Univerzitet u Beogradu
Matematički fakultet

Master rad

Sistem Android i njegovo prilagođavanje

Lazar Radovanović
Broj indeksa: 1020/2010

Mentor:
Dr Filip Marić
Matematički fakultet Univerziteta u Beogradu

Članovi komisije:
Dr Saša Malkov
Matematički fakultet Univerziteta u Beogradu

Dr Miroslav Marić
Matematički fakultet Univerziteta u Beogradu

Sadržaj

Predgovor

Operativni sistem Android se može sagledati iz nekoliko uglova. Najveći broj ljudi ga posmatra kao još jedan operativni sistem koji se nalazi na mobilnim telefonima, ne shvatajući ili uopšte ne ulazeći u to šta zapravo stoji iza tog imena. Drugi krug ljudi ga vidi kao jednu globalno popularnu softversku platformu, na kojoj mogu plasirati svoje proizvode u nadi da će dopreti do što većeg broja korisnika. I na kraju, tu su softverski inženjeri koji svakodevno žive sve njegove dobre i loše strane, gledajući kako ga mogu učiniti još boljim svojim svakodnevnim dostignućima. U ovom master radu, operativni sistem Android je prikazan iz ugla softverskih inženjera.

Uvodno poglavlje daje kratak uvid u istoriju ovog operativnog sistema. Govori se o njegovom korisničkom interfejsu, dostupnosti sistema na velikom broju uređaja i dobijanju naprednih korisničkih privilegija.

Drugo poglavlje je rezervisano za softversku arhitekuru operativnog sistema Android. Govori se o glavnim gradivnim elementima svakog softvera koji se izvršava na Android operativnom sistemu.

Ozbiljan razvoj softvera je gotovo nemoguće zamisliti bez upotrebe alata za kontrolu verzija koda. Android uvodi *Git* na velika vrata u svet programera, tako da je treće poglavlje posvećeno njemu.

Četvrto i peto poglavlje predstavljaju glavnu temu ovog master rada. Najpre se generalno govori o sistemu Android kao projektu otvorenog koda, njegovoj zajednici i mogućnošću uključivanja u sam razvoj. Nakon toga se upuštamo u modifikaciju samog operativnog sistema uz primer prilagođavanja operativnog sistema radi upotrebe u automobilu.

1 Uvod

1.1 Šta je operativni sistem Android?

Android je operativni sistem otvorenog koda, prvenstveno namenjen za prenosive uređaje osetljive na dodir, kao što su pametni telefoni i tableti. Zasnovan je na *Linux* [1] jezgru i trenutno postoje implementacije za ARM i Intel x86 arhitekture procesora [7].

Android projekat otvorenog koda (eng. *Android Open Source Project – AOSP*) je skup softvera i softverskih alata čijem razvoju upravlja kompanija *Google*. Sam Android razvijaju inženjeri kompanije Google sve dok poslednje izmene ne budu spremne za objavljivanje. U tom trenutku izvorni kôd postaje javno dostupan. Takav kôd je moguće pokrenuti bez modifikacija na izabranim uređajima (serija *Nexus* uređaja). Izvorni kôd se prilagođava od strane proizvođača hardvera (*OEM* [2]), kako bi ga bilo moguće pokrenuti na njihovom hardveru. Izvorni Android kôd (koji dolazi iz Google razvojnog centra) ne sadrži specifični upravljački softver (eng. *drivers*) koji je potreban za određene hardverske komponente (kao što su kamera, grafički procesor i druge). Google obezbeđuje velike nadogradnje sistema na svakih šest do devet meseci. Najnovija verzija u trenutnku pisanja nosi oznaku Android 6.0 – *Marshmallow* [3].

1.2 Istorija operativnog sistema Android

Kompanija Android je osnovana oktobra 2003. godine u Kaliforniji. Osnovali su je Endi Rubin i Rič Majner radi razvoja pametnih mobilnih telefona koji bi bili svesni lokacije svojih korisnika i njihovih potreba [4]. Prvobitna namera kompanije je bila razvoj operativnog sistema za pametne kamere ali kako to tržište nije bilo mnogo veliko, kompanija je pivotirala na razvoj operativnog sistema za mobilne uređaje koji bi se takmičio sa tada najpopularnijim sistemom *Symbian* [3]. Kompanija je poslovala tajno, otkrivajući samo da je radila na sistemu za mobilne uređaje. Osnivači kompanije su upali u finansiske probleme, pa je kompaniju kupio Google, 17. avgusta 2005. godine. Google se ovim potezom uključuje u segment mobilnih telefona, nudeći lako prilagodiv, fleksibilan i nadogradiv sistem proizvođačima mobilnih telefona kao i operaterima.

[1] Linux – operativni sistem zasnovan na modelu slobodnog i otvorenog razvoja i distribucije softvera.

[2] OEM (eng. *Original Equipment Manufacturer*) je kompanija koja koristi komponente ili proizvode druge kompanije kako bi izgradila sopstveni proizvod koji će prodavati pod svojim imenom.

[3] Symbian je operativni sistem namenjen za mobilne telefone koji trenutno razvija kompanija *Accenture.* Bio je najpopularniji sistem sve do 2010. godine kada ga je odmenio Android. Korišćen je od strane svih najvećih proizvođača hardvera kao što su: Nokia, Siemens, Motorola, Samsung, Sony-Ericsson, itd.

Novembra 2007. godine osnovano je udruženje *Open Handset Aliance* [4] sa ciljem razvoja otvorenih standarda za mobilne uređaje. Istog dana, Android (kao mobilna platforma zasnovana na Linux jezgru verzije 2.6) je predstavljen kao prvi proizvod ovog konzorcijuma. Prvi komercijalno dostupni telefon koji pokreće operativni sistem Android bio je *HTC Dream,* predstavljen 22. oktobra 2008. godine.

Svoju Nexus seriju uređaja Google predstavlja 2010. godine. Uređaje ove serije karakteriše uvek najnovija verzija Android operativnog sistema i to da su proizvod kompanije koje u tom trenutku Google proglasi za svog partnera. Google koristi ovu seriju mobilnih telefona za predstavljanje novih softverskih i hardverskih dostignuća u novim verzijama operativnog sistema.

Svako veliko izdanje operativnog sistema je nazvano, po abecednom redu, po nekom desertu ili slatkišu. Tako na primer verziji 1.5 – *Cupcake* je prethodila verziji 1.6 – *Donut*. Verovatno najveće unapređenje operativnog sistema je došlo sa verzijom 5.0 koja nosi naziv *Lollipop*. Najnovija verzija 6.0 je dobila naziv po poznatoj američkoj poslastici – *Marshmallow.*

1.3 Korisnički interfejs

Korisnički interfejs operativnog sistema Android je zasnovan na direktnom rukovanju prikazanim objektima na ekranu. Ulaz se uglavnom vrši putem dodira ekrana pokretima koji odgovaraju pokretima u realnom svetu. Tako, na primer u aplikaciji za čitanje novina, strane možete okretati baš kao što okrećete strane papirnih novina – pokretom prsta sa desne strane na levu. Dodatni hardver kao što je akcelometar, žiroskop i senzor brzine se koriste za dodatne zahteve korisnika kao na primer podešavanje orijentacije ekrana u zavisnosti od položaja uređaja, kontrolisanje vozila u simulaciji vožnje, brojanje koraka koje je korisnik prešao u toku neke fizičke aktivnosti i slično.

Prilikom pokretanja operativnog sistema korisniku se prikazuje početni ekran koji je aplikacija koja služi za pokretanje drugih aplikacija (eng. *launcher),* prikaz odabrane pozadine i omogućava manipulaciju interaktivnim komponentama sistema - *Widgets* [5]. Svaka aplikacija je predstavljena svojom ikonicom koja se nalazi u listi instaliranog softvera telefona. Kao i svaka druga aplikacija, i početni ekran operativnog sistema Android se može prilagoditi ili čak zameniti nekom drugom aplikacijom. Ova funkcionalnost omogućava korisnuku dodatne pogodnosti prilikom prilagođavanja sistema. Na primer, sistem se može transformisati u specijalizovani softver za automobile koji bi se zajedno sa hardverom ugrađivao u kontrolnu tablu vozila.

[4] Open Hendset Alijans (OHA) je konzorcijum sa ciljem donošenja standarda za mobilne uređaje. Čine ga kompanije kao što su: Google, HTC, Sony, Dell, Intel, Motorola, Qualcomm, Samsung, Nvidia, T-Mobile, itd.

[5] Widget – predstavljaju jednu od najbitnijih komponenti početnog ekrana Android operativnog sistema. Mogu se posmatrati kao efikasan i lako vidljiv prikaz najbitnijih karakteristika neke aplikacije. Korisnik ih po želji može dodavati, sklanjati i menjati im veličinu.

Na vrhu početnog ekrana se obično nalazi statusna linija koja prikazuje važne informacije o samom uređaju i njegovom povezivanju. Tu se nalazi trenutno stanje napunjenosti baterije, datum i jačina signala mobilne mreže. Statusna linija se može proširiti pokretom na dole čime se prikazuju dodatne informacije koje generišu druge aplikacije na samom uređaju.

1.4 Problemi vezani za širok spektar Android uređaja

Operativni sistem Android je projekat otvorenog koda i njegovo korišćenje se ne naplaćuje. To otvara beskrajne mogućnosti njegove modifikacije i prilagođavanja. Kao direktna posledica otvorenosti sistema Android je i ogroman broj hardverskih uređaja na kojima se on može koristiti. Prema istraživanju kompanije *OpenSignal* [6] postoji 18.796 različitih uređaja [8]. Pored svih modela telefona postoje i specijalizovani sistemi zasnovani na ovom operativnom sistemu kao što su pametni televizori, ručni satovi, pametne naočare, slušalice, portabilni muzički uređaji, kontroleri za vide igre itd. Upravo spomenuta raznovrsnost hardvera je i jedan od najozbiljnijih problema sa kojima se suočavaju programeri operativnog sistema Android. Uzmimo kao primer razvoj aplikacije koja treba da se izvršava na svim uređajima od neke unapred zadate verzije operativnog sistema (na primer od verzije 4.0 – *Ice cream sandwitch*). Spektar ovakvih uređaja je ogroman. U trenutku pisanja 94% aktivnih uređaja imaju ovu ili noviju verziju sistema, što znači da će velika većina korisnika videti vašu aplikaciju na *Google Play Store* prodavnici.

Jedan od najozbiljnijih problema sa kojim se svaki programer Android aplikacije mora izboriti je prilagođavanje korisničkog interfejsa širokom spektru rezolucija i veličina ekrana. U praksi, ovo znači izradu više različitih verzija samog korisničkog interfejsa. Tako, na primer, imaćete jednu verziju interfejsa (što uključuje i propratne grafičke resurse, ikone, itd.) za uređaje sa velikim ekranima (tablet uređaje), jednu verziju za telefone sa malim ekranima i jednu verziju za telefone sa velikim ekranima. Ovome treba pridodati da za svaku verziju morate razmišljati i o orijentaciji ekrana (da li korisnik drži telefon normalno ili zarotirano – *portrait or landscape*) što obično znači još dodatnog posla.

Osim ogromnih problema koje donose raznovrsni ekrani koji se ugrađuju u Android uređaje, svaki programer aplikacija za ovaj operativni sistem mora voditi računa i o SDK [7](eng. *Software development kit*) verziji koja se nalazi na uređaju. Na primer, od SDK verzije može zavisiti da li uređaj podržava *Bluetooth* tehnologiju i ukoliko ne bezbolno obavestiti korisnika da ne može koristiti određene funkcionalnosti aplikacije.

[6] OpenSignal je kompanija specijalizovana za bežično povezivanje uređaja. 2013. je proglašena najinovatnijom kompanijom koja se bavi mobilnim tehnologijama.

[7] SDK (eng. *Software Development Kit)* predstavlja skup alatki koje omogućuju izradu aplikacija za određen softverski paket, računarski sistem, video igru, operativni sistem i slično.

1.5 Dobijanje naprednih korisničkih prava

Android je zasnovan na operativnom sistemu Linux koji kao što je poznato pokreće veliki broj ogromnih servera i individualnih računara. Očekivano, programeri ali i svi oni koji vole informacione tehnologije, žele da otključaju sve skrivene mogućnosti sistema Android i podese ga po sopstvenom nahođenju. Preciznije, da postanu specijalni *root* korisnik dobijanjem naprednih korisničkih prava (ili drugačije - rutovanjem uređaja). Pored slobode i moći koja se postiže dobijanjem naprednih korisničkih prava određenog uređaja, otvara se i (vrlo izgledna) mogućnost fatalne greške i gubljenja svih podataka. Dobijanje naprednih korisničkih prava samog uređaja je prilično jednostavno imajući u vidu količinu lako dostupnih informacija na internetu. Dovoljno je samo potražiti upustvo za dobijanje naprednih korisničkih prava uređaja koji koristite i vrlo brzo ćete doći do lepo formulisanih koraka koje samo treba jednostavno ispratiti. Neke od najvećih prednosti naprednih korisničkih prava su:

1. Instaliranje ne odobrenih (ili zabranjenih) aplikacija od strane kompanije Google.
2. Unapređenje verzije operativnog sistema sa ne zvaničnim verzijama koje mogu doći od samog proizvođača mobilnog uređaja (*HTC, Sony, Samsung*…), ne zavisnih zajednica (na primer *CyanogenMod*) ili sopstveno prevođene verzije sistema.
3. Podizanje ili spuštanje takta procesora (eng. *overclocking or underclocking).*
4. Uklanjanje nepoželjnih aplikacija.

