei Symbian operacinėje sistemoje).

Realizuojant sistemą buvo iškelti du pagrindiniai uždaviniai:

- Užtikrinti nepriklausomumą nuo platformos;
- Supaprastinti programines sąsajas.

Iš atliktos srities analizės žinome, kad pasirinkto metodo, užtikrinančio programinės įrangos nepriklausomumą nuo platformos, panaudojimas neigiamai įtakoja programinės įrangos vykdymo greitį. Šioje dalyje ištirsime, kokią vykdymo greičio dalį prarandame, programos kūrimui naudodami realizuotą karkasą. Pasiūlysime karkaso tobulinimo galimybes greitaveikai pagerinti. Pakeitimus suprojektuosime ir realizuosime.

Greitaveikos lyginamasis tyrimas atliktas Symbian operacinėje sistemoje, naudojant Nokia N95 išmanųjį telefoną. Matavimams atlikti buvo naudojami du to paties funkcionalumo programos variantai, realizuoti skirtingomis priemonėmis:

1. Konkrečiai Symbian OS platformai skirta programos versija (realizuota Symbian OS programų kūrimo priemonėmis);
2. Nuo platformos nepriklausanti programos versija (realizuota naudojantis darbo metu suprojektuotu ir realizuotu karkasu).

Eksperimentui naudojamos programos funkcionalumui realizuoti panaudota:

- išmaniojo telefono resursų nuskaitymas (adresų knygutės informacijos nuskaitymas);
- aritmetiniai veiksmai su simbolių eilutės tipo kintamaisiais (sumos operacija);
- simbolių eilučių palyginimo operacijos;

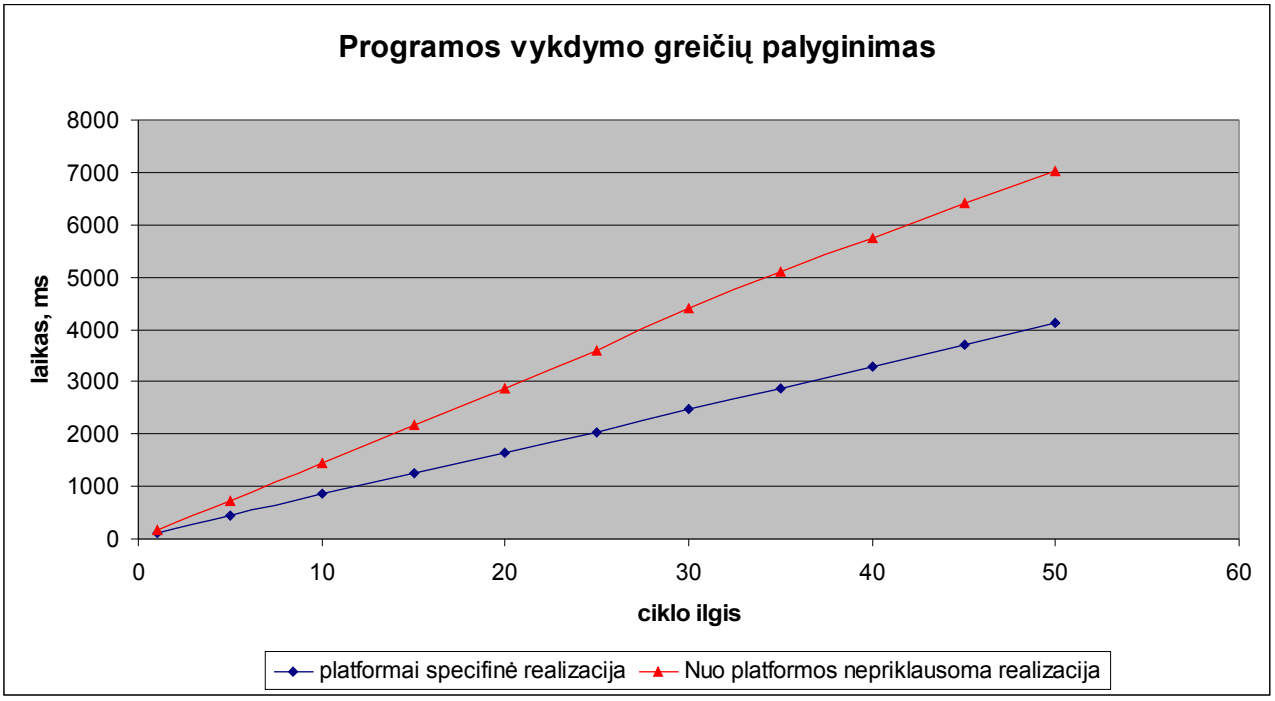
- informacijos išsaugojimas failų sistemoje.

Programos pagrindinio ciklo ilgis priklauso nuo adresų knygutėje esančių įrašų kiekio (vieno ciklo metu apdorojamas vienas adresų knygutės įrašas).

Eksperimentas buvo kartojamas adresų knygutėje esant: 5, 10, 15, 20, 25, 30, 35, 40, 45 ir 50 įrašų. Eksperimento metu sukaupti duomenys pateikti 1-ame ir 2-ame prieduose. Eksperimento rezultatai pateikti 6-oje lentelėje ir 25-ame paveikslėlyje. Vykdymo greičiai išreikšti milisekundėmis.

6 lentelė. Programų, realizuotų skirtingomis priemonėmis, vykdymo greičio palyginimas.

Programos versija	Ciklų skaičius									
	5	10	15	20	25	30	35	40	45	50
Prog. Nr.1 vykdymo laikas	447	850	1243	1643	2042	2481	2874	3287	3707	4114
Prog. Nr.2 vykdymo laikas	736	1462	2175	2881	3592	4401	5089	5754	6409	7032



25 pav. Skirtingomis priemonėmis realizuotos programos vykdymo greičio palyginimas.

Atlikus lyginamuosius programos vykdymo greičio tyrimus Symbian OS sistemoje paaiškėjo, kad tą patį funkcionalumą turinčios programos, realizuotos mūsų sukurtomis priemonėmis, vykdymo greitis vidutiniškai 74% atsilieka nuo programos, realizuotos specifinėmis Symbian sistemos priemonėmis, vykdymo greičio.

