SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Renato Turić

RAZVOJ ANDROID APLIKACIJE U ECLIPSE RAZVOJNOM OKRUŽENJU

ZAVRŠNI RAD

Varaždin, 2012.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Renato Turić

Redoviti student

Broj indeksa: 38263/09-R.

Smjer: Informacijski sustavi

Preddiplomski studij

RAZVOJ ANDROID APLIKACIJE U ECLIPSE RAZVOJNOM OKRUŽENJU

ZAVRŠNI RAD

Mentor:

Tihomir Orehovački, mag. inf.

Varaždin, rujan 2012.

Sadržaj

1. UVOD

Android je programsko okruženje za mobilne uređaje bazirano na otvorenom kôdu (engl. Open Source) koje se sastoji od kompleta programske opreme: operacijskog sustava, programske međuopreme (engl. Middleware) i ključnih aplikacija za pokretne uređaje. Također, sadrži mnoštvo sučelja za izradu aplikacija (API, eng. application programming interface) koja razvojnim inženjerima omogućuju samostalan razvoj i rad na aplikacijama.

Operacijski sustav Android je još uvijek novost u svijetu pokretnih uređaja. Konkurencija je drugim operacijskim sustavima za pametne pokretne uređaje kao što su Appleov iPhone OS te razne izvedenice Windows Mobilea i Symbiana. Android karakterizira otvorena arhitektura sa specifičnim odnosom između aplikacija i prikaza na ekranu te mogućnost da se promijeni i svaka uobičajena aplikacija koja dolazi s operacijskim sustavom, što govori dovoljno o fleksibilnosti koju pruža.

Predstavljanje Androida i njegove razvojne okoline za pisanje aplikacija (SDK, engl Software development kit) započelo je u studenom 2007. godine od strane Googlea i grupacije Open Handset Alliance (OHA). Open Handset Alliance je konzorcij s ciljem razvoja otvorenih standarda za pokretne uređaje, promocija inovacija i prilagodba uređaja korisniku s poboljšanom izvedbom i pristupačnom cijenom. Broji nekoliko desetaka članica među kojima se nalazi i Google, nešto više o povijesti Android operacijskog sustava će biti riječi kasnije.

U ovom radu će se također obraditi nekoliko primjera aplikacija koje zajedno daju jednu osnovu za daljnji razvoj aplikacija te temeljna načela kako Android operacijski sustav funkcionira te ćemo se upoznati sa razvojnim okruženjem Eclipse i programskim jezikom Java.

2. ANDROID OPERACIJSKI SUSTAV

Kako bi se što bolje upoznali sa Android operacijskim sustavom najprije će biti opisana povijest Android operacijskog sustava. Najpoznatija stvar u vezi Androida je sigurno činjenica da je danas pod vlasništvom Google-a. No, to nije uvijek bilo tako. Android kao projekt je započeo 2003. godine. Dvije godine nakon osnivanja Android projekta, pod vodstvom Android Incorporation tvrtke, projekt preuzima Google. Android Incorporation je osnovan u Palo Alto u Kaliforniji. Osnivači tvrtke Android Incorporation su:

- **Andy Rubin** – suosnivač tvrtke Danger.
- **Rich Miner** – suosnivač tvrtke Wildfire Communications.
- **Nick Sears** – nekadašnji potpredsjednik u T-Mobile-u.
- **Chris White** – jedan od glavnih članova tvrtke Web TV, zadužen za razvoj dizajna i sučelja.

Od samog početka tvrtka Android Incorporation je poslovala tajno, ali su dali odmah do znanja da će se tvrtka baviti mobilnim softverom. Tijekom nekog vremena zapali su u financijsku krizu i tada ih preuzima Google, što je već spomenuto ranije. Preuzimanjem tvrtke u Google se prebacuju svi osim Nick Searsa [Yadav M., 2011].

Nakon tog događaja, 2007. godine izdana je beta verzija danas poznatog Android SDK[1]. Također iste godine osnovano je udruženje pod nazivom Open Handset Alliance[2], koje je okupljalo velike tvrtke kao što su Google, LG, Motorola, Nvidi, Samsung Electronics i mnogo drugih. OHA je predstavio svoj prvi proizvod – Android, platforma za mobilne uređaje izgrađen na Linux kernelu verzije 2.6. Godinu dana nakon toga izdan je i prvi Android uređaj, HTC Dream G1 sa operacijskim sustavom Android 1.0.

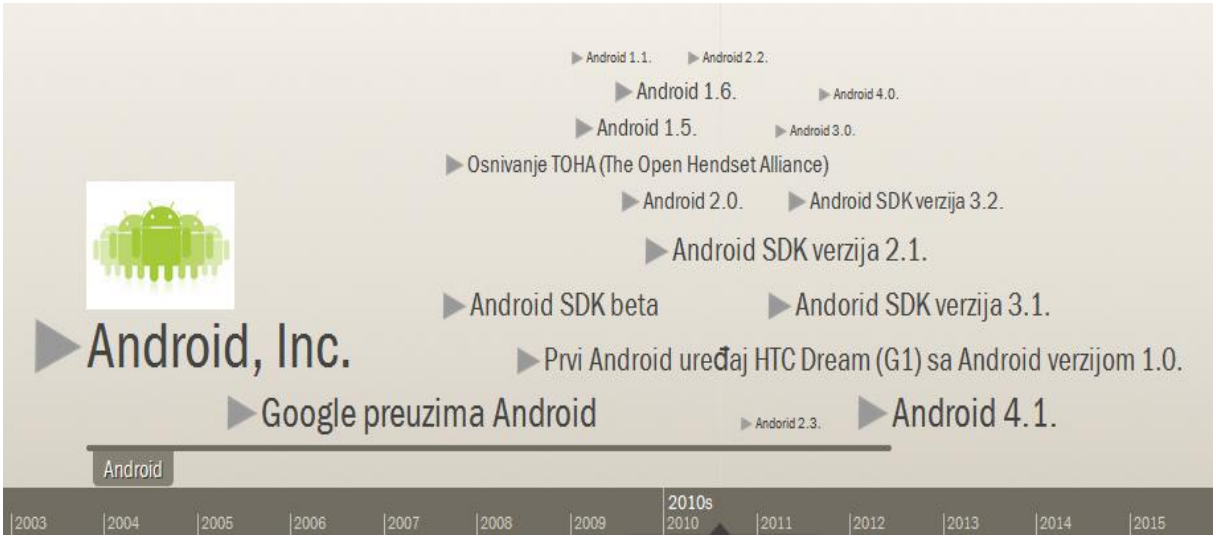
HTC Dream G1 je imao 3,2“ zaslon osjetljiv na dodir. Procesor je proizvođača Qualcomma, a njegov naziv je bio MSM7201A, radnog takta od 528MHz. Baterija izdrži do sto trideset sati u stanju pripravnosti. Memorija telefona HTC Dream G1 je 256 MB ROM-a te 192MB RAM-a. Telefon je imao microSD utor za kartice te je s telefonom dobivena i memorijska kartica od 1 GB, u ponekim zemljama i do 2 GB. Tipkovnica je bila klasična klizna QWERTY tipkovnica od 5 redaka. Što se tiče video performansi dotični mobitel je mogao učitati H.264, 3GPP, MPEG4 te

[1] SDK – (eng. Software Development Kit) – daje platformu za izradu novih aplikacija.
[2] Open Handset Alliance – Savez kojem je glavni zadatak razvoj otvorenih standarda za mobilne uređaje.

3GP formate. Također treba naglasiti da je imao Wi-Fi i Bluetooth. To su samo neka od hardverskih specifikacija prvog Android uređaja HTC Dream G1(Slika 2.1.)..

Slika 2.1. HTC Dream G1 – prvi Android uređaj (Android verzija: Android 1.0)

Na sljedećoj slici (Slika 2.2.) možemo vidjeti vremensku liniju razvoja Android, njegove verzije operacijskog sustava, verzije paketa za razvoj softvera (engl. Software Development Kit), vrijeme osnivanja te preuzimanje od strane Googlea.

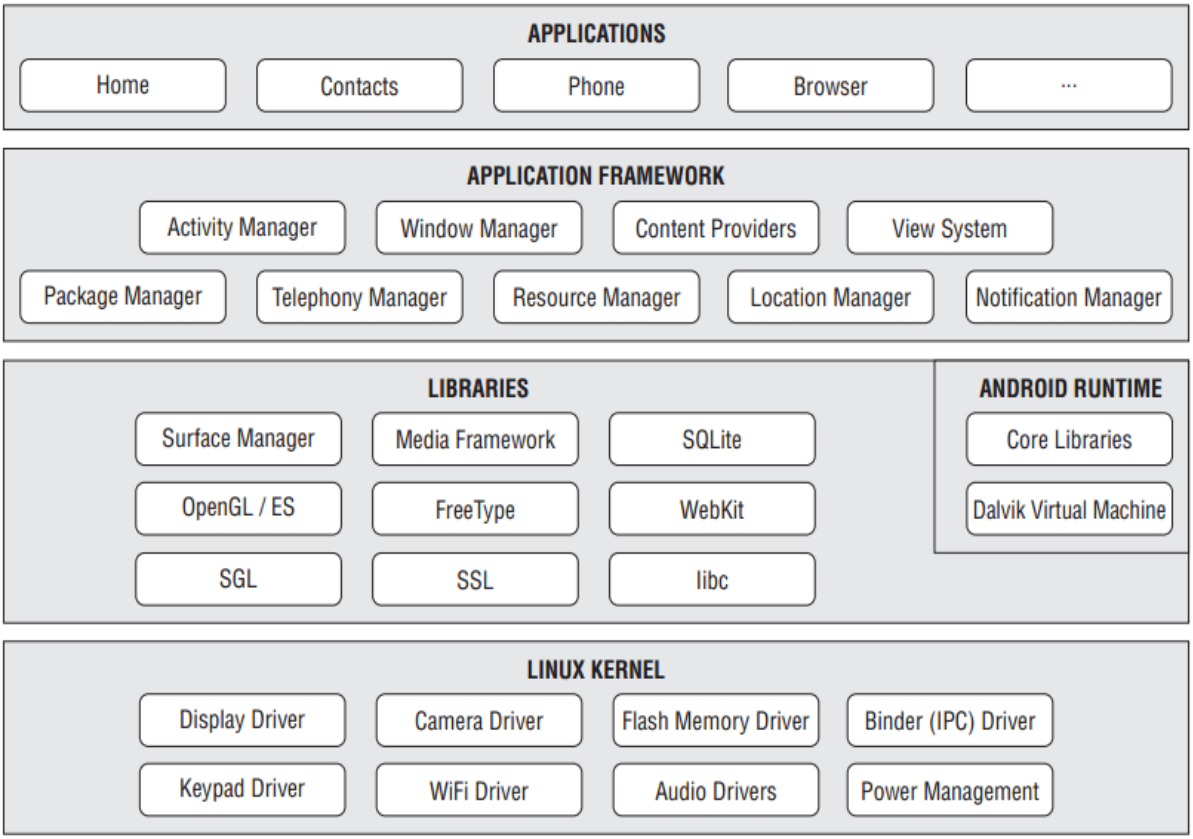


Slika 2.2. Vremenska linija Androida

U slijedećem dijelu rada će biti opisana struktura Android operacijskog sustava, a nakon toga će biti riječi o pojedinoj verziji Android operacijskog sustava.

