Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Hribar

Razvoj medplatformne knjižnice za uporabo v mobilnih in spletnih aplikacijah

DIPLOMSKO DELO
UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

doc. dr. Dejan Lavbič
MENTOR

Ljubljana, 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

S prisotnostjo velikega števila različnih platform in izvajalnih okolij je razvoj in testiranje današnjih aplikacij zelo otežen. Če želimo izdelati, med uporabniki, široko sprejeto aplikacijo, mora le ta biti prisotna na vseh večjih mobilnih platformah in na namiznih računalnikih. Zaradi tega se pojavlja potreba po sočasnem razvoju, ki pa seveda prinaša tudi številne težave. V okviru diplomske naloge naj študent pregleda metode za medplatformni razvoj, ki bi omogočal čim večjo prenosljvost aplikacije, skladnost z ostalimi domorodnimi aplikacijami, hkrati pa naj bo stroškovno čim bolj učinkovita. V okviru diplomske naloge je najprej potrebno raziskati možne pristope k medplatformnem razvoju, poiskati metodo, ki ustreza naštetim omejitvam in prikazati postopek za uspešno uporabo na platformah iOS, Android, Windows Phone in v spletni aplikaciji.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani izjavljam, da sem avtor dela, da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali drugem visokošolskem zavodu, razen v primerih kjer so navedeni viri.
S svojim podpisom zagotavljam, da:

- sem delo izdelal samostojno pod mentorstvom doc. dr. Dejana Lavbiča,
- so elektronska oblika dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko in
- soglašam z javno objavo elektronske oblike dela v zbirki "Dela FRI".

— Miha Hribar, Ljubljana, julij 2014.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Hribar

Razvoj medplatformne knjižnice za uporabo v mobilnih in spletnih aplikacijah

POVZETEK

Razvoj aplikacij za več različnih platform je težaven. Odpira veliko možnosti za napake, oteži testiranje in odpravljanje napak ter skoraj onemogoči sočasno nadgrajevanje aplikacij. Rezultat so dolgotrajni razvojni cikli in počasno dodajanje funkcionalnosti, kar v današnjem svetu zagonskih podjetij ni zaželeno.

Kljub razlikam med posameznimi platformami je ponavadi veliko kode z identično funkcionalnostjo, ki jo je potrebno razviti za vsako platformo posebej. Velikokrat je v podjetju za vsako od platform zadolžen drug razvijalec, še bolj pogosto pa razvoj na različnih platformah ne poteka sočasno. Rešitev iz te zagate je razvoj medplatformne knjižnice.

Cilj diplomske naloge je razvoj knjižnice za specifikacijo RFC 5545, ki omogoča generiranje ponavljajočih koledarskih dogodkov in jo je možno uporabiti v spletni, iOS, Android in Windows Phone aplikaciji. Pregledali bomo različne možne pristope, navedli prednosti in slabosti ter na koncu izbrali najprimernejšo rešitev za implementacijo knjižnice.

Kljucne besede: medplatformna knjižnica, iOS, Android, Windows Phone, JavaScript, Emscripten, LLVM

University of Ljubljana
Faculty of Computer and Information Science

Miha Hribar

Developing a cross platform library for use in mobile and web applications

ABSTRACT

Developing applications for different platforms is complicated. It opens a lot of avenues for mistakes, complicates testing and bugfixing, while almost completely destroys any chance of simultaneous application upgrade. The result of this are prolonged development cycles and slow feature creep, which in todays "startup" world is not an option.

Despite the differences between platforms, they most likely share a lot of functionality which has to be developed for each platform. Most of the time each platform is handled by a different developer and usually not simultaneously with other applications. The solution to this problem is to develop a cross-platform library.

The goal of the thesis is to develop a library for the RFC5545 specification, which enables applications to schedule and display recurring events. The library will then be used in a web, iOS, Android and Windows Phone application. We will outline different approaches to writing the shared library, list the pros and cons and in the end decide on the best approach.

Key words: cross platform library, iOS, Android, Windows Phone, JavaScript, Emscripten, LLVM

ZAHVALA

Rad bi se zahvalil mentorju doc. dr. Dejanu Lavbiču za strokovno svetovanje in predvsem za potrpežljivost pri nastajanju diplomskega dela. Hvala družini za vso podporo in finančno pomoč pri študiju. Hvala vsem ostalim, ki ste mi stali ob strani.

In seveda hvala Petri, ker mi pustiš, da sem to, kar sem.

— Miha Hribar, Ljubljana, julij 2014.

KAZALO

Seznam uporabljenih kratic

ARC angl. *Automatic Reference Counting*; avtomatično štetje referenc za sproščanje pomnilnika, ki v jezikih Objective-C in Swift olajša razvoj aplikacij.

ARM angl. *Acorn RISC Machine, Advanced RISC Machine*; 32 in 64-bitna procesorska arhitektura RISC, največkrat uporabljena v vgrajenih sistemih zaradi nizke električne porabe.

ART angl. *Android Runtime*; android pogon, ki bo sčasoma zamenjal trenutni pogon Dalvik; nudi veliko izboljšav v hitrosti in stabilnosti aplikacij.

CI angl. *Continuous integration*; zvezna integracija.

CLI angl. *Common Language Infrastructure*; specifikacija, ki omogoča prevod različnih jezikov v enotno vmesno kodo; uporabljeno v ogrodjih .NET, Mono in Portable.NET.

CSS angl. *Cascading Style Sheets*; stilna predloga, v kateri so zapisana pravila za obliko spletne strani.

GPL angl. *GNU General Public License*; ena izmed najbolj razširjenih licenc odprtokodnih projektov, ki omogoča komercialno uporabo, redistribucijo in spremembe odprtokodnih projektov, pod pogojem, da izpeljano delo (angl. *derived work*) uporablja isto licenco[1].

HTML5 angl. *HyperText Markup Language*; 5. revizija označevalnega jezika(angl. *markup language*) namenjenega stukturiranju in opisovanju vsebine na svetovnem spletu.

IDE angl. *Integrated Development Environment*; integrirano razvojno orodje za lažje in hitrejše razvijanje programov.

JIT angl. *Just-in-time compilation*; tehnika, znana tudi pod imenom dinamično prevajanje, kjer se prevod ne zgodi pred zagonom programa, ampak ravno pred.

JNI angl. *Java Native Interface*; programsko ogrodje, ki omogoča kodi, napisani v jeziku Java, klicanje domorodnih aplikacij in knjižnic.

JVM angl. *Java virtual machine*; virtualen pogon, ki je zmožen poganjati Java zlogovno kodo.

LGPL angl. *GNU Lesser General Public License*; odprtokodna licenca, največkrat uporabljena za uporabo programskih knjižnic, saj omogoča razvoj knjižnic pod drugimi licencami, če te samo uporabljajo in ne spreminjajo knjižnic izdanih pod LGPL licenco[1].

LLVM angl. *Low Level Virtual Machine*; skupek prevajalniških infrastrukturnih knjižnic, ki omogočajo optimizacijo programov napisanih v različnih programskih jezikih.

NDK angl. *Native Development Kit*; ogrodje, ki na platformi Android omogoča pripravo knjižnic za vključitev v domorodne aplikacije.

PHP angl. *PHP: Hypertext Preprocessor*; strežniški skriptni jezik.

QML angl. *Qt Meta Language ali Qt Modeling Language*; označevalni jezik namenjen gradnji uporabniških vmesnikov v ogrodju Qt.

RFC angl. *Request For Comment*; publikacija IEFT (Internet Engineering Task Force) v kateri so predstavljeni standardi.

SDK angl. *Software development kit*; paket za razvoj programske opreme.

STL angl. *Standard Template Library*; C++ knjižnica, ki vsebuje razne programske konstrukte in algoritme.

SWOT angl. *Strengths, Weaknesses, Opportunities, and Threats*; analiza prednosti, slabosti, priložnosti in nevarnosti.

TDD angl. *Test-driven development*; testno usmerjeni razvoj.

XML angl. *Extensible Markup Language*; razširljivi označevalni jezik; format podatkov za izmenjavo strukturiranih dokumentov v spletu.

YARV angl. *Yet another Ruby VM*; eden izmed virtualnih pogonov za jezik Ruby.

1 Uvod

Dandanes uporabljamo več različnih naprav sočasno. V lasti imamo najverjetneje prenosni računalnik, pametni telefon in po možnosti še tablični računalnik. Ko najdemo aplikacijo, ki nam je všeč, od te pričakujemo brezhibno delovanje na vseh naših napravah.

Z razvijalskega vidika je taka pričakovanja uporabnikov zelo težko uresničiti, še posebej če pri razvoju sodeluje zelo malo ljudi. Dodatno se stvari zapletejo, če vse te naprave uporabljajo različne operacijske sisteme. Tako imamo lahko Windows prenosnik, Android[1] telefon in Apple tablico. Zahteven uporabnik na tem mestu pričakuje, da je izbrana aplikacija na voljo na vse naštetih platformah in da na vseh platformah deluje identično.

Za razvijalca smo ravnokar opisali nočno moro. Da zadovolji potrebe uporabnikov, je primoran razviti isto aplikacijo za vsako platformo posebej. Četudi omejimo razvoj

[1] Operacijski sistem, razvit pri podjetju Google, namenjen uporabi na mobilnih napravah.

Operacijski sistem	2013 število	2013 delež (%)	2012 število	2012 delež (%)
Android	758.719,9	78,4	451.621,0	66,4
iOS	150.785,9	15,6	130.133,2	19,1
Microsoft	30.842,9	3,2	16.940,7	2,5
BlackBerry	18.605,9	1,9	34.210,3	5,0
Ostali	8.821,2	0,9	47.203,0	6,9
Skupaj	967.775,8	100	680.108,2	100

Tabela 1.1 Razpredelnica svetovne prodaje pametnih telefonov v letih 2013 in 2012 glede na mobilni operacijski sistem (v tisočih). Opazimo lahko hud padec prodaje naprav BlackBerry. [52].

na najbolj razširjene platforme (iOS[2], Android in Windows Phone[3]), s čimer pokrijemo več kot 98% vseh mobilnih naprav, kot je razvidno iz tabele 1.1, smo ravnokar našteli tri povsem različne tehnologije, tri različne jezike in s tem tri priložnosti za povsem različne težave pri implementaciji naše aplikacije. Veliko truda in energije je potrebno, da so te aplikacije poenotene in da skladno sledijo razvoju novih funkcionalnosti.

Izkušen razvijalec bo pri predstavitvi problema takoj pomislil na medplatformni razvoj, ki si ga bomo ogledali v drugem poglavju. Omenili bomo tako imenovane "celovite" metode, kot so Qt[2] in Xamarin[3], "hibridne", kot sta recimo PhoneGap[4] in Appcelerator Titanium[5], ter "deljene" metode npr. Lua[6], Haxe[7] in C++[8]. Vsaka izmed omenjenih metod ima svoje prednosti in slabosti, izbor primerne pa je povsem odvisen od problema, ki ga želimo rešiti.

Tretje poglavje bomo začeli s pregledom standarda RFC 5545[43], zakaj ga sploh potrebujemo in katere probleme nam pomaga reševati. Nato bomo pregledali predhodno opisane metode, si ogledali, zakaj smo se odločili za razvoj knjižnice s pomočjo jezika C++, ter jo tudi zgradili. Predstavili bomo glavne razrede in metode naše knjižnice ter predstavili nekaj primerov uporabe.

V četrtem poglavju bomo pokazali, kako lahko knjižnico s pomočjo jezikovnih ovojev (angl. *wrapper*) uspešno uporabimo v jezikih Objective-C (iOS), Java (Android) in C# (Windows Phone). Predstavili bomo tudi način, kako lahko C++ knjižnico prevedemo v jezik JavaScript s pomočjo orodja Emscripten[9], in knjižnico uporabili tudi v spletni

[2]Mobilni operacijski sistem, razvit pri podjetju Apple. Najdemo ga na napravah iPhone, iPad in iPod.

[3]Mobilni operacijski sistem, razvit pri podjetju Microsoft.

aplikaciji.

V zaključku bomo pretehtali, kako primeren je razvoj medplatformne knjižnice na predstavljen način in če se morda obetajo novi načini, ki bi razvijalce rešili iz podobnih zagat.