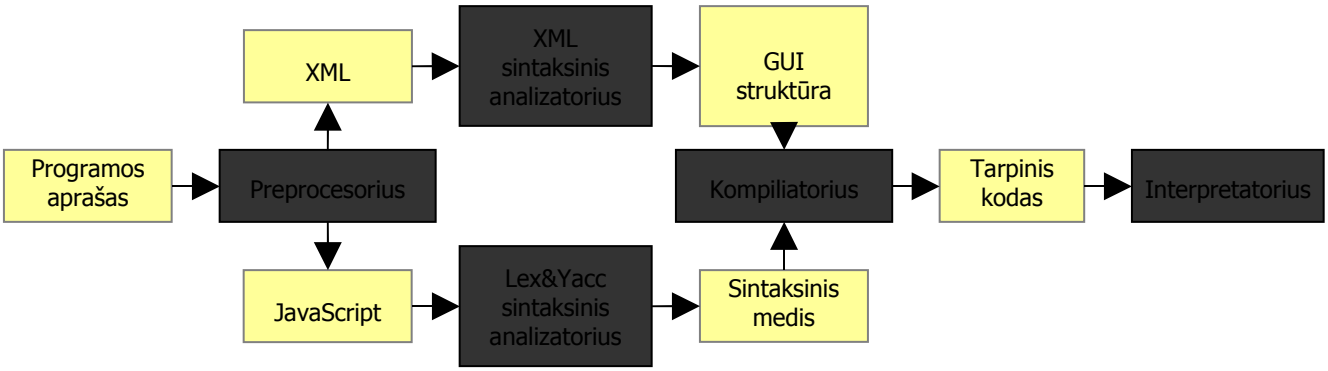
Norėdami įvertinti programinių sąsajų sudėtingumą, išmatavome abiejų programos versijų kodo eilučių kiekį. Šių programų išeities tekstai pateikti 4-ame ir 5-ame prieduose. Kodo eilučių kiekis atitinkamai:

- 244 kodo eilutės, naudojant Symbian OS priemones;
- 75 kodo eilutės, naudojant mūsų realizuotą karkasą.

Matome, kad mūsų nagrinėjamu atveju tam pačiam funkcionalumai pasiekti Symbian OS platformoje, naudojant Symbian OS priemones (SDK), prireikė 169 kodo eilutėmis (maždaug 3,2 karto) daugiau, nei naudojant mūsų sukurtą karkasą. Šiuo konkrečiu atveju kodo eilučių sumažėjimą daugiausia lėmė aukštesnio lygio sąsaja adresų knygutės informacijai nuskaityti. Kodo eilučių kiekį taip pat įtakojo ir paprastesnės operacijos su simbolių eilutės tipo kintamaisiais. Atsižvelgdami į kodo eilučių sumažėjimą galime teigti, kad mūsų sukurto karkaso programinės sąsajos paprastesnės nei standartinės Symbian OS programinės sąsajos.

5.2. Sistemos tobulinimo galimybių analizė ir patobulinimų realizavimas

Norint paspartinti programų vykdymą, buvo nuspręsta sistemoje realizuoti komponentą (toliau kompiliatorius), kuris sintaksinį medį perverstų į tarpinį kodą. Tarpinis kodas artimas mašininiam kodui, todėl jo interpretavimas greitesnis nei sintaksinio medžio. Tokiu atveju, pirmą kartą paleista programa bus vykdoma ilgiau dėl pradinių skriptų kompiliavimo į tarpinį kodą. Tačiau prireikus pakartoti jau anksčiau vykdytus veiksmus, jie bus vykdomi greičiau, dėka jau anksčiau sukompiliuoto tarpinio kodo. Skriptų apdorojimo procesas, panaudojus kompiliatorių, pateiktas 26-ame paveikslėlyje.

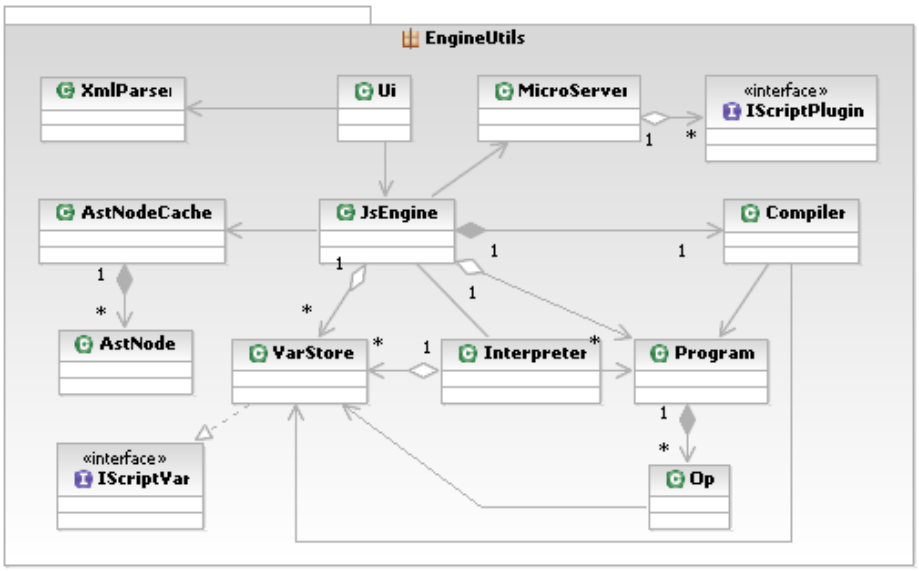


26 pav. Skriptų apdorojimo procesas, į procesą įtraukus kompiliatorių.

Kompiliatoriaus funkcionalumas realizuojamas *EngineUtils* pakete. Sukuriamos trys naujos klasės:

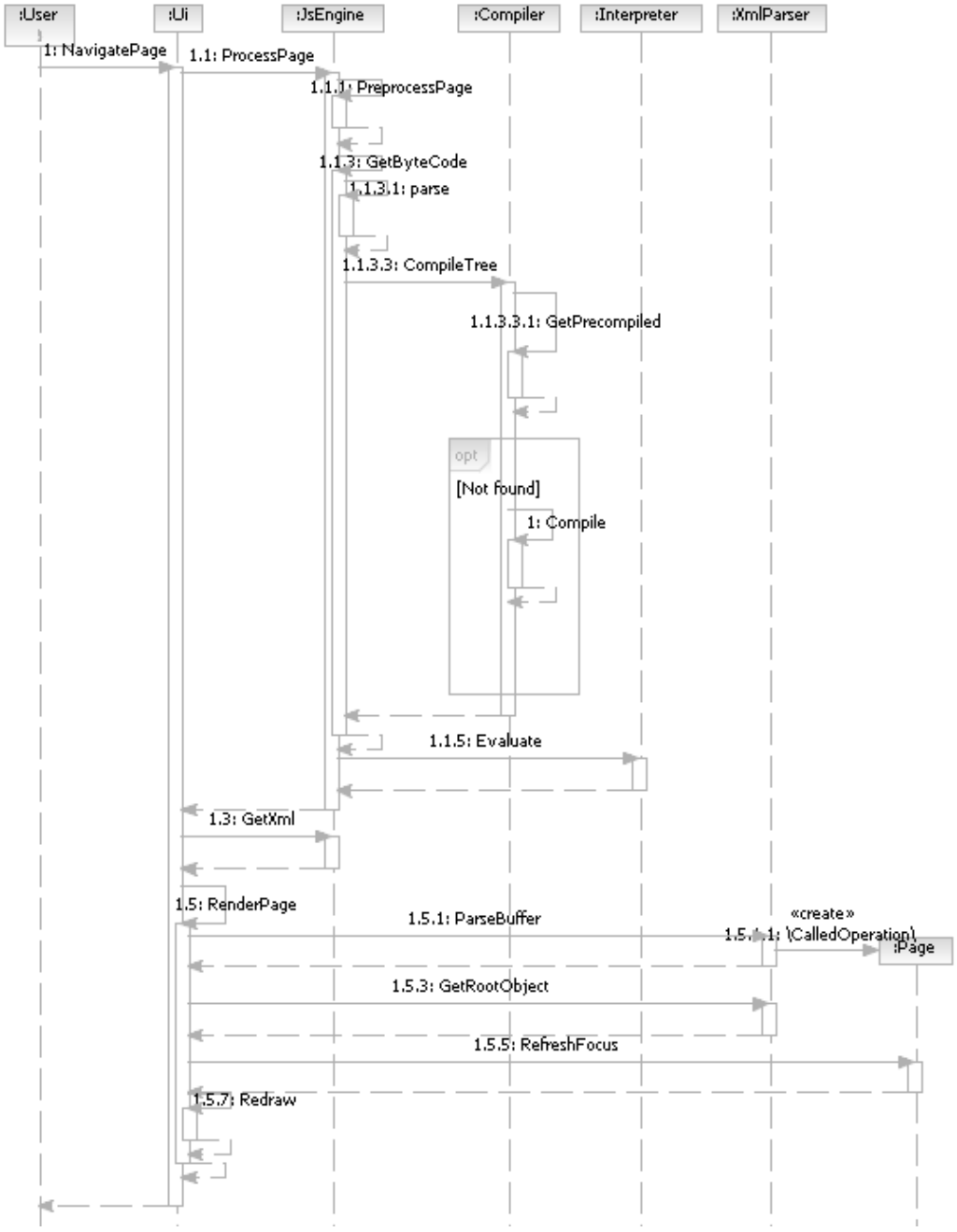
- *Compiler* – klasė, iš sintaksinio medžio sugeneruojanti operacijų seką (tarpinį kodą);
- *Program* – klasė, sauganti sukompiliuotą operacijų seką;
- *Op* – operaciją apibrėžianti klasė.

