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

Dejan Lavbič
MENTOR

Ljubljana, 2014

Namesto te strani se vstavi *original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!*

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 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, junij 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 zaželjeno.

Kljub različnosti 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[28], 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 najbolj primerno 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 different plaftorms, 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 RRULE RFC5545[28] specification, which enables applications to schedule and display recurring events. The library will then be used in an 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 potrpežjivost pri nastanku 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, junij 2014.

KAZALO

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.

To je seveda zelo težko doseči, sploh z majhno ekipo. Dodatno se stvari zakomplicirajo, če so vse te naprave na različnih operacijskih sistemih. Tako imam lahko Windows prenosnik, Android telefon in Apple tablico. Kot razvajen uporabnik pričakujem, 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 od platform. Četudi omejimo razvoj na najbolj priljubljene platforme iOS, Android in Windows Phone, 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 funckionalnosti.

Izkušen razvijalec bo pri predstavitvi problema takoj pomislil na medplatformni razvoj, ki si ga bomo ogledali v drugem poglavju. Omenili bomo tako imenovane "celotive" metode, kot so Qt[1] in Xamarin[2], "hibridne" kot sta recimo PhoneGap[3] in Appcelerator Titanium[4], ter "deljene" metode npr. Lua[5], Haxe[6] in C++[7].

V tretjem poglavju si bomo ogledali zakaj smo se odločili za razvoj knjižnice s pomočjo C++, ter jo tudi zgradili. V četrtem poglavju jo bomo ovili v nekaj ovojev in uspešno uporabili v Objective-C (iOS), Javi (Android) in C# (Windows Phone). Predstavili bomo tudi način, kako lahko C++ kodo prevedemo v JavaScript s pomočjo orodja Emscripten[8], in knjižnico uporabili tudi v spletni aplikaciji.

Pa začnimo.

2 Pregled metod medplaformnega razvoja

Predno 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, delnih in deljenih metod.

2.1 Celovit

Celovita metoda za razvoj uporablja ogrodje, s pomočjo katerega aplikacijo pripravimo za različne platforme. Velika večina tako napisane izvorne kode je uporabljena na vseh destinacijskih 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[1] je ogrodje za grafično programiranje za več platform s pomočjo jezika C++ in QML[1]. Omogoča nam sočasni razvoj za platforme OSX, Linux, Windows, Android in

[1] Qt Meta Language ali Qt Modeling Language

iOS. Podpira tudi uporabo HTML5 namesto QML, kar pomeni, da spletni razvijalci lahko uporabijo že obstoječe znanje in učenje novega jezika ni potrebno.

Qt projekt je povsem odprtokoden in dovoljuje uporabo v skladu z licencama GPL v3[9] in LGPL v2.1[10], 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 pred časom Microsoft kupil podjetje Nokia, 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 primeren za izdelavo aplikacij, ki vključujejo kompleksne algoritme, za katere bi porabili preveč časa pri prepisovanju na različne platforme. Lep primer tega sta aplikaciji Mathematica[11] in multimedijski predvajalnik VLC[12].

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.

2.1.2 Xamarin

Xamarin[2] je ogrodje za sočasen razvoj aplikacij za platforme iOS, Android, Mac in Windows v jeziku C#. Izjaha iz projekta Mono[13], ki omogoča uporabo ogrodja .NET na različnih platformah. Ogrodje omogoča razvoj aplikacij, katerih izgled je skladen z ostalimi domorodnimi aplikacijami.

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

Xamarin je primeren za izdelavo aplikacij za več različnih platform, kjer je ključnega pomena končna grafična skladnost z ostalimi domorodnimi aplikacijami. Kot primer si lahko ogledamo aplikacijo za poslušanje glasbe Rdio[14], 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 funckionalnosti posameznih platform, ko se te nadgradijo, določen riziko predstavlja tudi muhavost posameznih

platform pri omejitvah uporabe tega ogrodja, sploh če nadgradnja povzroči nedelovanje takih aplikacij.

2.1.3 Adobe Air

Adobe Air[15] 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 za razvoj mobilnih aplikacij omejen na uporabo jezika ActionScript. V času pisanja diplomske naloge ogrodje ne omogoča zagon 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 naprimer Angry Birds[16].

