UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Simčič

Avtomatizacija naročil v taksi službi

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Dušan Kodek

Ljubljana, 2012

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Matej Simčič,

z vpisno številko 63030051,

sem avtor diplomskega dela z naslovom:

Avtomatizacija naročil v taksi službi

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Dušana Kodeka
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 12.1.2012 Podpis avtorja:

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01781/2011
Datum: 04.10.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ SIMČIČ**

Naslov: **AVTOMATIZACIJA NAROČIL V TAKSI SLUŽBI**
AUTOMATION OF ORDERS IN TAXI SERVICE

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Naročanje taksijev poteka preko klicnega centra, ki je ob dnevnih konicah in ob večjih prireditvah pogosto preobremenjen. Avtomatizacija naročil razbremeni dispečerja v klicnem centru tako, da za naročanje poskrbi računalnik, ki naročilo pošlje najbližjemu prostemu taksistu. Za izvedbo avtomatizacije je potrebno zagotoviti, da ima računalnik podatke o trenutni lokaciji taksistov in o njihovi zasedenosti. V ta namen zasnujte rešitev, ki omogoča, da se podatki o vsakem od taksijev s pomočjo Android aplikacije v mobilnem telefonu preko internetnega omrežja avtomatsko prenašajo na strežnik, kjer se zapisujejo v podatkovno bazo. Podatki naj vsebujejo lokacijo vozila, ki se pridobi z GPS sprejemnikom, in zasedenost sedežev v vozilu, ki se ugotovi s pomočjo na dotik občutljivih uporov. Izdelajte vso potrebno programsko in strojno opremo. Pravilnost delovanja preverite in podajte oceno delovanja ter načine za izboljšave.

Mentor:
prof. dr. Dušan Kodek

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Kazalo

Seznam uporabljenih kratic in simbolov

API – (ang.) Application programming interface

DVM – (ang.) Dalvik virtual machine

GPS – (ang.) Global Positioning System

XML – (ang.) Extensible Markup Language

SDK – (ang.) Standard Development Kit

AVD – (ang.) Android Virtual Device

APK – (ang.) (Android) Application Package File

ZIP – datotečni format za stiskanje datotek

JAR – (ang.) Java Archive, ZIP datoteka Javanskih razredov

EAR – (ang.) Enterprise Archive, stisnjena datoteka, ki vsebuje WAR in JAR ter še nekatere druge datoteke

WAR – (ang.) Web application Archive, spletna aplikacija stisnjena v arhiv

Java EE – (ang.) Java Enterprise Edition, Java namenjena spletnemu razvoju

Java ME – (ang.) Java Mobile Edition, Java namenjena mobilnim napravam

Java SE – (ang.) Java Standard Edition, standardna različica Jave

HTML – (ang.) HyperText Markup Language, jezik za izdelavo spletnih strani

JSP – (ang.) Java Server Pages, javanska tehnologija za izdelavo dinamičnih spletnih strani

POM – (ang.) Project Object Model, konfiguracijska datoteka za Maven

JVM – (ang.) Java Virtual Machine, javanski virtualni stroj

SQL – (ang.) Structured Query Language, povpraševalni jezik za delo s podatkovnimi bazami

GORM – (ang.) Grails Object Relational Mapping

SRAM – (ang.) Static Random Access Memory, statični bralno/pisalni pomnilnik

EEPROM – (ang.) Electrically Erasable Programmable Read-Only Memory, električno izbrisljiv in programabilen bralni pomnilnik

TX – (ang.) – transmit, oddajanje

RX - (ang.) – recieve, sprejem

GSM – (ang.) Global System for Mobile Communications orig. Groupe Special Mobile standard mobilnih komunikacij

WiFi – (izhaja iz Wireles Fidelity) brezžično omrežje

TDMA – (ang.) Time Division Multiple Access

MAC naslov – (ang.) Media Access Control address – enolični (fizični) naslov dodeljena omrežnim napravam

bit/s – bitov na sekundo

MHz – (ang.) megahertz, frekvenca megaherc

GHz – (ang.) gigahertz, frekvenca gigaherc

MD5 – (ang.)Message-Digest Algorithm, znan algoritem, ki se uporablja v kriptografiji

HTTPS – Hypertext Transfer Protocol Secure, varna, kriptirana spletna povezava

SSL – Secure Socket Layer, kriptografski protokol

Povzetek

Avtomatizacija se v zadnjih letih bliskovito razvija. Njena prednost je predvsem zmanjševanje stroškov in hitrejše ter zanesljivejše izvajanje opravil, ki bi jih sicer izvajal človek. Začela se je v proizvodni industriji, kasneje pa se je razširila tudi na druge sektorje. Današnja tehnologija omogoča avtomatizacijo najrazličnejših področij, kjer trenutno še vedno ključno vlogo igra človek.

Diplomska naloga obravnava izdelavo sistema, ki omogoča avtomatizirano naročanje taksi prevoza. Sistem sestavljajo štiri komponente. Gre za dve mobilni aplikaciji, ki ju uporabljajo vozniki taksijev in stranke. Povezuje ju spletna aplikacija, ki teče na strežniku. Poleg tega je na mobilno aplikacijo taksista povezano vezje, ki služi zaznavanju prisotnosti potnikov v vozilu s pomočjo na pritisk občutljivih uporov, ki so nameščeni v sedežih vozila.

Prvi del diplomske naloge opisuje tehnologije in strojno opremo uporabljeno za izvedbo sistema. Drugi del opisuje delovanje posameznih komponent in njihov razvoj.

Izdelana rešitev je uporabna za današnje taksi službe, saj bi razbremenila njihov klicni center ter tudi omogoča preprečevanje t. i. sive ekonomije, to je opravljanje storitve taksi prevoza brez beleženja vožnje.

Ključne besede: avtomatizacija, taksi, Arduino, Android, Bluetooth, Amarino, odjemalec – strežnik.

Abstract

Automation is rapidly growing in the last years. The advantages it brings are cost reduction, faster and better performance of tasks that would be otherwise done by humas. It began in the manufacturing industry and later expanded to other sectors. Today's technology allows the implementation of automation in a wide range of areas.

The thesis deals with the implementation of a system that allows automated ordering of a taxi. The system consists of four components. They are two mobile applications used by taxi drivers and their customers. The two mobile applications are connected by a web application which runs on a server. In addition, the mobile application used by the taxi driver is connected to a circuit that is used to detect the presence of passengers in the vehicle. This is acchieved through pressure-sensitive resistors, which are installed in the vehicle seats.

The first part of the thesis describes the technology and hardware used to implement the system. The second part describes the functioning of individual components and their development.

The implemented solution is useful in today's taxi services where it would relieve the calling centers and also help preventing gray economy which in this case would be performing the service without logging the service performed.

Keywords: automation, taxi, Arduino, Android, Bluetooth, Amarino, client – server.

1 UVOD

Ljudje si življenja brez računalnikov ne znamo več predstavljati. Živimo v dobi, ki se jo vse pogosteje poimenuje »doba vseprisotnega računalništva«. Na vsakem koraku nas v takšni ali drugačni obliki spremljajo računalniki. Nekateri današnji pametni telefoni že vsebujejo večjedrne procesorje; po zmogljivosti jih lahko primerjamo z nekaj let starejšimi osebnimi računalniki. Hiter razvoj tehnologije omogoča vgrajevanje visoko zmogljivih računalniških sistemov v različne naprave. Na ta način lahko naprave in procese avtomatiziramo, jih nadziramo na daljavo itd. S pomočjo naštetega počnemo tudi določene stvari, ki jih prej ni bilo mogoče izvajati, saj bi od človeka zahtevale napore in odzivnost, ki jih človek ni zmožen.

Diplomska naloga predstavlja rešitev za težave, s katerimi se srečujejo v današnjih taksi družbah. Težave so v teh primerih predvsem v preobremenjenost klicnega centra v dnevnih konicah in ob večjih prireditvah. Poleg tega pa se v taksi družbah srečujejo tudi s pojavljanjem sive ekonomije. S samo avtomatizacijo skušamo razbremeniti klicni center tako, da naročila ne sprejema več dispečer v klicnem centru, ampak namesto njega za to poskrbi računalnik, ki tudi posreduje naročilo naprej najbližjemu razpoložljivemu taksistu. Poleg tega sem uporabil tudi vezje, ki ga je moč vgraditi v sedeže vozila, kjer omogoča zaznavanje prisotnosti potnikov v samem vozilu. Ti podatki se preko internetnega omrežja periodično pošiljajo na strežnik, kjer se zapisujejo v podatkovno bazo. Poleg podatka o prisotnosti potnikov v vozilu se na strežnik pošilja tudi trenutna lokacija in status o tem, ali je taksi prost ali zaseden. Taksi je zaseden takrat, ko se v njem nahaja potnik, ali pa takrat, ko je taksi sprejel naročilo in je na poti k stranki.

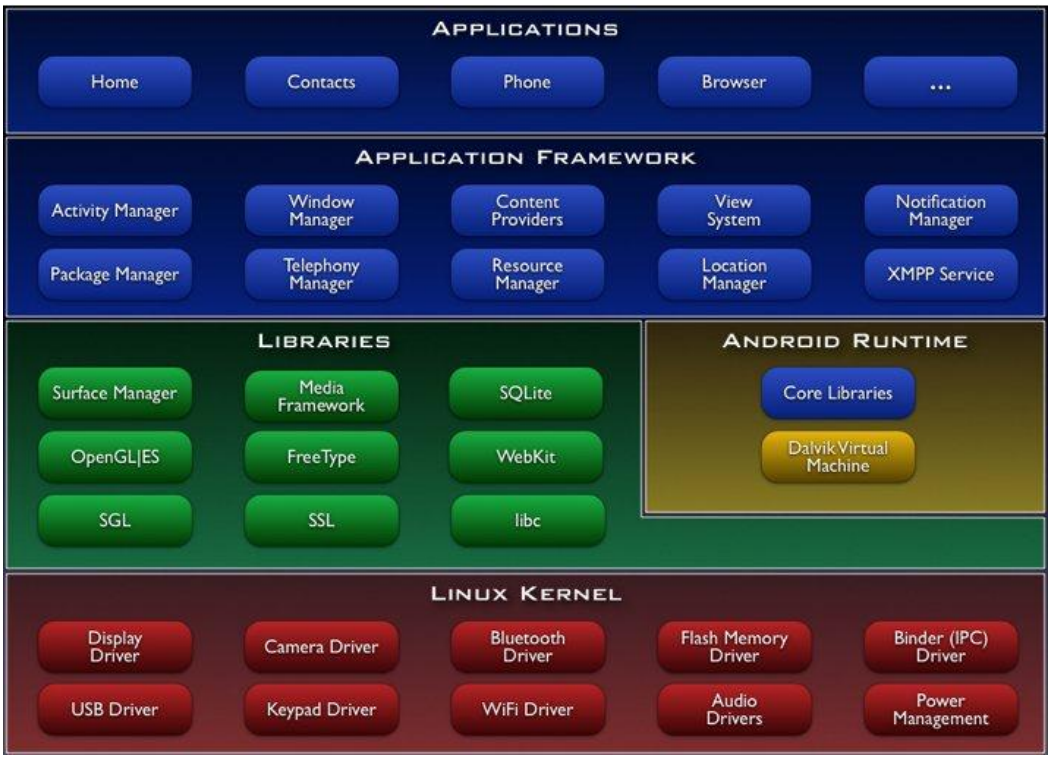
Pogoj za rabo te rešitve je uporaba t. i. pametnega mobilnega telefona z operacijskim sistemom Android s strani naročnika in taksista. Naročnik je v tem primeru posameznik – stranka, ki si želi naročiti taksi. Naročilo poteka tako, da stranka na svoji mobilni aplikaciji pritisne gumb za naročilo, na kar aplikacija pošlje zahtevek preko internetnega omrežja na strežnik, kjer se zahteva sprocesira. Strežnik preveri lokacijo stranke ter to lokacijo primerja s trenutno prostimi taksiji in najustreznejšemu pošlje naročilo. Ta ima možnost naročilo sprejeti ali zavrniti, kar se tudi sporoči na strežnik, ki glede na odločitev tudi ustrezno reagira. Če je naročilo sprejeto, uporabnik dobi potrdilo o naročilu, v nasprotnem primeru se posreduje naročilo naslednjemu najbližjemu prostemu taksistu.

2 UPORABLJENE TEHNOLOGIJE

2.1 Android

Android je operacijski sistem za mobilne naprave, kot so mobilni telefoni in tablični računalniki. Njegove temelje je postavilo podjetje Android (ki ga je avgusta 2005 kupilo podjetje Google). Ekipa razvijalcev pod vodstvom Andyja Rubina, ki je tudi en izmed soustanoviteljev podjetja Android, je nato do konca razvila operacijski sistem Android. Novembra 2007 je svoj obstoj predstavilo združenje Open Handset Alliance, ki ga sestavljajo razna tehnološka podjetja, med njimi tudi Google, Intel, HTC, Motorola, LG, Samsung Electronics, Broadcom Corporation, Texas Instruments, T-Mobile. Njihov cilj je razviti odprte standarde za mobilne naprave. Na ta dan so tudi predstavili svoj prvi produkt – Android operacijski sistem.