Nedetalizuota paketo *EngineUtils* klasių diagrama, po kompiliatoriaus realizavimo, pateikta 27-ame paveikslėlyje.



27 pav. Paketo EngineUtils klasių diagrama, realizavus kompiliatorių.

28-ame paveikslėlyje pateikta sekų diagrama vaizduoja naujo programos „puslapio“ apdorojimo veiklos eigą, sistemoje realizavus kompiliatorių. Diagramoje matome, kad gramatinio analizatoriaus iš JavaScript kodo sukonstruotas sintaksinis medis perduodamas ne *Interpreter* (kas buvo daroma ankstesnėje karkaso versijoje), o *Compiler* komponentui. *Compiler* komponentas pirmiausia ieško nurodyto sintaksinio medžio atitikmens jau sukompiliuoto tarpinio kodo lentelėje. Neradus atitikmens, atliekamas sintaksinio medžio kompiliavimas. Rezultatas vėliau perduodamas *Interpreter* komponentui.



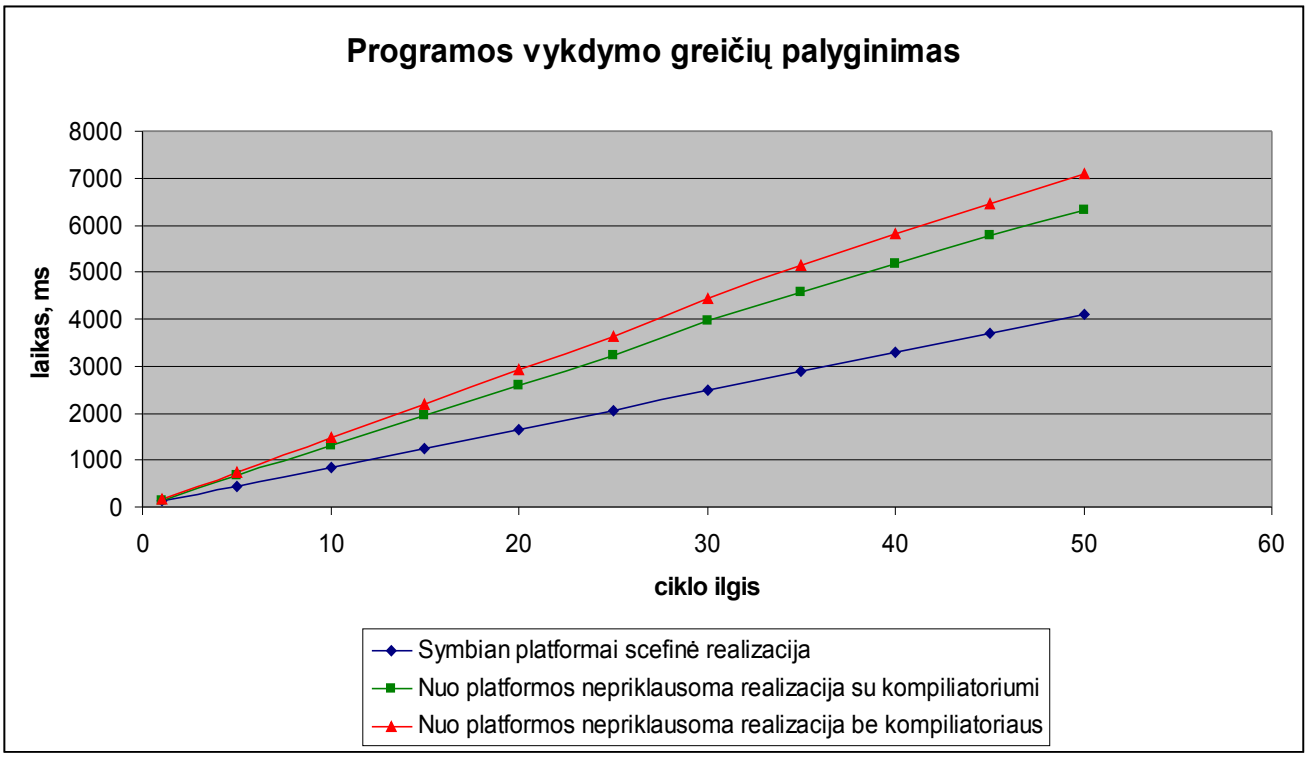
28 pav. Sekų diagrama, vaizduojanti perėjimą iš vieno programos lango į kitą, sistemoje realizavus kompiliatorių.

Realizavus sistemos patobulinimus, buvo pakartotas greitaveikos lyginimo eksperimentas. Gauti duomenys (3-čias priedas) palyginti su surinktais ankščiau. Palyginimas pateiktas 7-oje lentelėje ir 29-ame paveikslėlyje. Matome, kad sistema, interpretuodama tarpinį kodą, vykdymui sugaišo maždaug 18% mažiau laiko, nei interpretuodama sintaksinį medį. Kadangi šiuo konkrečiu atveju visa programa realizuota viename puslapyje, o tiriama programos logika atliekama tik paspaudus klavišą, į išmatuotą laiką neįeina kompiliavimui sugaištas laikas. Kadangi tiriama programa nedidelė, kompiliavimui skirtas laikas taip pat nedidelis. Atlikus patobulinimus,

eksperimentui naudojama programos versija, realizuota sukurto karkaso pagalba, įvykdyta apytiksliai 56% lėčiau nei versija, realizuota standartinėmis Symbian OS priemonėmis.