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 ga bomo ogledali v nadaljevanju). Uporaba tudi ni primerna za razvoj klasičnih mobilinih aplikacij, saj je prilagajanje domorodnim aplikacijam precej zahtevno, še posebej kadar na platformi pride do posodobitve izgleda.

2.2 Hibriden

Hibridna metoda za razvoj aplikacij uporablja spletne tehnologije v sožitju z 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[17] je odprtokodni projekt, ki omogoča objavo spletnih aplikacij kot domorodne. 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.

Projekt PhoneGap[3] je dejansko samo ena od distribucij projekta Apache Cordova, ki poleg vseh obstoječih funkcionalnosti ponuja tudi razne storitve na katerih delajo v podjetju Adobe.

Za razvoj aplikacij razvijalci lahko uporabljajo spletne tehnologije HTML, CSS in JavaScript. S pomočjo ogrodij jQuery Mobile in Sencha Touch 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, ki služi kot most med kodo napisano v jeziku JavaScript in domorodno kodo.

Glavna prednost ogrodja Apache Cordova in predvsem distribucije PhoneGap je izredno 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 ta za prikazovanje izkorišča vgrajeno spletno okno. Trenutno je težko izdelati aplikacije, ki so grafično zahtevnejše, kar pomeni še toliko bolj pereč problem na napravah s slabšimi karakteristikami. Da se aplikacija po izgledu nebi 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 iz iOS6 na iOS7.

2.2.2 Appcelerator Titanium

Ogrodje Titanium[4] 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[18] (Android), JavaScriptCore (iOS)[29] ali vgrajenega JavaScript okolja (če aplikacijo poganjamo v brskalniku). Za pravilen vizualen izgled skrbijo namestniški elementi, ki uporabljajo domorodne grafične elemente, kar pomeni da vizualno aplikacije ne ločimo od ostalih domorodnih aplikacij. V času pisanja diplomske naloge ogrodje podpira iOS, Android, Blackberry, Tizen in spletne aplikacije.

Glavna prednost ogrodja Titanium ni t.i. način piši enkrat, uporabljaj povosd; njegova prednost je da lahko celotno aplikacijo izdelamo v enem jeziku - JavaScript-u. Le

redko se bomo srečali z domorodno kodo, saj ogrodje nudi široko paleto knjižnic.

Prav tako kot PhoneGap, je 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 Deljen

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 programskimi ovojev, s katerimi pripravimo knjižnico za vgradnjo v druge platforme.

2.3.1 Lua

Lua[5] 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.

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.

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

2.3.2 Haxe

Haxe[6] 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# ali Javo. Nudi tudi dodatne vmesnike za dostop do specifičnih metod ciljnega jezika.

Jezik se je prijel predvsem za razvoj iger, kjer naj bi dolgoročno 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[19] 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, Pyhton, Objective-C in C++.

Projekt izgleda zelo ambiciozen, a vse kaže da je šlo le za akademsko raziskavo, saj je v času pisanja diplome minilo že več ko leto dni odkar se je izvorna koda posodobila. Kljub temu se mi je projekt zdel zanimiv in ga je bilo vredno izpostaviti.