2.1. Struktura Androida

Znati glavnu strukturu Android OS-a je jako važno kako bi znali što sve Android operacijski sustav pruža razvojnim inženjerima. Slika 2.3. najbolje prikazuje strukturu Androida.



Slika 2.3. Struktura Android OS-a [Wei-Meng L., 2011., str 3]

Android operacijski sustav se sastoji od pet dijelova, a to su [Wei-Meng L, 2012.]:

- **Aplikacije** - prvi sloj koji se naziva aplikacije povezujemo zajedno sa Android uređajima kao što su mobiteli, kontakti, pretraživači itd. Povezujemo ih naravno i sa svim aplikacijama koje korisnik preuzme sa Android Marketa[3] i instalira.
- **Aplikacijski okvir** – tu se nalaze sve potrebne mogućnosti za Android razvojne inženjere kako bi mogli razvijati Android aplikacije.
- **Biblioteke** – biblioteke sadrže glavni kôd, koji zapravo čini Android operacijski sustav – njegove glavne mogućnosti i opcije. Primjeri tih biblioteka su SQLite, WebKit, OpenGL / ES i ostale, o kojima će biti riječi nešto kasnije.

[3] Android Market – prostor na webu gdje se nalaze razne aplikacije i softveri za Android uređaje, danas se Android Market zove Google Play

- **Android vrijeme izvršavanja** (engl. Runtime) – ovaj sloj se nalazi na istom sloju kao i sloj biblioteka. Razlog tomu je to što taj sloj sadrži neke ključne biblioteke koje omogućavaju razvoj Android aplikacija koristeći Java programski jezik. Tu se također nalazi i Dalvik virtualni stroj, odnosno emulator.
- **Linux jezgre** – ovo je sama jezgra na kojoj je osnovan Android operacijski sustav. Ta jezgra sadrži drivere za razne hardverske komponente Android uređaja.

Prvi sloj, aplikacije, se sastoji od nekoliko osnovnih komponenti, a to su [Wei-Meng L, 2012.]:

- **Home -** prikazuje aplikacije, programčiće (engl. Widgets) i prečace. Također podržava promjenjivu pozadinu.
- **Phone -** podržava klasične telefonske funcije kao i kontrolu poziva, konferencijske razgovore, sporedne usluge i laku integraciju s aplikacijom Contacts.
- **Web Browser -** je pretraživač baziran na WebKitu sa svim njegovim mogućnostima, podržava HTML i XHTML.
- **Email -** osigurava pristup poslužiteljima e-maila koji se obično mogu naći na Internetu i podržava POP3, IMAP4 i SMTP.
- **Media Player -** omogućava upravljanje, uvoz i reprodukciju sadržaja kodiranih na razne načine.
- Treba još spomenuti i ostale komponente koje se nalaze u ovom sloju a to su **Alarm Clock**, **Calculator**, **Calendar**, **Camera**, **Contacts**, **IM**, **MMS**, **Settings**, **Voice Dialer** i ostale.

Sljedeći sloj predstavlja sloj biblioteka u Android operacijskom sustavu. U nastavku će se pojasniti pojedina biblioteka i ukratko reći za što koja biblioteka služi [Dujmović, 2010]:

- **Surface manager** – prikaz prozora za pojedine aplikacije.
- **OpenGL / ES** – biblioteka koja služi za grafički 2D i 3D prikaz.
- **SGL (Scene Graph Library)** – biblioteka na vrhu OpenGL biblioteke koja služi za 2D i 3D prikaz.
- **Media Framework** – odgovoran za reproduciranje i rad sa multimedijalnim datotekama.
- **Free Type** – fontovi.
- **SSL (Secure Socket Layer)** – omogućuje sigurnosnu komunikaciju preko Interneta.
- **SQLite** – implementira bazu podataka (engl. Database Engine).

- **WebKit** – jezgra web preglednika koji podržava Javascript i ostale standarde na mobilnom uređaju.
- **Libc (System C library)** – implementacija standardne sistemske biblioteke programskog jezika C (libc) izvedene iz operativnog sustava BSD[4].

Sloj aplikacijskog okvira (engl. Application Framwork) napisan je u Java programskom jeziku i sadrži programske komponente koje koriste sve aplikacije uređaja. Kako bi sama struktura bila što jasnija u nastavku su opisane i komponente ovoga sloja. Neki od važnijih elemenata ovoga sloja su [Dujmović, 2010]:

- **Activitiy Manager** – upravljanje životnim ciklusom aplikacije.
- **Package Manager** – drži informacije o aplikacijama koje su instalirane na sustav.
- **Window manager** – upravljanje aplikacijskim prozorima.
- **Telephony Manager** – API koji se koristi pri izradi aplikacija za upravljanje pozivima.
- **Content Provider** – odgovoran za zajedničko korištenje podataka od strane više aplikacija.
- **Resource Manager** – služi za pohranu dijelova aplikacije koji nisu kôd (npr. slike).
- **View System** – sadrži bazu gotovih grafičkih prikaza i alata.
- **Location Manager** – upravljanje lokacijski temeljenim uslugama.
- **Notification Manager** – upravljanje obavijestima i događanjima (npr. dospijeće poruke, nadolazeći sastanak).

2.2. Hardverske komponente Androida

Svaki razvojni inženjer prije nego što se počne baviti izradom aplikacija za Android uređaje trebao bi znati što on pruža sa hardverske strane. Ono što treba izdvojiti kao osnovne hardverske elemente su zaslon osjetljiv na dodir (engl. Touchscreen), GPS, Akcelerometar, SD kartica i ostale hardver komponente kao što su kompas, kamera, bluetooth.

Ekran osjetljiv na dodir – danas većina android uređaja ima ekran osjetljiv na dodir. Upravo takav ekran razvojnim inženjerima, a samim time i običnim korisnicima, pruža gomilu novih mogućnosti sa raznim gestama ruku (swipe, flip, drag, pinch to zoom). Ono što ovdje također treba naglasiti je to da zaslon može raspoznati dva ili više dodira odjednom (engl. Multitouch).

[4] BSD (Berkeley Software Distribution)– operacijski sustav za računala, a verzija je UNIX®-a. Razvijen na Kalifornijskom sveučilištu Berkeley, 1970. godine.

GPS – ova komponenta omogućava da saznamo točnu lokaciju uređaja, odnosno korisnika. Samim time možemo pratiti kretanje određenog korisnika, odnosno dotičnog uređaja. Ako je potrebno GPS se može koristiti na način da korisniku daje smjernice kako doći do određenog mjesta.

Akcelerometar – uređaj koji mjeri akceleraciju. To znači da se uz pomoć njega može saznati da li je određeni android uređaj u pokretu ili ga korisnik u tom trenutku trese. Također se može saznati u kojem smjeru je uređaj okrenut. To pruža razvojnim inženjerima sasvim novi pogled na izradu aplikacija.

SD kartica – prenosivi medij za pohranu podataka koji se nalazi u samom Android uređaju. Android verzija 2.2 pruža mogućnost da se aplikacije instaliraju na samu SD karticu.

2.3. Verzije Androida

U ovom poglavlju ćemo se osvrnuti na Android verzije, kako bi vidjeli kako se dotični operacijski sustav razvijao i što on danas sve nudi sa svojom najnovijom verzijom. Početak razvoja Android operacijskog sustava je započeo 22. listopada 2008. godine. Ono što je obilježilo prvu verziju Androida su notifikacijski prozor, programčići (engl. Widgets) na početnom zaslonu i cijeli sustav zvani The Android Market.

Notifikacijski prozor je bio jedinstven i bogat potrebnim informacijama za korisnika, da je Apple-u za iOS[5] trebalo sveukupno tri godine da napravi učinkovit dizajn za notifikacije koji je bio dobar i sveobuhvatan kao i notifikacijski prozor na Android uređajima. Što se tiče tehničke strane, notifikacijski prozor je ostao isti kroz cijeli razvoj Android operacijskog sustava [Ziegler, 2011].

Google je od samog početka imao velike planove za programčiće. Jedini veći problem u samom početku je bio taj što razvojni inženjeri nisu mogli napravit svoje vlastite programčiće koji bi korisnici postavili na svoj početni zaslon.

Android market, danas Google Play, je jedan od najvažnijih dijelova Androida. Jedna kvalitetna i centralizirana trgovina sa aplikacijama je nužna za svaki operacijski sustav mobilnih uređaja.

[5] iOS – operacijski sustav za mobilne uređaje, izrađen 2007. godine od strane Apple inc.

Ono što treba naglasiti je to da je Android grafičko sučelje razvijao Google uz pomoć Švedske firme TAT[6] [Yadav M., 2011]. Jedno od najznačajnijih elemenata koji je razvio TAT je sigurno naširoko poznat analogni sat na početnom zaslonu koji se nalazi na svim Android uređajima. Kasnije TAT preuzima tvrtka RIM[8] i prekidaju suradnju sa Google-om te počinju raditi za BlackBerry pametne telefone.

Ono što je još karakteristično za prvu verziju Androida je web pretraživač koji je omogućavao pretraživanje HTML i XHTML stranica u njihovim punim potencijalima. Također, tu je i višezadačnost (engl. Multitasking), Wi-Fi i slanje izravnih poruka u realnom vremenu (engl. Instant Messagaing).

2.3.1. Android 1.1

Android verzija 1.1 nije donijela ništa revolucionarno. Konkretno, Android verzija 1.1 je donijela veći broj popravaka i za određene greške (engl. Bugs) i dodala neke manje opcije kao što su ažuriranje mapa. Ta Android verzija je bila dostupna samo za T-Mobile G1 (Slika 2.4.).