Ta sistem sestavljajo jedro, ki je bazirano na jedru operacijskega sistema Linux (ang. Linux kernel), in razne knjižnice ter API-ji (ang. Application programming interface).



Slika 1: Arhitektura operacijskega sistema Android

Na Sliki 1 je prikazanih 5 plasti operacijskega sistema Android. Na najnižjem nivoju se nahaja samo jedro, v tem primeru je to Linux kernel 2.6, prilagojeno za potrebe upravljanja s

porabo energije ter za izvajalno okolje (ang. runtime environment). Na naslednjem nivoju se nahajajo knjižnice, ki so prevedene v optimizirano kodo za naprave, ki imajo manjši glavni pomnilnik in procesorje z nižjo porabo energije. Na tem nivoju se nahaja tudi zvočno in grafično ogrodje (ang. media framework) z vključenimi grafičnimi in zvočnimi kodeki, ki morajo biti zelo dobro optimizirani.

Izvajalno okolje je sestavljeno iz Dalvikovega virtualnega stroja (ang. DVM – Dalvik virtual machine) in pa javanskih knjižnic. DVM je preveden v strojno kodo, ki je operacijskemu sistemu najbližja. Služi kot interpreter za izvajanje javanske objektne kode (ang. Java byte code), saj je operacijski sistem ne razume. Tako DVM to kodo prevede v kodo, ki je operacijskemu sistemu razumljiva in hkrati tudi bolj optimalna za delovanje na procesorju mobilnega telefona.

V aplikativnem ogrodju se nahajajo API-ji, ki so prav tako kot aplikacije pisani v javanskem jeziku. Ti so namenjeni aplikacijam in enaki API-ji so na voljo tudi programerjem. Na ta način se programerju, ki razvija aplikacijo, ni potrebno ukvarjati s poznavanjem delovanja raznih komponent na nižjem nivoju, ampak namesto njega za to poskrbijo API-ji. To si lahko predstavljamo tako, da v kodi zahtevamo pridobitev trenutne lokacije in ta preko GPS in API poskrbi za to, da ta informacija »pride do nas«. Zadevo si lahko predstavljamo tudi kot črno škatlo.

Aplikacije se nahajajo na najvišjem – aplikativnem nivoju. Slednji je za uporabnika najpomembnejši, saj so vse akcije, ki jih uporabnik izvaja na telefonu, interakcija z raznimi aplikacijami. Vsaka aplikacija teče v svojem t. i. peskovniku – v »izoliranem« okolju, ki je ločeno od ostalih resursov sistema. Na ta način neka aplikacija ne more posegati v delo oz. pomnilniške lokacije neke druge aplikacije, razen če je ob namestitvi uporabnik dal pravice aplikaciji [3].

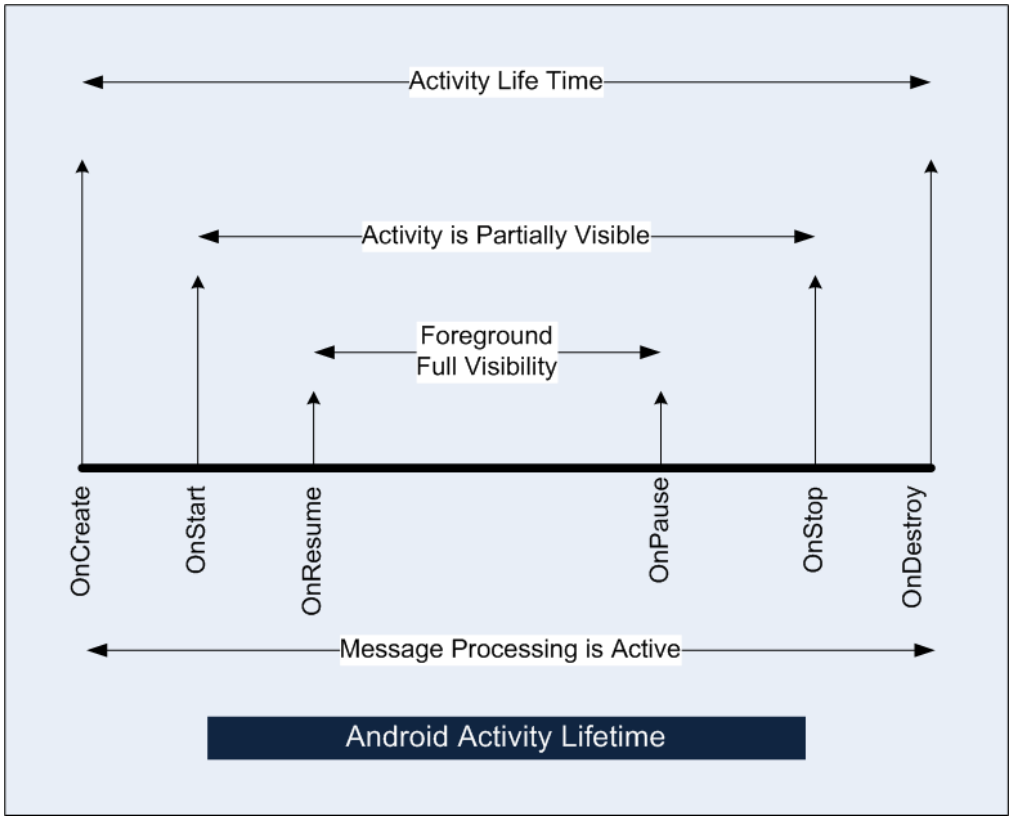
2.1.1 Sestava Android aplikacij

Android aplikacije so sestavljene iz štirih različnih komponent. To so: aktivnosti (ang. activities), storitve (ang. services), sprejemniki (ang. broadcast recievers) in ponudniki vsebin (content providers).

Aktivnosti in njihov življenjski cikel

Za uporabniški grafični vmesnik mora aplikacija vsebovati vsaj komponento aktivnost. Ta predstavlja neko grafično podobo, neke vrste okno, ki ga uporabnik vidi v določenem trenutku. Gre za en sam pogled, kar pomeni, da je aplikacija navadno sestavljena iz več aktivnosti. Tako je na primer okno brskalnika ena aktivnost, okno z nastavitvami pa druga. Med aktivnostmi se preklapljamo s prožilci (ang. intents), ki prikličejo drugo aktivnost. Vse aktivnosti so podrazredi razreda *android.app.Activity* in njihov življenjski cikel je odvisen od

metod tipa onXYZ(), v katerih definiramo, kako naj se aplikacija obnaša ob določenih trenutkih svojega življenjskega cikla.



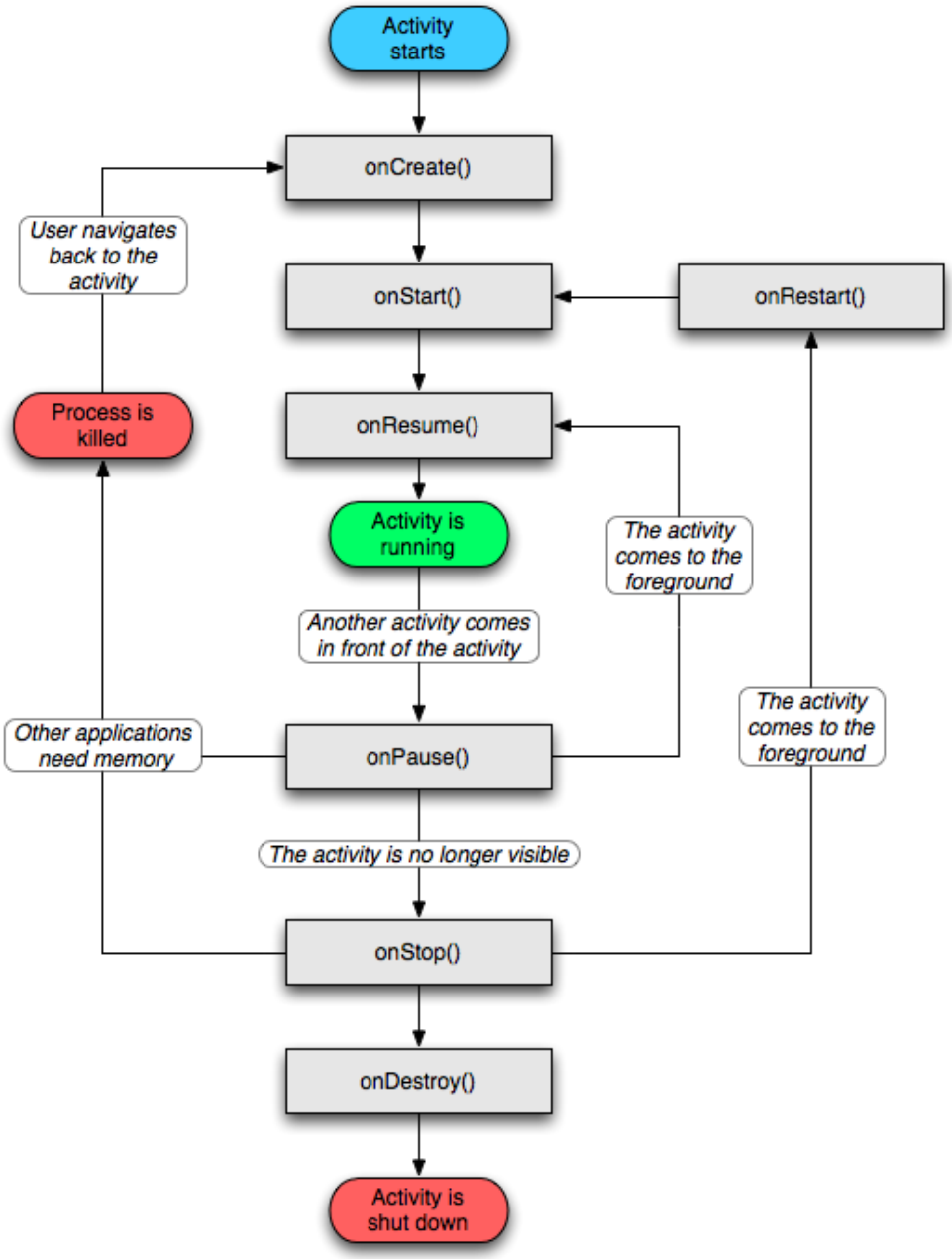
Slika 2: Življenjski cikel aktivnosti

Slika 2 prikazuje omenjene metode tipa onXYZ(), ki so *onCreate, onStart, onResume, onPause, onStop* in *onDestroy.* Vse te metode samodejno pokliče operacijski sistem med izvajanjem aplikacije, ko uporabnik zažene ali zapre aplikacijo, ter takrat, ko uporabnik preklaplja med aktivnostmi.

onCreate metoda poskrbi za inicializacijo aktivnosti. Vse operacije v tem koraku se zgodijo v ozadju in uporabniku niso vidne na ekranu. *onStart* nato privede samo aktivnost v ospredje, torej postane aktivnost uporabniku vidna na ekranu.

Po tej metodi se vedno pokliče tudi metoda *onResume*, ki se v življenjskem ciklu aplikacije lahko pokliče večkrat. Vsakič, ko v ospredje pride neka druga aktivnost in trenutno aktivnost prekrije, se pri prejšnji aktivnosti kliče metoda *onPause*, kjer lahko sprostimo razne resurse (npr. GPS) ali izvedemo druge poljubne akcije. Ko se potem takšna aktivnost vrne v ospredje, se pokliče metoda *onResume*, kjer lahko spet izvedemo poljubne akcije za ponovno aktiviranje aktivnosti. Ravno tako se metodi *onPause* in *onResume* kličeta takrat, ko celotno aplikacijo pošljemo v ozadje, kar se na primer zgodi, ko med rabo aplikacije prejmemo telefonski klic.

onStop metoda se kliče, ko aktivnost umaknemo iz ospredja in tudi iz samega ozadja aplikacije. Za tem se bo nad to aktivnostjo klicala bodisi metoda *onDestroy*, ki bo samo aktivnost zaključila in tudi odstranila njene podatke iz pomnilnika, bodisi metoda *onRestart*, ki bo aktivnost spet priklicala v uporabo.



Slika 3: Življenjski cikel aktivnosti

Storitve in sprejemniki

Storitve in sprejemniki omogočajo aplikaciji izvajanje opravil v ozadju in dodatne funkcionalnosti drugim komponentam. Sprejemnike prožijo dogodki; le-ti se izvajajo samo za kratek čas, medtem ko se storitve lahko izvajajo dalj časa.

Obe omenjeni komponenti sta namenjeni t. i. izvajanjem v ozadju. Gre za operacije, ki se izvajajo ločeno od samega grafičnega vmesnika, torej so uporabniku nevidne. Navadno se ti dve komponenti poganjata kot ločeni niti (ang. thread), saj se v nasprotnem primeru zgodi, da aplikacija postane za čas izvajanja takšnega opravila neodzivna. Če bi bilo takšno opravilo dolgo, bi operacijski sistem aplikacijo »ubil«, ker bi jo imel za neodzivno.

Ponudniki vsebin

Shranjevanje in branje podatkov v Android aplikacijah poteka preko ponudnikov vsebin. Prav tako se jih lahko uporablja za skupno rabo podatkov med dvema ali več aplikacijami. Seveda mora imeti aplikacija za to ustrezne pravice.