2 Pregled metod medplatformnega razvoja

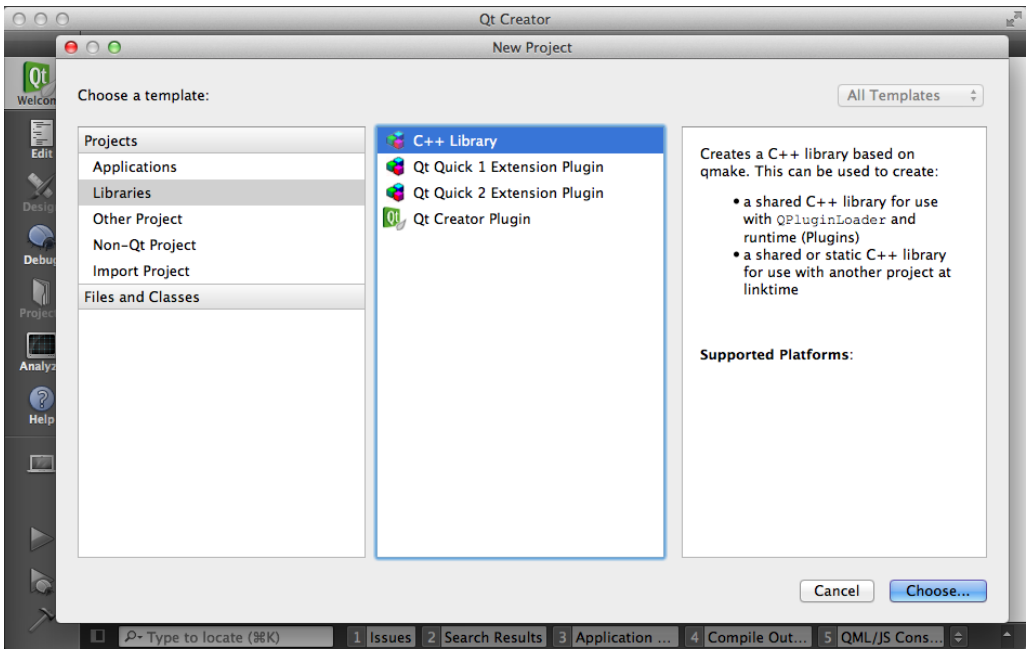
Preden postavimo omejitve razvoja naše aplikacije, si poglejmo različne metode medplatformnega razvoja in v katerih primerih jih je pametno uporabiti. Kot je pričakovati, jih je kar nekaj. Razdelili jih bomo v skupine celovitih, hibridnih in deljenih metod.

2.1 Celovite metode

Celovita metoda za razvoj uporablja ogrodje, s pomočjo katerega aplikacijo pripravimo za različne platforme. Velika večina tako napisane izvorne kode je uporabljena na vseh ciljnih platformah, za kar poskrbi ogrodje. Rezultat te metode je domorodna aplikacija (angl. *native application*), ki jo je možno objaviti v trgovinah posameznih platform in pri tem ne kršijo (ponavadi) strogih pravil.

2.1.1 Qt

Qt[2] je ogrodje za grafično programiranje za več platform s pomočjo jezika C++ in QML. Omogoča nam sočasni razvoj za platforme Mac OSX, Linux, Windows, Android in iOS. Podpira tudi uporabo HTML5 namesto QML, kar pomeni, da spletni razvijalci



Slika 2.1 Zaslonska slika orodja Qt Creator.

lahko uporabijo že obstoječe znanje, zato učenje novega jezika ni potrebno. Za olajšanje razvoja so razvili IDE Qt Creator (slika 2.1).

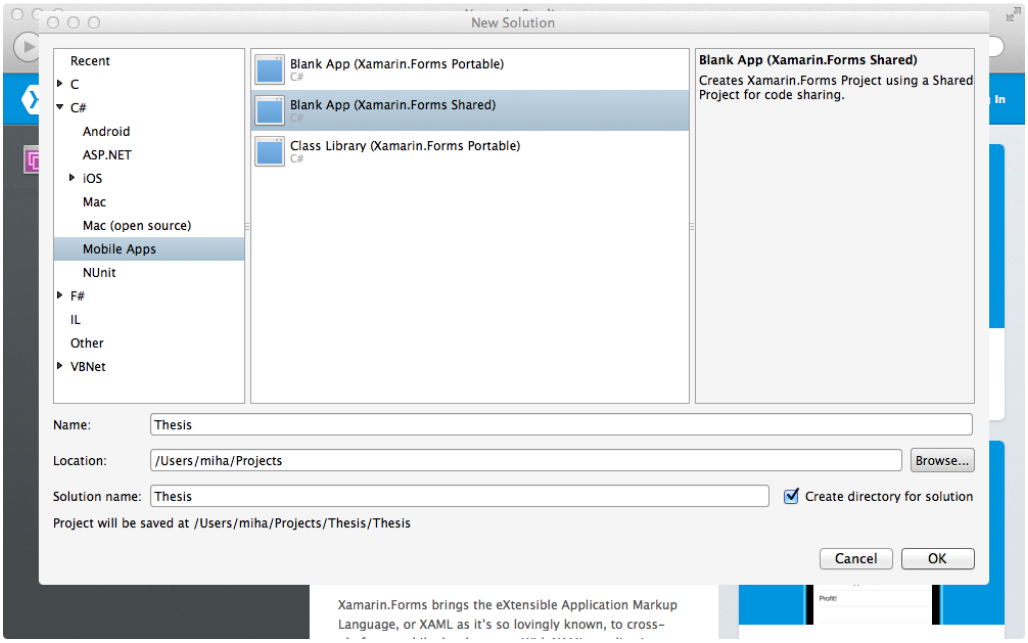
Projekt Qt je povsem odprtokoden in dovoljuje uporabo v skladu z licencama GPL v3[10] in LGPL v2.1[11], a če želite orodje uporabiti za razvoj mobilne aplikacije, boste morali za to odšteti 149$ mesečno.

Projekt so vrsto let uspešno razvijali v podjetju Nokia, kjer so ga uporabili kot glavno orodje za razvoj aplikacij na platformi Symbian. Ko je Nokio pred časom kupil Microsoft, je projekt prevzela novonastala organizacija Qt Project, ki projekt vodi še danes.

Qt je še posebej privlačen zaradi podpore namiznih platform, kot so Windows, Mac OSX in Linux. Odlikuje ga tudi zagreta skupnost razvijalcev.

Ogrodje Qt je primerno za izdelavo aplikacij, ki vključujejo kompleksne algoritme, za katere bi porabili preveč časa pri prilagajanju za različne platforme. Lep primer tega sta aplikaciji Mathematica[12] in multimedijski predvajalnik VLC[13].

Glavne slabosti Qt so neskladnost z izgledom ostalih aplikacij na mobilnih platformah, plačljiva licenca za razvoj mobilnih aplikacij ter končna velikost samih programov. Manjka tudi napovedana podpora za platformo Windows Phone.



Slika 2.2 Zaslonska slika orodja Xamarin Studio.

2.1.2 Xamarin

Xamarin[3] je ogrodje za sočasen razvoj aplikacij za platforme iOS, Android, Mac in Windows v jeziku C#. Izhaja iz projekta Mono[14], ki omogoča uporabo ogrodja .NET[15] na različnih platformah. Ogrodje omogoča razvoj aplikacij, katerih izgled je skladen z ostalimi domorodnimi aplikacijami.

Ogrodje odlikuje integrirano razvojno okolje (IDE) Xamarin Studio (slika 2.2), ki razvoj aplikacij znatno olajša. Omogoča testiranje tako v emulatorju/simulatorju kot tudi na samih napravah.

Xamarin je primeren za izdelavo aplikacij za platforme, kjer je ključnega pomena končna grafična skladnost z ostalimi domorodnimi aplikacijami. Kot primer si lahko ogledamo aplikacijo za poslušanje glasbe Rdio[16], ki je na voljo za iOS, Android in Windows Phone.

Glavna slabost ogrodja Xamarin je cena, saj se paketi začnejo šele pri 299$/mesec za vsakega razvijalca in vsako platformo posebej. Za majhno ekipo je lahko taka začetna cena enostavno previsoka. Vprašljiva je tudi hitrost dodajanja funkcionalnosti posameznih platform, ko se te nadgradijo. Določeno tveganje predstavlja tudi muhavost posameznih platform pri omejitvah uporabe tega ogrodja, sploh če nadgradnja povzroči

nedelovanje takih aplikacij.

2.1.3 Adobe Air

Adobe Air[17] je brezplačno ogrodje, ki omogoča zagon iste aplikacije na platformah iOS, Android, Mac, Windows in Linux, zagon aplikacije pa je možen tudi iz spletnega brskalnika. Čeprav za razvoj namiznih aplikacij omogoča uporabo HTML in Javascript, je razvoj mobilnih aplikacij omejen na uporabo jezika ActionScript. V času pisanja diplomske naloge ogrodje ne omogoča zagona na platformi Windows Phone, vendar so razvijalci podporo že napovedali.

Izbor orodja je še posebej uporaben za aplikacije, v katerih uporabniški vmesnik ni potrebno prilagajati posamezni platformi. Ravno zaradi tega je orodje priljubljeno med razvijalci iger, ko je na primer Angry Birds[18].

Kot glavno slabost ogrodja Adobe Air bi navedel upadanje zanimanja za orodje Flash. Špekuliramo lahko tudi o planih podjetja Adobe, saj so pred kratkim kupili podjetje Nitobi, ki je avtor ogrodja PhoneGap (katerega si bomo ogledali v nadaljevanju). Uporaba tudi ni primerna za razvoj klasičnih mobilnih aplikacij, saj je prilagajanje domorodnim aplikacijam precej zahtevno še posebej, kadar na platformi pride do posodobitve izgleda.

2.2 Hibridne metode

Hibridna metoda za razvoj aplikacij uporablja spletne tehnologije v sožitju z domorodno kodo za posamezno platformo (t.i. premostitvena tehnika), ki omogoča dostop do glavnih funkcij naprav (kot so kamera, pospeškomer in podobno). Tako kot pri celovitih metodah je tudi tu rezultat domorodna aplikacija, ki jo je možno objaviti v trgovinah posameznih platform.

2.2.1 Apache Cordova / PhoneGap

Ogrodje Apache Cordova[19] je odprtokodni projekt, ki omogoča objavo spletnih aplikacij kot domorodne[42]. V času pisanja diplomske naloge ogrodje podpira iOS, Android, Windows Phone, Blackberry, Palm WebOS, Bada in Symbian. Na vseh omenjenih platformah nam ogrodje Apache Cordova omogoča dostop do funkcij naprave, ko so naprimer kamera in pospeškomer. Isto aplikacijo je možno zagnati tudi v spletnem brskalniku, a je za to potrebno nekaj dodatnega dela. Ogrodje Cordova je izdano pod odprtokodno licenco Apache 2.0[20].