Slika 2.4. T-Mobile G1

2.3.2. Android 1.5 (Cupcake)

Android 1.5 poznatiji pod kodnim imenom Cupcake dolazi na tržište sa puno očekivanih noviteta koji su bili potrebni kako bi Android i dalje bio konkurentan. Google prvi puta koristi poseban

[6] TAT - The Astonishing Tribe, Švedska firma za interaktivni dizajn. Od 2010. godine je pod vlasništvom Research In Motion Limited (RIM)

način imenovanja za verzije Android operacijskog sustava. Od tada pa sve do danas Google je za svaku izdanu verziju koristio ime određenog slatkiša.

Android Cupcake verzija je donijela mnogo sitnih promjena što se tiče grafičkog sučelja. Spomenute promjene su uglavnom bile tipa usavršavanje postojećih elemenata. Godine 2009. je izdan i prvi mobitel, sa Android 2.0 operacijskim sustavom, koji je bio u potpunosti osjetljiv na dodir, odnosno nije imao fizičku tipkovnicu - HTC Magic (Slika 2.5.).

Slika 2.5. HTC Magic – prvi mobitel na tržištu koji nije imao fizičku tipkovnicu

Jedan od noviteta su i proširene opcije što se tiče programčića. U verziji 1.0 i 1.1 programčići nisu dosegli svoj puni potencijal, dok je verzija 1.5 to promijenila. U verziji 1.5 gotovo svaka aplikacija je imala jedan ili više programčića na raspolaganju. To je od velikog značenja za Android jer su tako i dalje nastavili imati jedan od najfleksibilnijih početnih zaslona u svijetu mobitela.

Ono što treba naglasiti je to da Android u ovoj verziji usavršava opciju kopirati-zalijepiti (engl. Copy-Paste), koja u ranijim verzijama nije bila zadovoljavajuća. Nedostatak u prijašnjim verzijama što se tiče spomenute opcije je to da je bio poprilično limitiran. Jedan od primjera je da običan korisnik nije mogao kopirati običan tekst (engl. Plain text) iz prozora web pretraživača ili Gmail-a. Android 1.5 donosi popravljen dio sa pretraživačem dok dio za Gmail neće biti dostupan u još nekoliko daljnjih verzija. U ovoj verziji je također dodana i dugo očekivana opcija snimanja video sadržaja, te su dodane i neke skupne operacije za Gmail sustav (npr.

višestruko brisanje poruka). Ugrađene su i opcije uploada sadržaja na Youtube i Picasa[7] te poboljšana animacija za listanje sadržaja [Ziegler, 2011].

2.3.3. Android 1.6 (Donut)

Verzija 1.6 nije donijela toliko noviteta kao verzija 1.5. U ovoj verziji se na Android uređajima pojavila podrška za CDMA[8]. Možda je najznačajniji novitet u Donut verziji neovisnost rezolucije, što znači da je Android po prvi puta u povijesti mogao biti pokrenut na različito velikim zaslonima.

Ono što još treba spomenuti za ovu verziju je unaprijeđena traka za pretraživanje (engl. Search Box) koja se nalazila na početnom zaslonu. U prijašnjim verzijama spomenuta traka za pretraživanje je služila samo za pretraživanje interneta, dok se u ovoj verziji mogao pretraživati i sadržaj na samom mobitelu kao što su aplikacije, kontakti i drugo. Android 1.6 donosi potpuno redizajnirani Android Market i redizajnirano sučelje za kameru.

2.3.4. Android 2.0/2.1 (Eclair)

Verzija 2.0 i 2.1, kodnog imena Eclair je donijela velik broj noviteta. Verzija se pojavila prvo na mobilnom uređaju Motorola Droid (Slika 2.6.), mobitel koji je po svojim osobinama jedan od najuspješnijih mobitela u povijesti.

Slika 2.6. Motorola Droid

Jedan od najznačajnijih noviteta je zasigurno Google Maps Navigation zbog kojih Android danas ima veliki utjecaj na tržištu. Ova navigacija se izdvajala od konkurencije zbog toga što je bila besplatna te je uključivala 3D prikaz karte, glasovno vođenje i informacije o prometu. Eclair

[7] Picasa – softver za organizaciju i pregled slika. 2004. godine Google preuzima taj softver od tvrtke Lifescape
[8] CDMA - Code Division Multiple Access - Tehnika višestrukog prijenosa signala

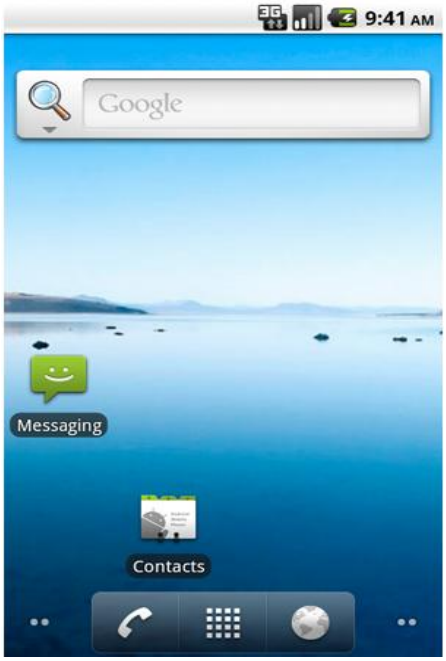
također donosi opciju zvanu Live Wallpapers, koja je zamijenila obične statične početne slike sa animiranim. Također, Ecliar je obogaćen sa opcijom speech-to-text. Android je prije toga imao samo opciju text-to-speech. Speech-to-text je na neki način bio revolucionaran na mobilnim uređajima. Korisnik je mogao glasovnom naredbom pretraživati svoj mobilni uređaj za određene kontakte, aplikacije ili slično. Ono što je još uvijek nedostajalo je zaslon osjetljiv na više dodira istovremeno za preglednike weba, ali su to sa nekim alternativama uspjeli nadoknaditi (npr. za zoom je potrebno dva puta kliknuti na željenom mjestu). Google u novoj verziji dodaje podršku za HTML5 stranice.

Osim toga donesene su neke sitne promjene, koje su samo ukrašavale novu verziju Androida. Primjer je bio novi zaslon za zaključavanje, alatna traka za kontakte te neka poboljšanja oko tipkovnice. Ono što je potrebno naglasiti je to da je većina noviteta donesena zajedno s verzijom 2.0, dok je u verziji 2.1 zakrpano par grešaka i poboljšane neke sitnice.

2.3.5. Android 2.2 (Froyo)

Verzija 2.2, kodnog imena Froyo je izdana sredinom 2010. godine. Prvi mobilni uređaj koji je ažuriran na Froyo verziju je Google Nexus One. Froyo prije svega ima primjetan redizajnirani početni zaslon (Slika 2.7. i 2.8.).

Slika 2.7. Početni zaslon Andorid verzije 2.1



Slika 2.8. Početni zaslon Andorid verzije 2.2

Osim redizajniranog početnog zaslona, tu je i potpuno redizajnirana 3D galerija slika sa puno novih animacija. Ono što je vjerojatno najznačajnije u ovoj verziji je podrška za točku za besplatni bežični pristup internetu (engl. Hotspot). Google je također dodao, danas tradicionalni, sigurnosni mehanizam za pristup početnom zaslonu. Google u ovoj verziji odlučuje da Android uvedu u enterprise okruženje, što znači usmjeriti ga organizacijama, odnosno u poslovanju [Ziegler, 2011].

2.3.6. Android 2.3 (Gingerbread)

Gingerbread je u puno pogleda malo unaprjeđenje za Android. Donosi mnoštvo sitnih promjena za grafičko sučelje početnog ekrana i pojedinih programčića. Većina promjena je nastala zbog toga da se smanji potrošnja baterije i da se u potpunosti iskoriste novi AMOLED[9] zasloni koji su bili ugrađeni na Android uređaje.

Nova verzija donosi alate za upravljanje baterijom i aplikacijama, što je pridonijelo manjoj potrošnji baterije i organiziranju aplikacija. Također se prvi put pojavila i prednja kamera na Android uređajima, čime su omogućeni video pozivi. Treba naglasiti da u Gingerbread verziji nije bila podržana opcija video čavrljanja, već je ta kamera predstavljala samo hardversku osnovu za tu mogućnost. Još jedna novost u Android 2.3 verziji je NFC tehnologija. NFC (Near Field Communication) ustvari govori o bežičnoj tehnologiji koja radi na malim udaljenostima. Rad se zasniva na principu magnetske indukcije koja se stvara među dvije antene uređaja. Između te dvije antene se stvara inducirano polje kroz koje se mogu slati električni impulsi, odnosno podaci [Pavlović, 2012.].

Nova verzija je donijela još dosta novosti u multimediji, ali je vrijedno spomenuti i poboljšane uvjete za razvojne inženjere u razvoju mobilnih igara za Android uređaje. Za novu verziju je izrađen novi sakupljač „smeća“ (engl. Garbage Collector) kako bi programeri igara mogli kreirati bolje animacije. Također su poboljšani i optimizirani pojedini događaji (engl. Events), odnosno metode, za izradu igara. Spomenuto je pridonijelo manjoj upotrebi samog procesora Android uređaja tijekom rada aplikacije. Navedeno je također trebalo osigurati položaj na tržištu igara i samim time dostići konkurenciju, pretežito iOS.

[9] AMOLED – (active-matrix organic light-emitting diode) tehnologija zaslona koja se koristi za mobilne uređaje i televiziju.

2.3.7. Android 3.x (Honeycomb)

Ono što treba naglasiti odmah na početku je to da Honeycomb označava sve verzije koje na početku imaju broj tri. Honeycomb verzija je odstupala od uobičajenog pristupa Google-a prema Androidu. Naime, verzija Honeycomb zapravo uopće nije bila za pametne telefone, već je bila namijenjena za Tablet-PC uređaje.

Google je potpuno redizajnirao grafičko sučelje Androida u Honeycomb verziji, jasno se mogao vidjeti prijelaz sa upečatljive zelene boje Androida na plavu boju (Slika 2.9.).

Slika 2.9. Prikaz početnog zaslona Android Honeycomb verzije

Verzija je također poznata po tome što je donijela kraj fizičkim tipkama predstavljanjem takozvane akcijske trake (engl. Action Bar). Ta trake se detaljno može vidjeti na prikazanoj Slici 2.9.. Honeycomb donosi i veliko poboljšanje u elementu višezadačnosti (engl. Multitaskinga).

2.3.8. Android 4.0 (Ice Cream Sandwich)

Pretposljednja verzija Androida, kodnog imena Ice Cream Sandwich, je verzija koja je najviše promijenila Android. Google se sa ovom verzijom vraća Nexus programu uz mobilni uređaj Galaxy Nexus. Ono što treba naglasiti je to da se Android 4.0 promjene dosta baziraju na Honeycomb verziji, iako je ta verzija bila usmjerena na Tablet-PC uređaje. Jedna od takvih

promjena je takozvani Android Beam. Android Beam je tehnologija koja uz pomoć NFC-a omogućava prijenos podataka čim uređaji ostvare fizički dodir, ali sada uz povećanu sigurnost. Beam tehnologija i njene mogućnosti su bile potpuno slobodne za razvojne inženjere [Ziegler, 2011].