Aplikacije nikoli ne dostopajo do podatkov direktno. Vedno namreč za branje ali shranjevanje podatkov uporabijo komponento *ContentResolver*, ki poskrbi za to, da posreduje zahtevo ustreznemu ponudniku vsebin.

Kot primer te komponente lahko vzamemo ponudnika vsebin, ki skrbi za podatke o osebah iz imenika (kontakti). Vsakič, ko v aplikaciji na telefonu dostopamo (beremo ali pišemo) do podatkov iz imenika, potrebujemo prav to komponento.

AndroidManifest.xml

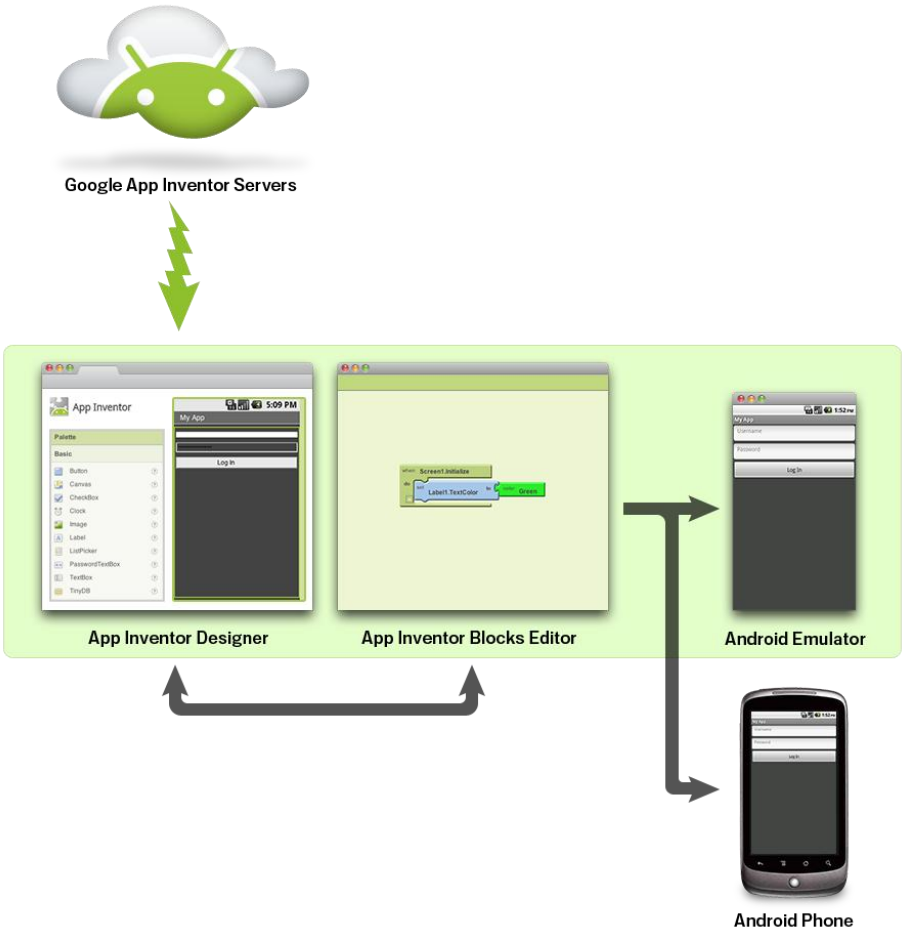
Tukaj velja omeniti tudi datoteko AndroidManifest.xml. Kot že končnica v imenu datoteke pove, gre za XML datoteko, v kateri so zapisani bistveni podatki o sami aplikaciji. Zato da lahko aplikacija uporablja npr. podatke o kontaktih, ima dostop do internetne povezave, uporablja GPS modul, imeti mora za to ustrezne pravice. Te pravice definiramo v datoteki AndroidManifest.xml in ob namestitvi se uporabniku izpišejo vse pravice, ki jih aplikacija potrebuje za svoje delovanje. Ravno tako v njej definiramo tudi pravice, ki jih mora neka druga aplikacija imeti, če hoče dostop do naše aplikacije. Uporabnik ob namestitvi potrdi dodelitev teh pravic aplikaciji. V njej so tudi podatki o aktivnostih, storitvah, sprejemnikih in ponudnikih vsebin, ki sestavljajo aplikacijo. Temu navajanju omenjenih komponent pravimo tudi deklaracija. Komponente, ki niso zajete v datoteki AndroidManifest.xml, sistemu ne bodo vidne in se zato ne bodo zagnale. Poleg tega je v tej datoteki definirana tudi minimalna verzija operacijskega sistema, ki mora biti nameščen na telefonu, da lahko aplikacijo namestimo. V njej je zapisano tudi ime same aplikacije in ikona, ki naj se uporabi za prikaz aplikacije v seznamu aplikacij [4].

2.1.2 Razvoj Android aplikacij

Za razvoj aplikacij za operacijski sistem Android obstajajo razvojni kompleti (ang. development kit), ki vsebujejo knjižnice in druge pripomočke; te omogočajo razvoj ter olajšujejo proces razhroščevanja in simuliranja aplikacije. Najbolj znana razvojna kompleta

sta Android SDK (ang. Android Software Development Kit), ki je namenjen programskemu jeziku Java, ter razvojni komplet Android NDK (ang. Native Development Kit), ki je namenjen programskemu jeziku C.

Razvoj aplikacij večinoma poteka v programskem jeziku Java z uporabo razvojnega kompleta Android SDK, ki je sestavljen iz razhroščevalnika, emulatorja, knjižnic, dokumentacije, primerov izvorne kode aplikacij in raznih vodičev. Poleg tega lahko aplikacije razvijamo tudi v programskem jeziku C s pomočjo Android NDK. Programski jezik C se uporablja predvsem za časovno kritične dele kode. Obstaja tudi razvojno orodje App Inventor for Android, ki je namenjeno ljudem brez znanja programiranja. V njem poteka razvoj aplikacij grafično po principu »povleci in spusti« (ang. drag and drop). Tu aplikacije gradimo s pomočjo miške tako, da komponente iz seznama povlečemo v aplikacijo, ki jo gradimo.

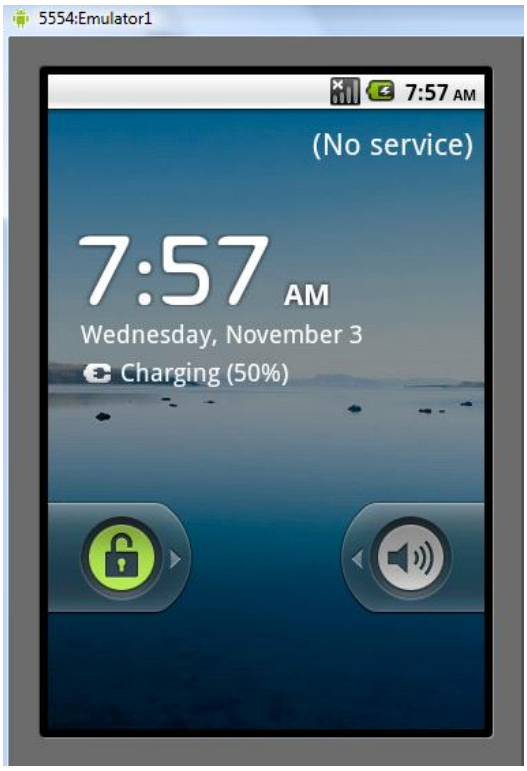


Slika 4: Razvoj Android aplikacij s pomočjo orodja App Inventor

Uradno podprto grafično orodje za razvoj aplikacij z uporabo programskega jezika Java in Android SDK-jem je Eclipse, ki sem ga tudi sam uporabljal med razvojem aplikacij za

diplomsko nalogo. V poglavju »Dodatek A« so zapisana navodila, kako vzpostaviti razvojno okolje za Android aplikacije s pomočjo grafičnega razvojnega orodja Eclipse.

Med samim razvojem lahko aplikacijo testiramo in razhroščujemo s pomočjo emulatorja, ki simulira obnašanje aplikacije na telefonu, ter razhroščevalnika (ang. debugger), kjer lahko v vsakem trenutku spremljamo vrednosti, ki so jih zavzele določene spremenljivke, izpise v dnevniku (ang. log), kjer se izpisujejo morebitne napake, ki so se zgodile oziroma tudi druge informacije, ki služijo predvsem za testiranje. Emulator omogoča konfiguracijo poljubne virtualne naprave (AVD – Android Virtual Device), s pomočjo katere lahko simuliramo izvajanje aplikacije na različnih telefonih z različnimi verzijami operacijskega sistema Android.



Slika 5: Android emulator

Ko smo z narejenim zadovoljni, se izvorna koda prevede in zapakira v t. i. APK datoteko (Android application package). Gre za ZIP format datoteke, ki ima drevesno strukturo na bazi JAR datotek ter končnico .apk. Ko tako datoteko poženemo na telefonu, se zažene namestitveni proces, s pomočjo katerega lahko aplikacijo namestimo.

Tu velja omeniti tudi Android Market, kamor lahko ustvarjene aplikacije naložimo. Od tu si jih lahko naložijo uporabniki s celega sveta oz. samo iz določenih držav, če smo določili tovrstno omejitev. Aplikacijo lahko objavimo na Android Market kot plačljivo ali pa kot prosto dosegljivo. V praksi je tako, da aplikacije, ki so prosto dostopne, vsebujejo mnoga oglasna sporočila, ki se jih »znebimo« z nakupom plačljive verzije. Seveda pa obstajajo tudi

prosto dostopne aplikacije brez oglasov, saj je to odvisno predvsem od tega, kakšne narave je aplikacija, in od avtorja samega, ki aplikacijo objavi.

2.2 Java

Java je objektno usmerjen programski jezik, ki je med najbolj razširjenimi na svetu. Razvil ga je James Gosling v podjetju Sun Microsistems (leta 2010 je podjetje Oracle kupilo podjetje Sun). Uporablja se tako za razvoj namiznih aplikacij kot tudi za razvoj spletnih aplikacij. Princip Jave je bil že od samega začetka, da mora biti neodvisna od platforme, na kateri teče. To pomeni, da je aplikacije narejene v Javi moč poganjati na različnih operacijskih sistemih brez spreminjanja same izvorne kode. Le-ta se namreč prevede v t. i. Java byte code, ki se potem poganja na že prej omenjeni javanski virtualni napravi (JVM). Z verzijo Jave 2 je Java začela izhajati v različnih konfiguracijah, ki so bile namenjene različnim tipom platform. Tako je naprimer Java EE ciljala predvsem na spletne aplikacije, Java ME je bila namenjena mobilnim aplikacijam in Java SE je bila t. i. standardna različica. Poleg tega se Java uporablja tudi pri pisanju kode za Android aplikacije s pomočjo prej omenjenih knjižnic in API-jev, ki jih prinaša Android SDK.

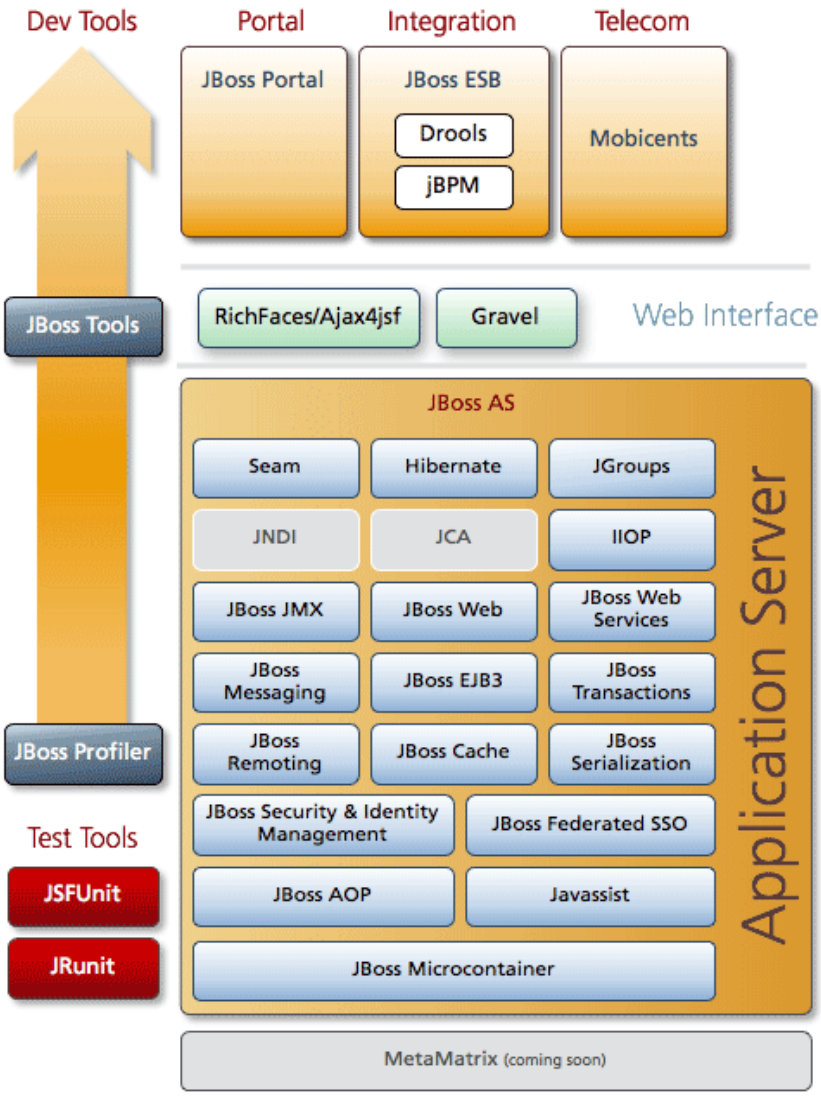
Za diplomsko nalogo sem uporabil Javo za (poleg razvoja Android aplikacije) razvoj aplikacije, ki teče na aplikacijskem strežniku. Konfiguracija, ki sem jo v tem primeru uporabil, je bila Java EE. Gre za aplikacijo, ki izvaja same operacije; slednje služijo za komunikacijo med strežnikom in odjemalci ter bazo.