Projekt PhoneGap[4] je dejansko samo ena od distribucij projekta Apache Cordova, ki poleg vseh obstoječih funkcionalnosti ponuja tudi razne storitve, na katerih delajo v podjetju Adobe. Vključuje tudi priročen program za ukazno vrstico, ki nam olajša delo s PhoneGap projektom (primer 2.1)

Primer 2.1 Primer uporabe programa phonegap za ukazno vrstico. Zadnja vrstica zažene pravkar ustvarjeno prazno aplikacijo na Android napravi ali emulatorju.

```
1 $ phonegap create thesis
2 $ cd thesis
3 $ phonegap run android
```

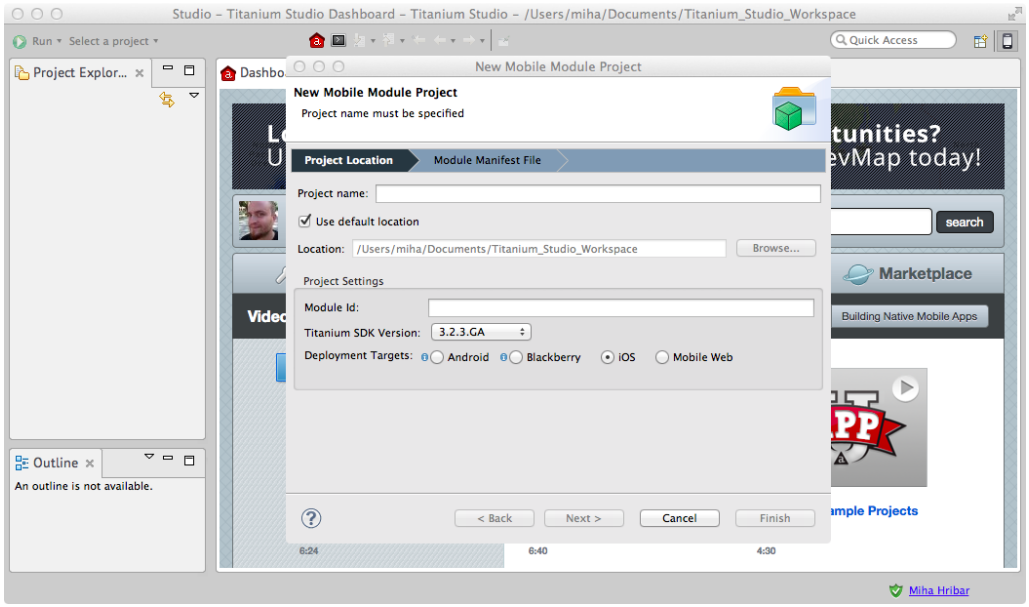
Za razvoj aplikacij razvijalci lahko uporabljajo spletne tehnologije HTML, CSS in JavaScript. S pomočjo ogrodij jQuery Mobile[21] in Sencha Touch[22] je možno izdelati aplikacije, katerih izgled je zelo lep približek ostalim aplikacijam na izbrani platformi. Če naletimo na funkcijo naprave, do katere nimamo dostopa, ali ugotovimo, da je JavaScript za določene naloge premalo učinkovit, lahko preprosto spišemo lasten vtičnik (angl. *plugin*), ki služi kot most med kodo, napisano v jeziku JavaScript, in domorodno kodo.

Glavna prednost ogrodja Apache Cordova in predvsem distribucije PhoneGap je izredno velika nezahtevnost ogrodja. Priporoča se predvsem za izdelavo prototipnih aplikacij, saj nam omogoča hiter razvoj in iteracijo.

Glavna slabost tega pristopa tiči v performanci in odzivnosti aplikacije, saj le-ta za prikazovanje izkorišča vgrajeno spletno okno. Trenutno je težko izdelati aplikacije, ki so grafično zahtevnejše, kar predstavlja še toliko bolj pereč problem na napravah s slabšimi karakteristikami. Da se aplikacija po izgledu ne bi ločila od domorodnih aplikacij, je potrebno vložiti kar nekaj dela, na koncu pa bo izurjen uporabnik najbrž vseeno opazil, da je aplikacija malce drugačna. Problem predstavlja tudi zamik podpore novim stilom grafičnih elementov, tako kot se je to zgodilo pri prehodu z iOS6 na iOS7.

2.2.2 Appcelerator Titanium

Ogrodje Titanium[5] nam omogoča izdelavo aplikacij za več platform hkrati s pomočjo JavaScript okolja, ki služi kot abstrakcijska plast med našo aplikacijo in domorodno kodo. Aplikacijo gradimo s pomočjo jezika JavaScript, ki se med uporabo aplikacije izvaja s pomočjo pogona V8[23] (Android), JavaScriptCore (iOS)[48] ali vgrajenega JavaScript okolja (če aplikacijo poganjamo v brskalniku). Za pravilen izgled skrbijo namestniški



Slika 2.3 Zaslonska slika orodja IDE Titanium Studio.

elementi, ki uporabljajo domorodne grafične elemente, kar pomeni, da aplikacije po izgledu ne moremo ločiti od ostalih domorodnih aplikacij. V času pisanja diplomske naloge ogrodje podpira iOS, Android, Blackberry, Tizen in spletne aplikacije. Tako kot PhoneGap je tudi Titanium na voljo pod odprtokodno licenco Apache 2.0. Za olajšanje razvoja so pri podjetju Appceletor razvili IDE Titanium Studio (slika 2.3), ki je prav tako na voljo brezplačno.

Glavna prednost ogrodja Titanium ni t.i. način "piši enkrat, uporabljaj povsod", temveč da lahko celotno aplikacijo izdelamo v enem jeziku - JavaScript-u. Le redko se bomo srečali z domorodno kodo, saj ogrodje nudi široko paleto knjižnic.

Tako kot PhoneGap je tudi Titanium še posebej uporaben pri razvoju prototipov aplikacij, kjer je cilj hiter razvoj in predstavitev aplikacije čim večjemu krogu uporabnikov.

Glavna slabost ogrodja je počasno dodajanje novih platform zaradi obsežnosti dela, ki ga tak podvig zahteva. Določene knjižnice za delo z domorodnimi elementi tudi niso najbolj performančne, manjka pa tudi napovedana podpora platformi Windows Phone.

2.3 Deljene metode

Deljena metoda za razvoj aplikacij omogoča uporabo dela aplikacijske kode na vseh platformah, za katere razvijamo. To lahko naredimo s pomočjo vgradnega skriptnega jezika (Lua), s pomočjo prevajanja iz izbranega programskega jezika v domorodnega (Haxe, XMLVM, Emscripten) ali pa z uporabo programskih jezikov C++ ali JavaScript in jezikovnih ovojev, s katerimi pripravimo knjižnico za vgradnjo v druge platforme.

2.3.1 Lua

Lua[6] je preprost vgradni skriptni jezik, ki ga odlikuje hitrost izvajanja in procesorska nezahtevnost. Vgradimo ga lahko v platforme Android, iOS, Symbian in Windows Phone, z nekaj potrpljenja pa lahko isto kodo zaženemo tudi v spletni aplikaciji. Lua je na voljo pod odprtokodno licenco MIT[24].

Ker gre za skriptni jezik, se znajdemo v zanimiv situaciji, kjer združujemo prevajane jezike z interpretiranimi jeziki. Velika prednost tega je hitro odzivanje na napake pri razvoju, saj razvijalcu ni potrebno čakati na prevod kode. Odpira tudi možnost posodobitve vgrajene knjižnice brez posodobitve celotne aplikacije.

Primer 2.2 Primer metode v skriptnem jeziku Lua, ki datum iz oblike 2014-07-14 razbije na leto, mesec in dan.

```
1  function get_date_parts(date_str)
2    _,_,y,m,d=string.find(date_str, "(%d+)-(%d+)-(%d+)")
3    return tonumber(y),tonumber(m),tonumber(d)
4  end
```

Čeprav je jezik Lua preprost za uporabo, se izkaže, da za kompleksnejše knjižnice ni primeren. Manjka podpora Unicode[1], boljša podpora rokovanju z napakami, boljša podpora starejšim verzijam in vgrajen razhroščevalnik (angl. *debugger*).

2.3.2 Haxe

Haxe[7] zase pravi, da je večplatformski programski jezik. Razvijalec lahko svojo aplikacijo napiše v jeziku Haxe, nato pa jo s pomočjo prevajalnika prevede v izvorno kodo jezikov PHP, ActionScript, Neko, JavaScript, C++, C# in Java. Nudi tudi dodatne vmesnike za dostop do specifičnih metod ciljnega jezika. Haxe prevajalnik je na voljo pod licenco GPL, haxe knjižnice, ki jih potrebujemo za razvoj aplikacij, pa so na voljo pod licenco MIT.

[1] Standard za poenoteno tekstovno enkodiranje.

Jezik se uporablja predvsem pri razvoju iger, kjer naj bi v prihodnosti zamenjal jezik ActionScript, ki ga uporablja orodje Flash.

Glavna slabost uporaba rešitve Haxe je majhna razvijalska skupnost. V primerjavi z ostalimi rešitvami je ta kar v manjšini. V zadnjem času sicer pridobiva nekaj zagona, a je trenutno vse premalo knjižnic, ki bi bile razvite za to platformo.

2.3.3 XMLVM

XMLVM[25] spada v isti razred kot Haxe - tako imenovanih prevajalcev iz enega jezika v drugega (ang. cross-compilers) - a se XMLVM tega loti na drugačen način. Medtem ko Haxe prevaja na nivoju izvorne kode, XMLVM to počne na nivoju zlogovne kode (ang. byte code). Izvorna koda je lahko napisana za navidezne stroje (ang. virtual machine) JVM, .NET CLI ali Ruby YARV, medtem ko je rezultat delujoč program za JVM, .NET CLI, Javascript, Python, Objective-C in C++. Ogrodje je na voljo pod odprtokodno licenco LGPL.

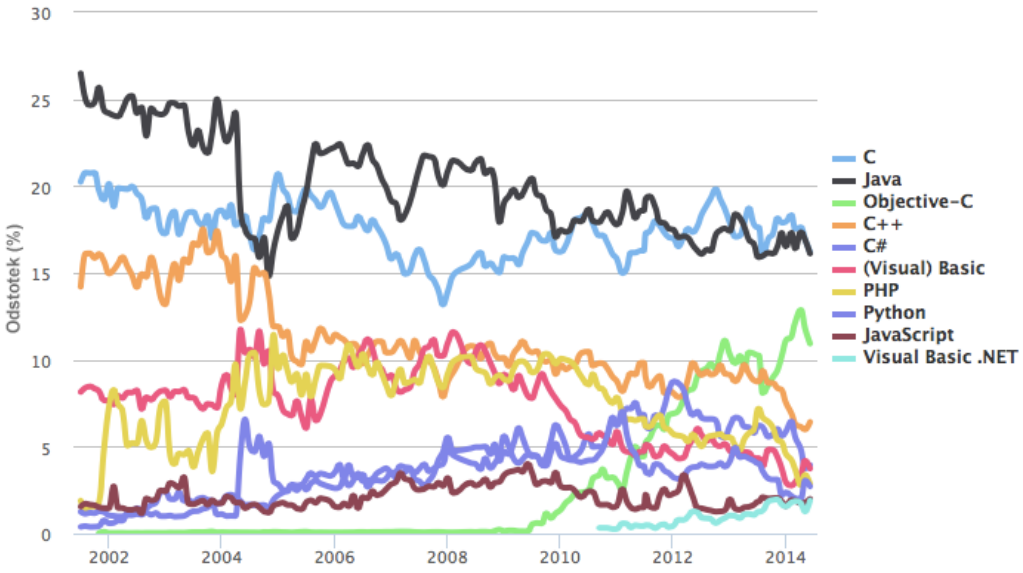
Projekt je bil zastavljen zelo ambiciozno, a vse kaže, da je šlo le za akademsko raziskavo. Od zadnje posodobitve izvorne kode je namreč minilo že več kot leto dni. Kljub temu se mi je projekt zdel zanimiv in ga je bilo vredno izpostaviti.

2.3.4 C++ in Emscripten

V kolikor nobena od naštetih možnosti ne zadošča našim potrebam, vseeno pa bi želeli imeti deljeno knjižnico, obstaja še ena možnost: uporaba jezika C++[8] in projekta Emscripten[9].

C++ je eden izmed najbolj razširjenih programskih jezikov. V času pisanja diplomske naloge zaseda četrto mesto na lestvici najbolj popularnih jezikov 2.4, pred njim so samo jeziki C, Java in Objective-C. Uporablja se ga v raznolikih projektih - od prevajalnikov, strežnikov do video igric.

Emscripten je projekt Mozillinih laboratorijev, ki omogoča prevajanje iz LLVM zlogovne kode v skriptni jezik JavaScript. LLVM si lahko predstavljamo kot vmesni sloj med izvorno (C, C++, Objective-C, Java, C#) in strojno kodo, ki poskrbi za visoko optimizacijo vmesne kode, to pa lahko potem prevedemo v ustrezen nabor ukazov za posamezne procesorje (ARM, x86 itd.). Emscripten tako predstavlja zadnjo fazo prevajalnika, le da vmesne kode iz LLVM ne prevede v ukaze specifičnega procesorja, ampak nazaj v jezik JavaScript. To pomeni, da lahko (z določenimi omejitvami) prevedemo skoraj vsak



Slika 2.4 Tiobe programming comunity index[26].

program v JavaScript in ga zaženemo v brskalniku. Celo grafično zahtevne aplikacije[2] niso problematične, saj Emscripten za prevod v JavaScript uporablja asm.js[27], kar je podmnožica jezika JavaScript, ki jo JavaScript pogoni znajo izredno dobro optimizirati[3].