2 Arhitektura operativnog sistema Android

2.1 Osnovne komponente operativnog sistema

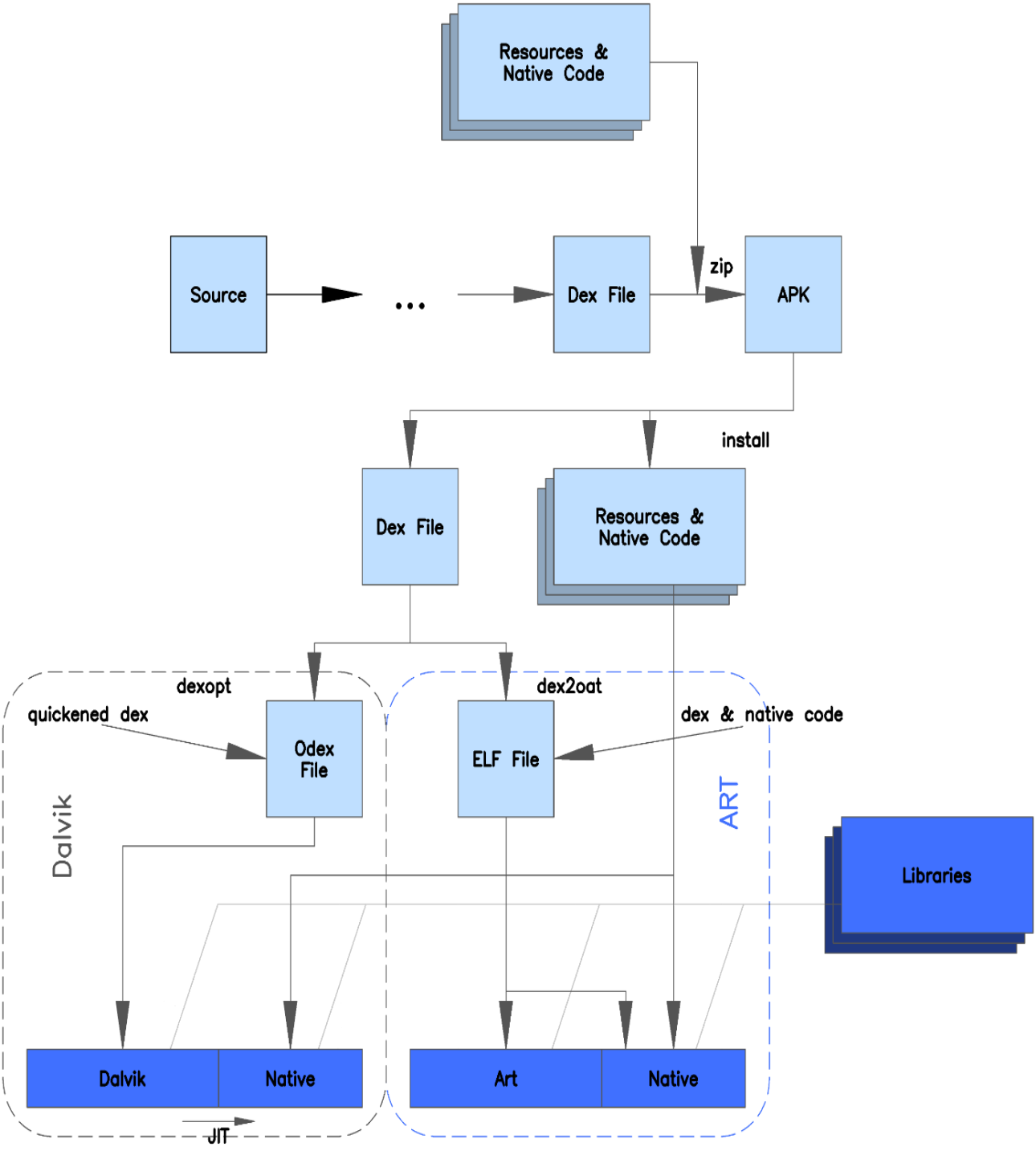
Prema mišljenju mnogih, Android je samo još jedna od distribucija operativnog sistema Linux. Od novembra 2013. godine trenutne verzije operativnog sistema Android su zasnovane na Linux jezgru verzije 3.x dok je na starijim verzijama (starije od 4.0) jezgro zasnovano na Linux jezgru verzije 2.6.x. Iako određene funkcije kojima je Google unapredio Linux jezgro, kao što je funkcija za upravljanje i potrošnju električne energije, nisu prihvaćene od strane mnogih programera iz Linux zajednice, Linus Torvalds je izjavio da će Android i Linux možda imati zajedničko jezgro u budućnosti ali da to ipak neće biti slučaj u narednih četiri ili pet godina.

Modifikovano Linux jezgro koje pokreće Android sistem je napisano u programskom jeziku C i direktno je odgovorno za: komunikaciju sa hardverom na kome se operativni sistem izvršava, upravljanju procesa i upravljanju memorijom. Iznad jezgra se nalaze sistemske biblioteke pisane u programskom jeziku C++, koje su odgovorne za rad sa grafikom, fontovima, multimedijom, čuvanjem podataka, obradom veb strana, itd.

Na njih se nadovezuje izvršni upravljač Android aplikacija - *Android Runtime* (*ART* ili njegov prethodnik *Dalvik*). Izvršni upravljač predstavlja neophodni skup alata i resursa (bajt kôd, virtuelne mašine i slično) potrebnih da se pokrenu i uspešno izvršavaju Android aplikacije. ART i Dalvik su kompatibilni upravljači izvršavanja koji pokreću *Dex bytecode*, tako da aplikacije razvijene za Dalvik mogu raditi i sa ART izvršnim upravljačem. Slika 2.1 prikazuje proces prevođenja Android modula (fajlovi sa ekstenzijom *.java)* i razliku između izvršnih upravljača izvršavanja [5].

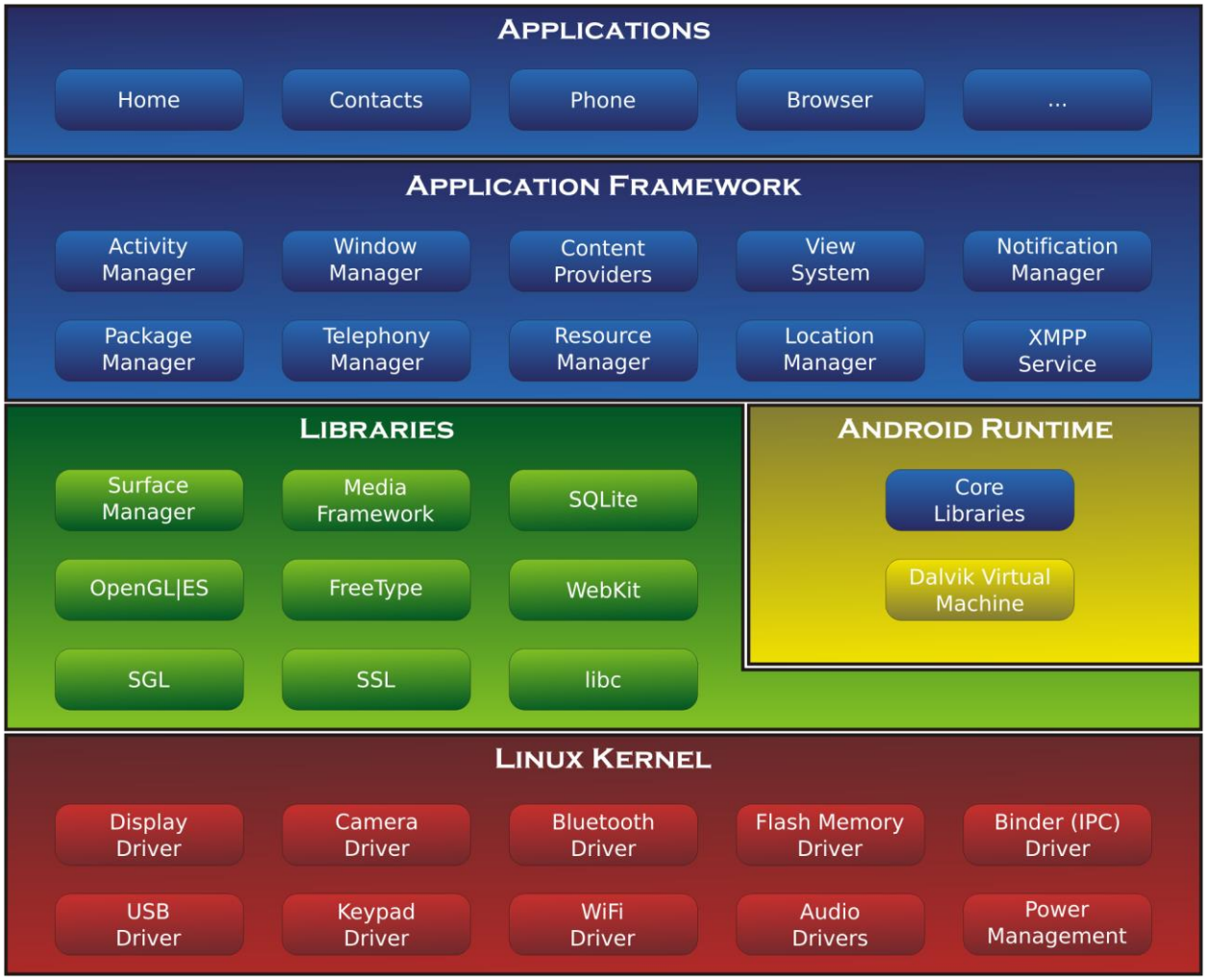
Ovde možemo postaviti pitanje zašto se Android izvorni kôd direktno ne prevodi u Dalvik bajt kôd, a odgovor možemo naću u istoriji razvoja sistema Android kada su programeri odlučili da se zasnivaju na Java bajt kodu a ne na samom programskom jeziku Java zbog učestalih promena i stalnih unapređenja koje su pratile razvoj programskog jezika Java davne 2005. godine. U isto vreme bajt kôd je bio i ostao zamrznut što je u mnogome olakšalo posao u ranoj fazi razvoja samog operativnog sistema. Neke od najznačajnijih prednosti novog ART izvršnog upravljača u odnosu na Dalvik su:

1. prevođenje ispred vremena (eng. *Ahead of time compilation),*
2. poboljšano sakupljanje otpadaka (eng. *Improved garbage collection),*
3. razvoj i otklanjanje grešaka su unapređeni,
4. unapređeni detalji prilikom bacanja izuzetaka i iznenadnog prestanka rada aplikacija.



Slika 2.1: Proces prevođenja

Iznad ART upravljača se nalazi aplikacioni okvir (eng. *Applicaton Framework)* napisan u programskom jeziku Java, gde se nalaze sve komponente bitne za same programere. Tu su razni upravljači koji se brinu za rad sa akcijama, prozorima, resursima, servisima, notifikacijama i ostalim komponentama jedne Android aplikacije (Slika 2.2).



Slika 2.2: Arhitektura operativnog sistema Android

Za distribuiranje Android aplikacija u vidu binarnih fajlova koristi se *.apk* format. Pored programskog koda (u obliku fajlova sa *.dex* ekstenzijom) u njemu su sadržani resursi aplikacije, sertifikati i drugi elementi neophodni sistemu za proces instalacije.

Sistemska memorija za skladište podataka na Android uređajima je podeljena na nekoliko particija. Postoji */system* particija koja se koristi za sistemske programe, tj aplikacije koje dolaze preinstalirane uz sam uređaj kao što su aplikacije za rad sa kontaktima, podešavanje telefona, obavljanje poziva, slanje SMS poruka i druge. Pored nje, */data* particija služi za skladištanje korisničkih podataka i novo-instaliranih aplikacija. Spoljna memorija (memorijske kartice) se u Android operativnom sistemu vezuje za specijalne particije koje se obično zovu po tipu spoljne memorije: */sdcard1, /sdcard2…*

2.2 Razvoj softvera na operativnom sistemu Android

Razvoj aplikativnog softvera za Android operativni sistem podrazumeva pisanje koda u programskom jeziku Java uz pomoć softverskog paketa specifičnog za Andriod. U okviru ovog paketa nalazi se veliki broj neophodnih i korisnih alata uz pomoć kojih se prevode, debaguju i testiraju aplikacije.

Prilikom razvoja softvera za Android operativni sistem, programeri koriste sledeće gradivne blokove uz pomoć kojih grade veće celine:

1. Aktivnosti (eng. *Activity*),
2. Namere (eng. *Intents),*
3. Servise (eng. *Services),*
4. Dobavljače podataka (eng. *Content providers),*
5. Primaoce obaveštenja (eng. *Broadcast receivers).*

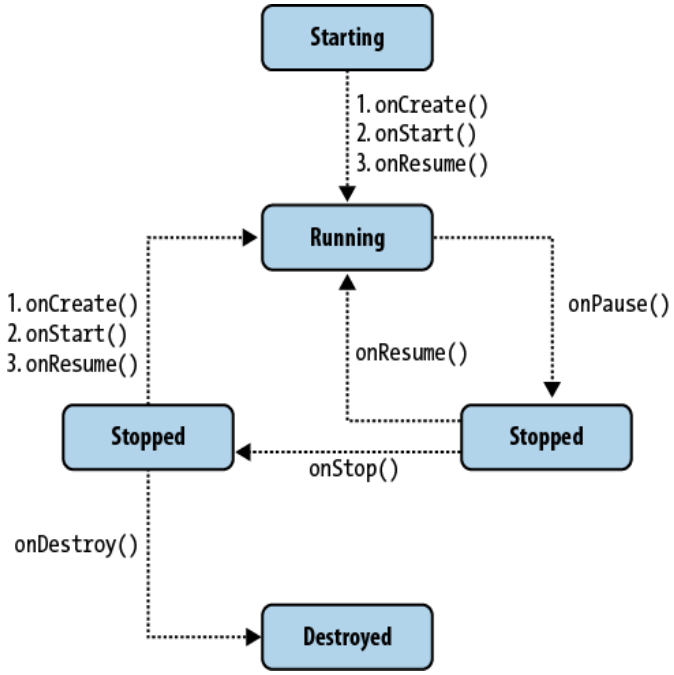
Kada se razmišlja o aplikaciji koju treba napisati dobro je prvo sagledati širu sliku. Potrebno je definisati kako će komponente aplikacije komunicirati, koji će biti glavni scenariji korišćenja aplikacije i da li će sve to skupa imati smisla. Na početku je dovoljno imati par grubih crteža koji će predstavljati glavne ekrane aplikacije, spisak njenih mogućnosti i način na koji će aplikacija pomoći u rešavanju stvarnog životnog problema. Gradivni elementi sistema Android samo treba da pomognu programerima da svoje zamisli sprovedu u delo.