Android Ice Cream Sandwich također uvodi biometriju kao korak sigurnosti (Primjer: otključavanje uređaja pomoću korisnikovog lica) što je u svakom pogledu inovativno i vrlo zanimljivo u svijetu pametnih telefona. Tu također imamo i vrlo dobro napravljen sustav za pregled i manipulaciju podataka.

Nova verzija donosi još mnoštvo sitnih promjena kao što su redizajnirani početni zaslon i ugradnja jednog od najboljih sustava za provjeru pravopisa. Također je zamijenjen font koji se koristi od početne verzije Androida 1.0, te je nazvan Roboto. Razlog tomu je jasniji i čitljiviji tekst na većim zaslonima [Ziegler, 2011].

2.3.9. Android 4.1 (Jelly Bean)

Android Jelly Bean je prvi put najavljen na Google I/O konferenciji 2012. Već tada se najavljivao kao Android koji će donijeti više noviteta od Ice Cream Sandwich Androida. Razlog tomu je taj što je trebao donijeti potpuno novi početak za Android usmjeren na Tablet-PC uređaje i usavršavanje svih elemenata Andorid 4.0 verzije [Ziegler, 2011].

Android Jelly Bean donosi puno kvalitetniju sliku samog zaslona te usavršen odaziv zaslona na više dodira istovremeno. Diktiranje zadataka mobitelu glasovnim naredbama je također usavršeno i po prvi puta nije bio potreban pristup Internetu kako bi korisnik iskoristio tu opciju.

Iako je Android do sada imao najbolji sustav za obavijesti na mobilnim uređajima u ovoj verziji su taj sustav obavijesti samo još više unaprijedili. Isto tako su razvojnim inženjerima omogućili dinamičniji prikaz obavijesti od svojih aplikacija. U novoj verziji se itekako može primijetiti poboljšani sustav programčića. Položaji programčića na početnom zaslonu se sada automatski pozicioniraju i automatski prilagođavaju vlastitu veličinu kako bi stali na željeno mjesto korisnika.

Imajući u obzir kako izgleda današnji početni zaslon klasičnog Android uređaja možemo vidjeti koliko je Google zapravo eksperimentirao oko dizajna, ali ne samo dizajna već i ostalih dijelova Android operacijskog sustava. Zbog svega toga Android je danas vrlo vjerojatno jedan od najboljih operacijskih sustava za mobilne uređaje na tržištu. Treba naglasiti da to ne vrijedi

samo za obične korisnike, već i za same razvojne inženjere aplikacija za Android operacijski sustav. U nastavku će biti riječi o načinu razvijanja aplikacija za Android. Kako bi to sve prošli kroz primjere u Eclipse alatu, morat ćemo proći kroz neke osnovne pojmove i pravila vezana za razvoj programa za Android.

2.4. Komponente Android aplikacije

Aplikacijske komponente su temeljni dijelovi Android aplikacije. Svaka komponenta je samostalna cjelina, igra specifičnu ulogu u radu aplikacije i ima svoj način interakcije sistema sa aplikacijom. Postoje četiri različita tipa aplikacijskih komponenata, gdje svaka ima svoj ciklus trajanja koji određuje proces pokretanja i prekida aplikacije [Marjanica, 2011]:

- **Aktivnosti** (engl. Activity) - Komponenta aktivnost predstavlja glavnu ulaznu točku korisnika u program. Aktivnost je najčešće korištena komponenta. Zadatak aktivnosti je prikaz korisničkog sučelja programa i omogućavanje interakcije korisnika sa programom (npr. slanje e-mail-a, pregled karte i dr.). Aplikacija se obično sastoji od više aktivnosti koje su međusobno povezane. Obično je jedna aktivnost označena kao početna aktivnost (engl. Main Activity) i ona se prezentira kod pokretanja aplikacije. Nakon aktiviranja početne aktivnosti pokreću se ostale funkcije programa po principu „last in, first out“. Ovim principom se štedi na radnoj memoriji mobilnog uređaja. Svaka aktivnost implementirana je kao klasa koja proširuje osnovnu klasu Activity. Activity klasa sadrži *onCreate(), onPause(), onRestart()* i druge metode koje određuju ciklus trajanja aktivnosti. Metoda *onCreate()* je ekvivalent Javinoj metodi *Main()* i ključna je za izvršavanje klase. Unutar *onCreate()* metode se uređuje korisničko sučelje (*setContentView()*). Svaka aktivnost ima svoje korisničko sučelje koje se sastoji od objekata deriviranih iz View i Viewgroup klasa.

- **Servisi** (engl. Services) - Servisi predstavljaju aplikacijsku komponentu koja izvršava zadatke u pozadini programa tijekom duljeg vremenskog perioda. Servisi nemaju grafičko sučelje. Ostale aplikacijske komponente mogu sa servisima uzajamno djelovati i izvoditi IPC međuprocesnu komunikaciju. IPC komunikacija je potrebna u slučajevima kad postoji ovisnost procesa, kad neki proces želi predati neku informaciju drugim procesima ili kad želimo provjeriti međusobno ometanje procesa. Na primjer, servisi mogu izvršavati mrežne transakcije ili slušanje glazbe iz pozadine dok korisnik radi sa drugom aplikacijom. Servisi mogu zauzeti dva oblika [Marjanica, 2011]:

- Started - Servis počinje kad aplikacijska komponenta pozove servis metodom StartService(). Tako pokrenut servis se izvršava i prekida svoj rad u pozadini neovisno o korisniku.
- Bound - Servis se veže za određenu komponentu metodom *bindService()*. Takav servis ima ciklus trajanja jednak komponenti za koju je vezan. Kako bi se kreirao servis potrebno je implementirati klasu Service, odnosno kreirati podklasu klase Service i deklarirati komponentu u manifest dokumentu.

- **Dobavljači sadržaja** (engl. Content Providers) - Dobavljači sadržaja predstavljaju komponentu aplikacije kojoj je zadaća dobavljanje i spremanje sadržaja. Ova komponenta je ujedno i jedini način za izmjenjivanje podataka među aplikacijama. Kreiranje dobavljača sadržaja se radi tako da se odredi lokacija za spremanje podataka. Dobavljač sadržaja daje sadržaj u formi tablice sa identifikatorom i ostalim atributima. Sadržaju se pristupa upitom pomoću takozvanih Content Resolvera. Upit sadrži adresu (URI) sadržaja, imena polja iz tablice i tip podataka – tekstualni tip (String), cjelobrojni tip (Integer) ili decimalni tip (Float). Svi dobavljači sadržaja spremaju podatke u SQLite bazu podataka. Naravno, moguće je koristiti PostgreSQL bazu podataka ili neki drugi način za spremanja sadržaja. Svaki dobavljač sadržaja implementiran je kao klasa koja proširuje osnovnu klasu *ContentProvider*.

- **Primatelji prijenosa** (engl. Broadcast Receiver) - Primatelji prijenosa imaju zadaću reagiranja na emitiranje obavijesti koje može dolaziti od strane sistema uređaja (npr. baterija je prazna, zaslon je uključen) ili od aplikacija (komunikacija s drugim aplikacijama). Ova komponenta nema korisničko sučelje, ali može pokrenuti aplikaciju kao rezultat primljene informacije. Također može koristiti klasu *NotificationManager* za upozorenje korisnika pomoću zvukova, vibracija ili svjetla.

3. ANDROID RAZVOJNO OKRUŽENJE

U ovom poglavlju će biti riječi o razvojnom okruženju, konkretnije o Eclipse te o SQL bazi podataka koja se danas koristi za Android aplikacije.

3.1. SQLite

SQLite je programska podrška koja omogućava samostalan, jednostavan za konfiguriranje, transakcijski i poslužiteljski neovisan sustav za upravljanje SQL bazama podataka. Ovo programsko rješenje je najraširenije programsko rješenje za SQL baze podataka, a njegov programski kod je zasnovan na principu javne domene (engl. Public Domain). To znači da se programski kod u cijelosti, kao i njegovi dijelovi, slobodno mogu kopirati, modificirati, objavljivati, koristiti, prodavati i distribuirati u originalnom SQLite formatu.

Programski kod je napisan u programskom jeziku C tako da zahtjeva minimalan broj pomoćnih programskih datoteka. Osim standardnih C biblioteka, potreban je minimalan set sistemskih biblioteka za funkcije vremena. Samo generiranje izvršnog koda je jednostavno i moguće ga je obaviti na velikom broju različitih platformi [Bucić, 2012].

Ono što karakterizira SQLite je to da nema konfiguracijsku datoteku, nema potrebe za instalacijom ili podešavanjem. SQLite također omogućava višestruki istovremeni pristup podacima. Što se tiče transakcija SQLite svaki upit ili promjenu nad podacima obavlja na siguran način, poštujući ACID principe. ACID (Atomicity, Consistency, Isolation and Durability) predstavlja model koji govori o tome da svaki sustav za upravljanje bazama podataka mora ispuniti određene uvjete kako bi sustav korektno radio [Chapple M., ***].

Neke od osnovnih karakteristika SQLite-a su te da podržava većinu standardnih SQL-92 naredbi te je zbog pojedinih specifičnosti uvedeno i nešto vlastitih nadogradnji nad standardnim setom SQL naredbi. Bitno je spomenuti da pojedine funkcionalnosti nisu implementirane, ali su pružene mogućnosti implementacije u novijim verzijama.

Glavni tipovi podataka u SQLite-u su [Penezić, 2012]:

- NULL - NULL vrijednost.
- INTEGER - standardna vrijednost cijelih brojeva s duljinom zapisa od 1, 2, 3, 4, 6, ili 8 byteova, ovisno o danoj vrijednosti.
- REAL - vrijednost realnog broja s 8 byteova u standardnom IEEE zapisu.

- TEXT - tekstualna vrijednost zapisana u jednom od podržanih načina zapisa (UTF-8, UTF-16BE ili UTF-16LE).
- BLOB - vrijednost pohranjena točno kako je dana.

Iako postoji mogućnost zapisa vremenskih podataka, isti se interno pretvaraju u jedan od gore navedenih tipova zapisa.