2.3.5 JavaScript

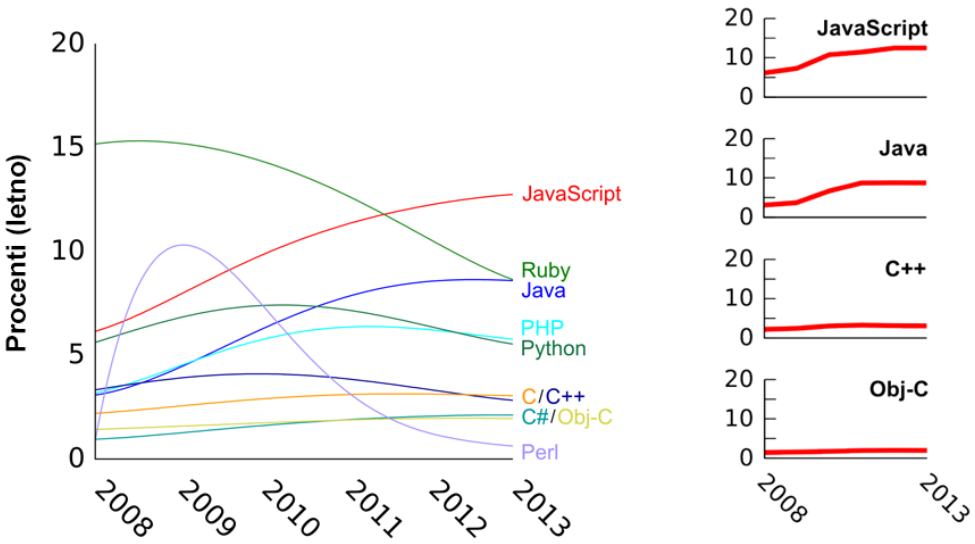
Namesto prevajanja v jezik JavaScript s pomočjo ogrodja Emscripten bi lahko celotno knjižnico napisali kar v jeziku JavaScript. V zadnjih nekaj letih je jezik JavaScript doživel izredno hitro rast tako v popularnosti kot tudi v zmogljivosti in funkcionalnostih, kar je tudi razvidno iz slike 2.5.

Pri vključitvi v svojo aplikacijo moramo biti malce bolj iznajdljivi. iOS je z verzijo 7 dodal knjižnico JavaScriptCore, ki omogoča mešanje domorodne in JavaScript kode.

Android je malce bolj problematičen, saj njegov SDK v času pisanja diplomske naloge ne nudi direktne implementacije. Zaradi tega smo primorani vključiti JavaScript pogon, kot so Rhino, V8 in podobni. V kolikor je pogon napisan v jeziku Java, je integracija preprosta, če pa izberemo pogon v drugem jeziku, mora za Android obstajati paket za

[2]Skupina Mozillinih inženirjev je grafično ogrodje Unreal v štirih dneh posodobilo do te mere, da je lahko s pomočjo orodja Emscripten grafično zahtevna aplikacija brezhibno delovala v brskalniku[50].

[3]S pomočjo asm.js so v podjetju Mozilla uspeli doseči le enkrat počasnejše izvajanje od domorodne kode, kar je izjemen dosežek.[47]



Slika 2.5 Jeziki novih projektov na spletni strani github.com, povzeto po[41]

uvoz.

Windows Phone prav tako kot Android ne nudi JavaScript implementacije, ki bi jo lahko uporabljali skupaj z domorodno kodo. Uporabimo lahko pogon V8 in knjižnico JavaScriptNet[28].

Ker je knjižnica napisana v jeziku JavaScript, jo je možno preprosto vgraditi v spletno aplikacijo. Pri tem se moramo držati le delov JavaScripta, ki so enotni na vseh platformah (ECMAScript specifikacija[29]).

Težji del je vgraditev pogona JavaScript na posamezno platformo. Dokler ne bo na voljo domorodnih integracij, se bomo težko zanesli na brezhibno delovanje pri nadgradnji operacijskega sistema. Kljub temu je ideja zelo zanimiva in vredna nadaljnje raziskave.

3 Razvoj knjižnice

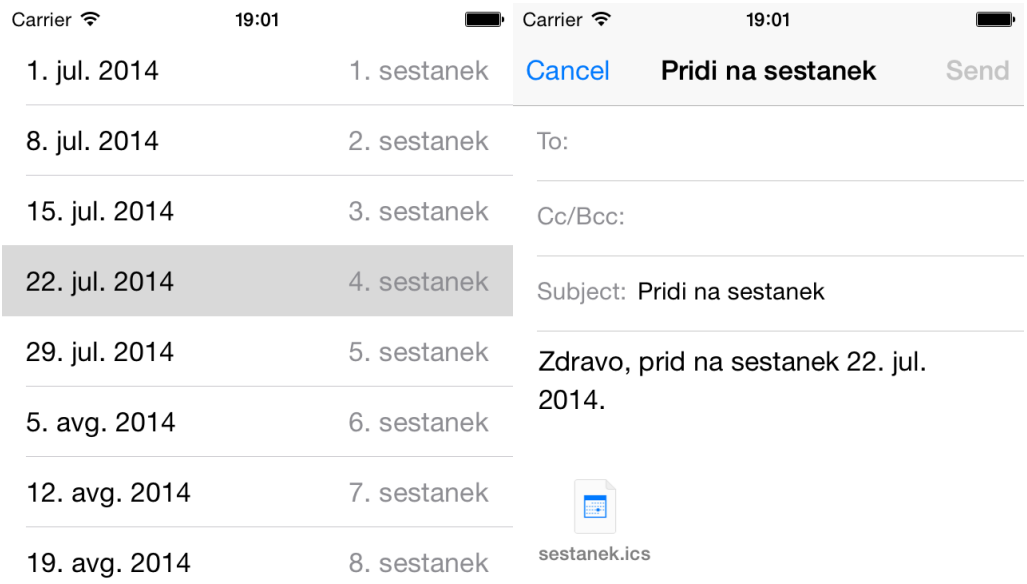
3.1 Uvod

Izbor primerne metode si bomo pogledali na primeru razvoja koledarske aplikacije, ki bo tekla na platformah iOS, Android, Windows Phone in tudi v spletni aplikaciji. V aplikaciji bo uporabnik lahko dodal koledarski dogodek, ki mu bo možno nastaviti različne frekvence ponavljanj, intervale in izjeme. Aplikacija bo nato za določen datumski interval prikazala vnešeni dogodek v preprostem seznamu (slika 3.1).

Vnešeni dogodek bo uporabnik aplikacije lahko delil z ostalimi preko e-poštnega sporočila, ki bo vseboval format iCalendar, opisan v specifikaciji RFC 5545[43], ki si ga bomo podrobneje ogledali v naslednjem poglavju.

3.2 Predstavitev specifikacije RFC 5545

Specifikacija RFC 5545[43] opisuje format iCalendar, ki se uporablja za dogovarjanje o urnikih sestankov in umeščanju na koledar. Gre za tekstovni format, katerega prepoznavna končnica datotek je `.ics`, in je dandanes v uporabi v vseh večjih koledarskih aplikacijah,



Slika 3.1 Zaslonska slika dveh delov preproste koledarske aplikacije.

kot so Google Calendar, Apple Calendar in ostale. Omogoča vse, kar potrebujemo za opisovanje dogodkov, med drugim:

- Datum začetka in konca dogodka
- Nastavitve opozoril
- Opis in priponke
- Seznam povabljenih ljudi
- Pravila za ponavljanje dogodka

Zaradi obsežnosti specifikacije se bomo v diplomski nalogi osredotočili zgolj na pravila za ponavljanje dogodkov[44]. Če primer 3.1 prevedemo v človeku prijazno obliko, bi to pomenilo "vsako nedeljo v januarju, ob 8:30 zjutraj in 9:30 zjutraj, vsako drugo leto".

Primer 3.1 Primer uporabe pravila RRULE spefikiacije RFC 5545.

```
1  FREQ=YEARLY;INTERVAL=2;BYMONTH=1;BYDAY=SU;BYHOUR=8,9;BYMINUTE=30
```

Vsak del pravila sestavlja par, sestavljen iz imena pravila in vrednosti. Vsak par je med seboj ločen z podpičjem (;), vsako pravilo pa je lahko določeno le enkrat. Vrstni

red načeloma ni pomemben, a zaradi združljivosti s starejšimi specifikacijami mora biti pravilo `FREQ` vedno na prvem mestu.

Pravilo `FREQ` predstavlja frekvenco ponavljanja dogodka. Možne vrednosti so:

- `SECONDLY`, za ponavljanje vsako sekundo
- `MINUTELY`, za ponavljanje vsako minuto
- `HOURLY`, za ponavljanje vsako uro
- `DAILY`, za dnevno ponavljanje
- `WEEKLY`, za tedensko ponavljanje
- `MONTHLY`, za mesečno ponavljanje
- `YEARLY`, za letno ponavljanje

Pravilo `INTERVAL` vsebuje pozitivno celo število, ki predstavlja interval ponavljanja frekvence. Če je frekvenca nastavljena na dnevno, interval pa na 8, pomeni, da se dogodek ponovi vsak 8. dan.

Pravilo `UNTIL` definira končni datum ponavljanja dogodka (vključno s končnim datumom). Definiramo ga lahko samo z datumom ali pa z datumom in uro.

Pravilo `COUNT` vsebuje pozitivno celo število, ki predstavlja število ponovitev dogodka. V kolikor ima pravilo nastavljeno tako `UNTIL` kot `COUNT`, obvelja tisto pravilo, ki ponavljanje dogodka prej konča.

Pravila `BYSECOND`, `BYMINUTE` in `BYHOUR` vsebujejo z vejico ločene sezname pozitivnih celih števil. Možne vrednosti `BYSECOND` so od 0 do 60, `BYMINUTE` od 0 do 59 in `BYHOUR` od 0 do 23. Če nastavimo pravilo `BYHOUR` na vrednost `8,20`, se bo dogodek ponovil ob 8h zjutraj in 8h zvečer.

Pravilo `BYDAY` vsebuje z vejico ločen seznam dnevov v tednu, kjer je `MO` ponedeljek, `TU` torek, `WE` sreda, `TH` četrtek, `FR` petek, `SA` sobota in `SU` nedelja. Pred oznako za posamezen dan lahko postavimo pozitivno ali negativno celo število, ki predstavlja N-to ponovitev v mesečnem ali letnem ponavljanju dogodka. Za mesečno ponavljanje dogodka vrednost `+1MO` pomeni vsak prvi ponedeljek v mesecu, `-1MO` vsak zadnji ponedeljek v mesecu, `MO` pa preprosto vsak ponedeljek v mesecu.

Pravilo `BYMONTHDAY` vsebuje z vejico ločen seznam celih števil z možnimi vrednostmi od -31 do -1 in 1 do 31, ki prestavljajo dneve v mesecu. Vrednost `-10` predstavlja 10.

dan od konca meseca. Pravilo BYMONTHDAY ne sme biti nastavljeno, če je frekvenca ponavljanja nastavljena na tedensko.

Pravilo BYYEARDAY vsebuje z vejico ločen seznam celih števil z možnimi vrednostmi od -366 do -1 in 1 do 366, ki predstavljajo dneve v letu. Vrednost -1 predstavlja zadnji dan v letu (31. december), -306 pa 306. dan od konca leta (1. marec). Pravilo BYYEARDAY ne sme biti uporabljeno, če je v uporabi dnevna, tedenska ali mesečna frekvenca ponavljanja dogodka.

Pravilo BYWEEKNO vsebuje z vejico ločen seznam celih števil z možnimi vrednostmi od -53 do -1 in 1 do 53, ki predstavljajo tedne v letu. Prvi teden v letu je tisti, ki ima vsaj 4 dni v koledarskem letu. Vrednost 3 predstavlja tretji teden v letu. Pravilo BYWEEKNO je lahko definirano le, če imamo opravka z letno frekvenco dogodka.