2.2.1 Aktivnosti

Aktivnost (eng. *activity)* je obično jedan ekran koji korisnik vidi na ekranu u jednom trenutku. Obično aplikacija ima više aktivnosti, i omogućava korisniku da prelazi sa jedne na drugu. Isto tako korisnik može vrlo lako preći u aktivnost druge aplikacije, završiti neki posao i vratiti se natrag u baznu aplikaciju. Na primer, prilikom boravka u Imenik aplikaciji, korisnik može aktivirati dugme za slanje tekstualne poruke nekom svom poznaniku i biti preusmeren u potpuno drugu aplikaciju za sastavljanje poruka.

Pokretanje aktivnosti može biti vrlo skupo. Ono uključuje startovanje novog Linux procesa, alociranje potrebne memorije za sve resurse, učitavanje svih komponenti grafičkog interfejsa, itd. Iz tog razloga Android sistem čuva pokrenute aktivnosti u radnoj memoriji uređaja u stanju pripravnosti radi mnogo bržeg ponovnog pokretanja. Naravno, u koliko se ispostavi da više nema dovoljno prostora u radnoj memoriji uređaja Android će nasilno ubiti pauzirane aktivnosti i osloboditi memoriju. Sve ove aktivnosti obavlja upravljač aktivnostima (eng. *Activity Manager).*

Životni cikulus aktivnosti u operativnom sistemu Android je prikazan na slici 2.3.



Slika 2.3: Životni ciklus aktivnosti

Kada aktivnost ne postoji u memoriji ona je u stanju pokretanja (eng. *starting)*. Prilikom pokretanja, aktivnost će proći kroz skup poziva od strane sistema koje programer može popuniti prilikom razvoja aplikacije. Po završetku startovanja, aktivnost je u fazi izvršavanja (eng. *running*). U ovoj fazi korisnik može imati interakciju sa korisničkim interfejsom, što dalje znači da može postojati samo jedna aktivnost u stanju izvršavanja u jednom trenutku. Sistem aktivnosti u ovom stanju daje najviši prioritet u smislu resursa (memorije i procesorskog vremena). Isto tako aktivnost mora brzo odgovarati na akcije korisnika, inače je sistem može proglasiti problematičnom i ubiti. Slučajevi kada aktivnost ne odgovara na akcije korisnika (eng. *Activity not responing – ANR*) nisu tako retki u operativnom sistemu Android i zato se mora voditi računa da se celokupno zahtevno procesiranje premesti u pozadinski proces kako ne bi opteretilo glavnu korisničku nit izvršavanja (eng. *Main or UI Thread)*.

Ukoliko aktivnost izgubi fokus, ona prelazi u stanje pauze (eng. *paused)*. Aktivnost je i dalje delimično vidljiva, ali neki drugi program ima fokus – on je u stanju izvršavanja. Svaka aktivnost prolazi iz stanja pauze u stanje potpune ne aktivnosti – stop stanje (eng. *stopped)*. Tada aktivnost nije vidljiva ali je još uvek u memoriji. Iz ovog stanja korisnik može vrlo brzo vratiti aktivnost u stanje izvršavanja ili sistem može odlučiti da ubije aktivnost i oslobodi memoriju. Kada aktivnost nije u memoriji ona je u stanju uništenja (eng. *destroyed)*.

2.2.2 Namere

Namere (eng. *intents)* u operativnom sistemu Android predstavljaju poruke koje se mogu slati između različitih komponenti sistema. One najčešće služe da pokrenu neku aktivnost ili servis i predstavljaju obaveštenje da se neki događaj odigrao (eng. *broadcast)*. Ako želimo da odreagujemo na neko određeno obaveštenje, potrebno je napraviti objekat klase primaoca obaveštenja (eng. *broadcast receiver)* i zatim se registrovati za sva obaveštenja od interesa (poglavlje 2.2.5). Namere su implementirane asinhrono, što znači da deo koda koji ih kreira i

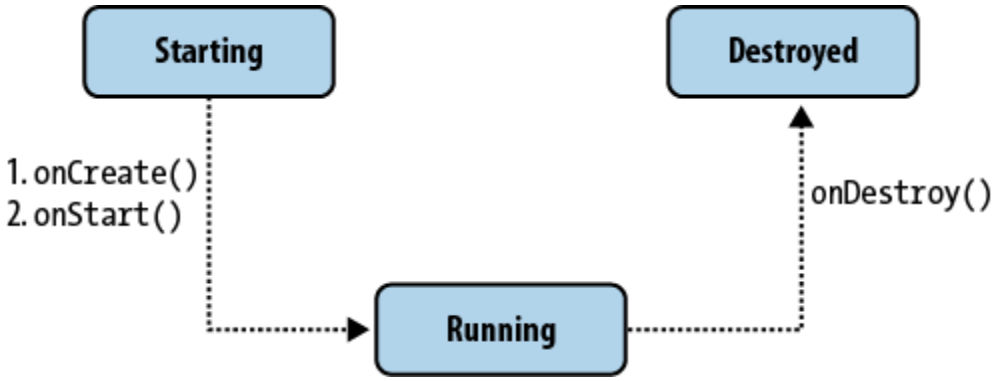
šalje ne mora da vodi računa o tome da li su izvršene ili ne. Kako namere mogu biti poslate sa određenim skupom podataka one se mogu posmatrati i kao jedan od načina ostvarivanja među-procesne komunikacije. Namere se dele u dve kategorije:

1. Implicitne namere (tvorac namere samo određuje tip primaoca).
2. Eksplicitne namere (u ovom slučaju tvorac namere tačno određuje ko će biti primaoc).

Na primer, u slučaju implicitne namere, programer može reći da želi da neko (neka aplikacija, nije bitno koja) otvori internet stranicu sa prosleđenom *URL* adresom. U drugom slučaju programer ne želi da sistem kontroliše ko će prikazati internet stranicu, već on eksplicitno određuje da specifičnoj aktivnosti koja ima mogučnost obrade veb strana bude dodeljen taj posao.

2.2.3 Servisi

Servisi (eng. *services)* se izvršavaju u pozadini i nisu im pridružene komponente korisničkog interfejsa. Glavna zabluda oko Android servisa je da oni dobijaju posebnu nit za izvršavanje, po ugledu na druge operativne sisteme, što ovde nije slučaj. Naime, oni dele istu (glavnu) nit izvršavanja i zato treba biti oprezan. Ukoliko servis radi veliku količinu posla treba sistemu ekplicitno reći da ga pokrene u posebnoj (pozadinskoj) niti. Postoje i specijalni servisi koji se podrazumevano pokreću u pozadinskoj niti – servisi namere (eng. *Intent Services)*. Životni vek servisa je prikazan na slici 2.4.



Slika 2.4: Životni vek servisa

Kao što se može videti na slici, servisi imaju znatno prostiji životni ciklus od aktivnosti. Takođe, prelaske iz jednog stanja u drugo može kontrolisati programer.

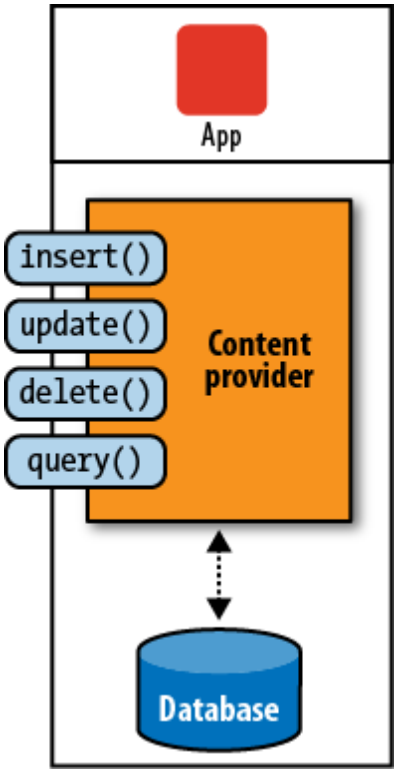
2.2.4 Dobavljači podataka

Kod operativnog sistema Android svaka aplikacija se izvršava u izolaciji ostavljajući malo prostora za deljenje svojih podataka. Upotrebom namera možemo preneti male količine podataka ali za trajne, velike količine podataka, treba koristiti dobavljače podataka (eng. *Content Providers).* Dobavljači podataka predstavljaju interfejse za deljenje podataka između

aplikacija. Zasnovani su na CRUD[8] principu (slika 2.5) i na neki način predstavljaju proksi ka bazama podataka.

Programerima su dostupni različiti sistemski dobavljači podataka koje može koristiti. Na primer, ukoliko je potrebno dobiti listu svih telefonskih brojeva iz korisničkog imenika možemo se obratiti dobavljaču kontakata (eng. *Contacts Provider),* a ako je ipak potrebno pročitati trenutna podešavanja korisnika, pomoć trebamo potražiti u okviru dobavljača podataka za podešavanja (eng. *Settings Provider).*

Dobavljači podataka pružaju veliku fleksibilnost u komuniciranju između komponenti sistema. Uz pomoć njih možemo pročitati i promeniti širok spektar podataka, počev od kontakata korisnika, preko sistemskih podešavanja (bežične mreže, grafičke teme, fontovi…), do organizacije fajl sistema korisnika. Koristeći fleksibilnosti operativnog sistema Android veliki proizvođači telefona se odlučuju da naprave potpuno novo grafičko okruženje, ostavljajući na taj način svoj pečat i poboljšavajući iskustvo korisnika. Takođe, programeri mogu imati svoje verzije sistemskih aplikacija.



Slika 2.5: Dobavljači podataka

2.2.5 Primaoci obaveštenja

Predstavljaju implementaciju šablona posmatrač (eng. *observer pattern)* u okviru Android operativnog sistema. Dakle, postoje događaji na koje se možemo registrovati, i u trenutku njihovog nastanka odreagovati na nama prikladan način. U operativnom sistemu Android se informacije o događajima stalno šalju. Na primer, ako je SMS poruka pristigla, baterija je pri kraju, ili imamo dolazni poziv sistem će poslati događaj o tome.

[8] CRUD – nastao je kombinovanjem četiri engleske reči (***C**reate, **R**ead, **U**pdate, **D**elete)* koje predstavljaju četiri osnovne funkcionalnosti koje se mogu primeniti nad trajno sačuvanim podacima (obično su to baze podataka).

Programeri imaju mogućnost da se pretplate na bilo koji od sistemskih događaja ili da sami pošalju sopstveno obaveštenje (događaj). Primaoci obaveštenja nemaju svoj korisnički interfejs, niti se mogu videti kao aktivni procesi u memoriji ali kada se dogode, registrovani primaoci dobijaju mogućnost da izvrše određeni kôd. Važno je napomenuti da sistem neće ovde dozvoliti kompleksne poslove, već samo proste pozive u vidu startovanja aktivnosti ili servisa. Čak šta više, Android će ograničiti vreme trajanja obrade obaveštenja na neku fiksnu vrednost (trenutno deset sekundi).

Ukoliko nije potrebno slati obaveštenja između aplikacija već samo u okviru sopstvenog programa Android nudi specijalan oblik slanja i primanja obaveštenja – lokalne primaoce obaveštenja (eng. *local broadcasters)*. Koristeći lokalni mehanizam za obaveštenja dobija se mnogo efikasnija implementacija primaoca zato što se na taj način eliminišu dodatne obrade za među-procesne komunikacije (eng. *cross-process communication*). Osim efikasnosti, lokalni mehanizam za obaveštenje ima i drugih prednosti:

1. ograničenje na svoju aplikaciju (znamo da obaveštenje neće napustiti našu aplikaciju i zato ne moramo voditi računa o curenju privatnih informacija),
2. nije moguće da druge aplikacije pošalju ove tipove obaveštenja našoj aplikaciji pa samim tim se i ne ostavlja sigurnosna rupa koju mogu iskoristiti zlonamerni korisnici.

2.2.6 Okruženje Android aplikacije

Aktivnosti, servisi, primaoci obaveštenja i namere žive u istom okruženju ili kontekstu aplikacije (eng. *Application Context)*. Kontekst omogućava aplikaciji da deli podatke između svojih komponenti. Aplikacioni kontekst se kreira prilikom prvog startovanja neke od aktivnosti ili servisa aplikacije i ostaje živ sve dok aplikacija ne bude ubijena i memorija oslobođena, što znači da ne zavisi od životnog veka aktivnosti.