Prilikom rada SQLite programsko rješenje koristi tri vrste zapisa [Penezić, 2012]:

- Zapis baze podataka - svi podaci u bazi podataka se nalaze u jednoj datoteci
- Zapis transakcijskog loga (engl. Rollback Journal) - privremena datoteka koja osigurava ACID implementaciju
- Privremeni zapisi - nastaju kao rezultat rada s podacima

Kao što smo već spomenuli SQLite je danas najraširenije programsko rješenje za SQL baze podataka, ali pogledajmo koji su nedostatci tog sustava za upravljanje bazama podataka [Bucić, 2012]:

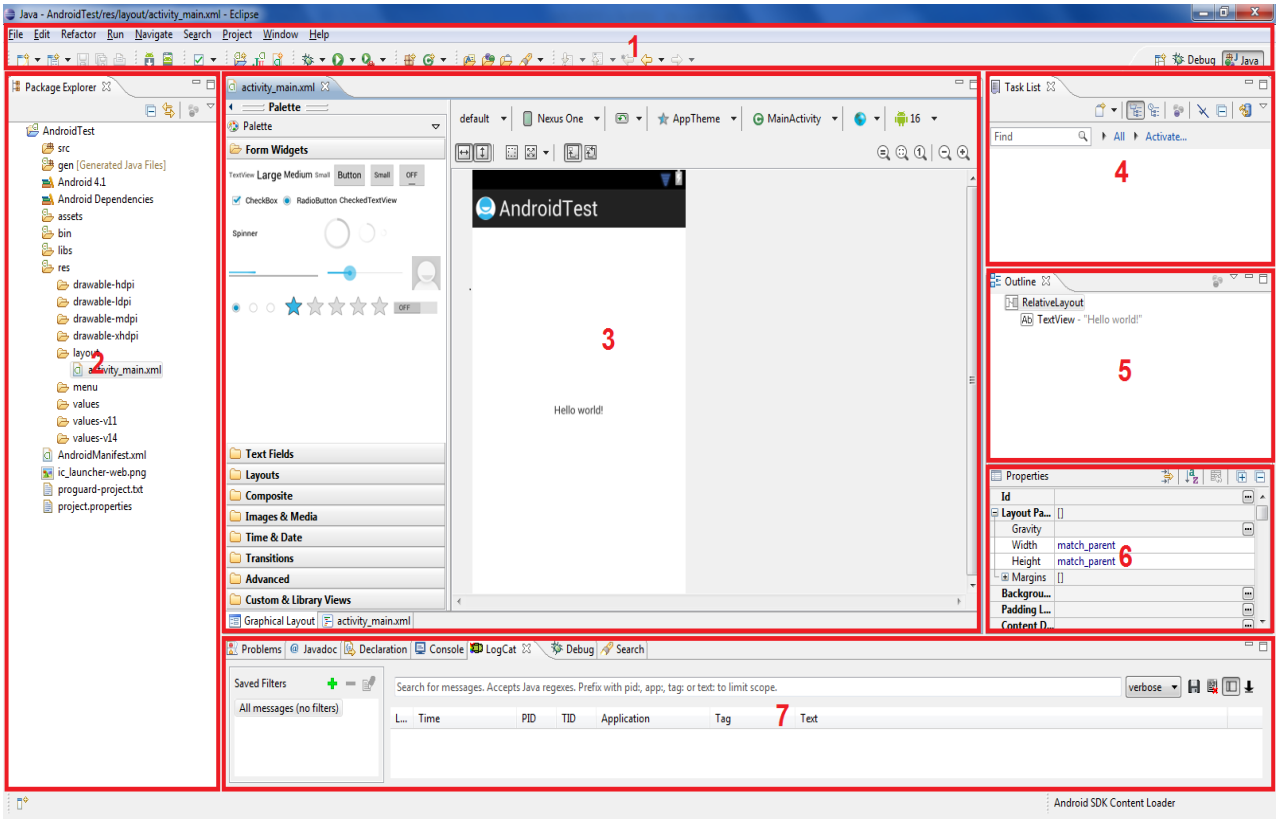
- ALTER TABLE nije u potpunosti podržan; samo RENAME TABLE i ADD COLUMN naredbe su podržane. Ostale naredbe poput DROP COLUMN, ALTER COLUMN, i ADD CONSTRAINT nisu implementirane
- Ugnježđene transakcije; SQLite dozovoljava samo pojedinačne transakcije. Ugnježđene transakcije dozvoljavaju kontrolu nad većim i kompleksnijim operacijama.
- RIGHT i FULL OUTER JOIN; LEFT OUTER JOIN je implementiran, ali RIGHT OUTER JOIN i FULL OUTER JOIN nisu. RIGHT OUT JOIN može biti implementiran jednostavnim obrtanjem redoslijeda tablica i modificiranjem „join“ uvijeta. FULL UTER JOIN može biti implementiran kao kombinacija ostalih relacijskih operacija koje SQLite podržava

3.2. Eclipse

U ovom će se dijelu opisati okruženje u kojem se danas razvijaju Android aplikacije, a to je Eclipse. Eclipse je višejezično, softversko razvojno okruženje koje se sastoji od integriranog razvojnog okruženja (IDE) i proširivog plug-in sistema. Eclipse je razvijen od strane Object Technology International kompanije kao Java softver otvorenog koda. Može se koristiti za razvoj aplikacija u Javi i drugim programskim jezicima putem različitih softverskih dodataka. Moguće je Eclipse nadograditi sa dodacima za programske jezike C, C++, COBOL, Perl, PHP, Python i

druge. Razvojno okruženje Eclipse-a je vrlo prilagodljivo, omogućava uređivanje različitih perspektiva koje se sastoje od preglednika i urednika (Slika 3.1.). Na slici je prikazano okruženje za razvoj Android aplikacija koja se sastoji, kao što možemo vidjeti, od 7 dijelova:

- glavne alatne trake(1)
- preglednika programskog paketa (engl. Package Explorer) (2)
- urednika za pisanje programskog koda (3)
- prozora za prikaz zadataka (4)
- prozora za prikaz glavnih kontrola koji se nalaze u vašoj aplikaciji(npr. TextView) (5)
- prozora sa svojstvima kontrola (6)
- prozora za prikaz rezultata debugiranja, prikaz konzole, grešaka i ostalo (7)



Slika 3.1. Izgled Eclipse razvojnog okruženja – Razvoj Android aplikacija

3.2.1. Java programski jezik

Java je objektno-orijentirani programski jezik razvijen u timu predvođenim James Goslingom u kompaniji Sun Microsystems početkom 1990. g. Ideja je bila stvoriti programski jezik koji bi bio nezavisan od operativnog sistema, baziran na C++ programskom jeziku, ali sa

pojednostavljenom sintaksom, stabilnijim radnim okruženjem i pojednostavljenom kontrolom memorije. Prema stručnjacima postoji Java pripada skupini viših programskih jezika i ima svojstvo prenosivosti. Prenosivost u programskim jezicima označava mogućnost izvršenja istog programskog koda na različitim računalima. Program koji je pisan u višem programskom jeziku se prevodi u strojni jezik, što znači da ne može biti izvršen izravno na računalu. Tu pretvorbu obavlja zasebni program predvodnik (engl. Compiler) [Marjanica, 2011]. Nakon što je program jednom preveden, može se izvršavati neograničen broj puta na istom računalu. Umjesto korištenja predvodnika moguće je koristiti interpreter koji prevodi naredbu po naredbu, prema potrebi. Kod programskog jezika Java se koristi kombinacija predvodnika i interpretera. Programi pisani u Javi se prevode u strojni jezik računala koje ne postoji. Takvo virtualno računalo se zove Java Virtual Machine (JVM). Ideja je da ako se kôd napiše i predvodi na jednoj platformi (npr. Mac OS X), taj isti bajt kôd se može izvršavati na svim ostalim platformama koje imaju JVM (npr. Microsoft Windows XP, Linux). U Java programskom jeziku su objektno-orijentirani principi obavezni. Sve je u Javi objekt, a izvorni kôd je pisan unutar klasa. Objekt u realnom svijetu može biti čovjek ili bilo što drugo. Objekt je definiran svojim stanjem i ponašanjem [Marjanica, 2011]. Na primjer, stanje objekta čovjek može biti da trči ili stoji, a ponašanje može biti brzina trčanja, smjer kretanja, itd.

U programskom jeziku stanje se opisuje sa varijablama, a ponašanje se definira sa metodama. Klasa predstavlja nacrt objekta. Na temelju klasa se proizvode objekti. Na primjer, klasa koja sadrži općeniti objekt lik može kreirati više likova sa različitim stanjem i ponašanjem. Klasa može označavati dio programskog kôda ili cijeli programski kôd. U pravilu, svaka klasa je deklarirana unutar datoteke sa istim imenom i ekstenzijom .java. Pravilo je da su imena klasa i datoteka u kojima su klase spremljene ista. Sve klase jedne aplikacije spremaju se u paket sa nazivom domene (npr. example.com.data) [Marjanica, 2011].

Paket predstavlja hijerarhijsko grupiranje razreda, a ime paketa odgovara strukturi direktorija na disku. Drugi pojam u Javi je metoda. Klase koje pokreću program moraju imati *Main()* metodu. Metode su zasebne cjeline unutar pojedine klase koje izvršavaju određene operacije [Marjanica, 2011]. Metode su u pojmu objektno-orijentiranog programiranja objekti, tj. jedinice koje imaju svoje ponašanje, drže podatke i mogu međusobno djelovati. Programiranje se sastoji od oblikovanja skupa objekata koji na neki način opisuju problem koji treba riješiti

3.2.1.1. Struktura Java datoteke

Programiranje u Javi se odvija u klasama. Ime klase je ujedno i ime programa. Svaka Java datoteka se sastoji od tri dijela [Marjanica, 2011]:

- Deklaracija paketa kojoj klasa pripada (engl. Package) pri čemu ime paketa predstavlja hijerarhijski prikaz strukture direktorija u kojem se nalazi datoteka.
- Opcionalni popis paketa koje treba uključiti (engl. Import).
- Deklaracija klase. Java programski jezik podržava ugniježđavanje, što znači da je unutar jedne klase moguće deklarirati druge klase. Da bi se neki program mogao izvršiti, on mora sadržavati metodu Main(). Deklaracija metode sadrži tri modifikatora, tj. *public*, *static* i *void*. *Public* označava javnu metodu, *void* metoda ne vraća ništa, a *static* metoda pripada samoj klasi te nije potrebno stvarati instancu kako bi se metoda mogla koristiti.