7 lentelė. Sintaksinio medžio ir tarpinio kodo interpretavimo greičių palyginimas.

Programos versija	Ciklų skaičius									
	5	10	15	20	25	30	35	40	45	50
Symbian platformai specifinė realizacija	447	850	1243	1643	2042	2481	2874	3287	3707	4114
Realizuota naudojant karkasą su kompiliatorium	663	1317	1959	2596	3236	3964	4585	5184	5774	6335
Realizuota naudojant karkasa be kompiliatoriaus pagalba	736	1462	2175	2881	3592	4401	5089	5754	6409	7032



29 pav. Programos vykdymo laiko palyginimas realizavus kompiliatorių.

5.3. Sistemos apribojimai ir galimybės

Nepriklausomumo nuo platformos ir programinių sąsajų paprastumo sąskaita sulėtėja programos vykdymo sparta. Todėl karkasas netinka naudoti programoms, kurių greitas vykdymas kritiškai svarbus. Karkaso priemonės taip pat netinka kurti didelėms ir sudėtingoms sistemoms, atliekančioms daugybę sudėtingų skaičiavimų, nes interpretatorius šiuos skaičiavimus atliks daugybę kartų lėčiau, nei tai atliktų kompiliuota programos versija.

Tačiau programos, kurios nėra itin sudėtingos, nereikalauja daug sudėtingų skaičiavimų ir kurioms vykdymo greičio sumažėjimas didelės reikšmės neturi, naudojant mūsų realizuotą karkasą bus sukurtos daug greičiau ir pigiau.

Pagrindinis darbo metu realizuoto karkaso apribojimas – tai, kad neskiriant didelių investicijų jo pagalba programinė įranga gali būti realizuota tik platformoms, kuriose programų kūrimui naudojamos C/C++ kalbos. Norint palaikyti daugiau platformų, reikėtų realizuoti karkaso versiją kita dažnai išmaniųjų telefonų operacinėse sistemose naudojama programavimo kalba – Java. Pakartotinai panaudojus tą patį modelį bei architektūrą, būtų sukurtos dvi nepriklausomos kodo bazės skirtingomis programavimo kalbomis. Tuomet karkaso pagalba realizuota programinė įranga galėtų pasiekti daugiau nei 90% į rinką išleidžiamų išmaniųjų telefonų.

6. IŠVADOS

1. Programinės įrangos išmaniesiems telefonams kūrimą apsunkina daugybė skirtingų ir nesuderinamų platformų. Norint pateikti programinę įrangą kuo didesniam vartotojų skaičiui, tenka realizuoti skirtingas versijas kelioms platformoms. Dėl sąlyginai sudėtingų programinių sąsajų ir specialistų trūkumo, tokios programinės įrangos kūrimo kaštai ganėtinai dideli. Atsižvelgiant į šias problemas, buvo apibrėžtas šio darbo tikslas – sukurti metodą nepriklausomai nuo platformos programinei įrangai išmaniesiems telefonams kurti.
2. Buvo apžvelgti literatūroje siūlomi ir galimi pritaikyti minėtos problemos sprendimo būdai. Atlikus analizę buvo nuspręsta nepriklausomumo nuo platformos užtikrinimui panaudoti interpretatorių. Programos funkcionalumui bei grafinės vartotojo sąsajos aprašymui pasirinktos plačiai naudojamos ir gerai žinomos interneto technologijas: JavaScript - programos funkcionalumui aprašyti, ir XML -vartotojo grafinės sąsajos struktūrai aprašyti.
3. Darbo metu buvo realizuotas nuo platformos nepriklausomos programinės įrangos išmaniesiems telefonams karkasas. Jo realizacija paremta siūlomu metodu. Realizuojant karkasą, buvo įvestas apribojimas programavimo kalbai. Ši karkaso realizacija palaiko tik tas platformas, kurioms programas galima realizuoti C/C++ programavimo kalbomis (apytiksliai 60% rinkos). Turint karkaso realizacijas dvejomis pagrindinėmis, išmaniuosiuose telefonuose naudojamomis, programavimo kalbomis, panaudojus aprašomą metodą, ta pati programinės įrangos realizacija galėtų būti naudojama 90% išmaniųjų telefonų rinkos.
4. Karkasas buvo projektuojamas taip, kad jo funkcionalumą ir palaikomų platformų kiekį būtų kuo paprasčiau praplėsti. Realizuojant karkasą buvo naudojamos standartinės C/C++ bibliotekos, kurias palaiko visos C/C++ platformos. Galutinėje karkaso realizacijoje 80% kodo buvo parašyta nepriklausomomis nuo platformos priemonėmis. Dėl šios priežasties, norint praplėsti palaikomų platformų kiekį, tereikia perrašyti 20% kodo dalį, priklausančią nuo platformos.
5. Buvo atliktas karkaso kokybės ir panaudojimo tyrimas. To paties funkcionalumo mini programa buvo realizuota dviem būdais: naudojantis sukurtu karkasu ir naudojantis Symbian OS specifinėmis priemonėmis. Tam pačiam funkcionalumui realizuoti Symbian OS specifinėmis priemonėmis prireikė 3,2 karto daugiau kodo eilučių nei naudojantis realizuoto karkaso priemonėmis. Tai rodo, jog karkaso pagalba programų kūrimas

paprastesnis. Tačiau karkaso pagalba realizuota programa buvo vykdoma 56% ilgiau nei naudojant standartines Symbian OS priemones. Tai rodo, kad karkasas netinka realizuoti programas, kurioms greitas vykdymas yra kritinės svarbos.

7. LITERATŪRA

1. *Samsung launches open mobile platform*. November 10th, 2009, Seoul, Korea. [žiūrėta 2010.05.10] Prieiga internete < http://www.bada.com/samsung-launches-open-mobile-platform/>
2. *Operating System Market Share*. Net Aplication. [žiūrėta 2010.05.10] Prieiga internete <http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=10>
3. *Gartner Says Worldwide Mobile Phone Sales to End Users Grew 8 Per Cent in Fourth Quarter 2009; Market Remained Flat in 2009*. Gartner. February 23, 2010. [žiūrėta 2010.05.11] Prieiga internete <http://www.gartner.com/it/page.jsp?id=1306513>
4. Harrison, R.; *Symbian OS C++ forMobile Phones*. JohnWiley and Sons Ltd, England, 2003.
5. Thai, T.; Lam, H.; *.NET Framework Essentials*, O'Reilly, 2003.
6. Cook, S.; *Domain-Specific Modeling and Model Driven Architecture*. MDA Journal, January 2004.
7. Sanchez, P.; Barreda, J.; Ocon, J.; *Integration of domain-specific models into a MDA framework for time-critical embedded systems*. Intelligent Solutions in Embedded Systems, 2008.
8. Forstner, B.; Lengyel, L.; Levendovszky, T.; Mezei, G.; *Model-Based System Development for Embedded Mobile Platforms.* Budapest University of Technology and Economics, 2006.
9. Balagtas-Fernandez, F.T.; Hussmann, H.; *Model-Driven Development of Mobile Applications*. Department of Computer Science, University of Munich, Germany, 2008.
10. Aycock, J.; *A brief history of just-in-time*. University of Calgary, Calgary, Canada 2003.
11. Bik, A.J.C.; Girkar, M.; Haghighat, M.R.; *Experiences with java™ jit optimization*. Intel Corporation, CA, 1999.
12. Yoshikawa, T.; Shimura, K.; Ozawa, T.; *Random program generator for Java JIT compiler test system*. Fujitsu Labs. LTD, Kawasaki, Japan, 2003.
13. *Java ME Technology*. Sun Developer Network. [Žiūrėta 2010.05.12] Prieiga internete <http://java.sun.com/javame/technology/index.jsp>
14. Boswell, D.; King, B.; Oeschger, I.; Collins, P.; Murphy, E.; *Creating Appications with Mozilla*. O'Reilly & Associates, Inc., Sebastopol, CA, 2002.
15. *Nokia enriches application development with Qt for S60*. Nokia Communications, 2008.
16. Blanchette, J.; Summerfield, M.; *C++ GUI Programming with Qt 4.* Prentice Hall; 2 edition, 2008.