3 Sistem za kontrolu verzija koda – Git

Jedan od ključnih izazova, u ranoj fazi razvoja operativnog sistema Android, bio je kako na najbolji način iskoristiti otvorenu filozofiju projekta i obezbediti podjednako dobar tretman za velike kompanije ali i za nezavisne programere (hobiste) koji žele doprineti razvoju projekta. Rešenje ovog problema se ogleda u tome da je celokupan izvorni kôd projekta pod kontrolom distribuiranog sistema za upravljanje verzijama dokumenata. Kako bismo preuzeli izvorni kôd Android projekta, odabrali verziju na kojoj želimo raditi i sačuvali trenutne promene, itd, potrebno nam je poznavanje softverskog alata za upravljanje verzijama dokumenata koji Android koristi – *Git.*

3.1 Osnovno o sistemima za kontrolu verzija

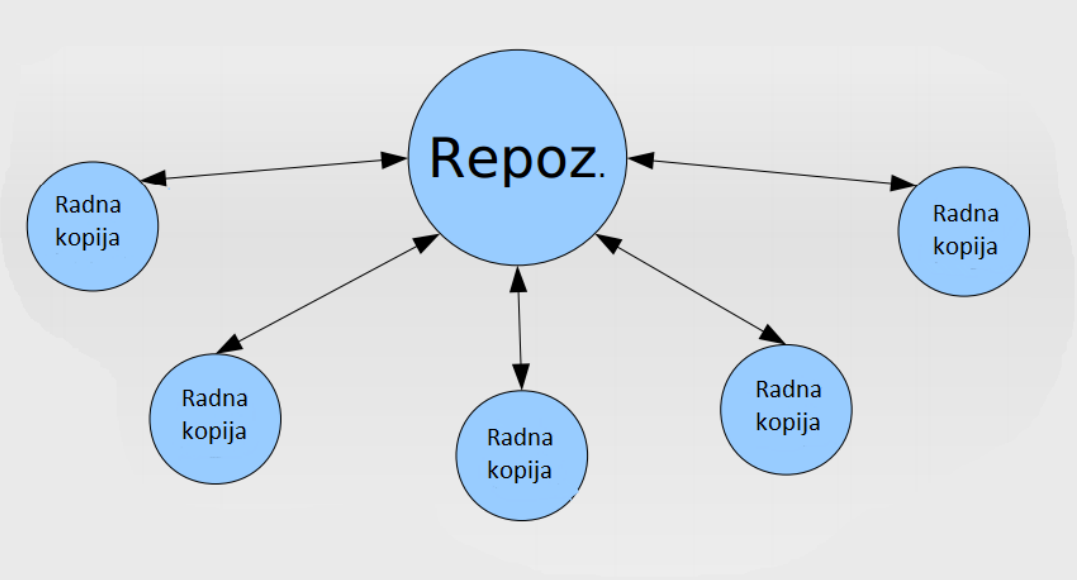
Kontrola verzija (eng. *Revision control)* predstavlja upravljanje verzijama dokumenata, izvornih datoteka računarskih programa, veb-sajtova i drugih skupova informacija. Promene se obično označavaju brojem ili nekim alfanumeričkim kodom koji se još naziva revizioni broj (eng. *revision number)* ili prosto revizija (eng. *revision).* Na primer, inicijalni skup fajlova se može označiti revizijom 1, a onda nakon svake promene samo uvećati broj revizije. Svakoj reviziji je dodeljeno i vreme koje označava kada su promene načinjene kao i ime osobe koja ih je napravila. Korisnik može čitati istoriju revizija, obnavljati/uklanjati pojedine verzije, vršiti poređenja između verzija ili objedinjavati više verzija u jednu [2].

Sistem za upravljanje verzijama softvera (eng. *Version control system*) je računarski program za kontrolu verzija izvornog koda. Danas su široko prihvaćeni i predstavljaju osnovni alat svakog tima koji se bavi razvojem softvera.

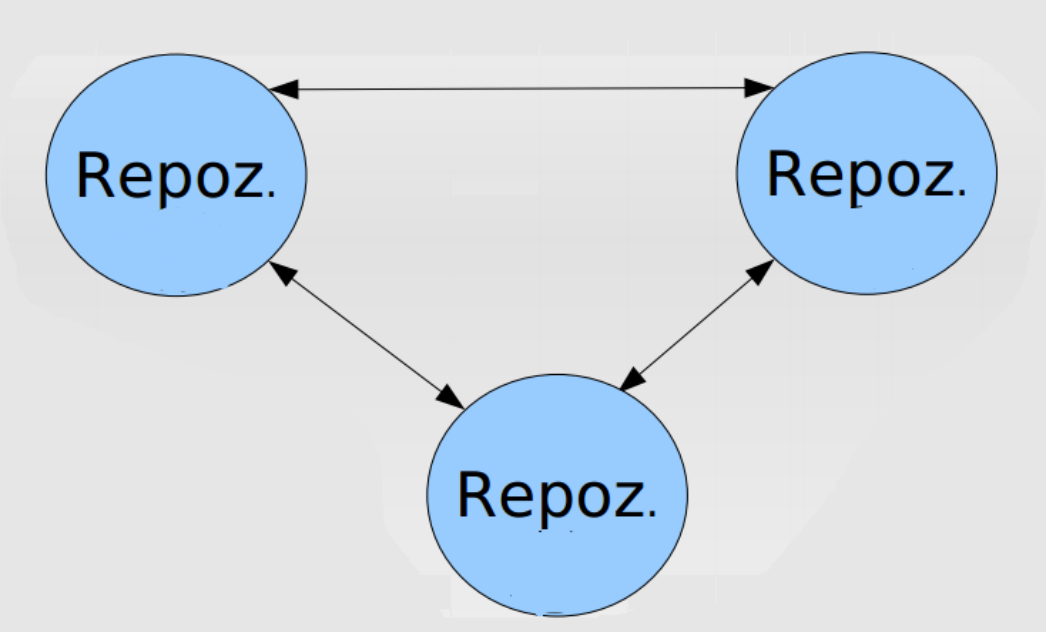
Prednosti ovakvih sistema su višestruke:

- omogućava se relativno „bezbolan” rad više programera na jednom softverskom projektu što je bez ovakvog sistema moguće samo uz stalnu saradnju programera uključenih u projekat (uz napomenu da ni stalna saradnja nije dovoljno efikasna ako projekat nije izrazito modularne prirode ili je broj programera veći od dva),
- omogućava se grananje projekta i kasnije spajanje grana,
- distribuirani sistemi upravljanja izvornim kodom omogućavaju nesmetan razvoj i bez konstantne veze sa središnjom bazom odnosno skladištem izvornog koda.

Sistemi za upravljanje verzijama softvera se mogu podeliti na centralizovane i decentralizovane u zavisnosti od načina na koji je implementiran sam repozitorijum (Slika 3.1 i Slika 3.2).



Slika 3.1: Centralizovan sistem za upravljanje verzijama softvera



Slika 3.2: Decentralizovan sistem za upravljanje verzijama softvera

3.2 Osnovno o Git-u

Git je decentralizovani sistem za kontrolu i upravljanje verzijama softvera sa naglaskom na brzini, integritetu podataka i podrškom za distributivni nelinearni proces rada. Kao i većina drugih distribuiranih sistema za kontrolu verzija, svaki Git repozitorijum je punopravni

primerak softverskog projekta sa kompletnom istorijom promena ne zavistan od internet konekcije ili centralnog servera. Tvorac Git sistema je Linus Torvalds[9] koji ga je razvio prilikom rada na Linux jezgru i od tada je široko prihvaćen kao alat za kontrolu verzija softvera [1].

3.3 Karektiristike Git-a

Dizajn i arhitektura Git-a su posledica Torvarldovog iskustva sa operativnim sistemom Linux i radom na ogromnom distribuiranom projektu kakav je operativni sistem. Ovo iskustvo ga je vodilo prilikom donošenja svih bitnih implementacionih odluka.

Snažna podrška ne linearnom razvoju softvera

Git podržava brzo grananje i spajanje i uključuje specifične alate za vizuelizaciju i navigaciju kroz nelinearnu razvojnu istoriju projekta. U Git-u se pretpostavlja da će određena promena biti višestruko integrisana (eng. *merged*) kako bude prolazila kroz različite pregledače koda (eng *reviewers*). Grane u Git-u su lagane i predstavljaju samo reference na određenu sačuvanu izmenu (eng. *commit),* na osnovu koje se može rekonstruisati kompletna struktura projekta.

Distribuirani razvoj

Git svakom programeru daje lokalnu kopiju kompletne istorije razvoja projekta. Promene se mogu kopirati sa jednog repozitorijuma na drugi i predstavljene su kao dodatne razvojne grane koje se mogu spajati na lak način sa drugim repozitorijumima.

Kompatibilnost sa postojećim sistemima i protokolima

Repozitorijumi mogu biti objavljeni putem HTTP, FTP, rsync ili Git protokola koristeći običnu ili zaštićenu konekciju (eng. *SSH - Secure Shell protocol*).

Efikasna kontrola velikih projekta

Kako pojedini projekti mogu postati preveliki, Git omogućava da se repozitorijum može preuzeti samo delimično (do određenog trenutka u prošlosti) ili da se preuzme samo jedna grana projekta.

[9] Linus Benedict Torvarlds – je Finski softverski inženjer, poznat po razvoju jezgra operatvinog sistema Linux.

Kriptografska autentifikacija istorije

Istorija u Git-u je organizovana tako da identifikacioni broj (eng. *Identification number – ID*) svake sačuvane izmene zavisi od cele istorije projekta do te specifične izmene. Kada je izmena objavljena, nemoguće je promeniti stare verzije bez jasnog traga u istoriji projekta.

Dizajn zasnovan na malim uslužnim programima

Git je dizajniran kao skup malih programa pisanih u programskom jeziku C. U okviru programskog paketa Git se nalaze i mnogobrojne skripte koje proširuju funkcionalnosti bazičnih Git programa i dodatno pojednostavljuju najčešće korišćene procedure.

Automatsko skupljanje otpadaka

Nakon pokretanja određenih Git komandi ili eksplicitnom komandom Git će izvršiti pregled repozitorijuma u cilju uklanjanja svih nepotrebnih fajlova, optimizacije repozitorijuma i poboljšavanju performansi sistema.

3.4 Instaliranje i konfigurisanje Git-a

Instaliranje Git-a na operativnom sistemu Linux bi trebalo da prođe bez većih problema. Iz komandne linije je samo potrebno pokrenuti sledeću komandu:

```
$ apt-get install git
```

Kada se Git konačno nađe na vašem sistemu, potrebno je izvršiti jednokratnu konfiguraciju kako biste ga prilagodili svojim potrebama. Git dolazi sa jednom vrlo zgodnom alatkom koja se naziva *git config.* Korišćenje ove jednostavne alatke omogućava lako i brzo podešavanje svih parametara Git-a, kao i kontrolisanje njegovog pojavljivanja i ponašanja. Podešavanja su raspoređena u tri nivoa:

1. Sistemski nivo (/etc/gitconfig) – Sadrži vrednosti za svakog korisnika na sistemu i sve njegove repozitorijume. Ako se prosledi opcija –*system* piše se u ovaj fajl.
2. Korisnički nivo (~/.gitconfig) – Pristupamo mu ako prosledimo --*global* parametar u okviru *git config* komande.
3. Projektni nivo (.git/config) – Odnosi se na repozitorijum u kome se korisnik trenutno nalazi.

Sistemski nivo ima najmanji prioritet, što znači da će podešavanja iz korisničkog nivoa sakriti podešavanja u sistemskom nivou. Slično važi i za projektni nivo koji ima najviši prioritet [1].

Prva stvar koju je neophodno podesiti je korisničko ime i e-mail adresa. Ove informacije su od vitalnog značaja zato što su upakovane u svaku Git izmenu (eng. *commit*). Podešavamo ih sledećim komandama:

```
$ git config --global user.name "Petar Perić"
$ git config --global user.email petar@gmail.com
```

Nakon neophodnih podešavanja korisničkih informacija, možemo izabrati podrazumevani editor teksta. Interfejs ovog editora će se pojaviti u situaciji kada je Git-u potrebna naša tekstualna poruka, na primer prilikom čuvanja nove izmene:

```
$ git config --global core.editor emacs
```

Još jedna korisna opcija je alatka za razrešavanje konflikta koji mogu nastati u toku spajanja fajlova. Ovde se kao korisno pokazalo korišćenje programa sa grafičkim okruženjem. Najčešće korišćeni alati su *Meld* i *Kdiff3:*

```
$ git config --global merge.tool kdiff3
```

I na kraju, uvek je korisno proveriti trenutna podešavanja. To možemo uraditi jednostavnim pokretanjem sledeće komande:

```
$ git config --list
```

3.5 Osnovne komande Git-a

Pre nego što možemo preuzeti izvorni kôd projekta na kome radimo (ili želimo kreirati potpuno novi repozitorijum) moramo uraditi inicijalizaciju Git-a pozivom sledeće komande:

```
$ git init
```

Nakon uspešne inicijalizacije napraviće se poddirektorijum sa imenom .git u kome će se naći svi neophodni fajlovi za repozitorijum. U ovom trenutku se ništa ne prati od strane Git-a. Na primer, ako želimo da pratimo novi fajl po imenu main.java, moramo ga dodati korišćenjem komande *git add*, nakon čega sledi čuvanje izmena, na sledeći način:

```
$ git add main.java
$ git commit -m 'Inicijalna verzija projekta.'
```

Ako ne želimo da kreiramo projekat od početka, već da nastavimo rad na postojećem (na primer AOSP projekat kojem želite da doprinesete) moramo klonirati repozitorijum na

sledeći način:

```
$ git clone url_adresa_projekta
```

Ova komanda će kreirati novi direktrorijum sa imenom naziva projekta, u njemu će inicijalizovati .git direktorijum i preuzeti sve fajlove repozitorijuma i čekirati (eng. *check out)* poslednju verziju projekta.

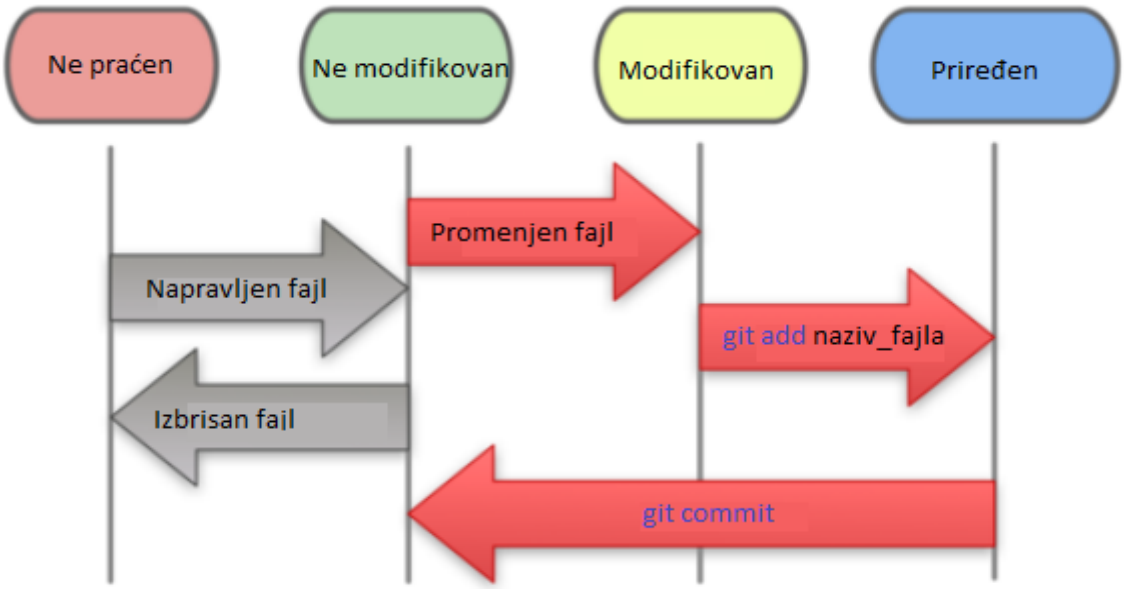
Kada imamo sve fajlove projekta na jednom mestu i njegovu radnu kopiju čekiranu možemo započeti sa radom (pravljenjem promena na fajlovima) na projektu i sačuvati izmene kada smatramo da treba (na primer, dodali smo novu metodu klase, rešili grešku u kodu i slično). Svaki fajl u Git-u može biti u jednoj od dve faze: praćen ili ne praćen. Praćen fajl (eng. *tracked file)* se nalazi u radnom direktorijumu od prošle verzije projekta i može biti modifikovan, ne modifikovan i priređen (eng. *modified, unmodified and staged).* Ne praćeni fajlovi su sve ostalo – bilo koji fajl koji nije priređen i koji nije deo zadnje verzije repozitorijuma. Kada se prvi put klonira projekat svi fajlovi su praćeni i ne modifikovani. Čim se neki fajl edituje, Git ga obeležava kao modifikovan. Sve modifikovane fajlove možemo dodati *git add* komandom i zatim sačuvati izmenjene fajlove nakon čega se ciklus ponavlja, što je ilustrovano na slici 3.3.