2.2.1 Aplikacijski strežnik Jboss

JBoss AS je odprtokodni aplikacijski strežnik, ki temelji na JaviEE (Java EE). Je med vodilnimi aplikacijskimi strežniki v Java EE okolju. Ponuja popolno podporo za delo v gruči (clustering support), s čimer lahko povečamo zanesljivost in performanse. To v praksi izgleda tako, da imamo več aplikacijskih strežnikov, ki si med sabo porazdeljujejo delo. Ti so lahko nameščeni na istem računalniku ali na različnih računalnikih. Taka namestitev omogoča tudi nemoteno delovanje v primeru, če eden izmed aplikacijskih strežnikov odpove.

Ima tudi vgrajen Tomcat spletni strežnik, na katerem se izvajajo servleti in JSP strani. Preprosto povedano – omogoča dinamično izvajanje spletnih strani, kar pomeni, da odjemalcu glede na zahteve generira HTML kodo, ki se uporabniku potem prikaže v brskalniku.

Obstaja v dveh različicah, in sicer v prosto dostopni obliki – JBoss Community Editionm, ter v komercialni obliki – JBoss Enterprise Edition.



Slika 6: Arhitektura aplikacijskega serverja JBoss 5

Za izbiro tega aplikacijskega strežnika sem se odločil zato, ker imam z njim že večletne izkušnje. Nanj sem namestil zaledno spletno aplikacijo, ki sprejema in oddaja informacije mobilnima aplikacijama taksista in naročnika.

2.2.2 Apache Maven

Maven je programsko orodje za avtomatizacijo prevajanja izvorne kode. Najpogosteje se ga uporablja v javanskih projektih. Podpira pa tudi nekatere druge programske jezike, kot na primer C#, Scala in Ruby. Pri svojem delu uporablja POM datoteko, ki je praktično XML datoteka. V njej je definiran projekt, in sicer so njegove komponente definirane kot t. i. artefakti. Sami artefakti predstavljajo module, ki imajo lahko med sabo tudi odvisnosti. To pomeni, da en modul uporablja neke komponente iz drugega modula; zato se mora v takšnem

primeru najprej prevesti modul, ki je nekemu drugemu modulu potreben za delovanje le-tega. Prevod besede odvisnost je v angleškem jeziku dependency in enaka beseda se uporablja tudi v sami konfiguraciji v prej omenjeni POM datoteki. Te odvisnosti definiramo tudi za knjižnice, ki jih potrebuje modul. Maven bo sam poskrbel za to, da te knjižnice prenese preko interneta iz t. i. skladišča (ang. repository) in jih lokalno shrani na disk v lokalno skladišče.

Za vsak modul imamo svojo POM datoteko, v kateri so definirane lastnosti samega modula, kot so npr. informacije o tem, kakšno je ime modula, njegova verzija, odvisnosti. Definiramo tudi naslov, na katerem se nahajajo skladišča, od koder lahko potem Maven črpa potrebne artefakte. Poleg tega lahko zapišemo tudi metapodatke, ki so v tem primeru informacije o avtorjih, podjetju itd.

Maven ima tudi vtičnike (plug-in), ki služijo različnim namenom. Primer takšnega vtičnika je recimo maven-war-plugin, ki skrbi za to, da se modul zapakira v t. i. WAR (ang. Web Application Archive). Gre za JAR datoteko s točno določeno strukturo. Podobno velja za EAR (Enterprise Archive) datoteko, v katero se zapakirajo WAR in JAR datoteke. Te potem npr. JBoss aplikacijski strežnik sam razpozna, jih razširi in spletno aplikacijo požene.

Tudi z uporabo Mavena imam večletne izkušnje. Uporabil sem ga pri razvoju zaledne spletne aplikacije, ki je razvita v programskem jeziku Java. V tem primeru Maven ne igra večje vloge, saj gre za manjši projekt, je pa raba tega orodja »dobra praksa«.

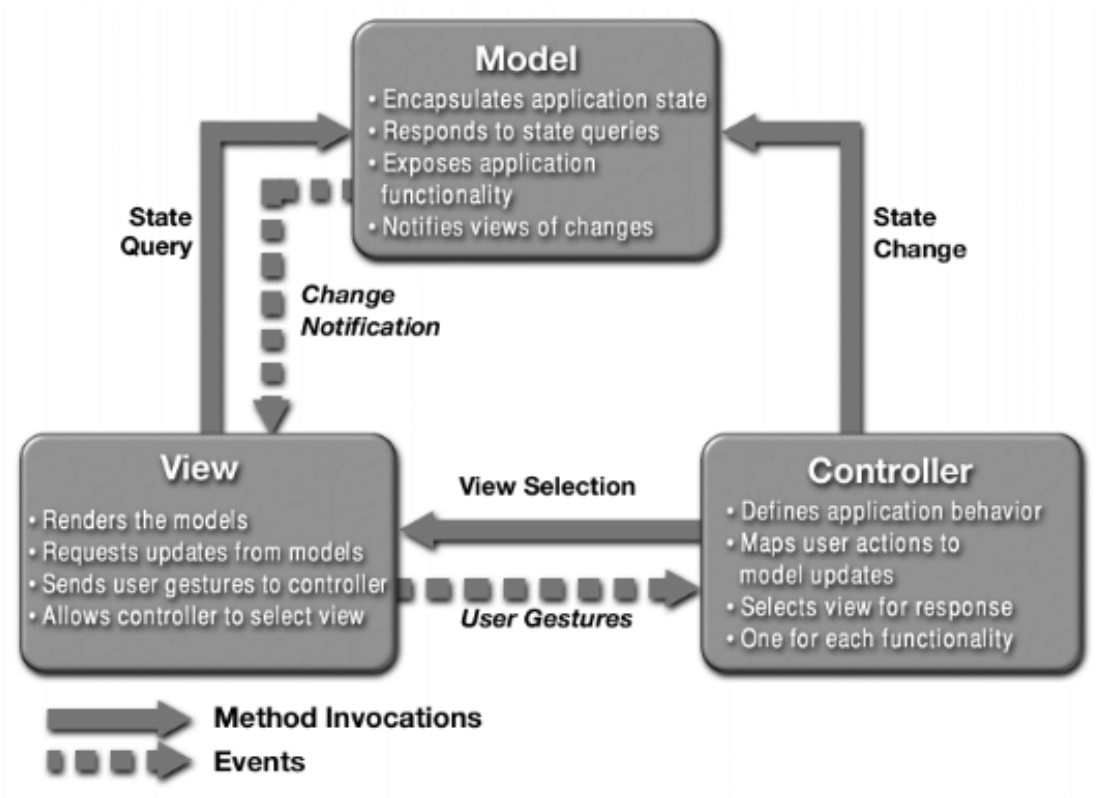
2.2.3 Groovy, Grails

Groovy
Groovy je objektno usmerjen programski jezik in se poganja na javanski platformi (JVM). Zato se tudi prevaja v javansko objektno kodo; za razliko od Jave je Groovy dinamičen jezik s podobnimi lastnostmi, kot jih imajo programski jeziki Ruby, Python in Perl.
Sintaksa tega programskega jezika je veliko bolj kompaktna, kot je sintaksa Jave. Je zanimiv in lažje naučljiv za programerje, ki poznajo programski jezik Java.
Groovy koda se prevede v kodo, ki potem teče na JVM-ju. Dobra lastnost tega je tudi ta, da lahko namesto Groovy sintakse uporabimo kar sintakso programskega jezika Java in jo bo znal prevajalnik uspešno prevesti. Prav tako lahko uporabimo razne javanske knjižnice.

Grails
Grails je odprtokodno spletno aplikacijsko ogrodje, ki je napisano v programskem jeziku Groovy. Narejeno je bilo z namenom ustvariti visoko produktivno aplikacijsko ogrodje za javansko platformo. Deluje po paradigmi »dogovor nad konfiguracijo« (ang. convention over configuration), ki poskuša čim bolj zmanjšati število odločitev, ki jih mora programer sprejemati. S tem se skuša poenostaviti zadeve za programerja, ki bi sicer (npr. pri uporabi Jave) tratil veliko časa z urejanjem raznih konfiguracijskih XML datotek. To pa ne pomeni,

da to ogrodje nima fleksibilnosti. Poleg tega, da nimamo XML konfiguracij, ima še druge izboljšave, ki sam razvoj pohitrijo. Že samo ogrodje namreč vsebuje razvojno okolje in tudi knjižnice, ki so potrebne za razvoj. Vgrajen ima strežnik Apache Tomcat, s pomočjo katerega lahko v fazi razvoja aplikacijo preverjamo. Kodo lahko spreminjamo; le-ta se dinamično prevede in osveži na samem strežniku, zato ne potrebujemo ponovnih zagonov samega strežnika. Še ena zanimiva funkcionalnost je t. i. dinamično dodajanje metod razredom. To pomeni, da lahko izvajamo razne metode, ki jih ni treba implementirati, temveč za to poskrbi ogrodje samo. Tako imajo na primer domenski razredi metode za shranjevanje, brisanje in iskanje dodane po bazi dinamično [9].

Grails ogrodje je bilo načrtovano po paradigmi model, pogled, nadzornik (ang. MVC – model, view controller).



Slika 7: MVC koncept

Kot model se v Grails ogrodju pojavljajo domenski razredi (ang. domain class), ki so praktično Groovy razredi. Vsak izmed njih predstavlja tabelo na podatkovni bazi. Vsebujejo samo deklarirana polja, ki dejansko odražajo polja znotraj tabele na podatkovni bazi. Poleg tega lahko imajo tudi druge definicije, kot na primer omejitve, ki določajo npr. dolžino polj, možnost polja, da zavzame prazno vrednost ipd. V njih lahko definiramo tudi akcije, ki se recimo zgodijo pred ali po vnosu oz. posodobitvi zapisa v neki tabeli. Še ena pomembna stvar, ki jo lahko definiramo, so relacije. Z njimi namreč določimo pripadnost nekega objekta

drugemu objektu in obratno. Tako bi na primer objekt knjiga imel relacijo, ki bi določala, da je knjiga v lasti avtorja, avtor pa bi imel definirano relacijo, ki bi mu določala lastništvo nad nobeno knjigo, eno ali več knjigami.

S pomočjo modela v kodi ni potrebno pisati nobenih SQL stavkov, temveč ustvarimo novo instanco modela, ki mu nato nastavimo vrednosti njegovih polj (atributov). Na koncu kličemo metodo *save*, ki je ena izmed prej omenjenih »dinamično dodanih« metod. Sproži se zapis podatkov na podatkovno bazo. To poenostavi in pohitri razvoj, saj ni treba skrbeti za odpiranje povezave do podatkovne baze, zapisovanja podatkov in zapiranja povezave, ker za nas poskrbi ogrodje samo s pomočjo GORM mapiranja (Grails relacijsko mapiranje objektov). Le-ta uporablja javansko ogrodje Hibernate za objektno relacijsko mapiranje. Objektno relacijsko mapiranje pomeni preslikovanje (shranjevanje) objektov v relacijsko podatkovno bazo.

Vloga nadzornika je izvajanje operacij, ki se izvedejo, preden se dejansko generira HTML koda, ki se prikaže uporabniku. V samem nadzorniku se lahko kličejo razni servisi ali druga zaledna koda, ki lahko izvaja razne operacije. Za primer vzemimo neko zajemanje podatkov iz baze (s pomočjo domenskih razredov). Te podatke nadzornik potem obdela in jih pripravi za prikaz uporabniku. To stori tako, da jih posreduje komponenti, ki ji pravimo pogled.

Pogled potem poskrbi za sam prikaz podatkov tako, da informacije, ki jih posreduje nadzornik, vključi v samo vsebino. Na koncu generira HTML in ga postreže uporabniku, da ga lahko vidi v svojem brskalniku.

Groovy ter Grails sem uporabil pri izdelavi spletne aplikacije, ki teče na spletnem strežniku Tomcat. Spletna stran služi prikazu grafičnih vsebin preko spletnega brskalnika. Za uporabo te tehnologije sem se odločil zato, ker predstavlja novost na področju razvoja spletnih aplikacij.