17. Butter, T. Aleksy, M. Bostan, P. Schader, M. *Context-aware User Interface Framework for Mobile Applications*. Univ. of Mannheim-Schloss, Mannheim, 2007.
18. Tak Lam; Ding, J.J.; Jyh-Charn Liu; XML Document Parsing: Operational and Performance Characteristics. Cisco Syst., San Jose, CA, 2008.
19. Su Cheng Haw; Krishna Rao, G.S.V.R.; *A Comparative Study and Benchmarking on XML Parsers*. Fac. of Inf. Technol., Multimedia Univ., Cyberjaya, 2007.
20. Niemann, T.; *A Compact Guide to Lex & Yacc*. ePaper Press

Platform independent software framework for smartphones

SUMMARY

With the introduction and popularity of wireless devices, the diversity of the platforms has also been increased. There are different platforms and tools from different vendors such as Microsoft, Sun, Nokia, SonyEricsson and many more. Because of the relatively low-level programming interface, software development for wireless devices platforms, such as Symbian, is a tiresome and error prone task. This paper introduces the problem of the software development for incompatible mobile platforms, moreover, it provides a solution based on easy to use web technologies – JavaScript and XML. In the presented approach, the platform-independence lies in JavaScript interpretation and XML structures based user interface.

The subject of research, object and subject relevance were detectable through literature analysis and through existing approaches are presented in the first part.

The platform independent mobile application framework was implemented to demonstrate the proposed solution. System components design and programming process are presented in the second part. The implemented system static view is provided by using class diagrams and dynamic view - by sequences and activities diagrams.

In the third part are provided assessment of the platform independent mobile software framework capabilities and quality. Program developed by using this framework was compared to program developed by using standard Symbian OS native means.

TERMINŲ IR SANTRUMPŲ ŽODYNAS

3G – trečios kartos mobiliojo ryšio technologija, sukurta 1999 m. Pažangesnė už GSM technologiją.

Wi-Fi – bevielio ryšio technologija, skirta įrenginiams naudojantiems plačiajuostį radijo ryšį.

GPS - visuotinė padėties nustatymo sistema. Leidžia nustatyti objekto koordinates bet kurioje pasaulio vietoje. Sistemos pagrindas – IT technologijų sąveika su planetą gaubiančiu GPS palydovų tinklu. Tai viena iš palydovinių navigacijos sistemų.

Android – Google korporacijos sukurta operacinė sistema išmaniesiems telefonams.

Symbian - Symbian Foundation kuriama operacinė sistema. Skirta išmaniesiems telefonams ir delniniams kompiuteriams. Symbian OS yra kilusi iš EPOC OS.

Bada – Samsung korporacijos kuriama nauja operacinė sistema išmaniesiems telefonams.

iPhone – Apple sukurtas išmanusis telefonas ir to paties pavadinimo jame naudojama operacinė sistema.

JavaSE – standartinė Java versija (angl. Java Standard Edition).

SDK – programinės įrangos kūrimo įrankių komplektas (angl. Software Development Kit).

JVM – Java tarpinio kodo (angl. bytecode) interpretatorius(ang. Java Virtual Mashine).

J2ME – Java kalbos versija skirta išmaniesiems telefonams (angl. Java 2 Micro Edition).

CSS – kalba, naudojama aprašyti HTML/XML komponentų išvaizdą (angl. Cascading Style Sheet).

JavaScript – prototipais paremta objektiškai orientuota skriptų rašymo klaba, paprastai naudojama dinaminių tinklalapių kūrimui.

XUL – XML notacija paremta vartotojo sąsajos aprašymo kalba (angl. XML User Interface Language).

XML – taisyklių rinkinys informacijos saugojimui dokumentuose elektroninėje erdvėje (angl. Extensible Markup Language).

XPCOM – nuo platformos nepriklausomas komponentų modelis (Cross Platform Component Model). Jo pagalba programuotojai gali įterpti papildomą funkcionalumą į XPEE.

DHTML – dinaminis HTML. Technologijų rinkinys, skirtas sąveikaujantiems su vartotojų interneto puslapiams kurti.

XPFE – nuo platformos nepriklausomų komponentų rinkinys, sukurtas Mozilla kompanijos.

C/C++ - bendros paskirties, kompiliuojama programavimo klaba, turinti tiek ir aukšto, tiek ir žemo lygio kalboms būdingų bruožų.

MDA – modeliu paremta architektūra (angl. Model Driven Architecture) yra programinės įrangos projektavimo metodas. Numato tam tikras programinės įrangos specifikavimo ir struktūros apibrėžimo gaires.

OMG (angl. Object Management Group) – konsorciumas, kurio pirminis tikslas buvo nustatyti objektiškai orientuotos programinės įrangos standartus. Šiuo metu konsorciumas didžiausią dėmesį teikia programų, sistemų, biznio modelių procesų modeliavimui ir modeliu paremtiems standartams.

bytecode – instrukcijų rinkinys, suprojektuotas greitam programinės įrangos vykdymui interpretatoriumi.

JIT – dinaminis kompiliavimas (angl. Just-in-time compilation). Technologija skirta programų vykdymo greičio padidinimui. Tai interpretatoriaus ir kompiliatoriaus hibridas. Kodas kompiliuojamas prieš pat interpretavimą.

PIM – nuo platformos nepriklausantis modelis (angl. Platform Independent Model).

PSM – platformai specifinis modelis (angl. Platform Specific Model).

OS – operacinė sistema (angl. Operating System).

DSM – sričiai specifinis modeliavimas (angl. Domain Specific Modeling). Programų inžinerijos metodologija. Apibrėžia grafinės, sričiai specifinės kalbos, panaudojimą sistemos aprašymui.

DSML – sričiai specifinė modeliavimo kalba (angl. Domain Specific Modeling Language).

UML – unifikuota modeliavimo kalba (angl. Unified Modeling Language), skirta programinės įrangos projektavimui.

SAX – paprasta programinė sąsaja XML formato skaitymui (angl. Simple API for XML). Sąsaja paremta duomenų nuskaitymų serijomis.

DOM – dokumento objektų modelis (angl. Document Object Model). Programinė sąsaja HTML ir XML objektų valdymui.