Glavna alatka za proveravanje statusa fajlova je *git status* komanda. Ako je pokrenemo odmah nakon kloniranja repozitorijuma dobićemo nešto slično ovome:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Ovo znači da vam je radni direktorijum čist, tj da nemate modifikovanih ili praćenih fajlova. Kada napravimo novi fajl (na primer *main.java*) informaciju da on trenutno nije praćen dobijamo izvršavanjem komande *git status*:

```
$ vim main.java
$ git status
# On branch master
# Untracked files:
# (use "git add <file>..." to include in what will be committed)
#
# main.java
```



Slika 3.3: Promena stanja fajla u Git-u

Kako bi naš novi fajl postao praćen (eng. *tracked)* potrebno ga je dodati u Git listu praćenih fajlova sledećom komandom:

```
$ git add main.java
$ git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# new file: main.java
```

Sada naravno možemo sačuvati izmene pozivom *git commit* komande čime ćemo zapamtiti trenutnu verziju fajla (presek stanja fajla nakon *git add* komande). Međutim ako pre čuvanja izmena opet promenimo naš fajl, njegova nova verzija će biti označena kao modifikovana, i ako želimo da sačuvamo tu najnoviju verziju moraćemo da pokrenemo *git add* komandu još jednom.

```
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   main.java
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
```

```
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   main.java
#
```

U svakom trenutku fajl možemo vratiti iz priređene faze u praćenu fazu pozivom *git reset* komande, a ako želimo da poništimo sve promene korisna je i *git reset –hard* komanda. Ako želimo da uklonimo fajl možemo pozvati *git rm* komandu.

Svakako jedna od najčešće korišćenih Git komandi je naredba za pregled istorije sačuvanih izmena (komanda *git log*) koja može dati izlaz sličan ovome:

```
commit d98f03fbb187224e28bfd28b8f62713704dca133
Author: lazar.radovanovic <lazar.radovanovic@sonymobile.com>
Date:   Wed Jun 25 23:50:32 2014 +0200

    Dodata zanimljiva funkcionalnost.

commit b1ffbe0ab0566ca6beda0e417cae0c7922e0a404
Author: lazar.radovanovic <lazar.radovanovic@sonymobile.com>
Date:   Wed Jun 25 23:50:03 2014 +0200

    Inicijalna promena.

    Change-Id: Ibe109d27aaa33c0cae4ad1c69304ea6a7732ce78
```

Ova komanda ima mnoštvo zanimljivih parametra. Na primer, sačuvane izmene možemo prikazati u jednoj liniji pozivom *git log –oneline* komande, zatim možemo izabrati vremenski interval za prikaz komandom *git log početni_sha1..završni_sha1*.

3.6 Pomoćne skripte za rad sa Git-om

Jedna jako korisna skripta koja nudi automatsko dovršavanje komandi (eng. *Auto-Completion),* je deo Git izvornog koda i može se naći na sledećoj veb adresi: https://github.com/git/git/blob/master/contrib/completion/git-completion.bash Preuzetu skriptu (git-completion.bash) možete iskopirati u lokalni .git direktorijum ili u globalni /etc/bash_completion.d/ kako bi se mogla koristiti u svakom repozitorijumu.

3.6.1 Repo skripta

Repo je skripta za upravljanje repozitorijuma pisana u programskom jeziku *Python*.

Repo objedinjuje Git repozitorijume kada je neophodno, izvršava objavljivanje koda na server i automatizuje pojedine delove razvoja Android-a. Repo nije napravljen da zameni Git već da ga učini lakšim za rad u kontekstu Android razvoja. U radu sa izvornim kodom Android projekta Repo se isključivo koristi za skidanje/objavljivanje izvornog koda preko mreže. Na primer, koristeći repo skriptu možete sa jednom komandom skinuti skup Git repozitorijuma u vaš lokalni radni direktorijum [6].

4 Projekat otvorenog koda – Android

Android projekat otvorenog koda obuhvata najznačajnije delove operativnog sistema koji nisu vlasništvo privatnih kompanija. Osnovni cilj projekta je da kreira otvorenu softversku platformu, podjednako dostupnu proizvođačima hardvera i programerima, omogućavajući im da implementiraju sopstvene ideje i objave uspešan proizvod koji će unaprediti iskustvo korišćenja mobilnih uređaja kod krajnjih korisnika [7].

4.1 Organizacija izvornog koda projekta

Za organizaciju ovako velikog projekta Google održava mnoštvo grana (ili tokova izvornog koda) kako bi odvojio stabilne verzije od onih eksperimentalnih. Sam proces objavljivanja nove verzije se može podelite u četiri faze:

1. U svakom trenutnku već postoji najnovija verzija platforme. Ona se obično oslikava jednom granom u sistemu.
2. Proizvođači uređaja i saradnici (eng. *contributors*) rade sa najnovijom verzijom platforme, popravljaju greške, izbacuju nove uređaje na tržište, isprobavaju nove funkcionalnosti i tako dalje.
3. Paralerno, Google radi na novoj verziji sistema imajući u vidu trenutne potrebe i ciljeve. Za ove potrebe Google bira jednog proizvođača i bira na kakvom uređaju želi da sledeća generacija sistema bude predstavljena u zavisnosti od toga u kom pravcu želi da razvija samu platformu.
4. Kada nova verzija konačno postane spremna, ona će biti javno objavljena u stablu izvornog koda i postati nova trenutna verzija čime se ciklus ponavlja.

Verzija (eng. *release)* odgovara javno poznatoj verziji platforme, na primer 2.3 ili 4.4. To je takođe i verzija softverskog skupa alata (eng. *Software Development Kit - SDK*). Eksperimentalne verzije se baziraju kako bi što više prihvatale doprinose zajednice saradnika i mogu vremenom da migriraju do stabilne verzije. Promene koje se pokažu kao stabilne će se prebaciti u *release* verziju. Ovo naravno važi samo za promene koje ne utiču na programski interfejs (eng. *Application Programming Interface - API*) platforme. Razvoj nove verzije platforme Android drži u tajnosti (eng. *private*) kako bi održao fokus na trenutnoj verziji sistema. Veliki proizvođači se često ne slažu sa ovom činjenicom, pre svega zato što ne mogu da predvide potrebna ulaganja u razvoj kako bi najnoviju verziju platforme preneli na svoje uređaje. Ipak, ovo je strategija kojom je Google odlučio da krene i za sada je ne menja.

4.2 Verzionisanje Android-a

Na visokom nivou razvoj Android-a se odvija oko familije imena koja su dobila naziv po

ukusnim poslasticama. Kodna imena se uparuju sa verzionim brojevima zajedno sa nivoima programskih interfejsa, što se može videti na tabeli 4.1:

Kodno ime	**Verzija**	**API nivo**
nema imena	1.0	API nivo 1
nema imena	1.1	API nivo 2
Cupcake	1.5	API nivo 3, NDK 1
Donut	1.6	API nivo 4, NDK 2
Eclair	2.0	API nivo 5
Eclair	2.0.1	API nivo 6
Eclair	2.1	API nivo 7, NDK 3
Froyo	2.2.x	API nivo 8, NDK 4
Gingerbread	2.3 - 2.3.2	API nivo 9, NDK 5
Gingerbread	2.3.3 - 2.3.7	API nivo 10
Honeycomb	3.0	API nivo 11
Honeycomb	3.1	API nivo 12, NDK 6
Honeycomb	3.2.x	API nivo 13
Ice Cream Sandwich	4.0.1 - 4.0.2	API nivo 14, NDK 7
Ice Cream Sandwich	4.0.3 - 4.0.4	API nivo 15, NDK 8
Jelly Bean	4.1.x	API nivo 16
Jelly Bean	4.2.x	API nivo 17
Jelly Bean	4.3.x	API nivo 18
KitKat	4.4 - 4.4.4	API nivo 19 i 20
Lollipop	5.0 - 5.1	API nivo 21 i 22
Marshmallow	6.0	API nivo 23

Tabela 4.1: Sve verzije operativnog sistema Android

4.3 Preuzimanje i izgradnja Android-a

Preporučeno okruženje za izgradnju (eng. *build*) Android sistema na kojem je prevođenje najviše testirano je neka od Ubuntu LTS distribucija. Možemo pretpostaviti da bi i ostala okruženja uz odgovarajuće alate bez većih problema mogla da prevode Android sistem. Za potrebe ovog rada korišćena je zvanična verzija Ubuntu operativnog sistema - LTS 12.04.

Pre preuzimanja i izgradnje (eng. *build)* Android projekta radno okruženje mora da zadovoljava sledeće uslove:

1. Ima instalirano 64-bitno Mac ili Linux okruženje.
2. Poseduje minimum 30GB slobodnog prostora (plus oko 9GB za izvorni kôd projekta).
3. Instaliran Python (neophodna verzija: 2.6-2.7).
4. Instaliran GNU Make[10] alat (neophodna verzija: 3.81-3.82).
5. Konfigurisan Java JDK verzije 7.

4.3.1 Podešavanje Linux okruženja za izgradnju projekta

Prvi korak predstavlja instalacija odgovarajućeg kompleta softvera za programski jezik Java (eng. *Java Development Kit - JDK*). Za glavnu (eng. *master)* granu potreban je JDK verzije 7. Možemo ga instalirati pokretanjem sledećih komandi iz terminala:

```
$ sudo apt-get update
$ sudo apt-get install openjdk-7-jdk
```

Opciono je potrebno podesiti putanje Java prevodioca do novo instalirane verzije. To se postiže vrlo jednostavno pokretanjem sledećih komandi iz terminala:

```
$ sudo update-alternatives --config java
$ sudo update-alternatives --config javac
```

Nakon toga se vrši instaliranje ostalih neophodnih paketa za izgradnju. Taj korak se jednostavno završava pokretanjem sledećih komandi:

```
$ sudo apt-get install git gnupg flex bison gperf build-essential \
  zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
  libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
  libgl1-mesa-dev g++-multilib mingw32 tofrodos \
  python-markdown libxml2-utils xsltproc zlib1g-dev:i386
$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

[10] GNU Make je softverski alat koji kontroliše izgradnju izvršnih fajlova programa koristeći izvorni kôd.

Prevodilac se može opcionalno konfigurisati da koristi alatku *ccache*. Ccache, kao što mu ime sugeriše, se ponaša kao keš za prevodilac i može znatno ubrzati proces ponovnog prevođenja projekta (eng. rebuild). Na primer, njegovo prisustvo će biti značajno kada pokrenete proces izgradnje nakon čišćenja prevedenih fajlova (što se lako postiže *make clean* komandom). Kako bismo aktivirali ovaj zgodan alat potrebno je da u naš .bashsrc (koji se nalazi u korisničkom home direktorijumu) dodamo sledeću liniju:

```
$ export USE_CCACHE=1
```

Podrazumevana lokacija za keš je ~/.ccache, ali možemo je promeniti podešavanjem keš direktorijuma sledećom linijom:

```
$ export CCACHE_DIR=<putanja do željenog direktorijuma>
```

Podrazumevana veličina keša je obično u rasponu od 50 do 100 gigabajta. Kako bi postavili to podešavanje potrebno je pokrenuti sledeću komandu jednom, pre prvog prevođenja (podešavanje će biti sačuvano unutar keš direktorijuma koji smo prethodno podesili):

```
$ prebuilts/misc/linux-x86/ccache/ccache -M 50G
```

Podrazumevan direktorijum za čuvanje rezultata prevođenja je *out* direktorijum. Lociran je u glavnom direktorijumu izvornog koda (odakle se pokreće prevođenje) projekta. Ponekad sama izgradnja može biti brža ako izlazni direktorijum premestimo na zaseban tvrdi disk. To se može postići sledećom komandom:

```
$ export OUT_DIR_COMMON_BASE=<putanja-izlaznog-direktorijuma>
```

Prethodnim koracima je naše Linux radno okruženje spremno za preuzimanje izvornog koda Android projekta.

4.3.2 Preuzimanje izvornog koda

Preuzimanje izvornog koda ovako velikog projekta je pojednostavljeno korišćenjem *repo* skripte koja nam olakšava manipulaciju velikog broja *Git* projekata. Instalacija *repo* skripte je prilično jednostavna i završava se izvršavanjem sledećih komandi:

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

Sada nam preostaje da podesimo našu mašinu (klijent) za pristup Android izvornom kodu. Ceo proces može biti podeljen u 3 koraka:

1. Kreiramo prazan direktorijum proizvoljnog imena za čuvanje svih potrebnih fajlova.