2.2.5 Apache HTTP strežnik

Apache HTTP strežnik je spletni strežnik, ki skrbi za serviranje HTTP strani. Zahteve, ki preko internetne povezave dospejo do strežnika s strani odjemalcev, so preko njega preusmerjene na pravo mesto, kjer se njihova zahteva sprocesira in jim vrne odgovor v obliki HTML strani. To si lahko predstavljamo tako, da na nekem fizičnem strežniku teče več spletnih aplikacij. Le-te so dosegljive na različnih vratih na lokalnem IP naslovu. Zahteve na Apache strežnik praviloma prihajajo na vrata 80 (oz. 443 za varno povezavo). Apache nato preveri konfiguracijo in zahtevo v skladu s konfiguracijo preusmeri na ustrezna vrata oz. na ustrezno spletno aplikacijo. Ta spletna aplikacija nato vrne odgovor v obliki HTML strani Apacheu, ki ga potem vrne odjemalcu, ki je bil sprožil zahtevo.

Obstaja precej modulov za Apache, ki omogočajo različne možnosti. Primer takega je recimo t. i. mod_auth, ki skrbi za avtentikacijo. S pomočjo tega lahko dostop do neke strani oz.

naslova oz. pot do neke datoteke zaščitimo z geslom. Še en tak primer je modul mod_proxy_balancer, ki skrbi za uravnoteženje obremenjenosti. Ta pride v poštev za spletne aplikacije, ki morajo obdelati veliko število zahtev v kratkem času. Takšne aplikacije poganjamo v gruči na enak način, kot je bilo omenjeno pri aplikacijskem strežniku JBoss. S tem modulom uravnavamo obremenjenost posameznih strežnikov.

Apache strežnik sem uporabil zato, da sprejema zahteve, ki jih posreduje ustreznemu aplikacijskemu oz. spletnemu strežniku. Ti »tečejo« v gruči in Apache dodeli zahtevo najustreznejšemu.

2.2.6 PostgreSQL 9.0

PostgreSQL (tudi Postgres) je zmogljiv odprtokodni sistem za upravljanje s podatkovnimi bazami. Razvija ga skupnost imenovana PostgreSQL Global Development Group. Je brezplačna in odprtokodna aplikacija izdana pod MIT licenco. Poleg podpore SQL jezika ponuja tudi druge funkcije. Primer takšne dodatne funkcionalnosti so recimo sprožilci. To so dogodki, ki jih sproži nek SQL stavek. V takšnem primeru se sprožilec zažene in izvede neke operacije (npr. preveri podatke, naredi operacijo nad podatki, preden jih zapiše itd.). Postgres ponuja tudi rabo tujih ključev, pogledov, indeksov itd.

Ima vgrajen tudi proceduralni jezik, ki se imenuje PL/pgSQL, in je primerljiv z Oraclovim PL/SQL proceduralnim jezikom. Služi izvajanju raznih procedur na sami podatkovni bazi.

Za podatkovno bazo sem izbral Postgres, ker imam s to tehnologijo izkušnje. Je med resnijšimi konkurenti »večjim igralcem« na tem področju (Oracle, IBM).

2.2.7 Amarino

Ime Amarino izhaja iz besedne zveze »Android meets Arduino«, ki pomeni »Android spozna Arduina«. Arduino je razvojna ploščica in prav tako se imenuje tudi programski jezik; uporablja se za programiranje mikrokrmilnika, ki se nahaja na ploščici.

Amarino služi lažjemu ustvarjanju takšnih aplikacij za Android operacijski sistem, ki se povezujejo preko Bluetooth povezave na Arduino ploščice. Sestavljajo ga Android aplikacija in knjižnice. Knjižnice imamo tako za razvoj aplikacij na Android operacijskem sistemu kot tudi za Arduino programski jezik. Slednje se uporabijo za lažje pošiljanje signalov na napravo, ki uporablja Android operacijski sistem oz. za prejemanje podatkov s takšne naprave. Imenuje se MeetAndroid.

Amarino sem uporabil pri komunikaciji med razvojno ploščico Arduino in mobilno aplikacijo taksista. Uporabil sem ga, ker mi je omogočil hiter in enostaven razvoj komunikacije med mobilno aplikacijo in razvojno ploščico.

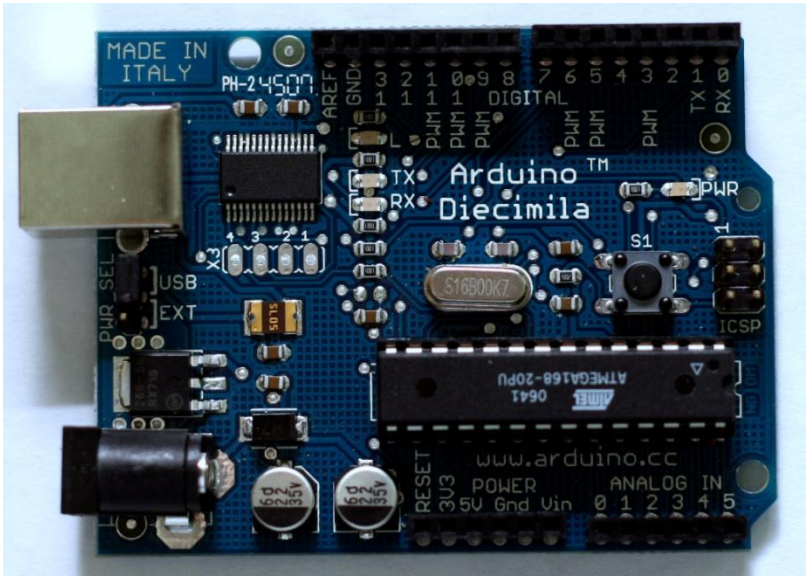
3 UPORABLJENA STROJNA OPREMA

Za izdelavo v uvodu opisane rešitve sem potreboval poleg naštetih programskih tehnologij tudi strojno opremo. Potreboval sem telefone z operacijskim sistemom Android, strežnik ter razvojno ploščico Arduino. Na slednjo sem povezal Bluetooth modul – BlueSmirf. Poleg tega sem potreboval tudi na dotik občutljive upore, ki so prav tako povezani na razvojno ploščico.

3.1 Arduino Diecimila

Arduino je enostavna odprtokodna platforma, ki je primerna za preproste in tudi za zahtevnejše projekte.

Ploščica ima 16 KB flash pomnilnika, od tega 2 KB zavzema t. i. zaganjalnik (bootloader), ki skrbi za izvedbo začetnih ukazov; ti začnejo z branjem ukazov in z izvajanjem le-teh. Zažene se takrat, ko ploščico priklopimo na napajanje, ali pa takrat, ko pritisnemo tipko reset. Poleg tega ima tudi 1 KB SRAM in 512 B EEPROM pomnilnika, v katerega lahko pišemo s pomočjo EEPROM knjižnice. Frekvenca ure znaša 16 MHz. Ima 14 digitalnih vhodov/izhodov ter 6 analognih vhodov. Vsakega od 14 vhodov/izhodov lahko namreč uporabimo kot vhod ali izhod. To naredimo programsko s pomočjo funkcij *pinMode, digitalWrite* ter *digitalRead*. Od tega sta priključka 0 in 1 posebna. Uporablja se ju za sprejemanje (RX) in pošiljanje (TX) TTL serijskih podatkov. Priključka 2 in 3 lahko skonfiguriramo tako, da prožijo prekinitve, in sicer ob padcu napetosti, ob povečanju napetosti ali pa ob spremembi.



Slika 8: Razvojna ploščica Arduino Diecimila

Za razvojno ploščico Arduino sem se odločil, ker je cenovno ugodna in zmogljiva razvojna ploščica.

3.2 Bluetooth modul – BlueSmirf

BlueSmirf je Bluetooth modul, ki je namenjen brezžični komunikaciji preko Bluetooth povezave. Lahko bi rekli, da nadomešča klasično serijsko povezavo, ki sicer poteka preko serijskega kabla. Gre za Bluetooth modul 1. razreda, ki ima preko 100 metrov dometa. Ima zelo nizko porabo – povprečno 25 mA. Deluje na frekvenci od 2,4 do 2,524 GHz in pri napetosti od 3,3 V do 6 V. Ima vgrajeno anteno, komunikacija je kriptirana. Serijska komunikacija poteka s hitrostjo od 2400 bit/s do 115200 bit/s.

Ima 6 priključkov, ki služijo za napajanje, pošiljanje in sprejemanje podatkov in kontrolo pretoka podatkov. Priključka za kontrolo podatkov imata oznako CTS (ang. clear to send) in RTS (ang. request to send). Priključka za sprejem in oddajanje imata oznako RX in TX. Preostala dva priključka služita za napajanje in imata oznako VCC in GND [6].

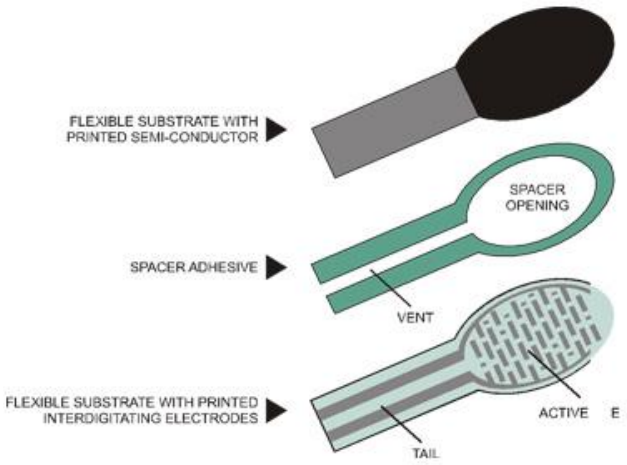
Bluetooth modul BlueSmirf sem uporabil za povezavo mobilne aplikacije z razvojno ploščico Arduino.

Slika 9: Bluetooth modul BlueSmirf

3.3 Na dotik občutljivi upori

To so upori s spremenljivo upornostjo. Ta se spreminja glede na pritisk, ki ga izvajamo na njihovo površino. Zaradi te lastnosti jih nekateri uvrščajo tudi med senzorje. Navadno niso zelo natančni, zato jih ne moremo uporabljati za npr. tehtanje. Razlika v natančnosti med dvema tovrstnima uporoma lahko znaša tudi 10 %. So pa prava izbira, če želimo samo zaznati, ali se nad njimi izvaja nek fizični pritisk ter kolikšna je njegova intenzivnost. Zgrajeni so iz treh plasti. Srednja plast je zlepljena na zgornjo in spodnjo plast s tem, da ti dve plasti

med sabo loči tako, da se ne dotikata. Plast izgleda kot nekakšen okvir, na katerem je nameščeno lepilo. V sredini je prostor, kjer se zgornja in spodnja plast stikata, ko pritisnemo na sam upor. Spodnja plast je tista, na kateri sta nameščeni elektrodi; povezavi med njima sta prekinjeni tako, kot prikazuje Slika 9. S pritiskom na upor pritisnemo zgornjo plast, na kateri se nahaja polprevodnik, na spodnjo plast in tako sprožimo pretok toka. Bolj kot pritisnemo, več povezav povežemo in na ta način manjšamo upornost. Na začetku je torej upornost neskončna, kar si lahko predstavljamo kot odprte sponke.



Slika 10: Zgradba na dotik občutljivega upora

Uporabil sem jih za zaznavanje prisotnosti potnikov v vozilu. Nameščeni so v sedežih in so povezani z razvojno ploščico Arduino.

3.4 Telefon z operacijskim sistemom Android in GPS sprejemnikom

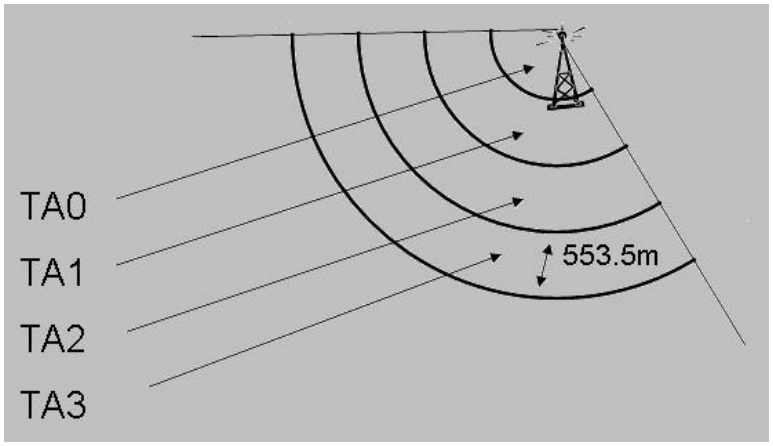
Uporabil sem mobilni telefon HTC Desire, ki ima nameščen operacijski sistem Android verzije 2.2. Vgrajen ima mikroprocesor Qualcomm QSD8250 Snapdragon s frekvenco 1 GHz. Vgrajena ima GPS modul in tudi Bluetooth modul. Poleg tega ima tudi WiFi sprejemnik.