3.3. Karakteristike Android programskog okruženja

Glavne karakteristike Android programskog okruženja su najbolje istaknute u politici organizacije Open Handset Alliance. Spomenute karakteristike su [Burnette, 2009]:

- otvorenost – programeru omogućava potpunu slobodu u razvoju novih i već postojećih aplikacija, a proizvođaču uređaja slobodno korištenje i prilagodbu platforme bez plaćanja autorskih prava;
- sve aplikacije su ravnopravne – što znači da ne postoji razlika između osnovnih jezgrenih aplikacija uređaja i dodatnih aplikacija. Svim aplikacijama omogućen je ravnopravni pristup resursima pokretnog uređaja što daje mogućnost potpune prilagodbe uređaja specifičnim potrebama individualnog korisnika;
- automatsko upravljanje životnim ciklusom aplikacije – omogućava nadzor pokretanja i izvršavanja aplikacija na sistemskoj razini optimiziranim korištenjem memorije i snage uređaja. Krajnji korisnik više ne brine o gašenju određenih aplikacija prije pokretanja drugih;
- rušenje granica "klasičnih" aplikacija – mogućnost razvoja novih i inovativnih aplikacija temeljenih na međusobnoj kolaboraciji tehnologija;
- brz i jednostavan razvoj aplikacija – omogućen je bogatom bazom korisnih programskih biblioteka (engl. Libraries) i alata za izradu aplikacija;

- visokokvalitetni grafički prikaz i zvuk – podržana 2D vektorska i 3D OpenGL (engl. Open Graphics Library) grafika, te ugrađeni kodeci svih često korištenih audio i video formata;
- kompatibilnost s većinom sadašnjeg i budućeg hardvera – uključuje prenosivost Android aplikacija na ARM, x86 i ostale arhitekture, te prilagodljivost sustava ulaznim i izlaznim komponentama.

4. PRIMJERI APLIKACIJA

U ovom poglavlju će se pokazati nekoliko primjera aplikacija koje će zajedno dati jednu vrstu osnove za daljnji razvoj aplikacija. Prije nego što će se početi sa radom, potrebno je instalirati potrebno okruženje za razvoj aplikacija, ti dodaci su:

- Eclipse
- Android SDK
- Android development kit (ADT)
- Android virtual devices (AVD)

Nakon što se instaliraju potrebni dodaci može se krenuti u razvoj Android aplikacija. Prvi primjer će biti implementirati listu, što je danas jako važan koncept u Android razvojnom inženjerstvu.

4.1. Primjer aplikacije: Liste

Ovaj primjer pokazuje kako implementirati jednostavnu listu unutar bilo koje Android aplikacije. Osim same implementacije liste prikazat će se jedan događaj (engl. Event), odnosno metoda koja nam pokazuje koji je element u listi trenutno označen od strane krajnjeg korisnika – *onListItemClick()*. Na taj način možemo konkretno odrediti što će se dalje desiti u aplikaciji. Metoda *onListItemClick()* je tipa void i ima četiri argumenta. Osim te metode postoje još metode koje služe da bi upotpunili funkcionalnost liste u Android okruženju.

Prije prelaska na kôd aplikacije, potrebno je znati da se android aplikacija sastoji od više dijelova, ali ono što se najviše izdvaja tijekom razvoja android aplikacije su tri dijela:

- Glavna aktivnost u obliku xml datoteke (Datoteka se nalazi u ../res/layout/)
- Glavna aktivnost u obliku java datoteke (Datoteka se nalazi u ../src/)
- Android manifest u obliku xml datoteka (Datoteka se nalazi u glavnoj mapi aplikacije)

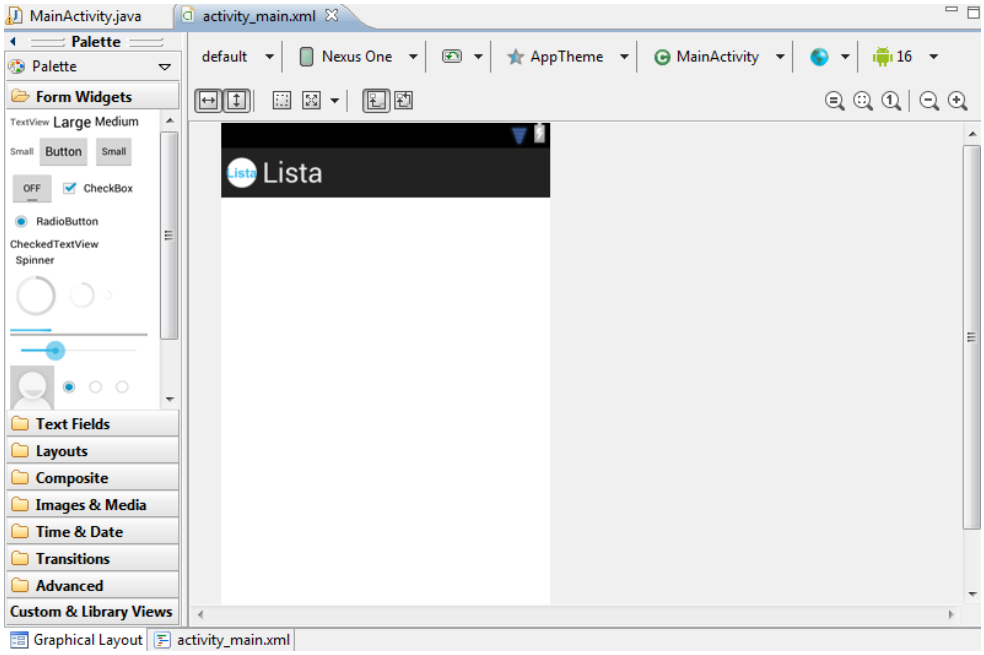
Datoteka glavna aktivnost u xml obliku nam predstavlja izgled (engl. Layout) same aplikacije, dok nam glavna aktivnost .java ekstenzije predstavlja onaj logični dio aplikacije. U datoteci Android manifest se nalaze razna dopuštenja i restrikcije android aplikacije (primjer: da li aplikacija ima pristup internetu, da li aplikacija može slati ili primati poruke). Sada ćemo

pogledat kôd primjera aplikacije liste. Prvo ćemo pogledati datoteku u kojoj je konstruiran izgled aplikacije, naziva *activity_main.xml*.

- Datoteka *activity_main.xml*:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
</RelativeLayout>
```

Kao što se može vidjeti datoteka sadrži nekoliko linija kôda koje konstruiraju cijeli izgled aplikacije (Slika 4.1.)



Slika 4.1. Izgled aplikacije Liste u Eclipseu

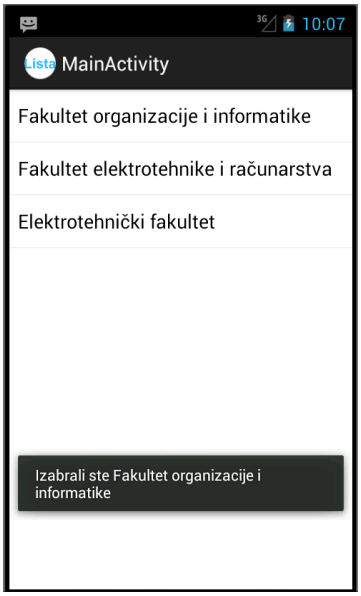
Ono što ovdje treba istaknuti su opcije *match_parent* i *wrap_content*. To su konstante kojima opisujemo visinu i dužinu određenog pogleda (engl. View). Konstanta *match_parent* označava da će dotični pogled, odnosno kontrola (primjer textview) zauzeti veličinu kao i njegov roditelj. Konstanta *wrap_content* označava da kontrola zauzima dovoljno velik prostor da obuhvati vlastiti sadržaj. Sljedeća datoteka je datoteka *MainActivity.java*.

- Datoteka *MainActivity.java:*

```
package com.example.lista;
import …

public class MainActivity extends ListActivity {

String[] fakulteti = new String[]{"Fakultet organizacije i informatike","Fakultet
elektrotehnike i računarstva","Elektrotehnički fakultet"};

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
```

```
        super.onCreate(savedInstanceState);
        setListAdapter(new
             ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,fakulteti));
    }

    @Override
    protected void onListItemClick(ListView l, View v, int position, long id)
    {
          Toast.makeText(this, "Izabrali ste " + fakulteti[position],
          Toast.LENGTH_LONG).show();
    }
}
```

Na samom početku se nalazi paket kojem aplikacija pripada te popis korištenih biblioteka za izradu aplikacije. Ono po čemu se razlikuje implementacija liste od obične android aplikacije je to što se klasa MainActivity (naziv glavne klase aplikacije) mora proširiti sa superklasom ListActivity što se može vidjeti u sljedećoj liniji kôda *public class MainActivity extends ListActivity*. U kôdu prvo kreiramo globalnu varijablu tipa string, u ovom slučaju varijabla fakulteti koja sadrži imena nekoliko fakulteta. Nakon toga dolazi *onCreate()* metoda koja se generira automatski tijekom kreiranja novog projekta, zajedno sa *super.onCreate(savedInstanceState)* linijom kôda. Spomenuta lista se sastoji od vrijednosti iz stringa fakulteti. Sljedeći dio je pozivanje metode *setListAdapter()* koja automatski puni zaslon sa *listView* kontrolom, odnosno pogledom. Unutar te metode se kreira novi *ArrayAdapter* koji će preuzeti podatke iz strigna fakulteti i popuniti s tim podacima cijelu listu. Nakon što je implementirana slijedi metoda *onListItemClick()* uz pomoću koje se odredi koji je element liste označen od strane korisnika. Na samom kraju uz pomoć Toast objekta pozivamo metodu *makeText()* kako bi prikazali koja je vrijednost označena. Time završava jednostavna aplikacija koja bi trebala pokazati osnove oko implementacije liste (Slika 4.2.).



Slika 4.2. Grafičko sučelje aplikacije Lista

4.2. Primjer aplikacije: Text-to-speech

Sljedeća aplikacija će pojasniti kako se koristi *TextToSpeech* klasa koja je dio Androidovog API[10]-a. *TextToSpeech* zajedno sa *RecognitionListener* sučeljem može biti od velike koristi u raznim aplikacijama. U ovom dijelu će biti riječ o *TextToSpeech* klasi i kako pretvoriti tekst u glasovnu naredbu. Bitno je spomenuti da se *RecognitionListener* sučelje bavi obrnutim mehanizmom, pretvaranje glasovne naredbe u tekst.

U samoj aplikaciji koja na zaslonu ima jedan *textView*, jedan *editText* koji služi za unos željenog teksta te jednu tipku (engl. Button) koja pokreće funkciju pretvorbe teksta u glasovnu naredbu. Prvi zadatak je implementirati algoritam koji će pratiti da li je došlo do nekakvog događaja od strane korisnika. Za tu radnju će poslužiti *onClick* metoda. Dakle, kada korisnik klikne na gumb automatski će se pokrenuti *onClick* metoda koja će sadržavati naredbe za preuzimanja teksta iz textView kontroole te metodu *izgovori* koju ćemo sami implementirati. Na kraju onClick metoda će izgledati ovako:

```
    public void onClick(View e)
    {
            EditText eText = (EditText)findViewById(R.id.editText1);
            String sText = eText.getText().toString();
            izgovori(sText);
}
```

Vlastita metoda *izgovori* će imati sljedeći kôd:

```
    private void izgovori(String sText)
    {
        textToSpeech.speak(sText, TextToSpeech.QUEUE_FLUSH, null);
}
```

Spomenuti dijelovi kôda čine glavni kostur aplikacije no aplikacija nije završena jer preostaje ispitati da li korisnikov uređaj na kojem je instalirana aplikacija ima instalirane potrebne podatke kako bi se uspješno kreirala instanca *TextToSpeech* klase. Za to će bit zaslužne sljedeće linije kôda, kao globalna varijabla:

```
    private int provjeraPodataka = 0;
```

Unutar *onCreate* metode se nalazi kôd u nastavku:

```
    Intent checkIntent = new Intent();
    checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
    startActivityForResult(checkIntent, provjeraPodataka);
```

[10] API - Application programming interface - predstavlja sučelje za programiranje aplikacija. Odnosno, skup određenih pravila i specifikacija koje programeri slijede tako da se mogu služiti uslugama ili resursima operacijskog sustava ili nekog drugog složenog programa.

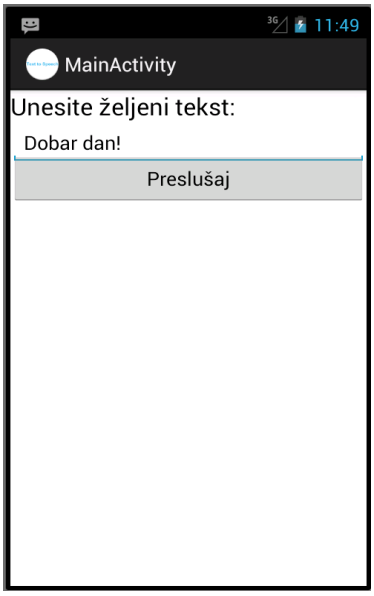
Nakon što smo pripremili sve kako bi provjerili da li su instalirani potrebni podaci, potrebno je implementirati sljedeću metodu:

```
protected void onActivityResult(int request, int result, Intent data)
{
        if(request == provjeraPodataka){
                if(result == TextToSpeech.Engine.CHECK_VOICE_DATA_PASS){
                        textToSpeech = new TextToSpeech(this, this);
                }else{
                        Intent installIntent = new Intent();
                        installIntent.setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DA
                        TA);
                        startActivity(installIntent);
                }
    }
}
```

Spomenuta metoda *onActivityResult* će provjeriti da li su potrebni podaci prisutni, ako jesu onda će se uspješno kreirati instanca *TextToSpeech* klase, u suprotnom će uputiti korisnika da instalira potrebne podatke. Kako bi se aplikacija završila potrebno je još dodati metodu koja će provjeriti da li se uspješno kreirana instanca već spomenute klase te dodijeliti jezik, odnosno operatora koji će izgovarati određeni tekst:

```
public void onInit(int initStatus)
{
  if(initStatus == TextToSpeech.SUCCESS)
  {
    if(textToSpeech.isLanguageAvailable(Locale.US)==TextToSpeech.LANG_AVAILABLE)
            textToSpeech.setLanguage(Locale.US);
  }else if(initStatus == TextToSpeech.ERROR){
    Toast.makeText(this,"Na žalost došlo je do greške...",Toast.LENGTH_LONG).show();
  }
}
```

Metoda *onInit* će također javiti grešku ako dođe do neuspješne inicijalizacije. Sa tom metodom je dovršena aplikacija Text-to-speech (Slika 4.3.).



Slika 4.3. Grafičko sučelje aplikacije Text-to-speech

4.3. Primjer aplikacije: Geste

U ovom primjeru aplikacije demonstrirati će se glavne funkcionalnosti oko uporabe gesti na zaslonu osjetljiv na dodir. U aplikaciji će biti prikazane osnovne geste kao što su pomicanje (engl. Scroll), bacanje (engl. Fling) te gesta dvostrukog dodira (engl. Double tap). Svaka gesta ima svoju implementiranu metodu unutar android API-a, a to su *onScroll()*, *onFling()* i *onDoubleTap()*. Osim spomenutih gesti postoje još i sljedeće geste:

- *onDoubleTapEvent()*
- *onDown()*
- *onLongPress()*
- *onShowPress()*
- *onSingleTapConfirmed()*
- *onSingleTapUp()*

Primjer aplikacije Geste sadrži dvije klase unutar glavne *MainActivity* klase, a to su klase *Platno* i *GestureListener*. Prvo što će se implementirati u ovoj aplikaciji je takozvano platno (engl. Canvas) na kojem će se nalaziti FOI logo te će se s njim moći upravljati uz pomoć gesta koje su implementirane u klasi *GestureListener*. Pogledajmo kako izgleda klasa *Platno*:

```
private class Platno extends View {
   private Matrix matrica;
   private Bitmap foiLogo;
   @Override
   protected void onDraw(Canvas canvas)
   {
     canvas.drawBitmap(foiLogo, matrica, null);
   }
}
```

U priloženom dijelu kôdu možemo istaknuti *drawBitmap()* metodu, ona nam slika naš logo na osnovu zadane matrice. Konstruktor spomenute klase je sljedeći:

```
public Platno(Context context) {
   super(context);
   matrica = new Matrix();
   geste = new GestureDetector(MainActivity.this, new GestureListener(this));
   foiLogo = BitmapFactory.decodeResource(getResources(), R.drawable.szfoi);
}
```

U konstruktoru se kreira i inicijalizira *GestureDetector* kojem je zadatak da prepozna, u slučaju da se dogodila neka gesta od strane korisnika, koja je to točno gesta bila. Kada prepozna o kojoj gesti je riječ, onda se ta informacija proslijedi *GestureListener* klasi. *GestureListener* klasa će pokrenuti određenu radnju ovisno o gesti. Također treba spomenuti da se u konstruktoru učitava

slika, u ovom slučaju FOI logo (*szfoi.png*), koja se nalazi u resursima projekta. Sljedeći korak je implementirati metodu koja će pri svakom dodiru gledati da li se dogodila nekakva gesta:

```
public boolean onTouchEvent(MotionEvent event) {
  return geste.onTouchEvent(event);

}
```

Ovaj kratki kôd će se baviti prepoznavanjem gesti, dok će sljedeći kôd ovisno o kojoj je gesti riječ pokrenuti određenu radnju:

```
    private class GestureListener implements GestureDetector.OnGestureListener,
      GestureDetector.OnDoubleTapListener {
           Platno pogled;
           public GestureListener(Platno view) {
                  this.pogled = view;
           }
        public boolean onDown(MotionEvent e)
        {
       return true;
        }
            public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float
        distanceY)
        {
          pogled.onMove(-distanceX, -distanceY);
             return true;
           }
           public boolean onDoubleTap(MotionEvent e)
        {
             pogled.onResetLocation();
             return true;
           }
        public boolean onFling(MotionEvent e1, MotionEvent e2, final float velocityX,
        final float velocityY)
        {
          final float distanceTimeFactor = 0.4f;
             final float totalDx = (distanceTimeFactor * velocityX/2);
             final float totalDy = (distanceTimeFactor * velocityY/2);
             pogled.onAnimateMove(totalDx, totalDy,(long) (1000 * distanceTimeFactor));
      return true;
        }
}
```

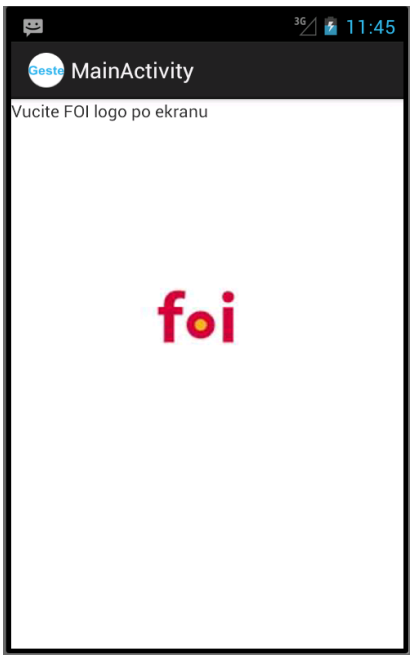
Klasa *GestureListener* je izvedena iz sučelja *OnGestureListener* i *OnDoubleTapListener*. Razlog tomu je taj što nas u aplikaciji zanimaju pokreti bilo kakve vrste za što je zaslužno *OnGestureListener* te događaji koji su pokrenuti dvostrukim dodirom zaslona za što se brine *OnDoubleTapListener* sučelje. Prva metoda unutar naše klase koja predstavlja nekakvu gestu je metoda *onScroll()*. Spomenuta metoda predstavlja gestu povlačenja, odnosno pomicanja. Kako bi se ova metoda realizirala metoda *onDown()* mora vratiti vrijednost *true*. Metoda *onScroll()* poziva metodu *onMove()* koja pripada objektu naše klase *Platno*. Metoda *onMove()* translatira našu matricu za udaljenost koju je napravio prst na zaslonu te ponovo crta našu sliku na drugom mjestu. Sljedeća metoda, *onDoubleTap()*, će resetirati položaj naše slike na svoj početni položaj.

Spomenuta radnja se realizira pozivanjem metode *onResetLocation()*. Metoda *onResetLocation()* sadrži svega dvije linije kôda:

```
public void onResetLocation()
{
matrica.reset();
invalidate();
}
```

Posljednja implementirana metoda je *onFling()*, ta metoda predstavlja gestu bacanja. Udaljenost koju će proputovati FOI logo efektom bacanja po zaslonu će se izračunati pomoću ubrzanja (engl. Velocity). Ubrzanje je određeno po prijeđenom putu mjereno u pikselima u određenom vremenu mjereno u sekundama. Kôd koji se nalazi unutar dotične metode izračunava udaljenost koju slika mora prijeći nakon što se desila gesta bacanja.

Kao što je već ranije spomenuto postoji još metoda za razne geste ali u ovom primjeru je cilj pokazati osnove oko implementacija gesti. Prikaz grafičkog sučelja aplikacije se može vidjeti na sljedećoj slici (Slika 4.4.).



Slika 4.4. Grafičko sučelje aplikacije Geste

4.4. Primjer aplikacije: SMS

Aplikacija SMS je aplikacija koja omogućava slanje kratkih poruka između dva emulatora ili dva Android uređaja koja imaju instaliranu dotičnu aplikaciju. Kako bi omogućili da naša aplikacija šalje i prima poruke najprije je potrebno dodati takozvana dopuštenja unutar Android manifest datoteke:

- Datoteka *AndroidManifest.xml*:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.sms"
  android:versionCode="1"
  android:versionName="1.0" >
  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.SEND_SMS"/>
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".MainActivity"
      android:label="@string/title_activity_main" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Ono što smo dodali unutar ove datoteke su sljedeće dvije linije kôda:

```
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

Po ključnim riječima uses-premission te SEND_SMS i RECEIVE_SMS možemo zaključiti da prva linija označava dopuštenje za slanje poruka dok druga za primanje poruka.