PRIEDAI

1 priedas. Eksperimente naudotos programos, realizuotos Symbian OS specifinėmis priemonėmis, vykdymo laiko matavimo rezultatų lentelė.

Nr	Įrašų kiekis adresų knygutėje										
	1	5	10	15	20	25	30	35	40	45	50
1	121	452	845	1262	1667	2068	2515	2877	3298	3708	4160
2	124	443	846	1242	1649	2042	2476	2866	3324	3711	4158
3	122	448	848	1235	1644	2048	2489	2864	3311	3677	4145
4	134	457	853	1247	1644	2033	2462	2866	3294	3699	4169
5	124	451	852	1234	1643	2038	2501	2887	3305	3713	4128
6	126	448	853	1244	1643	2032	2469	2870	3277	3705	4128
7	125	448	849	1246	1654	2051	2487	2863	3321	3695	4129
8	123	446	850	1244	1664	2045	2511	2867	3310	3718	4149
9	129	451	853	1244	1649	2048	2480	2859	3291	3693	4118
10	124	449	886	1243	1656	2042	2497	2866	3300	3707	4126
11	122	448	851	1250	1657	2044	2487	2867	3294	3724	4124
12	123	450	852	1242	1661	2044	2470	2883	3306	3719	4102
13	126	446	847	1244	1643	2045	2477	2891	3273	3695	4071
14	124	454	851	1244	1638	2041	2481	2882	3300	3702	4095
15	124	445	854	1250	1649	2045	2478	2865	3301	3692	4100
16	122	446	860	1246	1645	2048	2496	2869	3311	3726	4098
17	129	446	847	1249	1643	2047	2467	2873	3292	3741	4095
18	126	444	847	1237	1659	2064	2472	2879	3310	3731	4084
19	121	443	845	1238	1637	2037	2470	2877	3270	3721	4088
20	123	448	848	1238	1632	2036	2471	2880	3288	3709	4119
21	122	443	847	1243	1635	2032	2473	2878	3282	3766	4101
22	129	445	847	1263	1628	2035	2469	2883	3265	3735	4110
23	123	446	844	1237	1626	2028	2480	2888	3245	3732	4085
24	124	443	848	1238	1637	2037	2488	2874	3263	3671	4106
25	123	451	845	1245	1631	2043	2486	2884	3258	3689	4103
26	124	445	850	1240	1635	2040	2494	2869	3261	3688	4104
27	122	455	850	1242	1636	2052	2467	2876	3244	3691	4109
28	123	443	846	1242	1635	2044	2468	2866	3259	3670	4107
29	122	442	854	1239	1634	2035	2483	2882	3300	3681	4122
30	127	441	848	1236	1626	2037	2466	2897	3283	3706	4093
Vid	**124**	**447**	**851**	**1243**	**1643**	**2043**	**2481**	**2875**	**3288**	**3707**	**4114**
min	**121**	**441**	**844**	**1234**	**1626**	**2028**	**2462**	**2859**	**3244**	**3670**	**4071**
max	**134**	**457**	**886**	**1263**	**1667**	**2068**	**2515**	**2897**	**3324**	**3766**	**4169**

2 priedas. Eksperimente naudotos programos, realizuotos darbo metu sukurto karkaso priemonėmis, vykdymo laiko matavimo rezultatų lentelė.

nr.	Įrašų kiekis adresų knygutėje										
	1	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
1	184	777	1487	2197	2882	3796	4398	5114	5744	6401	7032
2	149	719	1456	2169	2858	3564	4385	5087	5698	6426	7041
3	153	738	1449	2165	2864	3586	4366	5070	5722	6427	6965
4	154	734	1457	2165	2857	3582	4373	5067	5719	6426	6975
5	158	730	1462	2157	2870	3578	4373	5074	5762	6434	6971
6	153	739	1460	2171	2862	3591	4382	5079	5718	6411	6991
7	165	728	1460	2161	2877	3582	4381	5077	5741	6392	7006
8	154	738	1454	2171	2863	3592	4376	5082	5742	6390	6986
9	161	733	1460	2170	2872	3581	4396	5070	5748	6394	7022
10	153	728	1475	2170	2872	3594	4387	5059	5754	6394	6901
11	152	746	1464	2162	2879	3588	4394	5059	5743	6399	6925
12	152	730	1461	2178	2875	3598	4421	5074	5759	6435	7041
13	150	730	1461	2174	2885	3594	4401	5078	5769	6383	7043
14	158	739	1470	2174	2894	3635	4407	5078	5771	6384	7060
15	159	733	1470	2169	2882	3604	4403	5065	5776	6396	7053
16	152	744	1464	2178	2898	3606	4400	5057	5756	6390	7070
17	152	736	1461	2169	2883	3576	4419	5104	5732	6414	7058
18	164	736	1473	2166	2896	3575	4403	5087	5765	6385	7013
19	154	736	1462	2179	2900	3584	4409	5094	5744	6395	6983
20	154	728	1471	2178	2908	3573	4426	5109	5743	6373	7037
21	153	765	1469	2182	2904	3575	4429	5094	5751	6411	7051
22	153	734	1464	2189	2898	3576	4420	5100	5761	6408	7054
23	164	736	1469	2182	2912	3578	4426	5114	5756	6418	7051
24	152	743	1460	2174	2900	3583	4429	5107	5760	6457	7053
25	152	733	1459	2174	2883	3602	4394	5116	5762	6431	7075
26	159	728	1476	2176	2896	3575	4421	5122	5772	6437	7080
27	154	727	1459	2189	2877	3573	4401	5105	5775	6405	7082
28	154	730	1450	2187	2876	3575	4407	5130	5781	6415	7090
29	152	737	1447	2186	2864	3572	4403	5116	5801	6437	7107
30	158	730	1456	2200	2867	3586	4400	5109	5802	6421	7169
vid	**156**	**736**	**1463**	**2175**	**2882**	**3593**	**4401**	**5090**	**5754**	**6410**	**7033**
min	**149**	**719**	**1447**	**2157**	**2857**	**3564**	**4366**	**5057**	**5698**	**6373**	**6901**
max	**184**	**777**	**1487**	**2200**	**2912**	**3796**	**4429**	**5130**	**5802**	**6457**	**7169**

3 priedas. Eksperimente naudotos programos, realizuotos patobulinto (su kompiliavimu) karkaso priemonėmis, vykdymo laiko matavimo rezultatų lentelė.