```
$ mkdir WORKING_DIRECTORY
$ cd WORKING_DIRECTORY
```

2. Pokrećemo *repo init* komandu sa URL adresom manifesta koji sadrži putanje do svih projekata koji će biti preuzeti.
   ```
   $ repo init -u https://android.googlesource.com/platform/manifest
   ```

3. Prilikom inicijalizacije repo će nas pitati za ime i e-mejl adresu. Potrebno je uneti pravo ime i validnu *gmail* adresu kako bi nam bilo omogućeno da pratimo naše izmene na globalnom servisu za pregled koda – Gerrit.

Ako je sve završeno kako treba repo će nas obavestiti porukom da je repozitorijum inicijalizovan u radnom direktorijumu. Biće kreiran *.repo* direktorijum u kome će se nalaziti sva repo podešavanja kao i pomenuti manifest fajl.

Na kraju ostaje nam samo da pokrenemo komandu za preuzimanje kompletnog stabla izvornog koda sa Google servera:

```
$ repo sync
```

Treba napomenuti da preuzimanje kompletnog izvornog koda može trajati više sati.

4.3.3 Izgradnja Android projekta

Ako prilikom inicijalizacije projekta (repo init komandom) nismo odabrali granu, podrazumevana je *master* grana koja sadrži najnovije izmene na kodu. Za sada ćemo se držati ove grane. Pre pokretanja komande za prevođenje moramo inicijalizovati radno okruženje pokretanjem *envsetup.sh* skripte koja se nalazi u *build* folderu:

```
$ source build/envsetup.sh
```

Zatim je potrebno izabrati željeni cilj, tj definisati željenu izlaznu konfiguraciju za proces prevođenja. To se postiže komandom *lunch* koja će nam izlistati sve moguće ciljeve od kojih biramo željeni ili možemo unapred, kroz dodatni argument, proslediti željeni cilj:

```
$ lunch aosp_arm-eng
```

Odlučujemo se za standardnu konfiguraciju (aosp_arm-eng) koja označava da želimo da prevodimo za emulator (alternativa je prevođenje za specifičan hardverski uređaj) sa svim uključenim opcijama za debaging. Kada smo odabrali željenu konfiguraciju možemo pokrenuti dobro poznatu komandu *make* i pokrenuti izgradnju projekta. Komanda *make* nam omogućava prevođenje u više nezavisnih niti prosleđivanjem –jN argumenta, gde N predstavlja broj niti. Najbolji rezultati se postižu kada se broj N postavi na dvostruku vrednost stvarnih (fizički prisutnih) jezgara centralnog procesora. Na primer ako naš centralni procesor ima 4 jezgara, prevođenje možemo pokrenuti na sledeći način:

```
$ make -j8
```

Kada se prevođenje završi, u zavisnosti od izabranog cilja u prethodnom koraku, možemo pokrenuti sistem na pravom uređaju ili u našem slučaju na emulatoru sledećom komandom:

```
$ emulator
```

4.4 Zajednica projekta otvorenog koda i načini doprinosa Android-u

Android je projekat otvorenog koda pod pokroviteljstvom kompanije Google. Svrha Android-a je da promoviše otvorenost koda u mobilnom svetu i da omogući širenje operativnog sistema na za sada nezamislive platforme. Niko ne može da predvidi gde će sve Android biti u budućnosti i zato je njegova otvorenost od ključnog značaja.

Najveći deo koda je licenciran pod Apache licencom verzije 2 (eng. *Apache Software License, Version 2.0),* dok je Linux jezgro pod *GPLv2* licencom.

Svi individualni saradnici na projektu otvorenog koda (eng. *contributors),* moraju potpisati poseban ugovor o saradnji za individualne saradnike (eng. *Individual Contributor License Agreement).* Ovaj ugovor definiše uslove pod kojima će intelektualno vlasništvo saradnika na razvoju Android-a doprineti njegovoj zajednici. Ugovor je tu da zaštiti saradnika ali i sam projekat i ne ograničava saradnike da koriste svoj doprinos (kôd) u bilo koje druge svrhe.

Pored individualnih saradnika, velike organizacije moraju pristati na kolektivni ugovor o doprinosu (eng. *Corporate Contributor License Agreement).* Ovaj tip ugovora omogućava korporacijama da autorizuju svoje verzije sistema, potpišu ga sa svojom licencom i zadrže u svom vlasništvu.

4.4.1 Prijavljivanje nedostataka

Jedan od najlakših načina da doprinesete projektu otvorenog koda je da otvorite novi nedostatak (eng. *bug).* Kako bi prijavili nedostatak zajednici, potrebno je pratiti sledeće korake:

1. Potražiti svoj nedostatak (velika je verovatnoća da je neko već primetio nedostatak koji želite prijaviti, pa je potrebno pretražiti spisak otvorenih ali i zatvorenih nedostataka).
2. Ako je vaš nedostatak već prijavljen, možete ga označiti kao bitnog i tako ga pomeriti ka vrhu u listi prioriteta za rešavanje.
3. Ukoliko se ustanovi da vaš nedostatak nije prijavljen, prijavite ga uz pomoć Google

alata za praćenje nedostataka (eng. *Google Issue Tracker)*.

Jednom prijavljen nedostatak je sada u globalnom sistemu praćenja. Njegov put od prijavljivanja do rešavanja se može sumirati sledećim fazama:

1. Nedostatak je prijavljen i ima status „Novi”.
2. Svi nedostaci prolaze kroz pregled osobe koja radi na održavanju sistema. Oni se smeštaju u jednu od četiri kategorije: novi, nije potrebno preduzeti akciju, otvoren i rešen.
3. Ako se za prijavljeni nedostatak pokaže da je validan, on prelazi u otvoreno stanje dok se ne reši kada dobija status „zatvoren”.
4. Svi zatvoreni nedostaci će se naći u nekoj od narednih verzija sistema.

4.4.2 Doprinos kodu

Kao i na svakom projektu otvorenog koda, zajednica će rado pregledati vaše promene na kodu. Android koristi alat za pregled promena koda poznatiji kao *Geritt.* Slika 4.1 prikazuje život promene kada je jednom dostavite zajednici. Iako možda deluje komplikovano, većinu tih stvari obavljaju softverski alati umesto vas.

Kako bi se promena dostavila zajednici, potrebno je pratiti nekoliko koraka. Najpre je potrebno napraviti novu granu uz pomoć *repo* skripte:

```
$ repo start IME_LOKALNE_GRANE IME_SERVERSKE_GRANE
```

Ova komanda će napraviti novu lokalnu granu sa zadatim imenom koja se oslanja na udaljenu (serversku) granu prosleđenu u zadnjem parametru komande. Programer može imati više lokalnih grana koje se mogu bazirati na različite udaljene (eng. *remote)* grane.

Kada napravite promene na kodu i detaljno ih testirate, spremni ste da ih pošaljete na pregled. To možete uraditi sledećim komandama:

```
$ git add -A
$ git commit
```

Nakon druge komande će vam se prikazati editor u kome treba uneti poruku o promeni. Kako će vaša promena završiti na javnom repozitorijumu, potrebno je pratiti savete za sastavljanje dobre poruke o promeni:

1. Poruka treba da ima jednolinijski naslov maksimalne dužine 60 karaktera, posle koga sledi prazna linija.
2. Nakon naslova, sledi opis promene. Ovde se treba fokusirati na to šta promena rešava i na koji način. Treba se truditi da se u što manje reči iznese što više korisnih informacija koje mogu biti korisne budućim pregledačima vašeg koda.

3. Treba uneti sve pretpostavke ili pozadinske informacije koje mogu biti značajne za buduće promene na delu koda koji se menja.
4. Svaka poruka o promeni koda mora da ima jedinstveni identifikator koji će *repo* sam generisati za vas.

Nakon prethodno navedenih koraka, promena je i dalje sačuvana samo na vašoj radnoj mašini. Kako biste promenu dostavili udaljenom serveru, potrebno je izvršiti sledeću komandu:

```
$ repo upload NAZIV_LOKALNE_GRANE
```

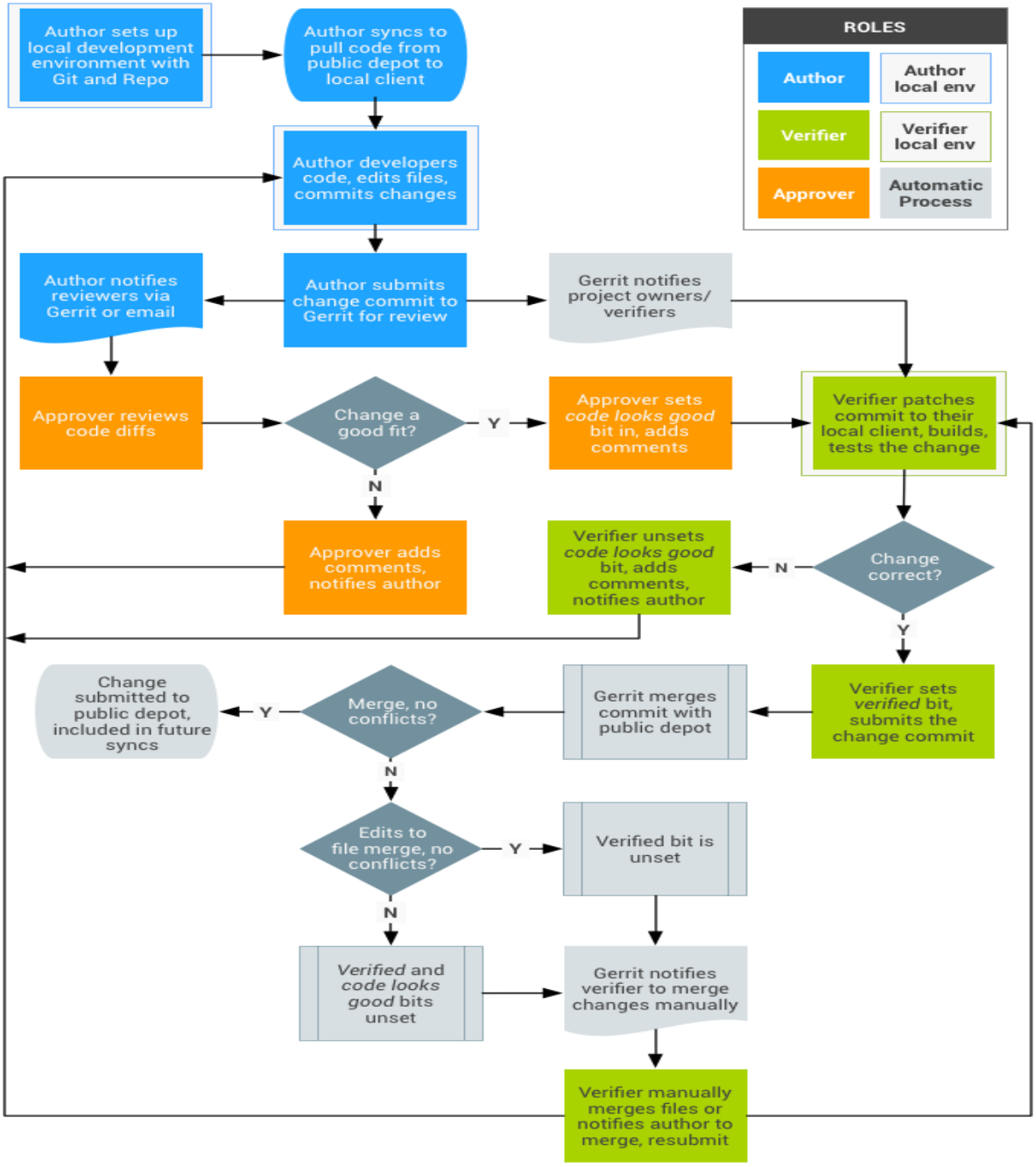
To je to. Vaša promena se nalazi na globalnom Android serveru i može se pregledati uz pomoć alata za pregled promena koda – *Geritt*. Kada drugi programeri vide vašu promenu, oni mogu ostaviti komentare na svaku liniju vašeg koda. Na primer, iskusniji programeri mogu videti potencijalne probleme sa vašim kodom koji ste vi prevideli. Ukoliko se to desi, morate reagovati i ispraviti potencijalne nedostatke. Kada ispravite greške, potrebno je postaviti novu zakrpu na vašu postojeću promenu. To možete uraditi sa sledećim komandama:

```
$ git add -A
$ git commit –amend
$ repo upload NAZIV_LOKALNE_GRANE
```

Izvršavanjem prethodno navedenih komandi, vaše izmene će biti vidljive na serveru u obliku nove zakrpe na postojećoj promeni.

Kada napokon pregledači budu zadovoljni vašom promenom, oni će vam odobriti promenu i sistem će je automatski pripojiti izvornom kodu projekta. Nakon ovoga, drugi korisnici koda mogu dovući vašu promenu. Ona će biti u istoriji projekta.

Svako može postati pregledač koda koji će voditi računa o kvalitetu promena tj koda koji će se nalaziti na budućim verzijama Android operativnog sistema. Ukoliko budete veoma dobri u tome, u smislu da ste aktivni, nalazite nedostatke u tuđem kodu i sami pravite kvalitetne promene možete postati verifikovani pregledač Android koda ili dobiti pravo da odlučujete o sudbini svake promene - da li će biti odobrena ili odbačena.



Slika 4.1: Životni ciklus promene koda

5 Prilagođavanje operativnog sistema Android

Kao što je više puta napomenuto u prethodnim poglavljima, suština Android operativnog sistema je u njegovoj otvorenosti, širokoj zajednici programera i lakom prilagođavanju. Pod prilagođavanjem Android operativnog sistema se podrazumeva preuzimanje trenutne verzije izvornog koda projekta i prilagođavanje koda sopstvenim potrebama koje kao rezultat treba da imaju novo nastalu verziju operativnog sistema. Nova (prilagođena – eng. *custom*) verzija sistema može da bude prevedena kako bi se izvršavala na mobilnim telefonima ali i na bilo kojem drugom uređaju. U prvom slučaju modifikacije mogu biti minimalne, zato što nije potrebno pisati nov upravljački sloj softvera (eng. *drivers)* za novi hardver. U slučaju da želimo dizajnirati novi hardverski uređaj na kome želimo pokrenuti operativni sistem Android, mogućnosti su ogromne ali je potrebno uložiti dosta resursa u sam razvoj. Mi ćemo se ovde ograničiti na prvu, mnogo prihvatljiviju i lakše izvodljivu varijantu.

5.1 Cilj i zahtevi prilagođavanja sistema

Kao primer prilagođavanja operativnog sistema Android, pokušaćemo da napravimo sistem koji će biti namenjen upotrebi u automobilu. Uređaj na kome će se prilagođeni operativni sistem izvršavati može biti mobilni telefon ili tablet.

Ranije pomenuti mobilni telefon ili tablet koji je opremljen našom, prilagođenom, verzijom operativnog sistema Android bi se ugrađivao u vozilo u delu instrument table automobila i imao bi konstantno napajanje kada je aktivan, dok bi se automatski gasio kada se isključi motor vozila u kome je ugrađen.

Funkcionalnosti koje bi ovakav sistem trebao da poseduje su:

1. komunikacija sa putnim računarom automobila u cilju prikupljanja svih neophodnih informacija,
2. jednostavan korisnički interfejs sa jasno vidljivim i dostupnim opcijama,
3. lak pristup putnim mapama i uključivanju navigacije,
4. reprodukcija multimedijalnog sadržaja,
5. konstantno merenje brzine automobila uz pomoć navigacionih mogućnosti uređaja – *GPS* prijemnika,
6. mogućnost vođenja putnih beleški.

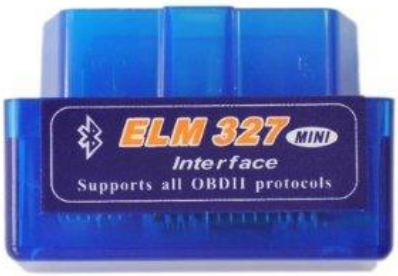
5.2 Hardverski deo sistema

Najvažniji deo našeg prilagođenog sistema predstavlja mobilni uređaj ili tablet koji bi se ugradio u vozilo. Android uređaj bi mogao da zameni postojeći radio uređaj koji postoji na vozilu ili da se ugradi nezavisno od njega. Što se tehničke specifikacije samog uređaja tiče potrebno je da može da pokrene najnoviju verziju operativnog sistema Android, da ima ugrađen

GPS prijemnik i *Bluetooth* adapter. Većina uređaja koji se trenutno mogu naći u prodaji bi zadovoljila ove zahteve.

Kako bi se multimedijalni sadržaj reprodukovao preko audio sistema koji postoji na vozilu potrebno je spojiti audio izlaz iz Android uređaja na automobilsko audio pojačalo. Za ovu svrhu bi se koristio običan analogan audio kabal sa potrebnim adapterima koji se lako može pronaći za sve tipove automobila. U slučaju novijih automobila može se koristiti i Bluetooth adapter telefona.

Svaki automobi proizveden od 2002. godine mora biti opremljen *OBDII* [11] interfejsom, koji se koristi za dijagnostiku kvarova [9]. Priključak na ovaj interfejs je standardizovan i obično se nalazi ispod volana automobila. Na njega se može priključiti bilo koji uređaj sposoban da pročita OBDII protokol. Za potrebe ovog rada korišćen je proizvod kineske kompanije sa oznakom *ELM 327 Mini*. Sposoban je da se poveže na sve vrste OBDII protokola na tržištu, a sa vašim Android telefonom komunicira preko Bluetooth konekcije. Prikazan je na slici 5.1:



Slika 5.1: OBD uređaj za povezivanje sa Android telefonom

5.3 Softverski deo sistema

Možemo ga podeliti u dve celine:

1. razvoj aplikacije koja će zameniti standardan početni ekran na Android uređaju (eng. *Launcher application)*,
2. modifikacija ostatka sistema kako bi zadovoljio zahteve našeg prilagođenog operativnog sistema.

[11] OBDII (eng. *On Board Diagnostics*) – je standardizovani interfejs koji se koristi prilikom dijagnostike kvarova na automobilu. Druga verzija ovog interfejsa donosi poboljšanja kako na polju dostupnih informacija tako i u definisanju standarda za potrebne konektore i informacije koje se moraju isporučiti.

Kada se operativni sistem Android pokrene na nekom uređaju, prvi ekran koji korisnik vidi je deo aplikacije čija je svrha pokretanje ostalih aplikacija. Ova aplikacija se obično sastoji iz dva dela. Prvog čini prilagodiva radna površina na kojoj korisnik može dodavati instalirane aplikacije koje će biti predstavljene svojim ikonicama i dodatke (eng. *widgets)*. Drugi deo početne aplikacije predstavlja listu svih instaliranih aplikacija. Aplikacije je moguće jednostavno pretraživati, brisati i pokrenuti.

Naš plan je da ove podrazumevane početne ekrane zamenimo sa našom verzijom aplikacije. Kako bi to postigli, gore pomenute aplikacije moramo izbrisati ili ukloniti iz spiska uključenih aplikacija za prevođenje.

5.3.1 Uklanjanje aplikacija iz Android sistema

Prevođenje Android operativnog sistema je zasnovano na *GNU make* alatu koji služi za organizaciju generisanja izvršnih fajlova, kao i drugih fajlova koji ne sadrže izvorni kôd u okviru nekog projekta. Kada korisnik pokrene komandu *make* u glavnom direktorijumu projekta, sistem pronalazi glavni (podrazumevani) fajl – *Makefile*. Glavni fajl za prevođenje sadrži mnoštvo drugih fajlova u zavisnosti od konfiguracije koju smo izabrali za prevođenje.

Kako bi izbacili ne željene aplikacije iz našeg sistema, moramo potražiti fajl *core.mk* u izvornom kodu projekta. Do njega možemo doći na sledeći način (pod pretpostavkom da smo u glavnom direktorijumu projekta):

```
$ cd platform_build/target/product/
$ nano core.mk
```

Pomenuti fajl sadrži listu aplikacija koje će biti prevedene i spakovane na sistemu. Skraćena verzija ovog fajla je prikazana u nastavku (isečak koda 5.1).

```
PRODUCT_PACKAGES += \
    Calculator \
    Calendar \
    CalendarProvider \
    CaptivePortalLogin \
    CertInstaller \
    Contacts \
    DeskClock \
    DocumentsUI \
    Email \
    Exchange2 \
    ExternalStorageProvider \
    InputDevices \
    KeyChain \
    Keyguard \
    LatinIME \
```

```
    Launcher \
    ManagedProvisioning \
    PicoTts \
    PacProcessor \
    libpac \
    Settings \

$(call inherit-product, $(SRC_TARGET_DIR)/product/core_base.mk)
```

Isečak koda 5.1: Fajl Core.mk

Sada samo trebamo ukloniti aplikaciju sa imenom „*Launcher*” i ona će biti preskočena prilikom izgradnje sistema. Takođe, ne želimo da naš sistem ima mogućnost automatskog zaključavanja ekrana, tako da možemo ukloniti i aplikaciju koja nosi ime „*Keyguard*”. Eventualno bi mogli da prođemo kroz ceo listing i uklonimo sve aplikacije koje nećemo koristiti čime bismo znatno rasteretili sistem. Doduše tada bismo povukli dosta zavisnih stvari po celom izvornom kodu pa bi imali znatno više posla da završimo prevođenje bez grešaka i bez uticaja na samu stabilnost sistema.

5.3.2 Dodavanje aplikacije u Android sistem

Kako bi dodali našu aplikaciju (koja predstavlja osnovnu pokretačku aplikaciju budućeg sistema) potrebno je uraditi nekoliko stvari. Najpre je potrebno dodati izvorni kôd aplikacije ili pripremljen binarni *.apk* fajl na sledećoj lokaciji:

```
$ cd package/app/
```

Ovde pravimo novi direktorijum sa nazivom „*CarLauncher*” i u okviru njega možemo prekopirati pripremljeni binarni .apk fajl ili ceo izvorni kôd. Odlučujemo se za prvu jednostavniju opciju. Pored binarnog fajla naše aplikacije potrebno je dostaviti i *make* specifikaciju kako bi sam proces prevođenja uzeo u obzir novo dodatu aplikaciju. Fajl se mora zvati „Android.mk” i potrebno je da ima sadržaj kao što je prikazan u isečku koda 5.2.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE_TAGS := optional

LOCAL_MODULE := CarLauncher

LOCAL_CERTIFICATE := ./keystore/CarLauncher.jks
```

```
LOCAL_SRC_FILES := ./apk/CarLauncher.apk

LOCAL_MODULE_CLASS := APPS

LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)

include $(BUILD_PREBUILT)
```

Isečak koda 5.2: Fajl Android.mk

Na kraju, kao i kod uklanjanja aplikacije, trebamo načiniti izmenu u *core.mk* fajlu i dodati naš paket listi svih paketa koji će biti prevedeni i instalirani na sistemu. Nova verzija ovog fajla bi trebalo da izgleda kao što je prikazano u isečku koda 5.3.

```
PRODUCT_PACKAGES += \
    CarLauncher \
    Calculator \
    Calendar \
    CalendarProvider \
    CaptivePortalLogin \
    CertInstaller \
    Contacts \
    DeskClock \
    DocumentsUI \
    Email \
    Exchange2 \
    ExternalStorageProvider \
    InputDevices \
    KeyChain \
    LatinIME \
    ManagedProvisioning \
    PicoTts \
    PacProcessor \
    libpac \
    Settings \

$(call inherit-product, $(SRC_TARGET_DIR)/product/core_base.mk)
```

Isečak koda 5.3: Finalna verzija fajla Core.mk

Sada možemo pokrenuti proces prevođenja i testirati modifikovani sistem na emulatoru. Kada se sistem uspešno pokrene, umesto standardnog početnog Android ekrana sada će biti prikazan početni ekran naše aplikacije.

Pored naše glavne aplikacije koja će služiti kao početna tačka korišćenja sistema potrebno je obezbediti još pratećeg softvera. Za navigacione potrebe se možemo osloniti na preinstaliranu aplikaciju *Google Maps* koja je pokazala da poseduje veoma pouzdane mape čak i na našim prostorima. Još jedan komad softvera od iste kompanije nam može poslužiti za vođenje beleški o automobilu. U pitanju je aplikacija *Google Keep* koja poseduje pregršt funkcionalnosti za rukovanje sa malim tekstualnim fajlovima.

Kada je u pitanju reprodukcija multimedijalnog sadržaja, pomoć ćemo potražiti u još jednom projektu otvorenog koda. Ovaj izvanredan alat za reprodukciju audio i video fajlova najrazličitijih formata su razvili volonteri i potpuno je besplatan. Radi se o projektu *VLC* koji je portovan na Android operativni sistem. Kako bismo obezbedili da se ova aplikacija nađe preinstalirana na našem prilagođenom Android sistemu, potrebno je potpuno besplatno preuzeti binarni *.apk* fajl sa njihovog zvaničnog sajta ili sa *Google Play* prodavnice i ponoviti identične korake kao pri ubacivanju glavnog pokretača programa sa početka ovog poglavlja.

5.4 Razvijanje početne aplikacije

Cilj ove aplikacije je da zameni standardni pokretač programa (slika 5.1), sa novom aplikacijom (slika 5.2) koja će sadržati usko specijalizovane funkcionalnosti, koje odgovaraju sistemu namenjenom za upotrebu u automobilu.

Slika 5.1: Standardan početni ekran Android sistema

Odabirom nekih od četiri dugmadi sa početnog ekrana, korisnik će biti preusmeren na odgovarajuću lokaciju i to:

1. Odabirom dugmeta „Mape“ korisnik se preusmerava na navigacionu aplikaciju kompanije Google – *Google Maps.*
2. Odabirom dugmeta „Održavanje“ korisnik se preusmerava na aktivnost aplikacije zadužene za ostvarivanje veze sa *OBDII* interfejsom i prikazivanju svih bitnih informacija dobijenih od putnog računara automobila.
3. Odabirom dugmeta „Multimedija“ korisnik se preusmerava na početni ekran alata za reprodukciju multimedijalnog sadržaja – *VLC Player*.
4. Odabirom dugmeta „Beleške“ korisnik se preusmerava na alat za vođenje kratkih tekstualnih poruka – *Google Keep*.

Slika 5.2: Novi početni ekran prilagođenog Android sistema

Kao što se vidi na slici 5.2, centralno mesto novog početnog ekrana zauzima posebno napravljena kontrola – pogled (eng. *View).* Početni ekran je dizajniran sa ciljem što jednostavnije navigacije kroz funkcionalnosti sistema, imajući u vidu da će korisnik ovakvog sistema u istom trenutku i upravljati automobilom.

5.4.1 Implementacija kontrole za prikaz brzine kretanja

Android nudi širok izbor različitih kontrola koje programeri mogu koristiti kako bi učinili svoje aplikacije lepe i funkcionalne. Svi oni nasleđuju osnovnu *View* klasu. Međutim, kada nam treba nešto specifično kao u ovom slučaju moramo se malo više potruditi i napraviti svoju kontrolu.