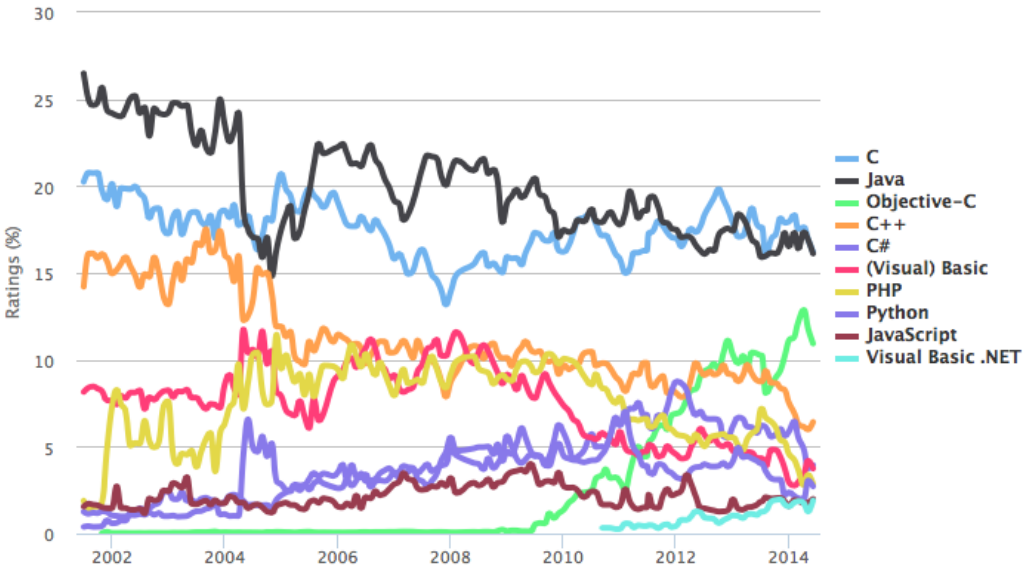
2.3.4 C++ in emscripten

V kolikor nobena od naštetih možnosti ne zadošča našim potrebam, želeli pa bi vseeno imeti deljeno knjižnico, obstaja še ena možnost: uporaba jezika C++[7] in projekta emscripten[8].

C++ je eden izmed najbolj razširjenih programksih jezikov. V času pisanja diplomske naloge zaseda četrto mesto na lestvici najbolj popularnih jezikov 2.1, pred njim so samo C, Java in Objecitve-C. Uporablja se ga v raznolikih projektih, od prevajalnikov, strežnikov, do video igric.

Emscripten je projekt Mozilinih laboratorijev, ki omogoča prevajanje iz LLVM[2] 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 skrbi 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 prevedemo skoraj vsak program

[2] Low Level Virtual Machine



Slika 2.1 Tiobe programming comunity index [20].

(z določenimi omejitvami) v JavaScript in ga zaženemo v brskalniku. Celo grafično zahtevne aplikacije[3] niso problematične, saj emscripten za prevod v JavaScript uporablja asm.js[22], kar je podmnožica jezika JavaScript, ki jo JavaScript pogoni znajo izredno dobro optimizirati[4].

2.3.5 JavaScript

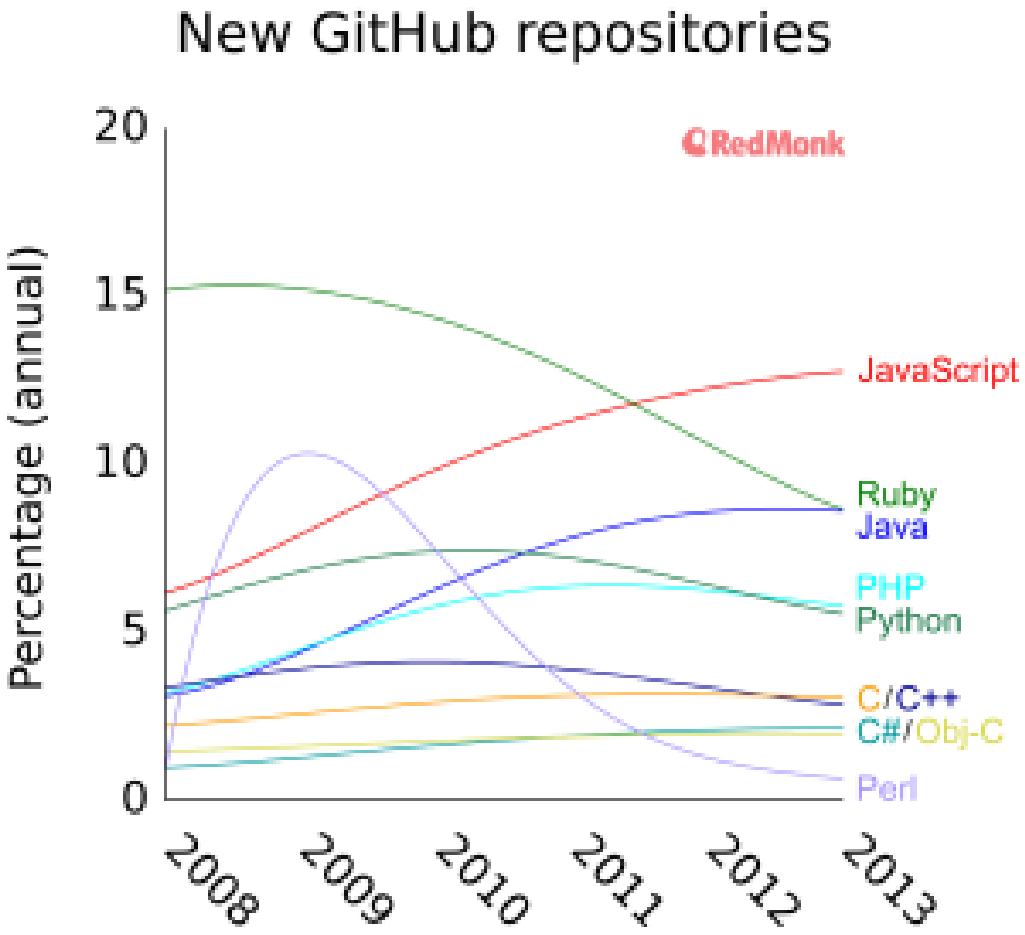
Namesto prevajanja v jezik JavaScript s pomočjo ogrodja Emscripten, bi lahko celotno knjižnico napisali kar v jeziku JavaScript. V zadnjih nekaj letih je jezik JavaScript doživel izredno hitro rast, tako v popularnosti, kot v zmogljivosti in funkcionalnostih, kar je tudi razvidno iz slike 2.2.