nr.	Įrašų kiekis adresų knygutėje										
	1	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
1	166	700	1340	1979	2596	3420	3962	4607	5175	5767	6335
2	134	648	1312	1954	2575	3211	3950	4583	5133	5789	6343
3	138	665	1305	1950	2580	3231	3933	4568	5155	5790	6275
4	139	661	1313	1950	2574	3227	3940	4565	5152	5789	6284
5	142	658	1317	1943	2586	3223	3940	4571	5191	5796	6280
6	138	666	1315	1956	2578	3235	3948	4576	5151	5776	6298
7	149	656	1315	1947	2592	3227	3947	4574	5172	5759	6312
8	139	665	1310	1956	2579	3236	3942	4578	5173	5757	6294
9	145	660	1315	1955	2587	3226	3960	4568	5178	5760	6326
10	138	656	1329	1955	2587	3238	3952	4558	5184	5760	6217
11	137	672	1319	1948	2594	3232	3959	4558	5174	5765	6239
12	137	658	1316	1962	2590	3241	3983	4571	5188	5797	6343
13	135	658	1316	1959	2599	3238	3965	4575	5197	5750	6345
14	142	666	1324	1959	2607	3275	3970	4575	5199	5751	6360
15	143	660	1324	1954	2596	3247	3967	4563	5204	5762	6354
16	137	670	1319	1962	2611	3249	3964	4556	5186	5757	6369
17	137	663	1316	1954	2597	3222	3981	4598	5164	5778	6359
18	148	663	1327	1951	2609	3221	3967	4583	5194	5752	6318
19	139	663	1317	1963	2613	3229	3972	4589	5175	5761	6291
20	139	656	1325	1962	2620	3219	3987	4603	5174	5741	6340
21	138	689	1323	1966	2616	3221	3990	4589	5181	5776	6352
22	138	661	1319	1972	2611	3222	3982	4595	5190	5773	6355
23	148	663	1323	1966	2623	3223	3987	4607	5186	5782	6352
24	137	669	1315	1959	2613	3228	3990	4601	5189	5817	6354
25	137	660	1314	1959	2597	3245	3959	4609	5191	5794	6374
26	143	656	1330	1960	2609	3221	3983	4614	5200	5799	6378
27	139	655	1314	1972	2592	3219	3965	4599	5203	5770	6380
28	139	658	1306	1970	2591	3221	3970	4622	5208	5779	6387
29	137	664	1304	1969	2580	3218	3967	4609	5226	5799	6403
30	142	658	1312	1982	2583	3231	3964	4603	5227	5785	6459
vid	**141**	**663**	**1318**	**1960**	**2596**	**3237**	**3965**	**4586**	**5184**	**5774**	**6336**
min	**134**	**648**	**1304**	**1943**	**2574**	**3211**	**3933**	**4556**	**5133**	**5741**	**6217**
max	**166**	**700**	**1340**	**1982**	**2623**	**3420**	**3990**	**4622**	**5227**	**5817**	**6459**

4 priedas. Eksperimentui naudotos programos, realizuotos darbo metu sukurtu karkasu, išeities kodas

```
 1: function execute()
 2: {
 3:    var start = device.gettime();
 4:    var handle = connector.open( "file:///CA/var/output.txt" );
 5:    contactsmanager.init();
 6:    var result = contactsmanager.retrievenext(100);
 7:    for ( var i = 0; i < result.length; i++ )
 8:    {
 9:       var contact = result[i];
10:       if (contact.title != null)
11:       {
12:          printToFile("title", contact.title);
13:       }
14:       if (contact.name != null)
15:       {
16:          printToFile("name", contact.name);
17:       }
18:       if (contact.additionalname != null)
19:       {
20:          printToFile("additionalname", contact.additionalname);
21:       }
22:       if (contact.familyname != null)
23:       {
24:          printToFile("familyname", contact.familyname);
25:       }
26:       if (contact.suffixname != null)
27:       {
28:          printToFile("suffixname", contact.suffixname);
29:       }
30:       if (contact.organisation != null)
31:       {
32:          printToFile("organisation", contact.organisation);
33:       }
34:       if (contact.homephone != null)
35:       {
36:          printToFile("homephone", contact.homephone);
37:       }
38:       if (contact.workphone != null)
39:       {
40:          printToFile("workphone", contact.workphone);
41:       }
42:       if (contact.mobilephone != null)
43:       {
44:          printToFile("mobilephone", contact.mobilephone);
45:       }
46:       if (contact.faxphone != null)
47:       {
48:          printToFile("faxphone", contact.faxphone);
49:       }
50:       if (contact.otherphone != null)
51:       {
52:          printToFile("otherphone", contact.otherphone);
53:       }
54:       if (contact.homeemail != null)
55:       {
56:          printToFile("homeemail", contact.homeemail);
57:       }
58:       if (contact.workemail != null)
59:       {
60:          printToFile("workemail", contact.workemail);
61:       }
62:       if (contact.otheremail != null)
63:       {
64:          printToFile("otheremail", contact.otheremail);
65:       }
66:       connector.write( handle, "\n", true );
67:    }
68:   connector.close( handle );
69: }
70:
```

```
71: function printToFile (handle, label, value)
72: {
73:    var workphone = label + ":" + value + " | ";
74:    connector.write( handle, contact.workphone, true );
75: }
```