Pravilo BYMONTH vsebuje z vejico ločen seznam pozitivnih celih števil z možnimi vrednostmi od 1 do 12, ki predstavljajo mesece v letu. Vrednost 2 predstavlja 2. mesec v letu.

S pravilom WKST definiramo dan začetka tedna. Možne vrednosti so iste kot pri pravilu BYDAY. Vrednost SU pomeni, da se delovni teden za nas začne v nedeljo. Če pravilo ni posebej nastavljeno, obvelja vrednost MO.

Pravilo BYSETPOS vsebuje z vejico ločen seznam celih števil z možnimi vrednostmi od -366 do -1 in 1 do 366. Predstavljajo N-to ponovitev znotraj pravil BYSECOND, BYMINUTE, BYHOUR, BYDAY, BYMONTHDAY, BYYEARDAY in BYWEEKNO. Primer 3.2 predstavlja zadnji delovni dan v mesecu.

Primer 3.2 Primer uporabe pravila za zadnji delovni dan v mesecu.

```
1 FREQ=MONTHLY;BYDAY=MO,TU,WE,TH,FR;BYSETPOS=-1
```

Pravila lahko v nekaterih situacijah povzročijo ponovitev dogodka na neobstoječ datum, kot je naprimer 30. februar. V takih primerih se ponovitev dogodka preskoči brez opozorila.

Začetni datum ponavljanja dogodka je definiran izven omenjenih pravil, in sicer v delu DTSTART. Datum prve ponovitve dogodka je vedno isti začetnemu datumu, ne glede na to, če je ta dan smiseln glede na opisana pravila ponavljanja.

Opisana pravila s predpono BY vedno na nek način spremenijo ponavljanje dogodka. Število dni lahko ta pravila omejijo kot v primeru FREQ=DAILY;BYMONTH=1, kjer dnevno ponavljanje omeji samo na mesec januar. Lahko pa jih tudi razširijo, kot v

	SECONDLY	MINUTELY	HOURLY	DAILY	WEEKLY	MONTHLY	YEARLY
BYMONTH	omeji	omeji	omeji	omeji	omeji	omeji	razširi
BYWEEKNO	-	-	-	-	-	-	razširi
BYYEARDAY	omeji	omeji	omeji	-	-	-	razširi
BYMONTHDAY	omeji	omeji	omeji	omeji	-	razširi	razširi
BYDAY	omeji	omeji	omeji	omeji	razširi	izjema 1	izjema 2
BYHOUR	omeji	omeji	omeji	razširi	razširi	razširi	razširi
BYMINUTE	omeji	omeji	razširi	razširi	razširi	razširi	razširi
BYSECOND	omeji	razširi	razširi	razširi	razširi	razširi	razširi
BYSETPOS	omeji	omeji	omeji	omeji	omeji	omeji	omeji

Tabela 3.1 Razpredelnica izjem pri uporabi pravil za ponavljanje dogodkov. **Izjema 1:** omeji, če je prisotno pravilo `BYMONTHDAY`, drugače razširi. **Izjema 2:** omeji za pravili `BYYEARDAY` in `BYMONTHDAY`, drugače razširi.

primeru `FREQ=YEARLY;BYMONTH=1,2`, kjer je enkratno letno ponavljanje razširjeno na ponovitev vsak januar in februar, vsako leto.

Če je v pravilu ponavljanja uporabljenih več pravil s predpono `BY`, se ta obravnavajo v vrstnem redu `FREQ`, `INTERVAL`, `BYMONTH`, `BYWEEKNO`, `BYYEARDAY`, `BYMONTHDAY`, `BYDAY`, `BYHOUR`, `BYMINUTE`, `BYSECOND`, `BYSETPOS`, `COUNT` in `UNTIL`.

Tabela 3.1 opisuje vse izjeme, ki jih lahko povzročijo pravila s predpono `BY`.

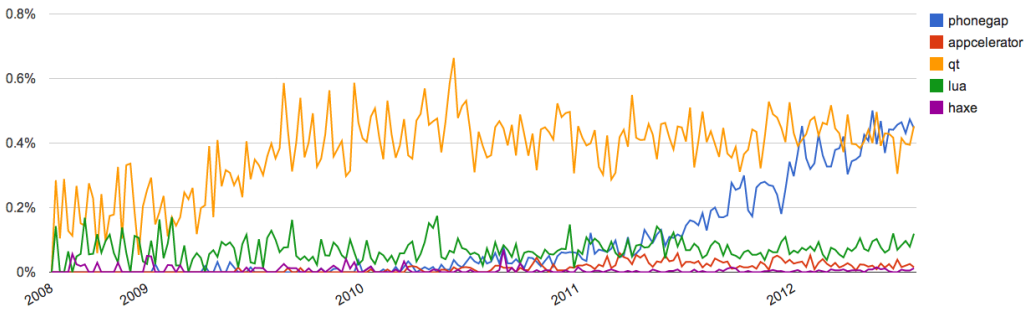
3.3 Omejitve

Preden se lotimo izbora primerne metode, postavimo nekaj omejitev:

1. Delovati mora na platformah iOS, Android, Windows Phone in spletu.
2. Zagotavljati grafično skladnost z ostalimi domorodnimi aplikacijami.
3. Imeti dovolj razgibano razvijalsko skupnost, da bomo lahko našli odgovore na nastale probleme.
4. Mora biti odporna na spremembe pri nadgradnjah platforme.
5. Biti cenovno ugodna.

3.4 Izbor primerne metode

Izbor primerne metode lahko začnemo s pregledom popularnih vprašanj na spletni strani Stackoverflow (slika 3.2). Vidimo lahko izjemno popularnost ogrodij Qt in PhoneGap. Kot opombo lahko omenim, da Xamarin na tem grafu ni prikazan zaradi premajhnega



Slika 3.2 Prikaz procenta tedenskih trendov vprašanj za nekaj od predlaganih rešitev na spletni strani Stackoverflow, kjer razvijalci iščejo rešitve problemov, na katere so naleteli.

	Qt	Xamarin	Air	Cordova	Titanium	Lua	Haxe	C++	JavaScript
Android	✓	✓	✓	✓	✓	✓	✓	✓	✓
iOS	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows Phone	✗	✓	✗	✓	✗	✓	✓	✓	✓
Spletna aplikacija	✗	✗	✓	✓	✗	✓	✓	✓	✓
Grafična skladnost	✗	✓	✗	✗	✓				
Skupnost	✓	✓	✗	✓	✓	✓	✗	✓	✓
Odpornost na nadgradnje	✗	✗	✗	✗	✗	✗	✗	✓	✗
Cena (mesečno)	149$	299$	-	-	-	-	-	-	-

Tabela 3.2 Pregled funkcionalnosti predstavljenih metod.

števila vprašanj. Prav tako ne bi bilo ravno smiselno vključiti jezik C++ ali JavaScript, saj ti podatki ne bi bili reprezentativni.

Vse omejitve, omenjene v prejšnjem poglavju, so predstavljene v tabeli 3.2. Prazne vrstice pri "grafični skladnosti" za Lua, Haxe, C++ in JavaScript so posledica nezmožnosti zadostitvi tem omejitvam, niso pa ovira, saj mora v tem primeru za grafično skladnost poskrbeti domorodna koda, ki ni napisana v omenjenih jezikih.

Iz tabele 3.2 je lepo razvidno, da le rešitev C++ zadošča vsem naštetim omejitvam. Namesto razvoja medplatformne aplikacije se bomo rajši odločili za razvoj medplatformne knjižnice, ki jo bomo nato vključili v domorodne aplikacije s pomočjo jezikovnih ovojev.

3.5 C++

Namesto razvoja lastne knjižnice bi lahko na tem mestu v naš projekt vključili odprtokodno knjižnico `libical`[30], a bomo za potrebe diplomske naloge rajši izbrali samostojno rešitev, saj bomo s tem lahko podrobneje raziskali, kaj vse je potrebno za razvoj podobne

knjižnice.

Izbrani del RRULE specifikacije RFC 5545 je na srečo dovolj preprost, da za razvoj potrebujemo le knjižnico STL. V kolikor bi naša rešitev zahtevala vključitev dodatne knjižnice - recimo vzpostavitev internetne povezave - bi lahko ta problem rešili na dva načina:

1. V našo rešitev bi vključili dodatno knjižnico `libcurl`. To bi povzročilo kar nekaj problemov pri vključevanju knjižnice na različnih platformah, saj bi morali knjižnico `libcurl` pripraviti za vsako platformo posebej.

2. Nalogo vzpostavitve in prenosa podatkov iz oddaljene lokacije bi lahko delegirali v domorodno kodo, ki bi po končanju prenosa rezultat vrnila v našo knjižnico.

Če izberemo prvi način, bo naša knjižnica podvajala funkcionalnost, ki že obstaja v domorodni kodi. Veliko lepša rešitev je uporaba delegiranja v domorodno kodo, saj lahko ta bolje izkorišča vse sposobnosti naprave. To sicer pomeni nekaj več kode v ovojih naše knjižnice (kar si bomo ogledali v poglavju 4), a omogoča boljšo odpornost na nadgradnje operacijskega sistema ciljne platforme.

Razvito knjižnico lahko v aplikacije za različne platforme vključimo na več načinov:

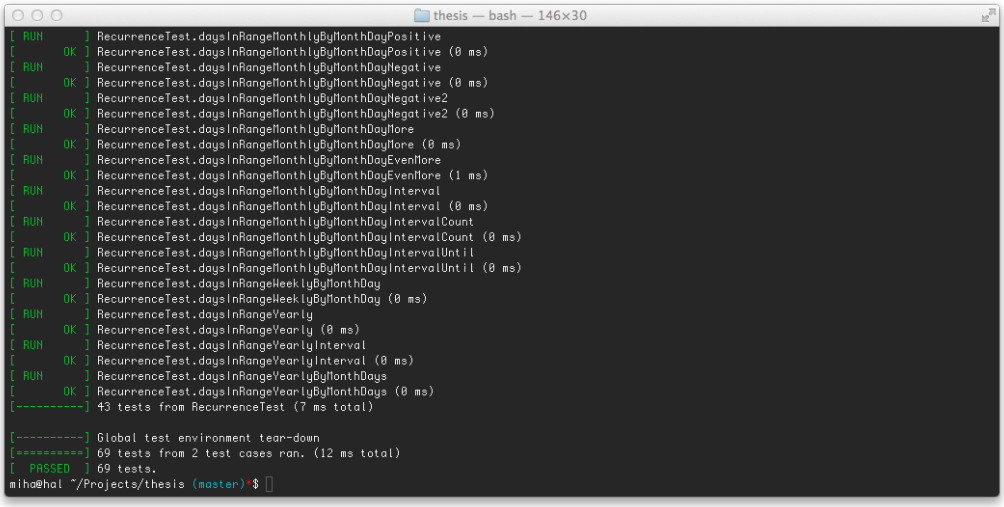
1. Z izvorno kodo C++, ki jo ciljni program vključi v svoj paket.

2. Statično knjižnico (angl. *static library*), ki jo ciljni program vključi v svoj paket.

3. Deljeno knjižnico (angl. *shared library*), ki jo ciljni program samo referencira, a je ne vključi neposredno v svoj paket.

Pri vseh naštetih načinih je potrebna dodatna ovojna koda (angl. *wrapper*), ki je različna za vsako ciljno platformo. Podrobnosti teh ovojev si bomo ogledali v poglavju 4.

Celotna izvorna koda knjižnice (skupaj z jezikovnimi ovoji) je na voljo na spletni strani github.com/mihahribar/thesis. Zaradi zagotavljanja kakovosti (angl. *quality assurance*) je bila knjižnica razvita s testno usmerjenim razvojem TDD[46]. Za implementacijo testov sta bili uporabljeni izvrstni knjižnici `googletest`[31] in `googlemock`[32], knjižnica pa se lahko pohvali z več kot 90% testno pokritostjo kode (angl. *code coverage*), kar je moč preveriti s pomočjo programa `gcov`.



Slika 3.3 Zaslonska slika testov C++ knjižnice.