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 SDK v času pisanja diplomske naloge ne nudi direktne implementacije. Zaradi tega smo primorani vključiti JavaScript pogon, kot je Rhino, V8 in podobni. V kolikor je pogon napisan v jeziku Java, je integracija

[3]Skupina Mozilinih inžinirjev 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[21].

[4]S pomočjo asm.js so v podjetju Mozilla uspeli doseči le enkrat počasnejše izvajanje od domorodne kode, kar je izjemen dosežek.[23]



Slika 2.2 Jeziki novih projektov na spletni strani github.com.

preprosta, če pa izberemo pogon v drugem jeziku, mora za Android obstajati paket za 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[24].

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).

Težji del je vgraditev pogona JavaScript na vsako od platform. Dokler ne bo na voljo domorodnih integracij, se bomo težko zanesli na brezhibno delovanje pri nadgradnji operacijskega sistema. Navkljub temu, je ideja zelo zanimiva in vredna nadaljne raziskave.

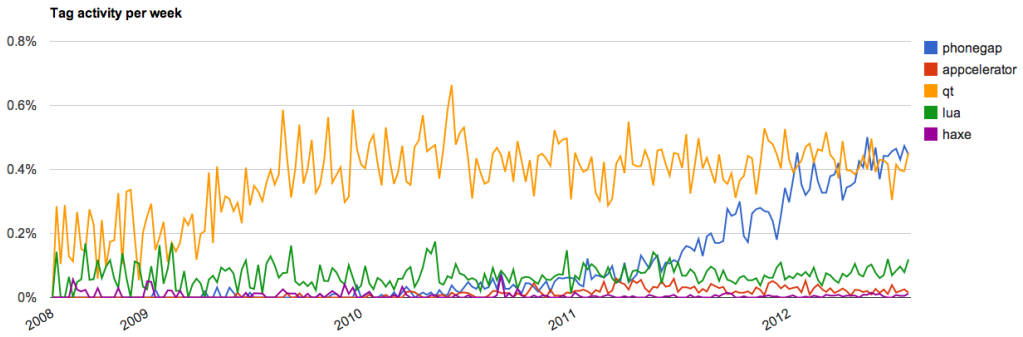
3 Razvoj knjižnice

3.1 Omejitve

Predno 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.