5 priedas. Eksperimente naudotos programos, realizuotos specifinėmis Symbian OS priemonėmis, išeities kodas.

```
 1: void CHelloWorldBasicAppUi::HandleCommandL( TInt aCommand )
 2: {
 3:    RFs iFs;
 4:    iFs.Connect();
 5:    RFile file;
 6:    file.Replace(iFs, KOutputFileName, EFileWrite | EFileStream );
 7:    CSkContactsManager* manager = new CSkContactsManager;
 8:    manager->Init();
 9:    RPointerArray<ContactData>* array = manager->RetrieveNext( 100 );
10:    if (array != NULL)
11:    {
12:       TInt count = array->Count();
13:       for (TInt i = 0; i < count; i++)
14:       {
15:          ContactData* contact = array->operator [](i);
16:          if (contact->getTitle())
17:          {
18:             PrinttToFile(file, _L8("title"), contact->getTitle()->Des());
19:          }
20:          if (contact->getName())
21:          {
22:             PrinttToFile(file, _L8("name"), contact->getName()->Des());
23:          }
24:          if (contact->getAdditionalname())
25:          {
26:             PrinttToFile(file, _L8("additionalname"), contact->getAdditionalname()->Des());
27:          }
28:          if (contact->getFamilyname())
29:          {
30:             PrinttToFile(file, _L8("familyname"), contact->getFamilyname()->Des());
31:          }
32:          if (contact->getSuffixname())
33:          {
34:             PrinttToFile(file, _L8("suffixname"), contact->getSuffixname()->Des());
35:          }
36:          if (contact->getOrganisation())
37:          {
38:             PrinttToFile(file, _L8("organisation"), contact->getOrganisation()->Des());
49:          }
40:          if (contact->getHomephone())
41:          {
42:             PrinttToFile(file, _L8("homephone"), contact->getHomephone()->Des());
43:          }
44:          if (contact->getWorkphone())
45:          {
46:             PrinttToFile(file, _L8("workphone"), contact->getWorkphone()->Des());
47:          }
48:          if (contact->getMobilephone())
49:          {
50:             PrinttToFile(file, _L8("mobilephone"), contact->getMobilephone()->Des());
51:          }
52:          if (contact->getFaxphone())
53:          {
54:             PrinttToFile(file, _L8("faxphone"), contact->getFaxphone()->Des());
55:          }
56:          if (contact->getOtherphone())
57:          {
58:             PrinttToFile(file, _L8("otherphone"), contact->getOtherphone()->Des());
59:          }
60:          if (contact->getHomeemail())
61:          {
62:             PrinttToFile(file, _L8("homeemail"), contact->getHomeemail()->Des());
63:          }
64:          if (contact->getWorkemail())
```

```
 65:          {
 66:             PrinttToFile(file, _L8("workemail"), contact->getWorkemail()->Des());
 67:          }
 68:          if (contact->getOtheremail())
 69:          {
 70:             PrinttToFile(file, _L8("otheremail"), contact->getOtheremail()->Des());
 71:          }
 72:          file.Write(_L8("\n"));
 73:       }
 74:       array->ResetAndDestroy();
 75:       delete array;
 76:    }
 77:    file.Flush();
 78:    file.Close();
 79:    if (manager)
 80:    {
 81:       delete manager;
 82:    }
 83: }
 84:
 85: class CSkContactsManager : public CBase
 86: {
 87:    public:
 88:       CSkContactsManager();
 89:       virtual ~CSkContactsManager();
 90:       virtual void ConstructL();
 91:       TInt Init( char16* aFilterString = NULL, char8* aFilterField = NULL);
 92:       RPointerArray<ContactData>* RetrieveNext( TInt32 aCount );
 93:    private:
 94:       const TFieldDef GetFieldType( const TDesC8& aFieldName );
 95:       const char * GetNameByContentType(const CContentType& aContentType);
 96:       ContactData* GetContactL(const TContactItemId &aContactId);
 97:    private:
 98:       CContactDatabase* iContactDatabase;
 99:       CContactIdArray* iContactIdArray;
100:       TInt iContactCursor;
101: };
102:
103: CSkContactsManager::CSkContactsManager()
104: {
105:    iContactIdArray = NULL;
106:    iFilterString = NULL;
107:    iSearchField = NULL;
108:    ConstructL();
109: }
110:
111: CSkContactsManager::~CSkContactsManager()
112: {
113:    if (iContactDatabase)
114:    {
115:       iContactDatabase->CloseTables();
116:       delete iContactDatabase;
117:    }
118:    if (iContactIdArray)
119:    {
120:       delete iContactIdArray;
121:    }
122: }
123:
124: void CSkContactsManager::ConstructL()
125: {
126:    iContactDatabase = CContactDatabase::OpenL();
127:    CContactItemViewDef* itemViewDef = CContactItemViewDef::NewL(EIncludeFields, EHiddenFields);
128:    itemViewDef->AddL(KUidContactFieldPrefixName);
129:    itemViewDef->AddL(KUidContactFieldGivenName);
130:    itemViewDef->AddL(KUidContactFieldAdditionalName);
131:    itemViewDef->AddL(KUidContactFieldFamilyName);
132:    itemViewDef->AddL(KUidContactFieldSuffixName);
133:    itemViewDef->AddL(KUidContactFieldCompanyName);
134:    itemViewDef->AddL(KUidContactFieldPhoneNumber);
135:    itemViewDef->AddL(KUidContactFieldFax);
136:    itemViewDef->AddL(KUidContactFieldEMail);
137:    CContactViewDef* viewDef = CContactViewDef::NewL(itemViewDef);
138:    iContactDatabase->SetViewDefinitionL(viewDef);
```

```
139:    iContactDatabase->SetDbViewContactType(KUidContactCard);
140: }
141:
142: TInt CSkContactsManager::Init()
143: {
144:    iFilterString = new char16[2];
145:    iFilterString[0] = L'\0';
146:    iSearchField = new char8[2];
147:    iSearchField[0] = '\0';
148:    if (iContactIdArray)
149:    {
150:       delete iContactIdArray;
151:       iContactIdArray = NULL;
152:    }
153:    CContactItemFieldDef* fields = new (ELeave) CContactItemFieldDef();
154:    const TFieldDef fieldDef  = GetFieldType(Str2Des(iSearchField));
155:    for (TInt i = 0; i < fieldDef.iCount; i++)
156:    {
157:       fields->AppendL(fieldDef.iTypes[i]);
158:    }
159:    iContactIdArray = iContactDatabase->FindLC(Str2Des(iFilterString), fields);
160:    CleanupStack::Pop();
161:    iContactCursor = 0;
162:    delete fields;
163:    return KErrNone;
164: }
165:
166: RPointerArray<ContactData>* CSkContactsManager::RetrieveNext(TInt32 aCount)
167: {
168:    RPointerArray<ContactData>* resArray = new RPointerArray<ContactData>;
169:    if (!iContactIdArray||iContactIdArray->Count()<=0||iContactCursor>=iContactIdArray->Count())
170:    {
171:       return NULL;
172:    }
173:    TInt index = 0;
174:    TInt last = iContactCursor + aCount;
175:    if (iContactIdArray->Count() < last)
176:    {
177:       last = iContactIdArray->Count();
178:    }
179:    for (TInt i = iContactCursor; i < last; i++)
180:    {
181:       TContactItemId itemId = (*iContactIdArray)[i];
182:       ContactData* contact = GetContactL(itemId);
183:       TInt err = resArray->Append(contact);
184:    }
185:    iContactCursor += aCount;
186:    return resArray;
187: }
188:
189: const TFieldDef CSkContactsManager::GetFieldType(const TDesC8& aFieldName)
190: {
191:    for (TInt i = 0; i < KTFieldNamesCount; i++)
192:    {
193:       TPtrC8 ptr ((const TUint8*) KFieldDefines[i].iName);
194:       if( ptr == aFieldName)
195:       {
196:          return KFieldDefines[i];
197:       }
198:    }
199:    return KFieldDefines[KDefaultFieldIndex];
200: }
201:
202: ContactData* CSkContactsManager::GetContactL(const TContactItemId& aContactId)
203: {
204:    ContactData* contact = new ContactData(aContactId);
205:    CContactItem* item = iContactDatabase->ReadContactLC(aContactId);
206:    CContactItemFieldSet& fieldSet = item->CardFields();
207:    TInt count = fieldSet.Count();
208:    for (TInt i = 0; i < count; i++)
209:    {
210:       CContactItemField& field = fieldSet[i];
211:       const CContentType& type = field.ContentType();
212:       const char * attrName = GetNameByContentType(type);
```

```
213:       if (attrName != NULL)
214:       {
215:          TPtrC text = field.TextStorage()->Text();
216:          contact->SetValue(Str2Des(attrName), text);
217:       }
218:    }
219:    CleanupStack::PopAndDestroy(item);
220:    return contact;
221: }
222:
223: const char * CSkContactsManager::GetNameByContentType(const CContentType& aContentType)
224: {
225:    for (TInt i = 0; i < KTFieldNamesCount; i++)
226:    {
227:       if (KFieldDefines[i].iCount == aContentType.FieldTypeCount())
228:       {
229:          TInt matches = 0;
230:          for (TInt y = 0; y < KFieldDefines[i].iCount; y++)
231:          {
232:             if (aContentType.ContainsFieldType(KFieldDefines[i].iTypes[y]))
233:             {
234:                matches++;
235:             }
236:          }
237:          if (matches == KFieldDefines[i].iCount)
238:          {
239:             return KFieldDefines[i].iName;
240:          }
241:       }
242:    }
243:    return NULL;
244: }
```