3.4.1 Določanje lokacije s pomočjo GPS ali omrežja (mobilnega/WiFi) na mobilnem telefonu

Naprave, v našem primeru telefoni, ki imajo vgrajene GPS module, znajo s pomočjo signala GPS satelitov določiti svojo lokacijo. Takšne naprave ne oddajajo nikakršnega signala, temveč ga samo sprejemajo. Vsak od omenjenih satelitov ima vgrajeno atomsko uro, ki zagotavlja visoko natančnost in oddajajo informacij o svoji lokaciji ter trenutni čas. Sateliti so

med sabo sinhronizirani tako, da vsi oddajajo te informacije ob enakem času. Signali potujejo od satelitov do uporabnika s svetlobno hitrostijo in dospejo do GPS sprejemnika ob različnem času; ta je odvisen od razdalje med satelitom in GPS sprejemnikom. Ko sprejemnik lahko določi razdaljo med sabo in še vsaj štirimi drugimi sateliti, zna izračunati svojo lokacijo v tri-dimenzionalnem prostoru. Zato telefon za sprejem signala ne sme imeti nobenih ovir nad sabo. Če signala nima, kar je pogosto posledica tega, da se uporabnik nahaja v zaprtem prostoru, kjer ni dosegljiv GPS signal, aplikacija poskuša določiti lokacijo s pomočjo t. i. omrežnih lokacij (ang. network location). Gre za metodo določanja pozicije omrežja, ki bazira bodisi na WiFi omrežjih bodisi na mobilnih omrežjih. Vsak GSM oddajnik ima namreč svojo identifikacijsko številko, s pomočjo katere lahko telefon, v tem primeru preko Googlovih spletnih storitev, razpozna tudi lokacijo oddajnika. S tem se lahko določi nek radij, ki pove, kje se trenutno nahajamo. Le-ta je z večanjem števila GSM oddajnikov v okolici čedalje natančnejši, saj se mobilni telefon poveže na tisti oddajnik, ki mu je najbližji. Samo oddaljenost od oddajnika mora poznati telefon, saj je to zahteva za uporabo metode dostopa TDMA, ki jo uporabljajo mobilni telefoni. Metoda deluje po principu časovnih rezin, kar pomeni, da morajo priti paketi v neki določeni časovni rezini, zato mora telefon poznati oddaljenost, da lahko oddajanje primerno prilagodi (zakasni), pošiljanje signalov.

S tem ko poznamo TA (oddaljenost od bazne postaje, ang. »timing advance«) čas treh ali več GSM oddajnikov, lahko določimo s pomočjo triangulacije zelo natančno lokacijo, kjer se nahajamo. Če je GSM oddajnikov manj, pri določanju lokacije ne moremo več zagotoviti neke večje natančnosti.



Slika 11: Oddaljenost od oddajnika

V primeru WiFi omrežij telefon ne zna sam določiti svoje trenutne lokacije, ampak mu pri tem v celoti pomaga Googlova spletna storitev. Google namreč shranjuje lokacijo uporabnika, če jo telefon pozna, in jo tudi mapira na WiFi omrežje, če smo nanj povezani. Če imamo v nastavitvah na telefonu omogočeno določanje lokacije s pomočjo WiFi ali mobilnih omrežij, se strinjamo s tem, da Google od nas lahko zajema podatke, ki omogočajo določanje lokacije. S tem Google pridobi informacijo o tem, kjer se nahaja določeno WiFi omrežje, ki definira nek IP naslov, ime omrežja ter MAC naslov usmerjevalnika, ki je unikaten. Lokacijo pa

pridobi po zgoraj omenjenih metodah, torej s pomočjo GPS, če je na voljo GPS signal, ali pa s pomočjo triangulacije GSM oddajnikov [2].

3.5 Strežnik

Za strežnik sem uporabil sem uporabil računalnik z naslednjimi tehničnimi specifikacijami:

- procesor Intel Core i7-980, 3.33GHz, Hexa-core
- pomnilnik DDR3 12GB PC1600
- trdi disk 3 x 500GB, RAID 5

Na strežnik sem namestil CentOS 6.0 64-bit operacijski sistem. Namestil sem tudi aplikacijski strežnik JBoss AS 5 ter Apache Tomcat 7.0.22. Na slednjem teče spletna aplikacija napisana v ogrodju Grails. Poleg naštetega sem namestil tudi podatkovno bazo PostgreSQL 9.0 in Apache HTTP strežnik. Le-tega sem skonfiguriral tako, da za zahteve, ki prispejo na naslov, ki se začne z/taxi preusmeri na aplikacijski strežnik JBoss, ostale zahteve pa preusmeri na spletno aplikacijo, ki teče na Tomcatu.

4 OPIS REŠITVE IN POSTOPKI IZVEDBE

4.1 Aplikacija za naročanje taksi prevoza

4.1.1 Opis rešitve

Aplikacija je namenjena uporabnikom s pametnim telefonom, ki ima nameščen operacijski sistem Android. Ko aplikacijo zaženejo, začne aplikacija s pridobivanjem trenutne lokacije uporabnika s postopkom, ki je opisan v poglavju 3.4.1. Na ta način dobimo v sami aplikaciji koordinate, in sicer zemljepisno širino in zemljepisno dolžino (ang. latitude, longitude). Uporabniku se izriše zemljevid, ki prikazuje okolico, kjer se trenutno nahaja. Center zemljevida je postavljen na mesto, kjer se uporabnik nahaja; na tem mestu je na zemljevidu postavljena oznaka (ang. placemark), ki nakazuje lokacijo, kot jo je določil telefon. Uporabnik ima nato možnost premika oznake na poljubno lokacijo. Oznaka dejansko ponazarja mesto, kamor želimo naročiti taksi. Ob premikanju same oznake aplikacija razpozna nove koordinate in s pomočjo Googlove storitve, ki ji posredujemo zemljepisno širino in dolžnino, vrne ulico za podane koordinate. Ko je uporabnik nastavil želeno mesto naročila, pritisne na gumb *naroči*. Aplikacija ga opozori, da bo njegova akcija sprožila naročilo in ga vpraša za dodatno potrditev. Ko uporabnik naročilo potrdi, se zahteva pošlje na strežnik. Uporabniku se nato prikaže nov pogled, kjer se izpiše, da je bilo naročilo oddano in le-to čaka na potrditev. Ko se stanje naročila spremeni, je uporabnik o tem obveščen, in sicer mu telefon zavibrira ter se na ekranu prikaže sporočilo o stanju naročila. Le-to je lahko bodisi uspešno bodisi neuspešno. Če je naročilo neuspešno, se uporabniku ponudi možnost za ponoven poizkus naročila, ki je vbistvu ponovljen (tukaj opisan) postopek. Razlika je v tem, da se uporabniku oznaka postavi na tisto mesto, ki ga je določil pri oddanem neuspešnem naročilu. V nasprotnem primeru je uporabnik seznanjen s tem, da je naročilo prišlo do taksi voznika, ki bo prišel na želeno lokacijo.

Slika 12: Aplikacija za naročilo taksi prevoza

4.1.2 Razvoj aplikacije

Razvoj aplikacije je potekal v grafičnem razvojnem orodju Eclipse s pomočjo Android SDK knjižnic in API-jev. Ob kreiranju projekta sem določil verzijo SDK-ja, in sicer sem le-to nastavil na vrednost 7, kar pomeni, da jo lahko namestimo na vsak telefon, ki ima nameščen Android verzije 2.1 ali več.

Zgradba

Aplikacija je zgrajena iz dveh aktivnosti ter drugih pomožnih razredov. Vsaka aktivnost predstavlja en pogled aplikacije. Prvi je pogled, ki ga prikazuje Slika 10, torej glavno okno same aplikacije, kjer uporabnik izvede naročilo. Druga aktivnost predstavlja pogled, kjer se uporabniku sporoči, da je bilo njegovo naročilo poslano. V isti aktivnosti dobi uporabnik tudi obvestilo o uspešnosti oz. neuspešnosti svojega naročila. Poleg razredov sestavljajo aplikacijo tudi konfiguracijske XML datoteke ter slike.

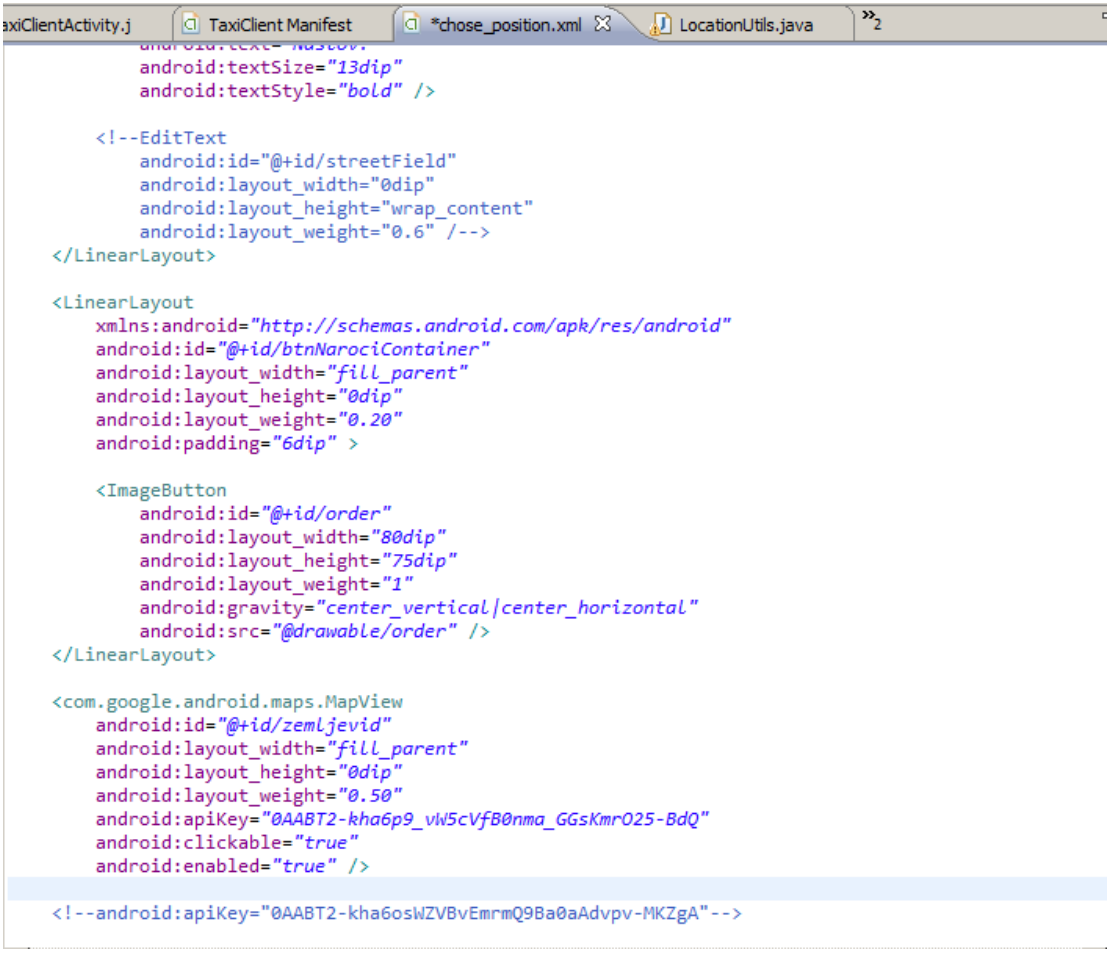
MapActivity, Google Maps (zemljevidi)

Aktivnost, ki na zaslonu prikazuje zemljevid, je javanski razred, ki razširja razred *MapActivity*. Tega moramo v naši aktivnosti nujno razširiti, če želimo uporabljati Googlovo storitev Google Maps (zemljevid). To pa še ni dovolj, da lahko storitev uporabljamo. Preden lahko dostopamo do podatkov, ki nam jih le-ta nudi, moramo na spletni strani http://code.google.com/android/maps-api-signup.html registrirati certifikat, s katerim bomo

našo aplikacijo tudi podpisali. To storimo tako, da na omenjeni spletni strani vnesemo MD5 »prstni odtis« (ang. MD5 fingerprint) certifikata v ustrezno polje in se strinjamo s pogoji uporabe storitve. Nato se generira ključ, ki ga vnesemo v aplikacijo, kar bo omogočilo uporabo Google Maps storitve.

MD5 »prstni odtis« sem dobil s pomočjo uporabe orodja KeyTool GUI, s katerim sem odprl certifikat in podatke prepisal. Omeniti je treba, da moramo za prikaz zemljevidov v emulatorju vzeti podatke iz certifikata, ki se nahaja v .android mapi. Ta se nahaja v uporabniških direktorijih. Na Linuxu bi to bilo npr. v direktoriju »/home/matej/.android/«, na Windows operacijskemu sistemu pa v direktoriju »C:/Users/Matej/.android/«. Ime certifikata pa je debug.keystore in je to privzeti certifikat, ki se ga uporablja za poganjanje aplikacije v t. i. razhroščevalnem načinu.