Izbrano rešitev želimo prikazati na primeru aplikacije, ki prikazuje ponavaljajoče dogodke s pomočjo standarda RFC 5545[28]. Zaradi poenostavitve se bomo osredotočili zgolj na del RRULE, ki definira pravila za ponavljanje dogodka. Kot je razvidno iz



Slika 3.1 Prikaz trendov za nekaj od predlaganih rešitev na spletni strani Stackoverflow, kjer razvijalci iščejo rešitve problemov na katere so naleteli.

primera 3.1, je s pomočjo standarda RFC 5545 možno opisati kar kompleksne vzorce ponavljanj. Če primer 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 standarda RFC 5545.

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

Ponavljajoč dogodek bo aplikacija prikazala v preprostem seznamu, ki bo prilagojen za vsako od izbranih platform.

// slika iOS seznama z vsemi podatki ki bodo prikazani

3.2 Izbor primerne metode

Izbor primerne metode lahko začnemo z pregledom popularnih vprašanj na spletni strani Stackoverflow (slika 3.1). Vidimo lahko izjemno popularnost ogrodij Qt in Phonegap. Kot opombo lahko omenim, da Xamarin na tem grafu ni prikazan zaradi premajhnega števila vprašanj. Prav tako nebi bilo ravno smiselno vključiti jezik C++ ali JavaScript, saj ti podatki niso 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 zadoščanja te omejitve, 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 edino rešitev C++ zadošča vsem naštetim omejitvam. Poglejmo si torej kako lahko izbrano rešitev pripravimo s pomočjo jezika C++, ter kako lahko nato isto rešitev uporabimo tudi v aplikaciji, ki teče spletnem brskalniku, s pomočjo

Omejitev/Rešitev	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 3.1 Pregled funkcionalnosti posametnih rešitev.

ogrodja Emscripten.

3.3 C++

Izbrani del RRULE specifikacije RFC 5545 je na srečo dovolj preprost, da za razvoj potrebujemo le STL (Standard Template Library) knjižnico. 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 povzorčilo kar nekaj problemov pri vključevanju knjižnice na različne platforme, saj bi morali knjižnico `libcurl` pripraviti za vsak platformo posebej.

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

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

Razvito knjižnico lahko v platformske aplikacije vključimo na več različnih 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 jo ne vključi direktno v svoj paket.

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

Primer uporabe knjižnjice RRULE RFC 5545 lahko vidimo v primeru 3.2. 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 knjižnjice.

Primer 3.2 Primer uporabe C++ knjižnjice 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);
```

4 Vključitev knjižnice v različne platforme

4.1 iOS

Platforma iOS primarno uporablja jezik Objective-C, ki je, kot ime namiguje, 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 arsenalu ne vključuje orodje 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 zna sam 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 poskribmo za ustrezno sprostitev C++ objekta predno 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  }
```

4.2 Android

Java, ki jo srečamo na platformi Android, se od odprtokodne Jave kar precej razlikuje. Na površju za razvijalce morda ni opaziti razlike, a se veliko razlik skriva pod gladino. Android ne uporablja Java virtualnega pogona (angl. *Java Virtual Machine*) ampak svoj pogon Dalvik, ki je prilagojen za uporabo na mobilnih napravh. 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 aplikacij.

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

1. JNI (Java Native Interface), ki poskrbi za komunikacijo med jezikoma Java in C++
2. NDK (Native Development Kit), ki poskrbi za pravilno prevajanje C++ kode za vsako od ciljnih arhitektur Android platforme (armeabi, armeabi-v7a, x86 in mips).

Uporaba orodja JNI je za razvijalca kar časovno potratna. Spisati je treba dva ovoja:

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.

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 prailno metodo na najdenem objektu z C++ argumenti (vrstice 21 - 29).
5. Če metoda vrne kak rezultat, C++ ovoj poskbi za pretvorbo nazaj v Java podatkovne tipe (vrstica 13).

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[25] 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 iz strani operacijskega sistema, kar botruje počasnejšemu izvajanju. Ker želimo biti malce bolj prijazni do platforme na kateri gostujemo, se držimo pravila: ko objekta ne potrebuje več, ga sprostimo s pomočjo klica `dispose` (naprimer v `finally try catch finally` konstruktu).

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 }
```

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  }
```

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++, to pa 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[26].