Za potrebe zvezne integracije (angl. *continuous integration*) projekt uporablja storitev Travis CI[33], ki za odprtokodne projekte nudi brezplačno avtomatizirano testiranje. Vse, kar je potrebno storiti, da Travis CI lahko testira izvorno kodo knjižnice, je zapisano v datoteki `.travis.yml`. Ta vsebuje ukaz za zagon testov knjižnice (v našem primeru `make tests`), kar na strežniku za zvezno integracijo sproži prenos in pripravo knjižnic `googletest` in `googlemock`, prevede vse potrebne datoteke z izvorno kodo ter zgradi testni program, ki zažene vse naše testne primere.

Primer 3.3 Primer uporabe C++ knjižnice RRULE standarda RFC 5545. Izbrani dogodek bi se s tem pravilom ponavljal tedensko, vsak ponedeljek, od 1. januarja 2014 naprej.

```
1 Recurrence rec = Recurrence(Weekly, Date(2014, 1, 1));
2 rec.setByDay("MO");
3 map<int, Date> days = rec.daysInRange(Date(2014, 2, 1), Date(2014, 2, 28));
4 // spremenljivka days bo vsebovala
5 // result[5] = Date(2014, 2, 3);
6 // result[6] = Date(2014, 2, 10);
7 // result[7] = Date(2014, 2, 17);
8 // result[8] = Date(2014, 2, 24);
```

Primer uporabe knjižnice RRULE RFC 5545 lahko vidimo v primeru 3.3. Razred `Date` vsebuje vso potrebno logiko za delo z datumi, kot so naprimer prištevanje, odštevanje ter primerjanje datumov. Razred `Recurrence` vsebuje logiko za ponavljanje dogodkov. Tip dogodka, ki se ponavlja, je poljuben in ni del knjižnice.

4 Vključitev knjižnice v različne platforme

4.1 iOS

Platforma iOS primarno uporablja jezik Objective-C, ki je objektna razširitev jezika C. Na srečo obstaja tudi variacija Objective-C++, ki nam omogoča souporabo jezikov Objective-C in C++ v istem projektu. Datoteki, v kateri želimo uporabljati C++, namesto končnice `.m` pripnemo končnico `.mm`.

iOS ne podpira uporabe deljene knjižnice (angl. *shared library*), omogoča pa uporabo statične knjižnice (angl. *static library*) ali izvorne kode. Za prvi primer izberimo uvoz izvorne kode.

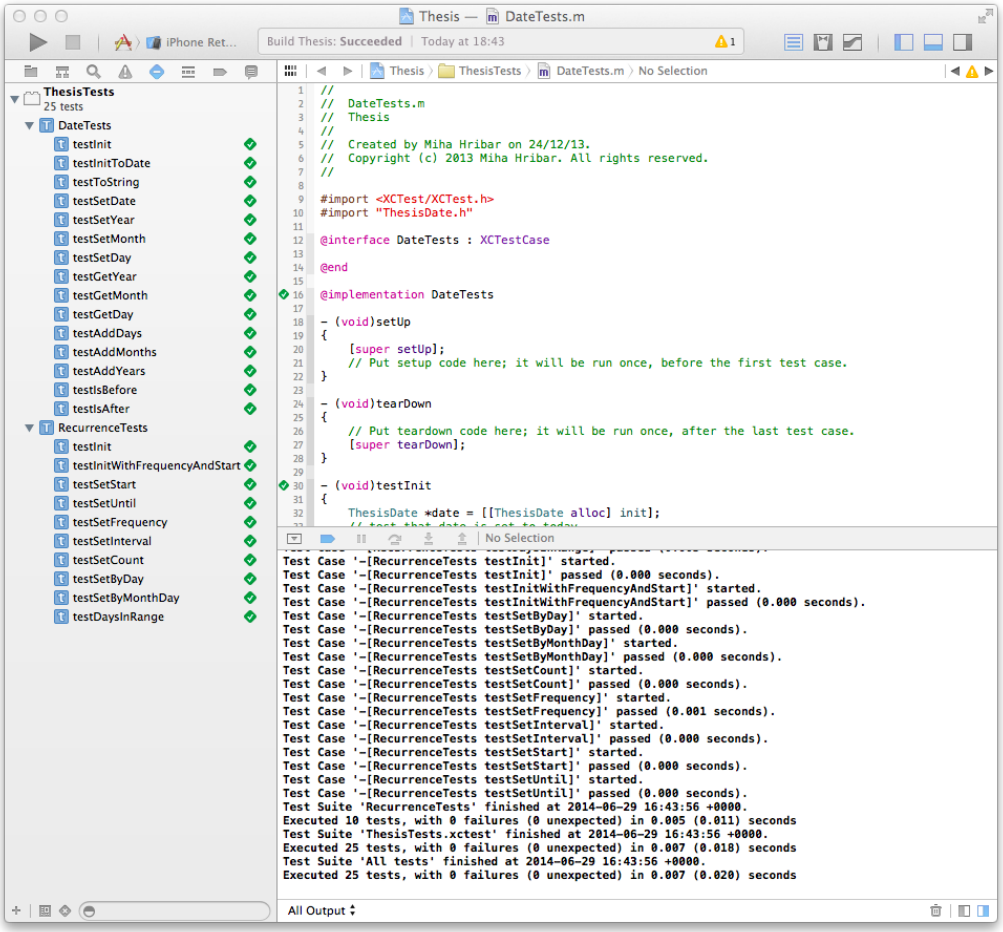
iOS v svojem arzenalu ne vključuje orodja za avtomatično sproščanje pomnilnika (angl. *garbage collection*). Od razvijalca se pričakuje, da za seboj počisti pomnilnik z uravnoteženimi ukazi `retain` in `release`, ko je koda opravila svoje delo. V ta namen je v integriranem razvijalskem okolju (angl. *Integrated Development Environment*) XCode na voljo kar nekaj orodij, ki nam omogočajo lažjo detekcijo puščanja pomnilnika (angl. *memory leak*). Kar rado se zgodi, da se pri štetju referenc razvijalec zmoti. Na srečo je v iOS5 Apple predstavil ARC (avtomatično štetje referenc - angl. *Automatic*

Reference Counting), s čimer so razvijalcem znatno olajšali delo, saj prevajalnik sedaj sam zna vnesti ukaze za sproščanje pomnilnika.

Implementacija iOS ovoja je dokaj preprosta (glej primer 4.1). Za vsakega C++ od razredov naredimo zrcalne Objective-C++ ovojne razrede, ki v inicializacijski metodi `init` poskrbijo za pravilno dodeljevanje in hranjenje C++ objekta (vrstica 14). Ko na določen objekt ne kaže več noben kazalec, se pred sprostitvijo pomnilnika pokliče metoda `dealloc`, v kateri poskrbimo za ustrezno sprostitev C++ objekta, preden pride do puščanja pomnilnika (vrstica 21). Klic v ovoj nato preprosto posreduje klic v C++ objekt (vrstica 25) in poskrbi za transformacije med C++ in Objective-C podatkovnimi tipi.

Primer 4.1 Primer Objective-C++ ovoja C++ razreda `Date`.

```
1  #import "ThesisDate.h"
2  #import "Date.hpp"
3
4  @interface ThesisDate() {
5     Thesis::Date* wrapped;
6  }
7  @end
8
9  @implementation ThesisDate
10
11 - (ThesisDate *)initWithYear:(NSInteger)year month:(NSInteger)month andDay:(
       NSInteger)day {
12    self = [super init];
13    if (self) {
14       wrapped = new Thesis::Date(year, month, day);
15       if (!wrapped) self = nil;
16    }
17    return self;
18 }
19
20 - (void)dealloc {
21    delete wrapped;
22 }
23
24 - (void)addDays:(NSInteger)days {
25    wrapped->addDays(days);
26 }
```



Slika 4.1 Zaslonska slika zagona testov iOS ovoja C++ knjižnice v razvojnem okolju XCode.

Razvojno okolje XCode vsebuje testno ogrodje `XCTest`, ki nam omogoča zagotavljanje kakovosti. Testi iOS ovoja testirajo samo klice v knjižnico, ne testirajo pa dejanske funkcionalnosti, saj bi to pomenilo podvajanje že obstoječih testov C++. Teste zaženemo iz menija Product → Test.

4.2 Android

Java, ki jo srečamo na platformi Android, se od odprtokodne Jave kar precej razlikuje. Android ne uporablja Java virtualnega pogona (angl. *Java Virtual Machine*), ampak svoj pogon Dalvik, ki je prilagojen za uporabo na mobilnih napravah. Pred kratkim je Google napovedal nov virtualen pogon, Android Runtime (ART), ki bo v prihodnosti zamenjal

Dalvik in vsebuje veliko izboljšav v hitrosti in stabilnosti izvajanja aplikacij.

Za izdelavo ovoja C++ kode moramo uporabiti dve ogrodji:

1. JNI (angl. *Java Native Interface*), ki poskrbi za komunikacijo med jezikoma Java in C++

2. NDK (angl. *Native Development Kit*), ki poskrbi za pravilno prevajanje C++ kode za vsako od ciljnih arhitektur Android platforme (armeabi, armeabi-v7a, x86 in mips).

Uporaba ogrodja JNI[34] je za razvijalca časovno kar potratna. Ovoj je potrebno napisati v dveh delih:

1. Java ovoj, ki izpostavi C++ razrede in metode s pomočjo direktive `native` (glej primer 4.2).

2. C++ ovoj, ki služi kot most med jezikoma Java in C++ ter poskrbi za transformacije med podatkovnimi tipi (glej primer 4.3).

Referenco na C++ objekt hranimo v Java delu JNI ovoja, C++ ovoj pa do reference dostopa preko metode `getHandle` (glej 4.3), ki spremenljivko `nativeHandle` poišče v Java delu JNI ovoja.

Primer 4.2 Primer Java ovoja C++ razreda `Date`.

```
1  public class Date
2  {
3        private long nativeHandle = 0;
4        public native void init(int year, int month, int day);
5        public native boolean isBefore(Date date);
6        public native void dispose();
7
8        public Date(int year, int month, int day) {
9              init(year, month, day);
10       }
11
12       static {
13             System.loadLibrary("thesis");
14       }
15 }
```

Java ovoje shranimo v direktorij `src`, medtem ko C++ JNI ovoje shranimo v `jni`. Poglejmo si, kako poteka komunikacija med Javo in C++ v primeru klica `isBefore` (glej primer 4.3):

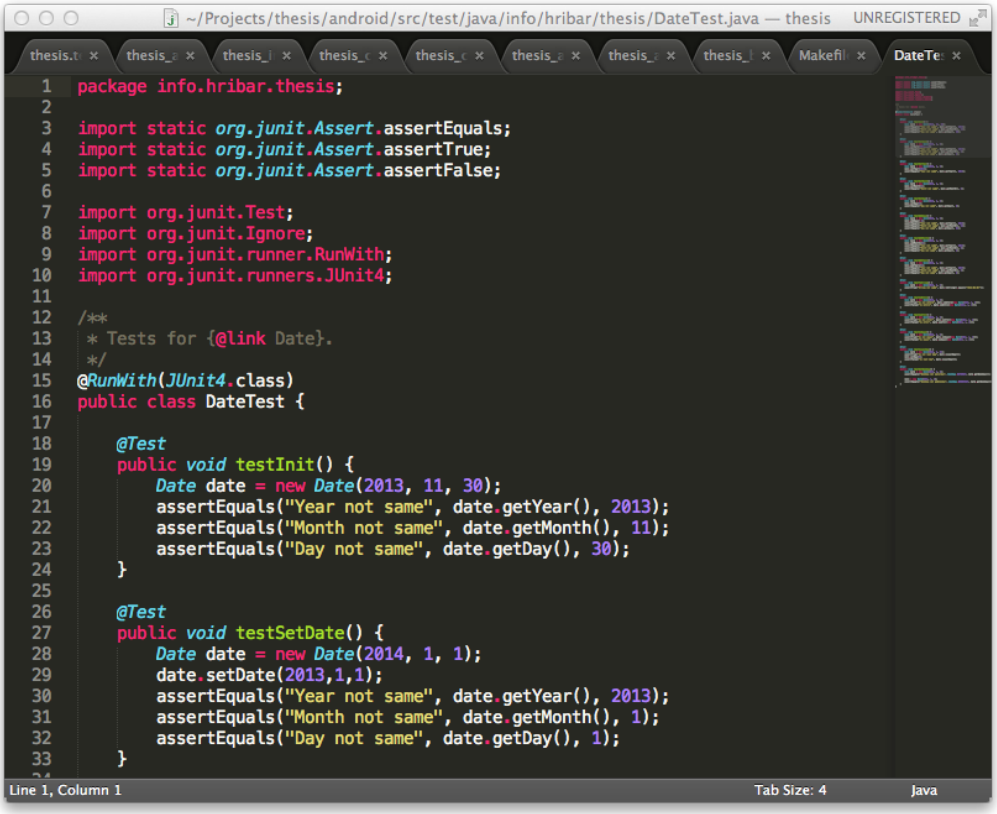
1. Java preko JNI najde pravo metodo v C++ ovoju s pomočjo dogovorjene poimenovalne sheme (vrstica 12)
2. C++ ovoj pretvori Java argumente v C++ argumente (vrstica 14).
3. C++ ovoj najde predhodno shranjen objekt (vrstice 40-44).
4. Pokliče pravilno metodo na najdenem objektu s C++ argumenti (vrstice 21-29).
5. Če metoda vrne kak rezultat, C++ ovoj poskrbi za pretvorbo nazaj v Java podatkovne tipe (vrstica 13).