Ključ, ki sem ga dobil, je takšen: »0AABT2-kha6p9_vW5cVfB0nma_GGsKmrO25-BdQ«. Vnesemo ga v konfiguracijsko XML datoteko, ki skrbi za postavitve elementov na sami aktivnosti. V njej definiramo, kje naj se neka komponenta nahaja, kakšne naj bodo njene dimenzije in podobno. Del te datoteke prikazuje Slika 11, kjer lahko vidimo v zadnji gruči kode zgoraj zapisan ključ. »*com.google.android.maps.MapView*« pa ponazarja komponento Google Maps, torej zemljevid. Ostale vrstice določajo njegovo širino, ki je v tem primeru določena kot »*fill_parent*«, kar pomeni, da se komponenta razširi do svoje nadrejene komponente, ki je v tem primeru celotno okno, višina je definirana kot 0dip. Dip (density-independent pixels) je merska enota, ki je bazirana glede na gostoto pikslov na zaslonu. To ne pomeni, da bo višina enaka nič, temveč se bodo komponente s to višino tako porazporedile, da bodo med sabo porabile enako prostora (bodo imele enako višino). Poleg tega lahko vidimo tudi to, da je zemljevid aktiviran (*enabled = »true«*) in nad njim lahko izvajamo operacije – pritiskamo (*clickable=»true«)*.



Slika 13: XML datoteka za prikaz glavne aktivnosti

Pravice

Aplikaciji sem moral dodati tudi ustrezne pravice, da lahko uporablja komponente potrebne za delovanje. Pravice se doda tako, da se v datoteko AndroidManifest.xml zapiše po eno vrstico konfiguracije za vsak modul, ki ga aplikacija potrebuje. Ena vrstica je videti takole: *<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>*
Pravice, ki sem moral dodati:

- *ACCESS_FINE_LOCATION* – pravica za določanje lokacije
- *INTERNET* – pravica za uporabo internetne povezave
- *VIBRATE* – pravica za uporabo vibracijskega zvonjenja telefona
- *ACCESS_NETWORK_STATE* – pravica za preverjanje razpoložljivosti internetne povezave
- *READ_PHONE_STATE* – pravica za branje podatkov telefona

Pomembne metode in njihovo delo

Pomembnejše metode v aplikaciji so: metoda za pridobivanje lokacije, metoda za oddajanje naročila ter metoda za zajemanje statusa naročila.

Metoda za pridobivanje lokacije se imenuje *getPosition.* Ne sprejema nobenih parametrov, vrača pa dva, in sicer x in y koordinati, ki predstavljata trenutno lokacijo. Izvede se tako, da se najprej preveri, ali so naprave za določanje lokacije sploh vključene. Če je odgovor negativen, potem sproži napako, ki uporabnika obvesti o tem, da mora biti paketni prenos podatkov ali brežično omrežje in GPS sprejemnik vključeni. V nasprotnem primeru izvedemo iskanje lokacije, kot je že bilo opisano. Uporabimo razred LocationListener, ki mu moramo implementirati metodo *onLocationChanged*. Metoda sprejema parameter, in sicer razred *Location*. Tega mu avtomatsko posreduje API, ko zazna, da je pridobljena nova lokacija. V omenjeni metodi določimo, kaj naj se zgodi ob dogodku prejema nove lokacije. Možnost imamo tudi preverjati natančnost dobljene lokacije, in sicer s klicem metode *getAccuracy* nad objektom tipa *Location.* Če je natančnost dovolj dobra, lahko z metodo *getPosition* zaključimo in pokličemo metodo za pridobivanje ulice s pridobljenimi parametri. To dobimo tako, da na objektu tipa *Geocoder* kličemo metodo *getFromLocation*, ki sprejme tri parametre. To so x in y koordinata ter maksimalno število vrnjenih rezultatov.
Dobljene podatke ob naročilu aplikacija pošlje na strežnik. Med podatki o sami lokaciji pošlje tudi identifikacijsko številko naprave (v tem primeru telefona). Na ta način se lahko vodi evidenco o strankah in o zlorabah. Uporabnike, ki zlorabljajo aplikacijo, se s pomočjo tega podatka blokira.

Ko je naročilo poslano, aplikacija izvaja periodična povpraševanja na strežnik. Ob oddaji naročila prejme unikatno številko naročila, s pomočjo katere dela poizvedbe na strežnik. Ko dobi potrjeno ali zavrnjeno naročilo, se zapiše ustrezno stanje na tekstovno polje v oknu.

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
<order_id>9df41a84-cc8b-41c6-8a28-493455b92332</order_id>
</order>
```

Slika 14: Prejet unikaten id naročila ob oddaji le-tega

4.2 Aplikacija, ki jo uporabljajo taksisti

4.2.1 Opis rešitve Android aplikacije

Izdelana rešitev za taksiste je sestavljena iz dveh delov. Prvi del je programski in temelji le na programu, ki se izvaja na telefonu. Drugi del pa predstavlja vezje, ki služi detekciji sopotnikov v avtomobilu. Vezje je sestavljeno iz razvojne ploščice Arduino, na katero so med sabo vzporedno vezani povezani upori, ki so občutljivi na dotik ter Bluetooth modul. To vezje nato preko Bluetooth povezave komunicira z aplikacijo, ki teče na telefonu. Vezje deluje tako, da s pomočjo uporov občutljivih na dotik, ki jih vgradimo v sedeže, zazna prisotnost osebe. Ta informacija se nato prenese do razvojne ploščice. Od tu gre informacija naprej preko Bluetooth modula do telefona, kjer informacijo dobi aplikacija in jo pošlje naprej na strežnik.

Aplikacija na telefonu služi temu, da podaja informacijo o trenutni lokaciji taksija ter o njegovi razpoložljivosti.

Slika 15: Android aplikacija za taksiste

4.2.2 Razvoj Android aplikacije

Zgradba

Tudi ta Android aplikacija ima dve aktivnosti. Poleg tega ima še druge razrede, ki so v pomoč tema dvema aktivnostma.

Prva aktivnost predstavlja zaslonsko masko, ki vsebuje tri večje gumbe. Gumbi skupaj zajemajo 80 % razpoložljivega zaslona, ostalih 20 % pa zajema prostor, kamor se zapisuje trenutno stanje (prost/zaseden/neaktiven). Gumbi so obarvani – prvi je rdeč, drugi zelen in tretji siv. Ob pritisku na zeleni gumb bo aplikacija na strežnik pošiljala podatek, da je taksi prost. To sicer ne velja v primeru, ko senzorika v sedežih zazna potnike. Pritisk na rdeči gumb nastavi trenutno stanje na zasedeno, medtem ko pritisk na sivi gumb sporoči na strežnik, da je taksi neaktiven in potem preneha s pošiljanjem nadaljnih podatkov na strežnik.

Druga aktivnost je namenjena prejemanju prejemanju informacije o novem prispelem naročilu. V takšnem primeru se odpre nova aktivnost, ki prikazuje zemljevid s trenutno lokacijo taksija in z lokacijo uporabnika, ki je oddal naročilo. Poleg tega se izpiše tudi ulica, od koder prihaja naročilo. Taksist ima na voljo dve možnosti. Naročilo lahko potrdi ali pa ga

zavrne. Če ga zavrne, bo naročilo posredovano drugemu taksistu. Če pa ga sprejme, bo aplikacija njegov status avtomatsko spremenila v zaseden ter bo na strežnik sporočila informacijo o tem, da je naročilo potrjeno.
Sama uporaba zemljevida je enaka kot pri aplikaciji za stranke. Tudi sam ključ za uporabo storitve Google Maps je isti.

Poleg aktivnosti imamo tudi druge komponente, ki služijo za samo pošiljanje podatkov, za komunikacijo z Arduino ploščico preko Bluetooth povezave itd.

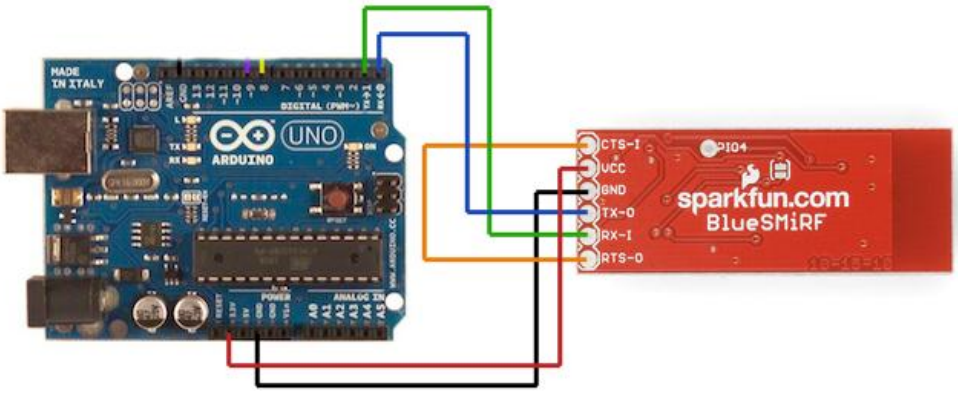
Pravice
Uporabljene pravice so enake kot pri aplikaciji namenjeni strankam.

Pomembne metode in njihovo delo
Tudi v tem primeru se uporablja metoda za določanje trenutne lokacije. Deluje po podobnem principu kot tista pri uporabniku. Edina razlika je tukaj ta, da se v tej aplikaciji določa lokacija samo s pomočjo GPS modula. Če telefon nima GPS signala, se podatki o lokaciji ne pridobijo. Lokacija taksija mora biti točna. Stranka ima možnost ročno spreminjati želeno mesto, kamor naj taksi pride.
Metoda *sendRequest* poskrbi za to, da se podatki pošiljajo na strežnik preko internetne povezave. Komunikacija poteka preko spletnih storitev (ang. web-services). Ker je protokol, po katerem potekajo informacije tipa HTTPS (varna povezava) in je certifikat na strežniški strani »domače izdelave«, sem moral spisati svoj razred, ki razširja razred *SSLSocketFactory*. Če namreč certifikat ni podpisan s strani verodostojnega overitelja (npr. Thawte, VeriSign ...), mu aplikacija ne zaupa in se sproži napaka. Z uporabo svojega razreda pa sem obšel to težavo. Seveda je pa najbolje imeti verodostojen SSL certifikat.

4.2.3 Vezje za detekcijo zasedenosti

Sestava
Vezje je sestavljeno iz na pritisk občutljivih uporov, razvojne ploščice ter Bluetooth modula.

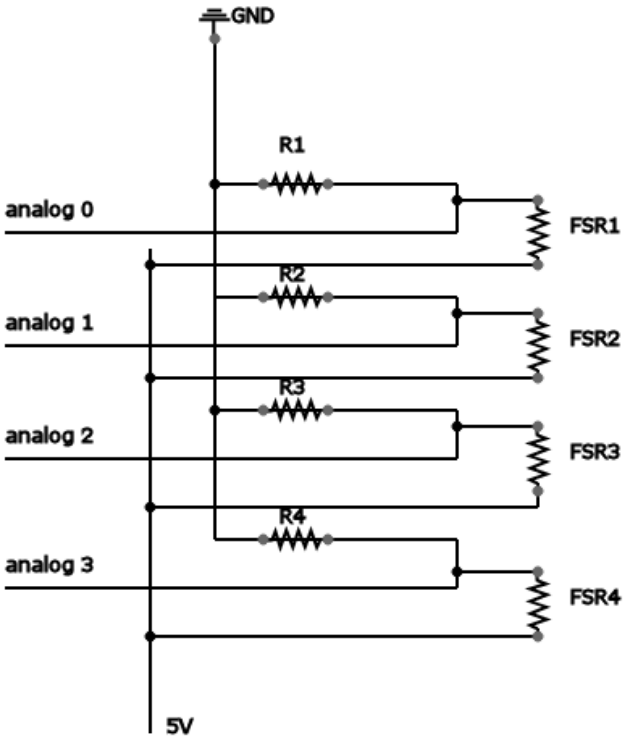
Bluetooth modul je na razvojno ploščico povezan tako, da ima RX priključek povezan na TX priključek ploščice in RX priključek na TX priključek ploščice. CTS in RTS priključka modula sta zvezana skupaj, VCC priključek pa je vezan na 5 V priključek ploščice in GND priključek je vezan na istoimenski priključek na ploščici.



Slika 16: Vezava Bluetooth modula na Arduino ploščico

Na dotik občutljivi upori so med sabo vzporedno vezani v mrežo, ki je vgrajena v sedežih. Za 4 sedeže imamo 4 takšne mreže, vsaka ima po 4 tovrstne na dotik občutljive upore. Vsaka takšna mreža je vezana na analogne vhode razvojne ploščice. Ob spremembi napetosti se le-ta sproži in zazna na razvojni ploščici. Ob detekciji se zažene procedura, ki na telefon pošlje ukaz. Telefon prejme obvestilo o informaciji in o tem obvesti ploščico, da je informacija prispela. Če ploščica ne prejme te informacije, jo poskuša periodično poslati na telefon, dokler ne dobi potrditve o sprejemu. Ko dobi potrditev, preneha z oddajanjem, dokler ne pride do spremembe stanja.

Slika 17 prikazuje priključitev uporov na razvojno ploščico. FSRx predstavlja nadomestni upor za 4 vzporedno vezane na dotik občutljive upore. Torej je vsak tak FSRx vgrajen v sedežu sopotnikov v taksi vozilu; na ta način imamo na razvojni ploščici tudi informacijo o tem, kateri sedež je zaseden oz. koliko jih je zasedenih, če jih je več.



Slika 17: Vezava uporov za 4 sedeže

Program na mikrokrmilniku

Programsko moramo najprej nastaviti hitrost prenosa, da bo le-ta enotna z Bluetooth modulom. Na obeh sem jo nastavil na 9600bps. To na Arduinu programsko naredimo z ukazom *Serial.begin(9600).*

Glavno vlogo ima funkcija loop, ki se neprestano izvaja. V njej v vsaki iteraciji preverjamo, kakšno je stanje na analognih vhodih, kar izvedemo s pomočjo funkcije *analogRead(pinNumber),* kjer je *pinNumber* številka analognih vrat (0, 1, 2, 3). Ko preberemo vrednosti analognih vrat, primerjamo prejšnje stanje s trenutnim; če je prišlo do spremembe, sporočimo spremembo telefonu preko Bluetooth modula. To pa naredimo s pomočjo funkcije, ki se nahaja v knjižnici MeetAndroid, in sicer je to funkcija *meetAndroid.send(newState).* newState je v tem primeru novo stanje. Čez lahko pošljemo karkoli v tekstovni obliki. Primer pošiljanja statusa zaseden bi lahko bil klic *meetAndroid.send(»ZASEDEN«)*, v mojem primeru pa sem namesto »ZASEDEN« uporabil kar številko 1. Če aplikacija na telefonu preko Bluetootha prejme številko 1, ji to pove, da vsaj na enem od sedežev nekdo sedi. V nasprotnem primeru prejme 0, kar ponazarja, da v avtomobilu ni potnikov.

4.3 Zaledna aplikacija na strežniku

4.3.1 Opis zaledne aplikacije

Zaledna aplikacija, ki teče na strežniku, povezuje prej omenjeni aplikaciji. Obe namreč vse svoje informacije pošiljata na strežnik, kjer strežnik podatke zajame in jih shrani na podatkovno bazo. Na strežniku se hranijo vse informacije o lokacijah in o stanju samih taksijev ob vsakem času. Ko uporabnik želi naročiti taksi, prejme zaledna aplikacija zahtevo, ki vsebuje samo lokacijo uporabnika ter njegov unikaten id. Aplikacija najprej na bazi preveri, če je uporabnik morda dodan na seznam takih uporabnikov, ki so aplikacijo zlorabljali. Če ga tam ne najde, bo njegovo zahtevo sprocesirala, kar poteka tako, da aplikacija preveri trenutno proste taksije in njihove lokacije. Tistemu, ki se nahaja najbližje lokaciji, od koder prihaja zahteva, nato aplikacija pošlje naročilo. Odločitev taksista se nato spet posreduje na zaledno aplikacijo, ki njegovo odločitev zabeleži na podatkovno bazo. Če taksist naročilo zavrne, aplikacija nadaljuje izvajanje in naročilo pošlje naslednjemu najbližjemu taksistu. Ko je naročilo sprejeto, informacijo sporoči stranki, ki prejme potrditveno sporočilo. Če se v roku 5 minut naročilo ne potrdi, le-to zapade in se informacija o tem sporoči uporabniku.

4.3.2 Razvoj zaledne aplikacije

Zaledno aplikacijo sem razvijal v programskem jeziku Java. Sestavljena je iz dveh modulov. Prvi skrbi za zaledne operacije, ki so operacije z bazo, obdelovanje in vračanje podatkov. Drugi pa skrbi za prejemanje in vračanje zahtev aplikacijam uporabnikov in taksistov.

Zaledni modul
Zaledni modul skrbi za procesiranje zahtev. Zahteve so namreč posredovane v ta modul, ki jih najprej zapiše na bazo. Za tem pripravi odgovor, ki ga bo drugi modul posredoval odjemalcu. Nna primer: modul iz baze zajame določene podatke, jih skupaj zapiše v XML strukturo in le-to vrne modulu, ki je podal to zahtevo. Na njem se izvajajo tudi periodična opravila. Primer tega je dnevno poročilo, ki se zažene enkrat dnevno in na emaile taksistov dostavi poročilo o prejetih naročilih.

Osprednji modul
Je modul, ki skrbi za komunikacijo z odjemalci. Odjemalci so v tem primeru Android aplikacije, ki pošiljajo razne zahteve. Modul te zahteve sprejme in jih posreduje naprej zalednemu modulu. Od slednjega dobi nek odgovor, ki ga posreduje odjemalcu. Komunikacija z odjemalcem poteka preko spletnih storitev (ang. web services), ki so razvite s pomočjo JAX-WS API-ja, ki je javanski API za spletne storitve in je tudi del Java EE platforme. Spletni servis ustvarimo tako, da kreiramo ustrezen razred, ki mu z ustreznimi anotacijami definiramo spletne metode. S tem definiramo predvsem, kaj bo ta spletni servis prejel kot parametre in kaj bo vrnil odjemalcu. Poleg tega potrebujemo še datoteko sun-

jaxws.xml, v kateri se določijo naslovi, kjer bodo poslušale te spletne storitve.
Maven ob prevajanju projekta ustvari tudi WSDL (ang. Web Services Description Language) datoteke. To so XML datoteke, ki definirajo spletne storitve. V njih je zapisano vse, kar potrebuje odjemalec poznati o storitvi, da lahko z njo komunicira. To je predvsem koristno, ko nas ne zanima, kakšna platforma »teče« na odjemalčevi strani, ker so te spletne storitve neodvisne od platforme.