Grafičko sučelje se sastoj od dvije *editText* kontrole u kojima se upisuje broj primatelja poruke i sami tekst poruke. Osim spomenutih kontrola tu se nalazi i tipka pošalji koja će pokrenuti metodu slanja poruke. Metoda koja je zaslužna za slanje poruka se zove *posaljiPoruku()* i tipa je void:

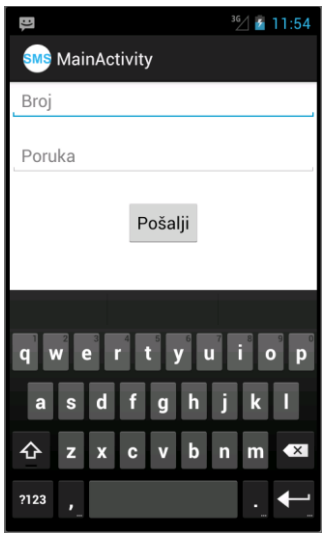
```
protected void posaljiPoruku(String noviBroj, String novaPoruka)
{
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(noviBroj, null, novaPoruka, null, null);
}
```

Za slanje poruka potrebna je klasa *SmsManager*. U kôdu koristimo metodu *getDefault()* uz pomoć koje dobijemo osnovnu instancu *SmsManager* klase. Nakon što smo kreirali instancu možemo iskoristiti metodu *sendTextMessage()* koja ima 5 argumenata. Nama su trenutno važni samo prvi argumenti, koji predstavlja broj primatelja poruke i treći argument koji predstavlja poruku koja se šalje. U primjeru aplikacije je dodano još nekoliko linija kôda unutar metode *posaljiPoruku()*, ali radi jednostavnosti je kôd ovdje skraćen i prikazan je samo osnovni dio koji je dovoljan za izvršavanje funkcije slanja poruke drugom uređaju, odnosno emulatoru. Osim

mogućnosti slanja poruka aplikacija mora imati i mogućnost primanja poruka što je realizirano sljedećim kôdom:

```
public void onReceive(Context context, Intent intent) {
     Bundle bundle = intent.getExtras();
     SmsMessage[] poruke=null;
     String tekst = "";
     if (bundle != null) {
           Object[] pdus = (Object[]) bundle.get("pdus");
           poruke = new SmsMessage[pdus.length];
           for(int i=0; i<poruke.length;i++){
                 poruke[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                 tekst += "Poruka od: " + poruke[i].getOriginatingAddress();
                 tekst += "\n";
                 tekst += poruke[i].getMessageBody().toString() + "\n";
           }
           Toast.makeText(context, tekst, Toast.LENGTH_LONG).show();
     }
}
```

Priloženi kôd je smješten u zasebnoj klasi koja se nalazi u datoteci *SMSPrimanje.java*, ime klase je *SMSPrimanje*. Ranije spomenuti kôd, koji služi za slanje poruka se nalazi u klasi MainActivity iz datoteke *MainActivity.java*. Implementacija koja je zaslužna za primanje poruka koristi pakete (engl. Bundle). U kôdu vidimo *if* uvjet koji provjerava da li je paket prazan ili ne. U slučaju da paket, odnosno *bundle*, sadrži neke podatke. Poruke ako postoje one su spremljene u polje *Object* u formatu PDU. PDU (Protocol Description Unit) format ne sadrži samo tekst poruke već i mnoge druge podatke o pošiljatelju kao što su podatci o pošiljateljevom SMS servis centru, vremenska oznaka i ostale informacije. U sljedećem dijelu kôda varijablu poruke koja je tipa *String* punimo sa postojećim podacima o primljenim porukama. Nakon toga sve što trebamo napraviti je iz te hrpe informacija izvući ono što nama treba. Spomenuto će se realizirati kroz jednu *for* petlju koja će izvući vrijednosti kao što je broj pošiljatelja i tekst poruke. Kada petlja završi spremljene vrijednosti iz varijable *tekst* će se ispisati na zaslonu aplikacije (Slika 4.5.).



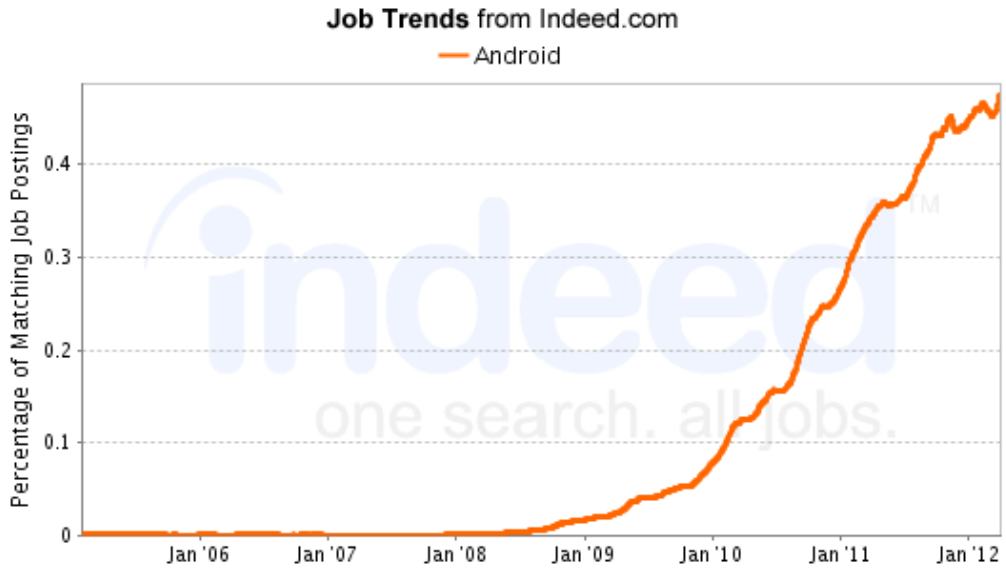
Slika 4.5. Grafičko sučelje aplikacije SMS

5. ZAKLJUČAK

Ovim radom je opisan Android operacijski sustav, njegova povijest o nastanku, opisane su i sve verzije spomenute tehnologije te struktura operacijskog sustava. Također je opisano razvojno okruženje te su dane četiri implementacije različitih aplikacija koje pružaju temeljnu osnovu za razumijevanje razvoja Android aplikacija u Eclipse razvojnom okruženju. Ono što je bitno naglasiti je to da razvoj aplikacija u spomenutom razvojnom okruženju karakterizira otvorenost, brzina, fleksibilnost i jednostavnost.

Pregledom povijest Android operacijskog sustava jasno je da se Android brzo razvija i kao operacijski sustav, ali i kao alat razvojnim inženjerima za razvoj aplikacija. Bitno je napomenuti da je svaka verzija bogatija novijim tehnologijama i alatima te samim time pruža veću mogućnost za razvoj inovativnih aplikacija.

Android je danas bez sumnje jedan od najrasprostranjenijih operacijskih sustava u mobilnom svijetu. Osim Androida tu su još vrijedni spomena Windwos Phone, iOS te Symbian. Jedan od razloga zašto je Android tako snažan na tržištu je Google Play. Google play prema novijim statistikama sadrži oko pola milijuna aplikacija i preko tri milijarde preuzimanja. Jedino tržište aplikacija koje je u rangu Androidovog je operacijskog sustava iOS. Veliki broj aplikacija na Google Playu je jasan pokazatelj da Android ima veliku zajednicu razvojnih inženjera. Razlog takve velike zajednice je taj što je Android razvojno okruženje potpuno besplatno i pruža brzo učenje uz mnogobrojnu literaturu koja je na raspolaganju svima. Još jedan dokaz o veličini Androida leži u povećanom rastu broja poslova na tržištu za razvojne inženjere (Slika 5.1.)



Slika 5.1. Prikaz povećane potražnje za Android razvojnim inženjerima

Kao što možemo vidjeti u zadnje četiri godine potražnja za Android razvojnim inženjerima je naglo porasla.

Stav je autora ovog teksta da se implementirane aplikacija koje su priložene u ovom radu ne mogu koristiti u realnom vremenu. Spomenute aplikacije mogu poslužiti za daljnji razvoj ideja u smislu nadogradnje određenih aplikacija ili korištenja određenih dijelova implementacija kako bi se izgradilo neko sasvim drugo programsko rješenje.

LITERATURA

1. Bucić T.: *Baza podataka za elektroničke knjižnice*, pp. 22-27, Diplomski rad, Osijek, 2012.
2. Dujmović A.: *Upravljanje videonadzorom cestovnog prometa mobilnim uređajem na android platformi*, pp. 9-11, Diplomski rad, Dubrovnik, 2010.
3. Marjanica A.: *GPS kolektor – prototip android aplikacije za prikupljanje prostornih podataka*, pp. 11-16 ,Zagreb, 2011.
4. Wei-Meng Lee: *Beginning Android Application development*, pp. 81-86, 149, 215-221, Wiley Publishing, Inc., Indianapolis, Indiana, 2011.
5. Chapple M.: *The Acid Model*, http://databases.about.com/od/specificproducts/a/acid.htm (Učitano 16.9.2012.)
6. Pavlović D.: *Što je i kako radi NFC?*, 2012., http://mob.hr/sto-je-i-kako-radi-nfc/ (Učitano: 11.9.2012.)
7. Penezić D.: *SQLite - najrašireniji transakcijska baza podataka*, 2012., http://sistemac.srce.unizg.hr/index.php?id=35&tx_ttnews%5Btt_news%5D=778&cHash=2e903c58643a650d171ee6e8314e3381 (Učitano: 16.9.2012.)
8. Pettersson L.: *SMS and the PDU format*, http://www.dreamfabric.com/sms/ (Učitano: 11.9.2012.)
9. Yadav M.: *History of Android*, 2012., http://www.tech2crack.com/history-Android/ (Učitano: 25.7.2012.)
10. Ziegler C.: *Android: A visual history*, 2011., http://www.theverge.com/2011/12/7/2585779/android-history (Učitano: 11.8.2012.)