Primer 4.3 Primer mosta med jezikoma Java in C++ razreda `Date`.

```
1  #include "info_hribar_thesis_Date.h"
2  #include "Date.cpp"
3
4  using namespace Thesis;
5
6  JNIEXPORT void JNICALL Java_info_hribar_thesis_Date_init(JNIEnv *env,
       jobject obj, jint year, jint month, jint day) {
7        Date *date = new Date(year, month, day);
8        setHandle(env, obj, date);
9  }
10
11 JNIEXPORT jboolean JNICALL Java_info_hribar_thesis_Date_isBefore(JNIEnv *env
       , jobject obj, jobject compare) {
12       Date *date = getHandle<Date>(env, obj);
13       return date->isBefore(getDate(env, compare));
14 }
15
16 JNIEXPORT void JNICALL Java_info_hribar_thesis_Date_dispose(JNIEnv *env,
       jobject obj) {
17       Date *date = getHandle<Date>(env, obj);
18       delete date;
19 }
20
21 Date getDate(JNIEnv *env, jobject date) {
```

```
22        jclass dateCls = env->GetObjectClass(date);
23        jmethodID mGetYear = env->GetMethodID(dateCls, "getYear", "()I");
24        jmethodID mGetMonth = env->GetMethodID(dateCls, "getMonth", "()I");
25        jmethodID mGetDay = env->GetMethodID(dateCls, "getDay", "()I");
26        jint year = env->CallIntMethod(date, mGetYear);
27        jint month = env->CallIntMethod(date, mGetMonth);
28        jint day = env->CallIntMethod(date, mGetDay);
29        return Date(year, month, day);
30    }
31
32    jfieldID getHandleField(JNIEnv *env, jobject obj)
33    {
34       jclass c = env->GetObjectClass(obj);
35       // J is the type signature for long:
36       return env->GetFieldID(c, "nativeHandle", "J");
37    }
38
39    template <typename T>
40    T *getHandle(JNIEnv *env, jobject obj)
41    {
42       jlong handle = env->GetLongField(obj, getHandleField(env, obj));
43       return reinterpret_cast<T *>(handle);
44    }
45
46    template <typename T>
47    void setHandle(JNIEnv *env, jobject obj, T *t)
48    {
49       jlong handle = reinterpret_cast<jlong>(t);
50       env->SetLongField(obj, getHandleField(env, obj), handle);
51    }
```

Bralca morda zanima, čemu služi metoda `dispose` (vrstice 16-19). Java nam nudi avtomatično sproščanje pomnilnika (angl. *garbage collection*), medtem ko C++ od razvijalca zahteva samostojno čiščenje pomnilniških naslovov, ki niso več v uporabi. Ko v Javi pride do sproščanja pomnilnika, se pokliče metoda `finalize`, a kot lahko preberemo v dokumentaciji[35], do klica ne prihaja prav pogosto in se za tako uporabo ne priporoča. Poleg tega je vsak razred, ki implementira metodo `finalize`, deležen malce večje obdelave s strani operacijskega sistema, kar botruje počasnejšemu izvajanju. Ker želimo biti malce bolj prijazni do platforme, na kateri gostujemo, se držimo pravila: ko objekta




```java
package info.hribar.thesis;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;

import org.junit.Test;
import org.junit.Ignore;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;

/**
 * Tests for {@link Date}.
 */
@RunWith(JUnit4.class)
public class DateTest {

    @Test
    public void testInit() {
        Date date = new Date(2013, 11, 30);
        assertEquals("Year not same", date.getYear(), 2013);
        assertEquals("Month not same", date.getMonth(), 11);
        assertEquals("Day not same", date.getDay(), 30);
    }

    @Test
    public void testSetDate() {
        Date date = new Date(2014, 1, 1);
        date.setDate(2013,1,1);
        assertEquals("Year not same", date.getYear(), 2013);
        assertEquals("Month not same", date.getMonth(), 1);
        assertEquals("Day not same", date.getDay(), 1);
    }
```

Slika 4.2 Zaslonska slika, ki prikazuje teste razreda `Date` Java ovoja.

ne potrebujemo več, ga sprostimo s pomočjo klica `dispose` (naprimer v `finally try catch finally` konstruktu).

Za testiranje Java ovoja smo uporabili ogrodji JUnit[36] in Maven[37]. Tako kot v primeru iOS ovoja se na tem mestu ne testira dejanske knjižnice, ampak zgolj povezavo iz C++ knjižnice v Java kodo. Teste lahko zaženemo z ukazom `mvn test`, ki s pomočjo orodja Maven poskrbi za prenos vseh potrebnih knjižnic in za zagon testov Java ovoja.

4.3 Windows Phone

Primarni jezik vseh Windows platform je C# in isto velja tudi za Windows Phone. Z 8. verzijo mobilnega operacijskega sistema je Microsoft odprl možnost souporabe C# in domorodne kode. To storimo z uporabo Windows Phone Runtime komponente (WinPRT), ki jo spišemo v jeziku C++ in nato uvozimo v naš Windows Phone projekt kot zunanjo referenco.

Našo knjižnico lahko uvozimo kot statično knjižnico (angl. *static library*) ali direktno kot C++ izvorno kodo (če imamo do nje dostop). Če se odločimo za uporabo statične knjižnice, moramo paziti, da ta uporablja le standardno knjižnjico (STL) in Win32 klice, ki so dovoljeni za Windows Phone aplikacije[38].

Funkcionalnost, ki jo rabimo v naši Windows Phone 8 aplikaciji, izvozimo v WinPRT C++ komponenti (glej primer 4.4).

Primer 4.4 C++ koda za izvoz funkcionalnosti knjižnice v JavaScript razreda `Date`.

```
1  namespace ThesisWINRT {
2      using namespace Windows::Foundation;
3      using Platform::String;
4
5      public ref class Date sealed {
6      public:
7          unsigned int GetLength(String^ strToParse);
8      };
9  }
```

4.4 Spletna aplikacija

C++ knjižnico moramo za uporabo v spletni aplikaciji prevesti v JavaScript. To lahko storimo s pomočjo ogrodja Emscripten, ki vzame LLVM zlogovno kodo in namesto prevoda v nabor ukazov za podprte procesorje prevede to kodo v JavaScript. Rezultat je knjižnica, ki jo lahko brez težav uvozimo v obstoječo spletno aplikacijo.

Primarno Emscripten prevaja celotne programe, ki imajo jasno definirane vhode in izhode. Te lahko v JavaScriptu sprožimo, kot bi jih v uporabniški vrstici (angl. *terminal*). V našem primeru gre vendarle za prevod C++ knjižnice, za kar Emscripten potrebuje dodatna navodila za izvoz funkcionalnosti (glej primer 4.5). Za te namene projekt Emscripten vsebuje `embind`[39], s pomočjo katerega lahko izvozimo dostop do razredov, podatkovnih tipov, pomnilniškega upravljanja in podobnih jezikovnih konstruktov.

Primer 4.5 C++ koda za izvoz funckionalnosti knjižnice v JavaScript razreda `Date`.

```
1  #include "emscripten/bind.h"
2  #include "src/Date.hpp"
3
4  using namespace emscripten;
5  using namespace Thesis;
6
```

```
7  EMSCRIPTEN_BINDINGS(date) {
8      class_<Date>("Date")
9          .constructor<int, int, int>()
10         .property("year", &Date::getYear, &Date::setYear)
11         .property("month", &Date::getMonth, &Date::setMonth)
12         .property("day", &Date::getDay, &Date::setDay)
13         .function("setDate", &Date::setDate)
14         .function("addDays", &Date::addDays)
15         .function("addMonths", &Date::addMonths)
16         .function("addYears", &Date::addYears)
17         .function("toString", &Date::toString)
18         .function("isBefore", &Date::isBefore)
19         .function("isAfter", &Date::isAfter)
20         .function("isEqual", &Date::isEqual)
21         .function("getWeekday", &Date::getWeekday)
22         .function("isLastDay", &Date::isLastDay);
23 }
```

Končni rezultat izvoza knjižnice v JavaScript lahko vidimo v primeru 4.6.

Primer 4.6 Primer uporabe izvoženega razreda `Date` v JavaScript.

```
1 var date = new Thesis.Date(2014, 10, 10);
2 date.addMonths(1);
3 console.log(date.toString());
```

5 Evaluacija

5.1 Pregled prednosti, slabosti, priložnosti in nevarnosti

Pred zaključnimi ugotovitvami si poglejmo prednosti, slabosti, priložnosti in nevarnosti (t.i SWOT analiza) izbrane metode.

5.1.1 Prednosti

- Metoda nam omogoča poenoten razvoj knjižnice za vse želene platforme.
- Možnost večje pokritosti z napisanimi testi (angl. *code coverage*), saj knjižnico zaradi poenotenega razvoja testiramo le na C++ nivoju.
- Manjše število napak pri razvoju aplikacij na posameznih platformah.
- Odpornost na nadgradnje gostujočega operacijskega sistema.
- Hitrost izvajanja.
- Vsa uporabljena orodja so brezplačna.

5.1.2 Slabosti

- Uporabniški vmesnik je potrebno razviti za vsako platformo posebej.
- V aplikacije se lahko še vedno prikradejo napake v uporabniškem vmesniku, ki jih moramo reševati za vsako platformo posebej.
- Potrebno predhodno znanje več različnih jezikov (C++, Java, C#, Objective-C in JavaScript).
- Za razvoj iOS aplikacije je potrebno kupiti Apple računalnik.
- Za razvoj Windows Phone aplikacije je potrebno kupiti licenčno kopijo operacijskega sistema Windows 8.1, ki je pogoj za uporabo brezplačnega orodja Visual Studio Express for Windows[40].

5.1.3 Priložnosti

- Podpora za nove platforme, ki podpirajo C++ (recimo Windows Desktop, Mac OSX itd.).
- Učenje novih programskih jezikov je za razvijalce priporočljivo, saj so pri tem primorani razmišljati o problemih na drugačen način[45].

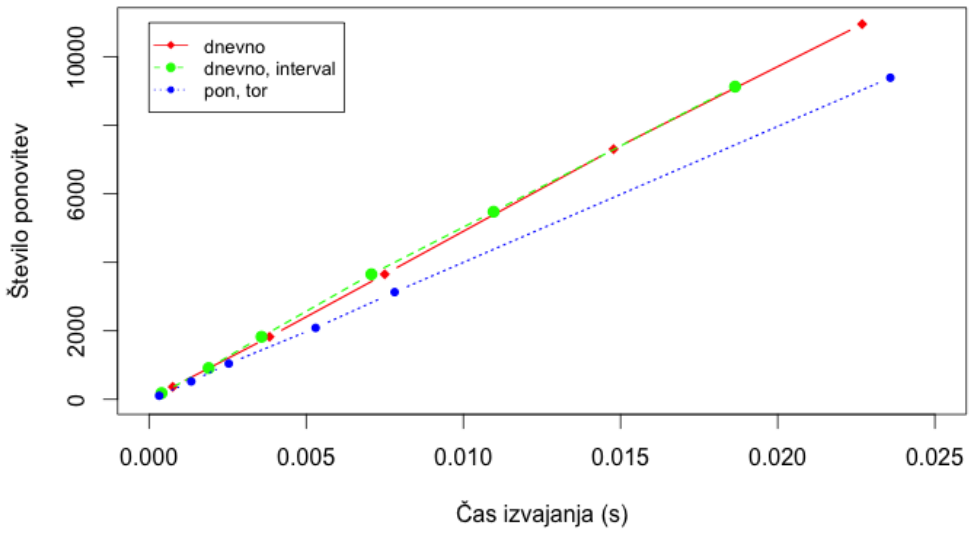
5.1.4 Nevarnosti

- Dolgotrajen razvoj, predvsem na začetku, dokler nimamo pripravljenih osnovnih jezikovnih ovojev.
- Težko spreminjanje knjižnice, možno le dodajanje novih metod zaradi združljivosti s starejšimi verzijami.