```
<!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2-12/14/2009 02:16 PM(ramkris)-.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
     xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:tns="http://ws.positioning.matejsimcic.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://ws.positioning.matejsimcic.com/" name="DataRetrieveWS">
<types>
<xsd:schema>
<xsd:import namespace="http://ws.positioning.matejsimcic.com/" schemaLocation="http://127.0.0.1:8080/services/DataRetrieveWS?xsd=1"/>
</xsd:schema>
</types>
<message name="availableTaxisTest">
<part name="parameters" element="tns:availableTaxisTest"/>
</message>
<message name="availableTaxisTestResponse">
<part name="parameters" element="tns:availableTaxisTestResponse"/>
</message>
<message name="availableTaxis">
<part name="parameters" element="tns:availableTaxis"/>
</message>
<message name="availableTaxisResponse">
<part name="parameters" element="tns:availableTaxisResponse"/>
</message>
<portType name="DataRetrieveWS">
```

Slika 18: Del wsdl definicije spletne storitve

5 SKLEPNE UGOTOVITVE

Razvoj same rešitve je bil zelo zanimiv. Predvsem zato, ker razvoja aplikacij v okolju Android pred tem nisem poznal. Ker pa imam predznanje programiranja v programskem jeziku Java, je bil sam proces učenja razvoja Android aplikacij zelo hiter. Poleg tega obstaja več skupin, kjer uporabniki delijo znanje in svoje izkušnje pri razvoju tovrstnih aplikacij. Zame je bila novost tudi komunikacija Android aplikacije z razvojno ploščico Arduino s pomočjo programskega paketa Amarino. Slednje omogoča veliko možnosti za razvoj raznih rešitev, kjer bi se telefon uporabljal za nadzor neke naprave na daljavo. Primer tega bi lahko bila tudi pametna hiša, saj ima Bluetooth modul, ki sem ga sam uporabil, zelo dober domet. Testiral sem ga tudi čez steno in je uspešno sprejemal in pošiljal signale. Tovrstno rešitev bi bilo sicer treba zelo dobro zaščititi, saj bi obstajale velike možnosti vdorov v samo Bluetooth omreže.

Izdelana rešitev, ki jo v diplomski nalogi tudi predstavljam, je dejansko uporabna v današnjih taksi družbah. Seveda obstaja veliko možnih izboljšav in dodatnih možnosti. Teh se zdaj, ko povzemam celotno rešitev, še posebej zavedam. Vsekakor bi bilo treba zelo dobro poznati samo poslovno okolje in vse težave, s katerimi se srečujejo taksi družbe, če bi želeli izdelati res takšno rešitev, ki bi jim v praksi služila in jim olajševala delo in odpravljala določene težave. Med samim razvojem sem sicer imel priliko sodelovati tudi s taksisti in sem spoznal, da nekateri tovrstne rešitve sprejemajo z navdušenjem. Spet pa drugi temu niso preveč naklonjeni.

Ta rešitev (v nekoliko izpopolnjeni obliki) se uporablja na projektu »Kjesitaksi«. Projekt je trenutno v fazi testiranja in se stalno posodablja ter s tem izpopolnjuje. Več o projektu je mogoče prebrati na spletni strani http://www.kjesitaksi.com.

Vsekakor menim, da so tovrstne ideje in rešitve koristne. Izdelati je mogoče zelo inovativne in uporabne stvari, ki za sam razvoj niso cenovno zahtevne.

Znanje, ki sem ga pridobil pri izdelavi diplomske naloge, bom uporabil tudi na drugih bodočih projektih. Razvoj mobilnih aplikacij za operacijski sistem Android je trenutno v zelo velikem porastu in je tudi tovrstno znanje zelo iskano, zato ocenjujem, da je bila izdelava te diplomske naloge zelo koristna za mojo osebno rast.

DODATEK A

NAMESTITEV IN NASTAVITVE ZA RAZVOJ ANDROID APLIKACIJ S POMOČJO ORODJA ECLIPSE

Navodila za vzpostavitev razvojnega okolja za Android aplikacije s pomočjo grafičnega orodja Eclipse so namenjena računalnikom, ki imajo nameščen operacijski sistem Windows. Pri drugih operacijskih sistemih je vzpostavitveni postopek zelo podoben. Paziti moramo, da pri prenašanju namestitvenih datotek izberemo takšno, ki ustreza našemu operacijskemu sistemu.

1. **Prenos Eclipse IDE**
   Iz spletne strani http://www.eclipse.org/downloads/ prenesemo razvojno okolje Eclipse, in sicer iz seznama izberemo »Eclipse IDE for Java Developers«. Izbrana datoteka je tipa ZIP.

   Ko je prenos zaključen, datoteko »razpakiramo« na poljubno mesto.

2. **Prenos Java Platforme**
   Iz spletne strani http://www.oracle.com/technetwork/java/javase/downloads prenesemo »Java Development Kit«.

   Po zaključenem prenosu prenešeno datoteko zaženemo in sledimo postopku namestitve.

3. **Prenos Android SDK**
   Iz spletne strani http://developer.android.com/sdk/index.html prenesemo Android SDK. Na izbiro imamo namestitveno datoteko in ZIP datoteko. Priporočena je izbira inštalacijske datoteke.

   Po zaključenem prenosu SDK namestimo oz. »razpakiramo« na želeno mesto, lokacijo namestitve pa si moramo zapomniti, saj bomo morali v Eclipsu definirati lokacijo, kjer se Android SDK nahaja.

4. **Konfiguracija Android SDK**
   Poženemo SDK Manager.exe, ki se nahaja v Start meniju oz. v direktoriju, kamor smo namestili Android SDK. Odpre se upravljalec paketov za Android SDK. V njem izberemo želene API-je, ki jih bomo uporabili. Le-ti so med sabo ločeni glede na

številko verzije operacijskega sistema Android, kateremu so namenjeni. Ti paketi vsebujejo tudi razne primere kode, dokumentacijo in izvorno kodo.

5. **Konfiguracija orodja Eclipse IDE**
   Po zagonu programa Eclipse v menuju Help izberemo »Install New Software«. Odpre se okno za izbiro spletnega naslova, iz katerega želimo namestiti nove komponente. Pritisnemo gumb add in v polje Location vnesemo spletni naslov https://dl-ssl.google.com/android/eclipse/ , polje Name je poljubno. Pritisnemo OK in program bo začel z zbiranjem informacij o razpoložljivih komponentah, ki jih lahko iz vnešenega naslova namestimo. Dobimo samo eno komponento, ki se imenuje »Developer Tools«. Komponento izberemo in potrdimo njeno namestitev.
   V menuju Window izberemo Preferences in odpre se okno z nastavitvami orodja Eclipse. V levem seznamu izberemo Android. Nato v polje SDK Location vnesemo pot do direktorija, kjer je nameščen Android SDK in pritisnemo OK.

Razvojno orodje Eclipse je pripravljeno na uporabo za razvoj Android aplikacij s pomočjo razvojnega kompleta Android SDK.

Seznam slik

Literatura

[1] (2011) http://www.appinventorbeta.com/learn/whatis/index.html

[2] (2011) http://blog.geckosmsapp.com/how-does-google-and-android-know-where-i-am-w-35219

[3] (2011) http://developer.android.com/guide/basics/what-is-android.html

[4] (2011) http://developer.android.com/guide/topics/manifest/manifest-intro.html

[5] (2011) http://www.nasm.si.edu/gps/work.html

[6] (2011) Priročnik za Bluetooth module http://www.sparkfun.com/datasheets/Wireless/Bluetooth/rn-bluetooth-um.pdf

[7] (2011) http://www.upr.si/fileadmin/user_upload/Novice/sporocila_za_javnost/AB-clanek.pdf

[8] Zigurd Mednieks: Programming Android: Java Programming for the New Generation of Mobile Devices, O'Reilly, 2011

[9] Glen Smith: Grails in Action, Manning, 2011