Kao što je slučaj i kod ostalih (sistemskih) kontrola, nasledićemo osnovnu *View* klasu. Kako bismo omogućili Android sistemu da komunicira sa našom kontrolom moramo obezbediti konstruktor koji će za argumente osim konteksta aplikacije primati i skup atributa koji je predstavljen klasom – *AttributeSet.* U okviru konstruktora ćemo izvršiti osnovnu inicijalizaciju potrebnih konstanti i komponenti kontrole (u okviru *initDrawingTools()* metode) što možete videti u isečku koda koji sledi (isečak koda 5.4).

```
public class Speedometer extends View implements SpeedChangeListener {
    private static final String TAG = Speedometer.class.getSimpleName();

    /**
     * Default constructor.
     */
    public Speedometer(Context context, AttributeSet attrs) {
        super(context, attrs);
        Log.d(TAG, "Speedometer(Context, AttributeSet) called");
        TypedArray a = context.getTheme().obtainStyledAttributes(attrs,
                R.styleable.Speedometer,
                0, 0);
        try {
            mMaxSpeed = a.getFloat(R.styleable.Speedometer_maxSpeed,
                DEFAULT_MAX_SPEED);
            mCurrentSpeed = a.getFloat(R.styleable.Speedometer_currentSpeed,
             0);
                ON_COLOR = a.getColor(R.styleable.Speedometer_onColor,
                ON_COLOR);
            OFF_COLOR = a.getColor(R.styleable.Speedometer_offColor,
                   OFF_COLOR);
            SCALE_COLOR = a.getColor(R.styleable.Speedometer_scaleColor,
                SCALE_COLOR);
            SCALE_SIZE = a.getDimension(R.styleable.Speedometer_scaleTextSize
                  ,SCALE_SIZE);
            READING_SIZE =
            a.getDimension(R.styleable.Speedometer_readingTextSize,
                READING_SIZE);
        } finally {
            a.recycle();
        }
        initDrawingTools();
    }
```

Isečak koda 5.4: Speedometer klasa

Dobro napisane kontrole bi trebale da omoguće identično ponašanje kao i one koje dolaze u standardnom skupu alata dostupnih programeru. Između ostalog, potrebno je da omogućimo da se naša nova kontrola može ugraditi u korisnički interfejs na standardan način - specifikacijom u *XML* fajlu. Ovo omogućuje vrlo lako prilagođavanje kontrole u smislu proizvoljnog određivanja njene visine i širine ali i specifičnih atributa kao što su maksimalna

brzina, upotrebljena boja za crtanje brzine… Kako bismo ovo omogućili potrebno je definisati proizvoljne atribute u okviru resursa aplikacije kao što je prikazano u isečku koda 5.5.

```
<resources>
    <declare-styleable name="Speedometer">
        <attr name="maxSpeed" format="float"/>
        <attr name="currentSpeed" format="float"/>
        <attr name="onColor" format="color"/>
        <attr name="offColor" format="color"/>
        <attr name="scaleColor" format="color"/>
        <attr name="scaleTextSize" format="dimension"/>
        <attr name="readingTextSize" format="dimension"/>
        </declare-styleable>
</resources>
```

Isečak koda 5.5: Novi atributi kontrole

U isečku koda 5.4 je prikazano kako se ovi atributi učitavaju u glavnom konstruktoru kontrole. Novu kontrolu sada jednostavno dodajemo korisničkom interfejsu (isečak koda 5.6). Naravno, nije neophodno popuniti sve atribute pošto smo obezbedili podrazumevane vrednosti.

```
    <com.radovanovic.lazar.carlauncher.views.Speedometer
        android:id="@+id/Speedometer"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:layout_width="300dp"
        android:layout_height="300dp"
        custom:currentSpeed="0"
        custom:maxSpeed="240"
        custom:scaleTextSize="12sp"/>
```

Isečak koda 5.6: Izgled *XML* fajla Speedometer kontrole

Svakako najvažniji korak u pisanju sopstvene kontrole je njeno prikazivanje na ekranu. Kako bi sami nacrtali našu kontrolu moramo predefinisati metodu *onDraw()*. Kao parametar ove metode dobijamo objekat klase *Canvas*, koji možemo zamisliti kao platno po kome možemo nacrtati našu kontrolu. Ona sadrži metode za crtanje linija, krugova, proizvoljnih putanja, teksta… Kada crtamo po platnu potrebno je izabrati odgovarajuću četkicu sa kojom želimo crtati, od koje će zavisiti način crtanja kao i boja kojom crtamo. Ovo je omogućeno uz pomoć klase *Paint*. Skraćena verzija implementacije ove metode je prikazana u isečku koda 5.7.

```
@Override
public void onDraw(Canvas canvas) {
    drawScale(canvas);
    drawLegend(canvas);
    drawReading(canvas);
}

private void drawLegend(Canvas canvas) {
    canvas.save(Canvas.MATRIX_SAVE_FLAG);
    canvas.rotate(-180, centerX, centerY);
    Path circle = new Path();
    double halfCircumference = radius * Math.PI;
    double increments = 20;
    for (int i = 0; i < this.mMaxSpeed; i += increments) {
        circle.addCircle(centerX, centerY, radius, Path.Direction.CW);
        canvas.drawTextOnPath(String.format("%d", i),
                circle,
                (float) (i * halfCircumference / this.mMaxSpeed),
                -30f,
                scalePaint);
    }

    canvas.restore();
}
```

Isečak koda 5.7: Metode za crtanje kontrole

5.4.2 Merenje brzine kretanja automobila

Od samog početka postojanja, uređaji sa operativnim sistemom Android bili su prepuni funkcija, a danas je gotovo nemoguće naći mobilni telefon ili tablet bez *GPS* prijemnika. Upravo smo ovaj mali hardverski senzor upotrebili za merenje brzine kojom se vozilo kreće.

Centralna klasa za rad sa lokacijama je *LocationManager*. Instancu ove klase dobijamo pozivom *getSystemService()* metode iz okruženja aplikacije (konteksta). Nakon toga moramo podesiti veliki broj parametra o tipu lokacije koju želimo dobiti, vremenskom intervalu na kojem želimo da primamo nove lokacije, koju preciznost same lokacije želimo, itd. Kada konačno podesimo sve ulazne parametre, ostaje samo da zatražimo obaveštenja o lokaciji.

Kada se promena lokacije zaista desi, sistem će pozvati specijalnu metodu (eng. *Callback)* preko koje će nam dostaviti objekat tipa *Location* iz koga možemo potom pročitati potrebne informacije i izračunati brzinu kretanja vozila (isečak 5.8).

```
@Override
public void onGPSUpdate(Location location) {
    double speed =  roundDecimal(convertSpeed(location.getSpeed()), 2);
```

```
        // Inform Speedometer about speed change.
        mSpeedometer.onSpeedChanged((float) speed);

        String speedString = "" +
                  roundDecimal(convertSpeed(location.getSpeed()), 2);
        Log.d(TAG,"SPEED UPDATE: " +  speedString);
    }

    private double convertSpeed(double speed) {
        return ((speed * Constants.HOUR_MULTIPLIER) *
                Constants.UNIT_MULTIPLIERS[Constants.INDEX_KM]);
    }
```

Isečak koda 5.8: Osvežavanje lokacije i računanje brzine uređaja

5.4.3 Komuniciranje sa putnim računarom automobila

Kao što je već pomenuto, komunikacija sa putnim računarom automobila se ostvaruje uz pomoć OBD interfejsa, koji je za programski jezik Java dostupan u obliku projekta otvorenog koda. Ovaj programerski okvir (eng. *framework)* možemo vrlo jednostavno pridružiti našem projektu koristeći *Gradle*[12] sistem za prevođenje (isečak koda 5.9).

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.github.pires:obd-java-api:1.0-RC9'
    }
```

Isečak koda 5.9: Dodavanje *Java OBD* programskog okvira

Pre početka komunikacije sa OBD interfejsom moramo ostvariti Bluetooth vezu sa našim adapterom za komunikaciju (ELM 327). Nakon uspešno ostvarene veze sa adapterom dobićemo otvorenu utičnicu (eng. *socket)* preko koje možemo obavljati svu neophodnu komunikaciju.

Centralna klasa OBD programskog okvira je *ObdCommand.* Komunikacija sa OBD interfejsom se odvija izvršavanjem metode *run* kojoj se prosleđuju ulazni i izlazni tokovi podataka (eng. *input and output streams)* prethodno dobijene utičnice (eng. *socket).* Kako se komande ne smeju izvršavati u paraleli, možemo ih staviti u red i izvršavati jednu po jednu. Ako je komanda uspešno izvršena rezultat trebamo prikazati korisniku. Metoda koja izvršava komande je prikazana u isečku koda 5.9.1.

[12] Gradle – je sistem za automatsko prevođenje programskog koda, zasnovan na deklarativnom programskom jeziku Groovy. Gradle koristi direktan aciklični graf kako bi odredio redosled po kome će zadati poslovi prevođenja biti izvršeni.

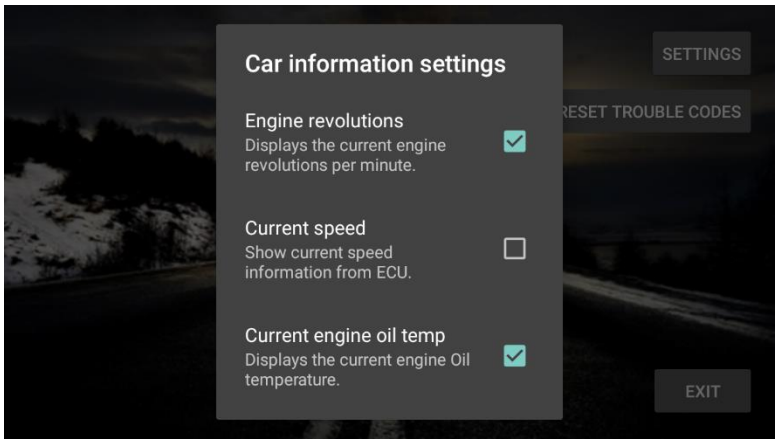
```
public void executeObdCommands() {
    if (mBluetoothSocket == null) {
        Logger.e(TAG, "No socket initialized.");
        return;
    }

    Logger.d(TAG, "executeObdCommands()");
    while (!mObdCommands.isEmpty()) {
        ObdCommand obdCommand = mObdCommands.poll();
        try {
            obdCommand.run(mBluetoothSocket.getInputStream(),
                    mBluetoothSocket.getOutputStream());

            // success! send result data.
            sendMessage(obdCommand.getName(), obdCommand.getResult());
            Logger.d("Result while executing " + obdCommand.getName()
                    + " command: ", obdCommand.getResult());
        } catch (Exception e) {
            Logger.e(TAG + obdCommand.getName(), e.toString());
        }
    }
}
```

Isečak koda 5.9.1: Izvršavanje OBD komandi

Sve komande koje možemo izvršavati su izvedene iz osnovne klase komandi (*ObdCommand).* Na primer, ako želimo pročitati trenutan broj obrtaja motora (eng. *RPM)* trebamo napraviti instancu klase *RPMCommand*, pozvati njenu metodu članicu *run* i pročitati rezultat koji će predstavljati broj obrtaja u minutu. Spisak komandi je dugačak, tako da smo korisniku ostavili mogućnost odabira inofrmacija koje želi dobiti od putnog računara (slika 5.3).



Slika 5.3: Odabir OBD komandi za prikazivanje

Putni računar automobila kontroliše rad motora i za njegov pravilan rad je neophodno da dobija ispravne informacije od svih senzora koji su ugrađeni u automobilu. Ukoliko neki senzor ne šalje ispravne podatke, putni računar će to registrovati i evidentirati u svom sistemu kao grešku u radu. Specijalizovana OBD komanda za čitanje ovih grešaka iz evidencije putnog računara je predstavljena klasom *TroubleCodesCommand*. Postoji mnogo razloga zašto je potrebno obrisati evidenciju grešaka putnog računara automobila (na primer, greška je nastala usled tehničkog pregleda vozila ili želimo da vidimo da li će se ponovo javiti). Specijalizovana komanda za ovu svrhu je predstavljena klasom *ResetTroubleCodesCommand.* Ova komanda treba biti aktivirana samo na eksplicitan zahtev korisnika (slika 5.4).

Slika 5.4: Prikazivanje rezultata izvršavanja OBD komandi

6 Zaključak

Predvodnik razvoja operativnog sistema Android je svakako kompanija Google. Međutim, i ostale velike kompanije (Samsung, HTC, LG, Sony…) svakodnevno daju svoj doprinos. Sve one imaju velike timove programera koji se bave razvojem softvera koji će dostaviti zajedno sa svojim mobilnim uređajima. U toku tog razvoja, svakodnevo se uočavaju problemi u svim delovima operativnog sistema. Problemi mogu biti nađeni u samom jezgru, nekoj sistemskoj aplikaciji ili servisu ili na bilo kom drugom mestu. Takvi problemi se rešavaju u okviru kompanija nakon čega se ispravljeni deo sistema doprinosi zajednici. Zahvaljujući ovome, operativni sistem je sa svakom narednom verzijom sve stabilniji, lakši za korišćenje i sa smanjenim brojem nedostataka.

Kao što se može i pretpostaviti, izvorni kôd samog projekta je ogroman. Međutim, kao što se moglo videti iz ovog rada, dovoljno je promeniti samo male delove sistema i dobiti željeni rezultat. Naravno, velike kompanije se neće zadovoljiti površnom modifikacijom sistema, već će želeti da preuzmu potpunu kontrolu nad projektom. U tom slučaju se javljaju problemi u vidu neprestanog pristizanja novih verzija (ili zakrpa) samog sistema pa su ove kompanije prinuđene da zamrznu svoju (lokalnu) kopiju koda na određenu verziju u cilju stabilizacije softvera na svom finalnom proizvodu.

Mogućnosti prilagođavanja Android operativnog sistema su velike. On se već nalazi na ogromnom broju uređaja uključujući pametne telefone, tablete, satove, narukvice, itd. Iz Google razvojnog centra je nedavno stigla još jedna popularna modifikacija ovog sistema za potrebe pametnih naočara (eng. *Google Glass)*, a ukoliko prilagođeni sistem ne zadovoljava potrebe korisnika vrlo lako se može zameniti nekim od mnogobrojnih alternativnih rešenja (Cyanogen Mod, Paranoid Android, OmniROM, SlimROM, AOKP, itd.).

Literatura

1. **Scott Chacon**, „Pro Git (Expert's Voice in Software Development)" Apress, 2009.

2. **Ben Collins-Sussman,** „Version Control with Subversion" Creative Commons 2011.

3. **History of Android**
   https://www.android.com/history/
   Datum pristupa: 3.5.2016.

4. **Rob Huddleston**, „Android fully loaded"
   Wiley Publishing Inc, 2011.

5. **Application Fundamentals**
   http://developer.android.com/guide/components/fundamentals.html
   Datum pristupa: 12.06.2014.

6. **Developing Android**
   https://source.android.com/source/developing.html
   Datum pristupa: 26.06.2014.

7. **Marko Gargenta**, „Learning Android"
   O'Reilly, 2011.

8. **Android Fragmentation Visualized**
   https://opensignal.com/reports/2014/android-fragmentation/
   Datum pristupa: 11. 2. 2015.

9. **Bob Henderson, John Haynes**: „OBD-II & Electronic Engine Management Systems".
   Haynes Inc, 2006.