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 funckionalnosti 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 orodja emscripten, ki vzame LLVM zlogovno kodo in namesto prevoda v nabor ukazov za podprte procesorje, prevede to kodo v JavaScript. Rezultat je knjižnjica, 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++ knjižnjice, za kar emscripten potrebuje dodatna navodila za izvoz funkcionalnosti (glej primer 4.5). Za te namene projekt emscripten vsebuje `embind`[27], s pomočjo katerega lahko izvozimo dostop do razredov, podatkovnih tipov, pomnilniškim upravljanje in podobne jezikovne konstrukte.

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 Ugotovitve

V diplomski nalogi smo pokazali, kako je možno poenostaviti sočasni razvoj aplikacij za več različnih platform. S pomočjo projekta emscripten in destinacijskih ovojev smo uspešno uporabili isto C++ knjižnico v iOS, Android, Windows Phone in spletni aplikaciji. Naloga se ni izkazala za prav preprosto, kar je bilo tudi pričakovano, predvsem zaradi razlik med izbranimi platformami. Diplomska naloga mi je ponudila priložnost osvežiti svoje znanje jezikov C++, Java, Objective-C, C# in JavaScript, ki jih v svojem delu ne uporabljam ravno vsakodnevno.

V bližnji prihodnosti pričakujem, da opisana rešitev ne bo več edini način vključitve knjižnjice na različne platforme. Apple je že začel prvi korak z ogrodjem `JavaScriptCore`, ki omogoča boljše mešanje JavaScript in domorodne kode. Podobne rešitve pričakujem tudi od ostalih izbranih platform, kar bi znatno olajšalo razvoj medplatformnih knjižnjic. Ko pride do tega, bom za svoje lastne projekte rajši izbral razvoj z jezikom JavaScript, ki smo ga predstavili v poglavju 2.3.5.

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

Literatura

[1] Qt projekt.
url: http://qt-project.org

[2] Xamarin.
url: https://xamarin.com

[3] Phonegap.
url: http://phonegap.com

[4] Appcelerator titanium.
url: http://www.appcelerator.com/titanium/

[5] Lua.
url: http://www.lua.org

[6] Haxe.
url: http://haxe.org

[7] C++.
url: http://www.cplusplus.com

[8] emscripten.
url: https://github.com/kripken/emscripten

[9] Gpl v3 licenca.
url: http://www.gnu.org/copyleft/gpl.html

[10] Lgpl v2.1 licenca.
url: https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

[11] Mathematica.
url: http://www.wolfram.com/mathematica/

[12] Vlc.
url: http://www.videolan.org/vlc/index.html

[13] Mono.
url: http://www.mono-project.com

[14] Rdio.
url: https://www.rdio.com

[15] Adobe air.
url: http://get.adobe.com/air/

[16] Angry birds.
url: https://www.angrybirds.com

[17] Apache cordova.
url: http://cordova.apache.org

[18] V8.
url: https://code.google.com/p/v8/

[19] Xmlvm.
url: http://xmlvm.org

[20] Tiobe index for june 2014 (Jun. 2014).
url: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[21] 'epic citadel' demo shows the power of the web as a platform for gaming (May 2013).
url: https://blog.mozilla.org/futurereleases/2013/05/02/epic-citadel-demo-shows-the-power-of-the-web-as-a-platform-for-gaming/

[22] asm.js.
url: http://asmjs.org

[23] Mozilla's asm.js gets another step closer to native performance (Dec. 2013).
url: http://techcrunch.com/2013/12/21/mozillas-asm-js-gets-another-step-closer-to-native-performance/

[24] Javascript.net.
url: https://github.com/JavascriptNet/Javascript.Net

[25] Android object documentation.
url: http://developer.android.com/reference/java/lang/Object.html

[26] Static libraries (c++/cx).
url: http://msdn.microsoft.com/en-us/library/windows/apps/hh771041.aspx

[27] Embind documentation.
url: https://github.com/kripken/emscripten/wiki/embind

[28] B. Desruisseaux, Specifikacija rfc 5545 (Sep. 2009).
url: http://tools.ietf.org/html/rfc5545

[29] O. Mathews, Javascriptcore and ios 7 (Sep. 2013).
url: http://www.bignerdranch.com/blog/javascriptcore-and-ios-7/