5.2 Performančna analiza

Performančno knjižnica ni zahtevna, kar smo lahko preverili na izračunu približno 10000 ponovitev dogodkov z različnimi pravili ponavljanj. Rezultati so vidni na sliki 5.1, iz njih pa lahko sklepamo, da ima naša knjižnica zahtevnost $\mathrm{O}(n)$.

Kljub računski nezahtevnosti naše knjižnice si vseeno poglejmo kakšne posledice ima lahko vključitev C++ knjižnice v različne jezike, ki so v uporabi na izbranih platformah.



Slika 5.1 Rezultati performančne analize C++ knjižnice.

5.2.1 Objective-C in C++

Razlika v hitrosti izvajanja med kodo, napisano v jeziku C++, in Objective-C kodo je zanemarljiva, saj je Objective-C za platformo iOS domoroden jezik (angl. *native programming language*). Pri vključitvi C++ knjižnice v našo aplikacijo je moč zaznati malenkost višjo porabo pomnilnika (zaradi podvojenih objektov v C++ knjižnici), vendar za vse to poskrbimo v metodi `dealloc`, v kateri sprostimo pomnilnik.

Z zagotovostjo lahko trdimo, da vključitev C++ knjižnice v Objective-C projekt ne bo imel večjega vpliva na hitrost izvajanja aplikacije.

5.2.2 Java in C++

V raziskavi Univerze na Dunaju[51] so ugotovili, da je Java s pomočjo JIT prevajalnika zmožna na preprostih primerih (recimo generiranje Fibonaccijevega zaporedja) v hitrosti izvajanja prehiteti C++ kodo. V nekoliko kompleksnejših primerih uporabe (recimo branje XML dokumentov), kjer prevajalnik JIT ni bil zmožen pohitriti Java kode, pa je bil C++ vseeno hitrejši.

Avtorji na koncu navedejo, da vključevanje C++ kode v Android aplikacije ne bi

smelo imeti velikega vpliva na hitrost, saj je bila hitrost izvajanje kode v večini primerov primerljiva.

5.2.3 C# in C++

Za večino primerov je hitrost izvajanja C# kode primerljiva izvajanju C++ kode. Razlike se začnejo kazati pri procesorsko zahtevnejših algoritmih, kjer se pokaže prednost C++ kode predvsem zaradi boljše izrabe pomnilnika[53]. Zavedati se moramo, da pri prevelikem prehajanju iz domorodne kode v C# kodo lahko pride do izničenja vseh prednosti C++ kode, saj v vmesnem delu pride do pretvarjanja podatkovnih tipov iz C++ sveta v C# svet (in obratno).

Po vsem tem lahko pridemo do istega zaključka kot v primeru Jave - vključitev C++ knjižnice v našo Windows Phone aplikacijo v večini primerov ne bo imelo prevelikega vpliva na hitrost.

5.2.4 JavaScript in C++

Na področju JavaScripta je bil v zadnjih nekaj letih videti velik napredek. S pomočjo asm.js[27], ki je podmnožica jezika JavaScript, je spletnim brskalnikom v letu 2014 uspelo doseči le 2-kratno upočasnitev v primerjavi z domorodno kodo. Ista raziskava iz leta 2011 je pokazala kar 12-kratno upočasnitev[49].

Neposredno pisanje v asm.js je dokaj nerealno, saj razvijalca sili v pisanje kode na način, ki ga najbrž ni vajen (glej primer 5.1). Namesto tega je asm.js še posebej primeren kot končni jezik za prevajanje med jeziki z ogrodjem, kot je Emscripten. Koda, prevedena iz jezika C++, bo zaradi tega v večini primerov hitrejša kot tista, ki jo razvijalec sam napiše v jeziku JavaScript.

Primer 5.1 Primer prevoda C kode v asm.js kodo.

```
1  // C koda
2  int f(int i) {
3    return i + 1;
4  }
5
6  // asm.js koda
7  function f(i) {
8    i = i|0;
9    return (i + 1)|0;
10 }
```

6 Ugotovitve

V diplomski nalogi smo pokazali, kako je možno poenostaviti sočasni razvoj aplikacij za različne platforme. S pomočjo projekta Emscripten in jezikovnih ovojev smo uspešno uporabili isto C++ knjižnico v iOS, Android, Windows Phone in spletni aplikaciji. Na ta način lahko skoraj celotno logiko aplikacije prenesemo v knjižnico, ki si jo vse platforme med seboj delijo. Še vedno moramo za vsako platformo posebej razviti uporabniški vmesnik, a lahko tako aplikacijo povsem prilagodimo zmožnostim operacijskega sistema.

Naloga se ni izkazala za prav preprosto, kar je bilo tudi pričakovano predvsem zaradi razlik med izbranimi platformami. Kot smo pokazali, performančno gledano prikazana rešitev končnih aplikacij najbrž ne bo preveč pohitrila, a to vendarle ni bil namen diplomske naloge.

V bližnji prihodnosti lahko pričakujemo, da opisana rešitev ne bo več edini način vključitve knjižnice v različne platforme. Apple je že začel prvi korak z ogrodjem `JavaScriptCore`, ki omogoča boljše mešanje JavaScript in domorodne kode. Podobne rešitve lahko kmalu pričakujemo tudi od ostalih izbranih platform, kar bo znatno olajšalo razvoj medplatformnih knjižnic. Ko pride do tega, bo poglavje 2.3.5 lahko lepo izhodišče

za novo raziskavo.

Bralec, ki bi opisane primere rad preizkusil, lahko izvorno kodo najde na spletni strani github.com/mihahribar/thesis.

Literatura

[1] Choose a licence (2014). Dostopno na:
http://choosealicense.com/licenses/

[2] QT projekt (2014). Dostopno na:
http://qt-project.org

[3] Xamarin (2014). Dostopno na:
https://xamarin.com

[4] PhoneGap (2014). Dostopno na:
http://phonegap.com

[5] Appcelerator Titanium (2014). Dostopno na:
http://www.appcelerator.com/titanium/

[6] Lua (2014). Dostopno na:
http://www.lua.org

[7] Haxe (2014). Dostopno na:
http://haxe.org

[8] C++ (2014). Dostopno na:
http://www.cplusplus.com

[9] Emscripten (2014). Dostopno na:
https://github.com/kripken/emscripten

[10] GPL v3 licenca (2014). Dostopno na:
http://www.gnu.org/copyleft/gpl.html

[11] LGPL v2.1 licenca (2014). Dostopno na:
https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

[12] Mathematica (2014). Dostopno na:
http://www.wolfram.com/mathematica/

[13] VLC (2014). Dostopno na:
http://www.videolan.org/vlc/index.html

[14] Mono (2014). Dostopno na:
http://www.mono-project.com

[15] .NET (2014). Dostopno na:
http://www.microsoft.com/net

[16] Rdio (2014). Dostopno na:
https://www.rdio.com

[17] Adobe air (2014). Dostopno na:
http://get.adobe.com/air/

[18] Angry birds (2014). Dostopno na:
https://www.angrybirds.com

[19] Apache cordova (2014). Dostopno na:
http://cordova.apache.org

[20] Apache license 2.0 (2014). Dostopno na:
http://choosealicense.com/licenses/apache-2.0/

[21] jQueryMobile (2014). Dostopno na:
http://jquerymobile.com

[22] Sencha touch (2014). Dostopno na:
http://www.sencha.com/products/touch/

[23] V8 (2014). Dostopno na:
https://code.google.com/p/v8/

[24] MIT license (2014). Dostopno na:
http://choosealicense.com/licenses/mit/

[25] XMLVM (2014). Dostopno na:
http://xmlvm.org

[26] TIOBE index for june 2014 (jun. 2014). Dostopno na:
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[27] asm.js (2014). Dostopno na:
http://asmjs.org

[28] JavaScript.NET (2014). Dostopno na:
https://github.com/JavascriptNet/Javascript.Net

[29] ECMAScript (2014). Dostopno na:
http://www.ecmascript.org

[30] libical (2014). Dostopno na:
http://www.citadel.org/doku.php/documentation:featured_projects:libical

[31] googletest (2014). Dostopno na:
https://code.google.com/p/googletest/

[32] googlemock (2014). Dostopno na:
https://code.google.com/p/googlemock/

[33] Travis CI - free hosted continuous integration platform for the open source community (2014). Dostopno na:
https://travis-ci.org

[34] Java SE 7 Java Native Interface - related APIs and developer guides (2014). Dostopno na:
http://docs.oracle.com/javase/7/docs/technotes/guides/jni/

[35] Android object documentation (2014). Dostopno na:
http://developer.android.com/reference/java/lang/Object.html

[36] JUnit - a programmer-oriented testing framework for Java. (2014). Dostopno na:
http://junit.org

[37] Apache maven (2014). Dostopno na:
http://maven.apache.org

[38] Static libraries (C++/CX) (2014). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/hh771041.aspx

[39] Embind documentation (2014). Dostopno na:
https://github.com/kripken/emscripten/wiki/embind

[40] Visual studio express (2014). Dostopno na:
http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

[41] D. Berkholz, GitHub language trends and the fragmenting landscape (maj 2014). Dostopno na:
http://redmonk.com/dberkholz/2014/05/02/github-language-trends-and-the-fragmenting-landscape/

[42] A. Charland, B. Leroux, Mobile application development: Web vs. native, Communications of the ACM 54.

[43] B. Desruisseaux, Internet calendaring and scheduling core object specification (sep. 2009). Dostopno na:
http://tools.ietf.org/html/rfc5545

[44] B. Desruisseaux, Recurrence rule, rfc 5545 (sep. 2009). Dostopno na:
http://tools.ietf.org/html/rfc5545#section-3.3.10

[45] A. Hunt, D. Thomas, The Pragmatic Programmer: From Journeyman to Master, Pragmatic Bookshelf, 1999.

[46] J. Langr, Modern C++ Programming with Test-Driven Development: Code Better, Sleep Better, Pragmatic Bookshelf, 2013.

[47] F. Lardinois, Mozilla's asm.js gets another step closer to native performance (dec. 2013). Dostopno na:
http://techcrunch.com/2013/12/21/mozillas-asm-js-gets-another-step-closer-to-native-performance/

[48] O. Mathews, JavaScriptCore and iOS 7 (sep. 2013). Dostopno na:
http://www.bignerdranch.com/blog/javascriptcore-and-ios-7/

[49] C. McAnlis, P. Lubbers, et al., HTML5 Game Development Insights, Apress, 2014.

[50] Mozilla, Epic Citadel demo shows the power of the web as a platform for gaming (maj 2013). Dostopno na:

https://blog.mozilla.org/futurereleases/2013/05/02/epic-citadel-demo-shows-the-power-of-the-web-as-a-platform-for-gaming/

[51] T. Peer, M. Wagner, Embedding C++ code in Android Java applications, Tech. rep., Institute of Computer Languages Compilers and Languages Group, Vienna University of Technology (okt. 2012).

[52] J. Rivera, R. van der Meulen, Gartner says annual smartphone sales surpassed sales of feature phones for the first time in 2013 (feb. 2014). Dostopno na: http://www.gartner.com/newsroom/id/2665715

[53] A. Whitechapel, S. McKenna, Windows Phone 8 Development Internals (Developer Reference), Microsoft Press, 2013.